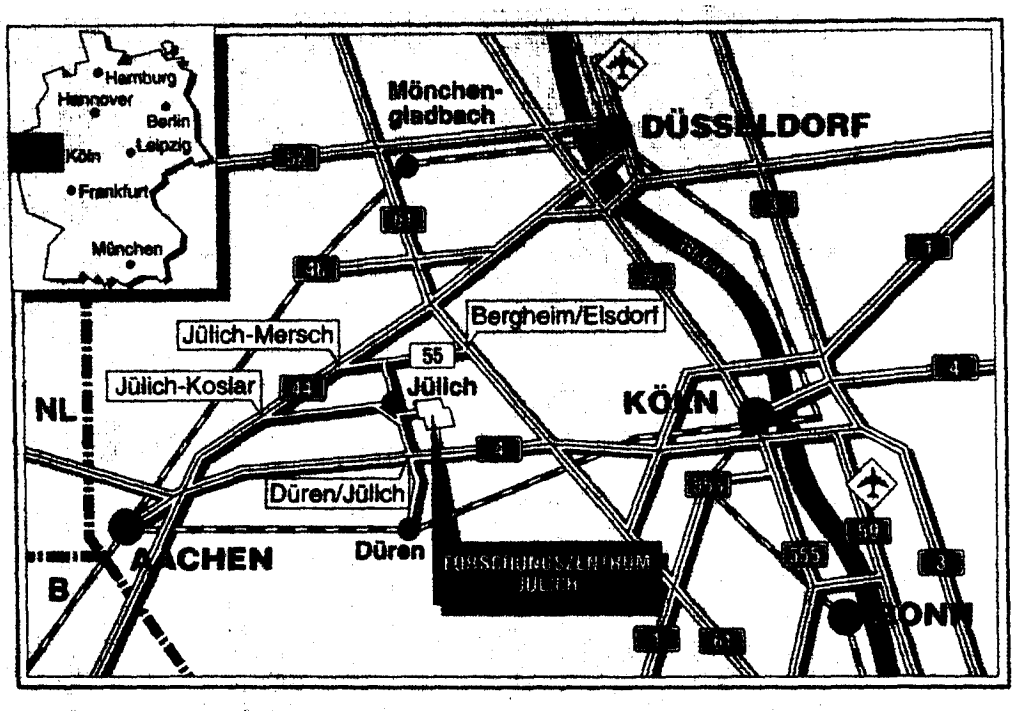
FORSCHUNGSZENTRUM JÜLICH GmbH

Zentralinstitut für Angewandte Mathematik

Anwendungen kombinatorischer Versuchspläne in der Parallelverarbeitung

Heribert C. Burg



Berichte des Forschungszentrums Jülich ; 2813
ISSN 0944-2952
Zentralinstitut für Angewandte Mathematik Jül-2813
D 82 (Diss. RWTH Aachen)

Zu beziehen durch: Forschungszentrum Jülich GmbH · Zentralbibliothek
D-52425 Jülich · Bundesrepublik Deutschland
Telefon: 02461/61-6102 · Telefax: 02461/61-6103 · Telex: 833556-70 kfa d

Anwendungen kombinatorischer Versuchspläne in der Parallelverarbeitung

Heribert C. Burg

Kurzfassung

Kombinatorische Versuchspläne (engl. experimental designs) gehen historisch auf landwirtschaftliche Versuchsanordnungen zurück, deren Entwurfsprinzipien eine Verzerrung der Experimentresultate durch Unterschiede in der Bodengüte ausschließen sollen. Mit der stürmischen Entwicklung der Kombinatorik in der zweiten Hälfte dieses Jahrhunderts wurden sowohl eine Vielzahl spezieller kombinatorischer Strukturen entwickelt als auch stetig neue Anwendungsgebiete für kombinatorische Versuchspläne erschlossen. Seit dem Aufkommen elektronischer Rechenanlagen wurde die Bereitstellung von Hilfsstrukturen für die Informatik eine zunehmend wichtigere Aufgabe der Kombinatorik. Mit dem Übergang zu (massiv-)parallelen Rechnerarchitekturen ist das Verständnis wie die Beherrschung paralleler Vorgänge zur großen Herausforderung an die Informatik erwachsen. Die vorliegende Arbeit analysiert daher Problemstellungen aus dem Bereich der Parallelverarbeitung mit den Mitteln der kombinatorischen Versuchspläne. Hierbei gilt es insbesondere, die strukturelle Verwandschaft der parallelen Strukturen mit den kombinatorischen Versuchsplänen nachzuweisen, wie sie in der jüngeren Literatur postuliert, aber bisher nicht gezeigt wurde. Mit dieser Zielsetzung wird zunächst eine Modellierung von Parallelrechnern und parallelen Algorithmen in einem kombinatorischen Konzeptrahmen durchgeführt. Die damit mögliche Auffassung von Objekten der Parallelverarbeitung als Versuchspläne erschließt die ganze Vielfalt der in der Kombinatorik vorhandenen Sätze und Gesetzmäßigkeiten für die Behandlung von Problemen der Parallelverarbeitung. Zu den betrachteten Aufgabenstellungen zählen das *Mapping*, das *Partitioning*, das *Embedding* sowie das *Routing*, wobei der in den letzten Jahren entwickelten Methode des *Randomized Routing* besondere Bedeutung beigemessen wurde.

Inhaltsverzeichnis

Abbildungsverzeichnis

Symbolverzeichnis

Allgemeine Symbole

$\|S\|$	Kardinalität der Menge S
$\max\{a_1, ..., a_n\}$	Maximum der Zahlen $a_1, ..., a_n$
$\min\{a_1, ..., a_n\}$	Minimum der Zahlen $a_1, ..., a_n$
(a_i)	Folge der Zahlen a_i
$\sum_{i=1}^{n} a_i$	Summe der Zahlen $a_1, ..., a_n$
$r \wedge s$	logische Und-Verknüpfung von r und s
$r \vee s$	logische Oder-Verknüpfung von r uns s
$\neg r$	logische Negation von r
$r \Leftrightarrow s$	Äquivalenz der Ausdrücke r und s
$r \Rightarrow s$	aus r folgt s
$a = b$	Gleicheit der Terme a und b
$a \neq b$	Ungleicheit der Terme a und b
$a < b$	a ist kleiner als b
$a > b$	a ist größer als b
$a \leq b$	a ist kleiner oder gleich b
$a \geq b$	a ist größer oder gleich b
$a \ll b$	a ist sehr klein gegen b
$x \equiv y \pmod{n}$	x ist kongruent zu y modulo n
$a \preccurlyeq b$	a vor b (partielle Ordnung)
$a \prec b$	a vor b (lineare Ordnung, totale Ordnung)
$x \in S$	x ist Element von S
$x \notin S$	x ist nicht Element von S
$\forall x$	Quantor: für alle x gilt
$\exists x$	Quantor: es existiert ein x, für das gilt
$\nexists x$	Quantor: es existiert kein x, für das gilt
$A \subseteq B$	A ist Teilmenge von B
$A \subset B$	A ist echte Teilmenge von B
$A \cup B$	Mengenvereinigung von A und B
$A \cap B$	Mengendurchschnitt von A und B
$A \setminus B$	Mengendifferenz von A und B

$f: A \to B$	Abbildung von A in B
$f: x \mapsto y$	x wird durch die Abbildung f auf y abgebildet
$f(x)$	Bild von x unter der Abbildung f
$\Pi(A)$	Permutation der Elemente der Menge A
$(i_1 \ldots i_r)$	r-Zyklus, zyklische Permutation der Länge r
$O(g)$	$O(g) := \{f \mid \exists c > 0\ \exists n_0 \in \mathbb{N}$ mit $\forall n \geq n_0\ f(n) \leq cg(n)\}$
$\Omega(g)$	$\Omega(g) := \{f \mid \exists c > 0\ \exists n_0 \in \mathbb{N}$ mit $\forall n \geq n_0\ f(n) \geq cg(n)\}$
$\Theta(g)$	$\Theta(g) := \{f \mid \exists c > 0\ \exists n_0 \in \mathbb{N}$ mit $\forall n \geq n_0\ \frac{1}{c} g(n) \leq f(n) \leq cg(n)\}$
$\mathbb{N}$	Menge der natürlichen Zahlen
$G = (V,E)$	Graph mit Knotenmenge V und Kantenmenge E
$GF(q)$	endlicher Körper mit Charakteristik q
$PG(m,q)$	projektive Geometrie der Dimension m über $GF(q)$

Versuchsplansymbole

V	endliche Grundmenge des Versuchsplans
$\mathscr{B}$	Mengensystem auf einer Grundmenge
$(V,\mathscr{B})$	kombinatorischer Versuchsplan auf V
v	Mächtigkeit von V, Kardinalität von $(V,\mathscr{B})$
$B = (a_1, \ldots, a_k)$	Block, $B \in \mathscr{B}$
b	Mächtigkeit von $\mathscr{B}$, Zahl der Blöcke des Versuchsplans
K	Menge der Blockgrößen, $K := \{\mid B \mid : B \in \mathscr{B}\}$
k	feste Blockgröße von $(V,\mathscr{B})$, falls $(V,\mathscr{B})$ k-uniform
$\mathscr{B}_x$	Teilmenge von $\mathscr{B}$, die alle Blöcke enthält, die x enthalten
r_x	Wiederholungsfaktor, $r_x := \mid \mathscr{B}_x \mid > 0$
R	Menge der Wiederholungsfaktoren, $R := \{r_x \mid x \in V\}$
r	eindeutiger Wiederholungsfaktor, falls $(V,\mathscr{B})$ (r-)regulär
$\lambda(S)$	Index von S in $\mathscr{B}$; Anzahl der Blöcke in $\mathscr{B}$, die S enthalten
Λ_t	$\Lambda_t = \{\lambda(S): \mid S \mid = t, S \subseteq V\}$
Λ	$\Lambda := \Lambda_2 = \{\lambda(s): \mid S \mid = 2, S \subseteq V\}$
λ_t	Balance bezüglich t-elementiger Teilmengen von V
λ	Balance bezüglich 2-elementiger Teilmengen von V, $\lambda := \lambda_2$
μ	Ordnung eines symmetrischen Plans
(v,k,λ)	Parametertripel eines BIBD, $t := 2$, $\lambda := \lambda_2$
t-(v,k,λ)	Parameterquadrupel eines t-Plans, $\lambda := \lambda_t$
t-$CB(v,k,\lambda)$	Parameterquadrupel eines zyklischen t-Plans, $\lambda := \lambda_t$

(v,k,Λ)	Parametertripel eines PBIBD
(v,K,λ)	Parametertripel eines PBD
(v,K,Λ)	Parametertripel eines PPBD
$\mathscr{B}_X$	Teilmenge der Blöcke von $\mathscr{B}$, die X enthalten
$\mathscr{B}_X\backslash X$	Menge von Blöcken, die man erhält, wenn aus jedem Block in $\mathscr{B}_x$ alle Elemente von X entfernt werden
$(V\backslash X,\mathscr{B}_x\backslash X)$	nach X abgeleiteter Plan, $X\subseteq V$
$(\overline{V,\mathscr{B}})$	Versuchsplan, der durch eine Permutation $\Pi(V)$ aus $(V,\mathscr{B})$ hervorgegangen ist
$\zeta = \{PC^1, ..., PC^n\}$	Zerlegung eines Versuchsplans in parallele Klassen PC^i
$(V, \{\mathscr{B}_i\}, I)$	Hyperplan, (I Indexmenge)
$(V^\times, \mathscr{B}^\times, \Phi, l)$	Versuchsplansequenz mit $V^\times = (V_1, ..., V_l)$, $\mathscr{B}^\times = (\mathscr{B}_1, ..., \mathscr{B}_l)$
l	Zahl der Versuchspläne in der Versuchsplansequenz
φ_{ij}	Relation von V_i nach V_j
Φ	Relationen der Versuchsplansequenz, $\Phi = (\varphi_{12}, ..., \varphi_{l-1,l})$
$(V,\mathscr{B})_1 \approx (V,\mathscr{B})_2$	Äquivalenz der Versuchspläne $(V,\mathscr{B})_1$ und $(V,\mathscr{B})_2$
$(V,\mathscr{B})_2 > (V,\mathscr{B})_1$	Versuchsplan $(V,\mathscr{B})_2$ emuliert Versuchsplan $(V,\mathscr{B})_1$
$(V,\mathscr{B})_1 < (V,\mathscr{B})_2$	Versuchsplan $(V,\mathscr{B})_1$ wird durch Versuchsplan $(V,\mathscr{B})_2$ emuliert
$(V,\mathscr{B})_2 \gtrsim (V,\mathscr{B})_1$	Versuchsplan $(V,\mathscr{B})_2$ emuliert Versuchsplan $(V,\mathscr{B})_1$ strikt
$(V,\mathscr{B})_1 \lesssim (V,\mathscr{B})_2$	Versuchsplan $(V,\mathscr{B})_1$ wird durch Versuchsplan $(V,\mathscr{B})_2$ strikt emuliert

Symbole zur kombinatorischen Beschreibung paralleler Strukturen

n	Ausbaustufe von Netzwerken
l	Zahl der Phasen in Algorithmen
m_i	Zahl der Module in Phase i
p	Zahl der Prozessoren des Parallelrechners
M_{ij}	Modul j in Phase i
P_r	Prozessorelement r der jeweiligen Topologie
$(V,\mathscr{B})_{ABC-n}$	Versuchsplan, der die Architektur ABC in Ausbaustufe n beschreibt
$(V^\times, \mathscr{B}^\times, \Phi, l)$	Versuchsplansequenz zur Beschreibung eines parallelen Algorithmus
$(V,\mathscr{B})_i$	Versuchsplan, der die Kommunikation in Phase i beschreibt
φ_{ij}	Relation, die den Übergang von Phase i nach Phase j beschreibt
$\Phi = (\varphi_{12}, ..., \varphi_{l-1,l})$	Folge der Relationen φ_{ij}
$\Delta(P_j, P_k)$	Distanz der Prozessorelemente P_j und P_k

$\Delta(M_{ij}, M_{ik})$	Distanz der Module j und k in Phase i
$\phi : V_A^* \rightarrow V_R$	Funktion, die einem Modul einen Prozessor zuweist
$\|\phi\|$	Güte der Mapping-Funktion ϕ (Bewertungsfunktion)
Z_i	Zielfunktion für die Kostenermittlung beim Mapping
$C(Z_i)$	Kosten nach Zielfunktion Z_i
C_{map}	Gesamtkosten eines Mapping-Vorganges

Boole'sche Prädikate

$\nabla_A(a_1, ..., a_k)$	das k-Tupel $(a_1, ..., a_k)$ bildet im Versuchsplan von A einen Block
$\rightleftharpoons(M_{ij}, M_{ik})$	in Phase i kommunizieren die Module j und k
$\Theta(P_r, M_{ij})$	in Phase i enthält Prozessor r Modul j

Kenngrößen für Rechnertopologien

d	Verknüpfungsdichte
C	Verbindungskosten
ρ	Redundanz
Δ_{max}	Durchmesser
δ	mittlere Weglänge
ψ	Verkehrsdichte
ω	Bandbreite

Abkürzungsverzeichnis

Mathematische Begriffe

DAG	Directed Acyclic Graph
GF	Galois-Feld (endlicher Körper)
FFT	Fast Fourier-Transform
FIFO	first in, first out (Prioritätenregelung)
FAFI	farthest to go, first (Prioritätenregelung)
NP	nichtdeterministisch-polynomial
oBdA	ohne Beschränkung der Allgemeinheit
PG	projektive Geometrie
PRBS	Pseudo Random Binary Sequence
ZE	Zeiteinheit
2D	zweidimensional
3D	dreidimensional

Topologien statischer Verbindungsnetzwerke

SBC	Simple Bus Computer
HBC	Hierarchical Bus Computer
CIC	Completely Interconnected Computer
STC	Star-Connected Computer
RCC	Ring-Connected Computer
CRC	Chordal-Ring-Connected Computer
TCC	Tree-Connected Computer
HCC	Hypertree-Connected Computer
CCC	Cube-Connected Computer
COC	Cube-Connected Cycle Computer
PCC	Pyramidal-Connected Computer
MCC2	Mesh-Connected Computer, two-dimensional
MCC3	Mesh-Connected Computer, three-dimensional
MBC2	Mesh-Bus-Connected Computer, two-dimensional
MBC3	Mesh-Bus-Connected Computer, three-dimensional

Versuchsplanstrukturen

BIBD	Balanced Incomplete Block Design
PBIBD	Partial Balanced Incomplete Block Design
PBD	Pairwise Balanced Design
PPBD	Partial Pairwise Balanced Design
PC	Parallel Class
PPC	Partial Parallel Class

Rechner-, Sprach- und Algorithmenmodelle

ADG	Attributed Directed Graph (Modell von Chiang/Fu)
CSL	Computation Structures Language (Modell von O'Dell/Browne)
CSP	Communicating Sequential Processes (Modell von Hoare)
DM	Design Machine (Modell von Beth/Hatz)
IDA	Information Dispersal Algorithm (Modell von Rabin)
LRA	Local Recursive Algorithm (Modell von Kung/Lewis/Jean)
NRCC	Network Routing Control Center (Modell von Kim/Reed)
PAG	Physical Architecture Graph (Modell von Kim/Browne)
PCM	Parallel Computation Model (Modell von Adiga/Browne)
PCS	Parallel Communication Scheme (Modell von Valiant)
SFG	Signal Flow Graph (Modell von Kung/Lewis/Jean)
VAG	Virtual Architecture Graph (Modell von Kim/Browne)

Flynns Architekturklassen

SISD	Single Instruction stream - Single Data stream
SIMD	Single Instruction stream - Multiple Data streams
MISD	Multiple Instruction streams - Single Data stream
MIMD	Multiple Instruction streams - Multiple Data streams

Rechnerentwurf

CMOS	Complementary Metal Oxid Semiconductor
PC	Personal Computer
VLIW	Very Long Instruction Word
VLSI	Very Large Scale Integration

Rechner

BVM	Boolean Vector Machine
CHiP	Configurable Highly Parallel computer
CHoPP	Columbia Homogeneous Parallel Processor
CM	Connection Machine
DAP	Distributed Array Processor
EGPA	Erlangen General Purpose Array
ELI	Enormously Long Instruction
KSR	Kendall Square Research
MPP	Massively Parallel Processor
NON-VON	Non-von-Neumann-Computer
PASM	Partitionable SIMD/MIMD machine
SUPRENUM	Superrechner für numerische Anwendungen
TRAC	Texas Reconfigurable Array Computer

1 Einführung

Die Strukturen, denen sich diese Arbeit widmet und die allgemein die Bezeichnung „Versuchsplan" führen, werden seit ihrem erstmaligen Auftreten von zwei sehr unterschiedlichen Bereichen der Mathematik für sich beansprucht: von der Kombinatorik und der Statistik. Während in der Kombinatorik die Struktur als solche, im Sinne eines Objektes der diskreten Mathematik, von Interesse ist, zielen die Anwendungen im Rahmen der Statistik mehr auf die den Strukturen innewohnenden statistischen Eigenschaften ab, die beispielsweise bei Stichprobenexperimenten den Einfluß von Verzerrungen beschreiben können. Dem Außenstehenden, wenn er nicht über weitergehende Kenntnisse der Statistik verfügt, fällt der kombinatorische Zugang im allgemeinen leichter, da viele relativ bekannte Strukturen wie Graphen, projektive Ebenen oder Lateinische Quadrate als Ausprägungsform von (kombinatorischen) Versuchsplänen aufgefaßt werden können. Auch im Rahmen dieser Arbeit soll die kombinatorische Sichtweise im Vordergrund stehen, da die Modellbildung den Kernpunkt der Arbeit darstellt. Die statistischen Eigenschaften der Versuchspläne bleiben jedoch nicht gänzlich unberücksichtigt, sondern werden für den Entwurf neuer Routing-Verfahren genutzt.

Die erstmalige systematische Untersuchung der Versuchsplanstruktur wird allgemein Euler zugewiesen, wenngleich davon ausgegangen werden muß, daß gleiche oder verwandte Strukturen bereits weitaus früher das Interesse insbesondere von Mathematikern geweckt haben werden, wenn beispielsweise die Historie der *Magischen Quadrate* betrachtet wird [D&K,74]. Bekanntgeworden ist Eulers *Problem der 36 Offiziere* [Eul,82], dessen Aufgabe in der Konstruktion einer (6×6)-Matrix besteht, deren Elemente 36 Offiziere repäsentieren. Die Offiziere sollen dabei so antreten, daß in jeder Reihe und in jeder Spalte je 1 Offizier aus einem von 6 verschiedenen Regimentern und je 1 Offizier mit einem von 6 verschiedenen Rängen steht. Es ist sicherlich trivial, (6×6)-Matrizen zu konstruieren, die den Teilanforderungen hinsichtlich Regimentern *oder* Rängen genügen. Das Problem der 36 Offiziere verlangt jedoch darüber hinaus, daß in einer gedachten Überlagerungsstruktur aus Regiment- und Rangmatrix jede Regiment-Rang-Kombination genau einmal auftritt. Euler vermutete, und dies wurde 1901 von Tarry bestätigt ([Tar,00], [Tar,01]), daß die gewünschte Überlagerungsstruktur für die Ordnung 6 nicht existiert. Die generelle Vermutung Eulers aber, die besagt, daß für alle geraden Ordnungen n, die nicht durch 4 teilbar sind, solche Paare orthogonaler Quadrate nicht existieren, wurde 1959 durch Bose, Shrikhande und Parker widerlegt, was ihnen in der Fachwelt den Namen *Euler's spoilers* einbrachte [BSP,60].

Euler bezeichnete die Elemente der einen quadratischen Teilmatrix mit lateinischen, die der anderen mit griechischen Buchstaben, daher sprach er eine Überlagerungsstruktur wie die gesuchte als Griechisch-Lateinisches Quadrat an. Daraus ergab sich dann rückwirkend die Bezeichnung „Lateinisches Quadrat" für die einfache Struktur, die bisweilen Euler selbst [Fla,69], bisweilen Fisher zugeschrieben wird [S&S,87]. Es waren also rein äußerliche Gründe, die zu dieser, zunächst unverständlichen, Benennung führten; eine mathematische Herleitung des Begriffes ist nicht gegeben. Nun ist zwar mit den historischen Anfängen die Begriffsbildung *Lateinisches Quadrat* als spezielle Form des Versuchsplans erklärt, nicht jedoch die Bezeichnung *Versuchsplan* als solche.

Seit dem ausgehenden 18. Jahrhundert wurde, beginnend in der Landwirtschaft, die systematische Planung von Experimenten entwickelt, wobei im wesentlichen darauf abgezielt wurde, äußere Einflüsse, die zu Verzerrungen der experimentellen Ergebnisse führen

können, auf ein Minimum zu beschränken. Ein typisches Experiment könnte etwa der Beurteilung verschiedener Düngemethoden gelten. Dazu wird eine landwirtschaftliche Nutzfläche entsprechend der Anzahl v der zu untersuchenden Dünger in v Bahnen aufgeteilt. Aus früheren Experimenten ist bekannt, daß die verwendeten Böden teilweise beachtliche Unterschiede hinsichtlich ihrer Güte bzw. ihrer Fruchtbarkeit aufweisen, so daß die Ergebnisse des Experimentes völlig unbrauchbar sein können, wenn die Bahn eines bestimmten Düngers sich zu großen Teilen mit dem besonders fruchtbaren Bereich des Feldes deckt. Es kann gezeigt werden, daß im Mittel eine optimale Entkopplung der Experimentresultate von den Bodengüteunterschieden gegeben ist, wenn das Feld wie eine $(v \times v)$-Matrix aufgeteilt wird und die unterschiedlichen Dünger in der Art ausgebracht werden, wie es durch die Anordnung in Lateinischen Quadraten vorgegeben ist. Soll darüber hinaus ein weiterer Faktor, etwa v Weizensorten, kontrolliert werden, so kann dies durch ein Griechisch-Lateinisches Quadrat geschehen.

In der Landwirtschaft und auch in anderen Bereichen sind viele weitere Experimente denkbar, bei denen die gleichen Überlegungen zu analogem Vorgehen führen [Che,58]. Street und Street [S&S,87] beschreiben ein Experiment von de Palluel aus dem Jahre 1788, in dem Futterarten für Schafe unterschiedlicher Züchtung untersucht werden. Experimente in der Medizin könnten etwa der vergleichenden Analyse unterschiedlicher Medikamente zur Behandlung der gleichen Krankheit dienen. Diese und ähnliche Anwendungen führten zur Bezeichnung Versuchsplan (engl.: *experimental design*) für die in der vorliegenden Arbeit behandelten Strukturen. Während im Deutschen für dieselben Strukturen auch die Bezeichnung *optimaler Versuchsplan* geläufig ist, spricht man im Englischen oft nur von *designs*, fügt andererseits aber gerne das Adjektiv *combinatorial* hinzu, wenn die kombinatorische Sichtweise im Vordergrund steht.

Die Anordnung, die durch ein Lateinisches Quadrat beschrieben wird, kann mathematisch so formuliert werden, daß in einem Paar aus einer Menge und einem *Mengensystem* zunächst die Grundmenge der Elemente festgelegt wird und mittels des Mengensystems dann die Einträge zeilenweise zu sogenannten *Blöcken* zusammengefaßt werden. Basierend auf diesem Formalismus können die Strukturen dahingehend abgewandelt werden, daß die einzelnen Blöcke, vormals Zeilen, nicht mehr alle Elemente der Grundmenge in fester Anordnung enthalten, sondern nur noch eine Teilmenge, wobei eventuell auch die Reihenfolge der Elemente innerhalb eines Blockes beliebig wird. Das Mengensystem wird somit zu einer Sammlung von Teilmengen der Grundmenge. Im Gegensatz zur Potenzmenge dürfen einzelne Teilmengen auch mehrfach auftreten.

Ein bekanntes Problem, daß sich so formulieren läßt, ist *Kirkman's Schulmädchenproblem* [Ahr,18]. Dieses stellt die Frage, ob für eine Klasse von 15 Mädchen sieben Anordnungen zu fünf Dreierreihen hintereinander so gefunden werden können, daß an sieben aufeinanderfolgenden Tagen jedes Mädchen mit jedem anderen Mädchen genau einmal in einer gemeinsamen Reihe marschiert. In derartigen kombinatorischen Strukturen von Dreierblöcken, die allgemein Steiner-Tripelsysteme, und in dem Fall der Aufteilbarkeit in Anordnungen, die alle Elemente enthalten, Kirkman-Tripelsysteme genannt werden [Jac,83], ist folglich von Interesse, ob jedes ungeordnete Paar gleichhäufig in allen Tripeln auftritt. Hier spielt die sogenannte *Balance* eine Rolle, die zu den statistischen Betrachtungen hinführt. Fisher war es schließlich, der ab dem Jahre 1926 die systematische Untersuchung der Versuchspläne im Hinblick auf die Gestaltung von Ex-

perimenten initiierte und zu einer eigenständigen Theorie ausbaute ([Fis,26], [Fis,60]). Mit der stürmischen Entwicklung der Kombinatorik in der zweiten Hälfte dieses Jahrhunderts ([Bec,64], [Sri,73], [Hal,86]) wurden sowohl eine Vielzahl spezieller kombinatorischer Strukturen entwickelt als auch stetig neue Anwendungsgebiete für kombinatorische Versuchspläne erschlossen ([S&H,84], [Mea,90]). Seit dem Aufkommen elektronischer Rechenanlagen wurde die Bereitstellung von Hilfsstrukturen zur Behandlung von Problemen der Informatik eine zunehmend wichtigere Aufgabe der Kombinatorik [O&W,76].

Mit dem Übergang zu (massiv-)parallelen Rechnerarchitekturen ist das Verständnis wie die Beherrschung paralleler Vorgänge zur großen Herausforderung an die Informatik erwachsen. Die vorliegende Arbeit zielt daher auf die Behandlung von Problemstellungen aus dem Bereich der Parallelverarbeitung mit den Mitteln der kombinatorischen Versuchspläne. Hierbei gilt es, die Fülle der Strukturen, Sätze und Gesetzmäßigkeiten der Kombinatorik der optimalen Versuchspläne zu nutzen, um typische Strukturen aus dem Bereich der Parallelverarbeitung in einem ersten Schritt zu beschreiben und dann, basierend auf diesen Beschreibungen, die sich dort stellenden Probleme mit kombinatorischen Mitteln zu lösen. Dabei soll die Erfassung der Objekte der Parallelverarbeitung, namentlich parallele Algorithmen und Multiprozessorrechner mit verteiltem Speicher, mit Hilfe der kombinatorischen Versuchspläne durchaus als eigenständige Aufgabe angesehen werden. Zwar ist in den letzten beiden Jahrzehnten eine große Zahl von wissenschaftlichen Publikationen über spezielle Probleme in der Parallelverarbeitung mit teilweise originären Lösungen erschienen, man kann jedoch sicherlich bei weitem noch nicht von einem geschlossenen, methodischen Gebäude sprechen, wie man es in den meisten Teilgebieten der Mathematik, insbesondere auch bei der Kombinatorik, vorfindet [Hoß,83].

Sogenannte Integrierte Konzepte beschreiben parallele Hardware und parallele Algorithmen mit gleichen Mitteln und eröffnen damit insbesondere neue Zugänge zum *Mapping*-Problem. Valiant zielt mit seinem *bulk-synchronous parallel model* auf die gleiche Problematik ([Val,90a], [Val,90b]). Er sieht das Fehlen eines einheitlichen Grundmodells für Parallelrechner, wie es mit dem von-Neumann-Modell im sequentiellen Fall geschaffen wurde, als eigentliche Ursache für den noch nicht erfolgten Durchbruch der Parallelrechner in der Anwendung an. Theoretische Vorarbeit zum Aufbau eines solchen Standardmodells der Parallelverarbeitung kann zumindest ansatzweise durch die Modellierung mit Hilfe kombinatorischer Versuchspläne geleistet werden. Ähnlich wie das von Valiant vorgeschlagene Modell zielt auch die Modellierung mit kombinatorischen Versuchsplänen weder ausschließlich auf Hardware noch auf Software ab, sondern schlägt eine konzeptionelle Brücke und eignet sich daher insbesondere zur Lösung des Mapping-Problems.

Bezüglich der Modellierung paralleler Strukturen ist es ein wichtiges Anliegen der vorliegenden Arbeit, das Versuchsplankonzept dem Graphentheoriekonzept gegenüberzustellen. Nach Mickunas [Mic,80] liefert die Graphentheorie dann nur wenig brauchbare Ergebnisse, wenn entweder der Durchmesser des Graphen oder die Zahl der Knoten pro Verbindung erhöht werden soll. In der vorliegenden Arbeit soll gezeigt werden, daß mit der Auffassung eines Graphen als eine spezielle, eingeschränkte Form eines kombinatorischen Versuchsplans die größere Mächtigkeit der Versuchsplanstruktur genutzt werden

kann, ohne die gewohnte und als Quasistandard akzeptierte Modellierung mittels Graphen aus den Augen zu verlieren.

Das Ziel der vorzunehmenden Untersuchungen ist vor allem darin zu sehen, die strukturelle Verwandschaft der Objekte aus Kombinatorik und Parallelverarbeitung aufzuzeigen, welche auch von Richard M. Karp postuliert wird [K&W,85]:

> ... We believe that combinatorial designs will find many applications in the design of efficient deterministic algorithms, and particularly in parallel algorithms, where they seem to fit so naturally.

Die Verwendung des Begriffes *parallel* auf Seiten der Kombinatorik scheint hier erste Indizien zu liefern ([Cam,76], [Phe,79]). Wenn die Gültigkeit des Karpschen Postulats nachgewiesen werden kann, ist es möglich, vom methodischen Gebäude der Kombinatorik hinreichend viel in die Parallelverarbeitung zu übertragen, um ein weiteres leistungsfähiges Instrumentarium für die Parallelität zu entwickeln.

Neben diesem Induzieren einer strengen Systematik in den Bereich paralleler Strukturen soll die umfangreiche Theorie über kombinatorische Versuchspläne für die Parallelverarbeitung genutzt werden, um durch die Übertragung von Gesetzmäßigkeiten aus dem Bereich der Kombinatorik in den Bereich der Parallelverarbeitung auf interessante und wichtige Fragestellungen, wie Mapping, Partitionierung und Emulation, neue Antworten zu finden. C. J. Colbourn und P. van Oorschot konstatieren, daß die Kombinatorik der optimalen Versuchpläne zu diesem Zweck hinreichend Instrumente zur Verfügung stellt: „... the tools of combinatorial designs form a valuable part of the equipment needed to solve problems of computation efficiently" [C&O,89]. In Anhang A sind diejenigen Definitionen und Sätze über kombinatorische Versuchspläne zusammengefaßt, die in der vorliegenden Arbeit Anwendung finden.

In einzelnen Teilbereichen der Informatik und speziell auch in der Parallelverarbeitung wurde bereits auf Versuchsplanstrukturen zurückgegriffen. Im Bereich der sogenannten *Skewing-Schemata* ([B&K,71], [Wij,89]), die den konfliktfreien Zugriff auf mehrdimensionale Datenstrukturen in verschränkten Speichern ermöglichen können, ist gezeigt worden, daß die Forderungen an ein entsprechendes Abspeicherungsmuster auf das Vorliegen eines *Knut-Vik-Plans* zurückgeführt werden kann [Sha,78]. Ein Knut-Vik-Plan ist eine Weiterentwicklung des Lateinischen Quadrats mit zusätzlichen Anforderungen an die Verteilung der Elemente über Diagonalen. Das Zugriffsproblem bzw. die zu seiner Lösung herangezogene Struktur stehen in Zusammenhang mit dem von Pólya beschriebenen *Problem der Superqueens* (*n-Damen-Problem*) [Pól,18]. Eine Superqueen, als hypothetische Schachfigur, verfügt dabei über die Zugmöglichkeiten einer Dame, die um zyklisches Schließen der Zugrichtungen über den Brettrand hinaus ergänzt werden. Shapiro zeigte, daß n nicht-angreifende Superqueens dann und nur dann auf einem $(n \times n)$-Schachbrett positioniert werden können, wenn n weder durch 2 noch durch 3 teilbar ist. Anwendungen des n-Damen-Problems und damit implizit des Knut-Vik-Versuchsplans ergeben sich ebenso bei den *Scheduling-Verfahren*, die die Optimierung der Zuteilung von Prozessen zu Prozessoren untersuchen [W&S,90]. Ein Beispiel für originäre Parallelisierung mit Hilfe optimaler Versuchspläne wird von Karp und Wigderson [K&W,85] beschrieben. Sie liefern einen parallelen Algorithmus, der mit $O(n^3/(\log n)^3)$ Prozessoren für einen gegebenen Graphen mit n Knoten eine maximal

unabhängige Menge in der Komplexität $O((\log n)^4)$ bestimmt. Lateinische Quadrate sind auch als Operatoren (*templates*) in der Bildverarbeitung verwendet worden [K&L,88]. Daneben sind starke Bindungen zur Codierungstheorie gegeben ([C&L,75], [B&J,83]).

Colbourn und van Oorschot [C&O,89] beschreiben einzelne Anwendungen, die Schwellwertschemata, Authentisierungscodes, Kernspeicher und die Organisation verteilter Datenstrukturen betreffen, dazu sei auch auf spezielle Arbeiten verwiesen ([AGR,68], [Ber,76], [BAG,69]), es wird jedoch kein Gesamtkonzept geliefert. Demgegenüber wird hier der Versuch vorgelegt, eine systematische, die einzelnen Instrumente integrierende Sicht paralleler Strukturen auf der Basis der Theorie der kombinatorischen Versuchspläne zu entwerfen. Dabei können in den einzelnen Teilbereichen nur prinzipielle Strukturen und Vorgänge berücksichtigt werden, da die Arbeit primär auf eine Gesamtaussage abzielt, womit in die Breite gehende Untersuchungen in den Teilbereichen ausgeschlossen sind.

Mit dieser Zielsetzung soll Kapitel 2 erste Anhaltspunkte für die strukturelle Verwandschaft von kombinatorischen Versuchsplänen und parallelen Objekten liefern. Der Zugang erfolgt zunächst über die parallelen Architekturen, für die in Abschnitt 2.1 eine kurze Klassifizierung vorgestellt wird. Diese dient auch zur Erläuterung der in der Arbeit auf der Hardware-Seite durchgeführten Restriktion. In Abschnitt 2.2 wird erläutert, wie Parallelrechner durch kombinatorische Versuchspläne beschrieben werden können und welche Bedeutung damit die für die Kombinatorik geschaffenen Eigenschaften von Versuchsplänen in der Parallelverarbeitung gewinnen. Darüber hinaus wird aufgezeigt, wie die Vorgehensweise systematisiert und die Untersuchungen auch auf andere parallele Strukturen ausgeweitet werden können. In Abschnitt 2.3 werden Beispiele für Forschungsarbeiten gegeben, die bereits Möglichkeiten zum Entwurf von Parallelrechnerarchitekturen auf der Basis von Versuchsplänen analysiert haben.

Ziel der Kapitel 3 und 4 ist es, die Eignung kombinatorischer Versuchspläne als strukturierende Elemente für die Parallelität herauszuarbeiten. Die Spezifikation mit Hilfe kombinatorischer Versuchspläne konzentriert sich in Kapitel 3 auf parallele Rechner, in Kapitel 4 auf parallele Algorithmen. Mittels der Beschreibung durch kombinatorische Objekte soll so eine Klassifizierung paralleler Strukturen entstehen, die sich unmittelbar aus der kombinatorischen Mathematik ergibt, wenn Parallelrechner und parallele Algorithmen als kombinatorische Versuchspläne verstanden werden.

Dies ist der Ausgangspunkt für weitere Strukturierungsmaßnahmen, die sich, nachdem für Rechner und Algorithmen bereits Versuchsplandarstellungen vorliegen, in Kapitel 5 konsequenterweise im Mapping, also der Abbildung paralleler Programme auf verteilte Rechner, fortsetzen. Die Verwendbarkeit mathematischer Werkzeuge der Kombinatorik am Beispiel der Aufgabenfelder Partitionierung und Einbettung (*embedding*) soll in Kapitel 6 demonstriert werden.

Zuweilen wird auf die wertvolle Balance-Eigenschaft kombinatorischer Versuchspläne im Hinblick auf Anwendungen in der Informatik verwiesen [C&O,89]; die in dieser Arbeit postulierte Anwendbarkeit auf die Lastbalance im Kommunikationsnetzwerk eines Parallelrechners ist jedoch bisher nicht analysiert worden. Dazu sollen in Kapitel 7 kombinatorisch motivierte Routing-Verfahren entworfen werden, wobei sowohl deterministische Routing-Strategien als auch zufällige von Interesse sind. Bei den zufälligen

Verfahren werden durch Ersetzung von Zufallsschemata durch kombinatorische Schemata neue Routing-Verfahren entwickelt und hinsichtlich ihrer Eignung untersucht.

Im abschließenden Kapitel 8 erfolgt eine Zusammenfassung der wesentlichen Ergebnisse sowie eine einführende Betrachtung in die zahlreichen offenen Probleme und die Möglichkeiten der Weiterentwicklung des Modells, der Problemlösungsansätze und der kombinatorischen Routing-Verfahren. Darüber hinaus soll die ganze Bandbreite der Anwendungsmöglichkeiten kombinatorischer Versuchspläne auch für die Teilgebiete der Parallelverarbeitung angedeutet werden, die im Rahmen der Arbeit nicht behandelt werden konnten.

2 Parallele Strukturen und Versuchspläne

2.1 Klassifizierung von Rechnerarchitekturen

Zahlreiche Klassifizierungschemata für Rechner sind in den letzten Jahrzehnten vorgeschlagen worden, von denen hier nur die Klassifizierungen von Flynn [Fly,66] und Händler [Hän,75] sowie als neuere Arbeit die Klassifikation von Duncan [Dun,90] erwähnt sein sollen. Wichtige Feststellungen in diesem Zusammenhang sind:

- Klassifizierungsschemata müssen einfach und schlüssig sein, wenn sie eine hinreichende Akzeptanz erfahren sollen; während Flynns Taxonomie nahezu jedem Informatiker bekannt ist, konnte sich Händlers Schema nicht allgemein durchsetzen.
- Eine Klassifizierung, die den Gesamtraum möglicher Architekturen logisch – nicht technologisch – vollständig erschließt, wie dies bei Flynn der Fall ist, induziert eine eingängigere und stabilere Ordnung als eine Klassifizierung, die sich auf Bäume unterschiedlicher Astlänge stützt, selbst dann, wenn wie bei Flynn einzelne Klassen faktisch leer bleiben.
- Entwicklungen auf dem Hardware-Sektor führen zu impliziten Klassifikationen, die mit einem zu entwerfenden Schema konform sein müssen bzw. in dieses integriert werden müssen.

Auf der Grundlage dieser Feststellungen läßt sich ein Klassifikationsschema entwickeln, das sich aus sechs oder mehr hierarchisch angeordneten Ebenen (*level*) aufbaut:

1. Ablaufprinzip
2. Flynns Taxonomie
3. Speicherorganisation
4. Topologie
5. Vektorisierungsfähigkeit
6. Kommerzielle Kategorie
7. Zusatzspezifikationen

Unter Berücksichtigung der logischen Abgeschlossenheit definiert eine Gliederung nach Ebene i eine Aufteilung jedes Segmentes aus Ebene $i-1$ auch dann, wenn auf diese Weise Teilklassen leer bleiben; beispielsweise betrifft das Kriterium *verteilter Speicher* in Ebene 3 nur diejenigen Architekturen aus Ebene 2, die über mehrere Prozessoren verfügen; der Teilraum (Kontrollfluß/SISD/verteilter Speicher) mit seinen Unterklassifizierungen ist also leer.

Ablaufprinzip. Es werden drei Grundprinzipien des Aktionsablaufs unterschieden:

- Kontrollflußprinzip
- Datenflußprinzip
- Systolisches Prinzip

Kontrollfluß bedeutet, daß die Reihenfolge der auszuführenden Operationen durch die Befehlsreihenfolge des Programms vorgegeben wird. Alle herkömmlichen Rechner basieren auf diesem Prinzip. In Datenflußrechnern sind Anweisungen dann ausführbar, wenn alle notwendigen Eingabevariablen vorliegen; die Anordnung der Befehle im Programm ist also ohne Bedeutung. Es können mehrere Befehle gleichzeitig ausführbar sein und beim Vorhandensein mehrerer Prozessoren auch ausgeführt werden. Die

systolischen Architekturen sind gekennzeichnet durch einen rhythmischen Wechsel von Datentransport und arithmetisch-logischen Operationen, der zu ihrer Bezeichnung führte [Kun,82]. Systolische Architekturen sind in der Regel aus einer großen Anzahl sehr einfacher Prozessoren aufgebaut und implementieren einen bestimmten Algorithmus.

Flynns Taxonomie. Es werden vier Architekturklassen unterschieden:

- Ein Befehlsstrom - ein Datenstrom (SISD)
- Ein Befehlsstrom - mehrere Datenströme (SIMD)
- Mehrere Befehlsströme - ein Datenstrom (MISD)
- Mehrere Befehlsströme - mehrere Datenströme (MIMD)

Die Klasse SISD umfaßt Einprozessorrechner und damit die herkömmlichen sogenannten von-Neumann-Rechner, während die anderen Klassen mehrere Prozessoren verlangen, sei es zur gleichzeitigen Bearbeitung verschiedener Datenströme oder zur gleichzeitigen Anwendung verschiedener Operationen auf einen oder mehrere Datenströme. Die Klasse MISD ist de facto leer bis auf einige wenige Spezialrechner. Die Klasse SIMD besteht im wesentlichen aus sogenannten Feldrechnern, deren Prozessorelemente zu jedem Zeitpunkt gleiche Operationen (synchron) auf eigenen Daten ausführen, während die meisten neueren Systeme als MIMD-Rechner auch die gleichzeitige Ausführung unterschiedlicher Befehle gestatten.

Speicherorganisation. Bei Systemen mit mehreren Prozessoren wird unterschieden, ob alle diese Prozessoren auf einen gemeinsamen (globalen) Speicher zugreifen können, der dann auch als Kommunikationsmedium dient, oder ob der Speicher auf die einzelnen Prozessor (lokal) verteilt ist, womit die Interprozessorkommunikation zwischen den Prozessoren durch den Austausch von Nachrichten über Verbindungswege erfolgen muß. Systeme, die sowohl lokalen als auch globalen Speicher besitzen, sind selten.

Topologie. Bei Multiprozessorrechnern mit verteiltem (lokalem) Speicher ist es entscheidend, wie die Verbindungsstruktur zwischen den Prozessoren aufgebaut ist, da sie die Möglichkeiten zur Kommunikation bestimmt. Man unterscheidet zunächst zwischen statischen und dynamischen Verbindungsnetzwerken. Bei statischen Verbindungsnetzwerken liegt die Struktur fest. Der Weg einer Nachricht von Prozessor A nach Prozessor B führt im allgemeinen über mehrere Zwischenstationen, also andere Prozessoren. In dynamischen Netzwerken wird für die Kommunikation zwischen zwei Prozessoren eine Leitung aufgebaut, wofür in der Regel auf mehreren Stufen Schaltelemente betätigt werden müssen. Für die verschiedenen möglichen Topologien wird auf die Klassifikationen von Feng [Fen,81] sowie Broomell und Heath [B&H,83] verwiesen; die wesentlichen statischen Topologien sind auch in Kapitel 3 der vorliegenden Arbeit Objekt der Untersuchungen. Eine Teilklassifizierung dynamischer Verbindungsnetzwerke ist:

- einstufige Netzwerke
- mehrstufige Netzwerke
  - blockierende Netzwerke
  - rekonfigurierbare Netzwerke
  - nicht-blockierende Netzwerke
- Kreuzschienenverteiler

Vektorisierungsfähigkeit. Das Prinzip der Vektorisierung oder des (Daten-)Pipelining kann an dieser Stelle nicht näher erläutert werden, stattdessen sei auf die Literatur verwiesen [H&J,88]. Es wird hier unterschieden in:

- Vektorrechner des Register-Register-Typs
- Vektorrechner des Speicher-Speicher-Typs
- Rechner mit Vektorkoprozessor
- Skalare Architekturen

Die Pipelines der Register-Register-Maschinen laden und speichern Vektoren über Register, während Speicher-Speicher-Maschinen direkt auf den Speicher zugreifen. Zusätzlich besteht die Variationsmöglichkeit, daß ein einzelner Prozessor eine oder mehrere Pipelines besitzen kann.[1]

Kommerzielle Kategorie. Die Übergänge zwischen den einzelnen Kategorien dieser Klassifizierungsebene sind oft fließend, insbesondere wenn Maximalleistungen (*peak performance*) als Maßstab betrachtet werden. Hier wird eine solche Betrachtungsweise im Hinblick auf die Entwicklungszyklen als ungeeignet angesehen; wesentlich wichtiger erscheint die Art und Weise, in der die Rechner eingesetzt werden. So können beispielsweise die Zahl angeschlossener Bildschirme oder typische Anwendungsprogramme herangezogen werden. Unter diesem Aspekt ergeben sich sechs Rechnerkategorien, zu denen sich die überwiegende Mehrheit der ausgelieferten Rechner eindeutig zuordnen lassen:

- Supercomputer
- Minisupercomputer
- Großrechner (*mainframes*)
- Minicomputer
- Arbeitsplatzrechner (*workstations*)
- Personalcomputer

Zusatzspezifikationen. Über die Unterscheidungskriterien der sechs bislang vorgestellten Ebenen hinaus, sind noch eine Reihe weiterer Kriterien denkbar, deren Bedeutung diskutierfähig ist. Sicherlich ist die Anzahl der Prozessoren von Interesse, die mit den

---

1 In der Literatur wird bisweilen ein einzelnes Segment einer Pipeline als vollwertiges Verarbeitungselement (*processing element*) aufgefaßt. Das hat zur Folge, daß die Vektorisierungsfähigkeit die Zuordnung zu Flynns Taxonomie beeinflußt, und führt für einen Vektorrechner entweder zur Klassifizierung SIMD, da der Standpunkt vertreten werden kann, daß mit einem einzigen (Vektor-)Befehl viele einzelne Vektorelemente bearbeitet werden, oder zur Klassifizierung MISD, wenn der gesamte Vektor als ein einheitliches Datum aufgefaßt wird, das zu gleicher Zeit in den verschiedenen Segmenten der Pipeline mit unterschiedlichen Operationen bearbeitet wird. Für die vorgestellte Klassifikation ist eine Pipeline, im Sinne des Vektorprinzips, jedoch nur Teil eines Prozessors und beeinflußt die Zugehörigkeit zu Flynns Klassen nicht. Andererseits sind sehr wohl Pipelines vollständiger Prozessoren denkbar, wie es in systolischen Architekturen anzutreffen ist. Dabei kann der Hardware-Aufwand für das Segment einer Vektor-Pipeline wesentlich größer sein als für einen schlichten Prozessor eines systolischen Feldes, auch kann der Beschleunigungsfaktor durch Vektorisierung oft höher sein als durch Parallelverarbeitung; die hier zugrundeliegende logische Sichtweise zwingt jedoch zu dem vorgestellten Schema.

Topologien nach Ebene 4 zusammenhängt. Auch ist das Vorliegen von Assoziativ- oder Spezialrechnern erwähnenswert. Ebenso können wichtige Architektureigenschaften, wie mehrstufige Speicherhierarchie und lange Instruktionsworte (VLIW), oder Sonderlösungen, wie *decoupled access* und *tag bits* [Gil,81], signifikant sein.

Die Hierarchie der definierten Ebenen resultiert in dem in Abb. 1 dargestellten Klassifikationsschema.

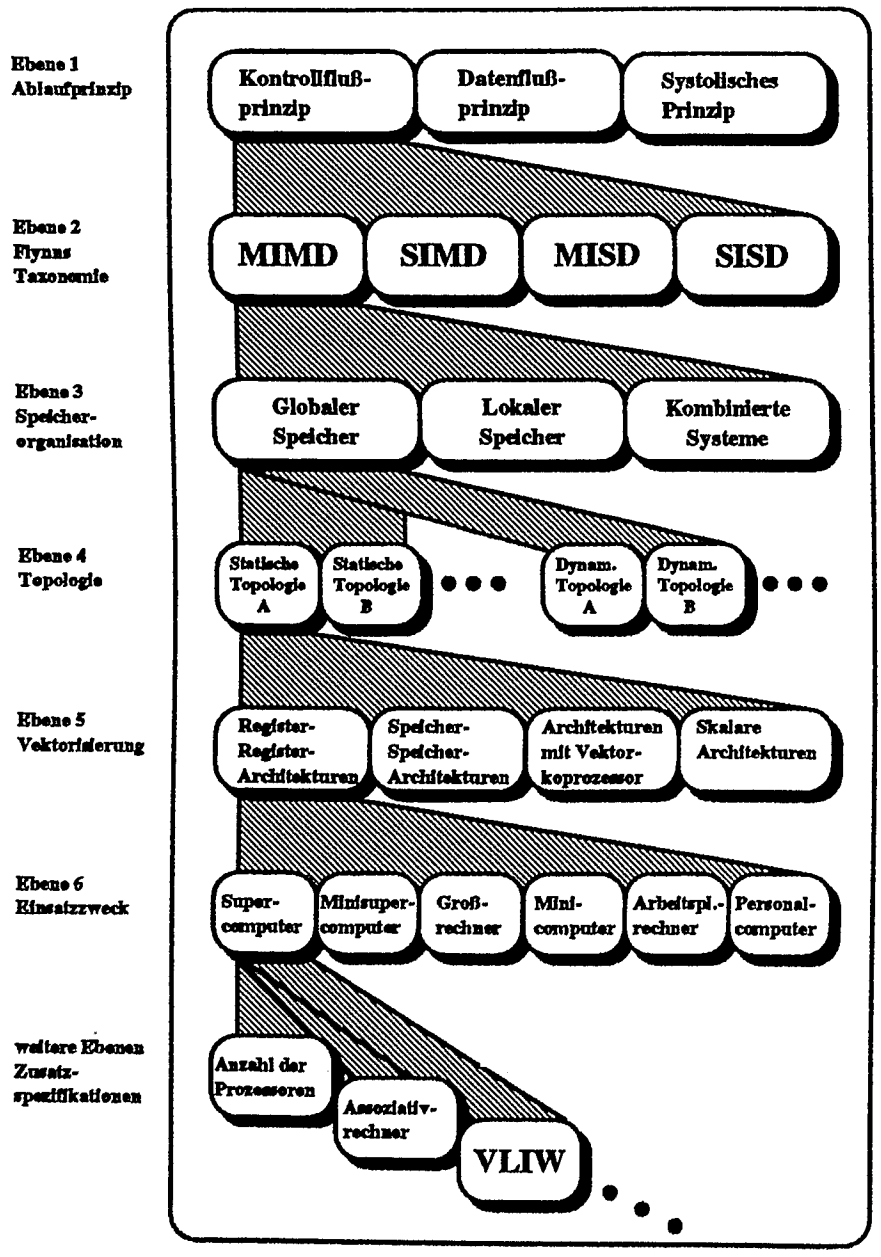


Abb. 1. Klassifizierung von Rechnerarchitekturen

2.2 Entsprechungen zwischen parallelen Strukturen und Versuchsplänen

In der Literatur existiert eine große Zahl von Arbeiten über Zusammenhänge zwischen (gewöhnlichen) Graphen und Netzwerken, die im wesentlichen Optimierungen aufgrund von Grapheneigenschaften anstreben ([Wil,72], [Cat,79], [M&R,82], [Agr,83], [S&R,86], [Y&&,88], [BBB,89], [Sch,91]). Hierbei scheinen Cayley-Graphen als Modelle besonders geeignet ([A&K,86], [LJD,93]). Untersuchungen betreffend den Zusammenhang zwischen Netzwerken und den gegenüber Graphen sehr viel leistungsfähigeren kombinatorischen Versuchsplänen sind kaum anzutreffen. Bermond et al. stellen Überlegungen betreffend der Verwendung von balancierten unvollständigen Blockplänen für den Entwurf von Netzwerken an, wobei sie darauf abzielen, den Durchmesser (vgl. Kapitel 3) des Netzwerks zu minimieren ([B&&,83], [BBS,84]). Während die in Abschnitt 2.3 angeführten Arbeiten die Eigenschaften der Versuchspläne ausschließlich zum Entwurf neuer Topologien nutzen, deutet allein Fiduccia die mögliche Analyse bestehender Netzwerktopologien mit Hilfe kombinatorischer Versuchspläne an [Fid,92].

Sei V eine endliche Menge von v Elementen, so wird eine Kollektion von Teilmengen von V als **Mengensystem** $\mathscr{B}$ auf V bezeichnet. Hierbei ist es im Unterschied zu einer Menge von Mengen gestattet, daß ein Mengensystem eine bestimmte Teilmenge mehr als einmal enthält, anderenfalls heißt das Mengensystem **einfach**.

Es ist auch möglich, daß die Reihenfolge der Elemente in den einzelnen Mengen wesentlich ist, wie beispielsweise bei Lateinischen Quadraten, wo jede Teilmenge alle Elemente der Menge V enthält, so daß man allgemein von **Blöcken** spricht. Da bei der in der vorliegenden Arbeit angestrebten Modellbildung (Kapitel 2 bis 6) die Reihenfolge der Elemente in einem Block ohne Bedeutung ist, werden Blöcke als Mengen behandelt, jedoch aus Gründen der Übersichtlichkeit mit runden Klammern notiert. Daher ist hier die Bezeichnung von $\mathscr{B}$ als Mengensystem korrekt. Es existieren auch wesentlich allgemeinere Definitionen, beispielsweise in der Form von *Inzidenzstrukturen* ([BJL,85], [Dem,70]), die aus einem Tripel von Punkten, Blöcken und Fahnen gebildet werden, oder als *Überdeckungsstrukturen* [OHW,87], für die eine große Zahl von praktischen Anwendungen gegeben ist. Für die vorliegende Arbeit wird die aus dem Paar von Grundmenge und Mengensystem gegebene formale Darstellung als ausreichend betrachtet.

Liegt nun eine endliche Menge V vor, sowie ein darauf basierendes Mengensystem $\mathscr{B}$, so wird das Paar $(V,\mathscr{B})$ als **(kombinatorischer) Versuchsplan** bezeichnet.

Beispiel 2.1

$(V,\mathscr{B})$ mit $V = \{0,1,2,3,A,B,C\}$, $\mathscr{B} = \{(0,1,2,A),(A,B,C),(0),(0,1),(A,B,C)\}$

Ein kombinatorischer Versuchsplan entsteht also durch Auswahl einer Anzahl von Teilmengen, eventuell aufgrund bestimmter Eigenschaften, aus einer gegebenen endlichen Menge. Dabei ist es zunächst zulässig, daß Blöcke einzelner Elemente existieren sowie freie Elemente, die in keinem Block auftreten. Die Mächtigkeit der dem Versuchsplan zugrundeliegenden Menge V wird als **Kardinalität** des Versuchsplans bezeich-

net. Falls alle Blöcke des Versuchsplans $(V,\mathscr{B})$ die gleiche Anzahl k von Elementen enthalten, wird der Versuchsplan als **k-uniform** bezeichnet.

Von den zahlreichen wichtigen Eigenschaften der kombinatorischen Versuchspläne soll hier nur auf die **Balance** eingegangen werden, da dieser für die in der vorliegenden Arbeit beabsichtigte Beschreibung paralleler Strukturen eine zentrale Bedeutung zukommt. Für eine Zusammenstellung aller hier verwendeten Strukturen und Gesetzmäßigkeiten wird auf Anhang A, für darüber hinausgehendes auf die Standardwerke ([BJL,85], [S&S,87], [Wal,88]) verwiesen.

Wählt man eine beliebige Teilmenge $S \subseteq V$ aus, so ist diese eine (nicht notwendigerweise echte) Teilmenge einer Anzahl von Blöcken aus $\mathscr{B}$. Die Anzahl der Blöcke in $\mathscr{B}$, die S enthalten, wird mit $\lambda(S)(\geq 0)$ bezeichnet und heißt der **Index von S in $\mathscr{B}$**. Betrachtet man alle t-elementigen Teilmengen von V, so werden die entsprechenden $\lambda(S)$ in der **t-Index-Menge** $\Lambda_t = \{\lambda(S): |S| = t, S \subseteq V\}$ des Mengensystems zusammengefaßt.

Beispiel 2.1 (Fortsetzung)

$S = \{0,1\} \Rightarrow \lambda(S) = 2$ wegen $S \subseteq (0,1,2,A)$, $S \subseteq (0,1)$
$\Lambda_3 = \{0,1,2\}$, da

- die dreielementige Menge {A,B,C} zweimal in den Blöcken von $\mathscr{B}$ auftritt
- die dreielementige Menge {0,1,2} einmal in den Blöcken von $\mathscr{B}$ auftritt
- alle übrigen dreielementigen Mengen in den Blöcken von $\mathscr{B}$ nicht auftreten

Ein Mengensystem heißt **t-balanciert**, falls seine t-Index-Menge nur einen einzigen Wert $\lambda_t > 0$ enthält. Ein 1-balanciertes Mengensystem zeichnet sich lediglich dadurch aus, daß (Einzel-)Elemente gleichhäufig in den Blöcken von $\mathscr{B}$ vorkommen. Jedes Mengensystem ist 0-balanciert, wobei λ_0 der Zahl der Blöcke entspricht. Ein t-balanciertes k-uniformes Mengensystem ist ebenso $(t-1)$-balanciert. Das zu Abb. 4 auf Seite 21 gehörige Beispiel 2.3 zeigt einen Versuchsplan, der 2-balanciert ist; alle ungeordneten Paare von Elementen von V treten genau zweimal in den Blöcken dieses Versuchsplans auf.

Mit diesen Definitionen gelangt man schließlich zur wichtigsten Struktur im Bereich kombinatorischer Versuchspläne: Ein **balancierter unvollständiger Blockplan (balanced incomplete block design, BIBD)** ist ein Paar $(V,\mathscr{B})$, wobei $\mathscr{B}$ ein k-uniformes Mengensystem auf V darstellt, das 2-balanciert mit Index $\lambda = \lambda_2$ ist. $(V,\mathscr{B})$ wird dann als (v,k,λ)-Plan bezeichnet. Neben dem BIBD existiert eine große Zahl weiterer, vereinfachter oder speziellerer, Strukturen, für die auf Anhang A verwiesen wird.

Im Rahmen dieser Arbeit werden ausschließlich Multiprozessorrechner nach dem Kontrollflußprinzip mit verteiltem Speicher und statischem Verbindungsnetzwerk entsprechend der in Abschnitt 2.1 gegebenen Klassifikation betrachtet, wobei sowohl SIMD- als auch MIMD-Rechner zugelassen sind. Die Bezeichnungen „Parallelrechner" sowie „Netzwerk" beschränken sich im folgenden auf diese Restriktion.[2] Ein **Netzwerk**

---

2 Es erscheint hier sinnvoller, eine Restriktion auf Rechner mit statischem Verbindungsnetzwerk vorzunehmen und innerhalb dieser eine umfassende Behandlung zu liefern, als eine auf einzelne

im Sinne der durchgeführten Restriktion besteht somit aus einer Menge von **(Prozessor-)Knoten** sowie aus zwischen diesen Knoten bestehenden **Verbindungen**. Eine Verbindung für genau zwei Knoten wird als **Link**, eine Verbindung für mehr als zwei Knoten wird als **Bus** bezeichnet. Der Anschluß eines Knotens an eine Verbindung heißt **Port**. Die Zahl der Ports, über die ein Knoten insgesamt verfügt, wird mit **Grad** bezeichnet. Verbindungen sollen im folgenden bidirektional nutzbar sein, womit eine Unterscheidung in Innen- und Außengrad hinfällig wird. Die Kapazität von Bussen wird mit 1 Paket pro Zeiteinheit angenommen, die der Links mit 1 Paket pro Zeiteinheit für jede Richtung (**Vollduplexbetrieb**). Darüber hinaus werden ausreichende Pufferungsmöglichkeiten vorausgesetzt.

Ausgangspunkt für die vorliegende Arbeit ist die Auffassung solcher Netzwerke als kombinatorische Versuchspläne. Hierbei entsprechen die v Elemente den Knoten, die b Blöcke den vorhandenen Verbindungen. Die Elemente jedes Blocks spezifizieren die Knoten, die auf der entsprechenden Verbindung kommunizieren können. Die jeweilige Blockgröße gibt also die Zahl der an diese Verbindung angeschlossenen Knoten an, während der Wiederholungsfaktor eines Elementes (vgl. Anhang A) die Zahl der Anschlüsse eines Knotens spezifiziert. Um auch Netzwerke mit Busverbindungen in ein einheitliches Schema integrieren zu können, sind herkömmliche Graphen ungeeignet, da sie einen (2-uniformen) Spezialfall kombinatorischer Versuchspläne darstellen, weshalb man die Versuchspläne auch als *Hypergraphen* bezeichnet (vgl. Anhang A). Es wird deutlich, daß diejenige Eigenschaft der Strukturen, die in der Kombinatorik mit Balance bezeichnet wird, auch für die Netzwerke von besonderer Wichtigkeit ist, da sie hier darüber Auskunft gibt, ob zwischen einer festen Zahl beliebiger Knoten eine identische Verknüpfungsdichte vorliegt. Existiert also beispielsweise ein (konstantes) λ_2, so besagt dies, daß zwischen allen Knoten*paaren* gleich viele Verbindungen bestehen. Dies ist insbesondere für Netzwerke mit unterschiedlichen Verbindungsarten von Interesse. Tab. 1 zeigt die Zusammenhänge zwischen Versuchsplan- und Netzwerkparametern.

Versuchsplan	formal	Parallelrechner
Element	a_i	Prozessor
Anzahl der Elemente	v	Anzahl der Prozessoren
Block	$B = (a_1, ..., a_m)$	Verbindung
Anzahl der Blöcke	b	Anzahl der Verbindungen
Blockgröße	k	Anzahl der Prozessoren der Verbindung
Wiederholungsfaktor	r_x	Anzahl der Ports eines Prozessors
Balance	λ	spezifische Verknüpfungsdichte

Tab. 1. Übertragung der Parameter

Aspekte beschränkte Behandlung sowohl statischer als auch dynamischer Netzwerke durchzuführen. Eine universelle Behandlung sowohl beider Kategorien als auch aller Aufgabenfelder würde den Rahmen dieser Arbeit übersteigen. Auf die speziellen numerischen Fähigkeiten der Prozessoren soll in der Arbeit nicht eingegangen werden, womit die Klassifizierungsebenen hinsichtlich Vektorisierungsfähigkeit und kommerzieller Verwendung für die vorliegende Arbeit ohne Bedeutung sind.

Die Auffassung eines Multiprozessorrechners mit statischem Verbindungsnetzwerk als Versuchsplan wird in Beispiel 2.2 erläutert.

Beispiel 2.2

Das in Abb. 2 dargestellte Netzwerk mit zwei Bussen und vier Links entspricht dem Versuchsplan $(V,\mathscr{B})$ mit
$V = \{0,1,2,3,4,5,6,7\}$, $\mathscr{B} = \{(0,1,2,3),(4,5,6,7),(0,5),(1,4),(2,7),(3,6)\}$.

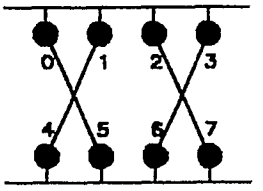


Abb. 2. Verbindungsnetzwerk mit 2 Bussen und 4 Links

Aufgrund der widerstrebenden Zielfunktionen bei der Entwicklung von Verbindungsnetzwerken, der effizienten Kommunikation einerseits, d. h. Maximierung direkter Knotenverbindungen, und der möglichst kostengünstigen Herstellung andererseits, d. h. Minimierung direkter Verbindungen, ist man beim Entwurf derartiger Systeme naturgemäß an einem in beider Hinsicht befriedigenden Kompromiß interessiert. Faßt man nun ein Netzwerk als kombinatorischen Versuchsplan auf, so ist es möglich, die Gesetzmäßigkeiten der kombinatorischen Theorie der optimalen Versuchspläne für die Lösung des Problems heranzuziehen.

Sieht man Prozessorknoten als die Elemente und Knotenverbindungen als Blöcke eines kombinatorischen Versuchsplans an, wie dies in der vorliegenden Arbeit vorausgesetzt wird, so kann man sich das aus der Kombinatorik bekannte Faktum zunutze machen, daß in einem *symmetrischen* Plan (vgl. Anhang A) – für den Bereich der Blockpläne – die Zahl der Blöcke (also der Verbindungen, d.h. der Kosten) minimal wird.

Ähnlich wie die Behandlung der Rechner kann auch die formale Behandlung paralleler Algorithmen durchgeführt werden. Faßt man einen parallelen Algorithmus als eine Menge separat ausführbarer Prozesse auf, so können diese als Grundmenge eines Versuchsplans angesehen werden, dessen Mengensystem die Kommunikation von Prozessen beschreibt; mit anderen Worten: Paarweise untereinander kommunizierende Prozesse bilden einen gemeinsamen Block im Mengensystem. Hierbei ist es, wie in Kapitel 4 ausgeführt, von Fall zu Fall möglich und sinnvoll, den parallelen Algorithmus in einzelne Phasen aufzutrennen und für jede Einzelphase einen eigenen Versuchsplan bereitzustellen. Der parallele Algorithmus wird dann insgesamt durch eine Folge von Versuchsplänen und zwischen diesen bestehenden Relationen beschrieben.

Die sowohl für parallele Rechner als auch Algorithmen somit mögliche Beschreibung durch Versuchspläne induziert in beiden Bereichen eine Klassifizierung hinsichtlich kombinatorischer Strukturen, die für viele Problemstellungen, insbesondere aber das

Mapping (vgl. Kapitel 5), genutzt werden kann. Darüber hinaus ist so aber auch eine allgemeine Anwendung von Gesetzmäßigkeiten der Kombinatorik optimaler Versuchspläne in der Parallelverarbeitung möglich und die zahlreichen Instrumente, die die ausgereifte Theorie der Kombinatorik bereitstellt, werden für die Probleme im Bereich Parallelität nutzbar. Interessante, zum Teil ungelöste Problemstellungen aus der Parallelverarbeitung können so auf allgemein gelöste Probleme aus der Kombinatorik zurückgeführt werden.

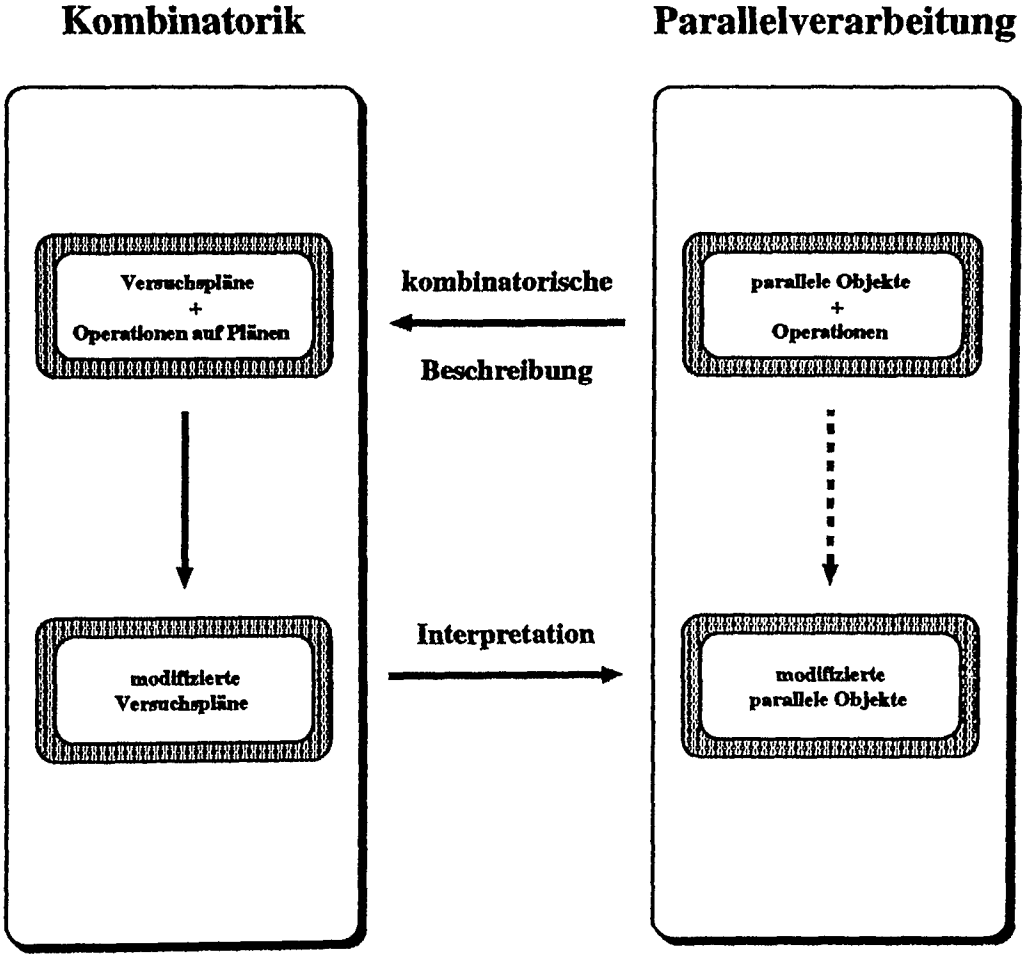


Abb. 3. Transformationstechnik: Rückführung von Problemstellungen der Parallelverarbeitung auf bekannte Probleme der Kombinatorik

Für die Übertragung der Gesetzmäßigkeiten aus der Kombinatorik der optimalen Versuchspläne in die Parallelverarbeitung nutzt die vorliegende Arbeit die in Abb. 3 dargestellte Transformationstechnik. Für eine Aufgabenstellung, die sich beispielsweise auf die parallelen Strukturen $A_1, \ldots, A_n$ und die Operation Φ im Bereich Parallelverarbeitung bezieht, transformiert man die Strukturen in ihr Versuchsplanäquivalent $(V,\mathcal{B})_1,\ldots,(V,\mathcal{B})_n$ mit $(V,\mathcal{B})_i = A_i^T$, $i = 1,\ldots,n$ und wendet die transformierte Operation Φ^T im Kombinatorikbereich an. Die auf diesem Wege im Kombinatorikbereich als modifizierte Versuchpläne erhaltene Lösung ist dann durch „Interpretation" in den Parallelverarbeitungsbereich zurückzutransformieren und führt so zur Lösung des eigentlichen Pro-

blems. Diese Technik wird in der vorliegenden Arbeit nur für einzelne Kernbereiche genutzt, die Möglichkeiten sind damit aber bei weitem nicht ausgeschöpft.

2.3 Parallelrechnerarchitekturen auf Versuchsplanbasis

In diesem Abschnitt wird mit Ergebnissen aus der Literatur verdeutlicht, daß die kombinatorischen Versuchspläne nicht nur zur Klassifizierung bestehender sondern ebenso zum Entwurf neuer effizienter Netzwerke dienen können, womit ein weiteres Indiz für die strukturelle Verwandschaft von parallelen Objekten und Versuchplänen gefunden ist.

2.3.1 Topologien auf der Basis projektiver Ebenen

Eine spezielle Form eines symmetrischen Versuchsplans stellen projektive Ebenen dar. Überlegungen zur Verwendung projektiver Ebenen als Grundlage für den Entwurf von Rechnertopologien sind von Mickunas [Mic,80] vorgenommen worden. Eine projektive Ebene (vgl. Anhang A) ist ein symmetrischer kombinatorischer Versuchsplan (d. h. $v = b$), für den die Parameter (v,k,λ) wegen $\lambda = 1$ die Form $(k^2-k+1,k,1)$ haben. Mit $\mu = k - \lambda$ haben die Parameter die Form $(\mu^2+\mu+1, \mu+1, 1)$, und die Ebene heißt dann **von der Ordnung μ**. Äquivalent zu dieser kombinatorischen Formalisierung der projektiven Ebene ist die im allgemeinen besser bekannte geometrische Definition, die auf der projektiven Inzidenzebene und der Gültigkeit des Satzes von Pappos-Pascal beruht [Kli,92].

Beispiel 2.3: Projektive Ebene der Ordnung 2

Eine projektive Ebene der Ordnung $\mu = 2$ ist ein 2-(7,3,1)-Plan $(V,\mathscr{B})$, d. h. die Zahl der Elemente beträgt $v = 7$, die Blockgröße beträgt $k = 3$ und wegen $\lambda(=\lambda_2) = 1$ kommt jedes ungeordnete Paar genau einmal in $\mathscr{B}$ vor. Für diesen in Abb. 4 dargestellten Versuchsplan ergibt sich:
$V = \{0,1,2,3,4,5,6\}$
$\mathscr{B} = \{(0,1,2),(2,3,4),(4,5,0),(0,6,3),(1,6,4),(2,6,5),(1,3,5)\}$

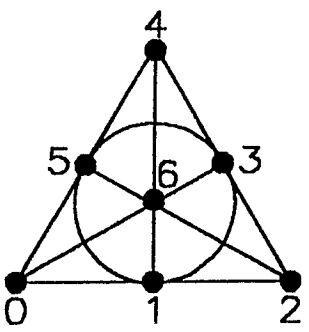


Abb. 4. Projektive Ebene der Ordnung 2

Das Beispiel zeigt, daß, mit Ausnahme der projektiven Ebene der Ordnung 1 mit drei Prozessoren, Busse anstelle von Links Verwendung finden. Busse beinhalten die Möglichkeit, die Zahl der direkt verbundenen Knoten zu erhöhen und damit die mittlere Weglänge zu minimieren, ohne die Zahl der Verbindungen über Gebühr zu erhöhen.

Dieses Vorgehen birgt jedoch die Gefahr der Überlastung von Bussen. Neben der Zahl der angeschlossenen Knoten pro Bus ist auch die Zahl der Busse von Interesse, zu denen ein einzelner Knoten Zugang besitzt. Es ist offensichtlich, daß eine Balance des Systems dahingehend wünschenswert ist, daß die Zahl der Knoten pro Bus nicht allzu sehr differieren sollte.

Unter Berücksichtigung der in Anhang A aufgeführten Ergebnisse über symmetrische Versuchspläne, insbesondere also auch projektive Ebenen, ist festzustellen, daß diese Strukturen aufgrund ihrer Balanciertheit, wobei hier kombinatorische und praktische Bedeutung des Begriffes zusammenfallen, und ihrer Symmetrie den Optimalitätsanforderungen in idealer Weise entsprechen, was die Entwicklung entsprechender Hardware nahelegt; dies ist jedoch noch nicht geschehen. Der Balancewert λ ist bei den projektiven Ebenen stets 1. Das bedeutet, daß jedes ungeordnete Paar von Elementen aus V, also hier: Paar von Knoten, genau einmal innerhalb der Blöcke des Versuchsplans auftritt; die jeweils zu wählende Verbindung für das Versenden einer Nachricht zwischen zwei Knoten ist so eindeutig bestimmbar. Damit wird das Routing in einer solchen Topologie sehr einfach.

Für den allgemeinen Entwurf projektiver Ebenen kann auf die konstruktive Theorie projektiver Geometrien zurückgegriffen werden: Es ist möglich, für jedes μ, das als Primzahlpotenz darstellbar ist, eine projektive Ebene der Ordnung μ derart zu konstruieren, daß

1. die projektive Ebene $(V,\mathscr{B})$ genau $v = \mu^2 + \mu + 1$ Knoten enthält,
2. die projektive Ebene $(V,\mathscr{B})$ genau $b = \mu^2 + \mu + 1$ Verbindungen enthält,
3. je zwei verschiedene Knoten genau eine gemeinsame Verbindung haben,
4. je zwei verschiedene Verbindungen genau einen gemeinsamen Knoten haben,
5. jeder Knoten mit genau $\mu + 1$ Verbindungen inzidiert, d.h. $r_x = \mu + 1 \; \forall x \in V$ und
6. jede Verbindung mit genau $\mu + 1$ Knoten inzidiert, d.h. $k = \mu + 1 \; \forall B \in \mathscr{B}$.

Wie insgesamt für die Anwendbarkeit kombinatorischer Versuchspläne in der Parallelverarbeitung gilt auch bei der Verwendung projektiver Ebenen, daß auf diesem Wege viele Ergebnisse der zugehörigen Mathematik für die Informatik erschlossen werden. Für den Fall, daß die Zahl der erforderlichen Knoten nicht der Zahl der Knoten einer möglichen projektiven Ebene entspricht, können abgeleitete Pläne bzw. Residuenpläne (vgl. Anhang A) der projektiven Ebene mit ausreichender Ordnung so ausgewählt werden, daß eine hinreichende Balance des Systems gegeben ist [Mic,80].

Falls die Kommunikationsanforderungen an ein zu entwickelndes Netzwerk ein hohes Maß lokaler Kommunikation innerhalb von Subnetzwerken, jedoch nur wenig globale Kommunikation reflektieren, kann man wiederum auf ein Ergebnis aus der Theorie zurückgreifen; es besagt, daß in einer gegebenen projektiven Ebene, die durch ein Galois-Feld induziert wird, die projektive Ebene der Ordnung $\mu = 2$, die auch als *Fano-Konfiguration* bezeichnet wird, häufig als Substruktur auftritt, falls innerhalb des Galois-Feldes bestimmte Voraussetzungen vorliegen.

2.3.2 Topologien mit mehreren gemeinsamen Speichern

Es gilt allgemein als erwiesen, daß das Prinzip des gemeinsamen Speichers bei einer großen Zahl von Prozessoren nicht mehr effizient umgesetzt werden kann, so daß man zu Systemen mit ausschließlich lokalen Speichern übergehen muß, wiewohl die Programmierung dieser Rechner mit verteiltem Speicher wesentlich aufwendiger ist. Hagemann [Hag,91] schlägt daher als Alternative ein Konzept vor, in welchem mehrere gemeinsame Speicher existieren, so daß je zwei Prozessoren zumindest einen gemeinsamen Speicher haben.

Ähnlich wie bei Mickunas basiert die Entwurfsstrategie von Hagemann im wesentlichen auf projektiven Ebenen, aber auch auf affinen Ebenen sowie verwandten Klassen von Blockplänen. Eine Zusammenstellung geeigneter Strukturen kann dem Anhang des Standardwerks von Beth, Jungnickel und Lenz [BJL,85] entnommen werden. Bezüglich der Konstruktion derartiger kombinatorischer Strukturen verweist Hagemann explizit auf die Möglichkeit der Generierung mit Hilfe von Differenzmengen (vgl. Anhang A). Der grundlegende Unterschied zum Vorgehen von Mickunas besteht darin, daß hier die in Blöcken (geometrische Diktion: Linien) zusammengefaßten Elemente (Knoten) nicht als über einen Bus gekoppelt und über lokalen Speicher verfügend betrachtet werden. Vielmehr bedeutet das Enthaltensein in einem gemeinsamen Block für die Prozessoren der Architektur den Zugriff auf einen gemeinsamen Speicher. Die Zahl der notwendigen gemeinsamen Speicher entspricht dann der Zahl der Blöcke des Versuchsplans.

Wird beispielsweise die in Abb. 4 dargestellte projektive Ebene der Ordnung $\mu = 2$ als Entwurfsgrundlage herangezogen, so verfügt der zu konstruierende Rechner über 7 Speicher, die jeweils drei Prozessoren gemeinsam sind. Wie im Entwurfskonzept von Mickunas ist dann aufgrund des Balancewertes $\lambda = 1$ für je zwei kommunzierende Prozessoren der zu benutzende gemeinsame Speicher eindeutig. Das Verfahren von Hagemann erscheint jedoch nur für Systeme mittlerer Parallelität geeignet und nicht für die hier primär betrachteten Systeme hoher bzw. massiver Parallelität.

2.3.3 Versuchsplanarchitekturen

Die von Beth und Hatz vorgeschlagenen Versuchsplanarchitekturen (*design machines*) ([B&H,92a], [Hat,92]) stellen einen weiteren Ansatz dar, um die günstigen Eigenschaften kombinatorischer Versuchspläne zum Entwurf neuer Architekturen zu nutzen. Die Blöcke der Versuchspläne werden bei Hatz weder durch Busse noch durch gemeinsame Speicher realisiert, sondern es wird auf einen Spezialbaustein, den sogenannten *Rotor*, zurückgegriffen.

Ein **Rotor** [B&H,91] ist ein Permutationsbaustein, der topologisch einem Kreuzschienenverteiler entspricht, d.h. alle an ein und denselben Rotor angeschlossenen Prozessoren sind direkt miteinander verbunden, dies jedoch nur für einen gewissen Zeitraum. Basierend auf einer 1-Faktorisierung (vgl. Anhang A) des dem Rotor zugrundeliegenden vollständigen Graphen gerader Knotenzahl werden, entsprechend der Zahl r der 1-Faktoren, in einem aus r Takten bestehenden Zyklus alle diese Faktoren durchlaufen, so daß während eines Gesamtzyklus alle 1-1-Verbindungen einmal realisiert werden. Das verwendete Rotationsprinzip ist bereits in dem Buch von Ahrens [Ahr,18] zu finden.

Kann man sicherstellen, daß der Rotortakt dem r-fachen des Taktes des Gesamtnetzwerks entspricht, so wird virtuell die Funktion eines Kreuzschienenverteilers bzw. eines vollständig verbundenen Graphen zur Verfügung gestellt. Gegenüber der standardmäßigen (Voll-)Implementierung des Kreuzschienenverteilers hat der Rotor einen wesentlich geringeren Flächenbedarf bezüglich der Herstellung als VLSI-Baustein. Rotoren sind bereits in 2μ-CMOS-Technologie konstruiert und erfolgreich getestet worden [B&H,91].

Die in einem Block eines geeigneten Versuchsplans befindlichen Elemente werden über einen dedizierten Rotor miteinander verbunden; die Zahl der notwendigen Rotoren entspricht also der Zahl der Blöcke des zugrundeliegenden Versuchsplans.

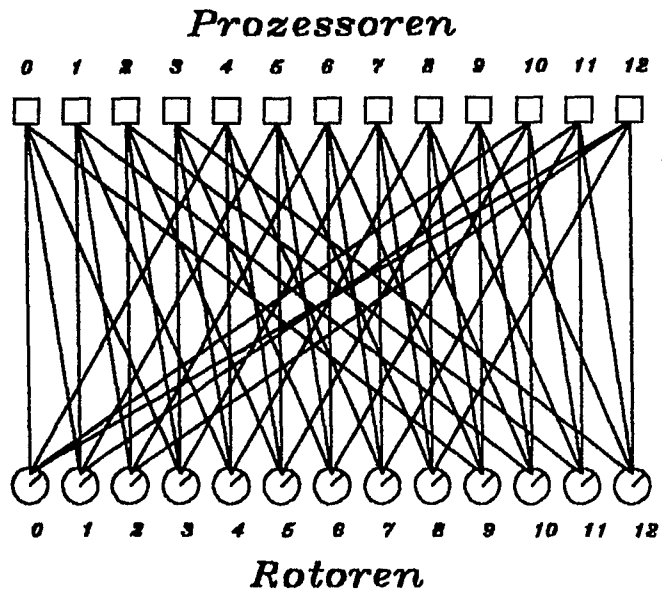


Abb. 5. Design Machine DM(13,4): Darstellung des zugrundeliegenden Versuchsplans als bipartiter Graph.

Beispiel 2.4: Design Machine DM(13,4)

Die Versuchsplanarchitektur DM(13,4) ist durch das Tupel $(Z_{13}, Z_{13}, PG(3,3), T_0, T_1, T_2)$ gekennzeichnet. Hierbei liefern die ersten beiden Parameter die Zahl der in der Topologie befindlichen Prozessoren und Rotoren, die also jeweils mit den Indizes $\{0,1,...,12\}$ zu versehen sind. Der dritte Parameter, hier die projektive Ebene der Ordnung 3, gibt an, wie Prozessoren und Rotoren miteinander zu verknüpfen sind, und die übrigen Parameter T_i bezeichnen die jeweils in den unterschiedlichen Rotorstellungen sich ergebenden Topologien. Bei der Darstellungsform des Versuchsplans als bipartiter Graph (vgl. Anhang A) ergibt sich dann die in Abb. 5 verdeutlichte *Design Machine* DM(13,4), wobei 13 die Zahl der Prozessoren der Architektur bezeichnet, während 4 der Zahl der Ports pro Prozessor entspricht.

Mit der Einführung des Rotor-Permutationsbausteins scheint ein vielversprechender Ansatz gefunden, um die Zahl der Ports pro Prozessor – der wesentliche Kostenfaktor

eines Entwurfs – zu reduzieren, ohne die Busse zu überlasten. Da Beth und Hatz für ihre Implementierungsversuche den Transputer als Prozessor vorsehen, sind Versuchspläne mit Blockgröße 4 naturgemäß von besonderem Interesse. Die von ihnen geplanten experimentellen Untersuchungen werden zeigen, ob der Rotorbaustein und die darauf basierenden *Design Machines* die hohen Erwartungen erfüllen können.

2.3.4 Permutationsnetzwerke auf der Basis von Differenzmengen

Kilian, Kipnis und Leiserson [KKL,87] beschreiben die Konstruktion von speziellen Permutationsnetzwerken, sogenannten zyklischen Shift-Netzwerken (*cyclic shifter, barrel shifter*), deren Konstruktion sich durch die Generierung zyklischer symmetrischer BIBDs aus einer Differenzmenge als Basisblock (vgl. Anhang A) ergibt. Abb. 6 zeigt als Beispiel ein Shift-Netzwerk basierend auf der Differenzmenge {0,1,3,9} für Z_{13}.

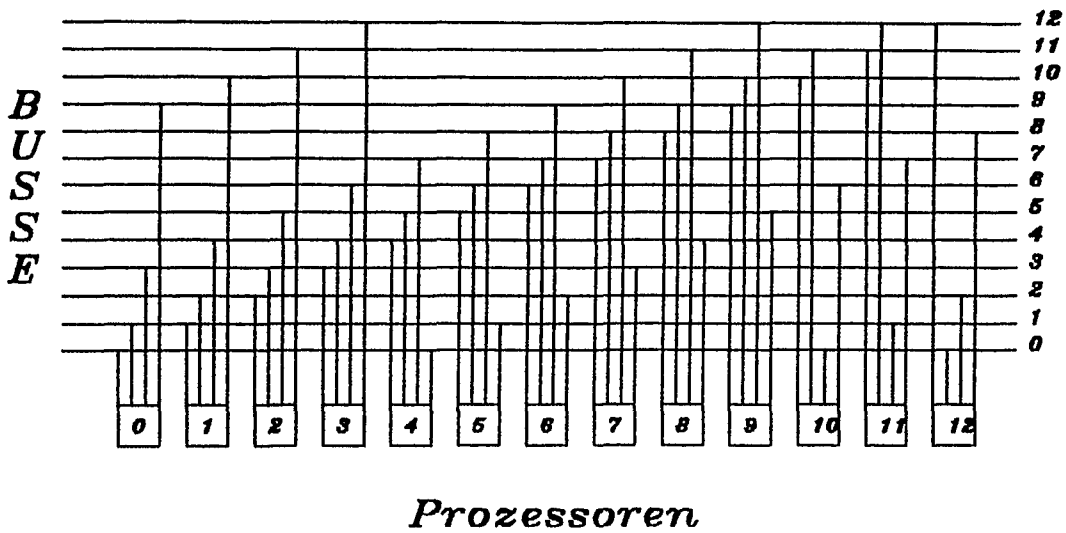


Abb. 6. Zyklisches Shift-Netzwerk für 13 Prozessoren

Das in Abb. 6 gezeigte Beispiel ist nach den in Anhang A dargestellten Zusammenhängen nur eine andere Darstellungsform des dem Rotor-Netzwerk in Abb. 5 zugrundeliegenden symmetrischen Versuchsplans.

Wie in den oben dargestellten Entwurfskonzepten ist auch hier die Kostenreduktion das vordringliche Ziel der Konstrukteure, die insbesondere auf die notwendige Anzahl von Pins pro Chip abheben. Hier ergeben sich folgende Aussagen [KKL,87], die sich aus der Kombinatorik optimaler Versuchspläne, speziell der Differenzmengen, ergeben:

- Falls die Menge der Permutationen eine Gruppe mit p Elementen bildet, kann jede Permutation in dieser Gruppe in 1 Zeiteinheit durch einen Entwurf mit $O(\sqrt{p \log p})$ Pins pro Chip realisiert werden.

- Ist die Gruppe abelsch (kommutativ), so reichen $O(\sqrt{p})$ Pins pro Chip aus.
- Es existiert eine uniforme Topologie (d.h. die Zahl der an einen Bus angeschlossenen Prozessoren ist für alle Busse konstant) für eine Permutationsmenge Π mit k Pins pro Chip genau dann, wenn es eine Differenzmenge für Π mit k Elementen gibt.

3 Kombinatorische Klassifizierung statischer Netzwerke

Netzwerke, wobei sich diese Bezeichnung entsprechend der eingeführten Restriktion ausschließlich auf statische Verbindungsnetzwerke von Multiprozessorrechnern mit verteiltem Speicher bezieht, werden im folgenden mit einem festen Schema erfaßt:

- In Abschnitt A. werden weitere Bezeichnungen der Topologie angegeben, um die betreffenden Bezüge herzustellen. Außerdem wird auf eine von Fall zu Fall vorzunehmende Spezialisierung hingewiesen; aufgrund der Fülle der Sonderformen können hier nur prinzipielle Strukturen aufgenommen werden. Zusätzlich werden die Topologien mit einer 3-Buchstaben-Kurzform versehen, um die formale Behandlung zu erleichtern.

- In Abschnitt B. wird auf Implementierungen der jeweiligen Architektur verwiesen, um den Bezug zur Praxis herzustellen und insbesondere die Verwendung der verschiedenen Topologien durch die Rechnerhersteller zu verdeutlichen.

- In Abschnitt C. erfolgt schließlich die kombinatorische Spezifikation:

  1. Hierzu wird zunächst ein Abbildungsbeispiel gegeben, um die Konstruktionsprinzipien zu veranschaulichen.
  2. Dann werden die am speziellen Beispiel gefundenen Versuchsplanparameter, Zahl der Knoten (v), Zahl der Blöcke (b), Blockgröße (k) und Wiederholungsfaktor (r_x), auf den allgemeinen Fall erweitert bzw. durch die Expansionsstufe (Ausbaustufe, Dimension) n der Topologie parametrisiert.
  3. Darauf erfolgt die Bestimmung der für Netzwerke wichtigen Kenngrößen Verknüpfungsdichte (d), Verbindungskosten (C), Redundanz (ρ), Durchmesser (Δ_{max}), mittlere Weglänge (δ), Verkehrsdichte (ψ) und Bandbreite (ω). Abschließend erfolgt eine Aussage darüber, wie die Anzahl der Prozessoren der Topologie von Ausbaustufe zu Ausbaustufe anwächst.
  4. Sodann wird die aus Sicht der kombinatorischen Versuchspläne wichtigste Eigenschaft, die Balance des Systems, analysiert.
  5. Damit gelangt man schließlich zu einer Klassifizierung im Sinne der Theorie der kombinatorischen Versuchspläne, die deren Anwendung für die Netzwerktheorie ermöglicht.

Innerhalb des Bereichs der partiell balancierten unvollständigen Blockpläne (PBIBD) (vgl. Anhang A) wird bei den Verdichtungen (*packings*) noch einmal zwischen Linear-, Polynomial- und Exponentialverdichtungen unterschieden. Diese Bezeichnungen beziehen sich auf die Form, in der die Expansionsstufe n in den Parameter v des Parameterquadrupels eingeht. Die Betonung dieses Zusammenhangs erscheint aus kombinatorischer Sicht zunächst ungewöhnlich, ist jedoch bei der in Kapitel 5 eingeführten Methode des strukturorientierten Mapping eine wertvolle Hilfe.

Da die Kenngrößenbezeichnung in der Literatur nicht einheitlich erfolgt, werden sie an dieser Stelle definierend zusammengestellt. Die **Verknüpfungsdichte** d ermittelt sich aus der Division der Zahl der Verbindungen durch die Zahl der Knoten. Die **Verbindungskosten** C repräsentieren das gleiche Verhältnis in der Schreibweise von Komplexitätsordnungen. Die **Redundanz** ρ beschreibt die Fähigkeit des Netzwerks, nach Ausfall einer Verbindung noch Kommunikation zu gewährleisten. Sie ermittelt sich folglich aus der Minimalzahl von Verbindungen zwischen je zwei Knoten minus 1

($\rho = \min \{r_x \mid x \in V\} - 1$). Der **Durchmesser** Δ_{max} bezeichnet die maximale Distanz Δ (vgl. Kapitel 5) zwischen zwei beliebigen Knoten im Netz. Die durchschnittliche Distanz wird durch die **mittlere Weglänge** δ verkörpert. Die **Verkehrsdichte** ψ errechnet sich aus dem Produkt von mittlerer Weglänge und der Gesamtzahl aller Knoten im Netzwerk. Je größer die durchschnittliche Verkehrsdichte eines Netzwerks desto größer ist die Wahrscheinlichkeit der Verzögerung der Nachricht aufgrund der Blockierung der Verbindung durch eine andere Nachricht [RLH,88]. Die **Bandbreite** ω bestimmt die Zahl der Nachrichten, die gleichzeitig im gesamten Netzwerk empfangen bzw. gesendet werden können. Beispielsweise errechnet sich die Bandbreite bei auf Links basierenden Netzwerken mit konstanter Portzahl pro Prozessor und angenommener bidirektionaler Nutzbarkeit der Links aus dem Produkt der Anzahl der Prozessoren mit der Zahl der Ports pro Prozessor. Die Untersuchung der Expansionsfähigkeit erfolgt nur in logischer Hinsicht, d.h. ohne Berücksichtigung technologischer Probleme.

Bezüglich der Aussagekraft der verwendeten Kenngrößen zur Beurteilung der Güte einer Topologie wird hier nur festgestellt, daß es sicherlich falsch ist, mit Hilfe von Kenngrößen nach einer universal besten Topologie zu suchen; nur für eine bestimmte Anwendung kann nach der jeweils besten Zieltopologie gesucht werden. Levitan untersucht die Eignung verschiedener Kenngrößen zur Beurteilung von Netzwerken [Lev,85]. Für eine vergleichende Analyse von Netzwerken auf der Basis von Kenngrößen können die Arbeiten von Wittie [Wit,81], Bhuyan [Bhu,84] und Mudge et al. [M&&84] herangezogen werden. Saad und Schultz vergleichen die wesentlichen hier aufgeführten Topologien anhand der Komplexitäten bezüglich verschiedener Kommunikationsformen [S&S,89].

Für die in diesem Kapitel ohne Literaturreferenz angeführten Implementierungen wird auf einen Bericht von Dongarra und Duff verwiesen [D&D,89].

3.1 Balancierte unvollständige Blockpläne (BIBD)

3.1.1 Einfache Bussysteme

A. Weitere Bezeichnungen.
Simple Bus Computer, SBC

B. Implementierungen.
CULLER 8, (Metastruktur der) CDC CYBERPLUS, ELXSI 6400, Encore Multimax 310/320, FLEX/32 Multicomputer

C. Kombinatorische Spezifikation.

(1) Beispiel.
Das in Abb. 7 dargestellte einfache Bussystem der Stufe $n = 4$ entspricht folgendem Versuchsplan:
$V = \{0,1,2,3\}$
$\mathscr{B} = \{(0,1,2,3)\}$
$v = 4, b = 1, r_x = 1\ \forall x \in V$
$\Lambda_2 = \{\lambda(S): |S| = 2, S \subseteq V\} = \{1\}$

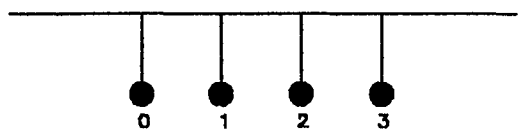


Abb. 7. Einfaches Bussystem der Stufe 4

(2) Versuchsplanparameter. Ein SBC $(V,\mathscr{B})$ der Stufe n besitzt $v = n$ Knoten und $b = 1$ Block der Größe $k = n$. Der Wiederholungsfaktor beträgt $r_x = 1\ \forall x \in V$.

(3) Kenngrößen. Die Verknüpfungsdichte des SBC beträgt $d = 1/v$, die Redundanz ist $\rho = 0$, der Durchmesser ist $\Delta_{max} = 1$ und die Bandbreite ist $\omega = v$. Die Verbindungskosten betragen $O(1/v)$, die mittlere Weglänge ist $O(1)$ und die Verkehrsdichte ist $O(v)$. Ein SBC kann beliebig expandiert werden.

(4) Balance. SBC der Stufe n sind 1-balanciert mit $\lambda_1 = 1\ (= r_x)$, 2-balanciert mit $\lambda_2 = 1$ und darüber hinaus t-balanciert für $t \leq n$ mit $\lambda_t = 1$.

(5) Klassifizierung. Ein SBC $(V,\mathscr{B})$ der Stufe $n\ (=: t)$ ist ein t-Plan und daher insbesondere ein balancierter unvollständiger Blockplan (BIBD).[3]

---

3 Obwohl es sich um einen vollständigen Versuchsplan handelt, da der einzige Block alle Elemente von V umfaßt, werden die Anforderungen an einen BIBD nach der Definition in Anhang A dennoch erfüllt, da Unvollständigkeit nicht zwingend vorausgesetzt wurde.

3.1.2 Vollständig verbundene Rechner

A. Weitere Bezeichnungen.
Completely Interconnected Computer, CIC, vollständiger Graph K_n (Graphentheorie)

B. Implementierungen.
(Viele Multiprozessorprojekte mit nur einigen wenigen Prozessoren waren vollständig verbundene Systeme.)

C. Kombinatorische Spezifikation.

(1) Beispiel.
Der in Abb. 8 dargestellte vollständig verbundene Rechner der Stufe $n = 6$ entspricht folgendem Versuchsplan:
$V = \{0,1,2,3,4,5\}$
$\mathscr{B} = \{(0,1),(1,2),(2,3),(3,4),(4,5),(5,0),(0,2),(0,3),(0,4),(1,3),(1,4),(1,5),(2,4),(2,5),(3,5)\}$
$v = 6, b = 15, r_x = 5\ \forall x \in V$
$\Lambda_2 = \{\lambda(S): |S| = 2, S \subseteq V\} = \{1\}$

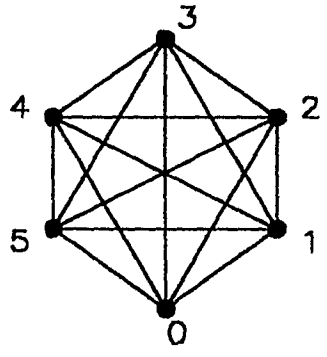


Abb. 8. Vollständig verbundener Rechner der Stufe 6

(2) Versuchsplanparameter. Ein CIC $(V,\mathscr{B})$ der Stufe n besitzt $v = n$ Knoten und $b = n(n-1)/2$ Blöcke der Größe $k = 2\ \forall B \in \mathscr{B}$. Der Wiederholungsfaktor beträgt $r_x = n - 1\ \forall x \in V$.

(3) Kenngrößen. Die Verknüpfungsdichte des CIC beträgt $d = (v-1)/2$, die Redundanz ist $\rho = v - 2$, der Durchmesser ist $\Delta_{max} = 1$ und die Bandbreite ist $\omega = v(v-1)$. Die Verbindungskosten betragen $O(v)$, die mittlere Weglänge ist $O(1)$ und die Verkehrsdichte ist $O(v)$. Ein CIC kann beliebig expandiert werden.

(4) Balance. CICs der Stufe n sind 1-balanciert mit $\lambda_1 = n - 1\ (= r_x)$ und 2-balanciert mit $\lambda_2 = 1$.

(5) Klassifizierung. Ein CIC $(V,\mathscr{B})$ der Stufe n ist ein balancierter unvollständiger Blockplan (BIBD).

3.2 Partiell balancierte unvollständige Blockpläne (PBIBD)

3.2.1 Linearverdichtungen

3.2.1.1 Sternförmige Rechner

A. Weitere Bezeichnungen.
Star-Connected Computer, STC

B. Implementierungen.
(Kern der Verbindungsstruktur im) Carnegie-Mellon Cm* ([Gil,81])

C. Kombinatorische Spezifikation.

(1) Beispiel.
Der in Abb. 9 dargestellte Star-Connected Computer der Stufe (Außenknoten) $n = 8$ entspricht folgendem Versuchsplan:
$V = \{0,1,2,3,4,5,6,7,8\}$
$\mathscr{B} = \{(0,1),(0,2),(0,3),(0,4),(0,5),(0,6),(0,7),(0,8)\}$
$v = 9, b = 8, r_0 = 8, r_x = 1\ \forall x \in V \setminus \{0\}$
$\Lambda_2 = \{\lambda(S): |S| = 2, S \subseteq V\} = \{0,1\}$

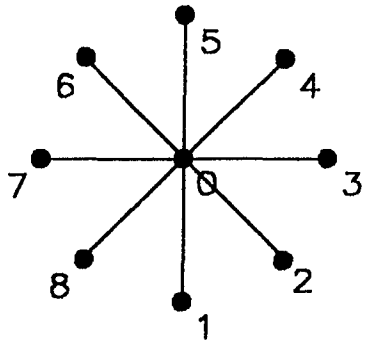


Abb. 9. Sternförmiger Rechner der Stufe (Außenknoten) 8

(2) Versuchsplanparameter. Ein STC $(V,\mathscr{B})$ der Stufe n besitzt $v = n + 1$ Knoten und $b = n$ Blöcke der Größe $k = 2\ \forall B \in \mathscr{B}$. Der Wiederholungsfaktor beträgt $r_0 = n$ für das Zentrum und $r_x = 1$ für die peripheren Knoten.

(3) Kenngrößen. Die Verknüpfungsdichte des STC beträgt $d = n/(n+1)$, die Redundanz ist $\rho = 0$, der Durchmesser ist $\Delta_{max} = 2$ und die Bandbreite ist $\omega = v$. Die Verbindungskosten betragen $O(1)$, die mittlere Weglänge ist $O(1)$ und die Verkehrsdichte ist $O(v)$. Ein STC kann unter Berücksichtigung der Kapazität des Zentralknotens beliebig expandiert werden.

(4) Balance. Ein STC der Stufe n ist nicht 1-balanciert (unterschiedliche Wiederholungsfaktoren) und daher auch nicht 2-balanciert.

(5) Klassifizierung. Ein STC $(V,\mathscr{B})$ mit n Ästen ist eine $2-(n+1,2,1)$-Verdichtung und somit auch ein partiell balancierter unvollständiger Blockplan (PBIBD).

3.2.1.2 Ringverknüpfte Rechner

A. Weitere Bezeichnungen.
Ring-Connected Computer, RCC

B. Implementierungen.
(Substruktur des) Kendall Square Research KSR-1 ([Web,93]), (Substruktur der) CDC CYBERPLUS, PS 2000

C. Kombinatorische Spezifikation.

(1) Beispiel.
Der in Abb. 10 dargestellte Ring-Connected Computer der Stufe $n=8$ entspricht folgendem Versuchsplan:
$V=\{0,1,2,3,4,5,6,7\}$
$\mathscr{B}=\{(0,1),(1,2),(2,3),(3,4),(4,5),(5,6),(6,7),(7,0)\}$
$v=8, b=8, r_x=2\ \forall x\in V$
$\Lambda_2=\{\lambda(S):|S|=2, S\subseteq V\}=\{0,1\}$

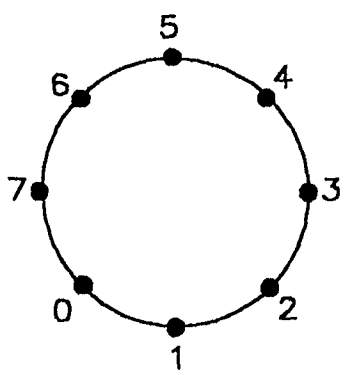


Abb. 10. Ringförmiger Rechner der Stufe 8

(2) Versuchsplanparameter. Ein RCC $(V,\mathscr{B})$ der Stufe n besitzt $v=n$ Knoten und $b=n$ Blöcke der Größe $k=2\ \forall B\in\mathscr{B}$. Der Wiederholungsfaktor beträgt $r_x=2\ \forall x\in V$.

(3) Kenngrößen. Die Verknüpfungsdichte des RCC beträgt $d=1$, die Redundanz ist $\rho=1$, der Durchmesser ist $\Delta_{max}=v/2$ und die Bandbreite ist $\omega=2v$. Die Verbindungskosten betragen $O(1)$, die mittlere Weglänge ist $O(v)$ und die Verkehrsdichte ist $O(v^2)$. Ein RCC kann beliebig expandiert werden.

(4) Balance. Ein RCC der Stufe n ist 1-balanciert mit $\lambda_1 = 2\ (= r_x)$. Die 2-Indexmenge ist $\Lambda_2 = \{0,1\}$.

(5) Klassifizierung. Ein RCC $(V,\mathscr{B})$ der Stufe n ist eine $2-(n,2,1)$-Verdichtung und somit auch ein partiell balancierter unvollständiger Blockplan (PBIBD).

3.2.1.3 Verstrebte Ringe

A. Weitere Bezeichnungen.
Chordal-Ring-Connected Computer, CRC

B. Implementierungen.
ELI-512 ([P&&,84])

C. Kombinatorische Spezifikation.
Hier wird eine spezielle Form mit Diagonalverknüpfungen betrachtet; es existieren jedoch auch andere Formen mit einer wesentlichen größeren Zahl von, gegenüber dem Ring zusätzlichen, Verbindungen [Dot,84].

(1) Beispiel.
Der in Abb. 11 dargestellte Chordal-Ring-Connected Computer der Stufe $n = 8$ entspricht folgendem Versuchsplan:
$V = \{0,1,2,3,4,5,6,7\}$
$\mathscr{B} = \{(0,1),(1,2),(2,3),(3,4),(4,5),(5,6),(6,7),(7,0),(0,4),(1,5),(2,6),(3,7)\}$
$v = 8, b = 12, r_x = 3\ \forall x \in V$
$\Lambda_2 = \{\lambda(S): |S| = 2, S \subseteq V\} = \{0,1\}$

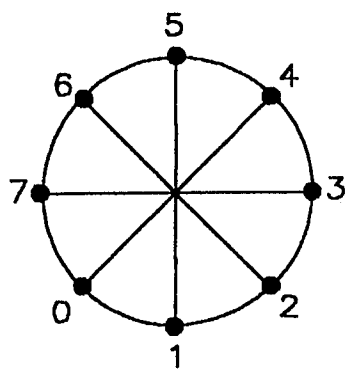


Abb. 11. Rechner mit verstrebtem Ring der Stufe 8

(2) Versuchsplanparameter. Ein CRC $(V,\mathscr{B})$ der Stufe n besitzt $v = n$ Knoten und $b = 3n/2$ Blöcke der Größe $k = 2\ \forall B \in \mathscr{B}$. Der Wiederholungsfaktor beträgt $r_x = 3\ \forall x \in V$.

(3) Kenngrößen. Die Verknüpfungsdichte des CRC beträgt $d = 3/2$, die Redundanz ist $\rho = 2$, der Durchmesser ist $\Delta_{max} = v/4$ und die Bandbreite ist $\omega = 3v$. Die Verbindungs-

kosten betragen $O(1)$, die mittlere Weglänge ist $O(v)$ und die Verkehrsdichte ist $O(v^2)$. Ein CRC kann für gerade Knotenzahlen v beliebig expandiert werden.

(4) Balance. Ein CRC der Stufe n ist 1-balanciert mit $\lambda_1 = 3\ (= r_x)$. Die 2-Indexmenge ist $\Lambda_2 = \{0,1\}$.

(5) Klassifizierung. Ein CRC $(V,\mathscr{B})$ der Stufe n ist eine $2-(n,2,1)$-Verdichtung und somit auch ein partiell balancierter unvollständiger Blockplan (PBIBD).

3.2.2 Polynomialverdichtungen

Im folgenden werden nur solche maschenverbundenen Rechner betrachtet, deren Verbindungsstruktur zyklisch über die Ränder geschlossen ist und die daher auch die Bezeichnung *Torus* tragen. Ein zyklischer Ring von Verbindungen in einer Torusstruktur wird im folgenden als **Masche** bezeichnet. Des weiteren werden nur Strukturen betrachtet, deren Ausdehnung in alle Dimensionen (zwei oder drei) gleich ist. Bei den maschenverbundenen Strukuren, die auf Bussen aufbauen (MBC2, MBC3), wird eine (komplette) Masche durch einen (einzelnen) Bus realisiert.

3.2.2.1 Zweidimensional maschenverbundene Rechner

A. Weitere Bezeichnungen.
Mesh-Connected Computer, MCC2, zweidimensionales Gitter

B. Implementierungen.
ICL DAP1, ICL DAP2, Active Memory Technology DAP 510, Active Memory Technology DAP 610, MasPar MP-1, MasPar MP-2, Loral MPP, Intel Paragon XP/S ([E&K,93])

C. Kombinatorische Spezifikation.

(1) Beispiel.
Der in Abb. 12 dargestellte zweidimensional maschenverbundene Rechner der Stufe $n = 3$ entspricht folgendem Versuchsplan:
$V = \{0,1,2,3,4,5,6,7,8\}$
$\mathscr{B} = \{(0,1),(1,2),(2,0),(3,4),(4,5),(5,3),(6,7),(7,8),(8,6),(0,3),(3,6),(6,0),(1,4),(4,7),(7,1),(2,5),$
$(5,8),(8,2)\}$
$v = 9, b = 18, r_x = 4\ \forall x \in V$
$\Lambda_2 = \{\lambda(S): |S| = 2, S \subseteq V\} = \{0,1\}$

(2) Versuchsplanparameter. Ein MCC2 $(V,\mathscr{B})$ der Stufe n besitzt $v = n^2$ Knoten und $b = 2n^2$ Blöcke der Größe $k = 2\ \forall B \in \mathscr{B}$. Der Wiederholungsfaktor beträgt $r_x = 4\ \forall x \in V$.

(3) Kenngrößen. Die Verknüpfungsdichte des MCC2 beträgt $d = 2$, die Redundanz ist $\rho = 3$, der Durchmesser ist $\Delta_{max} = \sqrt{v}$ und die Bandbreite ist $\omega = 4v$. Die Verbindungs-

kosten betragen $O(1)$, die mittlere Weglänge ist $O(\sqrt{v})$ und die Verkehrsdichte ist $O(v\sqrt{v})$. Ein MCC2 kann im Rahmen von Quadratzahlen expandiert werden.

(4) Balance. Ein MCC2 der Stufe n ist 1-balanciert mit $\lambda_1 = 4 \cdot (= r_x)$. Die 2-Index-Menge ist $\Lambda_2 = \{0,1\}$

(5) Klassifizierung. Ein MCC2 $(V,\mathscr{B})$ der Stufe n ist eine $2-(n^2,2,1)$-Verdichtung und damit insbesondere ein partiell balancierter unvollständiger Blockplan (PBIBD). Für $n = 2$ ist das Mengensystem nicht einfach und der MCC2 eine $2-(n^2,2,2)$-Verdichtung.

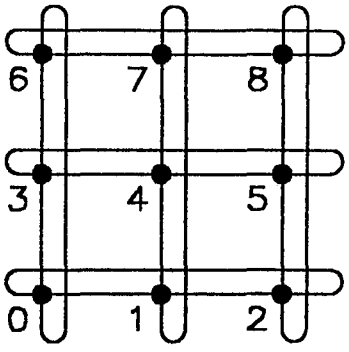


Abb. 12. Zweidimensional maschenverbundener Rechner der Stufe 3

3.2.2.2 Dreidimensional maschenverbundene Rechner

A. Weitere Bezeichnungen.
Mesh-Connected Computer, MCC3, dreidimensionales Gitter

B. Implementierungen.
CRAY MPP ([Oed,93])

C. Kombinatorische Spezifikation.

(1) Beispiel.
Der in Abb. 13 dargestellte dreidimensional maschenverbundene Rechner der Stufe $n = 3$ entspricht folgendem Versuchsplan:
$V = \{0,1,2,3,\ldots,26\}$
$\mathscr{B} = \{(0,1),\ldots,(25,26)\}$
$v = 27, b = 81, r_x = 6\ \forall x \in V$
$\Lambda_2 = \{\lambda(S): |S| = 2, S \subseteq V\} = \{0,1\}$

(2) Versuchsplanparameter. Ein MCC3 $(V,\mathscr{B})$ der Stufe n besitzt $v = n^3$ Knoten und $b = 3n^3$ Blöcke der Größe $k = 2\ \forall B \in \mathscr{B}$. Der Wiederholungsfaktor beträgt $r_x = 6\ \forall x \in V$.

(3) Kenngrößen. Die Verknüpfungsdichte des MCC3 beträgt $d = 3$, die Redundanz ist $\rho = 5$, der Durchmesser ist $\Delta_{max} = 3/2\sqrt[3]{v}$ und die Bandbreite ist $\omega = 6v$. Die Verbindungskosten betragen $O(1)$, die mittlere Weglänge ist $O(\sqrt[3]{v})$ und die Verkehrsdichte ist $O(v\sqrt[3]{v})$. Ein MCC3 kann im Rahmen von Kubikzahlen expandiert werden.

(4) Balance. Ein MCC3 der Stufe n ist 1-balanciert mit $\lambda_1 = 6\ (= r_x)$. Die 2-Index-Menge ist $\Lambda_2 = \{0,1\}$ bzw. $\Lambda_2 = \{0,2\}$ für $n = 2$.

(5) Klassifizierung. Ein MCC3 $(V,\mathscr{B})$ der Stufe n ist eine $2-(n^3, 2,1)$-Verdichtung und somit insbesondere ein partiell balancierter unvollständiger Blockplan (PBIBD). Für $n = 2$ ist das Mengensystem nicht einfach und der MCC3 eine $2-(n^3, 2,2)$-Verdichtung.

Für den Fall $n = 2$ besitzen MCC2 bzw. MCC3 gegenüber den Bus-Systemen MBC2 bzw. MBC3 doppelte Kapazität.

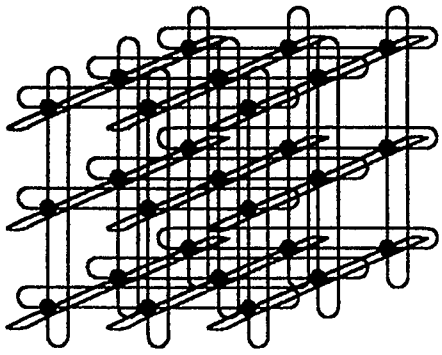


Abb. 13. Dreidimensional maschenverbundener Rechner der Stufe 3: Knotennumerierungen entfallen aus Gründen der Übersichtlichkeit; das Adressierungsschema des MCC2 ist entsprechend zu erweitern.

3.2.2.3 Zweidimensional maschenverbundene Rechner mit Bussen

A. Weitere Bezeichnungen.
Mesh-Bus-Connected Computer, MBC2, zweidimensionales Gitter mit Bussen

B. Implementierungen.
(Maschenstrukturen auf der Basis von Bussen werden von Fiduccia vorgeschlagen [Fid,92].)

C. Kombinatorische Spezifikation.

(1) Beispiel.
Der in Abb. 14 dargestellte zweidimensional maschenbusverbundene Rechner der Stufe $n = 3$ entspricht folgendem Versuchsplan:
$V = \{0,1,2,3,4,5,6,7,8\}$
$\mathscr{B} = \{(0,1,2),(3,4,5),(6,7,8),(0,3,6),(1,4,7),(2,5,8)\}$

$v = 9, b = 6, r_x = 2\ \forall x \in V$
$\Lambda_2 = \{\lambda(S): |S| = 2, S \subseteq V\} = \{0,1\}$

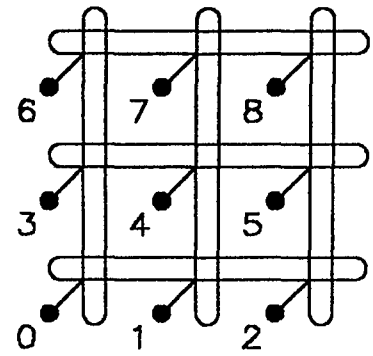


Abb. 14. Zweidimensional maschenbusverbundener Rechner der Stufe 3

(2) Versuchsplanparameter. Ein MBC2 $(V,\mathscr{B})$ der Stufe n besitzt $v = n^2$ Knoten und $b = 2n$ Blöcke der Größe $k = n\ \forall B \in \mathscr{B}$. Der Wiederholungsfaktor beträgt $r_x = 2\ \forall x \in V$.

(3) Kenngrößen. Die Verknüpfungsdichte des MBC2 beträgt $d = 2/\sqrt{v}$, die Redundanz ist $\rho = 1$, der Durchmesser ist $\Delta_{max} = 2$ und die Bandbreite ist $\omega = 2v$. Die Verbindungskosten betragen $O(1/\sqrt{v})$, die mittlere Weglänge ist $O(1)$ und die Verkehrsdichte ist $O(v)$. Ein MBC2 kann im Rahmen von Quadratzahlen expandiert werden.

(4) Balance. Ein MBC2 der Stufe n ist 1-balanciert mit $\lambda_1 = 2\ (= r_x)$. Die 2-Index-Menge ist $\Lambda_2 = \{0,1\}$

(5) Klassifizierung. Ein MBC2 $(V,\mathscr{B})$ der Stufe n ist eine $2-(n^2,n,1)$-Verdichtung und damit insbesondere ein partiell balancierter unvollständiger Blockplan (PBIBD).

3.2.2.4 Dreidimensional maschenverbundene Rechner mit Bussen

A. Weitere Bezeichnungen.
Mesh-Bus-Connected Computer, MBC3, dreidimensionales Gitter mit Bussen

B. Implementierungen.
(Maschenstrukturen auf der Basis von Bussen werden von Fiduccia vorgeschlagen [Fid,92].)

C. Kombinatorische Spezifikation.

(1) Beispiel.
Die Abb. 13 auf Seite 38 zeigt die grundlegende Struktur des dreidimensional maschenbusverbundenen Rechners der Stufe $n = 3$. Analog zur Abbildung des zweidimensional maschenbusverbundenen Rechners, Abb. 14 auf Seite 39, wird ein

zyklischer Umlauf (Masche) jedoch nur durch einzelnen Bus realisiert. Der MBC3 der Stufe $n = 3$ entspricht daher folgendem Versuchsplan:
$V = \{0,1,2,3,...,26\}$
$\mathscr{B} = \{(0,1,2),...,(24,25,26)\}$
$v = 27, b = 27, r_x = 3\ \forall x \in V$
$\Lambda_2 = \{\lambda(S): |S| = 2, S \subseteq V\} = \{0,1\}$

(2) Versuchsplanparameter. Ein MBC3 $(V,\mathscr{B})$ der Stufe n besitzt $v = n^3$ Knoten und $b = 3n^2$ Blöcke der Größe $k = n\ \forall B \in \mathscr{B}$. Der Wiederholungsfaktor beträgt $r_x = 3\ \forall x \in V$.

(3) Kenngrößen. Die Verknüpfungsdichte des MBC3 beträgt $d = 3/\sqrt[3]{v}$, die Redundanz ist $\rho = 2$, der Durchmesser ist $\Delta_{max} = 3$ und die Bandbreite ist $\omega = 3v$. Die Verbindungskosten betragen $O(1/\sqrt[3]{v})$, die mittlere Weglänge ist $O(1)$ und die Verkehrsdichte ist $O(v)$. Ein MBC3 kann im Rahmen von Kubikzahlen expandiert werden.

(4) Balance. Ein MBC3 der Stufe n ist 1-balanciert mit $\lambda_1 = 3\ (= r_x)$. Die 2-Index-Menge ist $\Lambda_2 = \{0,1\}$

(5) Klassifizierung. Ein MBC3 $(V,\mathscr{B})$ der Stufe n ist eine $2-(n^3, n, 1)$-Verdichtung und somit insbesondere ein partiell balancierter unvollständiger Blockplan (PBIBD).

3.2.3 Exponentialverdichtungen

3.2.3.1 Hyperwürfel

A. Weitere Bezeichnungen.
Cube-Connected Computer, CCC, Hypercube, binary N-Cube

B. Implementierungen.
Floating Point Systems Serie T, Caltech Cosmic Cube, Caltech/JPL Mark II, Caltech/JPL Mark IIIfp, Intel iPSC1, Intel iPSC2, Intel iPSC/860, NCUBE/four, NCUBE/seven, NCUBE/ten, CCI Navier Stokes Computer, Columbia University CHoPP, Paralex Gemini, University of New Mexiko UNMHC, AT&T BTL-Hypercube, Ametek System 14, PC-CUBE, MAC-CUBE, (Metastruktur der) Thinking Machines CM-2 ([Röd,91])

C. Kombinatorische Spezifikation.
Der Fall $v = 2, b = 1$ (1-dimensionaler Fall) wird hier nicht als Hypercube angesehen (vgl. 3.1.2: vollständig verbundene Rechner). Die Knotennumerierungen entsprechen dem Gray-Code.

(1) Beispiel.
Der in Abb. 15 dargestellte Hypercube der Stufe (Dimension) $n = 3$ entspricht folgendem Versuchsplan:
$V = \{0,1,2,3,4,5,6,7\}$
$\mathscr{B} = \{(0,1),(1,2),(2,3),(3,0),(0,7),(1,6),(2,5),(3,4),(4,5),(5,6),(6,7),(7,4)\}$

$v = 8, b = 12, r_x = 3\ \forall x \in V$
$\Lambda_2 = \{\lambda(S): |S| = 2, S \subseteq V\} = \{0,1\}$

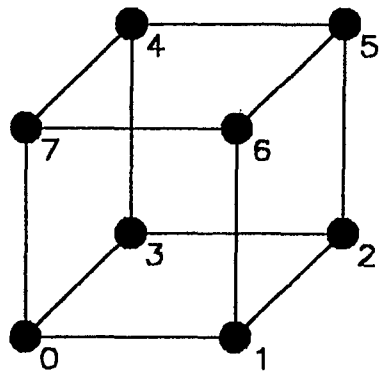


Abb. 15. Hyperwürfel der Stufe (Dimension) 3

Der Hypercube der Stufe (Dimension) $n = 4$ ist in Abb. 34 auf Seite 120 dargestellt.

(2) Versuchsplanparameter. Ein Hypercube $(V,\mathscr{B})$ der Stufe n besitzt $v = 2^n$ Knoten und $b = n2^{n-1}$ Blöcke der Größe $k = 2\ \forall B \in \mathscr{B}$. Der Wiederholungsfaktor beträgt $r_x = n\ \forall x \in V$.

(3) Kenngrößen. Die Verknüpfungsdichte des CCC beträgt $d = (\log_2 v)/2$, die Redundanz ist $\rho = \log_2 v - 1$, der Durchmesser ist $\Delta_{max} = \log_2 v$ und die Bandbreite ist $\omega = v \log_2 v$. Die Verbindungskosten betragen $O(\log v)$, die mittlere Weglänge ist $O(\log v)$ und die Verkehrsdichte ist $O(v \log v)$. Ein CCC kann im Rahmen von Zweierpotenzen expandiert werden.

(4) Balance. Hypercubes sind 1-balanciert mit $\lambda_1 = n\ (= r_x)$ Die 2-Index-Menge ist $\Lambda_2 = \{0,1\}$

(5) Klassifizierung. Ein Hypercube $(V,\mathscr{B})$ der Dimension n ist ein partiell balancierter unvollständiger Blockplan (PBIBD). Wegen $\lambda_2 \leq 1$ kann der Hypercube der Stufe n auch als $2 - (2^n,2,1)$-Verdichtung bezeichnet werden.

3.2.3.2 Rechner mit würfelartig verknüpften Zyklen

A. Weitere Bezeichnungen.
Cube-Connected Cycle Computer, COC

B. Implementierungen.
(vorgeschlagen von Preparata und Vuillemin [P&V,81]), Boolean Vector Machine (BVM) ([P&&,84])

C. Kombinatorische Spezifikation.
Hier wird nur die spezielle Form betrachtet, in der die Zahl der Knoten in einem Zykel immer genau der Dimension des zugrundeliegenden Würfels entspricht.

(1) Beispiel.
Der in Abb. 16 dargestellte Cube-Connected Cycle Computer der Stufe $n = 3$ entspricht folgendem Versuchsplan:
$V = \{0,...,23\}$
$\mathscr{B} = \{(0,1),(1,2),(2,0),...,(21,22),(22,23),(23,21),(0,3),...,(19,21)\}$
$v = 24, b = 36, r_x = 3\ \forall x \in V$
$\Lambda_2 = \{\lambda(S): |S| = 2, S \subseteq V\} = \{0,1\}$

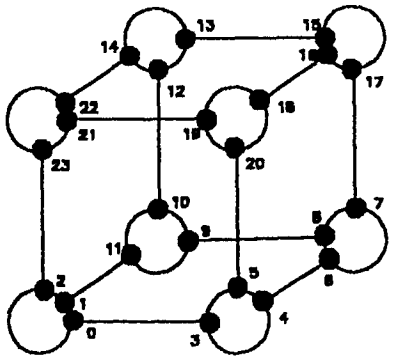


Abb. 16. Hyperwürfel mit Zyklen der Stufe 3

(2) Versuchsplanparameter. Ein COC $(V,\mathscr{B})$ der Stufe n besitzt $v = n2^n$ Knoten und $b = 3n2^{n-1}$ Blöcke der Größe $k = 2\ \forall x \in V$ Der Wiederholungsfaktor beträgt $r_x = 3\ \forall x \in V$.

(3) Kenngrößen. Die Verknüpfungsdichte des COC beträgt $d = 3/2$, die Redundanz ist $\rho = 2$, der Durchmesser ist $\Delta_{max} = 2n$ und die Bandbreite ist $\omega = 3v$. Die Verbindungskosten betragen $O(1)$, die mittlere Weglänge ist $O(n)$ und die Verkehrsdichte ist $O(nv)$. Ein COC kann in bestimmtem Rahmen (vgl. Knotenzahl) expandiert werden.

(4) Balance. Ein COC der Stufe n ist 1-balanciert mit $\lambda_1 = 3\ (= r_x)$. Die 2-Index-Menge ist $\Lambda_2 = \{0,1\}$.

(5) Klassifizierung. Ein COC $(V,\mathscr{B})$ der Stufe n ist eine $2-(n2^n,2,1)$-Verdichtung und somit auch ein partiell balancierter unvollständiger Blockplan (PBIBD).

3.2.3.3 Binärbäume

A. Weitere Bezeichnungen.
Tree-Connected Computer, TCC

B. Implementierungen.
ip-Systems TX3, NON-VON, DADO, X-Tree

C. Kombinatorische Spezifikation.

(1) Beispiel.
Der in Abb. 17 dargestellte Tree-Connected Computer der Stufe (Baumhöhe) $n = 3$ entspricht folgendem Versuchsplan:[4]
$V = \{0,1,2,3,4,5,6,7,8,9,10,11,12,13,14\}$
$\mathcal{B} = \{(0,1),(0,2),(1,3),(1,4),(2,5),(2,6),(3,7),(3,8),(4,9),(4,10),(5,11),(5,12),(6,13),(6,14)\}$
$v = 15, b = 14, r_0 = 2, r_x = 1\ \forall x \in \{1,2,3,4,5,6\}, r_y = 3\ \forall y \in \{7,8,9,10,11,12,13,14\}$
$\Lambda_2 = \{\lambda(S): |S| = 2, S \subseteq V\} = \{0,1\}$

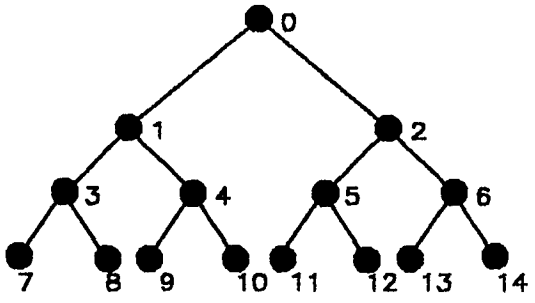


Abb. 17. Binärbaum-Rechner der Stufe (Baumhöhe) 3

(2) Versuchsplanparameter. Ein TCC $(V,\mathcal{B})$ der Stufe n besitzt $v = \sum_{i=0}^{n} 2^i = 2^{n+1} - 1$ Knoten und $b = \sum_{i=0}^{n} 2^i - 1 = 2^{n+1} - 2$ Blöcke der Größe $k = 2\ \forall B \in \mathcal{B}$. Die Wiederholungsfaktoren betragen $r_0 = 2$ für die Wurzel, $r_x = 1$ für die Blätter und $r_y = 3$ für alle übrige Knoten.

(3) Kenngrößen. Die Verknüpfungsdichte des TCC beträgt $d = 1$, die Redundanz ist $\rho = 0$, der Durchmesser ist $\Delta_{max} = 2 \log_2 v$ und die Bandbreite ist $\omega = 2v$. Die Verbindungskosten betragen $O(1)$, die mittlere Weglänge ist $O(\log v)$ und die Verkehrsdichte ist $O(v \log v)$. Ein TCC kann im Rahmen von Zweierpotenzen expandiert werden.

(4) Balance. Ein TCC der Stufe n ist aufgrund unterschiedlicher Wiederholungsfaktoren nicht 1-balanciert und daher auch nicht 2-balanciert.

(5) Klassifizierung. Ein TCC $(V,\mathcal{B})$ der Stufe n ist eine $2-(2^{n+1}-1,2,1)$-Verdichtung und somit auch ein partiell balancierter unvollständiger Blockplan (PBIBD).

---

4 Die Numerierung der Binärbaum- sowie der Hyperbaumrechner könnte ebenso wie die der maschenverbundenen Rechner im Hinblick auf andere Operationen wesentlich systematischer oder effizienter erfolgen. Dies ist hier jedoch nicht erforderlich, so daß Konsistenz mit den anderen Topologiebeispielen im Vordergrund steht.

3.2.3.4 Hyperbäume

A. Weitere Bezeichnungen.
Hypertree-Connected Computer, Hypertree, HCC

B. Implementierungen.
(projektiert durch Goodman und Sequin [G&S,81])

C. Kombinatorische Spezifikation.

(1) Beispiel.
Der in Abb. 18 dargestellte Hypertree Computer der Stufe (Baumhöhe) $n = 3$ entspricht folgendem Versuchsplan:
$V = \{0,1,2,3,4,5,6,7,8,9,10,11,12,13,14\}$
$\mathscr{B} = \{(0,1),(0,2),(1,3),(1,4),(2,5),(2,6),(3,7),(3,8),(4,9),(4,10),(5,11),(5,12),(6,13),(6,14),(1,2),$
$(3,5),(4,6),(7,11),(8,12),(9,13),(10,14)\}$
$v = 15, b = 21, r_0 = 2, r_x = 2\ \forall x \in \{7,8,...,14\}, r_y = 4\ \forall y \in \{1,2,3,4,5,6\}$
$\Lambda_2 = \{\lambda(S): |S| = 2, S \subseteq V\} = \{0,1\}$

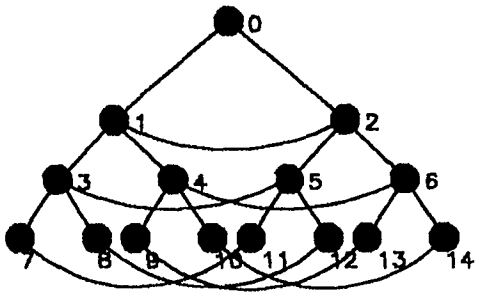


Abb. 18. Hyperbaum-Rechner der Stufe (Baumhöhe) 3

(2) Versuchsplanparameter. Ein HCC $(V,\mathscr{B})$ der Stufe n besitzt $v = \sum_{i=0}^{n} 2^i = 2^{n+1} - 1$ Knoten und $b = b_1 + b_2 = \sum_{i=0}^{n} 2^i - 1 + \sum_{i=0}^{n-1} 2^i = 3(2^n - 1)$ Blöcke der Größe $k = 2\ \forall B \in \mathscr{B}$. Die Wiederholungsfaktoren betragen $r_0 = 2$ für die Wurzel sowie $r_x = 2$ für alle Blätter und $r_y = 4$ für alle übrige Knoten.

(3) Kenngrößen. Die Verknüpfungsdichte des HCC beträgt $d \approx 3/2$, die Redundanz ist $\rho = 1$, der Durchmesser ist $\Delta_{max} = 2 \log_2 v - 1$ und die Bandbreite ist $\omega = 3v$. Die Verbindungskosten betragen $O(1)$, die mittlere Weglänge ist $O(\log v)$ und die Verkehrsdichte ist $O(v \log v)$. Ein HCC kann im Rahmen von Zweierpotenzen expandiert werden.

(4) Balance. Ein HCC der Stufe n ist aufgrund unterschiedlicher Wiederholungsfaktoren nicht 1-balanciert und daher auch nicht 2-balanciert.

(5) Klassifizierung. Ein HCC $(V,\mathscr{B})$ der Stufe n ist eine $2 - (2^{n+1} - 1,2,1)$-Verdichtung und somit auch ein partiell balancierter unvollständiger Blockplan (PBIBD).

3.2.3.5 Pyramidenförmig verknüpfte Rechner

A. Weitere Bezeichnungen.
Pyramidal-Connected Computer, PCC

B. Implementierungen.
Erlangen General Purpose Array (EGPA) (in modifizierter Form [M&S,76])

C. Kombinatorische Spezifikation.

(1) Beispiel.
Der in Abb. 19 dargestellte Pyramidal-Connected Computer der Stufe $n = 2$ (Ebenenzahl − 1) entspricht folgendem Versuchsplan:
$V = \{0,1,...,20\}$
$\mathcal{B} = \{(0,1),(0,2),(0,3),(0,4),(1,2),(2,3),(3,4),(4,1),(1,5),(1,6),(1,7),(1,8),(2,9),(2,10),(2,11),(2,12),$
$(3,13),(3,14),(3,15),(3,16),(4,17),(4,18),(4,19),(4,20),(5,6),(6,7),(7,8),(8,5),(9,10),(10,11),$
$(11,12),(12,9),(13,14),(14,15),(15,16),(16,13),(17,18),(18,19),(19,20),(20,17)\}$
$v = 21, b = 40, r_0 = 4, r_x = 7\ \forall x \in \{1,2,3,4\}, r_y = 3\ \forall y \in V \setminus \{0,1,2,3,4\}$
$\Lambda_2 = \{\lambda(S): |S| = 2, S \subseteq V\} = \{0,1\}$

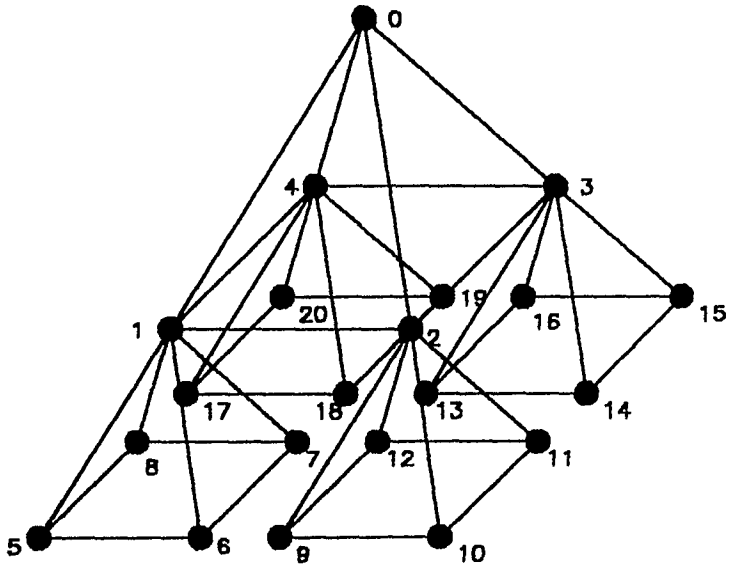


Abb. 19. Pyramidenförmiger Rechner der Stufe (Ebenenzahl − 1) 2

(2) Versuchsplanparameter. Ein PCC $(V,\mathcal{B})$ der Stufe n besitzt $v = \sum_{i=0}^{n} 4^i = (4^{n+1}-1)/3$ Knoten und $b = 2\sum_{i=1}^{n} 4^i = 2(4^{n+1}-1)/3 - 2$ Blöcke der Größe $k = 2\ \forall B \in \mathcal{B}$. Die Wiederholungsfaktoren betragen $r_0 = 4$ für den Wurzelknoten, $r_x = 3$ für die Knoten der untersten Ebene und $r_y = 7$ für alle übrigen Knoten.

*(3) **Kenngrößen.*** Die Verknüpfungsdichte des PCC beträgt $d \approx 2$, die Redundanz ist $\rho = 2$, der Durchmesser ist $\Delta_{max} = \log_4 v$ und die Bandbreite ist $\omega \approx 5v$. Die Verbindungskosten betragen $O(1)$, die mittlere Weglänge ist $O(\log v)$ und die Verkehrsdichte ist $O(v \log v)$. Ein PCC kann im Rahmen von Viererpotenzen (vgl. o.) expandiert werden.

*(4) **Balance.*** Ein PCC ist aufgrund unterschiedliche Wiederholungsfaktoren nicht 1-balanciert und daher auch nicht 2-balanciert.

*(5) **Klassifizierung.*** Ein PCC $(V,\mathscr{B})$ der Stufe n ist ein partiell balancierter unvollständiger Blockplan (PBIBD) und hier speziell eine $((4^{n+1}-1)/3,2,1)$-Verdichtung.

3.3 Partielle paarweise balancierte Pläne (PPBD)

3.3.1 Hierarchische Bussysteme

A. Weitere Bezeichnungen.
Hierarchical Bus Computer, HBC

B. Implementierungen.
SUPRENUM ([Gil,88])

C. Kombinatorische Spezifikation.
Hier werden nur Hierarchische Bussyteme mit einem Systembus betrachtet.

(1) Beispiel.
Das in Abb. 20 dargestellte hierarchische Bussystem mit $b_C = 3$ Clusterbussen zu je $n = 4$ Knoten entspricht folgendem Versuchsplan:
$V = \{0,1,2,..., 11, A,B,C\}$
$\mathscr{B} = \{(0,1,2,3,A),(4,5,6,7,B),(8,9,10,11,C),(A,B,C)\}$
$v = 15, b = b_C + 1 = 4, r_x = 1\ \forall x \in \{0,1,..,11\}, r_y = 2\ \forall y \in \{A,B,C\}$
$\Lambda_2 = \{\lambda(S): |S| = 2, S \subseteq V\} = \{0,1\}$

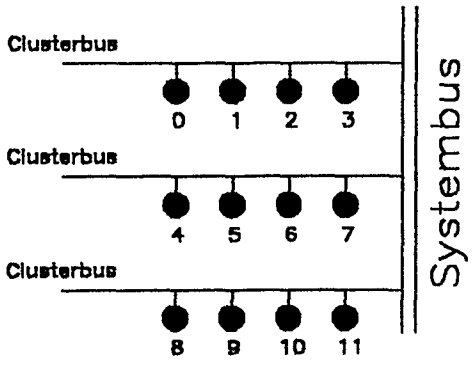


Abb. 20. Hierarchisches Bussystem mit 3 Clusterbussen und 1 Systembus

(2) Versuchsplanparameter. Ein HBC $(V,\mathscr{B})$ mit b_C Clusterbussen zu je n Knoten und 1 Systembus besitzt $v = b_C n + b_C = (n+1)b_C$ Knoten inclusive Hilfsknoten, $b = b_C$ Blöcke der Größe $k_C = n + 1$ und 1 Block der Größe $k_S = b_C$. Die Wiederholungsfaktoren betragen $r_K = 1$ für die (Prozessor-)Knoten, $r_H = 2$ für die Hilfsknoten.

(3) Kenngrößen. Die Verknüpfungsdichte des HBC beträgt $d = (b_C + 1)/nb_C$, die Redundanz ist $\rho = 0$, der Durchmesser ist $\Delta_{max} = 3$ und die Bandbreite ist $\omega = b_C$. Die Verbindungskosten betragen $O(1/n)$, die mittlere Weglänge ist 2 und die Verkehrsdichte ist $O(v)$. Ein HBC kann proportional zur Zahl der Clusterbusse expandiert werden.

*(4) **Balance.*** Ein HBC ist aufgrund unterschiedliche Wiederholungsfaktoren nicht 1-balanciert und daher auch nicht 2-balanciert.

*(5) **Klassifizierung.*** Ein HBC $(V,\mathscr{B})$ mit b_C Clusterbussen und 1 Systembus wird durch einen partiellen paarweise balancierten Plan (PPBD) mit den Parametern $(v,K,\Lambda_2) = ((n+1)b_C, \{k_C, k_S\}, \{0,1\})$ beschrieben und kann daher auch als nicht-uniforme Verdichtung bezeichnet werden.

3.3.2 Allgemeine würfelverbundene Zyklen

Realisiert man im Cube-Connected Cycle Computer (vgl. 3.2.3.2) die Zykel durch jeweils einen Bus, so wird der COC $(V,\mathscr{B})$ durch einen partiellen paarweise balancierten Plan (PPBD) mit den Parametern $(v,K,\Lambda_2) = (n2^n, \{n,2\}, \{0,1\})$ beschrieben und kann daher auch als nicht-uniforme Verdichtung bezeichnet werden.

3.4.1 Rekonfigurierbare Netzwerke

Mit „rekonfigurierbar" werden an dieser Stelle Rechner bezeichnet, die verschiedene Topologien im Sinne der bisher behandelten statischen Verbindungsnetzwerke annehmen können, nicht aber Topologien mit Schaltelementen, die auch während einer Kommunikationsaufgabe ihre Konfiguration verändern können, also von ihrer Funktionsweise dynamische Verbindungsnetzwerke darstellen. Auch diese werden in der Literatur mit dem Begriff rekonfigurierbar belegt. Zu den im statischen Sinne rekonfigurierbaren Netzwerken gehören beispielsweise Transputer-Systeme, wie das MEIKO-Konzept, die manuell umkonfiguriert werden können. Derartige Systeme können durch Hyperpläne (vgl. Anhang A) beschrieben werden, also Familien von Versuchsplänen über der gleichen Menge V.

3.4.2 Kombinationen von Topologien

Entsteht das statische Verbindungsnetzwerk eines Rechners durch Kombination verschiedener hier aufgeführter Topologien, so kann auch dies mit optimalen Versuchsplänen beschrieben werden. Beispielsweise sind Ringe sternförmiger Topologien denkbar oder auch Binärbäume, an deren Blättern sich komplette Ringe befinden ähnlich dem Kendall Square KSR-1 [Web,93]. Tsai, Bagherzadeh und Kim schlagen eine Kombination aus Maschen- und Baumverbindungen vor [TBK,91]. Kumar und Patnaik beschreiben einen erweiterten Hypercube, dessen „Knoten" wiederum Hypercube-Rechner darstellen [K&P,92]. Obwohl diese Rechner keiner der bekannten Einzel-Topologieklassen entsprechen, fallen sie dennoch in eine bestimmte Klasse von Versuchsplänen, so daß sich mit Hilfe kombinatorischer Beschreibung ein neuer Zugang zum *Mapping*-Problem (vgl. Kapitel 5) ergibt.

3.4.3 Noch nicht integrierte Architekturen

Mit einfachen Erweiterungen können auch andere Formen statischer Verbindungsnetzwerke beschrieben werden, beispielsweise Architekturen, bei denen sich nicht an allen Endpunkten von Verbindungen Prozessoren befinden, wie etwa beim sogenannten *fat-tree* [G&L,85], der im Verbindungsnetzwerk der Connection Machine CM-5 Verwendung gefunden hat. Der *fat-tree* verfügt zwar über eine Baumstruktur, wie bei den hier besprochenen Tree-Topologien, es sind jedoch nur die Blätter des Baumes mit Prozessoren besetzt.

4 Beschreibung paralleler Algorithmen durch Versuchspläne

4.1 Charakterisierung paralleler Algorithmen

4.1.1 Gründe für eine Klassifizierung von Algorithmen

Über die beim Entwurf sequentieller Algorithmen zu treffenden Entscheidungen hinaus sind bei der Entwicklung paralleler Algorithmen eine Reihe zusätzlicher Aspekte zu berücksichtigen. Neben der Zahl der maximal sinnvoll einzusetzenden Prozessoren spielt die Frage der Datenallokation eine entscheidende Rolle. Weiterhin muß die Festlegung von Prozessen, sofern diese wählbar ist, im Hinblick auf eine balancierte Lastverteilung und somit gute Auslastung des Zielrechners erfolgen. Daher nimmt der Schwierigkeitsgrad sowohl in Entwicklung als auch Implementierung beim Übergang vom sequentiellen zum parallelen Fall aufgrund der zusätzlichen Freiheitsgrade der Entscheidungsprozesse um ein Vielfaches zu [Jam,87].

Um eine hohe Effizienz bei der Ausführung eines Programms auf einem Rechner zu gewährleisten, kann es entscheidend sein, daß die Struktur des Verbindungsnetzwerks des einzusetzenden (verteilten) Rechners die Kommunikationsstruktur des auszuführenden Programms hinreichend gut widerspiegelt. Diese Quasi-Emulation kann naturgemäß nur von einer rekonfigurierbaren Architektur für verschiedene Typen von Algorithmen geleistet werden. Anderenfalls – und das ist die Realität aufgrund der gegenwärtigen Nichtverfügbarkeit derartiger Rechner – muß zur Implementierung eines bestimmten Algorithmus die jeweils geeignetste Rechnertopologie ausgewählt werden. Ist nur ein Rechnersystem verfügbar, so ist die Problemstellung offensichtlich hinfällig.

Es ist nun sicherlich sinnvoll, die für einen bestimmten Algorithmus brauchbarste Architektur einmalig zu ermitteln, damit nicht für jedes Implementierungsvorhaben diese Optimierung erneut geleistet werden muß. Da viele Algorithmen gleiche oder ähnliche Grundstrukturen aufweisen, ergibt sich, daß im allgemeinen nicht für einen einzelnen Algorithmus sondern für eine Klasse von Algorithmen die beste Topologie zur Implementierung gefunden werden muß. Beispielsweise kann das Prinzip des *Divide-and-Conquer* in vielen Algorithmen angetroffen werden, wie bei der Summation von Zahlen oder beim Quicksort-Verfahren [Kne,90]. Hat man einmal nachgewiesen, daß die Rechnertopologie in der Form des binären Baumes diejenige ist, die das Divide-and-Conquer-Schema optimal reflektiert, so wird die Suche nach dem idealen Rechner für einen bestimmten Algorithmus bereits durch Identifikation des Divide-and-Conquer-Prinzips in diesem Algorithmus geleistet. Die Klassenbildung trägt also in erheblichem Maße zur Reduktion des Implementierungsaufwands bei.

Die hier dargestellte Situation der Klasseneinteilung anhand der algorithmischen Grundprinzipien, wie *Divide-and-Conquer*, *greedy* oder *dynamisches Programmieren* [Kri,89], ist eine grobe Vereinfachung der tatsächlichen Situation, da der Raum der Algorithmen nur eindimensional dargestellt wurde. Um eine brauchbare Klasseneinteilung zu erreichen, muß man jedoch eine Vielzahl verschiedener Aspekte berücksichtigen, welche dann einen mehrdimensionalen Raum zur Charakterisierung paralleler Algorithmen aufspannen.

Neben der Problematik der optimalen Abbildung paralleler Algorithmen auf bereits existierende parallele Hardware kann man die ideale Zieltopologie für einen praxisrele-

vanten Algorithmus auch als Entwurfsvorbild für neue Rechnerkonstruktionen nutzen. Dies ist in gewissen Spezialfällen, insbesondere solchen aus der Nachrichtentechnik und der digitalen Bildverarbeitung, mit dem Entwurf von Spezialprozessoren bereits geschehen. Andererseits erhofft man sich brauchbare und zuverlässige Vorhersagen über die mit einem bestimmten Algorithmus auf einer bestimmten Hardware erzielbare Leistung. Die Bereitstellung eines einheitlichen Rahmens für die Beurteilung paralleler Algorithmen dient zum Vergleich mehrerer Algorithmen zur Lösung des gleichen Problems. Das Idealbild künftiger Rechnertopologien wird jedoch zweifelsfrei mit dem Attribut der Rekonfigurierbarkeit beschrieben.

4.1.2 Kriterien zur Charakterisierung paralleler Algorithmen

Kung [Kun,80] klassifiziert Algorithmen direkt im Hinblick auf zu verwendende Rechnerkategorien und teilt parallele Algorithmen in die Klassen SIMD, MIMD und systolisch ein. Diese Vorgehensweise wird hier nicht nachvollzogen, da die generelle Leistungsfähigkeit eines parallelen Algorithmus oft gerade dadurch begrenzt wird, daß er mit Blick auf eine bestimmte Architektur entworfen wurde [P&Y,88]. Die einem bestimmten Problem innewohnende Parallelität, die durch einen Algorithmus formuliert sein kann, ist von wesentlich größerem Interesse als die Parallelität eines (festen) Programms [Val,82b]. Die für den jeweiligen Algorithmus geeignete Topologie soll sich daher aus den algorithmen-spezifischen Eigenschaften herleiten, die für diesen Algorithmus zutreffen. Jamieson [Jam,86] liefert eine Zusammenstellung wesentlicher Attribute zur Charakterisierung paralleler Algorithmen, die in der vorliegenden Arbeit in komprimierter und modifizierter Form aufgegriffen wird:

- Natur des Parallelismus
- Grad des Parallelismus
- Charakter des Algorithmus
- Grundlegende Operationen
- Datenstrukturen und Datenabhängigkeiten
- Kommunikationsgeometrie

Natur des Parallelismus. Falls auf hinreichend großen Datenfeldern gleiche, voneinander unabhängige, Operationen ausgeführt werden, wie beispielsweise bei der Multiplikation eines Vektors mit einem Skalar, so kann das Feld der Daten partitioniert werden. Das bedeutet, daß die jeweiligen Einzeloperationen auf den verschiedenen Datenelementen des Feldes von entsprechend vielen Prozessoren gleichzeitig bearbeitet werden können. Man spricht von **Datenparallelismus**. In diesem speziellen Falle reicht es für die parallele Abarbeitung aus, daß die verschiedenen Prozessoren des Systems zu einer Zeit exakt die gleiche Operation ausführen. Daraus resultiert, daß hier ein SIMD-System (vgl. Abschnitt 2.1) zum Einsatz kommen kann. Falls in einem Algorithmus zur gleichen Zeit verschiedene Operationen ausgeführt werden sollen, und sei es auch auf verschiedenen Elementen ein und desselben Datenfeldes, so müssen die Prozessoren des Rechnersystems in der Lage sein, zu gleicher Zeit voneinander unabhängige bzw. unterschiedliche Operationen ausführen zu können. Diese Anforderung wird durch ein System gewährleistet, welches als MIMD bezeichnet wird (vgl. Abschnitt 2.1). Es ist offensichtlich, daß ein SIMD-System durch ein MIMD-System emuliert werden kann, und somit

Datenparallelismus sehr wohl auch von einem MIMD-System genutzt werden kann. Mit einem solchen MIMD-Rechner können jedoch auch Algorithmen parallelisiert werden, die *Multitasking*, d. h. Aufteilung in parallele Prozesse, nicht aufgrund von Datenparallelismus ermöglichen, wohl aber aus Teilprozessen bestehen, die zumindest teilweise parallel ausgeführt werden können. Diese Art des Parallelismus wird daher als **Funktionsparallelismus** bezeichnet. Die Art des Parallelismus beeinflußt die Speicherung der Daten, die Zuweisung von Prozessen zu Prozessoren und die grundlegende Entscheidung darüber, welches Parallelrechnerparadigma zum Einsatz kommt; also die Abwägung zwischen MIMD und SIMD. Bisweilen können auf den unterschiedlichen Granularitätsniveaus desselben Algorithmus verschiedene Arten von Parallelismus angetroffen werden.

Grad des Parallelismus. Die maximale Aufteilung eines parallel zu bearbeitenden Datenfeldes in möglichst kleine, noch separat zu behandelnde Teile, bestimmt die Größe der Datenbasiseinheit, die bearbeitet wird. Dieses Maß wird als **Datengranularität** bezeichnet. Im allgemeinen kristallisiert sich eine eindeutige, brauchbare Datenbasisgröße nur dann heraus, wenn Datenparallelismus vorliegt. Die Datengranularität steht in Beziehung zur Datenspeicherung, zu den Kommunikationsanforderungen, zu den Prozessorfähigkeiten und den lokalen Speicheranforderungen. Die minimal erforderliche Speicherbandbreite wird im wesentlichen durch die Datengranularität bestimmt. In den meisten Fällen sind feingranulare Algorithmen vom SIMD- und grobgranulare Algorithmen vom MIMD-Typ. Die **Modulgranularität** beschreibt den Anteil der Berechnungen, der unabhängig von anderen Prozessen erfolgen kann. Sie ist daher ein Maß für die Häufigkeit der notwendigen Synchronisation. Falls die Granularitäten innerhalb eines Algorithmus stark divergieren, nimmt die Modulgranularität Einfluß auf die Entscheidung betreffend SIMD- oder MIMD-Modus und somit die Zuordnung von Prozessen zu Prozessoren, die Kommunikationsanforderungen und die Ausgewogenheit der Prozessorauslastung. Die aus feinkörniger Modulgranularität resultierende häufige Synchronisation erfordert ein schnelles Verbindungsnetzwerk. Im allgemeinen sind Algorithmen mit grober Modulgranularität für asynchrone Multiprozessoren gut geeignet [Kun,80]. Der Grad des Parallelismus steht sowohl in Beziehung zur Daten- als auch zur Modulgranularität; die gröbere der beiden Granularitäten bestimmt die obere Schranke der Parallelität. Der Parallelitätsgrad beeinflußt direkt die Wahl der Maschinengröße hinsichtlich der Zahl der Prozessoren und liefert somit den maximal möglichen Speedup. In der Praxis ist massiver Parallelismus häufig mit dem SIMD-Paradigma verbunden. Hier liegt jedoch kein logischer Zwang vor. Andererseits wird die Verwendung lokaler Speicher anstelle eines gemeinsamen Speichers im massiv-parallelen Falle aufgrund technischer Möglichkeiten zwingend erforderlich.

Charakter des Algorithmus. Zusätzlich zu den Synchronisationsanforderungen, die durch die Prozeßgranularität hervorgerufen werden, entstehen weitere aufgrund von Präzedenzbedingungen. Die Präzedenzrelationen können beispielsweise durch Präzedenzgraphen (Präzedenzmatrix, Konnektivitätsmatrix, vgl. Abschnitt 4.2.2) veranschaulicht werden. Die Synchronisationsanforderungen wirken sich auf die Zuweisung von Prozessen zu Prozessoren und das *Scheduling* der verschiedenen Komponenten des Algorithmus aus [Prz,92]. Die Abfolge von Prozeßerzeugung und -vernichtung beeinflußt die Prozessorauslastung, das Scheduling der Subprozesse, die Speicherorganisation und den Kommunikationsbedarf. Liegen Prozeßaufteilung und Synchronisationsbedarf vor Aus-

führung fest, so wird der parallele Algorithmus als **statisch** bezeichnet, ergeben sich Interaktionen erst zur Laufzeit, so handelt es sich um einen **dynamischen** Algorithmus [Qui,88]. In der Regel sind statische Algorithmen vom Typ SIMD, während dynamische Algorithmen MIMD erzwingen. Darüber hinaus macht ein dynamischer Algorithmus entweder einen gemeinsamen Speicher oder ein sehr schnelles Verbindungsnetzwerk erforderlich.

Grundlegende Operationen. Die grundlegenden Operationen bestimmen die notwendigen Fähigkeiten der Prozessoren. Ebenso nehmen sie Einfluß auf den Kommunikationsbedarf, der zum Teil durch die Bandbreite spezifiziert wird, sowie auf die Speicherorganisation. Datentypen und Genauigkeit zielen ebenfalls auf die Leistungsfähigkeit der Prozessoren sowie auf die Speicheranforderungen und die Kommunikationsbandbreite. Weiterhin entstehen Forderungen bezüglich der sprachlichen Mittel.

Datenabhängigkeiten und Datenstrukturen. Die Datenabhängigkeiten spielen die größte Rolle bei der Bestimmung der Datenallokationsmuster und der Kommunikationscharakteristik. Sie haben ebenso einen wesentlichen Anteil an der Entscheidung für globalen oder lokalen Speicher. Die Analyse der Datenabhängigkeiten gehört zu den Kernbestandteilen der Abbildung paralleler Algorithmen auf parallele Hardware. Viele Algorithmen haben natürliche Datenstrukturen (Vektoren, Matrizen, Bäume, ...), auf denen die Operationen ausgeführt werden. Die Fähigkeit der Architektur, diese Datenstrukturen und die damit zusammenhängenden Zugriffsmuster zu unterstützen und mögliche Regelmäßigkeiten durch Architekturmerkmale auszunutzen, hat großen Einfluß auf die Leistung, die mit dem Algorithmus erzielt wird; dies zeigt auch die Konstruktion von Spezialprozessoren.

Kommunikationsgeometrie. Faßt man die Module des zu beschreibenden Algorithmus als die Knoten eines Graphen auf und fügt man für miteinander kommunizierende Module Pfade zwischen den entsprechenden Knoten ein, so definiert der resultierende Graph die Kommunikationsgeometrie des parallelen Algorithmus. Es ist häufig möglich, für ein bestimmtes Problem Algorithmen mit unterschiedlicher Kommunikationsgeometrie zu entwerfen [Kun,80]. Ein Beispiel hierfür ist das Sortieren, welches auf Netzwerken erfolgen kann, deren Geometrie einem ein- oder mehrdimensionalen Feld oder einem Shuffle-Exchange entspricht.

Sieht man die oben angeführten Eigenschaften als Dimensionen an, so spannt ihr Kreuzprodukt den Raum paralleler Algorithmen auf. Somit ist eine Taxonomie von Algorithmen möglich, indem man jedem Algorithmus seine Position innerhalb dieses n-dimensionalen Raumes zuweist. Naturgemäß sind viele Teilräume des Gesamtraumes schlicht leer oder von geringem Interesse. Dies ist beim Aufbau einer logisch konsistenten Klassifizierung jedoch im allgemeinen nicht vermeidbar und wird daher in Kauf genommen. Bei allen diesen Charakteristiken muß unterschieden werden zwischen wählbaren und erzwungenen Eigenschaften. Beispielsweise setzt SIMD-Verarbeitung Uniformität zwingend voraus; ist andererseits SIMD einsetzbar, so kann im allgemeinen auch ein MIMD-System mit annähernd gleicher Effizienz eingesetzt werden.

4.2 Etablierte Beschreibungsmethoden für die Struktur paralleler Algorithmen

4.2.1 Sprachkonzepte

Imperative Programmiersprachen. Mit dem Aufkommen paralleler Rechner wurden auch sprachliche Mittel notwendig, um die parallele Ausführung von Progammen oder Programmteilen zu spezifizieren. Hierzu wurden verschiedene Wege beschritten [BST,89]:

- Entwicklung neuer Sprachen speziell im Hinblick auf Parallelisierung
  Beispiel: CSP (Communicating Sequential Processes) [Hoa,85]
- Erweiterung von Sprachen um parallele Konstrukte
  Beispiel: FORTRAN D [HKT,92]
- Rechnerspezifische Präprozessoren für parallelitätsdefinierende Direktiven
  Beispiel: Microtasking (CRAY-Multitasking-Konzept) [Bur,89]

Mit diesen Methoden kann Parallelität so beschrieben werden, daß Compiler und Betriebssystem eines Multiprozessorsystems in der Lage sind, einen parallelen Ablauf mehrerer Prozesse zu bewerkstelligen. In den traditionellen imperativen Programmiersprachen wie FORTRAN, PASCAL und C erlauben die herkömmlichen Kontrollstrukturen a priori keinen parallelen Ablauf und müssen auf Spracherweiterungen oder Präprozessoren zurückgreifen. Auf die Erläuterung grundlegender paralleler Konstrukte wird an dieser Stelle verzichtet; statt dessen wird exemplarisch auf die Monographie von Brandes [Bra,88] hingewiesen.

Deklarative Programmiersprachen. Im Gegensatz zu den imperativen Programmiersprachen arbeiten deklarative Programmiersprachen nicht mit Variablen und Wertzuweisungen sondern durch direktes Operieren auf Funktionen. Deklarative Programmiersprachen verfügen aufgrund ihrer funktionalen Ablaufstruktur über implizite Parallelität; hier wurde quasi schon immer auf die Einsetzbarkeit paralleler Rechner „gewartet". Zu den deklarativen Programmiersprachen zählen sowohl Logik-Sprachen wie PROLOG [S&S,86] als auch funktionale Sprachen [Loo,89].

Das Datenflußkonzept. Das im wesentlichen von Jack B. Dennis am MIT entwickelte Datenflußkonzept [Den,74] stellt eine völlige Abkehr vom bis dahin bekannten Kontrollflußprinzip dar. Statt der gewohnten Ablaufkontrolle durch explizit eingefügte Kontrollstrukturen ergibt sich in einen Datenflußprogramm die aktuelle Abfolge der auszuführenden Instruktionen implizit. Die Reihenfolge, in der die Anweisungen eines Programms aufgeführt sind, ist nicht maßgebend; vielmehr sind alle Anweisungen ausführbar, deren Inputvariablen bereits vollständig vorliegen. Sind entsprechend viele Prozessoren verfügbar, können alle diese Anweisungen gleichzeitig ausgeführt werden. Auf diese Weise kann jeglicher vorhandener Parallelismus direkt ausgenutzt werden. Aufgrund des Fehlens von effizienten Hardware-Umsetzungen konnte sich das Konzept bislang in der Praxis noch nicht durchsetzen.

4.2.2 Präzedenzgraph, Präzedenzmatrix, Prozeßsystem

Ein **Präzedenzgraph** ist ein gerichteter azyklischer Graph (*directed acyclic graph, DAG*) (V,E), dessen Knoten zu individuellen Anweisungen korrespondieren. Eine gerichtete Kante vom Knoten v_1 zum Knoten v_2 bedeutet, daß die Anweisung v_2 erst ausgeführt werden darf, wenn die Ausführung der Anweisung beendet ist, die durch den Knoten v_1 repräsentiert wird [P&S,85].

Die Darstellungsform des Präzedenzgraphen ist universell in dem Sinne, daß sich alle vollparallelen, teilüberlappenden und natürlich auch sequentiellen Situationen damit modellieren lassen. In dieser Hinsicht entspricht der Präzedenzgraph in seiner Funktionalität beispielsweise der Kontrollstruktur FORK/JOIN, während das von Dijkstra vorgeschlagene Konzept PARBEGIN/PAREND nicht die gleiche Mächtigkeit besitzt [P&S,85]. Die Modellierungsfähigkeiten reichen jedoch für alle praxisrelevanten Fälle aus. Abb. 21 zeigt ein Beispiel für einen Präzedenzgraphen.

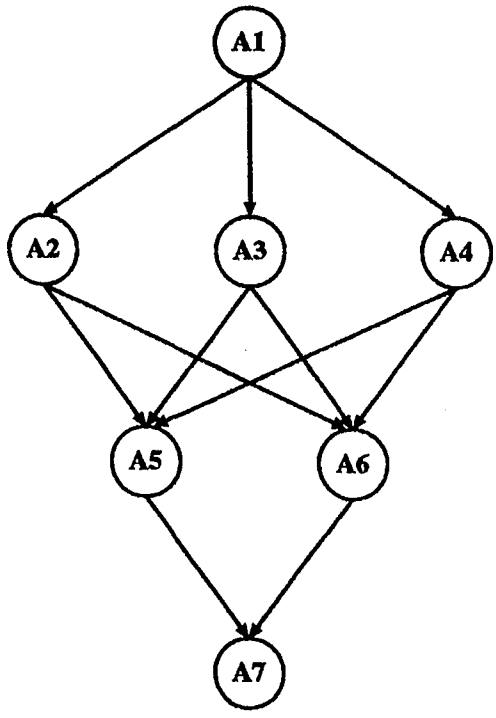


Abb. 21. Präzedenzgraph

Ein dem Präzedenzgraphen gleichwertiges Modellierungswerkzeug stellt die sogenannte *minimale Konnektivitätsmatrix* [KNU,75] dar, die als Adjazenzmatrix des Präzedenzgraphen aufgefaßt werden kann. Eine minimale Konnektivitätsmatrix K läßt sich mit der Formel $K = R \wedge \neg R^2$ aus der sogenannten *Präzedenzmatrix* R berechnen, die auch die indirekten (transitiven) Beziehungen der Anweisungen (Prozesse) widerspiegelt. Ein weiteres äquivalentes Modellierungskonzept ist das *Prozeßsystem* [KNU,75], das die Beziehungen formal in aufzählender Form beschreibt. Beispiel 4.1 zeigt das Prozeßsy-

stem, die Präzedenzmatrix und die minimale Konnnektivitätsmatrix zu dem in Abb. 21 dargestellten Präzedenzgraphen.

Beispiel 4.1

a) Prozeßsystem

$\wp = (\{P\}, S, s_0)$ mit $P = (\Phi, C)$ bzw.

$\wp = (\{(\Phi,(\{A1, A2, ..., A7\}, \prec))\}, S, s_0)$ mit

$\prec := \{A1 \prec A2, A1 \prec A3, A1 \prec A4, A2 \prec A5, A2 \prec A6, A3 \prec A5, A3 \prec A6,$
$A4 \prec A5, A4 \prec A6, A5 \prec A7, A6 \prec A7\}$

b) Präzedenzmatrix R

$\prec$	A1	A2	A3	A4	A5	A6	A7
A1	0	1	1	1	1	1	1
A2	0	0	0	0	1	1	1
A3	0	0	0	0	1	1	1
A4	0	0	0	0	1	1	1
A5	0	0	0	0	0	0	1
A6	0	0	0	0	0	0	1
A7	0	0	0	0	0	0	0

c) minimale Konnektivitätsmatrix K

$\rightarrow$	A1	A2	A3	A4	A5	A6	A7
A1	0	1	1	1	0	0	0
A2	0	0	0	0	1	1	0
A3	0	0	0	0	1	1	0
A4	0	0	0	0	1	1	0
A5	0	0	0	0	0	0	1
A6	0	0	0	0	0	0	1
A7	0	0	0	0	0	0	0

4.2.3 Graphentheoretische Beschreibung

Zahlreiche Autoren haben unterschiedlichste, vor allem graphentheoretische, Modelle zur Beschreibung von Parallelität vorgeschlagen ([L&L,90], [M&E,92]). Diesen Entwürfen liegen verschiedene Schwerpunkte hinsichtlich ihrer Zielrichtungen zugrunde:

- die Gestalt modularer paralleler Systeme und ihrer Verbindungsstruktur
- Kontrolltechniken, die es erlauben, daß Prozeduren und Daten gemeinsam von mehreren Prozessoren genutzt werden.
- Beziehungen zwischen sequentiellen Programmen und ihrer transformierten Darstellung in parallele Form
- Vorhersage bezüglich der Leistungszunahme und der zusätzlichen Kosten beim Übergang vom sequentiellen zum parallelen Rechner

Abhängigkeitsgraphen bilden die Grundlage für Compiler-Optimierung und das Erkennen von Parallelität in Programmen [Bro,85].

Das UCLA-Modell von Estrin und Turn [E&T,63] basiert auf sogenannten d.a.b.-Graphen (*directed acyclic bilogic graphs*) und zielt auf den Einsatz auf einem Rechner mit veränderlicher Struktur. Das Modell setzt auf einem FORTRAN-Quellprogramm auf und kann automatisch hergeleitet werden. Die Knoten entsprechen einzelnen FORTRAN-Anweisungen und der Kontrollfluß wird durch Pfeile repräsentiert. Jedem Knoten ist eines der folgenden vier Paare zugeordnet: (*,*), (*,+), (+,*), (+,+). Hierbei bezieht sich das erste Element eines Paares auf die Inputlogik und das zweite Element auf die Outputlogik des jeweiligen Knotens. '+' repräsentiert XOR-Logik, während '*' AND-Logik bezeichnet. Anstelle von Determinismus wird angesichts des Fehlens von Variablenhistorien nur „Legalität" überprüft. Ein Graph heißt **legal**, wenn es einen eindeutigen Startknoten und einen eindeutigen Endknoten gibt und alle Subgraphen, die sich während der Simulation ergeben, vom AND-Typ sind.

Petri-Netze [Rei,82] sollen dazu dienen, sowohl statische als auch dynamische Parallelitätsbedingungen zu charakterisieren und insbesondere soll die Koordination asynchroner Ereignisse modelliert werden. Ein **Petri-Netz** ist ein Tripel $\wp$ = (W,U,C), wobei $W = \{w_1, w_2, ..., w_n\}$ eine endliche Menge von Transitionen ist, die zu Ereignissen korrespondieren, und $U = \{u_1, u_2, ..., u_m\}$ eine endliche Menge von Stellen ist, die zu Bedingungen korrespondieren. Die Kontrolle C besteht im wesentlichen aus einer Menge $A = \{a_1, a_2, ..., a_p\}$ von Pfeilen, die Elemente von W mit Elementen von U verbinden und umgekehrt. Falls $a_l = (u_j, w_k)$, so ist u_j eine Input-Stelle für die Transition w_k und die Menge der Input-Stellen für w_k ist definiert durch $I_k = \{u_j \mid (u_j, w_k) \in A\}$. Output-Stellen für eine Transition sowie die Menge der Output-Stellen O_k sind analog definiert.

Stellen sind in der Lage, Marken zu empfangen. Sie können entweder voll sein, d. h. sie besitzen eine oder mehrere Marken, oder sie sind leer, d. h. sie besitzen keine Marken. Da Stellen Bedingungen repräsentieren, zeigt eine volle Stelle eine erfüllte Bedingung an. Das Erfülltsein von Bedingungen bestimmt das Schalten von Transitionen, d. h. das Eintreten von Ereignissen, und das Schalten von Transitionen wiederum dirigiert das Erfüllen von Bedingungen. Um zu schalten muß eine Transition alle eigenen Input-Stellen gefüllt haben. Das Schalten von w_k bewirkt dann das Entfernen jeweils einer Marke von allen Elementen von I_k und das Belegen jedes Elementes von O_k mit einer, eventuell zusätzlichen, Marke. Ein **Zustand** ist eine Instantiierung der Netzhistorie und wird durch eine bestimmte Verteilung von Marken über die Stellen spezifiziert. Das Schalten einer Transition ändert den Zustand des Netzes. Eine Simulation des Netzes orientiert sich an Algorithmus 1.

Das Konzept des Determinismus wird im Rahmen der Petri-Netze durch die Konzepte „Sicherheit" und „Lebendigkeit" ersetzt. Ein Petri-Netz heißt **sicher**, wenn jede Stelle zu keiner Zeit mehr als eine Marke besitzen kann. Eine Transition w_l heißt **lebendig**, wenn für einen gegebenen Ausgangszustand stets eine Folge von Zuständen existiert, so daß w_l schalten wird. Ein Netz heißt **lebendig**, wenn alle seine Transitionen lebendig sind. Determinismus ist nicht gegeben, da Konflikte möglich sind. Zwei Transitionen sind im **Konflikt**, wenn sie eine gemeinsame Input-Stelle besitzen und die beiden Transitionen aktiviert werden.

Algorithmus 1: Simulation im Petrinetz

1. Festlegung der Ausgangsbedingungen, d. h. die Menge derjenigen Stellen $U_0 \subseteq U$, die bereits zu Anfang Marken besitzen. Dies ist der Startzustand.
2. In einem aktuellen Zustand wählt man irgendeine der Transitionen aus, die in der Lage sind zu schalten, und schaltet diese. So wird der nächste Systemzustand definiert.
3. Das Verfahren bricht entweder nach einer vorher festgelegten Zahl von Schaltungen ab oder dann, wenn kein weiteres Schalten erfolgen kann.

Das Modell von Rodriguez [Rod,69] stellt hinsichtlich Interpretation der Operatoren und Präzisierung der Kontrolle ein im Sinne einer Verfeinerung über die Modelle von Estrin/Turn und Petri hinausgehendes Konzept dar. Es existieren zwei Arten von Pfeilen: Datenpfeile, die sowohl einen Zustand als auch einen Wert besitzen, und Kontrollpfeile, die nur einen Zustand besitzen. Die Beendigung einer Berechnung beeinflußt nicht nur den Transfer von Werten sondern ebenso Zustandsänderungen in hereinkommenden und hinausgehenden Pfeilen. Eine Simulation im Modell von Rodriguez verläuft dann entsprechend Algorithmus 2.

Algorithmus 2: Simulation nach Rodriguez

1. Im Ausgangszustand einer Simulation sind alle Pfeile untätig mit Ausnahme derjenigen, die mit aktivierten Eingabestationen verbunden sind.
2. Der Zustand des Graphen wird bestimmt durch die Zustände und Dateninhalte der Pfeile.
3. Der Zustand des Graphen entscheidet, welche Knoten ihre Operationen ausführen können.
4. Die **Ausführungssequenz** (Simulation) ist dann die Folge der Zustände des Graphen.
5. Ein Endzustand ist ein Zustand, in dem kein Knoten aktiviert werden kann.
6. Ein Verklemmungszustand ist ein Endzustand, mit wenigstens einem Pfeil, der nicht im Zustand „untätig" ist.

Notwendige und hinreichende Bedingungen, die gewährleisten, daß eine Simulation nicht in einen Verklemmungszustand läuft, sind bekannt. Der Determinismus der Modelle, die zu einem regulären Ende kommen, wurde nachgewiesen.

Das Graphenmodell PCM (*Parallel Computation Model*) von Adiga und Browne [A&B,86] dient dazu, das Verhalten von Programmen zu analysieren, die in der Sprache CSL formuliert wurden ([Bro,85],[O&B,86]). Diese wurde an der Universität von Texas speziell für den *Texas Reconfigurable Array Computer* (TRAC) entwickelt. CSL unterstützt die dynamische Strukturierung von Berechnungen über mehrere Phasen, von denen jede unterschiedliche Grade und Typen von Parallelität beinhalten kann. Eine

parallele Berechnung besteht aus einer Sammlung von Tasks und die Spezifikation von Abhängigkeiten zwischen ihnen.

Das PCM-Modell basiert auf dem Petri-Netz-Modell und ist mit Erweiterungen versehen, die es erlauben, die Leistung paralleler Berechnungen auszuwerten. Jede Transition t_i besitzt ein korrespondierendes Prädikat $\pi_i(V)$ und eine Transitionsprozedur $\phi_i(V)$. Die Prädikate sind auf Programmvariablen definiert und werden benutzt, um Transitionen zu aktivieren. Die Transitionsprozeduren operieren ebenso auf Elementen von V und können diese modifizieren. Die Funktion $\tau_i(V)$ spezifiziert die Verzögerung, die mit der jeweiligen Transition t_i verbunden ist, und kann ebenso eine Funktion auf der Menge der Programmvariablen sein. Das PCM-Modell stellt ein Konzept dar, um den Konfigurationenraum einer gegebenen parallelen Berechnung zu studieren und zu entscheiden, welche Konfiguration für eine gegebene Berechnung und gegebene Architektur die geeignetste ist.

Das Modell von Bokhari [Bok,81], welches direkt im Hinblick auf eine Anwendung im Rahmen des Mapping (vgl. Kapitel 5) konzipiert ist, setzt nicht mehr auf einzelnen Anweisungen auf, sondern spricht nur noch von (Programm-)Modulen, deren expliziter Aufbau a priori nicht restringiert ist und auch nicht näher interessiert. Das Abstraktionsniveau ist also wesentlich höher oder gröber als bei den vorangegangenen Modellen.

Ein Problem, welches mittels Parallelverarbeitung auf einem verteilten Rechner berechnet werden soll, wird modelliert durch einen Graphen $G_p = (V_p, E_p)$; hierbei repräsentiert die Knotenmenge V_p die Menge der im Programm vorhandenen Module. Eine Kante $(m_i, m_j) \in E_p$ bedeutet, daß die Module m_i und m_j miteinander kommunizieren. Aufgrund der Symmetrie und der Irreflexivität der Kommunikation kann man das Problem auch als Funktion F_p: $V_p \times V_p \rightarrow \{0,1\}$ auffassen, für die gilt: $F_p(m_i, m_j) = F_p(m_j, m_i)$ und $F_p(m_i, m_i) = 0\ \forall m_i, m_j \in V_p$. $F_p(m_i, m_j) = 1$ bedeutet dann, daß die Module m_i und m_j miteinander kommunizieren, bzw. daß im Graphen G_p eine Kante $(m_i, m_j) \in E_p$ existiert. Abb. 22 zeigt einen möglichen Problemgraphen nach Bokhari.

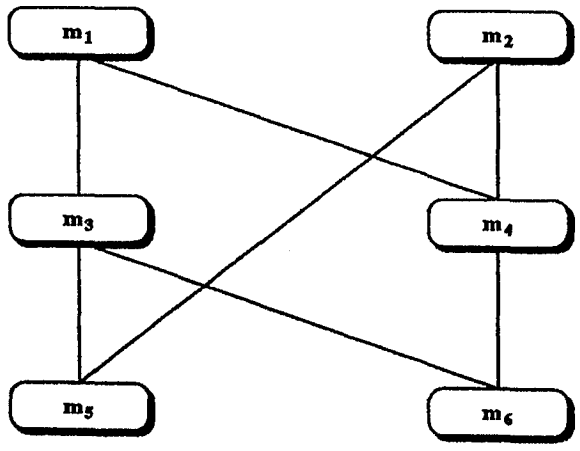


Abb. 22. Problemgraph nach Bokhari

4.3 Kombinatorische Beschreibung paralleler Algorithmen

4.3.1 Konzeption und Restriktionen

Das Studium paralleler Phänomene und der zugehörigen Algorithmentransformationstechniken erfordert einen strengen mathematischen Rahmen [F&M,85]. Jede formale Behandlung von Parallelität muß mit dem Entwurf eines formalen Modells beginnen [Par,87]. Die ersten – graphentheoretischen – Modelle zur Beschreibung paralleler Berechnungen liegen etwa 30 Jahre zurück. Bereits zu dieser Zeit wurden Klassenbildung und Algorithmentransformationstechniken gefordert [K&M,66] oder gar rekonfigurierbare Rechner vorgeschlagen [E&T,63]. Die Verwendung kombinatorischer Hilfsmittel, insbesondere der Theorie der optimalen Versuchspläne, zur Beschreibung paralleler Phänomene dient im Fall der Spezifikation paralleler Algorithmen der Verfolgung zweier Ziele:

- Es soll die strenge Ordnung der kombinatorischen Theorie auf die Formulierung der Parallelität übertragen werden. Hierzu dienen Definitionen und Gesetzmäßigkeiten der kombinatorischen Theorie der optimalen Versuchspläne.

- Es soll die größere Mächtigkeit der kombinatorischen Beschreibungsmethode gegenüber der üblichen, graphentheoretischen Beschreibungsmethode gezeigt werden, wobei die, als Quasistandard akzeptierte, graphentheoretische Sichtweise nicht völlig verloren gehen soll.

Liegen umfangreichere parallele Algorithmen in einer imperativen Programmiersprache vor, so erweisen sich die auf einzelnen Anweisungen basierenden Modelle von Estrin und Turn, Petri, Rodriguez sowie Adiga und Browne als zu feingranular und damit unpraktikabel für die praktische Anwendung; im Modell von Bokhari hingegen geht die zeitliche Komponente, also die Präzedenzen, völlig verloren, weshalb diese Abstraktion hier als zu weitgehend angesehen wird. Die erstgenannten Modelle sind im wesentlichen nur dazu geeignet, das Wesen der Parallelität zu verstehen. Sie eignen sich dazu, bestimmte Einzelsituationen in parallelen Abläufen nachzuvollziehen. Soll die Parallelisierung auf einem Rechner auf diesem feinstgranularen Niveau durchgeführt werden, so erscheint das Mittel der funktionalen Programmierung oder das Datenflußprinzip als geeigneter. Das hier zu enwickelnde Modell, das auf kombinatorischen Versuchsplänen basiert, ist daher vom Abstraktionsniveau zwischen den feingranularen Modellen und dem Modell von Bokhari angesiedelt.

Die Anlage des Modells ist von seiner Wichtigkeit her kaum zu überschätzen; ist das Modell schlicht falsch oder auch nur ungeeignet, so können alle weiterführenden Betrachtungen im Sinne des „ex falso quodlibet" völlig unbrauchbar werden. Die Auswahl des geeigneten Modells ist von entscheidender Bedeutung, da es hinreichend ausdrucksstark sein muß, um die Berechnungsanforderungen, die die verschiedenen Aktivitäten und zeitlichen Zwänge beinhalten, hinreichend beschreiben zu können; andererseits muß das Modell einfach genug sein, damit es in die Praxis umsetzbar ist [Mok,85].

Um die Problematik der Mehrdeutigkeit des Prozeßbegriffs zu vermeiden und gleichzeitig eine Abstraktion von der mit dem Prozeßbegriff eng assoziierten Aufwands- oder Granularitätsvorstellung zu erreichen, wird im folgenden der Begriff **Modul** verwendet,

der für mathematisch-theoretische Modelle besser geeignet ist. Da für einen bestimmten Algorithmus je nach Anwendung eine Modularisierung in mehr oder minder starkem Maße erfolgen soll, wird hier von einem geeigneten Beschreibungsmechanismus gefordert, daß die unterschiedlich zu wählenden Granulierungen einheitlich, im Sinne von logischer Konsistenz, formalisiert werden können. Ein Präzedenzgraph beispielsweise zwingt zu möglichst feingranularer Strukturierung; es ist nicht möglich, mehrere Module zu einem Gesamtmodul zusammenzufassen, was hinsichtlich einer, in Relation zur Modulzahl, geringen Prozessorzahl durchaus von Interesse sein kann. Die Vorteilhaftigkeit, verschiedene Ebenen der Granulierung im Modell zur Verfügung zu haben, wird auch von Yau und Shatz [Y&S,82] betont.

Ein paralleler Algorithmus, wobei in einer ersten Ausbaustufe des Modells nur statische Algorithmen betrachtet werden, die mit dem PARBEGIN/PAREND-Konstrukt modellierbar sind [P&S,85], wird als eine Menge $M = \{M_1, M_2, ..., M_n\}$ selbständiger Module aufgefaßt. Zwischen diesen Modulen können Präzedenzbeziehungen bestehen, anderenfalls ist parallele Ausführung von zwei oder mehr Modulen möglich. Zeitgleich oder teilüberlappend aktive Module können miteinander kommunizieren. Wie bei Preparata [Pre,84] sowie Wu und Shu [W&S,90] wird ein Modul als atomar angesehen; die innere Struktur des Moduls ist nicht analysierbar. Schleifen und Verzweigungen, die sich innerhalb von Modulen befinden, verschwinden durch diese Abstraktion. Außerhalb der separat ausführbaren Module befindliche Verzweigungen werden durch Platzhaltermodule, externe Schleifen durch explizite Notation der Einzelmodule aufgelöst. Ein entsprechender Ablaufgraph ist somit zyklenfrei, wie dies auch an anderer Stelle gefordert wird [K&B,88].

Ein paralleler Algorithmus besteht im allgemeinen aus mehreren Teilalgorithmen bzw. Abschnitten, die im folgenden als **Phasen** bezeichnet werden. Können aufgrund des makroskopischen Modellansatzes keine Phasen erkannt werden, so ist der gesamte Algorithmus in eine Phase zu fassen; die Genauigkeit des hier vorgestellten Modells sinkt dann auf die des Modells von Bokhari ab. Innerhalb der Phasen erfolgen in den in dieser Phase existenten Modulen Operationen, die eventuell Modifikationen von Daten bewirken. Die Gesamtheit der in allen Phasen und Modulen ausgeführten Operationen entspricht der Semantik des parallelen Programms.

Jede Phase des parallelen Algorithmus wird im kombinatorischen Algorithmenmodell durch einen eigenen Versuchsplan modelliert. Kommunizierende Module werden durch Blöcke des zugehörigen Mengensystems repräsentiert. Um die Beziehungen von Modulen in verschiedenen Phasen i,j zu beschreiben, dienen Relationen φ_{ij}, die Module des Versuchsplans für Phase i auf Module des Versuchsplans für Phase j abbilden. Diese Relationen werden in der Relationenfolge Φ zusammengefaßt. Φ beschreibt also insbesondere auch Präzedenzrelationen bzw. Abhängigkeitsbeziehungen zwischen Modulen. Man kann die Zusammenhänge auch so auffassen, daß aufgrund der zwischen Modulen bestehenden Präzedenzen neue Phasen generiert werden müssen. Hinzu kommt die Kreierung neuer Phasen wegen einer geänderten Kommunikationsstruktur unter den Modulen. Der parallele Algorithmus kann dann modelliert werden durch das Gesamtgebilde, das aus der Folge der Versuchspläne und der Relationenfolge Φ besteht. Diese Struktur wird als **Versuchsplansequenz** bezeichnet (vgl. Anhang A).

4.3.2 Berücksichtigung der Algorithmencharakteristiken nach Jamieson

Die Darstellung der Kommunikationsstruktur eines Algorithmus als kombinatorischer Versuchsplan ist naturgemäß eine sehr theoretische, mathematische Herangehensweise. Um zu zeigen, daß praktische Erwägungen dennoch nicht zu kurz kommen und somit die Untersuchungen letztendlich auch von praktischem Nutzen sind, wird nun dargelegt, wie und in welchem Maße die von Jamieson vorgeschlagenen, praxisnahen Kenngrößen zur Charakterisierung von Algorithmen in das kombinatorische Modell miteinfließen.

Natur des Parallelismus. Es wird nicht zwischen Daten- und Funktionsparallelismus unterschieden; entscheidend ist nur, ob grundsätzlich parallele Ausführung möglich ist. Jedoch tritt in der praktischen Anwendung im Zusammenhang mit dem Mapping aufgrund inhärenter Zwänge die Unterscheidung zwischen Daten- und Funktionsparallelismus implizit zutage; wenn beispielsweise ein massiv-paralleles System effizient genutzt werden soll, so wird der entsprechend geforderte hohe Grad an Parallelismus im allgemeinen nur von einem datenparallelen Algorithmus geliefert.

Grad des Parallelismus. Die feinstmögliche Granulierung des Algorithmus, sei sie nun durch die gröbere Datengranularität oder durch die gröbere Modulgranularität bestimmt, legt die maximale Zahl der Module eines Phasenversuchsplans fest. Ebenso wie sich im Modell von Jamieson der Grad des Parallelismus aus Daten- und Modulgranularität ergibt, so bestimmt der Grad des Parallelismus im kombinatorischen Modell die Zahl der maximal zu berücksichtigenden Module.

Charakter des Algorithmus. Es werden nur statische Algorithmen betrachtet, wobei notwendige Synchronisationen im kombinatorischen Modell in Form von Kommunikation auftreten. Muß beispielsweise ein Modul auf das Ende einer Berechnung warten, die von einem anderen Modul durchgeführt wird, so erhält das wartende Modul nach Abschluß der Operationen eine Nachricht vom rechnenden Modul. Je häufiger die Kommunikationsgeometrie wechselt, desto mehr Phasen werden unterschieden und desto größer wird die Zahl der Versuchspläne der Versuchsplansequenz. Je häufiger Kommunikation zwischen Modulen einer Phase erfolgt, desto höher wird die spezifische Verknüpfungsdichte des zugehörigen Phasenversuchsplans, die formal durch die t-Indexmenge Λ_t verkörpert wird.

Grundlegende Operationen. Je nach Art der grundlegenden Operationen wird mehr oder minder starke Kommunikation erforderlich. Datentypen können Einfluß auf die notwendige Kommunikation nehmen, Genauigkeiten spielen beim kombinatorischen Modell keine Rolle. Im Hinblick auf die zunehmende Verwendung von Mikroprozessoren mit 64 bit-Arithmetik, wie des Intel i860, kann eine ausreichende Rechengenauigkeit im allgemeinen auch für verteilte Systeme vorausgesetzt werden, wenn Sonderfälle, die 1bit-Prozessoren einsetzen, an dieser Stelle unberücksichtigt bleiben.

Datenabhängigkeiten und Datenstrukturen. Der bei Jamieson festgestellte Einfluß von Datenabhängigkeiten auf die Kommunikationscharakteristik kommt auch innerhalb des Versuchsplans zum Tragen. Darüber hinaus nimmt die Zahl der Relationen zwischen verschiedenen Phasen mit wachsender Zahl von Datenabhängigkeiten zu. Die vom Pro-

gramm verwendeten Datenstrukturen nehmen insbesondere durch die davon bestimmten Datenzugriffsmuster Einfluß auf Art und Häufigkeit der Kommunikation.

Kommunikationsgeometrie. Die Kommunikationsgeometrie ist exakt der Punkt, an der die besondere Eignung kombinatorischer Versuchspläne zur Modellierung paralleler Strukturen zur Geltung kommt. Aufgrund der Vielseitigkeit kombinatorischer Versuchspläne kann hier mit mächtigen Instrumenten weit mehr ausgedrückt werden, als dies mit herkömmlichen Modellierungswerkzeugen, insbesondere Graphen, möglich ist.

4.3.3 Beschreibung der Kommunikationsstruktur durch Versuchspläne

Die Bezeichnungskonventionen und formale Syntax, deren sich die vorliegende Arbeit bedient, sind im Symbolverzeichnis im Anschluß an das Inhaltsverzeichnis zusammengefaßt. Für die graphische Darstellung der Kommunikationsgeometrie werden die drei in der Abbildung Abb. 23 dargestellten Symbole verwendet.

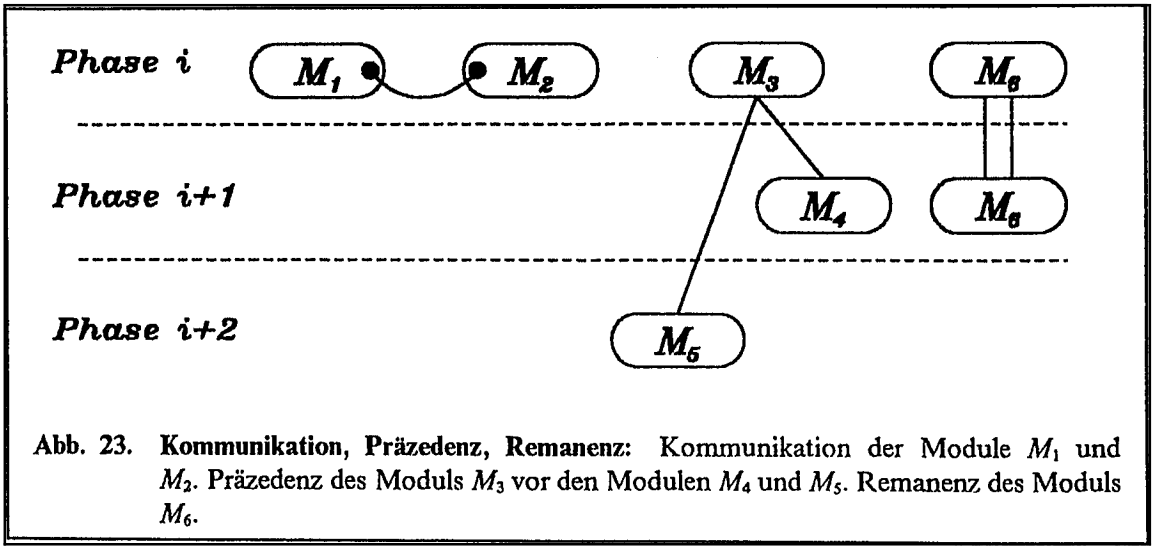


Abb. 23. Kommunikation, Präzedenz, Remanenz: Kommunikation der Module M_1 und M_2. Präzedenz des Moduls M_3 vor den Modulen M_4 und M_5. Remanenz des Moduls M_6.

Für die Pseudocodedarstellung der Algorithmen wird konform zur durchgeführten Restriktion auf das PARBEGIN/PAREND-Konstrukt [P&S,85] zurückgegriffen. Dieses entspricht den Grundsätzen der strukturierten Programmierung, welche durch das mächtigere FORK/JOIN und äquivalente Konstrukte aufgrund deren Realisierung unter Verwendung von Sprungbefehlen sicherlich verletzt würde. Darüber hinaus wird hier aus Gründen der Übersichtlichkeit das Konstrukt SEQBEGIN/SEQEND analog zum SEQ-Konstrukt der Sprache OCCAM [P&R,87] benutzt, um sequentielle Teilabläufe innerhalb paralleler Konstrukte zu verdeutlichen. Die Anweisungsfolge PARBEGIN; M_1; M_2; PAREND; erlaubt die parallele Ausführung der Module M_1 und M_2. Die Anweisungsfolge SEQBEGIN; M_3; M_4; SEQEND; erzwingt die Präzedenz des Moduls M_3 vor dem Modul M_4. Die Anweisungsfolge SEQBEGIN; PARBEGIN; M_{i1}; ...; M_{in}; PAREND; PARBEGIN; M_{j1}; ...; M_{jm}; PAREND; SEQEND; erzwingt die paarweise Präzedenz aller Module der Form M_{ir}, $1 \leq r \leq n$, vor allen Modulen der Form M_{js}, $1 \leq s \leq m$. Eine Schachtelung gleicher Konstrukte entspricht semantisch der alleinigen Benutzung des äußeren Konstruktes und muß daher hier nicht diskutiert werden.

Die ebenfalls dem OCCAM entstammende Anweisung CHAN vereinbart Kommunikationskanäle für die nachstehende Kontrollstruktur ähnlich wie dies im CSL-Modell (*computation structures language*) von O'Dell und Browne [O&B,86] geschieht. Auszuführende Operationen in den einzelnen Teilprozessen werden hier nicht näher spezifiziert; in der Diktion von Krayl [KNU,75] kann man deshalb von uninterpretierten Prozeßsystemen sprechen. Unabhängige Prozesse, dies können auch einzelne Anweisungen sein, werden als Module aufgefaßt und innerhalb der hier verwendeten Notation mit M_i bezeichnet. Abweichend von der CHAN-Notation in OCCAM werden hier zur Vereinfachung anstelle expliziter Kanalkommunikationsanweisungen lediglich zu jedem Modul durch Parametrisierung die entsprechenden Kommunikationspartner angegeben. Dabei wird nicht zwischen lesender und schreibender, sendender und empfangender Kommunikation unterschieden; es ist auf dieser theoretischen Betrachtungsebene nur von Interesse, zwischen welchen Modulen grundsätzlich Kommunikation besteht. Weiterhin unterbleibt die hier unwesentliche Typfestlegung der Kanäle. Aus Symmetriegründen wird eine wechselseitige Kommunikation als Eigenschaft beider Module angesehen und tritt entsprechend zweifach als Parameter auf.

Nach den bislang angestellten Vorüberlegungen stellt die Bestimmung bzw. Festlegung der Phasen sicherlich den kritischsten Teil bei der Ermittlung der Kommunikationsgeometrie eines parallelen Algorithmus dar. Die Phaseneinteilung kann beispielsweise erfolgen

- anhand einer gegebenen funktionellen Aufteilung des Algorithmus in mehrere chronologische Teile (häufig: Initialisierung, Berechnung, Ausgabe),
- durch Erzeugung des zugehörigen Präzedenzgraphen (Präzedenzmatrix, Konnektivitätsmatrix); anschließend eventuell Zusammenfassung mehrerer Anweisungen zu einem Modul,
- durch Verwendung anderer, oben in Auswahl besprochener, Modelle als Zwischenschritte; Ausgangsparameter: Zahl der Zustände in Simulationen.

Für die Bestimmung der Zahl der Phasen gelten folgende unteren und oberen Schranken:

- Die maximale Zahl von Phasen entspricht der maximalen Tiefenstaffelung des zugehörigen Präzedenzgraphen.
- Die minimale Zahl der Phasen ergibt sich aufgrund der Modul- und Kommunikationsstruktur des Algorithmus: Genau dann, wenn sich die Zahl der aktiven Module ändert, oder die Kommunikationsstruktur zwischen den Modulen – auch bei gleichbleibenden Modulen – eine andere Gestalt annimmt, wird eine neue Phase per definitionem eingeführt. Eine darüber hinausgehende Einführung neuer Phasen geschieht fakultativ.
- Hat man innerhalb eines Programmes eine Sequenz der Länge r mit jeweils wiederum $s_1, \ldots, s_r$ Subsequenzen, so hat die den Algorithmus beschreibende Versuchsplansequenz mindestens $\sum_{i=1}^{r} s_i$ Phasen bzw. Einzelversuchspläne.

Ohne Zweifel stellt die Untersuchung eines parallelen Algorithmus hinsichtlich einer Aufteilung in separat ausführbare Module, und damit auch die hier, nach eingeführter

Restriktion, vorzunehmende Phaseneinteilung, ein äußerst kompliziertes Problemfeld dar, auf daß hier verständlicherweise nicht in der Breite eingegangen werden kann. Die Problematik wird hier deshalb dahingehend ausgeklammert, daß für die Modellierung mit Versuchsplänen die Modulaufteilung eines Algorithmus als gegeben betrachtet wird. In der Praxis mag dies zur Zeit noch mehrheitlich manuell geschehen; in der Zukunft werden mehr und mehr rechnerunterstützte Werkzeuge zum Einsatz kommen [B&H,92b]. Wie sich eine Phaseneinteilung eines realen Algorithmus ergeben kann, zeigen die Beispiele 4.4 bis 4.6.

Die Transformation eines parallelen Algorithmus in einen kombinatorischen Versuchsplan orientiert sich an Algorithmus 3.

Algorithmus 3: Kombinatorische Spezifikation paralleler Algorithmen

1. gegeben: paralleler Algorithmus
2. bestimme die Zahl l der Phasen (Zeitabschnitte) des Algorithmus
3. ermittle für jede Phase i die Zahl m_i der in ihr enthaltenen Module
4. bestimme für jede Phase die Kommunikationsbeziehungen zwischen den entsprechenden Modulen
5. spezifiziere für jede Phase i den entsprechenden Versuchsplan $(V,\mathscr{B})_i$ aufgrund der Kommunikationsbeziehungen der Module $M_{i1}, ..., M_{im_i}$
6. füge die einzelnen Versuchspläne zu einer Folge von Versuchsplänen zusammen, welche die Kommunikationsstruktur in allen Phasen beschreibt $((V,\mathscr{B})) = (V,\mathscr{B})_1 ... (V,\mathscr{B})_l$
7. verfolge die Beziehungen von Modulen zunächst in aufeinanderfolgenden Phasen und dann mit inkrementell zunehmendem Abstand und beschreibe diese durch Relationen φ_{ij}
8. vereinige die Einzelrelationen φ_{ij} zur Relationenfolge Φ
9. konstruiere aus der Folge der Versuchspläne $((V,\mathscr{B}))$ und der Relationenfolge Φ die Versuchsplansequenz $(V^\times, \mathscr{B}^\times, \Phi, l)$, die die Kommunikationsstruktur und Modulhierarchie des Algorithmus beschreibt

---

Kommunizieren aus einer Menge von n Modulen innerhalb einer Phase alle Module paarweise miteinander, so kann dies durch einen n-elementigen Block beschrieben werden. Besteht die wechselseitige Kommunikation nur zwischen einem Teil der Module, so werden nur diese in einem (1) Block zusammengefaßt, die übrigen Module bilden dann nochmals einen oder mehrere Blöcke. Ein Modul kann in mehreren Blöcken auftreten und zwar dann, wenn es mit verschiedenen Modulen kommuniziert, die selbst untereinander nicht kommunizieren. Module, die innerhalb einer bestimmten Phase mit keinem anderen Modul kommunizieren, bilden innerhalb des Versuchsplans, der diese Phase

beschreibt, einen 1-elementigen Block. Auf diese Weise treten alle Module zumindest einmal in Blöcken auf, es existieren keine freie Elemente, und Verifikationen sind sehr leicht möglich. Abb. 24 zeigt eine mögliche Kommunikationsgeometrie unter den Modulen einer Phase. Daraus ergibt sich folgender Versuchsplan für diese Phase: $(V,\mathscr{B})$ mit $V = \{M_1, M_2, M_3, M_4, M_5\}$, $\mathscr{B} = \{(M_1, M_2),(M_2, M_3, M_4),(M_5)\}$.

Abb. 24. Beispiel für Kommunikation innerhalb einer Phase

Wenn innerhalb eines parallelen Konstruktes ein Modul M parallel zu einer Sequenz aus Modulen $M_1, ..., M_n$ ausgeführt werden kann, aber nicht nur Modul M_1 sondern ebenso Module M_i, M_j, $2 \leq i,j \leq n$, mit Modul M kommunizieren müssen, dann bleibt Modul M bis zur Phase max $\{i,j\}$ noch aktiv, tritt also auch in den entsprechenden Versuchsplänen dieser Phasen in bestimmten Blöcken auf. Abb. 25 zeigt ein Beispiel für einen derartigen, als **Remanenz** bezeichneten, Fall.

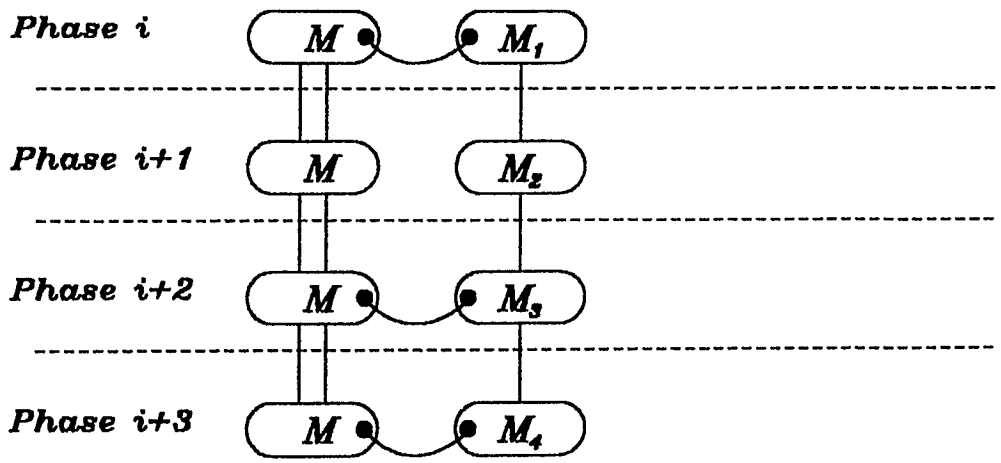


Abb. 25. Beispiel für notwendige Remanenz

Für die Versuchsplandarstellung bedeutet die Präzedenz aller Module einer Phase i vor einem Modul der Phase j, daß mit der Berechnung der Phase j erst begonnen werden kann, wenn alle Module der Phase i abgearbeitet sind. Die Aufteilung in Phasen ist jedoch nicht so zu verstehen, daß mit dem Beginn der Berechnungen in einer Phase $i+1$ grundsätzlich auf die Beendigung aller Module aus Phase i gewartet werden muß. Wenn beispielsweise ein Modul M in Phase i aufgesetzt wird und dann in späteren Phasen nicht mehr aktiv bleiben *muß*, weil keine Kommunikation mit anderen Modulen dieser Phasen $i+1, ..., i+r$ erfolgt, kann unter der Voraussetzung, daß keine Präzedenzen zwischen Modul M und Modulen aus Phase $i+1$ bestehen, die Phase $i+1$ auch dann bereits aufgesetzt werden, wenn Modul M noch nicht vollständig berechnet ist. Die Module

einer Phase j können daher beim Mapping (vgl. Kapitel 5) sofort auf den Zielrechner abgebildet werden, wenn keine Präzedenzen mehr vor diesen Modulen bestehen und ausreichend Prozessoren verfügbar sind. Zur Überprüfung, ob alle Präzedenzen vor einer Phase j abgearbeitet sind, werden alle Relationen φ_{ij} mit $i < j$ betrachtet, also alle, deren Bild in Phase j liegt. Abb. 26 verdeutlicht diesen Sachverhalt am Beispiel.

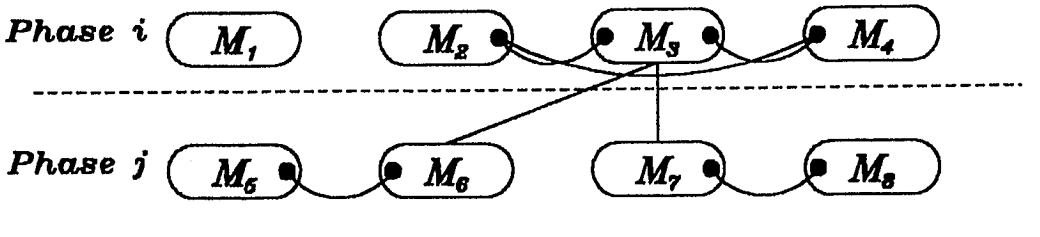


Abb. 26. Präzedenzen und Phasen

Module der Phase j können auf den Rechner im Wege des Mapping abgebildet werden, sobald die Module M_2, M_3, M_4 abgearbeitet sind. Auf die Beendigung von Modul M_1 muß nicht gewartet werden. Da Modul M_3 als Kommunikationspartner für die Module M_2 und M_4 dienen muß, wirkt sich die Präzedenz des Moduls M_3 vor Modulen aus Phase j dahingehend aus, daß mit den Berechnungen der Phase j auch auf die Beendigung der Module M_2 und M_4 gewartet werden muß. Besitzt der Rechner lediglich vier Prozessoren, so kann zunächst nur eines der beiden Modulpaare aus Phase j abgebildet werden, solange Modul M_1 noch aktiv ist.

Je nach Art des verwendeten Rechners und den zur Verfügung stehenden Scheduling-Mechanismen kann man zunächst ein Mapping auf virtuelle Prozessoren vorsehen; die Zuordnung der virtuellen Prozessoren zu den physikalisch tatsächlich vorhandenen übernimmt dann das Betriebssystem oder eine darüberliegende *Scheduling*-Oberfläche [Prz,92].

Aus Gründen der Übersichtlichkeit wurden die Module so indiziert, daß eine möglichst einfache Zuordnung zu Konstrukten, Phasen und benachbarten Modulen möglich wird. Dies darf jedoch nicht zu der Annahme verleiten, die kombinatorische Strukturierung beziehe sich in irgendeiner Form auf diese Indizes. Dies ist nicht der Fall; die Module könnten ebenso „sprechende Namen" besitzen, wie dies in größeren Software-Systemen aus Gründen der Verständlichkeit und Wartbarkeit praktiziert wird.

Eine sinnvolle Erweiterung des kombinatorischen Beschreibungsmodells bestünde darin, nicht nur Präzedenzen mittels Relationen darzustellen, sondern ebenso Datenübernahmen durch Module in späteren Phasen. Diese zusätzlichen Relationen könnten dazu dienen, die Zahl der Datentransferoperationen möglichst gering zu halten, wenn datenübernehmende Module in späteren Phasen nach Möglichkeit auf den gleichen Prozessor oder einen benachbarten abgebildet würden. Das Modell würde an Genauigkeit gewinnen, wenn für die unterschiedlichen Ausmaße der Kommunikation zwischen verschiedenen Modulpaaren, eine entsprechende Gewichtung in die Bewertung integriert würde.

4.3.4 Beispiele für die kombinatorische Beschreibungsmethode

Die im folgenden aufgeführten Beispiele mögen die kombinatorische Spezifikationsmethode hinsichtlich des notwendigen Buchführungsaufwandes als relativ aufwendig erscheinen lassen. Bei einer praktischen Anwendung würde diese Verwaltung jedoch vom Rechner übernommen, für den eine entsprechende Handhabung auch von mehreren tausend Modulen völlig problemlos wäre, wobei eine solche Modulanzahl bei dem hier projektierten makroskopischen Ansatz schon eine sehr feingranulare Applikation darstellt. Bei der graphischen Darstellung der Beispiele 4.4 bis 4.6 wird aus Gründen der Übersichtlichkeit nur ein signifikanter Teil der Präzedenzen abgebildet, der aufgrund der exakt gleichen Modulgranularitäten in den jeweiligen Phasen hier ausreichend ist. Um Konsistenz zwischen graphischer und kombinatorischer Darstellung zu gewährleisten, werden dann auch nur die abgebildeten Präzedenzen in der Relationenfolge der Versuchsplansequenz modelliert.

Beispiel 4.2 (Konstruiertes Beispiel)

a) Pseudocode

```
SEQBEGIN
  M11
  PARBEGIN
    M21
    M22
  PAREND
- CHAN com1,com2,com3,com4 :
  PARBEGIN
    M31{com1(M32),com1(M42)}
    SEQBEGIN
      M32{com1(M31),com2(M33),com3(M34)}
      M42{com1(M31),com2(M43)}
    SEQEND
    SEQBEGIN
      M33{com2(M32),com4(M34)}
      M43{com2(M42)}
      M53
    SEQEND
    M34{com3(M32),com4(M33)}
  PAREND
  M61
SEQEND
```

b) Graphische Darstellung der Modulbeziehungen

Abb. 27 veranschaulicht die Kommunikation der Module in den einzelnen Phasen.

c) Kombinatorische Beschreibung

Der parallele Algorithmus wird beschrieben durch die Versuchsplansequenz $(V^{\times}, \mathscr{B}^{\times}, \Phi, 6)$ mit $V^{\times} = (V_1, ..., V_6)$, $\mathscr{B}^{\times} = (\mathscr{B}_1, ..., \mathscr{B}_6)$.

Die Kommunikationsstruktur in den Phasen $i = 1,...,6$ wird durch folgende Versuchspläne beschrieben:

$i = 1$: $(V,\mathscr{B})_1$ mit $V_1 = \{M_{11}\}$; $\mathscr{B}_1 = \{(M_{11})\}$
$i = 2$: $(V,\mathscr{B})_2$ mit $V_2 = \{M_{21}, M_{22}\}$; $\mathscr{B}_2 = \{(M_{21}),(M_{22})\}$
$i = 3$: $(V,\mathscr{B})_3$ mit $V_3 = \{M_{31}, M_{32}, M_{33}, M_{34}\}$; $\mathscr{B}_3 = \{(M_{31}, M_{32}),(M_{32}, M_{33}, M_{34})\}$
$i = 4$: $(V,\mathscr{B})_4$ mit $V_4 = \{M_{31}, M_{42}, M_{43}\}$; $\mathscr{B}_4 = \{(M_{31}, M_{42}),(M_{42}, M_{43})\}$
$i = 5$: $(V,\mathscr{B})_5$ mit $V_5 = \{M_{53}\}$; $\mathscr{B}_5 = \{(M_{53})\}$
$i = 6$: $(V,\mathscr{B})_6$ mit $V_6 = \{M_{61}\}$; $\mathscr{B}_6 = \{(M_{61})\}$

Die Übergänge zwischen den einzelnen Phasen werden beschrieben durch die Folge der Relationen $\Phi = (\varphi_{12}, ..., \varphi_{56})$ mit:

$\varphi_{12} = \{(M_{11} \mapsto M_{21}),(M_{11} \mapsto M_{22})\}$
$\varphi_{23} = \{(M_{21} \mapsto M_{31}),(M_{21} \mapsto M_{32}),(M_{21} \mapsto M_{33}),(M_{21} \mapsto M_{34}),$
$(M_{22} \mapsto M_{31}),(M_{22} \mapsto M_{32}),(M_{22} \mapsto M_{33}),(M_{22} \mapsto M_{34})\}$
$\varphi_{34} = \{(M_{32} \mapsto M_{42}),(M_{33} \mapsto M_{43})\}$
$\varphi_{45} = \{(M_{43} \mapsto M_{53})\}$
$\varphi_{56} = \{(M_{53} \mapsto M_{61})\}$
$\varphi_{46} = \{(M_{31} \mapsto M_{61}),(M_{42} \mapsto M_{61})\}$
$\varphi_{36} = \{(M_{34} \mapsto M_{61})\}$

Für alle übrigen φ_{ij} $1 \leq i \leq 5$, $2 \leq j \leq 6$ gilt: $\varphi_{ij} = \{\}$

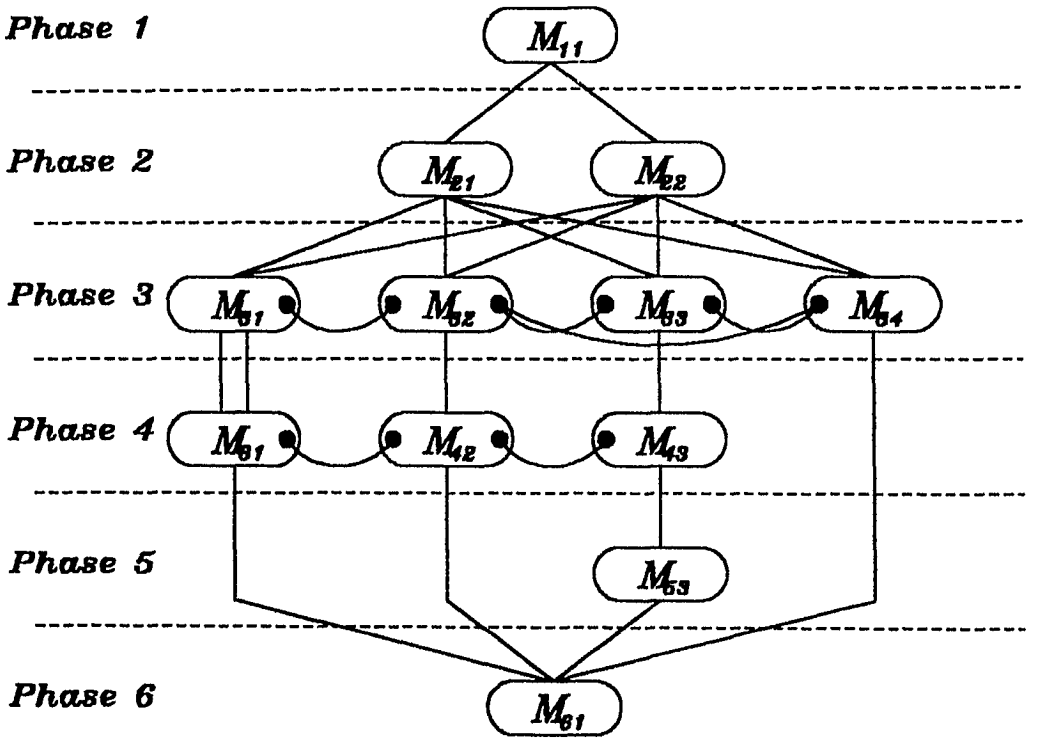


Abb. 27. Kommunikationsgeometrie des Algorithmus aus Beispiel 4.2

Beispiel 4.3 (Konstruiertes Beispiel)

a) Pseudocode

```
CHAN com1,com2,com3,com4,com5,com6 :
PARBEGIN
  M11{com1(M12),com2(M13),com3(M23)}
  M12{com1(M11),com4(M13)}
  SEQBEGIN
    M13{com2(M11),com4(M12)}
    M23{com3(M11)}
    CHAN com5,com6 :
    PARBEGIN
      M33{com5(M34)}
      M34{com5(M33),com6(M14)}
    PAREND
    M43
  SEQEND
  M14{com6(M34)}
PAREND
```

b) Graphische Darstellung der Modulbeziehungen

Abb. 28 veranschaulicht die Kommunikation der Module in den einzelnen Phasen.

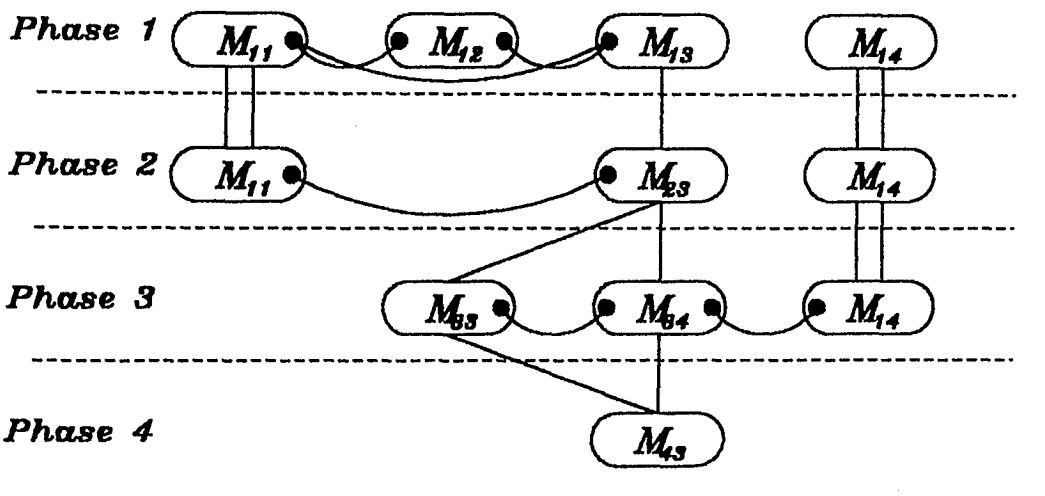


Abb. 28. Kommunikationsgeometrie des Algorithmus aus Beispiel 4.3

c) Kombinatorische Beschreibung

Der parallele Algorithmus wird beschrieben durch die Versuchsplansequenz $(V^{\times}, \mathcal{B}^{\times}, \Phi, 4)$ mit $V^{\times} = (V_1, ..., V_4)$, $\mathcal{B}^{\times} = (\mathcal{B}_1, ..., \mathcal{B}_4)$.

Die Kommunikationsstruktur in den Phasen $i = 1,...,4$ wird durch folgende Versuchspläne beschrieben:

$i = 1$: $(V,\mathscr{B})_1$ mit $V_1 = \{M_{11},M_{12},M_{13},M_{14}\}$; $\mathscr{B}_1 = \{(M_{11},M_{12},M_{13}),(M_{14})\}$
$i = 2$: $(V,\mathscr{B})_2$ mit $V_2 = \{M_{11},M_{23},M_{14}\}$; $\mathscr{B}_2 = \{(M_{11},M_{23}),(M_{14})\}$
$i = 3$: $(V,\mathscr{B})_3$ mit $V_3 = \{M_{33},M_{34},M_{14}\}$; $\mathscr{B}_3 = \{(M_{33},M_{34}),(M_{34},M_{14})\}$
$i = 4$: $(V,\mathscr{B})_4$ mit $V_4 = \{M_{43}\}$; $\mathscr{B}_4 = \{(M_{43})\}$

Die Übergänge zwischen den einzelnen Phasen werden beschrieben durch die Folge der Relationen $\Phi = (\varphi_{12}, ..., \varphi_{34})$ mit:
$\varphi_{12} = \{(M_{13} \mapsto M_{23})\}$
$\varphi_{23} = \{(M_{23} \mapsto M_{33}),(M_{23} \mapsto M_{34})\}$
$\varphi_{34} = \{(M_{33} \mapsto M_{43}),(M_{34} \mapsto M_{43})\}$

Für alle übrigen φ_{ij} $1 \leq i \leq 3$, $2 \leq j \leq 4$ gilt: $\varphi_{ij} = \{\}$

Beispiel 4.4 (Potenzierung nach H. T. Kung)

a) Vorbemerkungen

H. T. Kung präsentierte im Jahre 1974 ein originäres Verfahren zur Berechnung von x^n [Kun,74], welches nur zwei Schritte mit komplexen Multiplikationen und Divisionen und $\log n$ Schritte komplexer Additionen erfordert. Die Berechnung von x^n beruht auf folgender Identität:

$$\frac{n}{x^n - 1} = \sum_{j=1}^{n} \frac{\omega^j}{x - \omega^j}$$

Hierbei ist $\omega = e^{\frac{-2\pi i}{n}}$ eine n-te Einheitswurzel, deren Potenzen bereits durch Vorausberechnung zur Verfügung stehen. Die gesuchte Potenz x^n berechnet man daher wie folgt:

$$x^n = \frac{n}{\sum_{j=1}^{n} \frac{\omega^j}{x - \omega^j}} + 1$$

b) Ermittlung der Zahl der Phasen

- Die Addition $x - \omega^j$ erfolgt in 1 Phase.
- Die Division $\omega^j/(x - \omega^j)$ benötigt 1 Phase.
- Die Summenbildung nach dem Prinzip des rekursiven Doppelns benötigt $\log n$ Phasen.
- Die Division durch die Summe benötigt 1 Phase.
- Die Addition der 1 benötigt 1 Phase.

c) Ermittlung der Zahl der Module

Für die Berechnung der Potenz x^n ergeben sich bei möglichst feiner Granulierung:

- Additions- und Divisionsschritt benötigen je n parallele Module.

- Die Summenbildung startet mit n Modulen, diese Zahl halbiert sich in jedem Schritt bis schließlich 1 Modul übrigbleibt.
- Abschließende Division und Addition benötigen je 1 Modul.

d) Pseudocode (Setzungen: $r := \log n$, $s := n/2$, $t := n/4$)

```
SEQBEGIN
  PARBEGIN                          /* y = x - ω^j, j = 1,...,n
    M11
    M12
    ⋮
    M1n
  PAREND
  PARBEGIN                          /* ω^j/y, j = 1,...,n
    M21
    M22
    ⋮
    M2n
  PAREND
  CHAN com1,...,coms :
  PARBEGIN                          /* Summationsschritt 1
    M31{com1(M32)}                  /* jeweils Addition in benachbarten Modulen
    M32{com1(M31)}
    M33{com2(M34)}
    M34{com2(M33)}
    ⋮
    M3,n-1{coms(M3n)}
    M3n{coms(M3,n-1)}
  PAREND
  CHAN com1,...,comt :
  PARBEGIN                          /* Summationsschritt 2
    M41{com1(M42)}
    M42{com2(M41)}
    M43{com1(M44)}
    M44{com2(M43)}
    ⋮
    M4,s-1{comt-1(M4,s)}
    M4,s{comt(M4,s-1)}
  PAREND
  ⋮
  CHAN com1 :
  PARBEGIN                          /* Summationsschritt (log n)-1
    Mr+1,1{com1(Mr+1,2)}
    Mr+1,2{com1(Mr+1,1)}
  PAREND
  Mr+2,1                            /* Summationsschritt log n
  Mr+3,1                            /* abschließende Division
  Mr+4,1                            /* abschließende Addition
SEQEND
```

e) Graphische Darstellung der Modulbeziehungen

Abb. 29 veranschaulicht die Kommunikation der Module in den einzelnen Phasen.

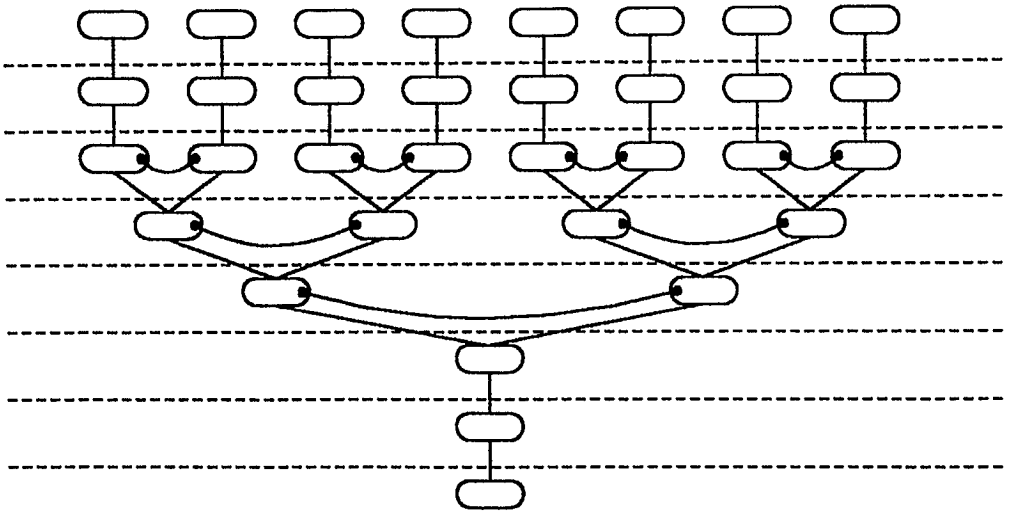


Abb. 29. Kommunikationsstruktur des Algorithmus aus Beispiel 4.4: (für $n = 8$)

f) Kombinatorische Spezifikation der Kommunikationsstruktur

Der parallele Algorithmus wird beschrieben durch die Versuchsplansequenz $(V^{\times}, \mathscr{B}^{\times}, \Phi, r+4)$ mit $V^{\times} = (V_1, ..., V_{r+4})$, $\mathscr{B}^{\times} = (\mathscr{B}_1, ..., \mathscr{B}_{r+4})$.

Die Kommunikationsstruktur in den Phasen $i = 1,...,r+4$ wird durch folgende Versuchspläne beschrieben:

$i = 1$: $(V,\mathscr{B})_1$; $V_1 = \{M_{11}, ..., M_{1n}\}$; $\mathscr{B}_1 = \{(M_{11}), ...,(M_{1n})\}$
$i = 2$: $(V,\mathscr{B})_2$; $V_2 = \{M_{21}, ..., M_{2n}\}$; $\mathscr{B}_2 = \{(M_{21}), ...,(M_{2n})\}$
$i = 3$: $(V,\mathscr{B})_3$; $V_3 = \{M_{31}, ..., M_{3n}\}$; $\mathscr{B}_3 = \{(M_{31}, M_{32}), ...,(M_{3,n-1}, M_{3n})\}$
$i = 4$: $(V,\mathscr{B})_4$; $V_4 = \{M_{41}, ..., M_{4s}\}$; $\mathscr{B}_4 = \{(M_{41}, M_{42}), ...,(M_{4,s-1}, M_{4s})\}$
$\vdots$
$i = r+2$: $(V,\mathscr{B})_{r+2}$; $V_{r+2} = \{M_{r+2,1}, M_{r+2,2}\}$; $\mathscr{B}_{r+2} = \{(M_{r+2,1}, M_{r+2,2})\}$
$i = r+3$: $(V,\mathscr{B})_{r+3}$; $V_{r+3} = \{M_{r+3,1}\}$; $\mathscr{B}_{r+3} = \{(M_{r+3,1})\}$
$i = r+4$: $(V,\mathscr{B})_{r+4}$; $V_{r+4} = \{M_{r+4,1}\}$; $\mathscr{B}_{r+4} = \{(M_{r+4,1})\}$

Die Übergänge zwischen den einzelnen Phasen werden beschrieben durch die Folge der Relationen $\Phi = (\varphi_{12}, ..., \varphi_{r+3,r+4})$ mit:

$\varphi_{12} = \{(M_{11} \mapsto M_{21}),(M_{12} \mapsto M_{22}), ...,(M_{1n} \mapsto M_{2n})\}$
$\varphi_{23} = \{(M_{21} \mapsto M_{31}),(M_{22} \mapsto M_{32}), ...,(M_{2n} \mapsto M_{3n})\}$
$\varphi_{34} = \{(M_{31} \mapsto M_{41}),(M_{32} \mapsto M_{41}),(M_{33} \mapsto M_{42}),(M_{34} \mapsto M_{42}), ...,(M_{3,n-1} \mapsto M_{4s}),(M_{3n} \mapsto M_{4s})\}$
$\vdots$
$\varphi_{r+1,r+2} = \{(M_{r+1,1} \mapsto M_{r+2,1}),(M_{r+1,2} \mapsto M_{r+2,1})\}$
$\varphi_{r+2,r+3} = \{(M_{r+2,1} \mapsto M_{r+3,1})\}$
$\varphi_{r+3,r+4} = \{(M_{r+3,1} \mapsto M_{r+4,1})\}$

Für alle übrigen φ_{ij} $1 \leq i \leq 3$, $2 \leq j \leq 4$ gilt: $\varphi_{ij} = \{\}$

Beispiel 4.5 (Column-Sweep-Algorithmus)

a) Vorbemerkungen

Der Column-Sweep-Algorithmus [H&W,83] dient zur Lösung linearer rekurrenter Systeme und macht von der Funktion des *broadcasting* [J&H,89] Gebrauch. Der Column-Sweep-Algorithmus benötigt $p = O(n)$ Prozessoren und wird relativ ineffizient für rekurrente Systeme $R(n,m)$ mit $m \ll n$.

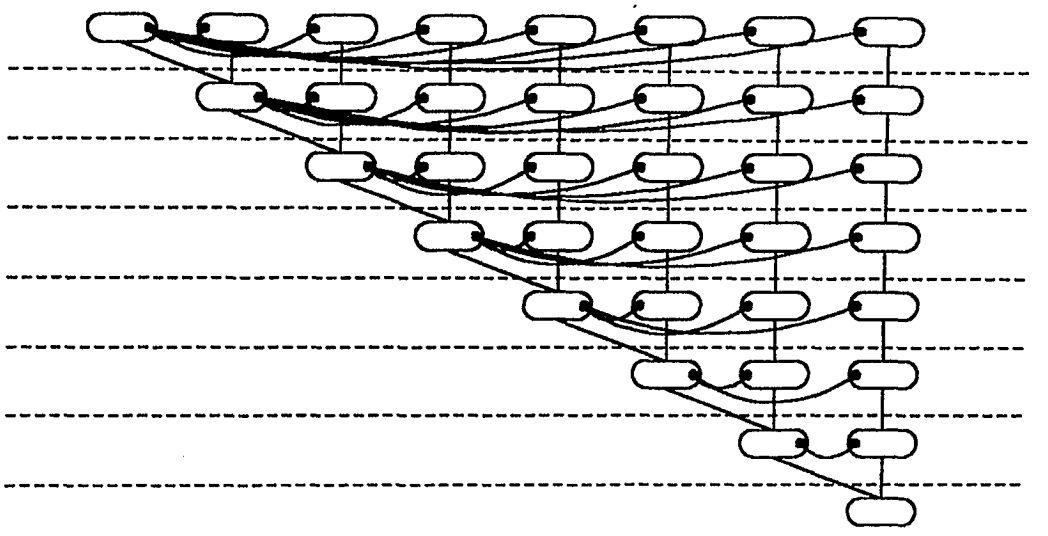


Abb. 30. Kommunikationsstruktur des Algorithmus aus Beispiel 4.5: (für $n = 8$)

b) Ermittlung der Zahl der Phasen

In Pseudonotation hat der Column-Sweep-Algorithmus die Gestalt von Algorithmus 4.

Algorithmus 4: Column-Sweep-Algorithmus

$x_1 = c_1$;
for $k = 1$ **step** 1 **until** $(n-1)$ **do**
  broadcast x_k **to** P_j $(\forall j > k)$;
  multiply $a_{jk} \times x_k$ $(\forall j > k)$;
  add $b_j = (a_{jk} \times x_k) + c_j$ $(\forall j > k)$;
  update $c_j \leftarrow b_j$ $(\forall j > k)$;
  $x_{k+1} \leftarrow b_{k+1}$;

Da sich die Modulstruktur und die Kommunikationstruktur innerhalb eines solchen kombinierten Schleifenschrittes nicht ändern, kann jeder Schleifendurchlauf in einer Phase erfolgen, so daß sich insgesamt $n - 1 + 1 = n$ Phasen ergeben.

c) Ermittlung der Zahl der Module

Die Zahl der Module ist zu Beginn n, sinkt dann in jeder Phase um 1, bis schließlich 1 Modul übrigbleibt.

d) Pseudocode

```
SEQBEGIN
  CHAN com1,...,comn-1 :
  PARBEGIN                                  /* 1. Schleifendurchlauf
    M11{com1(M12),com2(M13),...,comn-1(M1n)}
    M12{com1(M11)}
    M13{com2(M11)}
     :
    M1n{comn-1(M11)}
  PAREND
  CHAN com1,...,comn-2 :
  PARBEGIN                                  /* 2. Schleifendurchlauf
    M21{com1(M22),com2(M23),...,comn-2(M2,n-1)}
    M22{com1(M21)}
    M23{com2(M21)}
     :
    M2,n-1{comn-2(M21)}
  PAREND
  :
  CHAN com1 :
  PARBEGIN                                  /* n-1. Schleifendurchlauf
    Mn-1,1{com1(Mn-1,2)}
    Mn-1,2{com1(Mn-1,1)}
  PAREND
  Mn1
SEQEND
```

e) Graphische Darstellung der Modulbeziehungen

Abb. 30 veranschaulicht die Kommunikation der Module in den einzelnen Phasen.

f) Kombinatorische Spezifikation der Kommunikationsstruktur

Der parallele Algorithmus wird beschrieben durch die Versuchsplansequenz $(V^\times, \mathscr{B}^\times, \Phi, n)$ mit $V^\times = (V_1, ..., V_n)$, $\mathscr{B}^\times = (\mathscr{B}_1, ..., \mathscr{B}_n)$.

Die Kommunikationsstruktur in den Phasen $i = 1,...,n$ wird durch folgende Versuchspläne beschrieben:
$i = 1$: $(V,\mathscr{B})_1$; $V_1 = \{M_{11}, ..., M_{1n}\}$; $\mathscr{B}_1 = \{(M_{11}, M_{12}),(M_{11}, M_{13}), ...,(M_{11}, M_{1n})\}$
$i = 2$: $(V,\mathscr{B})_2$; $V_2 = \{M_{21}, ..., M_{2,n-1}\}$; $\mathscr{B}_2 = \{(M_{21}, M_{22}),(M_{21}, M_{23}), ...,(M_{21}, M_{2,n-1})\}$

$\vdots$

$i = n-1$: $(V,\mathcal{B})_{n-1}$; $V_{n-1} = \{M_{n-1,1}, M_{n-1,2}\}$; $\mathcal{B}_{n-1} = \{(M_{n-1,1}, M_{n-1,2})\}$

$i = n$: $(V,\mathcal{B})_n$; $V_n = \{M_{n1}\}$; $\mathcal{B}_n = \{(M_{n1})\}$

Die Übergänge zwischen den einzelnen Phasen werden beschrieben durch die Folge der Relationen $\Phi = (\varphi_{12}, ..., \varphi_{n-1,n})$ mit:

$\varphi_{12} = \{(M_{11} \mapsto M_{21}), (M_{12} \mapsto M_{21}), (M_{13} \mapsto M_{22}), ..., (M_{1n} \mapsto M_{2,n-1})\}$

$\varphi_{23} = \{(M_{21} \mapsto M_{31}), (M_{22} \mapsto M_{31}), (M_{23} \mapsto M_{32}), ..., (M_{2,n-1} \mapsto M_{3,n-2})\}$

$\vdots$

$\varphi_{n-2,n-1} = \{(M_{n-2,1} \mapsto M_{n-1,1}), \{(M_{n-2,1} \mapsto M_{n-1,1}), (M_{n-2,2} \mapsto M_{n-1,2})$

$\varphi_{n-1,n} = \{(M_{n-1,1} \mapsto M_{n1}), (M_{n-1,2} \mapsto M_{n1})$

Für alle übrigen φ_{ij} $1 \leq i \leq n-1$, $2 \leq j \leq n$ gilt: $\varphi_{ij} = \{\}$

Beispiel 4.6 (Schnelle Fourier-Transformation)

a) Vorbemerkungen

Betrachtet wird eine eindimensionale, komplexe Radix-2-Transformation der Länge $N = 2^n$ [Bur,89]. Die Schnelle Fourier-Transformation (FFT) zerfällt in drei Schritte: Initialisierung, Transformation und Bitreversion. Die Initialisierung bezeichnet die Bereitstellung der Potenzen der n-ten Einheitswurzeln; sodann folgt die eigentliche Transformation und schließlich die Bitreversion, in der die Transformierten derart umgespeichert werden, wie es einer Spiegelung ihrer als Binärzahlen aufzufassenden Indizes entspricht.

b) Ermittlung der Zahl der Phasen

- Die Berechnung der Potenzen der Einheitswurzel erfolgt in 1 Phase.
- Die Transformation erfolgt in $n = \log_2 N$ Phasen.
- Die Bitreversion erfolgt in 1 Phase.

c) Ermittlung der Zahl der Module

Für eine FFT der Länge $N = 2^n$ ergeben sich bei möglichst feiner Granulierung

- in der Initialisierung $N/2$ Module, da die Potenzen $\omega^0, ..., \omega^{N/2-1}$ benötigt werden,
- in der Transformation $N/2$ Module, da die Elementaroperationen, die sogenannten „*butterflies*", die die Berechnungen für zwei korrespondierende Knoten umfassen, aus Effizienzgründen im allgemeinen nicht aufgespalten werden, bzw. bei *in-place*-Berechnung nicht aufgespalten werden können [Bur,89],
- in der Bitreversion N Module für N Spiegelungen.

Da die Granularität der einzelnen Module der Bitreversion in Relation zur Transformation sehr klein ist, kann auch hier vereinfachend von einer Zusammenfassung von je zwei Modulen ausgegangen werden, so daß auch in der Bitreversion eine Partitionierung in N/2 Module vorgenommen wird. Dies ist auch im Hinblick auf prak-

tische Erwägungen vernünftig, da sich im allgemeinen große Dimensionen N und wesentlich kleinere Prozessorzahlen p gegenüberstehen.

d) Pseudocode (Setzungen: $n := \log N$, $r := N/2$, $s := N/4$)

```
SEQBEGIN
  PARBEGIN                                  /* Initialisierung
    M11
    M12
    ⋮
    M1r
  PAREND
  CHAN com1, ...,coms :
  PARBEGIN                                  /* Transformationsschritt 1, Kommunikationsdistanz '1'
    M21{com1(M22)}
    M22{com1(M21)}
    M23{com2(M24)}
    M24{com2(M23)}
    ⋮
    M2,r-1{coms(M2r)}
    M2r{coms(M2,r-1)}
  PAREND
  CHAN com1,...,coms :
  PARBEGIN                                  /* Transformationsschritt 2, Kommunikationsdistanz '2'
    M31{com1(M33)}
    M32{com2(M34)}
    M33{com1(M31)}
    M34{com2(M32)}
    ⋮
    M3,r-1{coms-1(M3,r-3)}
    M3r{coms(M3,r-2)}
  PAREND
  ⋮
  CHAN com1,...,coms :
  PARBEGIN                                  /* Transformationsschritt n, Kommunikationsdistanz 's'
    Mn+1,1{com1(Mn+1,s+1)}
    Mn+1,2{com2(Mn+1,s+2)}
    Mn+1,3{com3(Mn+1,s+3)}
    Mn+1,4{com4(Mn+1,s+4)}
    ⋮
    Mn+1,r-1{coms-1(Mn+1,s-1)}
    Mn+1,r{coms(Mn+1,s)}
  PAREND
  PARBEGIN                                  /* Bitreversion
    Mn+2,1
    Mn+2,2
    ⋮
    Mn+2,r
  PAREND
SEQEND
```

e) Graphische Darstellung der Modulbeziehungen

Abb. 31 veranschaulicht die Kommunikation der Module in den einzelnen Phasen.

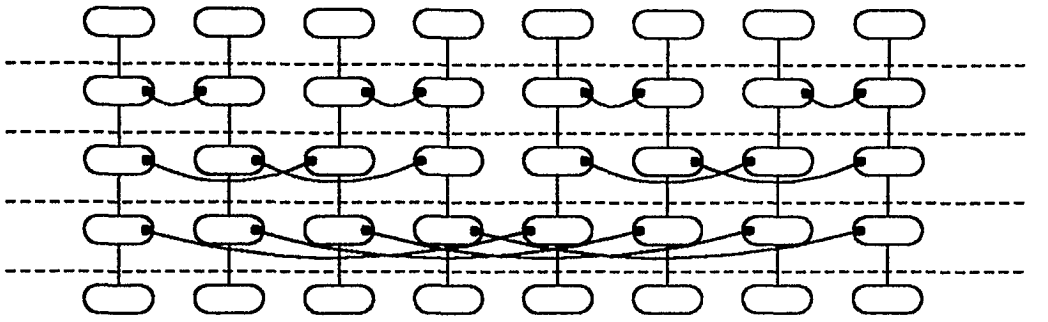


Abb. 31. Kommunikationsstruktur des Algorithmus aus Beispiel 4.6: (für $N = 16$ bei Verwendung von $N/2$ Modulen)

f) Kombinatorische Spezifikation der Kommunikationsstruktur

Der parallele Algorithmus wird beschrieben durch die Versuchsplansequenz $(V^{\times}, \mathscr{B}^{\times}, \Phi, n+2)$ mit $V^{\times} = (V_1, ..., V_{n+2})$, $\mathscr{B}^{\times} = (\mathscr{B}_1, ..., \mathscr{B}_{n+2})$.

Die Kommunikationsstruktur in den Phasen $i = 1,...,n+2$ wird durch folgende Versuchspläne beschrieben:

$i = 1$: $(V,\mathscr{B})_1$; $V_1 = \{M_{11}, ..., M_{1r}\}$; $\mathscr{B}_1 = \{(M_{11}), ...,(M_{1r})\}$

$i = 2$: $(V,\mathscr{B})_2$; $V_2 = \{M_{21}, ..., M_{2r}\}$; $\mathscr{B}_2 = \{(M_{21}, M_{22}),(M_{23}, M_{24}), ...,(M_{2r-1}, M_{2r})\}$

$i = 3$: $(V,\mathscr{B})_3$; $V_3 = \{M_{31}, ..., M_{3r}\}$; $\mathscr{B}_3 = \{(M_{31}, M_{33}),(M_{32}, M_{34}), ...,(M_{3r-2}, M_{3r})\}$

$\vdots$

$i = n+1$: $(V,\mathscr{B})_{n+1}$; $V_{n+1} = \{M_{n+1,1}, ..., M_{n+1,r}\}$;

$\mathscr{B}_{n+1} = \{(M_{n+1,1}, M_{n+1,s+1}),(M_{n+1,2}, M_{n+1,s+2}), ...,(M_{n+1,r}, M_{n+1,r-s})\}$

$i = n+2$: $(V,\mathscr{B})_{n+2}$; $V_{n+2} = \{M_{n+2,1}, ..., M_{n+2,r}\}$; $\mathscr{B}_{n+2} = \{(M_{n+2,1}), ...,(M_{n+2,r})\}$

Die Übergänge zwischen den einzelnen Phasen werden beschrieben durch die Folge der Relationen $\Phi = (\varphi_{12}, ..., \varphi_{n+1,n+2})$ mit:

$\varphi_{12} = \{(M_{11} \mapsto M_{21}),(M_{12} \mapsto M_{22}), ...,(M_{1r} \mapsto M_{2r})\}$

$\vdots$

$\varphi_{n+1,n+2} = \{(M_{n+1,1} \mapsto M_{n+2,1}),(M_{n+1,2} \mapsto M_{n+2,2}), ...,(M_{n+1,r} \mapsto M_{n+2,r})\}$

Für alle übrigen φ_{ij} $1 \leq i \leq n-2$, $1 \leq j \leq n-1$ gilt: $\varphi_{ij} = \{\}$

5 Mapping auf der Basis kombinatorischer Versuchspläne

5.1 Mapping paralleler Algorithmen auf Parallelrechner

Aufgrund der Entwicklung der Rechnerarchitekturen hin zu massiv-parallelen Systemen gewinnt die Kommunikationskomplexität mehr und mehr an Bedeutung, denn eine Reduktion der Berechnungskomplexität unter die Kommunikationskomplexität ist nutzlos [Gen,78]. Es muß daher darauf geachtet werden, auch die Kommunikation in vertretbarer Zeit zu bewerkstelligen. Deshalb wird versucht, Botschaften bzw. Daten möglichst effizient, d. h. auf kurzen Wegen und sparsam, zwischen den Rechnerknoten zu versenden. Da jedoch eine vollständige Vernetzung aller Prozessorelemente in einem Netzwerk bereits bei Systemen mittlerer Parallelität technologische Probleme und zu hohe Kosten verursacht, bestehen in der Regel nur zwischen einem geringen Teil der Prozessoren direkte Verbindungen. Eine naheliegende Optimierungstechnik besteht deshalb darin, die Kommunikationsstruktur des zu parallelisierenden Algorithmus zu analysieren und die einzelnen Daten- und Programmodule so auf das Rechnersystem abzubilden, daß miteinander kommunizierende Module möglichst auf benachbarten, d. h. direkt verbundenen, Knoten plaziert werden; zumindest sollte eine Minimierung der Kommunikationsdistanzen erreicht werden. Diese immer wichtiger werdende Problematik wird als *Mapping* bezeichnet. Berman und Snyder [B&S,84] spezifizieren zwei unterschiedliche Bestandteile des Mapping-Problems. Wenn die Verbindungsstruktur des parallelen Algorithmus von der Verbindungsstruktur des vorgesehenen parallelen Rechners abweicht, wird dies als **topologische Variation** bezeichnet. Ist – außerdem – die Zahl der vom Algorithmus benötigten Prozesse größer als die Zahl der Prozessoren des Netzwerks, so kommt noch eine **Kardinalitätsvariation** hinzu.

Für eine effiziente Nutzung paralleler Rechnersysteme ist es von entscheidender Bedeutung, daß das Verständnis der Beziehungen zwischen parallelen Algorithmen und parallelen Architekturen weit mehr als bisher zu Bewußtsein gelangt. Die Fakten, die die Zusammenhänge zwischen paralleler Software und Hardware beschreiben, können direkten Einfluß auf den Algorithmenentwurfsprozeß nehmen. Insbesondere kann dieser bei gegebener Zielarchitektur wesentlich beschleunigt werden; andererseits ermöglichen sie den Entwurf algorithmenspezifischer Rechner [Jam,87]. Weiterhin können Leistungsaussagen über ein gegebenes Algorithmen-Architektur-Paar getroffen werden. Mit Hilfe verschiedener Mapping-Methoden kann man eine Bibliothek von Algorithmen-Topologie-Paaren aufstellen; so kann eine einzelne, zu aufwendig erscheinende Mapping-Berechnung durch das Heranziehen eines Bibliothekseintrags geleistet werden. Die vollendetste Form der Abbildung paralleler Algorithmen auf parallele Architekturen ist die automatische Realisierung der Abbildung durch das Betriebssystem oder einen Präprozessor eines rekonfigurierbaren Rechnersystems. Derartige Architekturen sind aber bisher kaum über das Projektstadium hinaus zur Marktreife gelangt (CHiP [Sny,82], PASM [K&&,85], TRAC [DJL,85]).

Da ein optimales Mapping bezüglich seiner Komplexität in die Klasse der NP-vollständigen Probleme fällt [K&B,88], werden im allgemeinen nur heuristische Verfahren entworfen. Krämer und Mühlenbein analysieren Abweichungen zwischen zufälligen und optimalen Lösungen von Mapping-Aufgaben [K&M,89], die naturgemäß für dicht-vernetzte Topologien weniger bedeutend sind. Bei den in der Literatur vorgeschlagenen Mapping-Verfahren werden zum Teil sehr restriktive Einschränkungen vorgenommen; insbesondere wird die Klasse der erlaubten Rechnertopologien sehr eng de-

finiert. Abhängigkeitsbeziehungen zwischen verschiedenen Modulen, die meist als Tasks bezeichnet werden, bleiben häufig außer Betracht. Hinsichtlich der Optimierung werden unterschiedliche Strategien verfolgt und variierende Zielgrößenparameter betrachtet. Zu diesen gehören vor allem die kürzest mögliche Rechenzeit des abzubildenden Algorithmus, die kürzeste Kommunikationszeit oder die bestmögliche Lastverteilung auf dem Zielrechner. Praxisrelevante Beschränkungen werden in der Regel aufgehoben. Kim und Browne [K&B,88] charakterisieren die noch aktuelle Situation derart, daß zwar eine sehr große Zahl von Mapping-Verfahren existiert, diese jedoch ohne Ausnahme starke Einschränkungen vornehmen und daher ein geschlossenes logisches Konzept für eine generelle Behandlung verschiedenartiger Algorithmen und Rechnertopologien bislang nicht vorliegt. Insbesondere fehlt es an integrierten Konzepten, also der formalen Gleichbehandlung von paralleler Software und paralleler Hardware.

Setzt man die Präferenz darauf, die Hardware optimal auszunutzen, so muß man, wie Valiant [Val,90b], zu dem Schluß kommen, daß durch eine größere Zahl von Prozessen als Prozessoren noch Spielraum (*slackness*) zur flexiblen Behandlung gegeben ist. Will man jedoch den Algorithmus in möglichst geringer Zeit, d. h. in der Regel möglichst parallel, abarbeiten, so ist dies nur dann theoretisch möglich, wenn die Zahl der Prozessoren die Zahl der parallel ausführbaren Prozesse egalisiert, besser noch übersteigt.

Da ein großer Teil der Algorithmen durch nur einige wenige Paradigmen abgedeckt wird, kann ebenso für die Implementierung auf nur einige wenige Architekturen zurückgegriffen werden [Pre,84]. Preparata bemerkt weiterhin, daß gerade diejenigen Architekturen, die auf einfachen geometrischen Prinzipien basieren, für die Implementierung im allgemeinen die effizientesten sind. Systolische (und) VLSI-Architekturen sind die bei weitem am häufigsten ins Auge gefaßten Zielobjekte für das Mapping. Als Beispiele seien hier die Arbeiten von Preparata [Pre,84] sowie Fortes und Moldovan [F&M,85] genannt. Gründe hierfür sind die detailliert und hochspeziell definierten Hardware-Eigenschaften, die erste Anhaltspunkte liefern und somit die Zuordnung erleichtern.

Neben dem Charakterisierungsansatz von Jamieson existieren vor allem sprachenbasierte und graphenbasierte Verfahren. Die Programmiersprache stellt ein wichtiges Bindeglied zwischen Algorithmus und Architektur dar. Hier kommt es darauf an, daß die für den Algorithmus wichtigen Konstrukte in der Sprache effizient realisiert sind und daß die Kontrollstrukturen der Sprache auf der Architektur effizient implementiert sind [Jam,86]. Die Mittlerfunktion der Programmiersprache wird beispielsweise von Chen [Che,88] für das Mapping paralleler Algorithmen benutzt; hierbei handelt es sich notwendigerweise um eine architekturunabhängige Programmiersprache auf sehr hohem Abstraktionsniveau. Die Stärke graphischer Methoden liegt darin, formale Gesetzmäßigkeiten graphischer Transformationen und geometrischer Darstellungen auf verschiedene Teilaspekte des Mapping-Problems anzuwenden. Der graphentheoretische Mapping-Ansatz ist in gewissem Sinne allgemeiner als der Charakterisierungsansatz, da er nicht die Existenz und das Erkennen spezifischer Muster in Algorithmen verlangt. Neuere Verfahren sind teilweise nicht auf SIMD oder MIMD festgelegt [BKS,91] und nutzen die Technik des *simulated annealing* ([H&X,90], [M&K,91]).

5.2 Ausgewählte Beispiele vorgeschlagener Mapping-Strategien

5.2.1 Verfahren von Jamieson

Basierend auf den in Kapitel 4 vorgestellten Eigenschaften zur Charakterisierung paralleler Algorithmen und einer komplementären Menge von Eigenschaften zur Klassifizierung von parallelen Architekturen beschreibt Jamieson ein Verfahren für das Mapping paralleler Algorithmen auf eine rekonfigurierbare Architektur [Jam,87]. Der Grundgedanke dieses Verfahrens besteht in der Erkenntnis, daß viele Algorithmen eine identifizierbare Struktur besitzen, seien es ähnliche Kommunikationsmuster, gleiche Datenstrukturen oder gleiche Rechentechniken. Derart verwandte Algorithmen können oft aus gleichen Grundbausteinen aufgebaut werden.

Aufsetzend auf dem zu implementierenden Problem wird als Zwischenschritt ein sogenannter *virtueller Algorithmus* erzeugt, der den Punkt im Entwurfszyklus beschreibt, an dem entschieden werden muß, welche Berechnungsschritte durchzuführen sind. Mit Hilfe der Charakterisierungen von parallelen Algorithmen und Rechnern kann man dann den Übergang vom virtuellen Algorithmus zum (benötigten) architekturabhängigen Algorithmus bestimmen. Ideal wäre sicherlich eine eindeutige Vorschrift, die exakt festlegt, bei welchen Algorithmeneigenschaften welche Hardwareeigenschaften zu fordern sind. Angesichts der Komplexität der Beziehungen ist dies jedoch unrealistisch. Jamieson liefert stattdessen eine Übersicht über die Zusammenhänge die, basierend auf ca. 50 Algorithmen aus verschiedenen Bereichen, durch eine dreistufige Gewichtung die mehr oder minder starken Relationen zwischen Algorithmen- und Architekturcharakteristiken bewertet (vgl. Tab. 2 auf Seite 88).

5.2.2 Graphenbasierte Verfahren

In den graphenbasierten Verfahren werden Graphentransformationen benutzt, um die graphische Darstellung der Algorithmen auf die graphische Darstellung der Rechnertopologie abzubilden.

Das Verfahren von Kung, Lewis und Jean [KLJ,87] unterteilt das Mapping in drei Phasen. Zunächst wird ein lokal-rekursiver Algorithmus (LRA) entwickelt, der die Berechnungen darstellt. Die Datenabhängigkeiten zwischen den indizierten Variablen werden mit einem lokalisierten Datenabhängigkeitsgraphen beschrieben. In einem zweiten Schritt werden die Indexpunkte des LRA und ihre assoziierten Variablen auf eine abstrakte Feldarchitektur abgebildet, die mittels eines Signalflußgraphen (SFG) modelliert wird. In der dritten Phase erfolgt schließlich die Implementierung des SFG auf einem realen Rechner, wobei insbesondere systolische Architekturen von Interesse sind, da nur Algorithmen mit einem Mindestmaß an Regularität betrachtet werden, die sich in rekursiver Form darstellen lassen.

Nach Berman und Snyder [B&S,84] kann man einen parallelen Algorithmus als eine Familie von Graphen auffassen oder darstellen, indem man einen speziellen Graphen für jede Problemgröße heranzieht. Für das Mapping wird der zu implementierende Algorithmus dann zunächst auf den kleinsten Graphen der entsprechenden Familie

kontrahiert. Dieser wird dann auf die Hardware abgebildet, wobei die Zahl der Prozesse kleiner sein sollte als die Zahl der Prozessoren. Schließlich wird der wirkliche Algorithmus via „Multiplexing" auf der Hardware implementiert. Auf diese Weise werden zunächst Kardinalitätsdifferenzen und sodann Topologiedifferenzen ausgeglichen.

Algorithmencharakteristiken	*Architekturcharakteristiken*										
	Anzahl der Prozessoren	**Speicherorganisation**	**Speichergröße**	**Modus (SIMD oder MIMD)**	**Netzwerk**	**Synchronisation**	**Prozessorfähigkeiten**	**Datentypen**	**Adressierungsarten**	**Datenstrukturen**	**Ein-/Ausgabe**
Art des Parallelismus	1	2	1	3	■	1	■	■	■	■	■
Datengranularität	■	3	2	3	1	■	■	■	■	2	■
Modulgranularität	■	3	■	3	3	3	■	■	■	■	■
Grad des Parallelismus	3	2	■	2	■	■	■	■	■	■	■
Speicherbedarf	2	2	3	■	■	■	■	■	■	■	■
Uniformität	■	2	■	3	1	■	■	■	■	■	■
Synchronisation	■	2	■	3	2	3	■	■	■	■	■
Statik/Dynamik	■	3	■	3	2	1	■	■	■	■	■
Datenabhängigkeiten	■	2	■	1	3	■	■	■	■	■	■
Grundlegende Operationen	■	2	■	■	2	■	3	■	■	■	■
Datentypen	■	■	■	■	1	■	■	3	2	■	3
Datenstrukturen	■	2	■	1	2	■	■	■	1	3	■
Ein-/Ausgabe	■	1	1	2	2	■	■	■	■	■	3

Tab. 2. Zusammenhang zwischen parallelen Algorithmen und Architekturen nach Jamieson

Shen und Tsai, die ebenfalls ein graphenbasiertes Verfahren entwickelt haben [S&T,85], beschreiben die Hauptschwierigkeit beim Mapping folgendermaßen: Der Durchsatz kann in einem verteilten Rechner nur erhöht werden, wenn die Auslastung maximal balanciert wird, während gleichzeitig die Interprozessorkommunikation minimiert wird. Während die Minimierung der Interprozessorkommunikation nahelegt, die Berechnungen auf einem einzigen Prozessor durchzuführen, zwingt die Maximierung der Lastbalancierung zu einer rigorosen Verteilung der Arbeitslast unter den Prozessoren. Daher muß hinsichtlich dieser gegenläufigen Tendenzen ein Kompromiß gefunden werden. Das schließlich durchgeführte Mapping kann durch eine Matrix dargestellt werden.

Chiang und Fu [C&F,83] führen zur Spezifikation paralleler Algorithmen und paralleler Rechner sogenannte attributierte gerichtete Graphen (*attributed directed graphs*, ADG) ein. Durch die einheitliche Beschreibungsform für parallele Hardware und parallele

Software wird dann das Mapping ermöglicht. Das ADG-Modell stellt eine gewisse Erweiterung zum Petri-Netz-Modell dar und verfügt über zwei Arten von Knoten, Operationsknoten und Datenknoten. Diese entsprechen den Transitionen und Stellen in Petri-Netzen. Zusätzlich erhalten die Knoten Attribute bezüglich wesentlicher Anforderungen. Wie die übrigen hier beschriebenen Graphen-Verfahren kann auch das ADG-Modell von Chiang und Fu im allgemeinen nur Hinweise und Richtlinien vermitteln; direkte, eindeutige Algorithmen-Architekturzuordnungen sind in der Regel nicht zu erwarten.

Das Verfahren von Kim und Browne [K&B,88] basiert auf der Abbildung einer graphischen Darstellung einer parallelen Berechnungsstruktur ([Bro,85]) auf die graphische Darstellung einer Multiprozessorarchitektur. Im Wege des Abbildungsvorgangs können ein oder mehrere sogenannte virtuelle Architekturgraphen als Zwischenschritte erzeugt werden. Ziel ist die Minimierung der Gesamtausführungszeit der parallelen Berechnung. Die Vorgehensweise beruht auf **linearem Clustering**, dessen grundlegende Idee darin besteht, sequentiell voneinander abhängige Module einem einzigen Prozessor zuzuweisen, während wechselseitig unabhängige Module mehreren Prozessoren zugewiesen werden. Lineare Cluster werden im Hinblick auf ihre Gesamtausführungszeit ausgewählt mit einem Prozessor für jeden Knoten des Graphen und einem getrennten Verbindungskanal für jede Kante des Graphen. Mit Hilfe des linearen Clustering wird ein Berechnungsgraph in einen **virtuellen Architekturgraphen (VAG)** überführt. Der VAG repräsentiert eine optimale Multiprozessorarchitektur für den Berechnungsgraphen. Die optimale Architektur stellt einen Prozessor für jeden linearen Cluster zur Verfügung, so daß voneinander unabhängige Tasks, die verschiedenen linearen Clustern angehören, so parallel als möglich ausgeführt werden können. Darüber hinaus besitzt die optimale Architektur direkte Kommunikationskanäle für adjazente Cluster. Ein VAG kann in einen anderen VAG entwickelt werden, indem zwei oder mehr Cluster in einem Cluster zusammengefaßt werden, wenn dies sinnvoll ist. Nachdem der optimale VAG erzeugt wurde, erfolgt die Abbildung dieses VAG auf den **physikalischen Architekturgraphen (PAG)**, der die Zielarchitektur repräsentiert. Dieses abschließende Mapping wird als **physikalisches Mapping** bezeichnet, wobei Kim und Browne sowohl für homogene als auch für heterogene Architekturen Mapping-Algorithmen entwickelt haben. Das Schlüsselproblem ist die Reduktion der Mapping-Komplexität, ohne wesentliche Abstriche bei der Optimalität hinnehmen zu müssen. Ein ähnliches Verfahren wird von Yang, Bic und Nicolau vorgeschlagen [YBN,91].

Das Verfahren von Lin und Moldovan [L&M,85] basiert auf früheren Verfahren von Moldovan, die als Zielrechnertopologien ausschließlich systolische Architekturen vorsehen. Nunmehr sind Feldrechner als Implementierungshardware vorgesehen, die im Gegensatz zu systolischer Hardware über eine höhere Kapazität an lokalem Speicher verfügen und deren Verbindungsstruktur weniger restringiert ist, so daß sie im Hinblick auf die Eignung für verschiedene Algorithmen wesentlich flexibler sind. Ein Algorithmus wird beschrieben durch die Datenabhängigkeitsmatrix D, die Indexmenge I^n und die Menge der Berechnungen. Die Verbindungsnetzwerke werden durch P-Matrizen charakterisiert. Für einen gegebenen Algorithmus findet man dann basierend auf den Matrizen D und P die Transformation T. Der Π-Vektor von T bestimmt die Rechenzeit des Algorithmus. Die Matrix S bestimmt die Kommunikationszeit und die Zahl der erforderlichen Prozessoren. Durch Kombination verschiedener Π und S kann man die gewünschte Transformation auswählen. Je nach Zielvorgabe kann man entweder die Be-

rechnungszeit, die Kommunikationszeit oder die Zahl der einzusetzenden Prozessoren minimieren. Das Mapping-Verfahren besteht aus zwei Schritten, wobei im ersten Schritt eine Abbildung in der Art von Abbildungen auf VLSI-Felder erfolgt. Mit einer zweiten Abbildung wird versucht, den Kommunikationsaufwand des Zwischenergebnisses aus Schritt 1 zu senken. Die Zahl der vorhandenen Prozessoren wird zunächst nicht berücksichtigt. Man bestimmt die Datenabhängigkeiten, stellt sie als Vektoren dar und faßt sie in der Abhängigkeitsmatrix zusammen. Um die höhere Kapazität der Feldrechner an lokalem Speicher gegenüber den systolischen Systemen auszunutzen, wird die Abhängigkeitsmatrix erneut in Augenschein genommen und entweder zeilenweise oder spaltenweise vorgehend nachoptimiert. Das Verfahren von Lin und Moldovan ist insbesondere für das Mapping von Algorithmen mit geschachtelten Schleifen geeignet und kann auch für andere Zielrechner modifiziert werden.

Bokhari [Bok,81] beschreibt einen parallelen Algorithmus durch einen Graphen $G_p = (V_p, E_p)$, dessen Knoten die Module des Algorithmus repräsentieren (vgl. Abschnitt 3.2); die parallele Hardware wird durch einen Graphen $G_a = (V_a, E_a)$ beschrieben, wobei die Knoten des Graphen die Prozessorknoten des Rechnersystems widerspiegeln. Bokhari nimmt eine starke Restriktion dahingehend vor, daß die Mächtigkeit von V_p geringer sein muß als die Mächtigkeit von V_a, eine Festlegung, die nur für zukünftige höchstparallele Rechner als sinnvoll angesehen werden kann. Das Mapping von Algorithmenmodulen auf Prozessoren kann dann durch die Funktion f_m: $V_p \mapsto V_a$ bezeichnet werden. Die Güte des Mapping kann beurteilt werden durch die Zahl der Algorithmenkanten, die durch die Funktion auf Netzwerkkanten abgebildet werden. Als Algorithmenkante wird eine Kommunikation zwischen parallel auszuführenden Modulen angesprochen. Die Zahl der deckungsgleichen Kanten, also die Mapping-Güte, belegt Bokhari mit dem Begriff **Kardinalität**; die formale Bezeichnung ist $|f_m|$. Die Kardinalität kann mathematisch ermittelt werden aus der Gleichung:

$$|f_m| = \frac{1}{2} \sum_{x,y \in V_p} G_p(x,y)\, G_a(f_m(x), f_m(y))$$

Hierbei ist (vgl. Abschnitt 3.2) $G_p(x,y) = 1$, wenn die Module x und y miteinander kommunizieren, also im Problemgraphen zwischen den entsprechenden Knoten eine Kante existiert. $f_m(x)$ bzw. $f_m(y)$ bezeichnen die Prozessoren, auf die die Module x beziehungsweise y abgebildet werden. Der Ausdruck $G_a(f_m(x), f_m(y))$ wird daher nur gleich 1, wenn die Prozessoren, auf die x und y abgebildet werden, verbunden sind. Ein Summenglied wird daher nur gleich 1, falls kommunizierende Module auf direkt verbundene Netzwerkknoten fallen. Da bei der Summation über alle $x,y \in V_p$ jede Netzwerkkante zweifach gezählt wird, muß anschließend mit 1/2 multipliziert werden. Um das bestmögliche Mapping zu erreichen, muß aus den $(|V_p|)!$ Möglichkeiten zur Konstruktion von f_m diejenige (oder eine) mit maximaler Kardinalität ausgewählt werden. Bokhari weist nach, daß das so formulierte Mapping-Problem dem Graphenisomorphieproblem entspricht, insbesondere auch hinsichtlich der Berechnungskomplexität. Dies bringt es mit sich, daß eine Lösung in polynomialer Zeit nur für hinreichend eingeschränkte Fälle gefunden werden kann, nicht jedoch für den allgemeinen Fall. Weiterhin verweist Bokhari auf die Verwandschaft des Mapping mit der Bandbreitenreduktion bei dünnbesetzten quadratischen Matrizen und dem in industriellen Anwendungen häufig auftretenden quadratischen Zuordnungsproblem.

5.3 Kombinatorische Verfahren des Mapping

5.3.1 Konzeption und formale Syntax

Entsprechend den in Kapitel 2 getroffenen Restriktionen werden in dieser Arbeit nur verteilte Rechner mit homogenem Aufbau betrachtet. Heterogene Rechnersysteme mit unterschiedlichen Prozessortypen spielen auch in der Praxis nur eine untergeordnete Rolle. Dies gilt insbesondere bei den für die hier untersuchte Problemstellung des Mapping interessantesten Architekturen, den massiv-parallelen Rechnern. Für die hier vorgenommenen Untersuchungen ist es daher vom Standpunkt der Berechnungskomplexität ohne Bedeutung, auf welchem Prozessor ein bestimmtes Modul zur Ausführung kommt, da alle Prozessoren die gleiche Leistungsfähigkeit besitzen. Aufgrund der ebenfalls im Hinblick auf massiv-parallele Systeme hier zumeist angestrebten feinen Granulierung der parallelen Algorithmen ist in der Regel nur von einer geringen Granularitätsdifferenz verschiedener Module einer Phase auszugehen. Falls die Module einer Phase algorithmisch bedingt in ihrer Granularität dennoch stark voneinander abweichen, so ist dies durch den Benutzer durch Modulaufteilung und Konstruktion der Kommunikationsgeometrie zu berücksichtigen. Bei einer ausreichenden Zahl von Prozessoren können dann einzelne Module aktiv bleiben, während schon Module einer späteren Phase, vor der keine Präzedenzen mehr bestehen, zugeordnet werden. Daher wird hier ausschließlich eine gute Verteilung der Module und damit gute Auslastung der Prozessoren angestrebt; das Kernziel der Analyse bildet somit die Reduktion der Kommunikationskomplexität.

Im einzelnen sind verschiedene Anwendungsfälle des Mapping möglich:

- Stufe I: Auswahl der geeigneten Topologie/Konfiguration
  - Bestimmung einer optimalen Topologie für einen gegebenen Algorithmus
  - Auswahl der geeigneten Konfiguration für die optimale Implementierung eines parallelen Algorithmus auf einer rekonfigurierbaren Architektur
  - Auswahl einer geeigneten Rechnertopologie bei mehreren zur Verfügung stehenden Zielrechnern
  - Wirtschaftlichkeitsbetrachtungen auch im Falle nur eines verfügbaren Rechners
- Stufe II: Bestimmung der Modul-Prozessor-Zuordnungen
  - Beachtung der Kommunikationsgeometrie innerhalb der Phasen
  - Berücksichtigung der durch die Relationenfolge Φ gegebenen phasenübergreifenden Einflüsse

Es ist offensichtlich, daß mit der Stufeneinteilung I, II keine Chronologie postuliert wird, sondern nur eine Hierarchie von Arbeitsniveaus, da, um das Ergebnis in Stufe I zu erhalten, Optimierungsversuche innerhalb von Stufe II erfolgen müssen. Aufgrund der geringen Verfügbarkeit rekonfigurierbarer Rechner für praktische Anwendungen konzentriert sich diese Arbeit auf die Auswahl der besten Rechnertopologie aus einer gegebenen Menge von Architekturen, wiewohl beide Problemstellungen nicht unabhängig voneinander sind. Man kann zunächst aus den in Kapitel 3 klassifizierten Architekturen die optimale bestimmen oder eine reduzierte Gruppe von Rechnertypen ins Auge fassen,

die für eine tatsächliche Implementierung eher in Betracht kommen, sei es nun aus wirtschaftlichen oder technischen Gründen.

Für den Fall, daß die Zahl der Module in einer bestimmten Phase größer ist als die Zahl der zur Verfügung stehenden Prozessoren, kann beispielsweise das in Abschnitt 5.2.2 erwähnte Verfahren von Berman und Snyder herangezogen werden: Zunächst wird das Problem auf eine hinreichend kleine Problemgröße reduziert. Das Mapping erfolgt dann mit dem parallelen Algorithmus für das reduzierte Problem, also im vorliegenden Fall mit dem optimalen Versuchsplan, der das reduzierte Problem repräsentiert. Schließlich wird durch „Faltung" das Implementierungsvorhaben auf die eigentliche Problemgröße transformiert. Hier wird diese Vorgehensweise als für die allgemeine praktische Anwendung zu sophistisch angesehen und deshalb bei zu geringer Prozessorenzahl entweder von einer teilweise sequentiellen Ausführung der Module ausgegangen oder es erfolgt eine Verschmelzung mehrerer Module zu einem neuen Modul analog zum *linearen Clustering* bei Kim und Browne (vgl. Abschnitt 5.2.2).

Die zur formalen Behandlung der zu untersuchenden Mapping-Vorgänge notwendige Syntax ist im Symbol- und Abkürzungsverzeichnis im Anschluß an das Inhaltsverzeichnis zusammengefaßt. Die Booleschen Prädikate $\nabla_A(a_1, ..., a_k)$, $\rightleftharpoons(M_{ij}, M_{ik})$ und $\Theta(P_r, M_{ij})$ repräsentieren folgende Aussagen:

- $\nabla_A(a_1, ..., a_k)$: das k-Tupel $(a_1, ..., a_k)$ bildet im Versuchsplan von A einen Block
- $\rightleftharpoons(M_{ij}, M_{ik})$: in Phase i kommunizieren die Module j und k
- $\Theta(P_r, M_{ij})$: in Phase i enthält Prozessor r Modul j

Die Booleschen Prädikate liefern das Ergebnis 1, wenn die Aussage des Prädikats zutrifft, 0 sonst.

Die **Distanz zweier Prozessoren**, $\Delta(P_r, P_s)$, bezeichnet die minimale Zahl der Knoten, über die ein Senden von P_r nach P_s erfolgen muß. Es gilt:

- $\Delta(P_r, P_r) = 0$
- $\Delta(P_r, P_s) = 1$, falls eine direkte Verbindung zwischen den Knoten P_r und P_s besteht
- $\Delta(P_r, P_s) = \min \{\sum_{k=r}^{s-1} \Delta(P_k, P_{k+1})\}$ sonst

Hierbei seien P_k und P_{k+1} benachbarte, d. h. direkt verbundene, Prozessorelemente. Alle an einen (1) Bus angeschlossenen Prozessoren stehen zueinander in Entfernung 1; ein Bus kann jedoch zu einer Zeit nur von einer Nachricht benutzt werden. Die **Distanz zweier Module**, $\Delta(M_{ij}, M_{ik})$, wird festgelegt durch die Distanz der sie enthaltenden Prozessoren, d.h. $\Delta(M_{ij}, M_{ik}) := \Delta(P_r, P_s) \Leftrightarrow \Theta(P_r, M_{ij}) \wedge \Theta(P_s, M_{ik})$. Die Zuordnungsfunktion ϕ bezieht sich auf alle Phasen eines Algorithmus und kann daher als Funktionenschar $\phi_1 : V_{A1} \rightarrow V_R, ..., \phi_l : V_{Al} \rightarrow V_R$ aufgefaßt werden. Die auch als **Mapping-Funktion** bezeichnete Abbildung ϕ kann ebenso dargestellt werden durch eine Menge erfüllter ($= 1$) Prädikate: $\phi = \{\Theta(P_r, M_{ij})\}$. Ein paralleler Algorithmus A und ein ins Auge gefaßter Zielrechner R werden im folgenden als **Mapping-Paar (A,R)** bezeichnet.

Mit den kombinatorischen Hilfsmitteln aus Anhang A und den kombinatorischen Spezifikationen der Topologien für verteilte Rechner und der parallelen Algorithmen aus den Kapiteln 3 und 4 sind nun beispielsweise folgende Mapping-Strategien denkbar:

- Überdeckungsverfahren
- Funktionales Verfahren
- Strukturorientiertes Verfahren

5.3.2 Überdeckungsverfahren

Seien die Versuchpläne eines parallelen Algorithmus und die der zur Verfügung stehenden Zieltopologien gegeben, so lassen sich leicht die zugehörigen Adjazenzmatrizen (vgl. Anhang A) erstellen, wobei im Falle des Algorithmus für jede Phase eine Adjazenzmatrix zu erstellen ist.

Ein Block (a,b) bewirkt den Eintrag einer 1 in Zeile a, Spalte b sowie in Spalte a, Zeile b der Adjazenzmatrix. Zur Umsetzung eines Blocks der Form $(a_1, ..., a_k)$, der etwa einen Bus mit k angeschlossenen Knoten repräsentiert, müssen die $k(k-1)/2$ Einzelbeziehungen notiert werden. Jegliche Direktverbindung führt zum Eintrag einer 1; einzelne 1-Einträge können dabei durchaus mehrfach begründet sein. Aufgrund der hier vorausgesetzten bidirektionalen Nutzbarkeit ist nur die rechte obere Dreiecksmatrix der Adjazenzmatrix zu beachten. Im Falle des parallelen Algorithmus wird für jede Phase i eine Adjazenzmatrix erzeugt mit der Seitenlänge m_i, also der Zahl der Module in dieser Phase. Eine 1 in der Adjazenzmatrix für Phase i in Zeile r, Spalte s bedeutet dann, daß in Phase i die Module r und s miteinander kommunizieren.

Liegen die Adjazenzmatrizen von Rechner und Algorithmenphasen vor, so kann man ein geeignetes Mapping des parallelen Algorithmus auf eine der Architekturen ermitteln, indem man die Zahl der Überdeckungen der jeweiligen Algorithmenadjazenzmatrizen durch die Rechneradjazenzmatrix maximiert. Überdeckung bedeutet hierbei daß an der Stelle, an der in der Phasenadjazenzmatrix eine 1 eingetragen ist, auch in der Rechneradjazenzmatrix eine 1 eingetragen sein muß. Hierzu werden wiederum nur die rechten oberen Dreiecksmatrizen betrachtet, und die Algorithmenmatrizen innerhalb der Rechnermatrix „verschoben". Überschüssige 1-Einträge in der Rechneradjazenzmatrix können als *Don't Care* betrachtet werden. Ist bei diesem Approximationsvorgang, der auch Zeilen- und Spaltenvertauschungen gestattet, keine vollständige Überdeckung ermittelbar, müssen Näherungslösungen herangezogen werden. Eine vollständige Überdeckung einer Algorithmen(phasen)adjazenzmatrix durch eine Rechneradjazenzmatrix kann aus mehreren Gründen unmöglich sein:

- Der Rechnerversuchsplan kann den Algorithmenversuchsplan für diese Phase nicht emulieren (vgl. Abschnitt A); d.h. es existiert keine mögliche Anordnung der Algorithmenadjazenzmatrix in der Rechneradjazenzmatrix derart, daß die Verbindungen des Rechners der Kommunikationsgeometrie der Module in Phase i des Algorithmus entsprechen.
- Die mögliche Totalüberdeckung ist dadurch verhindert, daß

- der Rechner im Mehrprogrammbetrieb genutzt wird und die noch notwendigen Verbindungen zur Zeit durch andere Benutzer belegt werden.
- die noch notwendigen Verbindungen noch durch Module genutzt werden, die früheren Phasen des gleichen Algorithmus zuzurechnen sind, aber noch aktiv sind.[5]

Diese Punkte spielen bei der Auswahl der Topologie keine Rolle, sondern nur bei der direkten Modul-Prozessor-Zuordnung bei festem Mapping-Paar.

Bei mehreren zur Verfügung stehenden Rechnern wird dann zunächst derjenige ausgewählt, der die größte Zahl an Übereinstimmungen erzielt. Ist dieser beste Zielrechner ermittelt, oder nur ein Rechner gegeben, werden die Modul-Prozessor-Zuordnungen bestimmt.

Zwei einfache Beispiele mögen zur Verdeutlichung der Adjazenzmatrizen von Parallelrechnern genügen:

Beispiel 5.1: Adjazenzmatrix eines Hypercube der Dimension 2

	P_0	P_1	P_2	P_3
P_0	■	1	0	1
P_1	1	■	1	0
P_2	0	1	■	1
P_3	1	0	1	■

Beispiel 5.2: Adjazenzmatrix eines Chordal-Ring-Connected Computer der Stufe 4

	P_0	P_1	P_2	P_3
P_0	■	1	1	1
P_1	1	■	1	1
P_2	1	1	■	1
P_3	1	1	1	■

Als Beispiel für die Adjazenzmatrixdarstellung eines parallelen Algorithmus dient der Algorithmus aus Beispiel 4.2; dieser entspricht folgenden Adjazenzmatrizen:

Beispiel 5.3: Adjazenzmatrizen des Algorithmus aus Beispiel 4.2
Phasen 1 – 6

	M_{11}
M_{11}	■

---

[5] Aufgrund der Anlage des Algorithmenbeschreibungsmodells nach Kapitel 4 ist dieser Fall durchaus möglich. Er kann dann auftreten, wenn alle Präzedenzen vor Phase i abgearbeitet sind, die Module dieser Phase also zuordnungsfähig sind, aber noch Module aus früheren Phasen aktiv sind.

	M_{21}	M_{22}
M_{21}	■	0
M_{22}	0	■

	M_{31}	M_{32}	M_{33}	M_{34}
M_{31}	■	1	0	0
M_{32}	1	■	1	1
M_{33}	0	1	■	1
M_{34}	0	1	1	■

	M_{31}	M_{42}	M_{43}
M_{31}	■	1	0
M_{42}	1	■	1
M_{43}	0	1	■

	M_{53}
M_{53}	■

	M_{61}
M_{61}	■

Ist nun der hier durch Adjazenzmatrizen dargestellte Algorithmus auf einem der Rechner der Beispiele 5.1 und 5.2 zu implementieren, so ist unter Beachtung der vorhandenen Präzedenzen folgendes festzuhalten: Für die Adjazenzmatrix der Phase 3 des Algorithmus wird keine Totalüberdeckung durch die Adjazenzmatrix des Hypercube erzielt, wohl aber durch diejenige des Chordal-Ring-Connected Computer. Da für alle übrigen Phasen Totalüberdeckungen erzielt werden, würde von diesen beiden Rechnern die Wahl auf den CRC fallen. Dies bedeutet nicht, daß der Algorithmus nicht auf dem Hypercube ausführbar wäre, dies ist sehr wohl möglich und sogar in paralleler Weise; der CRC muß hier jedoch als effizienter angesehen werden, da er die Kommunikation der Module M_{32} und M_{34} in Phase 3 dadurch unterstützt, daß die Module auf direkt verbundene Prozessoren abgebildet werden können.

Die zu den Phasen 1, 2, 5, 6 gehörigen Adjazenzmatrizen geben für die Beurteilung mit dem Überdeckungsverfahren keine Anhaltspunkte, da keine Kommunikation zwischen den jeweiligen Modulen besteht. Mögliche Totalüberdeckungen einer Phasenadjazenzmatrix durch eine Rechneradjazenzmatrix liefern die gesuchten Modul-Prozessor-Zuordnungen. Für Phase 3 könnte dies beispielsweise die Zuordnung $\phi = \{\Theta(P_0, M_{31}), \Theta(P_1, M_{32}), \Theta(P_2, M_{33}), \Theta(P_3, M_{34})\}$ sein.

5.3.3 Funktionales Verfahren

In Analogie zur mathematischen Formulierung bei Bokhari [Bok,81], die auf dem Graphen-Isomorphie-Problem basiert, kann man ebenso mit Isomorphismen auf kombinatorischen Versuchsplänen (vgl. Anhang A) Mapping-Optimierungen für die Abbildung paralleler Algorithmen auf parallele Hardware vornehmen.

Sei ein Mapping-Paar (A,R) gegeben, so betrachte dessen Versuchspläne $(V,\mathscr{B})_{A1}, ..., (V,\mathscr{B})_{Al}$ und $(V,\mathscr{B})_R$. Existieren Bijektionen $\phi_1: V_{A1} \rightarrow V_R, ..., \phi_l: V_{Al} \rightarrow V_R$, so daß adjazente Knoten des Versuchsplans (kommunizierende Module) auf adjazente Knoten des Parallelrechners (direkt-verbundene Prozessoren) abgebildet werden, so sind die jeweiligen Phasenversuchspläne des Algorithmus A isomorph zum Versuchsplan des parallelen Rechners R. Die Bilder der Module, die im Versuchsplan einer Algorithmenphase einen Block bilden, bilden im Versuchplan des Rechners dann also wiederum einen Block. Die Bijektionen ϕ_i, die die Isomorphie belegen, liefern dabei direkt die Modul-Prozessor-Zuordnungen. Im Falle einer gegebenen Isomorphie zwischen den Versuchsplänen eines Mapping-Paares ist dann sicherlich ein effizientes Mapping-Paar gefunden.

Eine solche „vollkommene" Isomorphie über alle Phasen eines parallelen Algorithmus wird jedoch in den seltensten Fällen gegeben sein, so daß man sich mit Näherungslösungen zufrieden geben muß. Die Güte der jeweiligen „partiellen Isomorphie" und damit des Mapping läßt sich analog zur Vorgehensweise bei Bokhari durch Summenbildung ermitteln, wobei die Zahl der Blockübereinstimmungen von Algorithmen(phasen)plan und Rechnerversuchsplan bestimmt wird. Das hier vorgestellte Verfahren ist aufgrund der Aufteilung in Phasen jedoch wesentlich elaborierter und damit genauer als dasjenige von Bokhari. Insbesondere muß dem Verfahren von Bokhari vorgehalten werden, daß, wenn man alle Module ohne Phasenunterteilung in einen Topf (Graph, Versuchsplan) wirft, die Vernetzung (Verknüpfungsdichte) so gering wird, daß die Isomorphie nur sehr schwache Anhaltspunkte liefert. Miller [Mil,78] beschreibt ein Verfahren zur Entscheidung der Isomorphie zweier kombinatorischer Versuchspläne der Kardinalität v mit der Zeitkomplexität $O(v^{\log v})$.

Aufbauend auf der Kardinalitätsfunktion von Bokhari, wobei Graphen auf Versuchspläne zu übertragen sind, muß dann noch über die Phasen summiert werden. Die **Bewertungsfunktion** hat dann folgende Gestalt:

$$\|\phi\| := \sum_{l=1}^{l} \sum_{a_l \in V_{Al}} \nabla_{Al}(a_1, ..., a_k)\ \nabla_R(\phi(a_1), ..., \phi(a_k))$$

Die Präzision des hier vorgestellten funktionalen Verfahrens wächst weiter, wenn darüber hinaus auch die Datentransferkosten betrachtet werden. Durch diese Weiterentwicklungen kann man eventuell eine Entscheidung bei sonst gleichwertigen Mapping-Paaren finden, oder gar zu einer gänzlich anderen Entscheidung kommen.

Ohne Berücksichtigung der Tatsache, ob die gewählte Mapping-Funktion ϕ eine optimale ist, werden hier zwei Beispiele angegeben, an denen verdeutlicht werden soll, wie die Isomorphie zur Bewertung von Mapping-Funktionen herangezogen werden kann.

Beispiel 5.4

a) Mapping-Objekte

Zielrechnertopologie:
Pyramidal-Connected Computer (PCC) der Stufe 1 (vgl. 3.2.3.5)

Paralleler Algorithmus:
vgl. Pseudocode

```
SEQBEGIN
  M11
  CHAN com1 :
  PARBEGIN
    M21{com1(M22)}
    M22{com1(M21)}
  PAREND
  CHAN com1,com2 :
  PARBEGIN
    M31{com1(M32)}
    M32{com1(M31)}
    M33{com2(M34)}
    M34{com2(M33)}
  PAREND
  CHAN com1,com2 :
  PARBEGIN
    M41{com1(M42)}
    M42{com1(M41)}
    M43{com2(M44)}
    M44{com2(M43)}
  PAREND
SEQEND
```

b) Kombinatorische Spezifikation des Algorithmus

Der parallele Algorithmus wird beschrieben durch die Versuchsplansequenz $(V^{\times}, \mathscr{B}^{\times}, \Phi, 4)$ mit $V^{\times} = (V_1, ..., V_4)$, $\mathscr{B}^{\times} = (\mathscr{B}_1, ..., \mathscr{B}_4)$.

Die Kommunikationsstruktur in den Phasen $i = 1,...,4$ wird durch folgende Versuchspläne beschrieben:
$i = 1 : (V,\mathscr{B})_1$; $V_1 = \{M_{11}\}$; $\mathscr{B}_1 = \{(M_{11})\}$
$i = 2 : (V,\mathscr{B})_2$; $V_2 = \{M_{21},M_{22}\}$; $\mathscr{B}_2 = \{(M_{21},M_{22})\}$
$i = 3 : (V,\mathscr{B})_3$; $V_3 = \{M_{31},M_{32},M_{33},M_{34}\}$; $\mathscr{B}_3 = \{(M_{31},M_{32}),(M_{33},M_{34})\}$
$i = 4 : (V,\mathscr{B})_4$; $V_4 = \{M_{41},M_{42},M_{43},M_{44}\}$; $\mathscr{B}_4 = \{(M_{41},M_{42}),(M_{43},M_{44})\}$

Die Übergänge zwischen den einzelnen Phasen werden beschrieben durch die Folge der Relationen $\Phi = (\varphi_{12}, ..., \varphi_{34})$ mit:
$\varphi_{12} = \{(M_{11} \mapsto M_{21}),(M_{11} \mapsto M_{22})\}$
$\varphi_{23} = \{(M_{21} \mapsto M_{31}),(M_{21} \mapsto M_{32}),(M_{22} \mapsto M_{33}),(M_{22} \mapsto M_{34})\}$
$\varphi_{34} = \{(M_{32} \mapsto M_{41}),(M_{32} \mapsto M_{42}),(M_{34} \mapsto M_{43}),(M_{34} \mapsto M_{44})\}$

Für alle übrigen φ_{ij} $1 \leq i \leq 3$, $2 \leq j \leq 4$ gilt: $\varphi_{ij} = \{\}$

c) Kombinatorische Spezifikation der Zielrechnertopologie

Die Zielrechnertopologie wird beschrieben durch den Versuchsplan $(V, \mathscr{B})_{PCC-1}$ mit
$V = \{0,1,2,3,4\}$

$\mathcal{B} = \{(0,1),(0,2),(0,3),(0,4),(1,2),(2,3),(3,4),(4,1)\}$
$v = 5, b = 8, r_0 = 4, r_x = 3\ \forall x \in \{1,2,3,4\}$
$\Lambda_2 = \{\lambda(s): |S| = 2, S \subseteq V\} = \{0,1\}$

d) Zu analysierende Abbildungsfunktion ϕ

$\phi|_{t=1}: \{\Theta(P_0, M_{11})\}$
$\phi|_{t=2}: \{\Theta(P_1, M_{21}), \Theta(P_4, M_{22})\}$
$\phi|_{t=3}: \{\Theta(P_1, M_{31}), \Theta(P_2, M_{32}), \Theta(P_3, M_{34}), \Theta(P_4, M_{33})\}$
$\phi|_{t=4}: \{\Theta(P_1, M_{42}), \Theta(P_2, M_{41}), \Theta(P_3, M_{43}), \Theta(P_4, M_{44})\}$
Die Mapping-Funktion ϕ wird in Abb. 32 graphisch veranschaulicht.

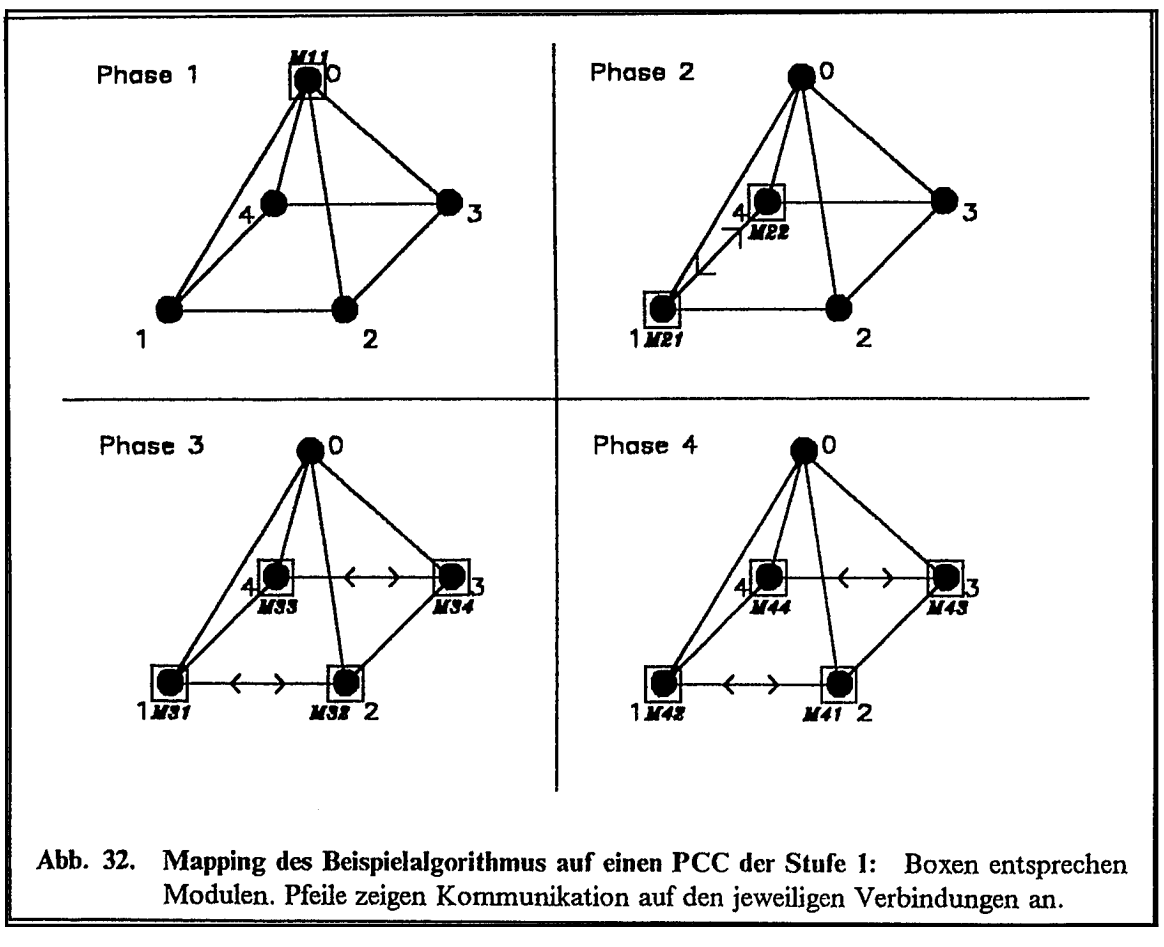


Abb. 32. Mapping des Beispielalgorithmus auf einen PCC der Stufe 1: Boxen entsprechen Modulen. Pfeile zeigen Kommunikation auf den jeweiligen Verbindungen an.

e) Bewertung des Mapping

$$\|\phi\|(A, PCC-1) = \sum_{i=1}^{4} \sum_{a,b \in V_{Ai}} \nabla_{Ai}(a,b)\ \nabla_{PCC-1}(\phi(a), \phi(b))$$
$$= 0 + 1 + 2 + 2 = 5$$

ϕ liefert in diesem Fall also alle 5 möglichen Blockübereinstimmungen und damit die vollständige Isomorphie.

Beispiel 5.5

a) Bezeichnung der Mapping-Objekte

Zielrechnertopologie:
Sternförmiger Rechner (STC) der Stufe $n = 4$ (vgl. 3.2.1.1)

Paralleler Algorithmus:
vgl. Algorithmus aus Beispiel 5.4

b) Kombinatorische Spezifikation des Algorithmus

vgl. Beispiel 5.4

c) Kombinatorische Spezifikation der Zielrechnertopologie

Die Zielrechnertopologie wird beschrieben durch den Versuchsplan $(V,\mathscr{B})_{STC-4}$ mit
$V = \{0,1,2,3,4\}$
$\mathscr{B} = \{(0,1),(0,2),(0,3),(0,4)\}$
$v = 5, b = 4, r_0 = 4$ (für das Zentrum), $r_x = 1\ \forall x \in \{1,2,3,4\}$ (für die peripheren Knoten)
$\Lambda_2 = \{\lambda(s): \mid S \mid = 2, S \subseteq V\} = \{0,1\}$

d) Zu analysierende Abbildungsfunktion ϕ

$\phi\text{-}|_{i=1} : \{\Theta(P_0, M_{11})\}$
$\phi\ |_{i=2} : \{\Theta(P_1, M_{21}), \Theta(P_2, M_{22})\}$
$\phi\ |_{i=3} : \{\Theta(P_1, M_{31}), \Theta(P_2, M_{33}), \Theta(P_3, M_{32}), \Theta(P_4, M_{34})\}$
$\phi\ |_{i=4} : \{\Theta(P_1, M_{42}), \Theta(P_2, M_{44}), \Theta(P_3, M_{41}), \Theta(P_4, M_{43})\}$
Die Mapping-Funktion ϕ wird in Abb. 33 graphisch veranschaulicht.

e) Bewertung des Mapping

$$\|\phi\|(A,STC-4) = \sum_{i=1}^{4} \sum_{a,b \in V_{Ai}} \nabla_{Ai}(a,b)\ \nabla_{STC-4}(\phi(a), \phi(b))$$
$$= 0 + 0 + 0 + 0 = 0$$

ϕ liefert in diesem Fall also keine der 5 möglichen Blockübereinstimmungen.

Für die Kostenanalyse im Bereich Datentransfer wird auf den Abschnitt 5.3.5 verwiesen.

5.3.4 Strukturorientiertes Verfahren

Entspricht das Kommunikationsverhalten eines bestimmten parallelen Algorithmus in den einzelnen Phasen einer bestimmten Struktur aus dem Bereich der Theorie der kombinatorischen Versuchspläne, etwa einem BIBD, so liegt es nahe, diesen Algorithmus auf eine Rechnertopologie abzubilden, die, entsprechend den Ergebnissen aus Kapitel 3, die

Bedingungen an die gleiche kombinatorische Struktur erfüllt. Ist aufgrund der Struktur einmal die geeignete Topologie gefunden, so kann über die Maximalzahl der Module einer Phase ($\max_{1 \le i \le l} m_i$) die Mindestzahl von Prozessoren dieser Topologie bestimmt und so der erforderliche Parallelrechner gefunden werden.

Im allgemeinen kann nicht davon ausgegangen werden, daß die Versuchspläne aller Phasen eines Algorithmus derselben Versuchsplankategorie angehören. Daher muß auch hier wie in den anderen Varianten des Mapping mit Näherungslösungen gearbeitet werden.

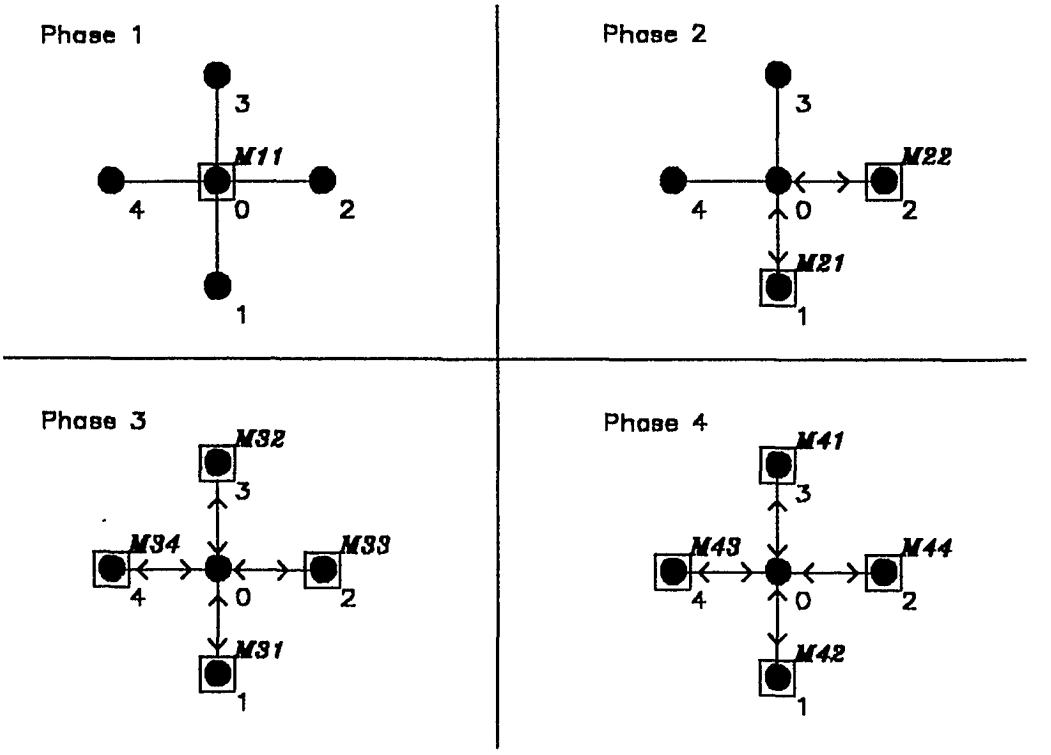


Abb. 33. Mapping des Beispielalgorithmus auf einen STC der Stufe 4: Boxen entsprechen Modulen. Pfeile zeigen Kommunikation auf den jeweiligen Verbindungen an.

Beispiel 5.6

Gegeben sei ein paralleler Algorithmus, der durch folgende Versuchsplansequenz beschrieben wird: $(V^{\times}, \mathscr{B}^{\times}, \Phi, 3)$ mit $V^{\times} = (V_1, V_2, V_3), \mathscr{B}^{\times} = (\mathscr{B}_1, \mathscr{B}_2, \mathscr{B}_3)$

Die Kommunikationsstruktur in den Phasen $i = 1,2,3$ wird durch folgende Versuchspläne beschrieben:

$i = 1$: $(V,\mathscr{B})_1$ mit $V_1 = \{M_{11},M_{12},M_{13}\}$, $\mathscr{B}_1 = \{(M_{11}),(M_{12}),(M_{13})\}$

$i = 2$: $(V,\mathscr{B})_2$ mit $V_2 = \{M_{21},M_{22},M_{23}\}$, $\mathscr{B}_2 = \{(M_{21},M_{22}),(M_{22},M_{23}),(M_{21},M_{23})\}$

$i = 3$: $(V,\mathscr{B})_3$ mit $V_3 = \{M_{31}, \ldots, M_{37}\}$, $\mathscr{B}_3 = \{(M_{31},M_{32},M_{33}),(M_{33},M_{34},M_{35}),(M_{35},M_{36},M_{31}),$
$(M_{31},M_{37},M_{34}),(M_{32},M_{37},M_{35}),(M_{33},M_{37},M_{36}),$
$(M_{32},M_{34},M_{36})\}$

Die Übergänge zwischen den einzelnen Phasen und die damit zusammenhängenden Datentransferkosten mögen zunächst einmal unberücksichtigt bleiben. Es ist dann festzustellen, daß der Versuchsplan für Phase 1 im Hinblick auf ein Mapping sowohl in Stufe I als auch Stufe II nicht berücksichtigt werden muß, da keine Kommunikation zwischen den entsprechenden Modulen stattfindet. Die Versuchspläne der Phasen 2 und 3 sind jedoch wesentlich und entsprechen aufgrund der Blockstruktur ihrer Mengensysteme jeweils der kombinatorischen Struktur eines balancierten, unvollständigen Blockplans (BIBD). Falls nun als Zieltopologien genau die in Kapitel 3 vorgestellten Topologien für das Mapping zur Verfügung stehen würden, so würde die Analyse des Algorithmus nach der strukturorientierten Mapping-Methode nur die Wahl zwischen einem einfachen Bussystem (SBC) und dem vollständig verbundenen Rechner (CIC) lassen, da diese als einzige innerhalb der in Kapitel 3 genannten Strukturen die Anforderungen an einen BIBD erfüllen. In Kapitel 2 wurde jedoch gezeigt, daß noch weitere Topologien denkbar sind, die einen BIBD repräsentieren. Bezüglich einer Entscheidung zwischen CIC und SBC würde im vorliegenden Fall die unterschiedliche Blockgröße in den verschiedenen Phasen eher für den CIC sprechen, da das SBC auf die Mächtigkeit der größten Grundmenge V_3 dimensioniert werden müßte und die Kommunikation auch in den ersten Phasen auf einer einzigen Verbindung bewältigen müßte. Der CIC wiederum ist die wesentlich teurere Architektur und müßte einen Dreierblock aus Phase drei unter Zuhilfenahme von drei Verbindungen simulieren.

Beispiel 5.7

Gegeben sei ein paralleler Algorithmus, der durch folgende Versuchsplansequenz beschrieben wird: $(V^{\times}, \mathscr{B}^{\times}, \Phi, 4)$ mit $V^{\times} = (V_1, ..., V_4)$, $\mathscr{B}^{\times} = (\mathscr{B}_1, ..., \mathscr{B}_4)$

Die Kommunikationsstruktur in den Phasen $i = 1, ...,4$ wird durch folgende Versuchspläne beschrieben:

$i = 1$: $(V,\mathscr{B})_1$ mit $V_1 = \{M_{11},M_{12}\}$, $\mathscr{B}_1 = \{(M_{11}),(M_{12})\}$

$i = 2$: $(V,\mathscr{B})_2$ mit $V_2 = \{M_{21},M_{22},M_{23},M_{24}\}$, $\mathscr{B}_2 = \{(M_{21},M_{22}),(M_{23},M_{24})\}$

$i = 3$: $(V,\mathscr{B})_3$ mit $V_3 = \{M_{31}, ..., M_{38}\}$, $\mathscr{B}_3 = \{(M_{31},M_{32},M_{33},M_{34}),(M_{35},M_{36},M_{37},M_{38})\}$

$i = 4$: $(V,\mathscr{B})_4$ mit $V_4 = \{M_{41}, ..., M_{416}\}$, $\mathscr{B}_4 = \{(M_{31},M_{35},M_{39},M_{313}),(M_{32},M_{36},M_{310},M_{314}),$
$(M_{33},M_{37},M_{311},M_{315}),(M_{34},M_{38},M_{312},M_{316})\}$

Die Übergänge zwischen den einzelnen Phasen und die damit zusammenhängenden Datentransferkosten bleiben zunächst unberücksichtigt. Die Analyse der Phasenversuchspläne des Algorithmus ergibt, daß es sich jeweils um partiell balancierte unvollständige Blockpläne (PBIBD) handelt. Dies reduziert die Menge der geeigneten Topologien, wenn wiederum die gesamten Topologien aus Kapitel 3 als zur Verfügung stehend vorausgesetzt werden, führt jedoch noch zu keinem eindeutigen Ergebnis. Es müssen also weitere Parameter der einzelnen Versuchspläne herangezogen werden.

Die Veränderung bzw. Zunahme der Modulzahl von Phase zu Phase ist polynomialer Natur, womit die geeignete Topologie unter den Polynomialverdichtungen zu suchen ist. Darüber hinaus liefert der Versuchsplan von Phase 4, daß der Zielrechner über mindestens 16 Prozessoren verfügen muß, will man den Algorithmus maximal parallel ausführen. Die Blockgrößen der Versuchspläne der Phasen 3 und 4 erzwingen die

Mindestblockgröße 4 im Rechnerversuchsplan; d.h. es ist ein Bussystem erforderlich mit Mindestbusgröße 4. Die bisher aufgestellten Forderungen spezifizieren nun den Zielrechner so weitgehend, daß innerhalb der zur Verfügung stehenden Rechnermenge (vgl. Kapitel 3) eine eindeutige Mapping-Zuordnung möglich ist. Die Analyse der Versuchsplansequenz des gegebenen Algorithmus liefert das Mapping-Paar (A,MBC2-4).

Bisher wurde ohne Berücksichtigung der phasenübergreifenden Relationen stets eine eindeutige Algorithmus-Rechner-Zuordnung ermittelt. Das muß jedoch nicht immer der Fall sein und die Relationenfolge kann entscheidend werden. Das weitere Vorgehen erfolgt dann ebenso wie bei den anderen beiden vorgestellten Verfahren anhand einer expliziten Kostenermittlung, die im nächsten Abschnitt beschrieben wird.

5.3.5 Kostenbestimmung einer Implementierung

Ziel des Mapping ist es, wie eingangs erläutert, eine möglichst kostengüngstige Abbildung der Module des Algorithmus auf die Prozessoren des Zielrechners zu erreichen. Kostengünstig ist dabei zum einen im Sinne von Bokhari [Bok,81] zu verstehen, d. h. es wird eine Minimierung der Kommunikationsdistanzen angestrebt. Dabei wird jegliche einmalige, einfache Kommunikation über die gleiche Distanz zwischen zwei Modulen bzw. den sie enthaltenden Prozessoren im Sinne des Kostenmaßes als gleichwertig angesehen. Diese Kosten werden als **Kommunikationskosten** bezeichnet. Wie bei Bokhari [Bok,88] werden Kommunikationskosten zwischen Modulen, die auf ein und denselben Prozessor abgebildet werden, vernachlässigt.

Zum anderen werden hier jedoch auch die Kosten berücksichtigt, die dadurch entstehen, daß ein Modul in Phase $i+1$ Daten eines Moduls aus Phase i übernimmt, um beispielsweise eigene Berechnungen darauf aufzubauen. Diese Kosten werden als **Datentransferkosten** bezeichnet. Werden Vorgänger- und Nachfolgermodul in diesem Sinne durch das Mapping auf den gleichen Prozessor des Zielrechners abgebildet, so sind die Datentransferkosten für diesen konkreten Fall gleich Null. Dies gilt jedoch nur für direkt aufeinanderfolgende Phasen oder wenn der betreffende Prozessor zwischenzeitlich nicht mit einem fremden Modul belegt worden ist.

Die Gesamtkosten ergeben sich daher aus der Summe folgender Bestandteile:

- den Kommunikationskosten innerhalb der einzelnen Phasen
- den Datentransferkosten beim Übergang von einer Phase zu chronologisch darauf folgenden

Hierbei ergeben sich die jeweiligen Kostenarten direkt aus der Distanz der miteinander kommunizierenden oder zueinander transferierenden Module. Einzelkosten addieren sich bei n-facher Kommunikation auf derselben Strecke plausiblerweise auf das n-fache.

Die notwendigen Informationen zur Bestimmung der beiden Kostenanteile werden durch die Versuchspläne des Mapping-Paares (Paralleler Algorithmus, Paralleler Rechner) geliefert. Zur Bestimmung der Kommunikationskosten zieht man die Versuchspläne $(V, \mathcal{B})_i$ der den Algorithmus beschreibenden Versuchsplansequenz heran. In einer be-

stimmten Phase i sind die miteinander kommunizierenden Module durch die Blöcke des Mengensystems $\mathscr{B}_i$ gegeben. Eine, für diese spezielle Phase i, optimale Abbildung der Module auf die Prozessoren bezüglich der Kommunikationskosten ist sicher gefunden, wenn in einem gemeinsamen Block befindliche Module auf direkt miteinander verbundene Prozessoren abgebildet werden. Zur rechnerischen Ermittlung der Kosten dienen folgende Zielfunktionen:

$$\min_{1 \le j,k \le m_i} \quad Z_1 := \sum_{i=1}^{l} (\Delta(M_{ij}, M_{ik}) \mid \rightleftharpoons (M_{ij}, M_{ik}))$$

Z_1 ist die Hauptzielfunktion, die zur Bestimmung des optimalen Mapping herangezogen wird. Mit Hilfe der Zielfunktion Z_1 werden die Kommunikationskosten innerhalb der einzelnen Phasen ermittelt und anschließend über die Phasen summiert. Z_1 dient ebenso zur Ermittlung der idealen Topologie für einen speziellen Algorithmus. Die Spezifikation eines *virtuellen Algorithmus* [Jam,87] ist ein komplementäres Verfahren.

Die Beispiele 5.8 bis 5.13 verdeutlichen, daß mit dieser Zielfunktion Z_1 die Kommunikationskosten wesentlich differenzierter ermittelt werden können, als dies mit Bokharis Bewertungsfunktion und auch mit der hier vorgestellten verfeinerten Isomorphiemethode möglich ist. Daher ist die vorliegende Kostenermittlung, selbst ohne Hinzunahme der darüber hinaus berücksichtigten Datentransferkosten, ein wesentlich brauchbareres Werkzeug bei der Beurteilung der Güte eines Mapping. Während die Isomorphiemethode nur die „direkten Treffer" zählt, werden mit der Zielfunktion Z_1 die tatsächlichen Kommunikationskosten der Implementierung ermittelt.

Die Zielfunktion Z_2 ermöglicht die Bestimmung der Datentransferkosten, die beim Übergang zwischen den verschiedenen Phasen entstehen. Dabei wird hier und auch in den anschließenden Beispielen vereinfachend angenommen, daß genau zwischen den Modulen verschiedener Phasen Datentransfer erfolgt, zwischen denen Präzedenzen bestehen. Da im allgemeinen Präzedenzen dadurch begründet sind, daß auf die Berechnung bestimmter Werte, die weiterverwendet werden sollen, gewartet werden muß, ist diese Setzung nicht praxisfremd.

$$\min_{\substack{0 \le r,s \le p-1 \\ 1 \le j \le m_i \\ 1 \le k \le m_{i+1}}} \quad Z_2 := \sum_{i=1}^{l-1} (\Delta(P_r, P_s) \mid ((M_{ij} \mapsto M_{i+1,k}) \in \varphi_{i,i+1}) \wedge \Theta(P_r, M_{ij}) \wedge \Theta(P_s, M_{i+1,k}))$$

Zur praktischen Unterstützung des gesamten Mapping-Verfahrens kann man die Hilfszielfunktion Z_3 heranziehen, die gewährleisten soll, daß beim Übergang zwischen zwei Phasen, möglichst wenige Umgruppierungen erfolgen müssen.

$$\max_{\substack{0 \le r \le p-1 \\ 1 \le j \le m_i}} \quad Z_3 := \sum_{i=1}^{l-1} (\Theta(P_r, M_{ij}) \mid \Theta(P_r, M_{i-1,j}))$$

Die durch die Hilfszielfunktion Z_3 ermittelten Kosten werden nicht hinzuaddiert, sondern sollen lediglich die Bestimmung des optimalen Mapping erleichtern.

Die Gesamtkosten zur Bestimmung der Güte des Mapping resultieren also aus der Addition der Kosten der beiden Hauptzielfunktionen Z_1 und Z_2:

$C_{map} = C(Z_1) + C(Z_2)$

Beispiel 5.8

a) b) c) d) vgl. Beispiel 5.4

e) Bestimmung der Kosten des Mapping

1) Kosten nach Zielfunktion Z_1:

Dies bedeutet einen Vergleich der Mengensysteme $\mathscr{B}_{PCC-1}$ und $\mathscr{B}_1, ..., \mathscr{B}_4$ und ergibt im einzelnen in Phase:

1. 1 Modul, keine Kommunikation, Kosten 0
2. 2 Module, 1 * Kommunikation, Module auf benachbarten Prozessoren, Kosten 1
3. 4 Module, 2 * Kommunikation, Module auf benachbarten Prozessoren, Kosten 2
4. 4 Module, 2 * Kommunikation, Module auf benachbarten Prozessoren, Kosten 2

$\Rightarrow$ Kosten nach Zielfunktion Z_1: $C(Z_1) = 5$

2) Kosten nach Zielfunktion Z_2:

Dies bedeutet einen Vergleich des Mengensystems $\mathscr{B}_{PCC-1}$ mit der Relationenfolge Φ und im einzelnen beim Übergang

1. von Phase 1 nach Phase 2:
   $\varphi_{12} = \{(M_{11} \mapsto M_{21}), (M_{11} \mapsto M_{22})\}$
   Dies bedeutet 2 Datentransfers.

   Das Mapping bewirkt folgende Zuordnungen:
   $\{\Theta(P_0, M_{11}), \Theta(P_1, M_{21}), \Theta(P_4, M_{22})\}$
   (P_0, P_1) und (P_0, P_4) sind jeweils direkte Nachbarn;
   formal: $\Delta(P_0, P_1) = 1$, $\Delta(P_0, P_4) = 1$
   Damit ergeben sich an Datentransferkosten nach Zielfunktion Z_2: $C_{12}(Z_2) = 2$

2. von Phase 2 nach Phase 3:
   $\varphi_{23} = \{(M_{21} \mapsto M_{31}), (M_{21} \mapsto M_{32}), (M_{22} \mapsto M_{33}), (M_{22} \mapsto M_{34})\}$
   Dies bedeutet 4 Datentransfers.

   Das Mapping bewirkt folgende Zuordnungen:
   $\{\Theta(P_1, M_{21}), \Theta(P_4, M_{22}), \Theta(P_1, M_{31}), \Theta(P_2, M_{32}), \Theta(P_3, M_{34}), \Theta(P_4, M_{33})\}$
   Da die Module M_{21}, M_{31} und M_{22}, M_{33} auf die jeweils gleichen Prozessoren abgebildet werden und die an den übrigen beiden Datentransfers beteiligten Prozessoren direkt benachbart sind, entstehen bei den 4 notwendigen Datentransfers nur Kosten von 2 nach Zielfunktion Z_2: $C_{23}(Z_2) = 2$

3. von Phase 3 nach Phase 4:
   $\varphi_{34} = \{(M_{32} \mapsto M_{41}), (M_{32} \mapsto M_{42}), (M_{34} \mapsto M_{43}), (M_{34} \mapsto M_{44})\}$
   Dies bedeutet 4 Datentransfers.

   Das Mapping bewirkt folgende Zuordnungen:
   $\{\Theta(P_2, M_{32}), \Theta(P_3, M_{34}), \Theta(P_1, M_{42}), \Theta(P_2, M_{41}), \Theta(P_3, M_{43}), \Theta(P_4, M_{44})\}$

   Da die Module M_{32}, M_{41} und M_{34}, M_{43} auf die jeweils gleichen Prozessoren abgebildet werden und die an den übrigen beiden Datentransfers beteiligten Prozessoren direkt benachbart sind, entstehen bei den 4 notwendigen Datentransfers nur Kosten von 2 nach Zielfunktion Z_2: $C_{34}(Z_2) = 2$

Somit entstehen entsprechend Zielfunktion Z_2 Kosten im Ausmaß von:
$C(Z_2) = \sum_{1 \le i \le 3, 2 \le j \le 4} C_{ij}(Z_2) = 2 + 2 + 2 = 6$

3) Gesamtkosten:

An Gesamtkosten ergeben sich: $C_{map} = C(Z_1) + C(Z_2) = 5 + 6 = 11$

Beispiel 5.9

a) b) c) d) vgl. Beispiel 5.5

e) Bestimmung der Kosten des Mapping

1) Kosten nach Zielfunktion Z_1:

Dies bedeutet einen Vergleich der Mengensysteme $\mathscr{B}_{STC}$ und $\mathscr{B}_1, \ldots, \mathscr{B}_4$ und ergibt im einzelnen in Phase:

1. 1 Modul, keine Kommunikation, Kosten 0
2. 2 Module, 1 * Kommunikation, Modulentfernung 2 Kosten 2
3. 4 Module, 2 * Kommunikation, Modulentfernung 2 Kosten 4
4. 4 Module, 2 * Kommunikation, Modulentfernung 2 Kosten 4

$\Rightarrow$ Kosten nach Zielfunktion Z_1: $C(Z_1) = 10$

2) Kosten nach Zielfunktion Z_2:

Dies bedeutet einen Vergleich des Mengensystems $\mathscr{B}_{STC}$ mit der Relationenfolge Φ und im einzelnen beim Übergang

1. von Phase 1 nach Phase 2:
   $\varphi_{12} = \{(M_{11} \mapsto M_{21}), (M_{11} \mapsto M_{22})\}$
   Dies bedeutet 2 Datentransfers.

   Das Mapping bewirkt folgende Zuordnungen:
   $\{\Theta(P_0, M_{11}), \Theta(P_1, M_{21}), \Theta(P_2, M_{22})\}$

(P_0, P_1) und (P_0, P_2) sind jeweils direkte Nachbarn;
formal: $\Delta(P_0, P_1) = 1$, $\Delta(P_0, P_2) = 1$
Damit ergeben sich an Datentransferkosten nach Zielfunktion Z_2: $C_{12}(Z_2) = 2$

2. von Phase 2 nach Phase 3:
$\varphi_{23} = \{(M_{21} \mapsto M_{31}), (M_{21} \mapsto M_{32}), (M_{22} \mapsto M_{33}), (M_{22} \mapsto M_{34})\}$
Dies bedeutet 4 Datentransfers.

   Das Mapping bewirkt folgende Zuordnungen:
   $\{\Theta(P_1, M_{21}), \Theta(P_2, M_{22}), \Theta(P_1, M_{31}), \Theta(P_2, M_{33}), \Theta(P_3, M_{32}), \Theta(P_4, M_{34})\}$
   Da die Module M_{21}, M_{31} und M_{22}, M_{33} auf die jeweils gleichen Prozessoren abgebildet werden entstehen hier keine Datentransferkosten. Die an den übrigen beiden Datentransfers beteiligten Prozessoren haben jeweils die Distanz 2, da alle Wege über den Zentralknoten P_0 führen. Daher entstehen bei den 4 notwendigen Datentransfers Kosten von 2 * 2 = 4 nach Zielfunktion Z_2: $C_{23}(Z_2) = 4$

3. von Phase 3 nach Phase 4:
$\varphi_{34} = \{(M_{32} \mapsto M_{41}), (M_{32} \mapsto M_{42}), (M_{34} \mapsto M_{43}), (M_{34} \mapsto M_{44})\}$
Dies bedeutet 4 Datentransfers.

   Das Mapping bewirkt folgende Zuordnungen:
   $\{\Theta(P_3, M_{32}), \Theta(P_4, M_{34}), \Theta(P_1, M_{42}), \Theta(P_2, M_{44}), \Theta(P_3, M_{41}), \Theta(P_4, M_{43})\}$
   Da die Module M_{32}, M_{41} und M_{34}, M_{43} auf die jeweils gleichen Prozessoren abgebildet werden entstehen hier keine Datentransferkosten. Die an den übrigen beiden Datentransfers beteiligten Prozessoren haben jeweils die Distanz 2, da alle Wege über den Zentralknoten P_0 führen. Daher entstehen bei den 4 notwendigen Datentransfers Kosten von 2 * 2 = 4 nach Zielfunktion Z_2: $C_{34}(Z_2) = 4$

Somit entstehen entsprechend Zielfunktion Z_2 Kosten im Ausmaß von:
$C(Z_2) = \sum_{1 \le i \le 3, 2 \le j \le 4} C_{ij}(Z_2) = 2 + 4 + 4 = 10$

3) Gesamtkosten:

An Gesamtkosten ergeben sich: $C_{map} = C(Z_1) + C(Z_2) = 10 + 10 = 20$

Bewertung der Mapping-Beispiele 5.8 und 5.9

Die Pyramidal-Struktur erweist sich als erheblich günstiger zur Implementierung des Beispielalgorithmus als die sternförmige Struktur; dies kann durch die ermittelten Kosten belegt werden. Es zeigt sich, wie dies schon häufig in der Literatur festgestellt wurde, daß die Überlastung des Zentralknotens, über den alle Wege führen, die Schwäche der sternförmigen Parallelrechnerarchitektur ist. Nur dann, wenn auch der zu implementierende parallele Algorithmus innerhalb der einzelnen Phasen ebenfalls „sternförmig" aufgebaut ist (*master-slave*-Konstellation), wird die sternförmige Topologie zum Vorteil und der Star-Connected Computer somit zu einem geeigneten Zielrechner. Resümierend muß der Star-Connected Computer daher als reiner Spezialrechner für solche Algorithmen angesehen werden.

Beispiel 5.10

a) Bezeichnung der Mapping-Objekte

Zielrechnertopologie:
2D-Mesh-Connected Computer (MCC2) der Stufe $n = 4$ (vgl. 3.2.2.1)

Paralleler Algorithmus:
Algorithmus aus Beispiel 4.5 (Column-Sweep-Algorithmus) mit $n = 4$

b) Kombinatorische Spezifikation des Algorithmus
Der parallele Algorithmus wird beschrieben durch die Versuchsplansequenz $(V^{\times}, \mathscr{B}^{\times}, \Phi, 4)$ mit $V^{\times} = (V_1, ..., V_4)$, $\mathscr{B}^{\times} = (\mathscr{B}_1, ..., \mathscr{B}_4)$.

Die Kommunikationsstruktur in den Phasen $i = 1,...,4$ wird durch folgende Versuchspläne beschrieben:
$i = 1$: $(V,\mathscr{B})_1$; $V_1 = \{M_{11},M_{12},M_{13},M_{14}\}$; $\mathscr{B}_1 = \{(M_{11},M_{12}),(M_{11},M_{13}),(M_{11},M_{14})\}$
$i = 2$: $(V,\mathscr{B})_2$; $V_2 = \{M_{21},M_{22},M_{23}\}$; $\mathscr{B}_2 = \{(M_{21},M_{22}),(M_{21},M_{23})\}$
$i = 3$: $(V,\mathscr{B})_3$; $V_3 = \{M_{31},M_{32}\}$; $\mathscr{B}_3 = \{(M_{31},M_{32})\}$
$i = 4$: $(V,\mathscr{B})_4$; $V_4 = \{M_{41}\}$; $\mathscr{B}_4 = \{(M_{41})\}$

Die Übergänge zwischen den einzelnen Phasen werden beschrieben durch die Folge der Relationen $\Phi = (\varphi_{12}, ..., \varphi_{34})$ mit:
$\varphi_{12} = \{(M_{11}\mapsto M_{21}),(M_{12}\mapsto M_{21}),(M_{13}\mapsto M_{22}),(M_{14}\mapsto M_{23})\}$
$\varphi_{23} = \{(M_{21}\mapsto M_{31}),(M_{22}\mapsto M_{31}),(M_{23}\mapsto M_{32})\}$
$\varphi_{34} = \{(M_{31}\mapsto M_{41}),(M_{32}\mapsto M_{41})\}$

Für alle übrigen φ_{ij} $1 \le i \le 3$, $2 \le j \le 4$ gilt: $\varphi_{ij} = \{\}$

c) Kombinatorische Spezifikation der Zielrechnertopologie

Die Zielrechnertopologie wird beschrieben durch den Versuchsplan $(V, \mathscr{B})_{MCC2-4}$ mit
$V = \{0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15\}$
$\mathscr{B} = \{(0,1),(1,2),(2,3),(3,0),(4,5),(5,6),(6,7),(7,4),(8,9),(9,10),(10,11),(8,11),(12,13),$
$(13,14),(14,15),(15,12),(0,4),(4,8),(8,12),(12,0),(1,5),(5,9),(9,13),(13,1),$
$(2,6),(6,10),(10,14),(14,2),(3,7),(7,11),(11,15),(15,3)\}$
$v = 16$, $b = 32$, $r_x = 4$ $\forall x \in V$
$\Lambda_2 = \{\lambda(s): |S| = 2, S \subseteq V\} = \{0,1\}$

d) Zu analysierende Mapping-Funktion ϕ

Entsprechend der Zahl der Phasen des Algorithmus läuft das Mapping ebenso in 4 Phasen ab:
$i = 1$: $\{\Theta(P_{12}, M_{11}), \Theta(P_{13}, M_{12}), \Theta(P_{14}, M_{13}), \Theta(P_{15}, M_{14})\}$
$i = 2$: $\{\Theta(P_9, M_{21}), \Theta(P_{10}, M_{22}), \Theta(P_{11}, M_{23})\}$
$i = 3$: $\{\Theta(P_6, M_{31}), \Theta(P_7, M_{32})\}$
$i = 4$: $\{\Theta(P_3, M_{41})\}$

e) Bestimmung der Kosten des Mapping

1) Kosten nach Zielfunktion Z_1:

Dies bedeutet einen Vergleich der Mengensysteme $\mathscr{B}_{MCC2-4}$ und $\mathscr{B}_1, \ldots, \mathscr{B}_4$ und ergibt im einzelnen in Phase:

1. 4 Module, 3 * Kommunikation, 2 * Distanz 1, 1 * Distanz 2, Kosten 4
2. 3 Module, 2 * Kommunikation, 1 * Distanz 1, 1 * Distanz 2, Kosten 3
3. 2 Module, 1 * Kommunikation, 1 * Distanz 1, Kosten 1
4. 1 Modul, 0 * Kommunikation, Kosten 0

$\Rightarrow$ Kosten nach Zielfunktion Z_1: $C(Z_1) = 4 + 3 + 1 = 8$

2) Kosten nach Zielfunktion Z_2:

Dies bedeutet einen Vergleich des Mengensystems $\mathscr{B}_{MCC2-4}$ mit der Relationenfolge Φ und im einzelnen beim Übergang

1. von Phase 1 nach Phase 2:
   $\varphi_{12} = \{(M_{11} \mapsto M_{21}), (M_{12} \mapsto M_{21}), (M_{13} \mapsto M_{22}), (M_{14} \mapsto M_{23})\}$
   4 Datentransfers, 1 * Distanz 2, 3 * Distanz 1, Kosten 5

2. von Phase 2 nach Phase 3:
   $\varphi_{23} = \{(M_{21} \mapsto M_{31}), (M_{22} \mapsto M_{31}), (M_{23} \mapsto M_{32})\}$
   3 Datentransfers, 1 * Distanz 2, 2 * Distanz 1, Kosten 4

3. von Phase 3 nach Phase 4:
   $\varphi_{34} = \{(M_{31} \mapsto M_{41}), (M_{32} \mapsto M_{41})\}$
   2 Datentransfers, 1 * Distanz 2, 1 * Distanz 1, Kosten 3

Somit entstehen entsprechend Zielfunktion Z_2 Kosten im Ausmaß von:
$C(Z_2) = \sum_{1 \le i \le 3, 2 \le j \le 4} C_{ij}(Z_2) = 5 + 4 + 3 = 12$

3) Gesamtkosten:

An Gesamtkosten ergeben sich: $C_{map} = C(Z_1) + C(Z_2) = 8 + 12 = 20$

Beispiel 5.11

a) Bezeichnung der Mapping-Objekte

Zielrechnertopologie:
2D-Mesh-Bus-Connected Computer (MBC2) der Stufe $n = 4$ (vgl. 3.2.2.3)

Paralleler Algorithmus:
Algorithmus aus Beispiel 5.10

b) Kombinatorische Spezifikation des Algorithmus

vgl. Beispiel 5.10

c) Kombinatorische Spezifikation der Zielrechnertopologie

Die Zielrechnertopologie wird beschrieben durch den Versuchsplan $(V,\mathscr{B})_{MBC2-4}$
$V = \{0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15\}$
$\mathscr{B} = \{(0,1,2,3),(4,5,6,7),(8,9,10,11),(12,13,14,15),(0,4,8,12),(1,5,9,13),(2,6,10,14),(3,7,11,16)\}$
$v = 16, b = 8, r_x = 2\ \forall x \in V$
$\Lambda_2 = \{\lambda(s): \mid S \mid = 2, S \subseteq V\} = \{0,1\}$

d) Zu analysierende Mapping-Funktion ϕ

vgl. Beispiel 5.10

e) Bestimmung der Kosten des Mapping

1) Kosten nach Zielfunktion Z_1:

Dies bedeutet einen Vergleich der Mengensysteme $\mathscr{B}_{MBC2-4}$ und $\mathscr{B}_1, ..., \mathscr{B}_4$ und ergibt im einzelnen in Phase:

1. 4 Module, 3 * Kommunikation, 3 * Distanz 1, Kosten 3
2. 3 Module, 2 * Kommunikation, 2 * Distanz 1, Kosten 2
3. 2 Module, 1 * Kommunikation, 1 * Distanz 1, Kosten 1
4. 1 Modul, 0 * Kommunikation, Kosten 0

$\Rightarrow$ Kosten nach Zielfunktion Z_1: $C(Z_1) = 3 + 2 + 1 = 6$

2) Kosten nach Zielfunktion Z_2:

Dies bedeutet einen Vergleich des Mengensystems $\mathscr{B}_{MBC2-4}$ mit der Relationenfolge Φ und im einzelnen beim Übergang

1. von Phase 1 nach Phase 2:
   $\varphi_{12} = \{(M_{11} \mapsto M_{21}),(M_{12} \mapsto M_{21}),(M_{13} \mapsto M_{22}),(M_{14} \mapsto M_{23})\}$
   4 Datentransfers, 1 * Distanz 2, 3 * Distanz 1, Kosten 5

2. von Phase 2 nach Phase 3:
   $\varphi_{23} = \{(M_{21} \mapsto M_{31}),(M_{22} \mapsto M_{31}),(M_{23} \mapsto M_{32})\}$
   3 Datentransfers, 1 * Distanz 2, 2 * Distanz 1, Kosten 4

3. von Phase 3 nach Phase 4:
   $\varphi_{34} = \{(M_{31} \mapsto M_{41}),(M_{32} \mapsto M_{41})\}$
   2 Datentransfers, 1 * Distanz 2, 1 * Distanz 1, Kosten 3

Somit entstehen entsprechend Zielfunktion Z_2 Kosten im Ausmaß von:
$C(Z_2) = \sum_{1 \le i \le 3, 2 \le j \le 4} C_{ij}(Z_2) = 5 + 4 + 3 = 12$

3) Gesamtkosten:

An Gesamtkosten ergeben sich: $C_{map} = C(Z_1) + C(Z_2) = 6 + 12 = 18$

Bewertung der Mapping-Beispiele 5.10 und 5.11

Die tendenziellen Unterschiede zwischen den maschenverbundenen Strukturen, die auf Links aufbauen, und denjenigen, die mit Bussen arbeiten, treten bei den hier eingesetzten Vertretern mit kleiner Prozessorzahl noch nicht allzu stark auf. Solange die Busse noch nicht zu stark belastet sind, kann wie im vorliegenden Fall durch die kürzere Kommunikationsdistanz die MBC-Struktur im Vorteil sein. Sobald Busse überlastet werden, steht die MCC-Struktur besser da. Würden der MBC-Struktur Zusatzkosten angerechnet, da ein Bus zu einem Zeitpunkt nur ein Paket transportieren kann, im gezeigten Beispiel wären dies 2 ZE in Phase 1 plus 1 ZE in Phase 2, so würde die MCC-Struktur im Beispiel die günstigere sein. Sieht man andererseits den Bus als *broadcast*-Medium an, das das von einem Knoten ausgesandte Paket in 1 ZE an alle übrigen Knoten des Busses übermitteln kann, so würde die MBC-Topologie noch effizienter arbeiten als errechnet.

Beispiel 5.12

a) Bezeichnung der Mapping-Objekte

Zielrechnertopologie:
Chordal-Ring-Connected Computer (CRC) der Stufe $n = 8$ (vgl. 3.2.1.3)

Paralleler Algorithmus:
FFT der Dimension $N = 2^3 = 8$ (vgl. Beispiel 4.6)

b) Kombinatorische Spezifikation des Algorithmus

Der parallele Algorithmus wird beschrieben durch die Versuchsplansequenz $(V^\times, \mathscr{B}^\times, \Phi, 5)$ mit $V^\times = (V_1, ..., V_5)$, $\mathscr{B}^\times = (\mathscr{B}_1, ..., \mathscr{B}_5)$.

Die Kommunikationsstruktur in den Phasen $i = 1,...,5$ wird durch folgende Versuchspläne beschrieben:
$i = 1$: $(V,\mathscr{B})_1$; $V_1 = \{M_{11}, ..., M_{18}\}$; $\mathscr{B}_1 = \{(M_{11}), ...,(M_{18})\}$
$i = 2$: $(V,\mathscr{B})_2$; $V_2 = \{M_{21}, ..., M_{28}\}$; $\mathscr{B}_2 = \{(M_{21},M_{22}),(M_{23},M_{24}), ...,(M_{27},M_{28})\}$
$i = 3$: $(V,\mathscr{B})_3$; $V_3 = \{M_{31}, ..., M_{38}\}$; $\mathscr{B}_3 = \{(M_{31},M_{33}),(M_{32},M_{34}),(M_{35},M_{37}),(M_{36},M_{38})\}$
$i = 4$: $(V,\mathscr{B})_4$; $V_4 = \{M_{41}, ..., M_{48}\}$; $\mathscr{B}_4 = \{(M_{41},M_{45}),(M_{42},M_{46}),(M_{43},M_{47}),(M_{44},M_{48})\}$
$i = 5$: $(V,\mathscr{B})_5$; $V_5 = \{M_{51}, ..., M_{58}\}$; $\mathscr{B}_5 = \{(M_{51}), ...,(M_{58})\}$

Die Übergänge zwischen den einzelnen Phasen werden beschrieben durch die Folge der Relationen $\Phi = (\varphi_{12}, ..., \varphi_{45})$ mit:
$\varphi_{12} = \{(M_{11}\mapsto M_{21}),(M_{12}\mapsto M_{22}), ...,(M_{18}\mapsto M_{28})\}$
$\vdots$
$\varphi_{45} = \{(M_{41}\mapsto M_{51}),(M_{42}\mapsto M_{52}), ...,(M_{48}\mapsto M_{58})\}$

Für alle übrigen φ_{ij} mit $|j-i|>1$ gilt: $\varphi_{ij}=\{\}$

c) Kombinatorische Spezifikation der Zielrechnertopologie

Die Zielrechnertopologie wird beschrieben durch den Versuchsplan $(V,\mathscr{B})_{CRC-8}$ mit
$V=\{0,1,2,3,4,5,6,7\}$
$\mathscr{B}=\{(0,1),(1,2),(2,3),(3,4),(4,5),(5,6),(6,7),(7,0),(0,4),(1,5),(2,6),(3,7)\}$
v = 8, b = 12, $r_x=3\ \forall x\in V$
$\Lambda_2=\{\lambda(s):|S|=2, S\subseteq V\}=\{0,1\}$

d) Zu analysierende Mapping-Funktion ϕ

Entsprechend der Zahl der Phasen des Algorithmus läuft das Mapping ebenso in 5 Phasen ab:
$i=1$: $\{\Theta(P_0, M_{11}), \Theta(P_1, M_{12}), ..., \Theta(P_7, M_{18})\}$
$\vdots$
$i=5$: $\{\Theta(P_0, M_{51}), \Theta(P_1, M_{52}), ..., \Theta(P_7, M_{58})\}$

e) Bestimmung der Kosten des Mapping

1) Kosten nach Zielfunktion Z_1:

Dies bedeutet einen Vergleich der Mengensysteme $\mathscr{B}_{CRC-8}$ und $\mathscr{B}_1, ..., \mathscr{B}_5$.

Anmerkung: Für die Kommunikation in Phase 4 können die Ringstreben genutzt wer den, so daß trotz der numerischen Differenz der Prozessornummern nur eine tatsächliche Kommunikationsdistanz 1 besteht. Es ergibt sich also in den einzelnen Phasen:

1. 8 Module, 0 * Kommunikation, Kosten 0
2. 8 Module, 4 * Kommunikation, Distanz 1, Kosten 4
3. 8 Module, 4 * Kommunikation, Distanz 2, Kosten 8
4. 8 Module, 4 * Kommunikation, Distanz 1, Kosten 4
5. 8 Module, 0 * Kommunikation, Kosten 0

$\Rightarrow$ Kosten nach Zielfunktion Z_1: $C(Z_1)=16$

2) Kosten nach Zielfunktion Z_2:

Dies bedeutet einen Vergleich des Mengensystems $\mathscr{B}_{CRC-8}$ mit der Relationenfolge Φ. Da stets alle Nachfolgemodule in der gleichen Prozessorposition verbleiben, entstehen keine Kosten aufgrund Zielfunktion Z_2:

$$\Rightarrow C(Z_2)=\sum_{1\le i\le 4,2\le j\le 5} C_{ij}(Z_2)=0+0+0+0+0=0$$

3) Gesamtkosten:

An Gesamtkosten ergeben sich: $C_{map}=C(Z_1)+C(Z_2)=16+0=16$

Beispiel 5.13

a) Bezeichnung der Mapping-Objekte

Zielrechnertopologie:
Hypercube (CCC) der Stufe $n = 3$ (vgl. 3.2.3.1)

Paralleler Algorithmus:
FFT der Dimension $N = 2^3 = 8$ (vgl. Beispiel 4.6)

b) Kombinatorische Spezifikation des Algorithmus

vgl. Beispiel 5.12

c) Kombinatorische Spezifikation der Zielrechnertopologie

Die Zielrechnertopologie wird beschrieben durch den Versuchsplan $(V,\mathscr{B})_{CCC-3}$
$V = \{0,1,2,3,4,5,6,7\}$
$\mathscr{B} = \{(0,1),(1,2),(2,3),(3,0),(0,7),(1,6),(2,5),(3,4),(4,5),(5,6),(6,7),(7,4)\}$
$v = 8$, $b = 12$, $r_x = 3\ \forall x \in V$
$\Lambda_2 = \{\lambda(s): |S| = 2, S \subseteq V\} = \{0,1\}$

d) Zu analysierende Mapping-Funktion ϕ

Wie der Algorithmus zerfällt auch das Mapping in 5 Phasen:
$i = 1$: $\{\Theta(P_0, M_{11}), \Theta(P_1, M_{12}), ..., \Theta(P_7, M_{18})\}$
$\vdots$
$i = 5$: $\{\Theta(P_0, M_{51}), \Theta(P_1, M_{52}), ..., \Theta(P_7, M_{58})\}$

e) Bestimmung der Kosten des Mapping

1) Kosten nach Zielfunktion Z_1:

Dies bedeutet einen Vergleich der Mengensysteme $\mathscr{B}_{CCC-3}$ und $\mathscr{B}_1, ..., \mathscr{B}_5$.

Anmerkung: Für den Abstand der Module zueinander ist zu beachten, daß durch die statische Zuordnung zu bestimmten Prozessoren und die Anordnung der Prozessoren im Hypercube entsprechend dem Gray-Code in den Transformationsphasen der FFT, also hier in den Phasen 2, 3 und 4, die Kommunikationsdistanz jeweils 1 beträgt. Es ergibt sich also in den einzelnen Phasen:

1. 8 Module, 0 * Kommunikation, Kosten 0
2. 8 Module, 4 * Kommunikation, Distanz 1, Kosten 4
3. 8 Module, 4 * Kommunikation, Distanz 1, Kosten 4
4. 8 Module, 4 * Kommunikation, Distanz 1, Kosten 4
5. 8 Module, 0 * Kommunikation, Kosten 0

$\Rightarrow$ Kosten nach Zielfunktion Z_1: $C(Z_1) = 12$

2) Kosten nach Zielfunktion Z_2:

Dies bedeutet einen Vergleich des Mengensystems $\mathscr{B}_{CCC-3}$ mit der Relationenfolge Φ. Da stets alle Nachfolgemodule in der gleichen Prozessorposition verbleiben, entstehen keine Kosten aufgrund Zielfunktion Z_2:

$$\Rightarrow C(Z_2) = \sum_{1 \le i \le 4, 2 \le j \le 5} C_{ij}(Z_2) = 0 + 0 + 0 + 0 + 0 = 0$$

3) Gesamtkosten:

An Gesamtkosten ergeben sich: $C_{map} = C(Z_1) + C(Z_2) = 12 + 0 = 12$

Bewertung der Mapping-Beispiele 5.12 und 5.13

Obwohl der Chordal-Ring-Connected Computer die für die Radix-2-FFT günstige Eigenschaft besitzt, daß für die Kommunikation der Prozessoren P_i und $P_{i+n/2}$ eine besondere Verbindung mit Kosten 1 besteht, kann der CRC die Effizienz des Mapping-Paares (Radix-2-FFT, Hypercube) nicht erreichen. Die Ursache ist die besondere Anordnung der Prozessoren im Hypercube, bei der zwei Prozessoren genau dann miteinander verbunden sind, wenn sich ihre Adressen, als Binärzahlen aufgefaßt, in genau einem Bit unterscheiden. Dies entspricht der Kommunikationsstruktur der Radix-2-FFT in optimaler Weise, da bei entsprechendem Mapping der Module jegliche notwendige Kommunikation nur über Distanz 1 erfolgen muß [Bur,89]. Vergleichbare Effizienz kann – im Bereich statischer Netzwerke – nur von einem CIC erbracht werden[6] , dessen Verbindungskosten für größere Dimensionen aufgrund des quadratischen Aufwands zu hoch werden. Der hier zum Vergleich herangezogene CRC ermöglicht nur in der ersten und in der letzten Transformationsphase der FFT die Kommunikationsdistanz 1. Daher tritt im hier durchgeführten Mapping einer Radix-2-FFT der Dimension 2^3 der Nachteil gegenüber dem Hypercube nur in einer Phase auf. Mit zunehmender Dimension der FFT wächst dieser Nachteil linear an.

5.3.6 Beurteilung kombinatorischer Mapping-Methoden

Es wird deutlich, daß die strukturelle Methode nur für die Mapping-Stufe I, der Auswahl des geeigneten Zielrechners für einen bestimmten Algorithmus, sinnvoll erscheint. Die Überdeckungsmethode eignet sich insbesondere für die Bestimmung der optimalen Zieltopologie durch die beste gefundene Überdeckung. Die explizite Zuweisung der einzelnen Module zu den Prozessoren des Rechners in den verschiedenen Phasen geschieht dann auf funktionalem Wege mit Hilfe der Isomorphismen. Die eigentliche Optimierung, die aus Gründen der praktischen Verwendbarkeit numerisch erfolgen muß, erfolgt dann mittels Zielfunktionen, die man beispielsweise auch im Bereich der linearen Optimierung bzw. des Operations Research findet. Zwar würde auch die alleinige Anwendung des funktionalen Verfahrens zum gleichen Ergebnis führen, durch die Vorschaltung der gröberen Verfahren kann jedoch Zeit eingespart werden, da ungeeignete Architekturen mit

---

6 Ein einfaches Bussystem (SBC) ist für eine größere Anzahl von Prozessoren nicht geeignet.

geringerem Aufwand ausgesondert werden können. Diese Überlegungen führen zu der in Algorithmus 5 zusammengefaßten Vorgehensweise:

Algorithmus 5: Mapping auf der Basis kombinatorischer Versuchspläne

1. gegeben:
   - der zu implementierende parallele Algorithmus, dessen Kommunikationsstruktur für l Phasen als Versuchsplansequenz der Länge l vorliegt
   - die zur Implementierung zur Verfügung stehenden Topologien in der formalen Gestalt kombinatorischer Versuchspläne
2. Vorauswahl geeigneter Zieltopologien durch Anwendung des strukturorientierten Verfahrens
3. Bildung einer Eignungshierarchie unter den ausgewählten Architekturen mit Hilfe des Überdeckungsverfahrens
4. Bestimmung der optimalen Zieltopologie durch Anwendung des funktionalen Verfahrens unter zusätzlicher Berücksichtigung phasenübergreifender Einflüsse
5. Festlegung der Modul-Prozessor-Zuordnungen auf Basis der im funktionalen Verfahren erzeugten Mapping-Funktion
6. Ermittlung der Kosten des Mapping durch Auswertung der Zielfunktionen Z_1 und Z_2
7. fakultative abschließende Beurteilung mit Abschätzung der Verluste gegenüber einer idealen Implementierung und Vergleich mit existierenden Implementierungen

---

6 Partitionierung und Einbettung mittels Versuchsplänen

6.1.1 Partitionierung in Parallelrechnern mit statischem Verbindungsnetzwerk

Um Systemressourcen effizient zu nutzen, werden Rechner heute im allgemeinen im Mehrprogrammbetrieb eingesetzt. Es werden also gleichzeitig mehrere Programme im System gehalten, um beispielsweise den Prozessor eines sequentiellen Systems zu nutzen, wenn die übrigen Programme zeitweilig nur Ein-/Ausgabe vornehmen. Mehrprogrammbetrieb gewinnt für Multiprozessorsysteme noch zusätzlich an Gewicht, da es mit der Vervielfachung der Rechenkapazität immer schwieriger wird, diese zufriedenstellend zu nutzen. Ein Multiprozessorrechner, der sich nicht in einer Experimentalumgebung befindet, wird daher häufig im Mehrprogrammbetrieb eingesetzt. Es muß also eine Aufteilung des Gesamtsystems erfolgen, damit den verschiedenen im System befindlichen Programmen Rechenkapazität zur Verfügung steht [W&Y,91].

Bei zukünftigen (massiv-)parallelen Systemen kann aus vielfachen Gründen das Kommunikationsprinzip des gemeinsamen Speichers nicht mehr zum Einsatz kommen; die durch das Netzwerk verbundenen Prozessoren tauschen Daten aus ihren lokalen Speichern per *Message Passing* aus (vgl. Kapitel 7). Das aber verursacht zusätzlichen Aufwand, und zwar umso mehr, je weiter die kommunizierenden Prozessoren voneinander entfernt sind. Der Begriff „Entfernung" bezieht sich hierbei nicht notwendigerweise auf eine räumliche Distanz, sondern mehr auf die Zahl der zu überwindenden Schaltstellen bzw. Prozessoren. Um das Gesamtsystem effizient zu nutzen, ist es also nicht ratsam, im Mehrprogrammbetrieb einem bestimmten Programm die p angeforderten Prozessoren beliebig über das System verteilt anzubieten. Zusammenhängende *Cluster* einer ausreichenden Zahl von Prozessoren scheinen eher angebracht.

Weiter gilt, daß gewisse Algorithmen und gewisse Topologien ideal aufeinander abgestimmt sind (vgl. Kapitel 5) wie das Paar (Radix-2-FFT, Hypercube), so daß es sicherlich vorteilhaft ist, nicht nur ein zusammenhängendes Cluster von Prozessoren einem bestimmten Programm zuzuteilen, sondern die ausgewählte Teilstruktur sollte nach Möglichkeit der Topologie der Gesamtstruktur entsprechen, wenn auch in kleinerer Ausbaustufe. Diese Betrachtungsweise geht von der Annahme aus, daß der zu implementierende Algorithmus auf die Topologie der Gesamtstruktur abzielt; dies ist sinnvoll, da die Möglichkeit besteht, daß das gesamte System exklusiv einem Programm zur Verfügung gestellt wird – wenn übrige Programme nur Ein-/Ausgabe machen – und da die Topologien in der Regel am besten in Teilstrukturen gleicher Art partitionierbar sind. Daneben ist auch der Fall möglich, daß auf dem – vielleicht einzig verfügbaren – Rechner ein Programm ablaufen soll, dessen Algorithmus besser zu einer anderen Topologie korrespondiert. Dieser Fall wird in Abschnitt 6.2 unter der Thematik Einbettung behandelt.

Es soll daher hier unter dem Begriff **Partitionierung** eine Aufteilung des Gesamtsystems in Teilsysteme verstanden werden, die die gleiche Topologie in einer reduzierten Ausbaustufe darstellen [Sie,80]. Von den in Kapitel 3 vorgestellten Architekturen sind manche mehr und manche weniger für dieses „Prinzip der Selbstähnlichkeit" geeignet. Ringe beispielsweise können die Struktur nicht bewahren. Im folgenden sollen Hyperwürfel, gitterartige Strukturen und baumartige Rechner exemplarisch untersucht

werden, da diese nicht nur geeignete Voraussetzungen zur Partitionierung bieten, sondern auch die weitaus am häufigsten realisierten Architekturen sind.

Die Betonung des Hypercube durch häufiges Heranziehen in diversen Beispielen resultiert aus mehreren Gründen:

- Der Hypercube ist bereits eine gut untersuchte Topologie. Daher wird durch die Verwendung des Hypercube als Standardbeispiel ein Vergleich der hier verwendeten Betrachtungsweise mit anderen Verfahren in der Literatur sicherlich am besten unterstützt [FCS,91].
- Die Hypercube-Topologie ist, wenn auch in jüngster Zeit zunehmend Gitter-Topologien Verwendung finden, die meist verwendete Architektur zur Implementierung von parallelen Systemen mit einer mittleren und großen Zahl von Prozessoren. Hierzu vergleiche man die jeweils im Teil B. der Architekturbeschreibungen von Kapitel 3 gemachten Angaben. Die Beliebtheit des Hypercube resultiert aus seinen attraktiven Verbindungseigenschaften, seiner Fehlertoleranz und seiner Modularität [D&H,88]. Daher wird die größte Praxisnähe der hier vorzunehmenden Analysen durch Verwendung dieser Netzwerkstruktur erreicht.
- Die Bauprinzipien des Hypercube sind komplexer oder charakteristischer als die der meisten anderen Strukturen, insbesondere so einfacher Strukturen wie Ring und Stern. Der Einfluß gewisser Modifikationen wie Partitionierung und Einbettung auf die zugrundeliegende Struktur eines verteilten Rechners kann daher am Beispiel des Hypercube in der Regel wesentlich deutlicher herausgearbeitet werden.
- Bezüglich des Verhältnisses zwischen idealem Parallelrechner mit gemeinsamem Hauptspeicher und dem Hypercube sei auf ein Ergebnis von Valiant verwiesen. Valiant [Val,88] hat gezeigt, daß für die Simulation einer parallelen Berechnung eines idealisierten Parallelrechners mit gemeinsamem Hauptspeicher durch einen Hypercube der zusätzliche Aufwand durch einen konstanten Faktor beschrieben werden kann.

Um jedoch die Ergebnisse in ihrer Allgemeingültigkeit nicht einzuschränken, müssen selbstverständlich auch andere Architekturen betrachtet werden.

6.1.2 Partitionierung durch Resolution kombinatorischer Versuchspläne

6.1.2.1 Partitionierung von Hypercubes

Dutt und Hayes [D&H,88] sehen die mögliche Zuweisung von Subcubes unterschiedlicher Größe an die verschiedenen Benutzer im Mehrprogrammbetrieb als eine wesentliche Eigenschaft des Betriebssystems in einem Hypercube-Rechner an. Die Partitionierung wird erforderlich, weil auf zukünftigen Systemen mit vielen tausend oder gar Millionen Prozessoren auch parallele Programme gerechnet werden, die über eine geringere Zahl von Modulen verfügen, das Gesamtsystem also allein nicht ausnutzen [C&L,89]. Das allgemeine Zuweisungsproblem, d.h. die Beantwortung der Frage, ob für eine zufällige Menge freier Subcubes einer zufälligen Anforderung mehrerer Subcubes gleichzeitig entsprochen werden kann, ist NP-vollständig. Bekannte (heuristische) Strategien zur Aufteilung eines Hypercube in Subcubes sind [D&H,88]:

- die lineare Strategie
  Auf eine Anfrage nach einem n-dimensionalen Würfel wird der erste Würfel mit freien Adressen der Form $m2^n$ bis $(m+1)2^n$ zugewiesen.
- die Buddy-Strategie
  Diese aus der Speicherverwaltung bekannte Strategie halbiert und verdoppelt (Speicher-, Adressen-) Bereiche, deren Größen Potenzen von 2 sind.
- Gray-Code-Strategien
  Hierbei wird der Gray-Code zur Bestimmung zusammenhängender Würfel verwendet.

Neben allgemeinen Effizienzüberlegungen sollte von einem guten Verfahren erwartet werden, daß, falls noch ein Subwürfel der angeforderten Größe verfügbar ist, dieser auch zugewiesen wird. Die lineare Strategie beispielsweise leistet dies nicht. Als schwieriges Problem erweisen sich weiterhin interne und externe Fragmentierung des Gesamtsystems [H&J,90b], die nicht vermieden werden können. Externe Fragmentierung kann jedoch minimiert werden, indem man einen Prozeß bereitstellt, der vollständige Subcubes erkennt, und darüber hinaus effiziente Task-Migration ermöglicht. Diese Aktionen bedingen jedoch einen erheblichen Rechenaufwand, so daß auch die dafür notwendigen Berechnungen nach Möglichkeit parallel, also verteilt, ausgeführt werden sollten [T&D,90].

Neben der sehr aufwendigen Methode, Jobs auf andere Subcubes zu migrieren, damit die freien Prozessoren einen der Anforderung genügenden Subcube bilden, schlagen Chen und Lai [C&L,89] vor, durch den Aufbau notwendiger Pfade virtuelle Subcubes hinreichender Größe zu bilden. Dies erfordert jedoch eine besondere Fähigkeit zur Etablierung von Direktverbindungen, wie sie beispielsweise im Intel iPSC/2 gegeben ist.

Die Möglichkeit der rekursiven Erweiterbarkeit einer Rechnertopologie, wie beispielsweise im Falle des Hypercube, beinhaltet gleichzeitig die Möglichkeit, der Aufteilung des Gesamtsystems in die Teilstrukturen kleinerer Ausbaustufe. Faßt man die Hypercube-Topologie, wie in Kapitel 3 beschrieben, als kombinatorischen Versuchsplan auf, so kann man die in Anhang A beschriebenen Gesetzmäßigkeiten zur Zerlegung von optimalen Versuchsplänen in parallele Klassen verwenden, um einen Hyperwürfel der Dimension n in 2^k Hyperwürfel der Dimension 2^{n-k} aufzuspalten.

Beispiel 6.1

Ein Hypercube der Dimension $n = 3$ entspricht folgendem Versuchsplan $(V,\mathcal{B})_{ccc-3}$:
$V_{ccc} = \{0,1,2,3,4,5,6,7\}$
$\mathcal{B}_{ccc} = \{(0,1),(1,2),(2,3),(3,0),(0,7),(1,6),(2,5),(3,4),(4,5),(5,6),(6,7),(7,4)\}$
$v = 8, b = 12, r_x = 3\ \forall x \in V$
$\Lambda_2 = \{\lambda(s): |S| = 2, S \subseteq V\} = \{0,1\}$

Wegen der bidirektionalen Nutzung der Verbindungen ist die Verbindung (3,0) gleichbedeutend mit der Verbindung (0,3); dies geht konform mit der Auffassung von Blöcken als Mengen. Aus Gründen der Übersichtlichkeit werden die Blöcke nach Anwendung der linearen Ordnung $\prec$, die auf der $<$-Relation basiert, daher in der äquivalenten Form (i,j) mit $i \prec j$ basierend auf $i < j$ aufgeführt. Entsprechend den Ausführungen in Anhang B induziert die lineare Ordnung $\prec$ der Menge V dann eine lexikographische Ordnung

$\prec$ im Mengensystem $\mathcal{B}$. Die lexikographische Ordnung bewirkt im Mengensystem des vorliegenden Hypercube folgende Ordnung:
$(0,1)\prec(0,3)\prec(0,7)\prec(1,2)\prec(1,6)\prec(2,3)\prec(2,5)\prec(3,4)\prec(4,5)\prec(4,7)\prec(5,6)\prec(6,7)$

Der Versuchsplan $(V,\mathcal{B})_{ccc-3}$ ist resolvierbar. Teilt man nun die Blöcke des Mengensystems $\mathcal{B}$ im Rahmen einer Resolution in dieser Reihenfolge den parallelen Klassen zu, d.h. zunächst wird versucht, die Klasse mit dem niedrigsten Index zu bedienen, so liefert die Resolution des Hypercube die parallelen Klassen:
$PC\mathfrak{k}_{ccc} = \{(0,1),(2,3),(4,5),(6,7)\}$
$PC\mathfrak{k}_{ccc} = \{(0,3),(1,2),(4,7),(5,6)\}$
$PC\mathfrak{k}_{ccc} = \{(0,7),(1,6),(2,5),(3,4)\}$
Beispielsweise kann der Block (1,2) nicht der Klasse $PC\mathfrak{k}_{ccc}$ zugeteilt werden, da das Element 1 mit dem Block (0,1) bereits in dieser Klasse vorhanden ist, so daß die nächstmögliche Klasse genommen wird.

Betrachtet man nun den Ausgangs-Hypercube (vgl. Abb. 15 auf Seite 41) in der Reihenfolge seiner Anordnung, die durch den Gray-Code gegeben ist, so würde der erste abzuspaltende Sub-Hypercube der Dimension 2 gerade aus den Knoten {0,1,2,3} bestehen. Besinnt man sich darauf, daß ein Hypercube der Dimension $n+1$ konstruiert wird, indem man zwei Hypercubes der Dimension n verwendet und so verbindet, daß jeweils entsprechende Knoten (parallele Knoten, Antipoden) miteinander verbunden werden, so heißt das – angewendet auf den vorliegenden Hypercube –, daß der zweite „Baustein"-Hypercube aus den Knoten {4,5,6,7} bestanden haben muß. Daraus folgt, daß die Hilfsverbindungen, die die beiden Ausgangsstrukturen der Dimension n zu dem neuen Gebilde der Dimension $n+1$ zusammenfügen, sich aus folgenden Verbindungen rekrutieren: {(0,7),(1,6),(2,5),(3,4)}. Diese also quasi „künstlich" eingeführten Verbindungen liefert aber gerade die oben aus dem Versuchsplan durch Resolution entstandene parallele Klasse $PC\mathfrak{k}_{ccc}$. Überträgt man nun die lexikographische Ordnung auf die gewonnenen parallelen Klassen, die selbst wieder kombinatorische Versuchspläne darstellen, so kann man die beiden Subcubes der Dimension 2 aus den parallelen Klassen $PC\mathfrak{k}_{ccc}$ und $PC\mathfrak{k}_{ccc}$ gewinnen, in dem man die jeweils (aufgrund der Ordnung) ersten beiden Blöcke aus den parallelen Klassen zu einem Subcube zusammenfügt und ebenso die jeweils letzten beiden. Damit ist die Partitionierung abgeschlossen.

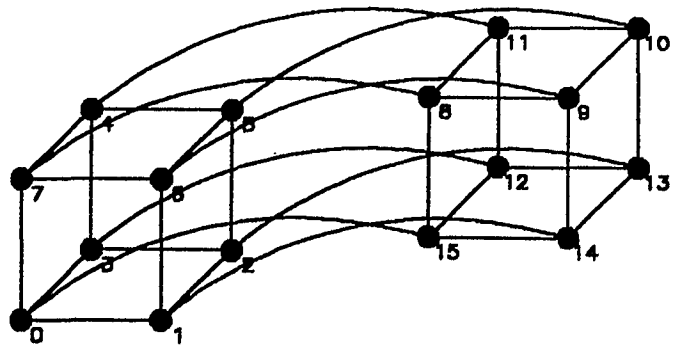


Abb. 34. Hypercube der Stufe (Dimension) 4

Beispiel 6.2

Ein Hypercube der Dimension $n = 4$ (vgl. Abb. 34) entspricht folgendem Versuchsplan $(V,\mathcal{B})_{ccc-4}$:

$V_{ccc} = \{0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15\}$

$\mathcal{B}_{ccc} = \{(0,15),(1,14),(2,13),(3,12),(4,11),(5,10),(6,9),(7,8),(0,7),(1,6),(2,5),(3,4),(8,15),$
$(9,14),(10,13),(11,12),(0,3),(1,2),(4,7),(5,6),(8,11),(9,10),(12,15),(13,14),(0,1),(2,3),$
$(4,5),(6,7),(8,9),(10,11),(12,13),(14,15)\}$

$v = 16, b = 32, r_x = 4\ \forall x \in V$

$\Lambda_2 = \{\lambda(s): |S| = 2, S \subseteq V\} = \{0,1\}$

Der Versuchsplan $(V,\mathcal{B})_{ccc-4}$ ist resolvierbar. Die lexikographische Ordnung $\prec$ erzeugt im Mengensystem $\mathcal{B}$ folgende Ordnung:

$(0,1) \prec (0,3) \prec (0,7) \prec (0,15) \prec (1,2) \prec (1,6) \prec (1,14) \prec (2,3) \prec (2,5) \prec (2,13) \prec (3,4) \prec (3,12) \prec (4,5)$
$\prec (4,7) \prec (4,11) \prec (5,6) \prec (5,10) \prec (6,7) \prec (6,9) \prec (7,8) \prec (8,9) \prec (8,11) \prec (8,15) \prec (9,10)$
$\prec (9,14) \prec (10,11) \prec (10,13) \prec (11,12) \prec (12,13) \prec (12,15) \prec (13,14) \prec (14,15)$

Teilt man nun die Blöcke des Mengensystems $\mathcal{B}$ in dieser Reihenfolge den parallelen Klassen zu, d.h., zunächst wird versucht, die Klasse mit dem niedrigsten Index zu bedienen, so liefert die Resolution des Hypercube die parallelen Klassen:

$PC^1_{ccc} = \{(0,1),(2,3),(4,5),(6,7),(8,9),(10,11),(12,13),(14,15)\}$
$PC^2_{ccc} = \{(0,3),(1,2),(4,7),(5,6),(8,11),(9,10),(12,15),(13,14)\}$
$PC^3_{ccc} = \{(0,7),(1,6),(2,5),(3,4),(8,15),(9,14),(10,13),(11,12)\}$
$PC^4_{ccc} = \{(0,15),(1,14),(2,13),(3,12),(4,11),(5,10),(6,9),(7,8)\}$

Betrachtet man nun den Ausgangs-Hypercube (vgl. Abb. 34) in der Reihenfolge seiner Anordnung, die durch den Gray-Code gegeben ist, so würde der erste abzuspaltende Sub-Hypercube der Dimension 3 gerade aus den Knoten {0,1,2,3,4,5,6,7} bestehen. Daraus folgt, daß der zweite „Baustein"-Hypercube aus den Knoten {8,9,10,11,12,13,14,15} bestanden haben muß. So ergibt sich, daß die Hilfsverbindungen, die die beiden Ausgangsstrukturen der Dimension n zu dem neuen Gebilde der Dimension $n+1$ zusammenfügen, sich aus folgenden Verbindungen rekrutieren: {(0,15),(1,14),(2,13),(3,12),(4,11),(5,10),(6,9),(7,8)}. Diese also quasi „künstlich" eingeführten Verbindungen liefert aber gerade die oben aus dem Versuchsplan durch Resolution entstandene parallele Klasse PC^4_{ccc}. Überträgt man nun die Ordnung auf die gewonnenen parallelen Klassen, die selbst wieder kombinatorische Versuchspläne darstellen, so kann man die beiden Subcubes der Dimension 3 aus den parallelen Klassen PC^1_{ccc} bis PC^3_{ccc} gewinnen, in dem man die jeweils (aufgrund der Ordnung) ersten vier Blöcke aus den parallelen Klassen zu einem Subcube zusammenfügt und ebenso die jeweils letzten vier. Damit ist die Partitionierung abgeschlossen.

Die Erweiterung auf den allgemeinen Fall führt zu folgendem

Satz: Ein Hypercube, welcher nach dem Gray-Code aufgebaut ist, ist stets resolvierbar.

Beweis: O.B.d.A. seien die Knoten des Hypercube jeweils von $0,\ldots,N-1 = 2^n-1$ numeriert, anhand der linearen Ordnung $\prec$ innerhalb von Blöcken jeweils

geordnet und die Blöcke des Hypercube darauf aufbauend jeweils lexikographisch geordnet. Sei der betrachtete Hypercube wie vorausgesetzt im Gray-Code adressiert.

(Induktion über die Hypercube-Dimension n, bzw. über den Aufbau des Gray-Code.)

I. Induktionsanfang ($n = 2$)

z.z.: Ein Hypercube der Dimension 2 ist resolvierbar.
Ein Hypercube der Dimension 2 wird durch folgenden Versuchsplan beschrieben: $(V,\mathscr{B})$ mit $V = \{0,1,2,3\}$ und $\mathscr{B} = \{(0,1),(0,3),(1,2),(2,3)\}$. Die Zerlegung des Hypercube in die Klassen $C^1 = \{(0,1),(2,3)\}$ und $C^2 = \{(0,3),(1,2)\}$ ist eine Resolution des Hypercube, weil C^1 und C^2 parallele Klassen sind, da sie Mengen von disjunkten Blöcken sind, die alle Elemente von V umfassen, und die Vereinigung der beiden Klassen das vollständige Mengensystem $\mathscr{B}$ ergibt. ⌖

II. Induktionsvoraussetzung

Ein Hypercube der Dimension n ist resolvierbar.

III. Induktionsdurchführung (Schluß von n auf $n + 1$*)*

Behauptung:
Ein Hypercube der Dimension $n + 1$ ist resolvierbar.

Beweis:
Gegeben sei ein Hypercube der Dimension n. Sei mit Induktionsvoraussetzung $PC^1, ..., PC^n$ eine Resolution des Hypercube der Dimension n. Der Hypercube der Dimension n wird nun dupliziert. Dann werden Knoten mit gleicher Adressierung miteinander verbunden. Es entsteht ein neuer Hypercube mit der Dimension $n + 1$. Nun werden die Knotenadressen des Gesamtgebildes um 1 Bit erweitert, wobei den Gray-Code-Knotenadressen des einen Baustein-Hypercube eine 0 vorangestellt wird, denen des zweiten eine 1. Auf diese Weise ist jede Knotenadresse im Gesamtgebilde eindeutig, da auch die früheren Adressen eindeutig waren, und die Adressierung entspricht insgesamt wieder dem Gray-Code (vgl. Anhang B).

Die Verbindungen, die die beiden Cubes der Dimension n zum neuen Hypercube der Dimension $n + 1$ verbinden, sind die $N = 2^n$ Verbindungen: $\{(0a_{0,n-1}...a_{0,0}, 1a_{0,n-1}...a_{0,0}), ..., (0a_{n-1,n-1}...a_{n-1,0}, 1a_{n-1,n-1}...a_{n-1,0})\}$.
Geht man nun mit Induktionsvoraussetzung von der Resolution des Hypercube der Dimension n aus: $\zeta = \{PC^1, ..., PC^n\}$ und erweitert jede parallele Klasse PC^i mit N Blöcken auf $2N$ Blöcke durch Ersetzung eines Blocks der Form $(a_{j,n-1}...a_{j,0}, a_{k,n-1}...a_{k,0})$ durch 2 Blöcke der Form

$(0a_{j,n-1}...a_{j,0}, 0a_{k,n-1}...a_{k,0})$ und $(1a_{j,n-1}...a_{j,0}, 1a_{k,n-1}...a_{k,0})$, so ergibt sich folgendes:

- Bezeichnet man die so durch Ersetzung der parallelen Klassen von ζ entstandenen Klassen doppelter Mächtigkeit mit $C^1, ..., C^n$ und die eingeführten Zwischenverbindungen zwischen beiden Baustein-Hypercubes als Klasse C^{n+1}, so bilden diese $n+1$ Klassen eine *Zerlegung* des Hypercube der Dimension $n+1$, da sie alle dort vorkommenden Verbindungen enthalten.

- Jede der entstandenen Klassen C^i ist eine *parallele* Klasse, weil

  1. die Ersetzungsoperation auf den parallelen Klassen des n-dimensionalen Hypercube die *Eindeutigkeit* der Elemente pro Klasse erhält; anstelle der Zahl $a_{i,n-1}...a_{i,0}$ sind nun die Zahlen $0a_{i,n-1}...a_{i,0}$ und $1a_{i,n-1}...a_{i,0}$ in der verdoppelten Klasse. Die Klasse C^{n+1} enthält aufgrund des Aufbaus der Zwischenverbindungen (vgl. o.) ebenfalls jede Zahl exakt einmal. Mit anderen Worten: Die Klassen $C^1, ..., C^{n+1}$ sind Mengen *disjunkter* Blöcke.
  2. Darüber hinaus sind in jeder Klasse C^i *alle* Knoten des $n+1$-dimensionalen Hypercube enthalten. Da nach Induktionsvoraussetzung eine Resolution des n-dimensionalen Hypercube gegeben war, und somit jede parallele Klasse alle Elemente dieses Hypercube enthielt, sind durch die Ersetzungsoperation entsprechend den Konstruktionsprinzipien des Gray-Code in jeder der Klassen $C^1, ..., C^n$ wiederum alle Knoten des verdoppelten Hypercube enthalten. Durch das Konstruktionsschema der eingeführten Verbindungen sind auch in C^{n+1} alle Knoten vertreten.

⇒ Durch die Klassen $C^1, ..., C^{n+1}$ wird eine Resolution des Hypercube der Dimension $n+1$ beschrieben.

q.e.d.

Es kann also festgehalten werden:

- Ein Hypercube $(V,\mathcal{B})_{ccc}$ der Dimension n ist stets resolvierbar und die Resolution liefert n parallele Klassen PC_{ccc}, PC_{ccc}, ..., PC_{ccc}.

- Jede dieser parallelen Klassen enthält definitionsgemäß alle Knoten der Gesamtstruktur, d.h.: $|V_{PCccc}| = |V_{ccc}| \quad \forall 1 \leq i \leq n$, mit anderen Worten, die PPC sind maximal.

- Jede dieser parallelen Klassen – dies ergibt sich aus dem vorherigen Punkt – besitzt die gleiche Zahl von (disjunkten) Blöcken, die gewissen Verbindungen im Hypercube entsprechen, repräsentiert also genau b/n Verbindungen des zu partitionierenden Hypercube.

- Eine dieser parallelen Klassen, und zwar diejenige mit dem höchsten Index, liefert genau die beim Aufbau „künstlich" eingeführten Verbindungen zur Verknüpfung der beiden Ausgangsstrukturen.

- Um einen Hypercube der Dimension n wieder auf zwei Subcubes der Dimension $n-1$ zurückzuführen, was das Ziel des Partitionierungsvorhabens ist, werden daher mittels Resolution diese „Schweißnähte" wieder ausgesondert.

- Die beiden Subcubes bestehen nun aus den Verbindungen, die in den übrigen $n-1$ parallelen Klassen enthalten sind.

- Die Zuordnung der Verbindungen dieser $n-1$ restlichen Klassen zu den beiden Subcubes erfolgt wiederum durch Einführen der lexikographischen Ordnung $\prec$ (vgl. Anhang B) auf die restlichen $n-1$ parallelen Klassen. Subcube A der Dimension $n-1$ wird gebildet aus den $(n-1)\,2^{n-2}$ Blöcken niedrigerer Ordnung jeder Klasse, Subcube B aus den $(n-1)\,2^{n-2}$ Blöcken höherer Ordnung.

Weitere Bemerkungen:

- Hier wurde die Beweisführung auf dem Gray-Code aufgebaut und die entsprechende Numerierung vorausgesetzt; die übliche Dualnumerierung ist jedoch ebenso zulässig.
- Eine untere Schranke für die Zahl n der parallelen Klassen bei der Resolution eines Hypercube der Dimension n ist bereits durch den Wiederholungsfaktor r_x gegeben. In Kapitel 3 wurde dieser mit $r_x = n$ für einen Hypercube der Dimension n bestimmt. Da nun eine parallele Klasse aus disjunkten Blöcken und die Resolution eines Versuchsplans aus disjunkten parallelen Klassen bestehen muß, erzwingt $r_x = n$ mindestens n parallele Klassen.
- Die Konkretisierung des Wertes n ergibt sich durch die mögliche (und stattfindende) Resolution des entsprechenden Versuchsplans anstelle einer gewöhnlichen Zerlegung.
- Die lexikographische Ordnung $\prec$ garantiert die Eindeutigkeit der Zusammensetzung der parallelen Klassen. Ohne die Einführung dieser Ordnung sind auch andere Klassenzuordnungen möglich.
- Benötigt man im Mehrprogrammbetrieb eines (massiv-)parallelen Rechners Teilstrukturen unterschiedlicher Größe, so ist es möglich, eine von beiden aktuell gefundenen Substrukturen weiter zu partitionieren, während die andere Teilstruktur intakt beibehalten wird.

Zur Partionierung eines Hypercube der Dimension 2^n in 2^k Subcubes der Dimension 2^{n-k} wird Algorithmus 6 k-fach angewandt.

Die Komplexitätsanalyse des Algorithmus 6 führt zu folgenden Ergebnissen: Da bei der hier vorgenommenen kombinatorischen Behandlung der Aufgabenstellungen stets vom Umgang mit Versuchsplänen ausgegangen wird, wird die jeweilige Versuchsplandarstellung einer Topologie als gegeben betrachtet; Schritt 1 wäre also kostenlos. Die Einführung der lexikographischen Ordnung erfordert zunächst b Vergleiche, um die Elemente in Blöcken entsprechend der linearen Ordnung $\prec$ zu sortieren und anschließend $b \log b$ Sortieroperationen. Die Resolution des Hypercube erfordert $N^2 \log N$ Operationen, da es einem ($\log N$)-fachen Sortieren durch Einfügen der $N = v$ Elemente entspricht. Das

Entfernen der parallelen Klasse PC^{1}_{ccc} geschieht in 1 Operation. Die lexikographische Ordnung in den übrigen Klassen wird kostenfrei aus der Resolution beibehalten. Das Auftrennen der $n-1$ Klassen ist unter Ausnutzung eines geeigneten Speicherungsschemas kostenfrei. Die Konstruktion der Subcubes erfolgt mit der gleichen Technik ebenfalls kostenfrei. Insgesamt ergibt sich dann die Komplexität:
$O(b) + O(b \log b) + O(N^2 \log N) + O(1)$
Unter Beachtung der Zusammenhänge $b = n2^{n-1}$ und $N = 2^n$ ergibt sich:
$O(b \log b) + O(N^2 \log N) = O(N^2 \log N)$
Ein Implementierung wird erst für ein massiv-paralleles System sinnvoll, welches als Zielrechner im Rahmen dieser Arbeit auch beabsichtigt ist.

Algorithmus 6: Partitionierung von Hypercubes in Subcubes

1. Erzeuge den Versuchsplan $(V,\mathcal{B})_{ccc}$, der den Hypercube spezifiziert, wie in Kapitel 3 beschrieben.

2. Führe die lexikographische Ordnung $\prec$ in das Mengensystem $\mathcal{B}_{ccc}$ ein.

3. Resolviere den Hypercube $(V,\mathcal{B})_{ccc}$ in n parallele Klassen PC^{1}_{ccc}, PC^{2}_{ccc}, ..., PC^{n}_{ccc}, wobei die geordneten Blöcke nacheinander parallelen Klassen mit möglichst niedrigem Index zuzuordnen sind.

4. Entferne die parallele Klasse PC^{1}_{ccc} aus der Resolution.

5. Lege die Ordnung $\prec$ auf die parallelen Klassen PC^{1}_{ccc}, PC^{2}_{ccc}, ..., PC^{n-1}_{ccc}.

6. Trenne jede parallele Klasse PC^{i}_{ccc}, $1 \leq i \leq n-1$, in je zwei partiell-parallele Klassen PPC^{i1}_{ccc} und PPC^{i2}_{ccc} auf, die die 2^{n-2} Blöcke niedrigerer bzw. höherer Ordnung enthalten.

7. Konstruiere Subcube A durch die Vereinigung aller partiell-parallelen Klassen PPC^{i1}_{ccc}, $1 \leq i \leq n-1$, und ebenso Subcube B vermöge $(V,\mathcal{B})_{ccc2} = \bigcup_{1 \leq i \leq n-1} PPC^{i2}_{ccc}$.

6.1.2.2 Partitionierung von Binärbäumen

Im folgenden sollen nur vollbesetzte Binärbaume betrachtet werden, wie sie in Kapitel 3 eingeführt wurden. Unvollständige Binärbaume würden die nun folgenden Betrachtungen nicht wesentlich beeinträchtigen, jedoch eine Fülle von zu berücksichtigenden Sonderfällen erzwingen. Weiterhin wird davon ausgegangen, daß die Knoten des Gesamtsystems systematisch numeriert sind, und zwar so, wie dies in Kapitel 3 beschrieben ist. Einzelknoten werden im folgenden nicht als Bäume angesehen, womit der kleinste Binärbaum aus einer Wurzel und ihren zwei Söhnen, insgesamt also aus drei Knoten besteht.

Wie im Falle der Hyperwürfel kann auch bei Binärbaumen von einem rekursiven Aufbau gesprochen werden, so daß ebendieser, durch inverses Vorgehen, zur Partitionierung des

Gesamtsystems in Teilbäume dient. Der wesentliche Unterschied der Binärbäume gegenüber den eben untersuchten Hyperwürfeln im Hinblick auf eine Partitionierung besteht darin, daß sich die Hypercubes vollständig in Subcubes auflösen lassen, während bei den Binärbäumen die Nutzung der ursprünglichen Wurzel nicht mehr möglich ist. Die Bezeichnung „vollständige Partitionierung" bezieht sich nur auf die Knoten des Gesamtsystems, nicht aber auf die zwischen diesen bestehenden Verbindungen; denn, wie oben gezeigt wurde, werden auch bei der Partitionierung von Hypercubes gewisse Verbindungen aus der Gesamtstruktur herausgelöst. Die alleinige Berücksichtigung der Knoten bezüglich einer Aufteilung und weiteren Nutzung ist jedoch sinnvoll, da die Knoten die Prozessoren und somit die Rechenkapazität des Gesamtsystems repräsentieren. Im Falle eines Binärbaums ergibt sich die Situation, daß bei Abspaltung der beiden Binärbäume, deren Wurzeln die Söhne der ursprünglichen Wurzel sind, die Isolierung dieser Ausgangswurzel eintritt, so daß deren Rechenkapazität ausfällt und die Leistungsfähigkeit des Systems im Gegensatz zum Hypercube zurückgeht. Für sequentielle Anwendungen wäre der Einzelprozessor der Wurzel natürlich weiterhin nutzbar. Einzelknoten werden hier jedoch nicht als vollständige Bäume angesehen; als Anforderungen werden daher nur reguläre Teilbäume des Gesamtbaumes betrachtet.

Beispiel 6.3

Ein Tree-Connected Computer der Stufe $n = 2$ entspricht folgendem Versuchsplan $(V,\mathscr{B})_{TCC-2}$:
$V_{TCC} = \{0,1,2,3,4,5,6\}$
$\mathscr{B}_{TCC} = \{(0,1),(0,2),(1,3),(1,4),(2,5),(2,6)\}$
$v = 7, b = 6$
$r_0 = 2$ (Wurzel), $r_x = 1\ \forall x \in \{3,4,5,6\}$ (Blätter), $r_y = 3\ \forall y \in \{1,2\}$ (übrige Knoten)
$\Lambda_2 = \{\lambda(s): |S| = 2, S \subseteq V\} = \{0,1\}$

Betrachtet man nun die Defintion eines abgeleiteten Versuchsplans (V\X,$\mathscr{B}$\X) (vgl. Anhang A) und initiert die Menge X mit der Wurzel des Ausgangsbaumes, d.h. $X = \{0\}$, so liefert die Ableitung des Versuchsplans $(V,\mathscr{B})_{TCC}$ nach X den Versuchsplan $(V\backslash\{0\},\mathscr{B}\backslash\{0\})$.

Dieser nach {0} abgeleitete Versuchsplan hat explizit folgende Gestalt:
$V_{TCC}\backslash\{0\} = \{1,2,3,4,5,6\}$
$\mathscr{B}_{TCC}\backslash\{0\} = \{(1,3),(1,4),(2,5),(2,6)\}$
$v = 6, b = 4,$
$r_1 = r_2 = 2$ (für die späteren Wurzeln)
$r_3 = r_4 = r_5 = r_6 = 1$ (für die späteren Blätter)
$\Lambda_2 = \{\lambda(s): |S| = 2, S \subseteq V\} = \{0,1\}$

Führt man nun wie im Fall der Hypercube-Partitionierung die lexikographische Ordnung $\prec$ in das Mengensystem des abgeleiteten Plans ein und spaltet dieses dann, anhand der Ordnung $\prec$, in die zwei Teilsysteme oberer und unterer Ordnung auf, so liefern die so entstandenen Mengensysteme, die maximal verfügbaren Teilbäume {(1,3),(1,4)} und {(2,5),(2,6)}, deren Wurzeln die Söhne der ursprünglichen Wurzel sind.

Beispiel 6.4

Ein Tree-Connected Computer der Stufe $n = 3$ (vgl. Abb. 17 auf Seite 43) entspricht folgendem Versuchsplan $(V,\mathscr{B})_{TCC-3}$.
$V_{TCC} = \{0,1,2,3,4,5,6,7,8,9,10,11,12,13,14\}$
$\mathscr{B}_{TCC} = \{(0,1),(0,2),(1,3),(1,4),(2,5),(2,6),(3,7),(3,8),(4,9),(4,10),(5,11),(5,12),(6,13),(6,14)\}$
$v = 15, b = 14$
$r_0 = 2$ (Wurzel), $r_x = 1\ \forall x \in \{1,2,...,6\}$ (Blätter), $r_y = 3\ \forall y \in \{7,8,...,14\}$ (übrige Knoten)
$\Lambda_2 = \{\lambda(s): |S| = 2, S \subseteq V\} = \{0,1\}$

Betrachtet man nun den abgeleiteten Versuchsplan $(V\backslash X,\mathscr{B}\backslash X)$, wobei die Menge X mit der Wurzel des Ausgangsbaumes initiiert wird, d.h. $X = \{0\}$, so liefert die Ableitung des Versuchsplans $(V,\mathscr{B})_{TCC}$ nach X den Versuchsplan $(V\backslash\{0\},\mathscr{B}\backslash\{0\})$.

Dieser nach {0} abgeleitete Versuchsplan hat explizit folgende Gestalt:
$V_{TCC}\backslash\{0\} = \{1,2,3,4,5,6,7,8,9,10,11,12,13,14\}$
$\mathscr{B}_{TCC}\backslash\{0\} = \{(1,3),(1,4),(2,5),(2,6),(3,7),(3,8),(4,9),(4,10),(5,11),(5,12),(6,13),(6,14)\}$
$v = 14, b = 12,$
$r_1 = r_2 = 2$ (für die späteren Wurzeln)
$r_x = 1\ \forall x \in \{7,8,...,14\}$ (für die späteren Blätter)
$r_y = 3\ \forall y \in \{3,4,5,6\}$ (für alle übrigen Knoten)
$\Lambda_2 = \{\lambda(s): |S| = 2, S \subseteq V\} = \{0,1\}$

Zunächst induziert man mit Hilfe der linearen Ordnung $\prec$ eine lexikographische Ordnung auf dem Mengensystem $\mathscr{B}_{TCC}\backslash\{0\}$. Beginnend bei dem kleinsten Element, {1,3}, beginnt man nun und weist der Teilstruktur $\mathscr{B}^A$ zunächst zwei Blöcke des Mengensystems zu, dann erhält die zweite Teilstruktur $\mathscr{B}^B$ ebenfalls zwei Blöcke. Anschließend werden je vier Blöcke zunächst $\mathscr{B}^A$ und dann die letzten vier verbleibenden Blöcke $\mathscr{B}^B$ zugeteilt. Die Partitionierung ist damit abgeschlossen. Die so entstandenen Mengensysteme $\mathscr{B}^A = \{(1,3),(1,4),(3,7),(3,8),(4,9),(4,10)\}$, $\mathscr{B}^B = \{(2,5),(2,6),(5,11),(5,12),(6,13),(6,14)\}$ sind die maximal verfügbaren Teilbäume; ihre Wurzeln sind die Söhne der ursprünglichen Wurzel.

Es kann also festgehalten werden:

- Die Aufteilung eines Binärbaums in die nächstkleineren Teilbäume bringt eine Leistungseinbuße des Gesamtsystems mit sich, da der Prozessor, der durch den Wurzelknoten repräsentiert wird, für die Dauer der Partitionierung ausfällt.

- Die Menge aller Knoten und aller Verbindungen, die nach der Partitionierung noch verfügbar sind, können ermittelt werden, indem man den Versuchsplan, der den Binärbaum verkörpert, nach der Menge X ableitet, die als einziges Element die Wurzel enthält.

- Führt man nun die lexikographische Ordnung $\prec$ in das Mengensystem des abgeleiteten Versuchsplans ein, so entstehen die gesuchten maximalen Teilbäume durch Zuteilen von zunächst je zwei, dann je vier Blöcken und so fort an die beiden zu generierenden Partitionen bis alle Blöcke des Gesamtsystems aufgeteilt sind.

Weitere Bemerkungen:

- Wie im Fall des Hypercube ist auch hier die Einführung der lexikographischen Ordnung $\prec$ wesentlich; neben der Nichteindeutigkeit der entstehenden Mengensysteme würde auch der Zusammenhang der beiden Teilbäume verloren gehen.
- Für die variierenden Anforderungen in einer Mehrbenutzerumgebung kann auf jeder Stufe der Partitionierung mit der Aufteilung einer der beiden erzeugten Teilbäume fortgefahren werden, während der andere in der gerade erzeugten Dimension benötigt und daher auch beibehalten wird.
- Aufgrund des Konstruktionsprinzips der Binärbäume geht die Verteilung der Blöcke in der beschriebenen Art stets auf.
- Bei einer mehrstufigen Partitionierung ist der Wurzelknoten eines aktuellen Teilbaums, nach dem also abgeleitet werden muß, einfach zu ermitteln, da er das kleinste Element der Menge V_A bezüglich der linearen Ordnung $\prec$ darstellt.
- Würde man anstelle von binären mit q-nären Bäumen arbeiten, so würden zunächst je q, dann je q^2, q^3 und so fort Blöcke zu den q entstehenden Teilbäumen zuzuweisen sein.

Zur Partitionierung eines Tree-Connected Computers der Stufe n in zwei Tree-Connected Computer der Stufe $n-1$ wende man Algorithmus 7 an:

Algorithmus 7: Partitionierung von Binärbäumen in Teilbäume

1. Erzeuge den Versuchsplan $(V,\mathscr{B})_{TCC}$, der den Tree-Connected Computer beschreibt, wie in Kapitel 3 beschrieben.

2. Leite den Versuchsplan $(V,\mathscr{B})_{TCC}$ nach der Wurzel, d.h. dem Knoten 0, ab. Bilde also den Versuchsplan $(V\backslash\{0\},\mathscr{B}\backslash\{0\})_{TCC}$.

3. Führe die lexikographische Ordnung $\prec$ in das Mengensystem des nach $\{0\}$ abgeleiteten Versuchsplans ein.

4. Stelle zwei Zielstrukturen, d.h. Mengensysteme, $\mathscr{B}^A$ und $\mathscr{B}^B$ bereit, die mit der leeren Menge initialisiert werden.

5. Der eingeführten Ordnung folgend, beginnend beim kleinsten Element, weise diesen Mengensystemen im k-ten Schritt, $1 \leq k \leq n-1$, 2^k Blöcke en bloc zu, d.h. zunächst erhält $\mathscr{B}^A$ 2^k Blöcke, anschließend erhält $\mathscr{B}^B$ ebenso 2^k Blöcke.

6. Nach $n-1$ Schritten ist die Partitionierung abgeschlossen und jede Teilstruktur hat $\sum_{i=1}^{n-1} 2^i = 2^n - 2$ Blöcke erhalten.

7. Je nach Anforderung wird nun mit der Partitionierung in einer bestimmten Stufe s mit einem oder mit beiden Teilbäumen sukzessive weiterverfahren, d.h. dieser Algorithmus wird auf die entstandenen Teilstrukturen angewendet.

Die Komplexitätsanalyse von Algorithmus 7 wird durch Betrachtung der Einzelkomplexitäten der Schritte 1 – 6 durchgeführt. Für Schritt 1 wird die Versuchsplandarstellung der Topologie als gegeben vorausgesetzt, d.h. es entstehen keine Kosten. Das Ableiten bedingt 1 Operation auf der Menge V und 2 Operationen, bei naivem Vorgehen b Operationen, auf dem Mengensystem $\mathscr{B}$. Das Einführen der lexikographischen Ordnung entspricht einem Sortiervorgang der Komplexität $O(b \log b)$. Schritt 4 erfolgt kostenfrei. Die Schritte 5 und 6 bedingen $b-2$ Operationen, da jeder Block nur genau einmal betrachtet werden muß. Insgesamt ergibt sich die Komplexität:
$O(b) + O(b \log b) + O(b) = O(b \log b)$
Unter Beachtung des Zusammenhangs $N = b + 1$, ergibt sich damit auch bezüglich der Knotenzahl N die Komplexität $O(N \log N)$. Eine Verwendung des Verfahrens ist nur für einen Parallelrechner mit einer großen Zahl von Prozessoren sinnvoll.

6.1.2.3 Partitionierung von maschenverbundenen Rechnern

Gegenwärtig werden in der Rechnerarchitektur gitterartige Topologien präferiert, während die Hypercube-Topologie an Akzeptanz zu verlieren scheint. Diese Entwicklung steht im Zusammenhang mit dem Übergang zu hoch- und massiv-parallelen Systemen. Die Verbindungskosten (vgl. Kapitel 3) favorisieren hier die Gitterstrukturen gegenüber den Hypercube-Strukturen, da im Gitter die Verbindungskosten pro Knoten auch bei wachsender Prozessorzahl konstant bleiben, während im Hypercube die Zahl der Verbindungen mit der Zahl der beteiligten Prozessoren wächst, wenn auch nur logarithmisch. Hinzu kommt, daß die bislang so sehr geschätzte Eigenschaft der Fehlertoleranz, die im Hypercube mit dem Redundanzwert $r_x = \log n - 1$ zu charakterisieren ist, bei den heutigen, wesentlich zuverlässigeren, verteilten Parallelrechnern nicht mehr so stark ins Gewicht fällt, so daß man die im Mesh-Connected Computer gegebene konstante, von der Rechnerausbaustufe unabhängige, Redundanz für ausreichend hält. Valiant [Val,88] hält jedoch für den Fall, daß keine Lokalität in der Kommunikation auftritt, $\log n$ Anschlüsse pro Knoten, wie dies im Hypercube gegeben ist, für notwendig. Eine weitere Ursache der Abkehr vom Hypercube könnte in der besseren Eignung gitterartiger Strukturen für die wichtige Klasse der systolischen Algorithmen zu finden sein.

Im folgenden sollen exemplarisch zweidimensional maschenverbundene Rechner betrachtet werden, die in Kapitel 3 mit der Bezeichung MCC2 klassifiziert wurden. Zur Partitionierung eines MCC2 der Stufe n sind nun zwei grundsätzliche Vorgehensweisen möglich:

- Partitionierung in 4 MCC2 der Stufe $n/2$
- Stufenweises Einreduzieren auf 1 $(n-k)(n-k)$-System mit $k = 1, ..., n-1$ in eine feste „Richtung" etwa SO

Man kann einen zweidimensional maschenverbundenen Rechner durch den Übergang von einem $n \times n$-System auf ein (1) $(n-1) \times (n-1)$-System sukzessive in eine feste Richtung, etwa SO, „einschrumpfen", bis man schließlich das kleinstmögliche System noch ausreichender Größe produziert hat, welches der Anforderung dann zugewiesen wird. Bei diesem Verfahren stehen im ersten Zwischenschritt nicht alle Prozessoren zur Verfügung, da sie keine Masche bilden. Es sind die Prozessoren des früheren

„Nordwest-Rahmens". Ein Vorteil dieses Verfahrens besteht darin, daß n nicht notwendigerweise gerade sein muß.

Da das oben erwähnte „Prinzip der Selbstähnlichkeit" jedoch konsequent fortgeführt werden soll, wird hier die Partitionierung eines MCC2 der Stufe n in 4 $(n/2 \times n/2)$-Teilsysteme untersucht. Dazu benötigt man als Ausgangssytem ein MCC2 gerader Stufenzahl. Daher sei n für die Dauer dieser Betrachtungen (Abschnitt 6.1) gerade. Bei dieser Partitionierungsmethode bleibt, wie bei der Partitionierung der Hyperwürfel, die Leistungsfähigkeit des Gesamtsystems erhalten, da die Summe der Teilstrukturen stets alle Knoten des Gesamtsytems umfaßt.

Faßt man die maschenverbundene Topologie als kombinatorischen Versuchsplan auf, wie dies in Kapitel 3 geschehen ist, so kann man die Partitionierung des MCC2 wie im Falle des Hypercube auf die Resolution optimaler Versuchspläne zurückführen.

Beispiel 6.5

Ein zweidimensional maschenverbundener Rechner der Stufe $n = 4$ entspricht folgendem Versuchsplan $(V,\mathcal{B})_{MCC2}$:

$V_{MCC2} = \{0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15\}$

$\mathcal{B}_{MCC2} = \{(0,1),(1,2),(2,3),(3,0),(4,5),(5,6),(6,7),(7,4),(8,9),(9,10),(10,11),(8,11),(12,13),$
$(13,14),(14,15),(15,12),(0,4),(4,8),(8,12),(12,0),(1,5),(5,9),(9,13),(13,1),(2,6),$
$(6,10),(10,14),(14,2),(3,7),(7,11),(11,15),(15,3)\}$

$v = 16, b = 32, r_x = 4\ \forall x \in V$

$\Lambda_2 = \{\lambda(s): |S| = 2, S \subseteq V\} = \{0,1\}$

Abb. 35 verdeutlicht, in welche Teilgitter der MCC2 partitioniert werden soll.

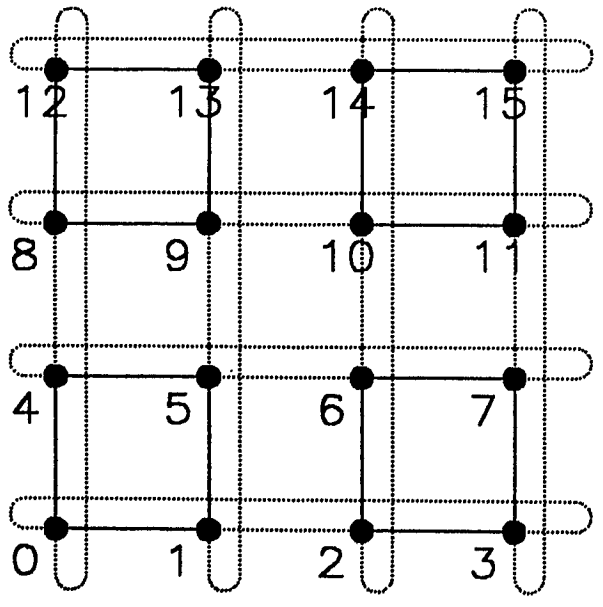


Abb. 35. Partitionierung eines MCC2 der Stufe 4 in 4 MCC2 der Stufe 2

Die Abbildung zeigt, daß bei der Aufteilung in Teilgitter die Maschen der Substrukturen nicht nach außen geschlossen werden können. Daher spielt die 1. Stufe der Partitionierung eine Sonderrolle.

Der Versuchsplan $(V,\mathscr{B})_{MCC2}$ ist resolvierbar.
Die lexikographische Ordnung $\prec$ (vgl. Anhang B) erzeugt im Mengensystem $\mathscr{B}$ folgende Ordnung:

$$(0,1)\prec(0,3)\prec(0,4)\prec(0,12)\prec(1,2)\prec(1,5)\prec(1,13)\prec(2,3)\prec(2,6)\prec(2,14)\prec(3,7)\prec(3,15)$$
$$\prec(4,5)\prec(4,7)\prec(4,8)\prec(5,6)\prec(5,9)\prec(6,7)\prec(6,10)\prec(7,11)\prec(8,9)\prec(8,11)\prec(8,12)\prec(9,10)$$
$$\prec(9,13)\prec(10,11)\prec(10,14)\prec(11,15)\prec(12,13)\prec(12,15)\prec(13,14)\prec(14,15)$$

Hierbei gilt wieder, daß, wegen der bidirektionalen Nutzung der Verbindungen, eine Verbindung (i,j) gleichbedeutend mit der Verbindung (j,i) ist; dies geht konform mit der Auffassung von Blöcken als Mengen. Aus Gründen der Übersichtlichkeit werden die Blöcke nach Anwendung der Ordnung daher in der äquivalenten Form (i,j) mit $i<j$ aufgeführt.

Teilt man nun die Blöcke des Mengensystems $\mathscr{B}$ in dieser Reihenfolge den parallelen Klassen zu, d.h. zunächst wird versucht, die Klasse mit dem niedrigsten Index zu bedienen, so liefert die Resolution des Mesh-Connected Computer MCC2 die parallelen Klassen:

$PC^1_{MCC2} = \{(0,1),(2,3),(4,5),(6,7),(8,9),(10,11),(12,13),(14,15)\}$
$PC^2_{MCC2} = \{(0,3),(1,2),(4,7),(5,6),(8,11),(9,10),(12,15),(13,14)\}$
$PC^3_{MCC2} = \{(0,4),(1,5),(2,6),(3,7),(8,12),(9,13),(10,14),(11,15)\}$
$PC^4_{MCC2} = \{(0,12),(1,13),(2,14),(3,15),(4,8),(5,9),(6,10),(7,11)\}$

Beispielsweise kann der Block (1,2) nicht der Klasse PC^1_{MCC2} zugeteilt werden, da das Element 1 mit dem Block (0,1) bereits in dieser Klasse vorhanden ist, so daß die nächstmögliche Klasse genommen wird.

Die zu erzeugenden Teilstrukturen (vgl. Abb. 35 auf Seite 130) verfügen über alle Knoten der Ursprungsstruktur, jedoch nur über die Hälfte der Verbindungen. Leicht verifiziert man, daß die wegfallenden Verbindungen genau diejenigen sind, die durch die Resolution des Versuchsplans $(V,\mathscr{B})_{MCC2}$ den parallelen Klassen PC^1_{MCC2} und PC^4_{MCC2} zugeteilt worden sind. Die Konstruktion der vier Teilgitter aus den parallelen Klassen PC^1_{MCC2} und PC^3_{MCC2} erfolgt ähnlich wie im Fall des Hypercube, wenn auch etwas aufwendiger. Anhand der bestehenden Ordnung $\prec$ werden die beiden gewonnenen parallelen Klassen PC^1_{MCC2} und PC^3_{MCC2} nun aufgespalten in je zwei partiell-parallele Klassen PPC^{11}_{MCC2}, PPC^{12}_{MCC2} und PPC^{31}_{MCC2}, PPC^{32}_{MCC2}, die jeweils die 4 „kleineren" und „größeren" Elemente der jeweiligen parallelen Klasse umfassen. Zwei der vier zu generierenden Teilgitter $\mathscr{B}^A$ und $\mathscr{B}^B$ werden nun aus Blöcken von PPC^{11}_{MCC2} und PPC^{12}_{MCC2} erzeugt; $\mathscr{B}^C$ und $\mathscr{B}^D$ rekrutieren ihre Blöcke aus den übrigen beiden partiell-parallelen Klassen. Während PPC^{11}_{MCC2} nun abwechselnd je einen Block an $\mathscr{B}^A$ und $\mathscr{B}^B$ zuteilt, gibt PPC^{12}_{MCC2} zunächst zwei Blöcke an $\mathscr{B}^A$ und anschließend zwei Blöcke an $\mathscr{B}^B$. $\mathscr{B}^C$ und $\mathscr{B}^D$ werden analog aus den partiell-parallelen Klassen PPC^{31}_{MCC2} und PPC^{32}_{MCC2} erzeugt. Alle Zuteilungen orientieren sich wie üblich immer an der Ordnung $\prec$ innerhalb der aufzuteilenden Struktur.

Damit ergibt sich
$\mathcal{B}^A = \{(0,1),(0,4),(1,5),(4,5)\}$,
$\mathcal{B}^B = \{(2,3),(2,6),(3,7),(6,7)\}$,
$\mathcal{B}^C = \{(8,9),(8,12),(9,13),(12,13)\}$,
$\mathcal{B}^D = \{(10,11),(10,14),(11,15),(14,15)\}$
und die Partitionierung ist abgeschlossen.

Für die 1. Stufe der Partitionierung und den speziellen Fall $n = 4$ kann zunächst einmal festgehalten werden:

- Die Resolution des MCC2 der Stufe 4 liefert 4 parallele Klassen PC^1_{MCC2}, PC^2_{MCC2}, PC^3_{MCC2}, PC^4_{MCC2}.

- Jede dieser parallelen Klassen enthält definitionsgemäß alle 16 Knoten der Gesamtstruktur, d.h.: $|PC^i_{MCC2}| = |V_{MCC2}| \quad \forall 1 \leq i \leq 4$.

- Jede dieser parallelen Klassen – dies ergibt sich aus dem vorherigen Punkt – besitzt die gleiche Zahl von (disjunkten) Blöcken, die gewissen Verbindungen im Ursprungs-MCC2 entsprechen, repräsentiert also genau ein Viertel der Verbindungen des zu partitionierenden Gitters.

- Die parallelen Klassen PC^3_{MCC2} und PC^4_{MCC2} liefern die zu eliminierenden Verbindungen.

- Um einen MCC2 der Stufe 4 auf vier Substrukturen der Stufe 2 zurückzuführen, was das Ziel des Partitionierungsvorhabens ist, werden daher mittels Resolution die Blöcke der parallelen Klassen PC^3_{MCC2} und PC^4_{MCC2} ausgesondert.

- Die vier Teilgitter bestehen nun aus den Verbindungen, die in den beiden übrigen parallelen Klassen enthalten sind.

- Die Zuordnung der Verbindungen zu den vier Teilgittern geschieht wie folgt: Es erfolgt zunächst eine Aufspaltung der beiden verbliebenen parallelen Klassen in partiell-parallele Klassen entsprechend der partiellen Ordnung $\prec$. Die partiell-parallelen Klassen mit Superscript il dienen zur Konstruktion der Teilstrukturen A und B. Die Teilgitter C und D werden aus den „höheren" partiell-parallelen Klassen gebildet. Hierbei weisen die Teilsysteme der parallelen Klasse PC^1_{MCC2} abwechselnd je einen Block den entsprechenden Teilgittern zu; die Teilsysteme der parallelen Klasse PC^2_{MCC2} vergeben die Blöcke in Gruppen von je zwei.

Weitere Bemerkungen:

- Im Gegensatz zum Hypercube ist die Zahl der parallelen Klassen, die durch die Resolution erzeugt werden, konstant und zwar gleich 4 (beim MCC2). Dies korrespondiert zu den Wiederholungsfaktoren r_x der beiden Topologien.
- Wie beim Hypercube bleiben alle Elemente des Versuchsplans der Gesamtstruktur nach der Resolution erhalten, d.h. $|\bigcup PC^i_{MCC2}| = |V_{MCC2}|$.
- Während beim Hypercube nur eine von n parallelen Klassen ausgesondert wird, sind dies beim MCC2 in der 1. Stufe der Partitionierung $2/n$ aller Verbindungen, für den Fall $n = 4$ also die Hälfte aller Verbindungen, die sich in zwei parallelen Klassen

wiederfinden. Die Partitionierung belastet daher den Mesh-Connected Computer im Verhältnis zum Cube-Connected Computer mit einem wesentlich höheren Verlust an Redundanz und damit an Fehlertoleranz.

- Die lexikographische Ordnung des Mengensystems garantiert auch hier die Eindeutigkeit der Zusammensetzung der parallelen Klassen. Ohne vorgenommene Ordnung sind auch andere Klassenzuordnungen möglich. Auch ist eine systematische Bezeichnung der Knoten im Gesamtgitter notwendig; hier wird von dem in Kapitel 3 bei der Beschreibung des MCC2 verwendeten Schema ausgegangen.
- Benötigt man im Mehrprogrammbetrieb eines (massiv-)parallelen Rechners Teilstrukturen unterschiedlicher Größe, so ist es möglich, wenn das aktuelle n gerade ist, eine oder mehrere der vier aktuell gefundenen Substrukturen weiter zu partitionieren, während die anderen Teilstrukturen intakt beibehalten werden.

Die so basierend auf der Resolution von Versuchsplänen konstruierte Partitionierung des MCC2 erweist sich als wesentlich komplexer als im Fall des Hypercube. Die Komplexität nimmt zu, wenn man vom speziellen Fall $n = 4$ auf den allgemeinen Fall übergeht. Hinzu kommt, daß bei dieser Vorgehensweise die 1. Stufe der Partitionierung gesondert betrachtet werden muß, da die Außenmaschen wegfallen. Daher wird nun eine auf Restklassen (vgl. Anhang B) basierende Partitionierungsmethode vorgestellt, die diese Nachteile vermeidet.

Beispiel 6.6

Untersucht wird erneut die Partitionierung des MCC2 der Stufe 4, dessen Versuchsplan bereits in Beispiel 6.5 gegeben ist. Die Restklasseneinteilung der Menge V_{MCC2} liefert die folgenden 4 Restklassen modulo 4: $RC_1 = \{0,4,8,12\}$ $RC_2 = \{1,5,9,13\}$ $RC_3 = \{2,6,10,14\}$ $RC_4 = \{3,7,11,15\}$
Die auf "$<$" basierende lineare Ordnung $\prec$ erzeugt in V die Ordnung: (0,1,2, ..., 15). Dies überträgt sich auf die Restklassen und resultiert in der oben notierten Anordnung. Eine Aufteilung der Restklassen in zwei Hälften unter Beachtung der Ordnung liefert:
$RC_1^1 = (0,4)$, $RC_1^2 = (8,12)$, $RC_2^1 = (1,5)$, $RC_2^2 = (9,13)$, $RC_3^1 = (2,6)$, $RC_3^2 = (10,14)$, $RC_4^1 = (3,7)$, $RC_4^2 = (11,15)$. Nun werden 4 Mengen $X_1, ..., X_4$ gebildet vermöge:
$X_1 = RC_1^1 \cup RC_2^1 = (0,1,4,5)$
$X_2 = RC_3^1 \cup RC_4^1 = (2,3,6,7)$
$X_3 = RC_1^2 \cup RC_2^2 = (8,9,12,13)$
$X_4 = RC_3^2 \cup RC_4^2 = (10,11,14,15)$
Zur Konstruktion der vier gewünschten Partitionen R_A, R_B, R_C und R_D leitet man den Versuchsplan $(V,\mathcal{B})_{MCC2}$ nach folgenden Mengen ab:
$R_A = (V,\mathcal{B})_{MCC2} \setminus (X_2 \cup X_3 \cup X_4)$
$R_B = (V,\mathcal{B})_{MCC2} \setminus (X_1 \cup X_3 \cup X_4)$
$R_C = (V,\mathcal{B})_{MCC2} \setminus (X_1 \cup X_2 \cup X_4)$
$R_D = (V,\mathcal{B})_{MCC2} \setminus (X_1 \cup X_2 \cup X_3)$
Die Partitionierung des MCC2 der Stufe 4 ist damit abgeschlossen.

Die Erweiterung auf den allgemeinen Fall mit gerader Ausbaustufe n führt zu Algorithmus 8.

Algorithmus 8: Partitionierung eines MCC2 der Stufe n in 4 Teilgitter der Stufe $n/2$

1. Konstruiere den Versuchsplan $(V, \mathscr{B})_{MCC2}$, der das zweidimensionale Gitter beschreibt
2. Zerlege V_{MCC2} in Restklassen modulo n. Es entstehen n Restklassen $R_1,\ldots, R_n$.
3. Führe die lineare Ordnung $\prec$ in die Menge V_{MCC2} und damit auch in die Restklassen ein.
4. Teile jede Restklasse R_i anhand der Ordnung $\prec$ in eine „obere" R_i^1 und „untere" Hälfte R_i^2 auf.
5. Konstruiere 4 Mengen $X_1,\ldots, X_4$ vermöge $X_1 = \bigcup_{1 \le i \le n/2} R_i^1$, $X_2 = \bigcup_{1 \le i \le n/2} R_i^2$, $X_3 = \bigcup_{n/2 \le i \le n} R_i^1$, $X_4 = \bigcup_{n/2 \le i \le n} R_i^2$.
6. Zur Konstruktion der vier gewünschten Partitionen R_A, R_B, R_C und R_D leite den Versuchsplan $(V,\mathscr{B})_{MCC2}$ nach folgenden Mengen ab:
   $R_A = (V,\mathscr{B})_{MCC2} \setminus (X_2 \cup X_3 \cup X_4)$, $R_B = (V,\mathscr{B})_{MCC2} \setminus (X_1 \cup X_3 \cup X_4)$,
   $R_C = (V,\mathscr{B})_{MCC2} \setminus (X_1 \cup X_2 \cup X_4)$, $R_D = (V,\mathscr{B})_{MCC2} \setminus (X_1 \cup X_2 \cup X_3)$
7. Die Partitionierung ist damit abgeschlossen.

Mit der Setzung $N = v$ für die Knotenzahl liefert die Komplexitätsanalyse von Algorithmus 8 folgende Ergebnisse: Die Versuchsplandarstellung wird kostenlos bereitgestellt. Die Zerlegung in Restklassen benötigt $O(N)$ Divisionsoperationen. Das Einführen der linearen Ordnung $\prec$ in die Menge V erfordert $O(N \log N)$ Sortieroperationen. Die Aufteilung in obere und untere Hälften geschieht durch Beachtung des Abspeicherungsmusters kostenfrei. Die Konstruktion der 4 Mengen $X_1, \ldots, X_4$ erfordert $4 \times n/2 = 2n$ Vereinigungsoperationen. Diese erfolgen mit dem *Union-Find*-Algorithmus mit Pfadkompression [AHU,74] und benötigen dann insgesamt $2 \times O(nG(n))$ Operationen, wobei $G(n)$ für praxisrelevante Größenordnungen als < 5 angenommen werden kann [AHU,74]. Die erforderlichen 4 Ableitungen von Versuchsplänen, erfordern zunächst noch einmal 8 Vereinigungsoperationen mit konstantem Aufwand. Zur eigentlichen Ableitung müssen alle Blöcke angesehen werden, was $2N$ Operationen erfordert. Mit dem Zusammenhang $N = n^2$ ergibt sich insgesamt die Komplexität:
$O(N) + O(N \log N) + O(\sqrt{N}\, G(\sqrt{N})) + O(1) + O(N) = O(N \log N)$

Bemerkungen:

- Durch die Ableitung von Versuchsplänen erfolgt die Herausnahme der entsprechenden Verbindungen stets korrekt; die erste Partitionsstufe, in der die Außenmaschen verloren gehen, muß daher im Gegensatz zum Resolutionsverfahren nicht gesondert behandelt werden.
- Ist die Stufenzahl eines entstandenen Subsystems wiederum gerade, so kann die Partitionierung in diesem Subsystem fortgesetzt werden und bei gegebenem geraden n sukzessive weiter bis das reduzierte System der Prozessoranforderung entspricht.

6.2.1 Einbettung in Parallelrechnern mit statischem Verbindungsnetzwerk

In Kapitel 5 wurde gezeigt, daß bestimmte parallele Algorithmen oder Klassen paralleler Algorithmen zu bestimmten Topologien für Parallelrechner ideal korrespondieren. Standardbeispiel ist hier die Abbildung einer Radix-2-FFT auf einen Cube-Connected Computer oder auch die Abbildung der Klasse der Divide-and-Conquer-Algorithmen auf Binärbaumarchitekturen. In der Praxis tritt nun häufig der Fall ein, daß für einen zu implementierenden Algorithmus weder dessen ideale Zieltopologie noch ein rekonfigurierbares verteiltes System zur Verfügung steht. In dieser Situation wäre es dann vorteilhaft, wenn das Verbindungsnetzwerk des zur Verfügung stehenden Multiprozessorrechners die ideale Zieltopologie nachbilden könnte. Somit wäre dann die Zuordnung der Module des Algorithmus zu den Prozessoren des Implementierungsrechners bestimmt und die optimale Kommunikationskomplexität könnte realisiert werden. Die Fachliteratur bezeichnet diese Problemstellung auch als **Einbettung** (*embedding*) einer Architektur A in eine Architektur B.

Hier soll im folgenden nur eine 1-1-Prozessorzuordnung berücksichtigt werden, nicht jedoch die Ersetzung mehrerer Prozessoren der gewünschten Topologie durch einen (1) Prozessor der einbettenden Topologie. Dies impliziert, daß die Zahl der Prozessoren des einbettenden Systems größer oder gleich der Zahl der Prozessoren des einzubettenden Systems sein muß. Weiterhin soll unter Einbettung einer Struktur A in eine Struktur B nur der Fall verstanden werden, bei der direkt verbundene Prozessoren der Struktur A auf direkt verbundene Prozessoren der einbettenden Struktur B abgebildet werden. Für die Thematik dieser Arbeit reicht diese Vereinfachung aus, da nur prinzipielle Betrachtungen angestellt werden sollen. In der Literatur wird darüber hinaus auch eine solche Einbettung akzeptiert, bei der direkt verbundene Knoten des gewünschten Rechners durch nicht benachbarte Knoten des einbettenden Rechners ersetzt werden. Dabei wird jedoch darauf geachtet, daß der Abstand entsprechender Knoten relativ klein ist. Der hierfür eingeführte Parameter ist die sogenannte **Kommunikationsverzögerung** (*dilation*). Sie bezeichnet den in einem einbettenden Rechner bestehenden Maximalwert für die Distanz zweier Knoten (vgl. Kapitel 5), die im einzubettenden Rechner direkt verbunden sind. Hätten beispielsweise Paare von Knoten, die im gewünschten Rechner benachbart sind, im einbettenden Rechner die jeweiligen Distanzen 1, 2 und 3, so würde man von Kommunikationsverzögerung 3 sprechen.

Weitere Kenngrößen für die Beurteilung der Güte einer Einbettung sind die Ausweitung der Zahl der Prozessoren beim Übergang auf das einbettende System. Dieser mit **Expansion** bezeichnete Parameter sollte möglichst gering ausfallen; diese Minimierung zielt auf die Partitionierung eines Gesamtsystems in Teilsysteme gleicher Struktur für den Fall des Mehrprogrammbetriebs. Die Expansion wird bestimmt durch die Division der Zahl der Prozessoren des einbettenden Rechners durch die Knotenzahl des einzubettenden Rechners. Hat der tatsächlich genutzte Rechner, oder die davon erforderliche Partition, die doppelte Zahl an Prozessoren wie der eigentlich gewünschte Rechner, so beträgt die Expansion 2. Naturgemäß sind die Parameter Kommunikationsverzögerung und Expansion nicht unkorreliert. Die für die vorliegende Arbeit getroffene Restriktion auf Kommunikationsverzögerung 1 kann unter Umständen eine größere Expansion erfor-

derlich machen. Ein weiterer üblicher Parameter ist der **Lastfaktor** [Wag,89]. Dieser ergibt sich aus dem Verhältnis der Zahl der Verbindungen, bei kombinatorischer Sichtweise also der Zahl der Blöcke, die der gewünschte Rechner besitzt, und der Zahl der Blöcke, die für dessen Einbettung benötigt werden. Erzwingt man, wie hier angenommen, Kommunikationsverzögerung 1, also die Simulation benachbarter Knoten durch benachbarte Knoten, so beträgt auch der Lastfaktor konstant 1.

6.2.2 Einbettung durch Emulation kombinatorischer Versuchspläne

Aus Gründen, die bereits in Abschnitt 6.1 erwähnt wurden, ist der Hyperwürfel eine interessante Architektur zur Einbettung anderer Topologien. Die Einbettung von Bäumen, maschenverbundenen Rechnern und Pyramiden in Hyperwürfel wird von Monien und Sudborough behandelt [M&S,88]. Wu, Wagner, Bier und Loe sowie Fang und Chen beschränken sich auf Binärbäume ([Wu_,85], [Wag,89], [B&L,89], [F&C,91]). Zweidimensionale Gitter werden von Chan und Chin [C&C,88], dreidimensionale von Liu und Huang [L&H,91] in den Hypercube eingebettet; Ho und Johnsson lassen darüber hinaus auch zyklisch über den Rand geschlossene Verbindungen zu [H&J,90a]. Krishnakumar, Hegde und Iyengar untersuchen die Einbettung der besonders in der Bildverarbeitung wichtigen Struktur des *quadtree* [KHI,91]. Greenberg und Bhatt betrachten sowohl die Einbettung von Gittern als auch von Bäumen in den Hypercube, wobei sie darauf abzielen, möglichst viele der vorhandenen Verbindungen zu nutzen [G&B,91]. Zusätzlich zu Gittern werden von Saad und Schultz [S&S,88] Ringe als Einbettungsobjekte analysiert, die auch in der vorliegenden Arbeit neben sternförmig verknüpften Rechnern in den Hypercube eingebettet werden sollen.

6.2.2.1 Einbettung von ringförmigen Rechnern in den Hypercube

Beispiel 6.7

Ein Ring der Stufe 4 wird durch folgenden Versuchsplan $(V,\mathcal{B})_{RCC-4}$ repräsentiert:
$V = \{0,1,2,3\}$
$\mathcal{B} = \{(0,1),(1,2),(2,3),(3,0)\}$
$v = 4, b = 4, r_x = 2\ \forall x \in V$
$\Lambda_2 = \{\lambda(s): |S| = 2, S \subseteq V\} = \{0,1\}$

Der kleinste Hyperwürfel, der das erste Kriterium erfüllt, mindestens ebensoviele Knoten zu besitzen wie die einzubettende Ringstruktur, ist der Hyperwürfel der Dimension 2 mit ebenfalls 4 Knoten, den man als eine Art „Quadrat" auffassen kann. Die Einbettungsmöglichkeit in den Hypercube ist hier sofort einsichtig: Während sich in der graphischen Darstellung scheinbar unterschiedliche Formen ergeben, sind die Versuchspläne beider Netzwerke identisch.

Beispiel 6.8

Offensichtlich sind Ringe mit einer ungeraden Zahl von Knoten bei den vorgegebenen Prämissen nicht zur Einbettung geeignet. Daher soll als nächstes ein Ring der Stufe 8 eingebettet werden. Dieser wird durch folgenden Versuchsplan $(V,\mathscr{B})_{RCC-8}$ spezifiziert:
$V = \{0,1,2,3,4,5,6,7\}$
$\mathscr{B} = \{(0,1),(1,2),(2,3),(3,4),(4,5),(5,6),(6,7),(7,0)\}$
$v = 8, b = 8, r_x = 2\ \forall x \in V$
$\Lambda_2 = \{\lambda(s): |S| = 2, S \subseteq V\} = \{0,1\}$

Der einbettende Hypercube muß mindestens 8 Knoten enthalten, um diesen Ring einbetten zu können. Der kleinste Hypercube, der dieser Anforderung genügt, ist der CCC der Dimension 3, der durch folgenden Versuchsplan $(V,\mathscr{B})_{CCC-3}$ repräsentiert wird.
$V = \{0,1,2,3,4,5,6,7\}$
$\mathscr{B} = \{(0,1),(1,2),(2,3),(3,0),(0,7),(1,6),(2,5),(3,4),(4,5),(5,6),(6,7),(7,4)\}$
$v = 8, b = 12, r_x = 3\ \forall x \in V$
$\Lambda_2 = \{\lambda(s): |S| = 2, S \subseteq V\} = \{0,1\}$

Aus den Konstruktionsprinzipien eines Hypercube ist bekannt, daß, wenn man die Knotenadressen 0, ..., 7 in den Gray-Code überträgt, gerade diejenigen Prozessoren miteinander verbunden sind, die in genau einem Bit differieren. Wählt man nun die ersten 8 Code-Worte – der ausreichenden Länge 3 – aus, so sind dies: 000, 001, 011, 010, 110, 111, 101, 100 entsprechend den Ziffern 0,...,7. Diese Ziffern, die mit den Ringadressen übereinstimmen, bilden also bereits eine Kette. Da sich, wie man sieht, auch die Adressen 0 und 7 in nur einem Bit unterscheiden, kann die Kette zum Ring geschlossen werden und die Einbettung ist abgeschlossen. Die übrigen vier im Hypercube verfügbaren Verbindungen werden nicht benötigt. Bezüglich der Zahl der zur Einbettung notwendigen Prozessoren ergibt sich für diesen konkreten Fall eine optimale Einbettung; ein 8-Prozessor-System wurde in ein 8-Prozessor-System eingebettet. Dies ergibt den bestmöglichen Expansionswert 1.

Beispiel 6.9

Es leuchtet unmittelbar ein, daß dies für eine beliebige (gerade) Anzahl n von Knoten im Ring nicht immer der Fall sein kann, da Hypercubes nur für Prozessorzahlen existieren, die Zweierpotenzen sind. Demnach kann die Einbettung eines Ringes mit 6 Knoten bestenfalls in einen Hypercube der Dimension 3, also mit 8 Prozessoren erfolgen. Für diese tatsächlich mögliche Einbettung beträgt die Expansion dann 4/3.

Faßt man einbettendes und einzubettendes Netzwerk als kombinatorische Versuchspläne auf, so resultiert die Frage der Einbettungsmöglichkeit in der Überprüfung der Tatsache, ob alle Blöcke des zu emulierenden Versuchsplans im Emulator-Versuchsplan enthalten sind. Hierbei ist eine Umbenennung der Knoten des Emulators mittels eines Endomorphismus gestattet. Für das Beispiel der eben angesprochenen Emulation eines Ringes mit 6 Prozessoren durch einen Hypercube der Dimension 3 kann man einen Sechserring im Hypercube bilden mit den Knoten {0,1,2,5,6,7}. Dieser besteht aus den Verbindungen {(0,1),(1,2),(2,5),(5,6),(6,7),(7,0)}. Um mit diesen Verbindungen des

Hypercube den Ring nachbilden zu können und eine Überprüfung durch Vergleich der kombinatorischen Versuchspläne zu ermöglichen, dient folgender Endomorphismus auf den Knoten des Hypercube:
$f: = \{(0\mapsto 0),(1\mapsto 1),(2\mapsto 2),(3\mapsto 6),(4\mapsto 7),(5\mapsto 3),(6\mapsto 4),(7\mapsto 5)\}$

Nach Anwendung des Endomorphismus f hat das Mengensystem des Hypercube folgende Gestalt:
$\bar{\mathscr{B}}_{CCC-3} = \{(0,1),(1,2),(2,6),(6,0),(0,5),(1,4),(2,3),(6,7),(7,3),(3,4),(4,5),(5,7)\}$

Der verwendete Endomorphismus kann auch als Permutation der Menge V_{CCC} angesehen werden. Die Topologie des Hypercube bleibt hierbei erhalten, nur die dezimalen Knotenadressen ändern sich. Dies kann man auch als virtuelle Adressierung zum Zwecke der Emulation auffassen. Abb. 36 zeigt den Hypercube mit den virtuellen und realen Adressen. Anhand der virtuellen Adressen läßt sich nun leicht der Verlauf des in den Hypercube eingebetteten Ringes (0-1-2-3-4-5) nachverfolgen.

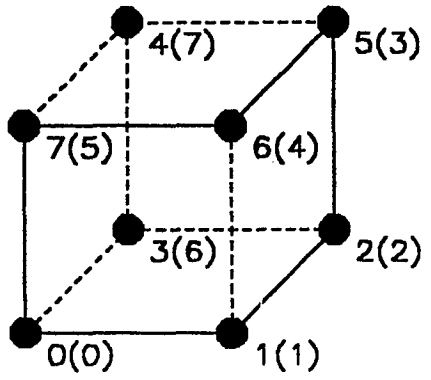


Abb. 36. Einbettung eines 6-Knoten-Ringes in einen Hypercube: Ziffern in Klammern bezeichnen die virtuellen Adressen. Die durchgezogene Linie skizziert die zur Einbettung verwendeten Verbindungen.

Leicht verifziert man, daß auch andere Positionen des Ringes im Hypercube möglich sind. Dieser Tatbestand kann bei größeren zur Verfügung stehenden Emulatoren dafür ausgenutzt werden, um mehrere Ringe, oder andere Topologien, gleichzeitig zu emulieren.

Die Frage, ob ein bestimmtes Netzwerk A in ein bestimmtes Netzwerk B eingebettet werden kann, kann man auch mit kombinatorischen Mitteln lösen. Falls eine Permutation der Elemente des Versuchsplans des Emulators existiert, so daß der sich durch die Permutation ergebende Versuchsplan $(\overline{V,\mathscr{B}})$ den Versuchsplan des einzubettenden Versuchsplans emuliert (vgl. Anhang A) so ist die Einbettung möglich. Die Einbettung statischer Netzwerke kann also auf die Emulation der zugehörigen Versuchspläne zurückgeführt werden. Ein Versuchsplan $(V,\mathscr{B})_B$ emuliert dabei einen Versuchsplan $(V,\mathscr{B})_A$ (strikt), wenn alle in $\mathscr{B}_A$ enthaltenen Blöcke auch in $\mathscr{B}_B$ enthalten sind. Permutationen der Menge V_B sind dabei zugelassen. Faßt man Mengensysteme als Mengen und Blöcke als deren Elemente auf, so kann dieser Zusammenhang auch durch die Teilmengenbeziehung $\mathscr{B}_A \subseteq \bar{\mathscr{B}}_B$ charakterisiert werden.

Auf das obige Beispiel der Einbettung des 6-Knoten-Ringes in den dreidimensionalen Hypercube angewendet bedeutet dies:
$\Pi := \{(0\mapsto 0),(1\mapsto 1),(2\mapsto 2),(3\mapsto 6),(4\mapsto 7),(5\mapsto 3),(6\mapsto 4),(7\mapsto 5)\}$

Die hier angewendete Permutation Π entspricht dem 5-Zyklus (3 6 4 7 5). Leicht verifiziert man, auch durch Betrachten der Abb. 36, daß die Transposition (5 7) den gleichen Zweck erfüllt hätte. Somit ergibt sich:
$\mathscr{B}_{RCC-6} = \{(0,1),(1,2),(2,3),(3,4),(4,5),(5,0)\}$
$\mathscr{B}_{CCC-3} = \{(0,1),(1,2),(2,3),(3,0),(0,7),(1,6),(2,5),(3,4),(4,5),(5,6),(6,7),(7,4)\}$
$\overline{\mathscr{B}}_{CCC-3} = \{(0,1),(1,2),(2,6),(6,0),(0,5),(1,4),(2,3),(6,7),(7,3),(3,4),(4,5),(5,7)\}$

Es gilt nun:

$$\forall B_i \in \mathscr{B}_{RCC-6}\ \exists B_j \in \overline{\mathscr{B}}_{CCC-3} \text{ mit } B_i = B_j \Rightarrow (V,\mathscr{B})_{RCC-6} \lesssim \overline{(V,\mathscr{B})}_{CCC-3}$$

Eine Permutation Π bildet einen Versuchsplan stets auf einen äquivalenten Versuchsplan ab:

$$\overline{(V,\mathscr{B})}_{CCC-3} = \Pi((V,\mathscr{B})_{CCC-3}) \Rightarrow \overline{(V,\mathscr{B})}_{CCC-3} \approx (V,\mathscr{B})_{CCC-3},$$

so daß die Emulation über den Zwischenschritt der Permutation deutlich wird:

$$\overline{(V,\mathscr{B})}_{CCC-3} \approx (V,\mathscr{B})_{CCC-3} \wedge (V,\mathscr{B})_{RCC-6} \lesssim \overline{(V,\mathscr{B})}_{CCC-3} \Rightarrow (V,\mathscr{B})_{RCC-6} \lesssim (V,\mathscr{B})_{CCC-3}.$$

Die Ausweitung auf den allgemeinen Fall führt zu folgendem

Lemma: In einem Hypercube der Dimension n sind Ringe mit gerader Knotenzahl bis zu 2^n emulierbar.

Beweis: Betrachte den Aufbau des Gray-Code (vgl. Anhang B): Der Übergang vom Gray-Code der Stufe n zum Gray-Code der Stufe $n+1$, und damit der Übergang von der Adressierung des n-dimensionalen Hypercube zur Adressierung des $n+1$-dimensionalen Hypercube, erfolgt durch eine Art Spiegelung:

Stufe n

$$\begin{matrix} a_{0,n-1} & \cdots & a_{0,0} \\ \vdots & & \vdots \\ a_{n-1,n-1} & \cdots & a_{n-1,0} \end{matrix}$$

Stufe n+1

$$\begin{matrix} 0a_{0,n-1} & \cdots & a_{0,0} \\ \vdots & & \vdots \\ 0a_{n-1,n-1} & \cdots & a_{n-1,0} \\ \hline 1a_{n-1,n-1} & \cdots & a_{n-1,0} \\ \vdots & & \vdots \\ 1a_{0,n-1} & \cdots & a_{0,0} \end{matrix}$$

Wird nun ein Ring mit gerader Knotenzahl $2N \leq 2^{n+1}$ emuliert durch die $2N$ fortlaufenden Knotenadressen des Hypercube, $A_1, ..., A_{2N}$, von denen sich jeweils N oberhalb $(A_1, ..., A_N)$, bzw. unterhalb $(A_{N+1}, ..., A_{2N})$ der eingezeichneten imaginären Spiegelachse befinden, so ergibt sich folgendes:

- Da aufeinanderfolgende Zahlen im Gray-Code sich nur in je einem Bit unterscheiden, unterscheiden sich die Zahlenpaare $(A_1, A_2),(A_2, A_3), ...,(A_{N-1}, A_N),(A_N, A_{N+1}), ...,(A_{2N-1}, A_{2N})$ nur in jeweils einem Bit, d.h. die Knoten mit diesen Adressen sind im Hypercube direkt verbunden.

- Da die Adresse A_{2N} durch Spiegelung aus der Adresse A_1 hervorgegangen ist, unterscheidet sich dieses Paar ebenfalls nur in einem Bit, nämlich dem am weitesten links befindlichen, die entsprechenden Knoten haben also eine direkte Verbindung.

$\Rightarrow$ Werden die Adressen A_i, $1 \leq i \leq 2N$ entsprechend dieser Vorschrift gewählt, dann liefert die Permutation Π mit $\Pi(A_i) = i - 1$ für Ringe mit gerader Knotenzahl bis zum Maximalwert $2N = 2^{n+1}$ im Hypercube der Dimension $n+1$ virtuelle Adressen für direkt verbundene Knoten in der Form $0, 1, ..., 2N-1$, d.h. die entsprechenden Ringe sind emulierbar.

q.e.d.

Es kann also festgehalten werden:

- In einem Hypercube der Dimension n sind Ringe mit gerader Knotenzahl bis zu 2^n einbettbar.

- Die Expansion beträgt hierbei im günstigsten Fall 1, im ungünstigsten Fall $2^n/(2^{n-1}+2)$

- Basierend auf dem Aufbau des Gray-Code (vgl. Anhang B) sind im Hypercube der Dimension $n+1$ Ringe bis zur Knotenzahl 2^n in gleicher Weise emulierbar wie beim Hypercube der Dimension n. Dies geschieht durch Übergang von Knotenadressen der Form $a_{i,0}...a_{i,n-1}$ zu Adressen der Form $0a_{i,0}...a_{i,n-1}$. Da man im $n+1$-dimensionalen Hypercube auch die Adressen der Form $1a_{i,0}...a_{i,n-1}$ hätte wählen können, bestehen in der $n+1$-dimensionalen Struktur doppelt so viele Möglichkeiten zur Einbettung von Ringen bis zur Knotenzahl 2^n.

- Aus der n-fachen Symmetrie des Hypercube ergibt sich die Konsequenz, daß zur Einbettung der Ringe verschiedene Positionen denkbar sind. Läßt man Adressenumbenennungen außer Betracht, so sind in einem Hypercube der Dimension 3 vier Positionierungen für einen einzubettenden Ring der Größe 8 möglich, für einen 4-Knoten-Ring sind es sechs unterschiedliche Positionen.

- Damit die hier geforderte durchlaufende Numerierung der Knoten im eingebetteten Ring gewährleistet ist, muß eventuell ein Automorphismus bzw. eine Permutation auf der Knotenmenge des Hypercube erfolgen, um den Ring zu schließen.

- Da auch eine identische Abbildung als Permutation aufgefaßt werden kann, kann aus Gründen einer systematischen Vorgehensweise stets eine Permutation vorgenommen werden.

- In der praktischen Anwendung wird man im allgemeinen zuerst eine Partitionierung des Hypercube-Gesamtsystems in Subcubes hinreichender Größe durchführen, um die übrigen Subcubes für weitere Anforderungen zur Verfügung zu haben.

Poplawski zeigt, daß für den Fall des Hypercube der T-Serie der Firma Floating Point Systems, die hier genutzte Technik nicht anwendbar ist, da in diesem speziellen Rechner bestimmte Verbindungen nicht gleichzeitig genutzt werden können [Pop,88]. Unter Zuhilfenahme des konstruktiven Beweises für das Lemma resultiert die Vorgehensweise zur kombinatorischen Einbettung in Algorithmus 9.

Algorithmus 9: Emulation von Ringen mit gerader Knotenzahl im Hypercube

1. Erzeuge den Versuchsplan des zu emulierenden Ringes $(V,\mathscr{B})_{RCC}$ wie in Kapitel 3 beschrieben.

2. Sei ein Ring mit gerader Knotenzahl N gegeben; wähle den kleinsten Hypercube aus, dessen Knotenzahl größer oder gleich der Zahl der Knoten des Ringes ist, und erzeuge dessen Versuchsplan $(V,\mathscr{B})_{CCC}$.

3. Definiere eine Permutation Π auf der Knotenmenge des Hypercube V_{CCC} nach dem im Beweis beschriebenen Prinzip, so daß sich ein Ring der gewünschten Knotenzahl schließt.

   $$\overline{(V,\mathscr{B})}_{CCC} = \Pi((V,\mathscr{B})_{CCC})$$

4. Die Permutation liefert virtuelle Adressen für die Knoten des Hypercube.

5. Der emulierte Ring kann jetzt im Hypercube durch Lokalisierung der virtuellen Adressen $0, \ldots, n-1$ realisiert werden.

   $$(\overline{(V,\mathscr{B})}_{CCC} \approx (V,\mathscr{B})_{CCC} \wedge \overline{(V,\mathscr{B})}_{CCC} \gtrsim (V,\mathscr{B})_{RCC}) \Rightarrow (V,\mathscr{B})_{CCC} \gtrsim (V,\mathscr{B})_{RCC}$$

6.2.2.2 Einbettung von sternförmigen Rechnern in den Hypercube

Zieht man zur Beschreibung eines sternförmigen Rechners dessen kombinatorische Spezifikation aus Kapitel 3 heran, so erkennt man, daß sich die Ausbaustufe des Star-Connected Computer (STC) im Wiederholungsfaktor r_x des Zentralknotens wiederfindet. Andererseits ist der Wiederholungsfaktor r_x im Hypercube – für alle Knoten – gleich n, so daß bereits aus dieser Tatsache deutlich wird, daß zur Einbettung eines sternförmigen Rechners der Stufe n, d.h. mit n Ästen, ein Hypercube der Dimension n erforderlich ist.

Beispiel 6.10

Gesucht ist eine Einbettung eines STC der Stufe 4 in einem minimal dimensionierten Hypercube hinreichender Größe. Ein STC der Stufe 4 wird spezifiziert durch folgenden kombinatorischen Versuchsplan $(V,\mathcal{B})_{STC-4}$:
$V = \{0,1,2,3,4\}$
$\mathcal{B} = \{(0,1),(0,2),(0,3),(0,4)\}$
$v = 5$, $b = 4$, $r_1 = 4$ für das Zentrum, $r_2 = 1$ für die peripheren Knoten
$\Lambda_2 = \{\lambda(s): \mid S \mid = 2, S \subseteq V\} = \{0,1\}$

Der notwendige Wiederholungsfaktor 4, wenn auch nur für einen einzelnen Knoten, bedingt einen Hypercube der Dimension 4, also mit 16 Knoten. Dieser wird durch folgenden Versuchsplan beschrieben:
$V = \{0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15\}$
$\mathcal{B} = \{(0,15),(1,14),(2,13),(3,12),(4,11),(5,10),(6,9),(7,8),(0,7),(1,6),(2,5),(3,4),(8,15),(9,14),$
$(10,13),(11,12),(0,3),(1,2),(4,7),(5,6),(8,11),(9,10),(12,15),(13,14),(0,1),(2,3),(4,5),(6,7),$
$(8,9),(10,11),(12,13),(14,15)\}$
$v = 16$, $b = 32$
$r_x = 4 \; \forall x \in V$
$\Lambda_2 = \{\lambda(s): \mid S \mid = 2, S \subseteq V\} = \{0,1\}$

Der hohe Preis der notwendigen enormen Expansion 16/5 wird etwas relativiert durch die Einfachheit der durchzuführenden Permutation auf den Knoten des Emulators. Im STC ist der o.B.d.A. mit der Adresse 0 versehene Zentralknoten mit allen übrigen Knoten der Topologie verbunden. Der Knoten mit der Adresse 0 im Hypercube ist, wie auch alle übrigen Knoten innerhalb dieser Architektur, mit denjenigen Knoten verbunden, deren Binäradresse sich in genau einem Bit unterscheidet, die also numerisch um eine Zweierpotenz differieren. Die erforderliche Permutation muß diese Kommunikationsstruktur auf die des Sterns abbilden.

Für das konkrete Beispiel ergibt sich: $\Pi \mid_{\{0,1,3,7,15\}} = \{(0 \mapsto 0),(1 \mapsto 1),(3 \mapsto 2),(7 \mapsto 3),(15 \mapsto 4)\}$
Dies kann zu einer Permutation auf V vervollständigt werden, beispielsweise zu dem 5-Zyklus (2 15 4 7 3). Betrachtet man das Mengensystem $\mathcal{B}_{CCC-4}$ des Hypercube nach Durchführung der Permutation, so entstehen aus den Verbindungen $\{(0,3),(0,7),(0,15)\}$ die Verbindungen $\{(0,2),(0,3),(0,4)\}$. Die darüber hinaus benötigte Verbindung (0,1) war bereits im ursprünglichen Hypercube vorhanden und wird durch die Permutation in sich selbst überführt. Somit enthält der Hypercube alle notwendigen Verbindungen und die Emulationsmöglichkeit des sternförmigen Rechners der Stufe 4 ist nachgewiesen.

Es gilt nun:

$$\forall B_l \in \mathcal{B}_{STC-4} \; \exists B_J \in \overline{\mathcal{B}}_{CCC-4} \text{ mit } B_l = B_J \Rightarrow (V,\mathcal{B})_{STC-4} \lesssim \overline{(V,\mathcal{B})}_{CCC-4}$$

Eine Permutation Π bildet einen Versuchsplan stets auf einen äquivalenten Versuchsplan ab:

$$\overline{(V,\mathcal{B})}_{CCC-4} = \Pi((V,\mathcal{B})_{CCC-4}) \Rightarrow \overline{(V,\mathcal{B})}_{CCC-4} \approx (V,\mathcal{B})_{CCC-4},$$

so daß die Emulation über den Zwischenschritt der Permutation deutlich wird:

$$\overline{(V,\mathscr{B})}_{CCC-4} \approx (V,\mathscr{B})_{CCC-4} \wedge (V,\mathscr{B})_{STC-4} \lesssim \overline{(V,\mathscr{B})}_{CCC-4} \Rightarrow (V,\mathscr{B})_{STC-4} \lesssim (V,\mathscr{B})_{CCC-4}.$$

Es kann also festgehalten werden:

- Zur Emulation eines sternförmigen Rechners mit n Ästen ist ein Hypercube der Dimension n erforderlich.
- Diese Erfordernis bedingt eine sehr große Expansion. Sie beträgt $2^n/(n+1)$.
- Benötigt wird eine Permutation, die Adressen der Form $2^i - 1$ auf Adressen der Form i abbildet.
- Die Verbindung (0,1) bleibt durch die Permutation unverändert.

Dies führt allgemein zu dem in Algorithmus 10 beschriebenen Vorgehen:

Algorithmus 10: Einbettung von sternförmigen Rechnern im Hypercube

1. Erzeuge den Versuchsplan des einzubettenden sternförmigen Rechners $(V,\mathscr{B})_{STC}$ wie in Kapitel 3 beschrieben.
2. Zur Einbettung eines Star-Connected Computer der Stufe n wird ein Hypercube der Dimension n benötigt. Erzeuge dessen Versuchsplan $(V,\mathscr{B})_{CCC}$.
3. Generiere eine Abbildung Π vermöge folgender Vorschrift: $\Pi|_{(1,3,7,15,\ldots,2^n-1)} := \{2^i - 1 \mapsto i \mid i = 1,..,n\}$. Ein solches Π kann stets zu einer Permutation vervollständigt werden.

   $$\overline{(V,\mathscr{B})}_{CCC} = \Pi((V,\mathscr{B})_{CCC})$$

4. Die Permutation liefert virtuelle Adressen für die Knoten des Hypercube.
5. Der emulierte sternförmige Rechner kann jetzt im Hypercube durch Lokalisierung der Verbindungen (0,1),...,(0,n) realisiert werden.

   $$(\overline{(V,\mathscr{B})}_{CCC} \approx (V,\mathscr{B})_{CCC} \wedge \overline{(V,\mathscr{B})}_{CCC} \gtrsim (V,\mathscr{B})_{STC}) \Rightarrow (V,\mathscr{B})_{CCC} \gtrsim (V,\mathscr{B})_{STC}$$

Weitere Bemerkungen:

- In einem Hypercube der Dimension n können simultan zwei Star-Connected Computer der Stufe n konfliktfrei emuliert werden. Hierbei ist die Position des ersten STC – bezogen auf dessen Zentralknoten – unter den 2^n Knoten des Hypercube frei wählbar.
- Im Hypercube der Dimension 3 muß der Zentralknoten des zweiten zu emulierenden STC dann den Platz an dem (einzigen) Knoten einnehmen, dessen Entfernung zum

Zentralknoten des ersten STC genau 3 beträgt; beide Zentralknoten besitzen also maximale Distanz (vgl. Abb. 37).

- Die notwendige Dimension des emulierenden Hypercube kann man unter Berücksichtigung der Ergebnisse aus dem vorherigen Abschnitt betreffend die Partitionierung auch daraus ableiten, daß ein Star-Connected Computer nicht in parallele Klassen zerlegbar ist, sondern nur aufteilbar in partiell-parallele Klassen, deren, für Stufe n, n benötigt werden.
- Die Emulation q-närer Bäume kann man, beginnend mit Höhe 1, auf die Emulation sternförmiger Rechner der Stufe q zurückführen.

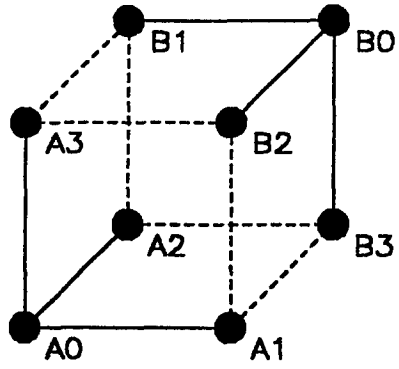


Abb. 37. Einbettung von zwei STC-3 in einen Hypercube: Die Adressen A0, ..., A3 bezeichnen die zur Einbettung des einen, die Adressen B0, ..., B3 die zur Einbettung des anderen STC-3 verwendeten Knoten des Hypercube. Die durchgezogenen Linien skizzieren die zur Einbettung verwendeten Verbindungen.

7 Entwicklung kombinatorisch motivierter Routing-Verfahren

7.1 Routing in Multiprozessorrechnern mit verteiltem Speicher

Die Fähigkeit eines universellen Parallelrechners, den inhärenten Parallelismus eines beliebigen Problems vollständig zu nutzen, kann auch heute noch, nach mehr als zwei Jahrzehnten der Forschung über Parallelverarbeitung, als ein ideales Ziel angesehen werden, das es nach wie vor zu erreichen gilt. Valiant und Brebner, die diese Feststellung bereits 1981 trafen [V&B,81], sehen hierbei das parallele Routing als das Kernproblem an, da es bei effizienter Ausführung einem realen Parallelrechner ermöglichen würde, einen idealen Parallelrechner mit vertretbarem Aufwand zu simulieren.

Gegenstand der Untersuchungen sind daher nun parallele Routing-Verfahren, deren zentrale Bedeutung für das massiv-parallele Rechnen häufig betont wird ([M&C,86], [KRT,88]). Für die immer wieder geforderte Skalierbarkeit als wichtigste Eigenschaft zukünftiger Parallelrechner postuliert Valiant [Val,90a], daß die Kommunikationsleistungen in stärkerem Maße wachsen müssen als die Rechenleistungen. Upfal und Wigderson [U&W,84] beschreiben einen Extremfall von Kommunikationsanforderungen in einem verteilten Rechner mit p Prozessoren, der dazu führt, daß selbst bei vollständiger Verbindungsstruktur (vgl. Kapitel 3) und voller Parallelisierbarkeit des Problems kein *Speedup* (> 1) erzielt wird. Als Hauptschwierigkeit erweist sich hierbei weniger die technische Realisierung einer ausreichenden Bandbreite als vielmehr die Entstehung von Engpässen durch lokale Konzentration von Nachrichten, wodurch erhöhter Zeitbedarf verursacht wird; mit anderen Worten: die verfügbare Bandbreite wird im allgemeinen nur zu einem sehr geringen Prozentsatz ausgenutzt. Bevor Maßnahmen zur Abhilfe von Kommunikationsengpässen in Netzwerken überlegt werden, sollten daher zunächst die auftretenden Muster bei Überlastungssituationen des Netzwerks analysiert werden. An dieser Stelle setzen die Untersuchungen in dieser Arbeit an, mit Hilfe kombinatorischer Versuchspläne und der ihnen eigenen Balance eine Lastbalance auf Verbindungsnetzwerken zu erreichen, so daß auf diesem Wege Komplexitätsverbesserungen für das Routing auf der betreffenden Architektur erzielt werden.

7.1.1 Deterministisches Routing in statischen Verbindungsnetzwerken

Für die hier vorzunehmenden Untersuchungen sind die verwendeten Rechner auf verteilte Rechner mit ausschließlich lokalem Speicher eingeschränkt worden. Diese Speichersituation bzw. -distribution bringt es gleichzeitig mit sich, daß die Kommunikation zweier Prozessoren in einem der hier betrachteten Systeme nur mittels *Message Passing* erfolgen kann, da kein gemeinsamer Speicher, das Kommunikationsmittel starkgekoppelter Systeme, zur Verfügung steht. **Message Passing** bedeutet die Versendung von Nachrichten über Wege in Netzwerken, die aus einer Kette direkt miteinander verbundener Prozessoren und diesen Verbindungen selbst bestehen. Eine Nachricht muß daher neben dem eigentlichen Inhalt der Nachricht auch die Zieladresse ständig mit sich führen, damit das Weitersenden in den Zwischenstationen korrekt fortgeführt werden kann. Für das eventuelle Auftreten von Fehlern und die Meldung derselben oder eine Ankunftsbestätigung des Zielknotens ist es ratsam, auch die Startadresse mitzuführen. Darüber hinaus kann es aus Gründen von Aufwandsabschätzungen und Balancierung der Lasten auf den verschiedenen Übertragungswegen sinnvoll sein, die Größe einer Nachricht ebenfalls zusätzlich als Teil der Gesamtnachricht aufzunehmen. Dieser Para-

meter ermöglicht dann auch Zwischenkontrollen betreffend teilweiser Verluste von Nachrichten.

Da aus den in Kapitel 5 angeführten Gründen normalerweise nicht zwischen allen Knoten eines verteilten Systems eine direkte Verbindung besteht, muß eine Nachricht in der Regel über mehrere Zwischenstationen, also andere Prozessoren, laufen. Mit dem Begriff **Routing** wird die Bestimmung solcher, für die Versendung von Nachrichten notwendigen, Wege zwischen verschiedenen Prozessoren eines verteilten Rechners bezeichnet. Der Weg von einem Startknoten s zu einem Zielknoten d wird im allgemeinen beschrieben durch die Netzwerkknoten, die auf diesem Weg durchlaufen werden. Für den speziellen Fall eines Hypercube-Rechners, in dem die als Binärzahlen aufzufassenden Adressen verbundener Knoten sich in genau einem Bit unterscheiden, besteht ein solcher Weg folglich aus entsprechenden Binärzahlen, wobei die Gesamtweglänge deshalb bereits a priori durch den *Hamming-Abstand* [Ham,87] von Start- und Zieladresse gegeben ist. Wie in diesem Fall kann man die jeweilige Vorgehensweise im allgemeinen formalisieren und in Form eines Routing-Schemas oder Routing-Algorithmus formulieren.

Effizientes Routing ist vor allem auch deshalb von großer Wichtigkeit, weil einerseits das Verhältnis von Kommunikationsleistung relativ zur Rechenleistung bei heutigen Rechnern noch zu gering ist, andererseits aber die Ergebnisse von Gentleman [Gen,78] belegen, daß eine Reduktion der Berechnungskomplexität unter die Kommunikationskomplexität die Gesamtkomplexität nicht verbessert. Neben den bereits in Kapitel 3 zu findenden Kenngrößen für die Bewertung oder Charakterisierung von Netzwerken ist auch die Einfachheit der Routing-Operationen von entscheidender Bedeutung für die Beurteilung einer bestimmten Topologie hinsichtlich ihrer Eignung als Form eines verteilten Rechners [Röd,91].

In Anlehnung an die Terminologie von Pippenger [Pip,84] erfolgt zunächst eine definierende Zusammenfassung der im folgenden notwendigen Fachtermini des Routing. Zwischen verschiedenen Prozessoren (Knoten) eines Netzwerks können **Nachrichten** versendet werden, die aus einem oder mehreren **Paketen** bestehen können. Derjenige Prozessor, der die Nachricht aussendet, wird als **Startknoten**, der Prozessor, der die Nachricht empfangen soll, wird als **Zielknoten** der Nachricht bezeichnet. Das **Kommunikationsproblem** besteht nun darin, alle Pakete in möglichst wenigen Zeiteinheiten von ihren Startknoten zu ihren Zielknoten zu bringen unter der Nebenbedingung, daß jede Verbindung in einer Zeiteinheit nur von einem Paket benutzt werden kann. Als **Permutation** wird der Spezialfall des Kommunikationsproblems bezeichnet, bei dem (jede Nachricht aus genau einem Paket besteht und) in einem System mit N Knoten N Pakete versendet werden, die jeweils unterschiedliche Start- und Zielknoten haben. Werden nicht alle Knoten belegt, ist die Zahl der Pakete also kleiner als N, so spricht man von einer **partiellen** Permutation. Dieser Fall wird von Valiant und Brebner [V&B,81] verallgemeinert zur **partiellen h-Relation**, die gestattet, daß sich bis zu h Knoten einen Start- bzw. Zielknoten teilen. Der Transport der Pakete, zur Lösung des Kommunikationsproblems, geschieht in Abhängigkeit bestimmter **Regeln**, die sowohl deterministischer als auch zufälliger Art sein können und in einem **Routing-Verfahren** zusammengefaßt werden. Die Anwendung dieser Regeln läßt jedes Paket einem bestimmten **Pfad** von seinem Startknoten zu seinem Zielknoten folgen.

Ein **Kommunikationsschema** besteht aus einem festgelegten statischen Verbindungsnetzwerk sowie einem zugehörigen Routing-Verfahren, das in der Lage ist, beliebige Kommunikation zwischen Paaren von Knoten des Netzwerks zu etablieren. Ein Kommunikationsschema heißt **lokal** (*oblivious*, vergeßlich), wenn der Pfad einesjeden Paketes nur von dessen eigenen Start- und Zielknoten abhängt, nicht jedoch von denen anderer Pakete. Als **Komplexität** (*delay*, Verzögerung) wird die Zahl der Zeiteinheiten bezeichnet, die maximal benötigt werden, um alle Pakete von ihren Startknoten zu ihren Zielknoten zu bringen. Mit **Population** wird die Zahl der Pakete bezeichnet, die sich zu einem Zeitpunkt in einem Knoten aufhalten, der Begriff **Stauung** (*congestion*) bezieht sich auf die Zahl der Pakete, die sich in einer Warteschlange befinden. Borodin und Hopcroft wiesen nach, daß ein deterministisches, lokales Kommunikationsschema für gewisse Permutationen mindestens die Komplexität $\Omega(N^{1/2})$ erfordert [B&H,82]. Stout und Wagar [S&W,90] beschreiben Techniken, die sowohl die Transferzeiten als auch das Verkehrsaufkommen reduzieren können. Hierzu gehören *Pipelining*, parallele Wege und veränderliche Paketgröße. Auf grobem Abstraktionsniveau kann man zwischen folgenden Routing-Verfahren unterscheiden [RLS,88]:

- starres Routing
- flexibles Routing
  - zentralisiertes flexibles Routing
  - verteiltes flexibles Routing

Während bei **starrem Routing** (*fixed-path routing*) der einmal festgelegte Weg für die Übertragung einer Nachricht von einem Startknoten zu einem Zielknoten, wenn man überhaupt Auswahlmöglichkeiten hat, nicht mehr verändert wird, kann man durch **flexibles Routing**, das auch als **adaptives Routing** bezeichnet wird, in jedem Zwischenknoten wählen, welche von den vorhandenen (Rest-) Routen zum Ziel die geeigneste ist. Die Kriterien, die für diese Auswahl herangezogen werden, beschreiben im allgemeinen den Zustand des Gesamtsystems, der sich durch Parameter wie Verkehrsaufkommen, Lastverteilung und Nutzung der Bandbreite beschreiben läßt. Während bei **zentralisisiertem flexiblem Routing** die Situation zentral bewertet wird und allen notwendigen Knoten mitgeteilt wird, entscheidet bei **verteiltem flexiblem Routing** der aktuell betroffene Zwischenknoten nur aufgrund seiner lokal bei ihm vorhandenen Informationen. Der Aufwand sowohl an Speicherplatz als auch an zusätzlicher Kommunikation ist bei zentralisierter Vorgehensweise erheblich größer.

Zu den Kriterien, die die Entscheidung für die Auswahl des nächsten Routing-Teilstücks unterstützen können, zählt beispielsweise auch der Füllungsgrad der Warteschlangen für die betreffenden Verbindungen. Wählt man lokal diejenige Verbindung aus, deren Warteschlange mit der geringsten Zahl von Nachrichten belegt ist, so spricht man von *shortest queue routing*. *NRCC-Routing* ist ein zentralisiertes Verfahren, welches in festen Zeitabständen die Netzwerkknoten mit Ranglisten über die geeignetsten Pfade versorgt. *Delta-Routing* und *hybrid weighted routing* sind Mischverfahren, die sich durch Kombination der NRCC- und shortest-queue-Verfahren ergeben [K&R,88].

Eine weitere, zunächst irrational erscheinende Möglichkeit des Routing besteht darin, in einer ersten Phase für die einzelnen Nachrichten beliebige Knoten als Zwischenziele zufällig auszuwählen, wodurch im Mittel eine gleichmäßige Verteilung über das System erreicht wird, und in der zweiten Phase das eigentliche Ziel mit den Mitteln des adaptiven

Routing anzusteuern. Man mag nach dem Sinn der Phase I fragen, welche ja unter Umständen die doppelte Pfadlänge für bestimmte Routen erfordert. Für Netzwerke mit einer geringen Zahl von Prozessoren bedeutet die Hinzunahme von Phase I sicherlich erhöhten Aufwand, für die hier interessierenden massiv-parallelen Systeme können jedoch Komplexitätsverbesserungen erzielt werden. Diese als *randomized routing* bekannte Verfahren werden im nächsten Abschnitt genauer analysiert.

7.1.2 Nichtdeterministisches Routing

Die Zufallstechnik hat seit ihrer Einführung in die Informatik durch Rabin, Solovay und Strassen ([Rab,76], [S&S,77]) vielfache Anwendung sowohl im Entwurf sequentieller als auch paralleler Algorithmen gefunden. Seit Anfang der achtziger Jahre sind auch Anwendungsmöglichkeiten im Routing bekannt [Val,82a]. Belege für die Wichtigkeit von Zufallsverfahren und ihre erhebliche und noch wachsende Bedeutung im Bereich Parallelverarbeitung findet man bei Leighton [Lei,90]. Bezüglich der bisherigen Beurteilung der Komplexität von Standardkommunikationsaufgaben verweist er darauf, daß man aus der relativ kleinen Zahl von *worst-case*-Fällen und der relativ kleinen Zahl von Standard-Routing-Situationen bezogen auf die Gesamtheit aller möglichen Routing-Aufgaben falsche Schlußfolgerungen zog, die Komplexitäten daher zu niedrig ansetzte, da die Standard-Routing-Situationen nahezu alle in den *worst-case*-Bereich fallen. Die Abhilfe besteht hier darin, eine ungünstige Routing-Aufgabe bzw. Permutation, z.B. für die Matrixinversion, durch die Komposition zweier zufälliger Permutationen zu ersetzen, um so die Zahl der Konflikte im Verbindungsnetzwerk des Parallelrechners erheblich zu reduzieren [Lei,92b]. Zum Zeitpunkt der ersten Arbeiten von Valiant, muß man die Methode der Zufalls-Routing-Verfahren eher als eine Technik sehen, um die Komplexität einer Permutation analytisch auf $O(\log N)$ zu reduzieren [Val,83], als daß die Vorstellung gegeben war, man könne mit Zufallsverfahren in der Praxis tatsächlich Vorteile gegenüber deterministischen Verfahren erreichen. Dies wird hier gefolgert aus dem Faktum, daß die von Leighton geführte Argumentation in den frühen Arbeiten von Valiant an keiner Stelle zu finden ist.

Durch unbalancierte Auslastung der Verbindungen zwischen den Prozessoren eines massiv-parallelen Rechners kann es zu starken Einbrüchen in der Leistung des Kommunikationssystems eines solchen Rechners kommen [H&J,86]. Derartige Situationen treten in der Praxis recht häufig auf [G&&,83], so daß Maßnahmen zur Abhilfe dringend vonnöten sind. Mit Hilfe der Methode des sogenannten *randomized routing* wird versucht, eine gleichmäßige Verteilung der Last über das gesamte Verbindungssystem zu erreichen. Valiant gelang es erstmals, den Aufwand zur Durchführung einer Permutation, auf einem Hypercube, mit Hilfe von Zufallstechniken von $O(\log^2 N)$ auf $O(\log N)$ zu reduzieren [Val,82a]. Zwar existieren (dynamische) Netzwerke, wie das Clos-Netzwerk [Clo,53] oder das Benes-Netzwerk [Ben,65], die aufgrund der Zahl ihrer Stufen die Realisierung einer Permutation in Komplexität $O(\log N)$ gestatten; der Aufwand zu ihrer Initialisierung beträgt jedoch günstigstenfalls $O(\log^2 N)$, so daß dies als die bestimmende Funktion für die Realisierung der Kommunikation angesehen werden muß [Ale,82]. Gleiches gilt für Batcher-Netzwerke [Bat,68]. Da Valiants Verfahren, Verfahren V, sowohl den Ausgangspunkt für die gesamte Entwicklung der parallelen Zufalls-Routing-Verfahren darstellt, als auch die Grundlage für die hier zu entwickelnden Ver-

fahren bildet, soll es exemplarisch etwas detaillierter erläutert werden, wobei das in Abb. 38 auf Seite 153 beschriebene Kommunikationsmodell gilt.

Verfahren V (Verfahren von Valiant)

Der Algorithmus besteht aus zwei getrennt ablaufenden Phasen; in Phase I wird jede Nachricht an ein zufälliges Ziel, d.h. an jeweils einen zufälligen (Zwischen-)Knoten, gesendet und in Phase II erfolgt die Weitersendung von den zufälligen Zwischenknoten zu den Zielknoten.

1. In Phase I wählt jedes Paket in jeder Zeiteinheit eine Kommunikationsrichtung (Adreßdimension) zufällig aus und bewegt sich mit Wahrscheinlichkeit 0,5 entlang dieser Verbindung. Eine Richtung kann für die gesamte Phase I nur einmal ausgewählt werden. Falls eine Nachricht sich dafür entscheidet, in eine bestimmte Richtung weiter zu laufen, reiht sie sich in die entsprechende Warteschlange im aktuellen Knoten ein.

2. In Phase II geschieht technisch dasselbe, nur wird jetzt deterministisch der kürzeste Pfad von dem in Phase I zufällig angesteuerten Zwischenknoten zum eigentlichen Zielknoten beschritten. Wählt eine Nachricht in Phase II eine bestimmte Richtung aus, so wird diese Verbindung – im Gegensatz zu Phase I – tatsächlich immer benutzt.

Jedem Nachrichtenpaket p ist eine Menge $U_p \subseteq \{1,...,n\}$ mit $n = \log_2 N$ zugeordnet. In Phase I enthält U_p diejenigen Richtungen, die noch nicht betrachtet, und damit insbesondere auch nicht benutzt, worden sind. In Phase II enthält U_p diejenigen Richtungen, die noch benutzt werden müssen, um an den Zielknoten zu gelangen. Jede Phase ist folglich beendet, wenn die Mengen U_p für alle Pakete p leer sind.

Valiant hat gezeigt, daß dieses zufällige, lokale Verfahren für jede Parameterkombination aus den in Abschnitt 7.2 genannten Modellen, Prioritäten und zufälligen Kommunikationsaufgaben (Permutation, Relation) mit großer Wahrscheinlichkeit nach $O(\log N)$ Kommunikationsschritten abgeschlossen ist. Das Verfahren kann verteilt implementiert werden, wenn in der ersten Phase die Menge der gewählten Dimensionen pro Paket mitübertragen wird. Nach Valiant eignet sich das Verfahren besonders, wenn gleichzeitig alle Knoten des Rechnersystems kommunizieren oder das Kommunikationsaufkommen ungleichmäßig und nicht vorhersehbar ist. Experimentelle Ergebnisse bestätigen die theoretische, stochastische Analyse des Verfahrens sowohl in zeitlicher Hinsicht als auch im Hinblick auf Stauung [Val,80]. Die Optimalität des Verfahrens, hinsichtlich der Komplexitäts*ordnung*, wurde von Valiant nachgewiesen; er zeigte darüber hinaus, daß unter gewissen Umständen die Länge der Pfade nahezu dem doppelten Durchmesser entsprechen kann, wenn eine möglichst günstige Komplexität erreicht werden soll [Val,83]. Da die Phasen I und II getrennt behandelt werden können, ist Phase I unabhängig von den Zielknoten und Phase II unabhängig von den Startknoten. Damit ergibt sich für das Gesamtverfahren nach Valiant die Unabhängigkeit des Laufzeitverhaltens von der jeweils zu realisierenden Permutation oder Relation [Val,90a]. Daher kann das

probabilistische Verhalten jeder Phase durch Simulation mit einer beliebigen Permutation, insbesondere auch der Identität, bestimmt werden. Diese als Überprüfbarkeit (*testability*) bezeichnete Eigenschaft ist unabhängig vom jeweiligen Arbeitsmodus der Warteschlangen.

Weitere wichtige Verfahren aus dem Bereich des *Randomized Routing*, für den Leighton eine gute Zusammenstellung liefert [Lei,92a], sollen nun im Gesamtkontext besprochen werden, wobei es an dieser Stelle sinnvoll ist, auch Verfahren für dynamische Netzwerke zu berücksichtigen. Valiant und Brebner [V&B,81] wendeten die nichtdeterministische Routing-Technik auf den *d-way-shuffle-graph* an. Hierbei ergaben sich Kommunikationsschemata mit Komplexität $O(\log N)$ und Grad $O(\log N/\log^2 N)$. Darüber hinaus postulierten sie die Existenz von Schemata mit Komplexität $O(\log N)$ und Grad $O(1)$. Dies wurde unabhängig von Aleliunas [Ale,82] für die gleiche Topologie und Upfal [Upf,84] für den *d-way-digit-exchange-graph* bestätigt. Pippenger [Pip,84] wies die Existenz eines parallelen Kommunikationsschemas für den *d-way-digit-exchange-graph* nach, das bei Stauung $O(1)$ mit hoher Wahrscheinlichkeit Komplexität $O(\log N)$ hat. Pippengers Verfahren besitzt jedoch nicht die Verifikationseigenschaft, da der Zeitaufwand von der jeweiligen Kommunikationsaufgabe abhängig ist. Ebenso wie Valiant arbeiten Aleliunas und Upfal mit zwei Phasen und einem lokalen Verfahren. Eine wichtige Kenngröße der zu untersuchenden Algorithmen ist die maximale Länge der Warteschlange in jedem Knoten; für die Routing-Verfahren von Valiant und Upfal beträgt diese $O(\log N)$, im Verfahren von Pippenger hingegen $O(1)$. Auch für das Verfahren von Rabin [Rab,89] sind Warteschlangen konstanter Länge ausreichend. Dieses als IDA (*information dispersal algorithm*) bezeichnete Verfahren besitzt darüber hinaus durch Einführung von Redundanz Fehlertoleranzeigenschaften. Aiello et al. entwickeln ein Zufalls-Routing-Verfahren für den Hypercube und zeigen, daß die asymptotisch optimale Komplexität $O(\log N)$ adaptives Vorgehen erzwingt [A&&,91]. Rajasekaran und Tsantilas [R&T,87] liefern einen Zufalls-Routing-Algorithmus für einen maschenverbundenen Rechner. Dieser leistet das Routing von N^2 Paketen in einem $(N \times N)$-MCC ohne zyklische Randverbindungen mit hoher Wahrscheinlichkeit in $2N + O(\log N)$ Schritten. Aufgrund der maximal möglichen Entfernung zweier Knoten in einer solchen Topologie ist das Verfahren als optimal anzusehen. Mitra und Cieslak ([M&C,86], [M&C,87]) arbeiten mit einem erweiterten Omega-Netzwerk, das man erhält, indem ein gewöhnliches Omega-Netzwerk [Law,75] für $N = 2^n$ Prozessoren, also mit n Stufen, mit einem Verteilungssnetzwerk (*scattering-network*) aus r Stufen zufällig arbeitender Schaltelemente gekoppelt wird, wobei sich r zwischen 1 und $n - 1$ bewegen kann. Szymanski und Fang betrachten sogenannte *hypermeshes*, die auf Kreuzschienenverteilern basieren, sowie erweiterte Banyan-Netzwerke als Zieltopologien zur Implementierung von Zufalls-Routing-Verfahren ([Szy,91], [S&F,91]). Das Zufallsverfahren von Greenberg und Leiserson [G&L,85] ist für die Topologie des *fat-tree* konzipiert (vgl. Abschnitt 3.4.3). Im Gegensatz zum Verfahren von Valiant bestimmt der Zufall nicht, welche von den verschiedenen möglichen Richtungen bzw. Verbindungen eine Nachricht zu einem bestimmten Zeitpunkt wählt, sondern *ob* die Nachricht gesendet wird. Mit der Integration des *fat-tree* und dem darauf basierenden Zufalls-Routing-Verfahren von Greenberg und Leiserson in das Verbindungsnetzwerk der Connection Machine CM-5 [Lei,92c] ist erstmals ein Zufallsverfahren für das Routing zum Bestandteil eines kommerziellen Produktes geworden.

7.2.1 Konzeption des Simulators und Meßumgebung

Anstelle eines stochastischen Beweises wird auf eine Simulation zurückgegriffen, da es offensichtlich nicht zu erwarten ist, wie in Abschnitt 7.1 ausgeführt, daß man die Komplexitätsordnung $O(\log N)$ zur Durchführung einer Permutation reduzieren kann. Für praktische Belange kann jedoch auch eine Beschleunigung um wenige Prozent wertvoll sein, sei es, daß man die hohe Zahl der insgesamt durchzuführenden Sendeoperationen betrachtet oder Echtzeitbedingungen unterstellt. Daher sind hier – im Gegensatz zu den in den anderen Kapiteln untersuchten Bereichen – auch die Multiplikationsfaktoren innerhalb einer Komplexitätsklasse von Interesse, die durch einen stochastischen Beweis wie bei Valiant [Val,82a] nur sehr schwierig und nicht hinreichend exakt bestimmt werden können [FCS,91].

Um die zu entwickelnden Routing-Verfahren geeignet mit Referenzverfahren vergleichen zu können, wurde ein Simulationsprogramm in der Sprache C entworfen. Anstelle eines möglicherweise anderweitig verfügbaren Simulators wurde ein eigenes Analysewerkzeug entwickelt, um zum einen das Kommunikationsmodell demjenigen von Valiant anzupassen – da dieses Verfahren für den Schwerpunktbereich der zufälligen Verfahren das Hauptreferenzverfahren darstellt – und zum anderen um, falls eine weitere Verfeinerung der Analyse notwendig werden sollte, durch Erweiterung des Programms um neue Analysekriterien flexibel reagieren zu können. In Abb. 38 sind die Annahmen des dem entwickelten Simulator zugrundeliegenden Kommunikationsmodells zusammengefaßt.

1. Jede Nachricht besteht aus genau 1 Datenpaket und alle Datenpakete haben identische Größe.
2. Die Übertragung der Daten ist fehlerfrei.
3. Es gilt das *store-and-forward-Modell*, d.h. ein Paket kann von einem Knoten erst dann weiterverwendet werden, wenn es vollständig übertragen wurde.
4. Das Versenden der Pakete erfolgt in synchronen, diskreten Schritten.
5. Es gilt Vollduplexbetrieb, d.h. eine Verbindung darf zu einer Zeit in jeder der beiden Richtungen von maximal einem Paket genutzt werden.
6. Die Verwaltungsarbeit in den Knoten erfolgt kostenfrei.
7. Die Übertragungszeit ist für alle Pakete konstant und unabhängig von der Zahl der insgesamt zu übertragenden Pakete und der benutzten Verbindungen.
8. Die Startzeit wird vernachlässigt und die Übertragungszeit eines Paketes wird für direkt verbundene Knoten auf 1 ZE (Zeiteinheit) gesetzt.

Abb. 38. Annahmen des Kommunikationsmodells

Die Übernahme des Kommunikationsmodells von Valiant aus den obigen Gründen erzwingt auch die Festlegung auf die Topologie des Hypercube, welche jedoch sehr konform zur Schwerpunktbildung in den bisherigen Kapiteln ist. Neben der Festlegung auf den Hypercube wurde auch die Valiantsche Anordnung übernommen, in jedem Experiment genau N Pakete auszusenden, wenn ein Hypercube mit $N = 2^n$ Knoten verwendet wird. Diese Zahl n, die also dem Logarithmus dualis sowohl der Anzahl der Prozessoren des Hypercube als auch der Zahl der zu versendenden Pakete entspricht, wird im folgenden als **Problemgröße** des Kommunikationsproblems bezeichnet.

Mögliche Varianten des Simulationsmodells ergeben sich dann durch beliebige Kombinationen der folgenden Parameter:

- Zahl der Warteschlangen pro Knoten

  1. **Modell A** (*1-port-communication-model*)
     Es existiert 1 (Abgangs-)Warteschlange pro Knoten.
  2. **Modell B** (*n-port-communication-model*)
     Es existieren $n = \log_2 N$ Warteschlangen pro Knoten, eine Warteschlange für jede Verbindung zu einem Nachbarknoten.

- Prioritäten für die Abarbeitung von Warteschlangen

  1. **FIFO-Priorität** (*first in, first out*)
     Dasjenige Paket, das zuerst an einem Knoten angekommen ist, wird auch zuerst weitergesendet.
  2. **FAFI-Priorität** (*farthest to go, first*)
     Dasjenige Paket, das noch den weitesten Restweg zu seinem Zielknoten hat, wird als erstes weitergesendet.

- Kommunikationsaufgaben (Art der Sender-Empfänger-Beziehungen)

  1. **Permutation** (*zufällige Kommunikationsaufgabe*)
     Jeder Prozessorknoten tritt genau einmal als Startknoten und genau einmal als Zielknoten auf.
  2. **Relation** (*zufällige Kommunikationsaufgabe*)
     Ein Knoten kann Sender und Empfänger mehrerer Pakete sein.
  3. **Matrixtransposition** (*deterministische Kommunikationsaufgabe*)
     Für gerade Problemgrößen $n = 2d$ werden Matrizen der Dimension $(2^d \times 2^d)$ transponiert. Die $N = 2^n$ Matrixelemente sind zunächst mittels der Abspeicherungsfunktion $sp(A_{ij}) = i \times 2^d + j;\ 0 \leq i,j \leq 2^d - 1$ in den Knoten des Hypercube abgelegt.

- Hypercube-Dimension
  Die Hypercube-Dimension ist (> 5) frei variierbar und nur durch die Speicherkapazität des Simulationsrechners beschränkt.

- Zahl der Durchläufe pro Simulationsexperiment (gleiche Parameter und Verfahren)
  Die Zahl der Durchläufe pro Simulationsexperiment ist frei variierbar.

Für die deterministische Kommunikationsaufgabe Matrixtransposition ist die Zuordnung bestimmter Pakete zu bestimmten Knoten im Gegensatz zu den zufälligen Kommunikationsaufgaben Permutation und Relation wesentlich. Daher muß für die Matrixtransposition der Gray-Code zwischengeschaltet werden, was für die zufälligen Aufgaben nicht notwendig ist.

Die Meßumgebung im Zentralinstitut für Angewandte Mathematik des Forschungszentrums Jülich basierte im April 1993 auf einem Rechner CRAY Y-MP M94/4256 [Cra,92] mit dem Betriebssystem UNICOS 7.0 und dem Cray-C-Compiler SCC 3.0 [Cra,91]. Alle im folgenden aufgeführten Messungen beziehen sich exakt auf diese Meßumgebung, wiewohl das Simulationsprogramm auch auf anderen C-Plattformen lauffähig ist und mit Ausnahme einer möglicherweise unterschiedlichen Charakterisitik der jeweiligen Zufallszahlengeneratoren auch gleiche Ergebnisse liefern sollte. Da die Untersuchungen nicht nur für synchrone Algorithmen sondern ebenso für asynchrone von Wert sein sollen, da diese an Bedeutung gewinnen, werden sowohl die maximalen als auch die mittleren Paketlaufzeiten der einzelnen Simulationsläufe betrachtet. In beiden Fällen wird dann wiederum über alle Läufe, in der Regel 100, des gesamten Simulationsexperimentes gemittelt. Für eine detaillierte Beschreibung des Simulationsprogramms wird auf eine spezielle Veröffentlichung verwiesen [Bur,93].

7.2.2 Sensitivitätsanalyse des Simulators

Für eine umfassende Analyse der Simulationsparameter wird auf die angegebene Veröffentlichung verwiesen; hier soll nur kurz das Verhalten des Simulators in Abhängigkeit der wesentlichen Parameter beleuchtet werden. Dazu werden alle Vorexperimente mit einheitlichen Parametervorgaben durchgeführt, die in Tab. 3 zusammengefaßt sind.

Parameter	Standardvorgabe
Modell	Modell A
Priorität	FIFO
Kommunikationsaufgabe	Permutation
Durchläufe pro Experiment	100
Verfahren	Verfahren D (vgl. Abschnitt 7.3)

Tab. 3. Standardparametervorgaben für die Sensitivitätsanalyse

Für die Sensitivitätsanalyse wird jeweils einer dieser Parameter variiert, während die übrigen festgehalten werden. Wenn keine speziellen Konstellationen bezeichnet werden, gelten die jeweiligen Feststellungen allgemein; insbesondere sind die hier anhand des Verfahrens D ermittelten Grundtendenzen auch für die anderen Verfahren zutreffend.

Zunächst interessiert die Brauchbarkeit der verwendeten Zufallszahlengeneratoren. Als Beispiel ist in Abb. 39 das Verhalten des CRAY-Zufallszahlengenerators abgetragen. Das Beispiel kann als repräsentativ in dem Sinne gelten, daß die Generatoren der übrigen verwendeten Rechner vergleichbare Ergebnisse liefern. Die Abbildung zeigt, daß der Zufallszahlengenerator die Adressen wie gewünscht verteilt, so daß die entstehenden Pfadlängen im Mittel eine hinreichend gute Normalverteilung zeigen. Die so entste-

henden mittleren Pfadlängen treffen denn auch die entsprechenden Erwartungswerte über die gesamte Spanne der Problemgrößen.

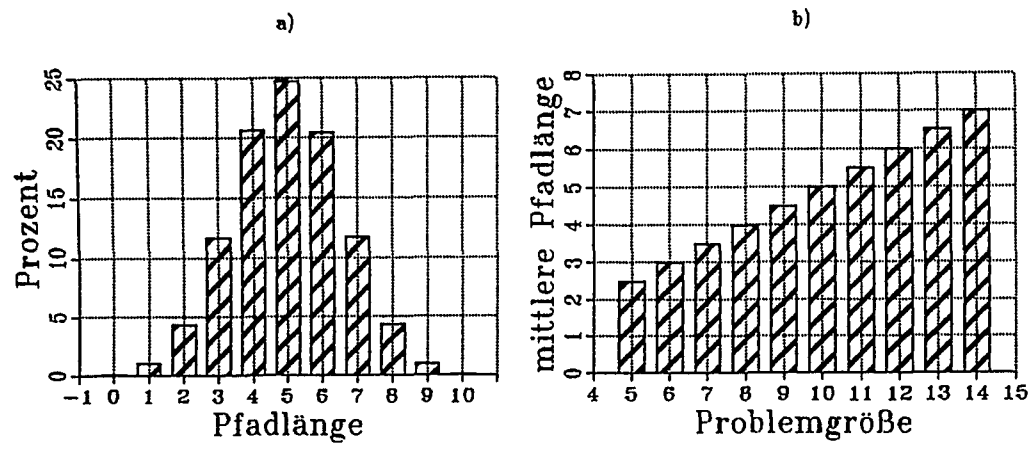


Abb. 39. Güte des Zufallszahlengenerators:
a) Verteilung der Pfadlängen bei Problemgröße 10
b) mittlere Pfadlängen

Nun muß untersucht werden, welche Zahl von Simulationsläufen notwendig ist, um stabile, verläßliche Ergebnisse zu erhalten. Abb. 40 zeigt, daß die Erhöhung der Anzahl der Simulationsläufe pro Experiment über 100 keinen wesentlichen zusätzlichen Stabilitätsgewinn erbringt. Die abgetragene mittlere Abweichung vom Mittelwert ergibt sich aus dem Quotienten von Varianz [Mcg,92] und der Anzahl der Simulationsläufe.

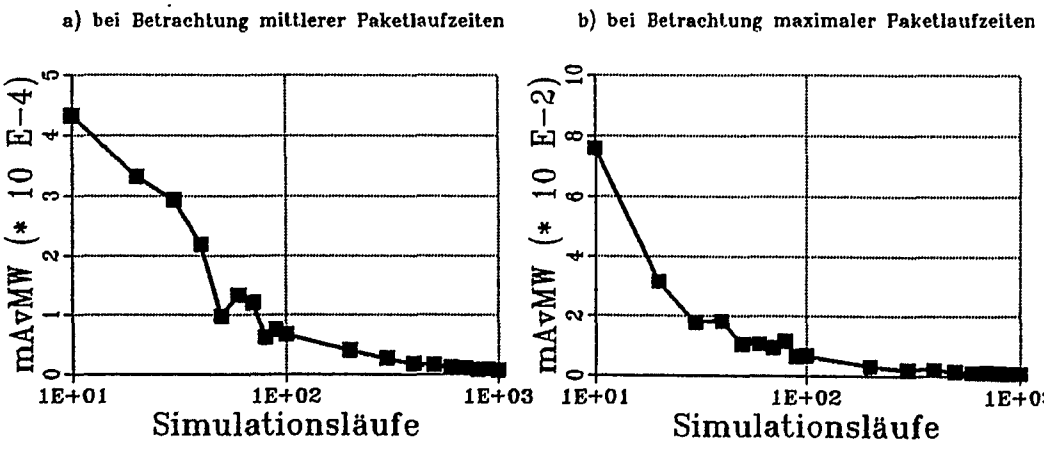


Abb. 40. Bestimmung der Anzahl der Simulationsläufe pro Experiment: Mittlere Abweichungen vom Mittelwert in Abhängigkeit von der Anzahl der Simulationsläufe pro Experiment bei Problemgröße 10

Darüber hinaus interessiert auch, welche Aussagekraft die Simulationsergebnisse hinsichtlich der Vorhersage der Komplexität einzelner durchzuführender Kommunikations-

aufgaben besitzen (vgl. Abb. 41). Für die mittleren Paketlaufzeiten werden die Simulationsdaten umso aussagekräftiger, je größer die betrachtete Problemgröße ist. Gerade für die besonders interessanten Fälle massiv-paralleler Kommunikation werden sehr stabile Ergebnisse erzielt. Für die maximalen Paketlaufzeiten muß zunächst festgestellt werden, daß die Varianz nicht abnimmt, die Simulationsdaten also für die Vorhersage von Einzelexperimenten nur bedingt geeignet sind. Darüber hinaus treten in Abhängigkeit von Modellen und Kommunikationsaufgaben drei unterschiedliche Verhaltensmuster auf.

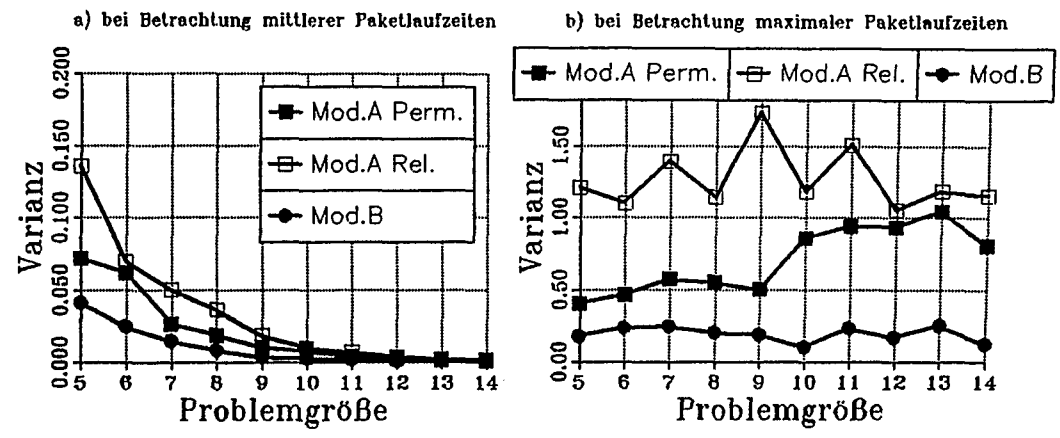


Abb. 41. Aussagekraft der Ergebnisse: Varianz in Abhängigkeit von der Problemgröße bei je 100 Simulationsläufen pro Experiment

Es folgt nun eine Grobbeurteilung der Modellvarianten. Abb. 42 zeigt, daß das Vorhandensein von einer Warteschlange für jede Abgangsrichtung im Knoten, wie in Modell B gegeben, zu verkürzten Paketlaufzeiten führt, wobei der Zeitgewinn für größer werdende Problemgrößen zunimmt.

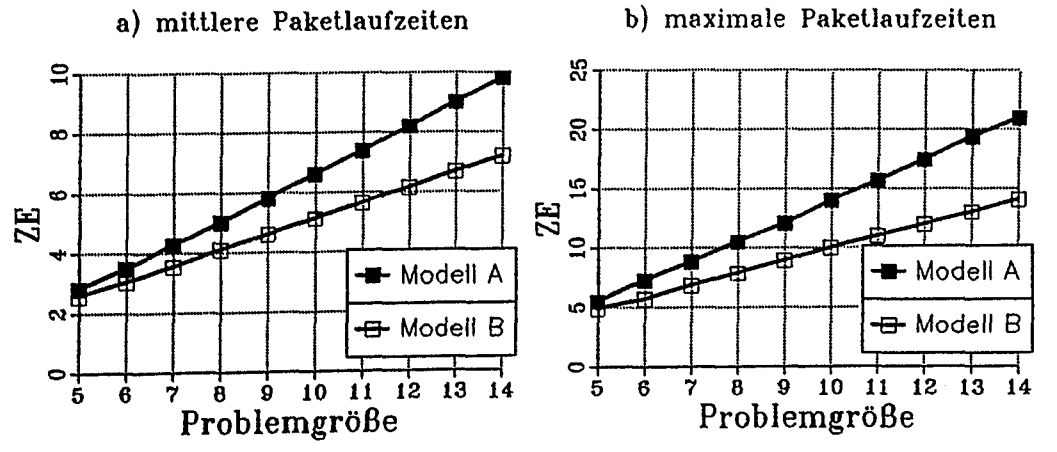


Abb. 42. Beurteilung der Modelle

Abb. 43 zeigt, daß das Vorliegen einer Permutation, die bedeutet, daß die Pakete vor und nach der Lösung der Kommunikationsaufgabe gleichmäßig über die Topologie verteilt sind, aufgrund ebendieser balancierten Verteilung zu einer besseren Nutzung der Bandbreite führt, wodurch gegenüber dem Relationsfall eine konstanter Zeitgewinn erwirtschaftet wird.

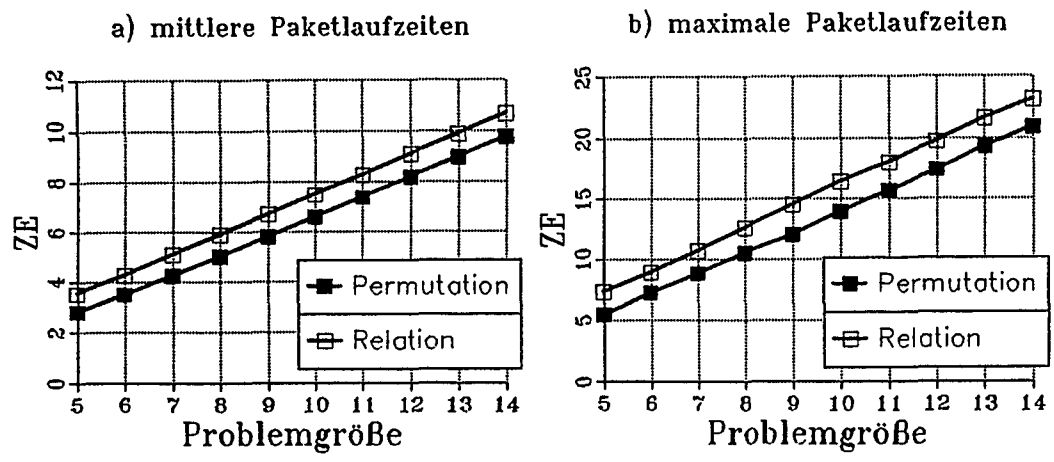


Abb. 43. Beurteilung der zufälligen Kommunikationsaufgaben

Abb. 44 zeigt, daß die Prioritäten im Mittel aller Paketlaufzeiten nahezu identische Ergebnisse liefern. Bei den maximalen Paketlaufzeiten erzielt die Priorität *farthest to go, first* (FAFI) jedoch merkliche Komplexitätsreduktionen gegenüber der FIFO-Priorität, die zudem mit wachsender Hypercube-Dimension zunehmen. Die FAFI-Priorität ist also insbesondere für synchrone Algorithmen von Interesse, wo auf die Ankunft des letzten Paketes gewartet werden muß, um auf allen Prozessoren den nächsten Prozeß starten zu können.

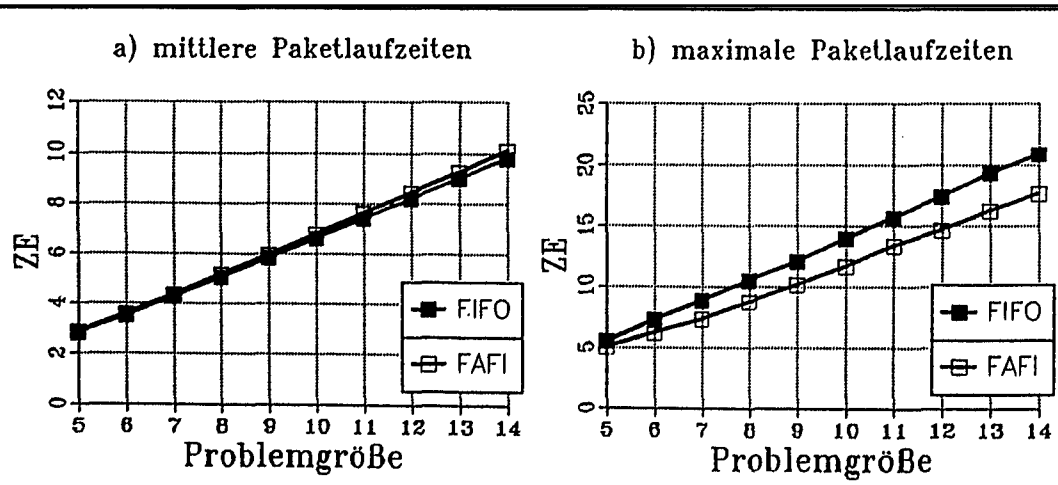


Abb. 44. Beurteilung der Prioritäten

7.2.3 Referenzverfahren

Für eine umfassende, formale Behandlung von Kommunikation auf dem Hypercube wird auf die Arbeiten von Bertsekas et al. [B&&,91] sowie Fox und Furmanski [F&F,88] verwiesen. Entsprechend den Ausführungen in Kapitel 3 verfügt ein Hypercube der Dimension n über $N = 2^n$ Knoten. Jeder Knoten ist mit n weiteren Knoten verbunden, d.h. der Grad beträgt n. Jeder Knoten wird durch eine n-stellige Binäradresse $(x_1...x_n)$ bezeichnet. Für jede Adresse a bezeichne $\overline{a}_i$ diejenige Adresse, die sich gegenüber der Adresse a genau in der i-ten Stelle unterscheidet. An der i-ten Stelle von $\overline{a}_i$ steht das Komplement der i-ten Komponente von a. Ein Knoten mit der Adresse a ist dann verbunden mit allen Knoten, die Adressen der Form $\overline{a}_i$, $i = 0,...,n-1$ haben. Durch die *Hamming-Distanz* [Ham,87] von Start- und Zieladresse ist bei den deterministischen Verfahren die insgesamt zurückzulegende Pfadlänge sofort gegeben. Für ein Paket bedeutet **Bewegung in Richtung** k, daß sich ein Paket, welches sich an einem Knoten mit der Adresse a befindet, sich zu dem Knoten mit der Adresse $\overline{a}_k$ bewegt.

Referenzverfahren für deterministische Routing-Verfahren

Als Referenzverfahren für die deterministischen Verfahren soll nun das direkte Verfahren, Verfahren D, dienen:

Verfahren D (Direktes Verfahren)

In jedem Routing-Schritt wird für jedes Paket die aktuelle Knotenadresse mit der Knotenadresse des Zielknotens verglichen. Der Vergleich beginnt beim niedrigwertigsten Bit und endet beim höchstwertigsten Bit. Die erste Binärstelle, an denen sich die beiden Adressen unterscheiden, liefert die Richtung, in die sich das Paket im nächsten Routing-Schritt bewegen muß. Sei beispielsweise a die Adresse des aktuellen Knotens und sei k die niedrigwertigste Stelle, an der sich beide Adressen unterscheiden, dann wird im nächsten Schritt der Knoten mit der Adresse $\overline{a}_k$ angesteuert.

Referenzverfahren für Zufalls-Routing-Verfahren

Um die korrekte Umgebung für das (Hauptreferenz-)Verfahren von Valiant zu gewährleisten, werden die Zufallsverfahren wie bei Valiant in zwei getrennten Phasen durchgeführt. Die sich dann durch Addition der in den beiden Phasen benötigten Zeiteinheiten ergebende Gesamtzeit eines einzelnen Paketes ist daher zunächst nur eine statistische Größe, die sich im wesentlichen nur zum Vergleich verschiedener Verfahren untereinander eignet, was hier aber auch Ziel der Untersuchungen ist. Neben dem Verfahren von Valiant (vgl. Verfahren V, Abschnitt 7.1) soll ein weiteres Verfahren, Verfahren Z, als Referenzverfahren dienen. Der wesentliche Unterschied des Verfahrens von Valiant gegenüber dem Verfahren Z besteht darin, daß im Verfahren V nicht nur die Zwischen-

adresse zufällig ausgewählt wird, sondern auch die Reihenfolge, in der die Richtungen betrachtet werden. Diese zusätzliche Variation gilt für beide Phasen des Verfahrens.

Verfahren Z (Standard-Zufallsverfahren)

Das Verfahren stützt sich auf den Zufallszahlengenerator des jeweiligen Rechners ab und läßt von diesem die zufälligen Zwischenadressen berechnen.

Daneben ist noch ein weiteres Verfahren denkbar, dessen rechnerbasierte Zufallszahlen nicht wie bei Verfahren Z selbst als Zwischenadressen dienen, sondern die gesetzten Bits in den Zufallszahlen liefern die Stellen, an denen die Bits der Startadressen zu invertieren sind. Die so modifizierten Startadressen stellen dann die zufälligen Zwischenadressen dar. Häufungen in einzelnen Knoten sind also, wie bei Verfahren Z, durchaus möglich [Bur,93]. Die unterschiedliche Verwendung der vom Generator gelieferten Zufallszahlen zur Produktion von zufälligen Zwischenadressen, wird dann entscheidend, wenn man zu den Verfahren mit pfadlängenreduzierter Phase I übergeht.

Referenzverfahren für Zufalls-Routing-Verfahren mit pfadlängenreduzierter Phase I

Ziel der Zufallsverfahren ist eine Entkopplung der Zieladressen von den Startadressen, damit bei den Standard-Routing-Situationen, beispielsweise aus dem Bereich der Matrixbearbeitung, die regelmäßigen Kommunikationsmuster, die für die Verstopfungen verantwortlich sind, nicht mehr auftreten. Es stellt sich nun die Frage, ob die mittleren Pfadlängen der Zufallsverfahren für Phase I tatsächlich im Mittel die Länge $n/2$ und damit insgesamt die doppelte Pfadlänge gegenüber den deterministischen Verfahren aufweisen müssen, um die gewünschte Entzerrung zu erreichen. In erster Näherung kann man versuchen, hier mit einer mittleren Pfadlänge $n/4$ auszukommen; dazu sind verschiedene Ansätze möglich.

Verfahren Y (modifiziertes Austausch-Zufallsverfahren)

Das Verfahren stützt sich auf den Zufallszahlengenerator des jeweiligen Rechners. Es werden pro Paket zwei Zufallszahlen generiert; diese werden mit bitweisem AND miteinander verknüpft. Diese neu entstandene Zufallszahl besitzt dann im Mittel $n/4$ gesetzte Bits ($a_i = 1$). Diese gesetzten Bits bestimmen diejenigen Stellen, an denen die Startadresse zu invertieren ist. Das Ergebnis ist die zufällige Zwischenadresse.

Abb. 45 zeigt, daß die angestrebten mittleren Pfadlängen im Vergleich zum deterministischen Verfahren D und zu dem typischen (voll) zufälligen Verfahren Z erreicht werden. Die maximalen Pfadlängen bewegen sich ein wenig oberhalb des Mittelwertes aus den Pfadlängen des deterministischen und des voll-zufälligen Verfahrens. Die Differenzverhältnisse der drei Verfahren zueinander bleiben jeweils über die gesamte Spanne der Problemgrößen erhalten.

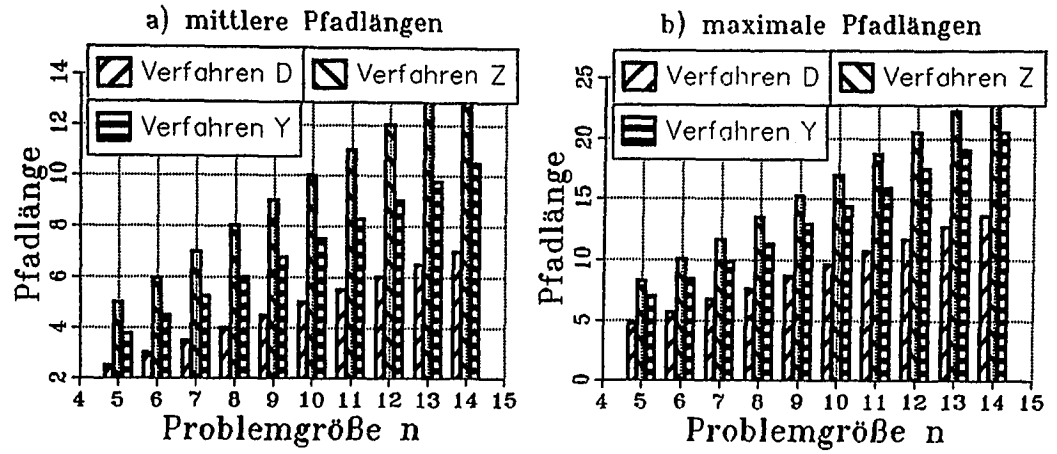


Abb. 45. Beurteilung des Referenzverfahrens Y: (Modell A / Priorität FIFO / Permutation)

Die Simulation zeigt weiterhin, daß sich das Faktum der verkürzten Pfadlängen auch auf die benötigten Zeiten überträgt (vgl. Abb. 46). Auch die Unterschiede im Übergang von mittleren Daten des Einzellaufs zu Maximaldaten des Einzellaufs werden wieder vorgefunden.

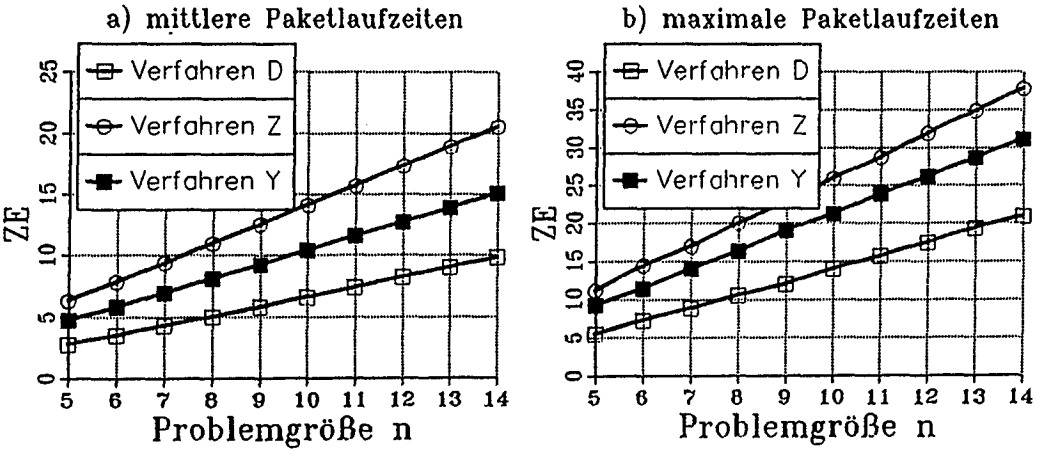


Abb. 46. Beurteilung des Referenzverfahrens Y: (Modell A / Priorität FIFO / Permutation)

Die hier an einer Beispielkonstellation demonstrierten Fakten sind für alle übrigen Parameterkombinationen ähnlich. Von Interesse ist nun, ob mit dieser pfadlängenreduzierten zufälligen Phase eine hinreichende Entkopplung von Start- und Zieladressen gewährleistet ist. Für bestimmte Architekturen wird das von Valiant bestritten [Val,83]. Mitra und Cieslak beispielsweise sehen jedoch auch kürzere Pfadlängen vor, da sich die Stufenzahl ihres Verteilungsnetzwerks zwischen 1 und log N bewegen kann [M&C,86]. Diese Fragestellung kann durch die Generierung deterministischer Routing-Situationen auf dem Simulator untersucht werden.

Vergleich deterministischer und zufälliger Verfahren

Es gilt nun noch zu verifizieren, daß bei praxisrelevanten, wichtigen Kommunikationsaufgaben Zufalls-Routing-Verfahren tatsächlich bessere Komplexitäten erzielen können als deterministische Verfahren. Dazu werden die Referenzverfahren D, Y und Z anhand der Durchführung einer *Matrixtransposition* miteinander verglichen. Da es sich um eine deterministische Kommunikationsaufgabe handelt, ist für das deterministische Verfahren D nur ein einziger Lauf notwendig. Abb. 47 zeigt, daß die zufälligen Verfahren Z und Y ab Problemgröße 10, also bei der Versendung von 1024 Paketen auf einem 10-dimensionalen Hypercube tatsächlich effizienter arbeiten als das deterministische Verfahren D.

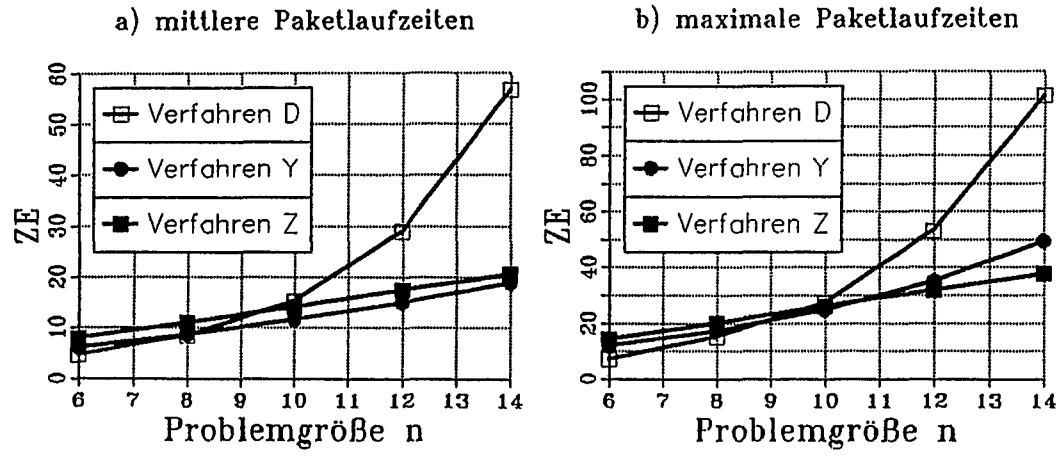


Abb. 47. Vergleich deterministischer und zufälliger Verfahren:
(Modell A / Priorität FIFO / Matrixtransposition)

Eine weitere wichtige Feststellung ist, daß Zufallsverfahren mit pfadlängenreduzierter Phase I durchaus eine hinreichende Entkopplung von Start- und Zielknoten gewährleisten können, so daß sie sicherlich für viele Kommunikationsaufgaben den voll-zufälligen Verfahren vorzuziehen sind. Auch Fulgham et al., die Zufallsverfahren ohne Phasentrennung untersuchen, stellen fest, daß es je nach Kommunikationsaufgabe und Problemgröße ausreichend ist, nur einen Teil der Bits zufällig zu ändern [FCS,91]. Vergleicht man die maximalen Paketlaufzeiten der Verfahren Y und Z aus Abb. 47 mit denjenigen aus Abb. 46 so ist jedoch zu beobachten, daß das voll-zufällige Verfahren Z bei zunehmender Problemgröße weniger anfällig für Verstopfungen ist. Während das deterministische Verfahren D bei der Matrixtransposition einen drastischen Leistungseinbruch verzeichnet und auch das halbzufällige Verfahren Y die angedeutete Grundsensibilität zeigt, entsprechen die Laufzeiten des Verfahrens Z den bei der Permutation erzielten. Dies steht im Einklang mit der von Valiant beschriebenen Verifikationseigenschaft, die gleiche Komplexitäten unabhängig von der Kommunikationsaufgabe garantiert.

7.3 Analyse kombinatorischer Routing-Verfahren

Bei der Darstellung der nun folgenden Meßergebnisse ist neben der in Abschnitt 7.2 eingeführten Definition der Problemgröße die Bezeichnung der Ordinate zu beachten. Die Werte auf der 'Y'-Achse liefern **Zeiteinheiten relativ zur Problemgröße** n. Die gemessenen Zeiteinheiten werden also durch die jeweilige Problemgröße dividiert und der Quotient wird abgetragen. Einerseits können so die abzulesenden Werte direkt als die für Kommunikationsaufgaben interessierenden Multiplikatoren der Größe $\log N$ bzw. der Komplexität $O(\log N)$ aufgefaßt werden, wobei N sowohl die Zahl der zu versendenden Pakete als auch die Zahl der Prozessoren des Hypercube beschreibt. Andererseits können die Differenzen zwischen den zu analysierenden kombinatorischen Verfahren und den Referenzverfahren bei Verwendung von absoluten Zeiteinheiten, auch bei logarithmischer Skalierung, in der Regel nur unzureichend wahrgenommen werden. Unter Berücksichtigung der festliegenden Komplexitätsschranken und den Setzungen des Modells, gilt die Präferenz der vorgenommenen Untersuchungen zunächst dem Verhältnis der Verfahren untereinander und erst in zweiter Linie dem absoluten Zeitverbrauch. Die tatsächlich benötigten Zeiteinheiten des jeweiligen Experimentes ergeben sich also durch Multiplikation der in den Diagrammen dargestellten Ordinatenwerte mit den jeweiligen Problemgrößen (Abszissenwerten).

7.3.1 Deterministische Routing-Verfahren auf der Basis Lateinischer Quadrate

Ziel der vorliegenden Arbeit ist die effizientere Ausnutzung der gegebenen Bandbreite. Es ist nun zu überlegen, wie die Balanceeigenschaft kombinatorischer Versuchspläne, hier konkret des Lateinischen Quadrats, auf die Routing-Problematik übertragen werden kann. Dazu wird ein erster Ansatz formuliert, für den die Struktur des Basisquadrats benötigt wird.

Als **(Lateinisches) Basisquadrat der Stufe** n wird im folgenden eine $(n \times n)$-Matrix verstanden, deren erste Zeile aus den Zahlen $0,1,\ldots,n-1$ besteht. Die zweite Zeile enthält die Zahlen $1,2,\ldots,n-1,0$ und sukzessive weiter; die letzte, n-te, Zeile enthält schließlich $n-1,0,1,\ldots,n-2$. Die Zeilen des Basisquadrats repräsentieren die Pakete in der Art, daß die Paketnummer modulo der Dimension n die Zeile bestimmt. Die durchnumerierten Pakete sind ihrer Nummer nach also zyklisch den Zeilen, modulo n, zuzuordnen. Für einen gegebenen Hypercube der Dimension n ist das Basisquadrat gleicher Dimensionierung heranzuziehen. Dies wird beispielhaft am Hypercube der Dimension 3 demonstriert (vgl. Tab. 4).

Paketnummer	*Basisquadrat*		
	0	**1**	**2**
0,3,6	0	1	2
1,4,7	1	2	0
2,5	2	0	1

Tab. 4. Basisquadrat für den Hypercube der Dimension 3

Alon, Barak und Manber [ABM,87] sowie Han und Finkel [H&F,88] verwenden ähnliche Strukturen mittels denen sie, auf Permutationen aufbauend, jedoch lediglich die spezielle Aufgabe des *broadcasting* betrachten.

Nimmt man an, daß die Spalten des Basisquadrats Dimensionen (Senderichtungen) im Hypercube repräsentieren, so ist aufgrund der Gesetzmäßigkeiten für Lateinische Quadrate garantiert, daß jedes Paket jede Richtung genau einmal betrachtet bzw. sich in der entsprechenden Richtung bewegt. Da in jeder Zeile und jeder Spalte eines Lateinischen Quadrats jeder Eintrag genau einmal auftritt, ist gewährleistet, daß alle Richtungen (fast) gleich belastet werden, insbesondere auch zu jedem konkreten Zwischenzeitpunkt und nicht nur in der Gesamtbilanz. Die Bezeichnung „fast gleich" resultiert aus drei Einflußfaktoren:

- Die Division der Zahl der Pakete N durch die Dimension n geht in der Regel nicht auf, so daß einzelne Zeilen ein Paket mehr vertreten als andere; die Differenz der Anzahl repräsentierter Pakete beträgt offensichtlich zwischen verschiedenen Zeilen maximal 1, so daß für eine hinreichend große Zahl von Paketen die Differenz vernachlässigbar wird und insbesondere im Hinblick auf die hier interessierenden massiv-parallelen Systeme von balancierter Verteilung gesprochen werden kann.

- Im Verlauf des Verfahrens werden mehr und mehr Pakete zugestellt, fallen also als Belastungsfaktor des Kommunikationsssytems aus. Dies kann einzelne Bereiche mehr betreffen als andere, im Mittel werden jedoch alle Zeilen und Teilwürfel gleichermaßen betroffen.

- Aufgrund von Stauungen ist es möglich, daß einzelne Pakete nicht in jedem Schritt einen Weg zurücklegen. Daraus resultiert, daß ein Paket i noch einmal den Eintrag von Spalte 1 berücksichtigen muß, während Paket j schon bei Spalte 2 aufsetzen kann und vielleicht bei Spalte 4 die erste Abweichung von aktueller Adresse und Zieladresse findet.

Balancierte Verteilung über die einzelnen Richtungen in den verschiedenen Schritten der Phase I, die aus der eben nachgewiesenen balancierten Verteilung auf die Zeilen resultiert, bedeutet a-priori jedoch noch keine balancierte Auslastung des Kommunikationssystems als solches. Aus der gleichmäßigen Verteilung auf die Richtungen kann noch nicht auf die gleichmäßige Verteilung auf die verschiedenen Verbindungen geschlossen werden. Hier werden die jeweils vorliegenden Kommunikationsaufgaben und -modelle möglicherweise eine Rolle spielen. Dies muß durch die Simulation verifiziert werden. Dabei ist auf die maximale Stauung zu achten, da diese den Gesamtzeitbedarf für die Zustellung eines einzelnen Paketes wesentlich mitbestimmt.

Bezüglich der Komplexität der Konstruktion des im folgenden verwendeten Basisquadrats wird hier davon ausgegangen, daß für eine feste, als Arbeitsplattform gewählte, Topologie, also hier ein Hypercube fester (Maximal-)Dimensionierung, das Basisquadrat kostenfrei, d.h. mit Komplexität $O(1)$, zur Verfügung gestellt wird, da die jeweilige Topologie für den Verlauf der Analyse eines bestimmten Routing-Verfahrens unverändert bleibt. Da hier nur verteilte bzw. lokale Verfahren von Interesse sind, muß das Basisquadrat in jedem Rechnerknoten vorliegen, sei es festverdrahtet oder abgespei-

chert, was aufgrund des geringen Speicherplatzverbrauchs problemlos möglich ist, oder durch Software realisiert werden.

Mit Hilfe des Basisquadrats sind nun mehrere Routing-Verfahren denkbar. Das Referenzverfahren D kann dahingehend abgewandelt werden, daß der in jedem Schritt notwendige Adressenvergleich nicht für jedes Paket beim niedrigwertigsten Bit beginnt, sondern die Startposition wird aus dem Basisquadrat entnommen.

Verfahren E (Einfaches kombinatorisches Verfahren)

In jedem Routing-Schritt wird für jedes Paket die aktuelle Knotenadresse mit der Knotenadresse des Zielknotens verglichen. Die Reihenfolge, in der die Binärstellen der beiden Adressen miteinander verglichen werden, ergibt sich aus den Einträgen im Basisquadrat der der Hypercube-Dimension entsprechenden Größe. Für Paket Nr. i wird die Zeile $i \bmod n$ herangezogen. Diese liefert die Reihenfolge, in der die Binärstellen miteinander verglichen werden.

Durch Simulation ist nun zu untersuchen, ob die durch das Lateinische Quadrat induzierte Verteilung der Aufsetzpunkte für den Adressenvergleich zu einer besseren Auslastung des Kommunikationssystems und damit zu einer reduzierten Komplexität führt.

Bei Modell A hat das Referenzverfahren D gegenüber dem Verfahren E im allgemeinen Vorteile, die Ergebnisse bezüglich maximaler Paketlaufzeiten in der Konstellation FIFO/Relation sind eine der wenigen Ausnahmen dieser Rangfolge, wie Abb. 48 zeigt.

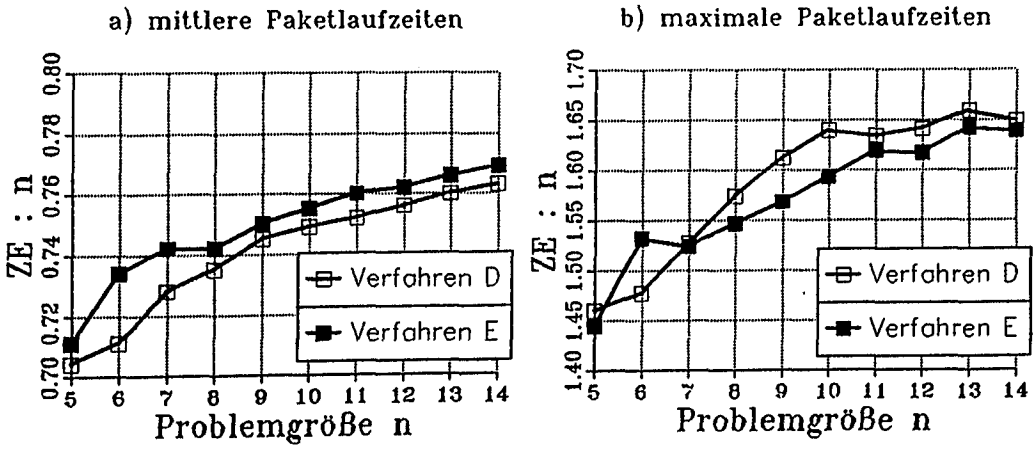


Abb. 48. Analyse des Verfahrens E: (Modell A / Priorität FIFO / Relation)

Bei Modell B ist die Situation umgekehrt, hier zeigt das kombinatorische Verfahren E durchweg die besseren Resultate, wie Abb. 49 beispielhaft zeigt. Die Ursache für das unterschiedliche Verhalten ist darin zu suchen, daß der Effekt des kombinatorischen

Verfahrens E im Modell A aufgrund von Warteschlangenstauungen in der Regel nicht zum Tragen kommt oder sich sogar nachteilig auswirkt, während bei Modell B die Stauungen im Mittel viel geringer und Verstopfungen damit seltener sind, wodurch eine höhere „Abgangsrate" bedingt ist, die Verfahren E für positive Ergebnisse verlangt.

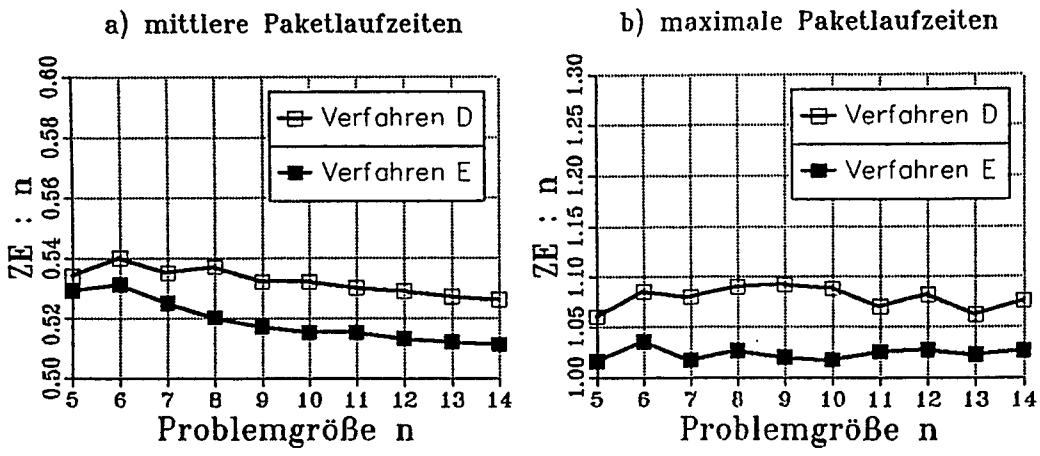


Abb. 49. Analyse des Verfahrens E: (Modell B / Priorität FIFO / Relation)

Man kann das Verfahren E nun dahingehend erweitern, daß man für jeden Routing-Schritt innerhalb einer Zeile des Basisquadrats zur nächsten Spalte übergeht. Das so entstehende Verfahren zeigt für Modell A ein mit Verfahren E vergleichbares Verhalten, für Modell B schneidet die modifizierte Variante schlechter ab [Bur,93].

Anstelle fester, zu Beginn festgelegter, Paketnummern kann man auch in jedem Schritt eine neue Numerierung vornehmen, wobei nur noch diejenigen Pakete betrachtet werden, die noch nicht zugestellt sind. Derartige Verfahren sollen hier nicht betrachtet werden, da sie globale Informationen voraussetzen und hier nur lokale (*oblivious*) Verfahren berücksichtigt werden. Dem einzelnen Paket ist nach dem ersten Routing-Schritt lokal nicht bekannt, wieviele Pakete noch nicht zugestellt sind.

7.3.2 Zufalls-Routing-Verfahren

Zufälligkeit ist die traditionelle Methode, um Verzerrungen zu eliminieren, die durch Stichproben-Vorgehen entstehen [Ale,82]. Diese Erzeugung einer gleichmäßigen Verteilung soll nun durch die Verwendung kombinatorischer Versuchspläne geleistet werden.

7.3.2.1 Verfahren auf der Basis Lateinischer Quadrate

Zufallstechniken bieten für zahlreiche Problemstellungen der Informatik elegante und effiziente Lösungsmöglichkeiten [R&S,89]. Da die in der Praxis verwendeten Rechner jedoch deterministisch arbeiten, verwendet man verschiedene Methoden, um den Zufall künstlich zu simulieren. Die wesentlichen Verfahren zur Derandomisierung sind:

- Generierung von Pseudozufallszahlen
  Die auf den herkömmlichen Rechnern eingesetzten Zufallszahlengeneratoren erzeugen keine wirklich zufälligen Zahlen, sondern arbeiten nach deterministischen Prinzipien und produzieren deshalb nur Pseudozufallszahlen. Eine **Pseudozufallszahlensequenz** ist dann eine Folge von Zahlen (oder Bits), die aus einer *Saat* (Ausgangsmenge) von Zahlen auf deterministischem Wege konstruiert wird, so daß ein Rechner mit endlichen Ressourcen ohne Zusatzinformationen im allgemeinen nicht erkennen kann, daß es sich nicht um tatsächlich zufällige Zahlen handelt [GKL,88].

- Reduktion des Wahrscheinlichkeitsraumes
  Das Ziel der Entrandomisierung besteht hier darin, den Wahrscheinlichkeitsraum so einzugrenzen, daß ein erschöpfendes, d.h. stets erfolgreiches, Absuchen der gesamten Menge möglich wird [Lub,85].

- Deterministischer Münzwurf
  Cole und Vishkin lösen das Problem der Bestimmung einer beherrschenden Menge von r Elementen, *r-ruling-set*, durch ein Verfahren, welches sie als deterministischen Münzwurf bezeichnen [C&V,86].

Nach Rajasekaran und Reif [R&R,87] existiert bislang kein paralleles Verfahren, welches die Technik der Ersetzung von Zufallsverfahren durch kombinatorische Konstruktionen implementiert. Auf diesem Feld soll nun Neuland betreten werden. Dazu wird auf die Idee zurückgegriffen, die ursprünglich hinter dem Einsatz der optimalen Versuchspläne stand: die Ersetzung zufälliger Stichproben durch deterministische Stichproben [K&W,85]. Aufsetzend auf dem Verfahren von Valiant [Val,82a] werden die Routing-Strecken in Phase I nun anstelle der Zufallszahlengeneratoren durch kombinatorische Versuchspläne vorgegeben.

Es ist wichtig zu beachten, daß die hier entwickelten 2-Phasen-Verfahren genau wie die anderweitig entwickelten Zufallsverfahren im wesentlichen dann günstige Ergebnisse erzielen, wenn es sich um eine „Standard"-Routing-Aufgabe handelt, die also ein sehr regelmäßiges Kommunikationsmuster liefert und deshalb bei deterministischer Vorgehensweise eine hohe Zahl von Konfliktsituationen und damit hohe Stauungen aufweist [Lei,92b]. Mit der Generierung von zufälligen Kommunikationsaufgaben, Permutationen oder h-Relationen, ist daher der Wert der 2-Phasen-Verfahren nur untereinander, nicht jedoch im Vergleich zu den deterministischen Verfahren zu beurteilen.

Ein erster Ansatz könnte darin bestehen, daß die Einträge in der zu jedem Paket gehörenden Zeile des Basisquadrats als die Reihenfolge betrachtet werden, in der die Bits der Startadresse zu invertieren sind. Die zufällige Zwischenadresse ist erreicht, wenn die gesamte Zeile abgearbeitet ist. Würde das so konstruierte Lateinische Quadrat unverändert verwendet werden, so wäre der Zielknoten einesjeden Paketes abhängig von seinem Startpunkt und damit die Entkopplung der beiden Phasen des Verfahrens nicht gegeben. Da ein Paket sich in jeder Richtung bewegt, wäre der Zielknoten der Knoten mit derjenigen Adresse, die aus den invertierten Bits der Startadresse besteht; Start- und Zieladresse wären also direkt abhängig voneinander, was im direkten Widerspruch zur gewünschten Entkopplung der Phasen I und II steht. Der nach Phase I angestrebte Zwischenknoten muß sich in gewisser Weise pseudozufällig ergeben und darf sich nicht

direkt aus dem ursprünglichen Startknoten ableiten lassen. Das Basisquadrat muß daher modifiziert werden.

Leicht verifiziert man, daß durch Vertauschen von Zeilen oder Spalten die Eigenschaften eines Lateinischen Quadrats nicht verletzt werden, da in jeder Zeile und Spalte jedes Element nach wie vor genau einmal auftritt. Da die Anforderungen an ein Lateinisches Quadrat erhalten bleiben, bleiben auch die Eigenschaften bezüglich der balancierten Verteilung über die Richtungen in jedem Schritt erhalten. Für unseren Fall sind derartige Manipulationen des durch das Lateinische Quadrat gegebenen Routing-Planes jedoch nicht hinreichend, da in jeder Zeile nach wie vor alle Dimensionen auftreten und daher das Zwischenziel der Phase I aus dem ursprünglichen Startknoten durch Inversion hervorgeht, die gewünschte Entkopplung also noch nicht geleistet ist. Man geht daher über zu einem partiellen Lateinischen Quadrat (vgl. Anhang A), welches über leere Einträge verfügt, so daß – während Phase I – nicht jedes Paket alle Richtungen durchläuft.

Verfahren C (Kombinatorisches Zufallsverfahren)

Die Zwischenadressen werden mittels eines Zufallszahlengenerators bestimmt. Die zu jedem Paket gehörige Zeile des Basisquadrats gibt die Reihenfolge an, in der die Richtungen zu untersuchen sind, ob eine Abweichung zwischen Start- und Zwischenadresse besteht.

Man kann das Verfahren C auch so auffassen, daß die Zeilen des Basisquadrats mit einer für jedes Paket individuellen Zufallszahl mittels eines bitweisen AND verknüpft werden. Auf diese Weise entsteht eine Familie partieller Lateinischer Quadrate, in der jedes Paket in der ihm zugewiesenen Zeile des Basisquadrats durch Verknüpfung mit der individuellen Zufallszahl bestimmte Richtungen auswählt.

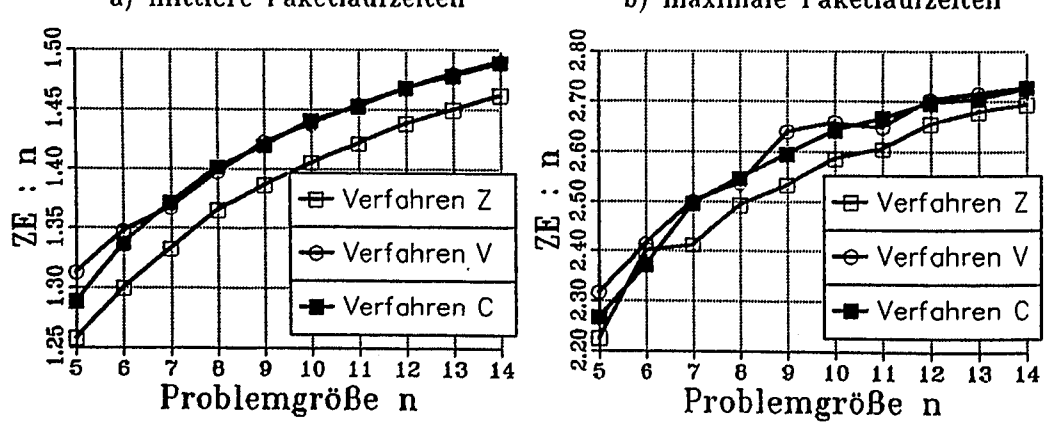


Abb. 50. Analyse des Verfahrens C: (Modell A / Priorität FIFO / Permutation)

Für den Fall (Modell A / Permutation / Priorität FIFO) zeigt das Verfahren C ein mit dem Verfahren von Valiant vergleichbares Verhalten, das Referenzverfahren Z arbeitet jedoch etwas günstiger, wie Abb. 50 zeigt.

Gleiches gilt für die FAFI-Priorität; für den Relationsfall liegen alle Verfahren etwa gleichauf. Die Verhältnisse ändern sich jedoch erheblich, wenn zu Modell B übergegangen wird, wie Abb. 51 zeigt.

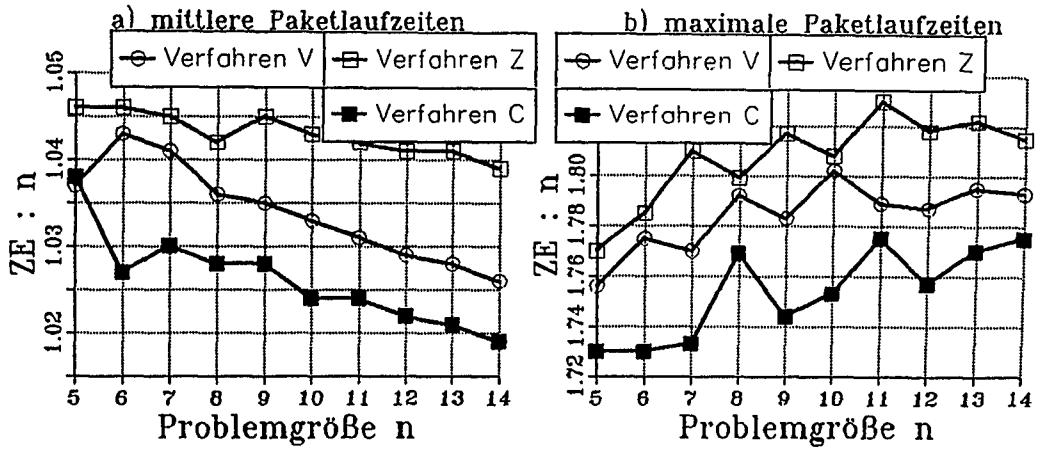


Abb. 51. Analyse des Verfahrens C: (Modell B / Priorität FIFO / Permutation)

Während bereits das Verfahren von Valiant einen Komplexitätsvorteil gegenüber dem Referenzverfahren Z erzielt, arbeitet das kombinatorische Verfahren C noch einmal deutlich besser als das Valiantsche Verfahren. Da dies auch für die übrigen Parameterkonstellationen mit Modell B gilt, können mit diesem neuen Ansatz also Verbesserungen erzielt werden, die sich sowohl für die mittleren Paketlaufzeiten als auch für die maximalen Paketlaufzeiten über die gesamte Spanne der Problemgrößen erstrecken.

7.3.2.2 Verfahren auf der Basis von Differenzmengen

Differenzmengen (vgl. Anhang A) können zur Erzeugung zyklischer symmetrischer balancierter Blockpläne verwendet werden und haben sich darüber hinaus als Standardbeschreibungsform eines zyklischen BIBD etabliert. Für beide gilt die notwendige Bedingung $\lambda(v-1) = k(k-1)$. Diese Bedingung ist ebenso für die Existenz binärer Pseudozufallsfolgen (PRBS) notwendig [H&A,70]. Im Falle der binären Pseudozufallsfolgen bezeichnet der Parameter v die Periodenlänge der Zufallsfolge, während der Parameter k die Anzahl der Einsen in einer Periode angibt. Der Parameter λ ergibt sich aus der Differenz $\lambda = k - n_1$, wobei n_1 die Anzahl der Gruppen aufeinanderfolgender Einsen in einer Periode bezeichnet.

Die speziellen Pseudozufallsfolgen, die sich hier zur Adreßgenerierung eignen, sind die PRBS mit maximaler Periodenlänge, die mit speziellen Schieberegistern erzeugt werden können. Die Schieberegistertechnik, auf die auch das Simulationsprogramm zurückgreift, soll im Rahmen dieser Arbeit nur kurz erläutert werden. Für Weiterführendes über

Schieberegisterfolgen, ihre Anwendungen und Zusammenhänge mit anderen Bereichen der diskreten Mathematik sei hier beispielhaft auf die Monographie von Lüneburg [Lün,79] verwiesen. In der Literatur wird gezeigt, daß sich maximal-periodische Schieberegister aufgrund ihrer Eigenschaften, beispielsweise, daß Nullen und Einsen sehr ausgewogen auftreten, in optimaler Weise als Zufallszahlengeneratoren eignen [Gol,67].

Ein **linearer Schaltkreis** auf einem Galoisfeld ist ein Schaltwerk, das aus den drei Bausteinen Addierer, Multiplizierer und Verzögerungsglied (*delay*) aufgebaut ist und durch einen gerichteten azyklischen Graphen dargestellt werden kann. Der Multiplizierer leistet die Multiplikation eines eingegebenen Körperelementes mit einem festen Körperelement. Für den Fall, daß man auf dem Boole'schen Körper, $GF(2)$, arbeitet, sind die Multiplikatoren 1 und 0 möglich, entsprechend einem Passieren des Elementes bzw. einem Unterbrechen der Leitung. Lineare Schaltkreise spielen eine bedeutende Rolle in der Codierungstheorie [O&V,86]. Ein **rückgekoppeltes Schieberegister** ist ein linearer Schaltkreis, dessen Registerinhalte (Delay-Inhalte) in jedem Takt um ein Glied nach rechts geschoben werden und der Output aus dem am weitesten links befindlichen Addierer als Input in das am weitesten links befindliche Register dient (Abb. 52).

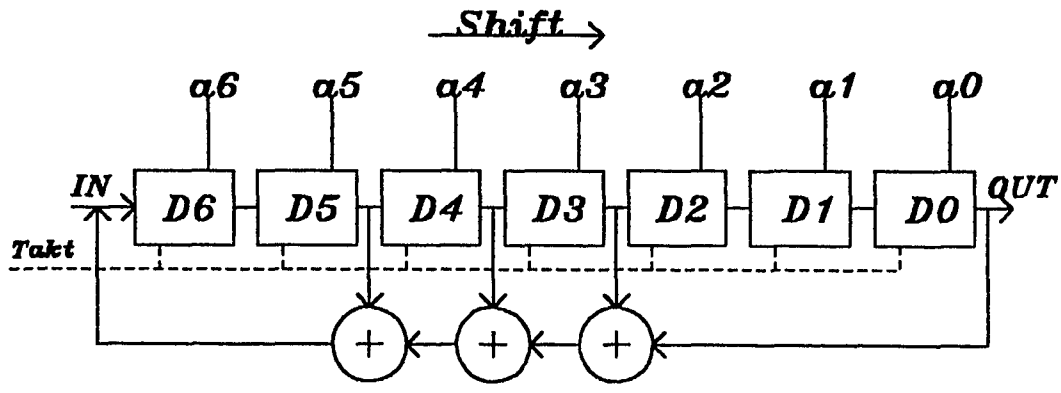


Abb. 52. 7-stufiges rückgekoppeltes Schieberegister: Die Inhalte der Delays 0 bis 6 können an den Stellen a0 bis a6 als 7-stellige Binärzahl abgegriffen werden.

Man kann ein rückgekoppeltes d-stufiges Schieberegister, d.h. mit d Delays, beispielsweise derart als Generatorsystem nutzen, daß man in jedem Takt die Inhalte der Register abgreift und als d-stellige Zahl über dem Galoisfeld auffaßt. Im folgenden wird als endlicher Körper stets $GF(2)$ angenommen.

Ein rückgekoppeltes d-stufiges Schieberegister heißt **maximal-periodisch**, wenn es, nach Initialisierung mit einem von Null verschiedenen Wert, d.h. zumindest ein Delay enthält den Wert 1, in $N-1=2^d-1$ aufeinanderfolgenden Takten alle Zahlen von 1 bis N einmal produziert. Dies wiederholt sich in Perioden von $N-1$ Takten. Die Bestimmung derjenigen Delays, deren Inhalt in einem d-stufigen Schieberegister als Input in einen Addierer dienen muß, um ein maximal-periodisches Schieberegister zu erhalten, kann über die Koeffizienten der irreduziblen Polynome vom Grad d über $GF(2)$ erfolgen [Lün,79].

Verfahrensbedingt, da die Null nicht als Startwert dienen kann und auch im Verlauf des Schiebens nicht auftritt, ist bei den $N = 2^n$ zufällig erzeugten Zwischenadressen des Verfahrens P jede Zahl zwischen 1 und $N - 1$ einmal vertreten, mit Ausnahme des Startwerts, der am Anfang und am Ende der Folge auftritt, da nur $N - 1$ verschiedene Zahlen erzeugt werden können.

Verfahren P (PRBS-Zufallsverfahren)

Die Zwischenadressen werden mittels eines Schieberegisters erstellt. Für den Hypercube der Dimension n wird das n-stufige maximal-periodische Schieberegister benutzt. Das Schieberegister wird mit einer von Null verschiedenen Zufallszahl initialisiert.

Während die kombinatorischen Verfahren, die auf Lateinischen Quadraten basieren, spürbare Vorteile nur für den Fall der *n-port-communication* (Modell B) erzielen konnten, zeigen sich für das auf Differenzmengen basierende Verfahren P gerade für Modell A gute Ergebnisse. In Abb. 53 ist beispielhaft die Konstellation mit Permutation und Priorität FIFO abgetragen, deren qualitative Aussage ebenso für die anderen Konstellationen mit Modell A gilt.

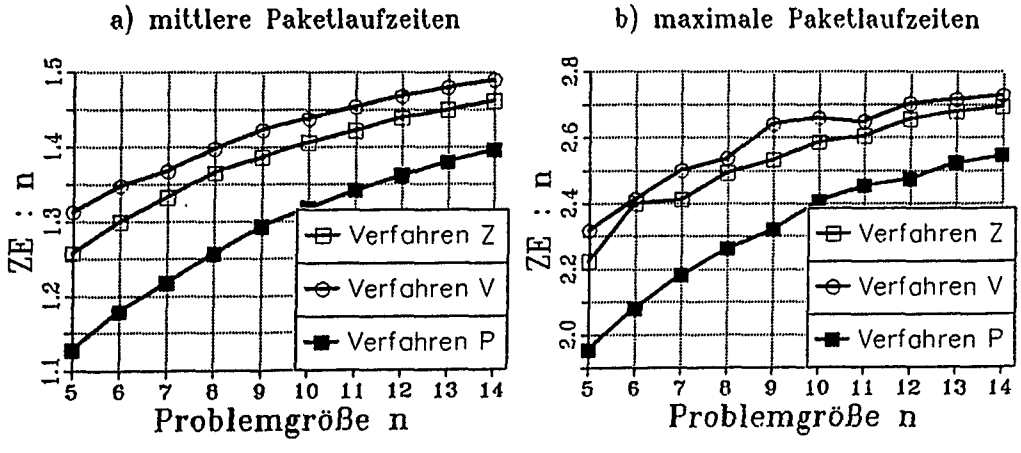


Abb. 53. Analyse des Verfahrens P: (Modell A / Priorität FIFO / Permutation)

Im Gegensatz zu den Verfahren E und C, die auf Lateinischen Quadraten beruhen, ist keine Leistungsabhängigkeit von den Modellen gegeben, da wie Abb. 54 zeigt, auch für die korrespondierende Konstellation mit Modell B das kombinatorische Verfahren P die Referenzverfahren übertrifft.

Andererseits ist innerhalb der Konstellationsmöglichkeiten mit Modell B eine Abhängigkeit von der Kommunikationsaufgabe gegeben; dazu ist Abb. 55 heranzuziehen.

In der Situation der Permutation, in der laut Definition zu Anfang und Ende der Kommunikationsaufgabe eine gleichmäßige Verteilung der Pakete über das System gegeben

ist, wird eine besonders gute Lastbalancierung während des gesamten Kommunikationsproblems erreicht, wenn auch die Zwischenadressen eine Permutation darstellen, wie es durch die Schieberegistersequenz annährend gegeben ist. Dies wird durch die gute Leistung des Verfahrens P in Abb. 54 verdeutlicht. Im Falle der Relation kann das Verfahren P zwar das Referenzverfahren Z unterbieten, ist jedoch dem Verfahren von Valiant nur für einzelne Hypercube-Dimensionen überlegen, wie Abb. 55 zeigt.

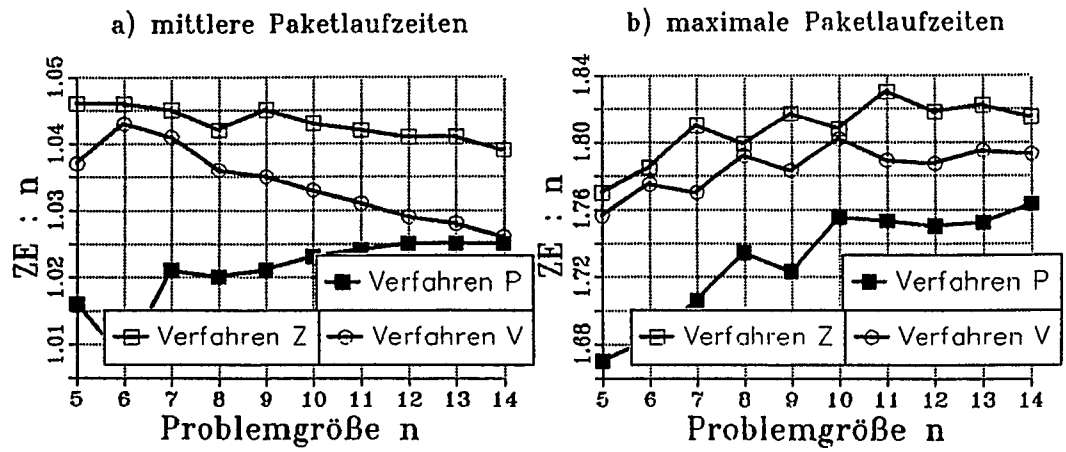


Abb. 54. Analyse des Verfahrens P: (Modell B / Priorität FIFO / Permutation)

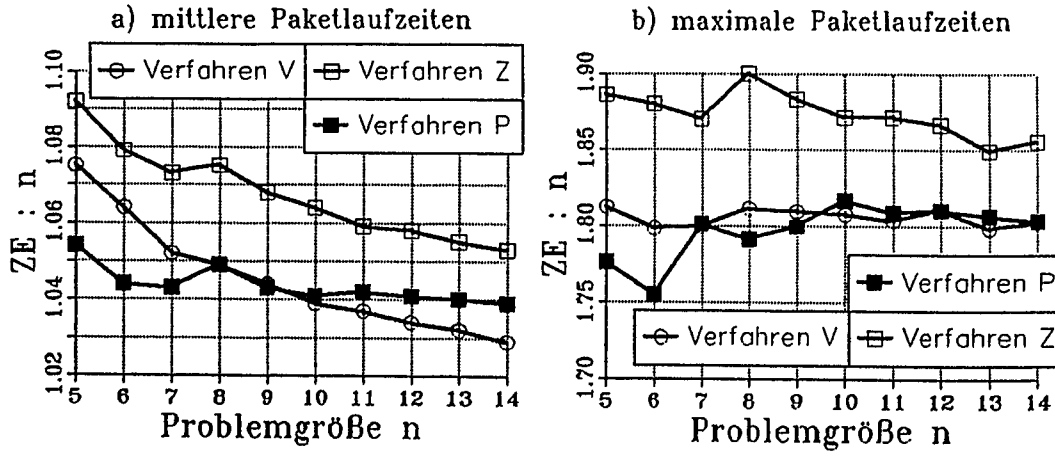


Abb. 55. Analyse des Verfahrens P: (Modell B / Priorität FIFO / Relation)

7.3.3 Zufalls-Routing-Verfahren mit pfadlängenreduzierter Phase I

7.3.3.1 Verfahren auf der Basis Lateinischer Quadrate

Die mit dem Referenzverfahren Y eingeführte Technik wird nun im Verfahren M aufbauend auf dem voll-zufälligen, kombinatorischen Verfahren C angewendet.

Verfahren M (modifiziertes Verfahren C)

Durch einen Zufallszahlengenerator werden in getrennten Durchläufen pro Paket zwei Zufallszahlen erzeugt. Diese werden dann mit bitweisem AND verknüpft und bilden dann eine neue Zufallszahl. Die gesetzten Bits dieser Zufallszahl liefern die zu invertierenden Stellen der Startadresse, woraus die zufällige Zwischenadresse erhalten wird. Die zu jedem Paket gehörige Zeile des Basisquadrats gibt die Reihenfolge an, in der die Richtungen zu untersuchen sind, ob eine Abweichung zwischen Start- und Zwischenadresse besteht.

Ähnlich wie in dem zugrundeliegenden Verfahren C ergibt sich auch hier eine Abhängigkeit der Leistung des zu untersuchenden kombinatorischen Verfahrens von den Modellen. Für Modell A arbeitet das Referenzverfahren Y im allgemeinen besser, nur für den Relationsfall werden bei den maximalen Paketlaufzeiten die besseren Werte durch das kombinatorische Verfahren M geliefert (Abb. 56).

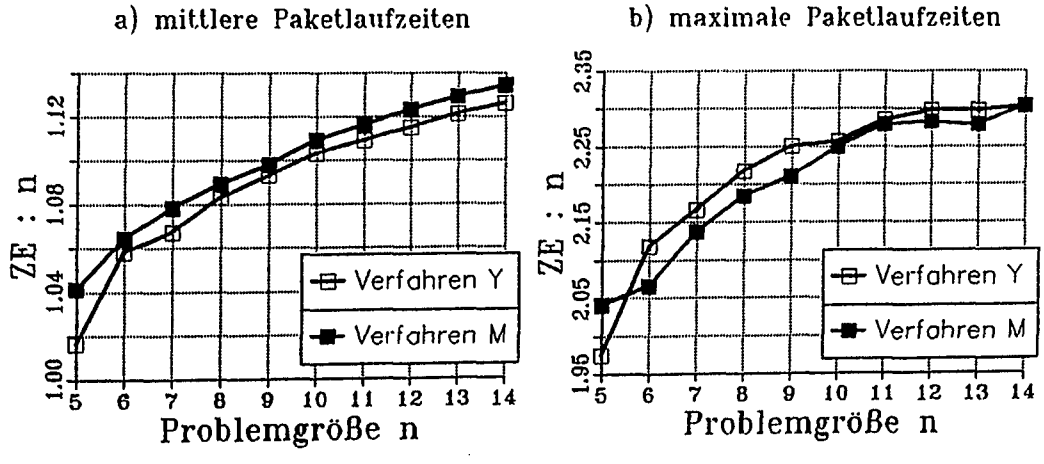


Abb. 56. Analyse des Verfahrens M: (Modell A / Priorität FIFO / Relation)

Für alle Konstellationen mit Modell B, und zwar sowohl für mittlere als auch maximale Paketlaufzeiten, behält das auf Lateinischen Quadraten basierende Verfahren M gegenüber dem Referenzverfahren Y eindeutig die Oberhand, wie Abb. 57 beispielhaft zeigt.

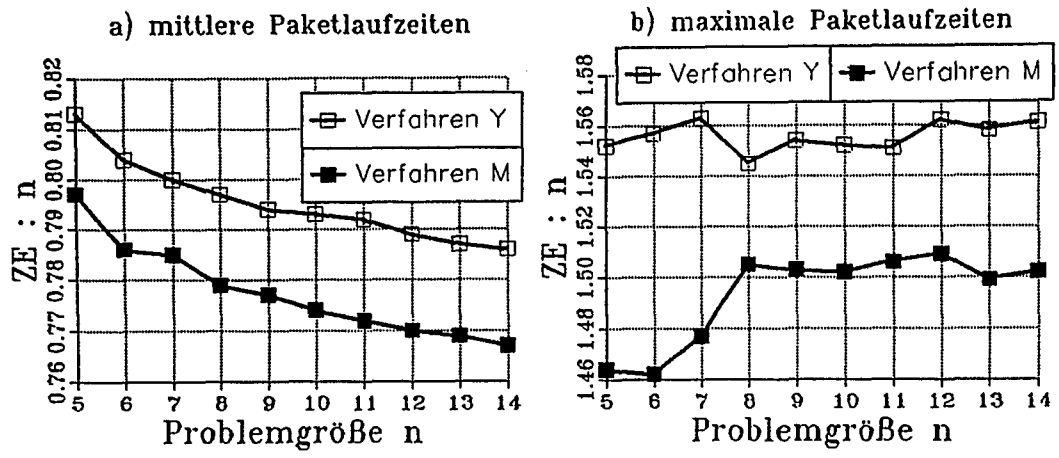


Abb. 57. Analyse des Verfahrens M: (Modell B / Priorität FIFO / Relation)

7.3.3.2 Verfahren auf der Basis von Differenzmengen

Die mit dem Referenzverfahren Y eingeführte Technik wird nun im Verfahren Q analog zu dem voll-zufälligen Verfahren P mit Schieberegistern durchgeführt.

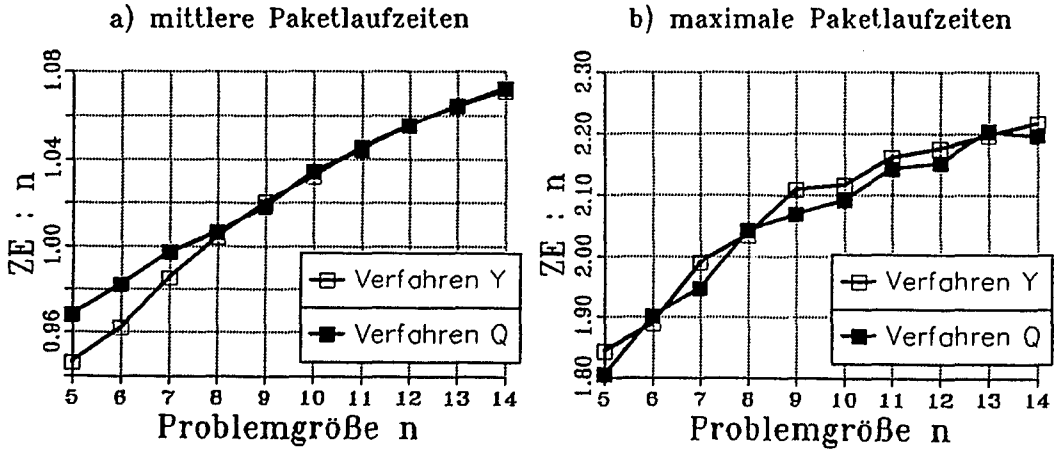


Abb. 58. Analyse des Verfahrens Q: (Modell A / Priorität FIFO / Permutation)

Abb. 58 zeigt, daß die Güte des auf Differenzmengen basierenden kombinatorischen Verfahrens Q vergleichbar ist mit derjenigen des Referenzverfahrens Y. Für alle übrigen Fälle werden vergleichbare Resultate mit dem Verfahren Q erzielt; auch das schlechtere Abschneiden gegenüber dem Referenzverfahren für kleine Problemgrößen bei den mittleren Paketlaufzeiten ist grundsätzlich zu beobachten.

Verfahren Q (modifiziertes Verfahren P)

Mittels eines Schieberegisters werden in getrennten Durchläufen pro Paket zwei Zufallszahlen erzeugt. Diese werden dann mit bitweisem AND verknüpft und bilden dann eine neue Zufallszahl mit im Mittel $n/4$ gesetzten Bits, die die Stellen liefern, an denen die jeweilige Startadresse zu invertieren ist. Die so modifizierten Startadressen bilden dann die zufälligen Zwischenadressen des Verfahrens.

7.3.4 Vergleich deterministischer und zufälliger Verfahren

Es gilt nun noch zu verifizieren, daß die entwickelten kombinatorischen Verfahren auch bei praxisrelevanten, wichtigen Kommunikationsaufgaben tatsächlich bessere Komplexitäten erzielen können als deterministische Verfahren und keine ungewollten Systematiken auftreten, die zu ungünstigem Verhalten bei deterministischen Routing-Aufgaben führen. Hierzu wird als Kommunikationsaufgabe die *Matrixtransposition* gewählt. Da es sich um eine deterministische Kommunikationsaufgabe handelt, ist für das deterministische Verfahren D nur ein einziger Lauf notwendig. In der graphischen Darstellung sind, wegen der hinreichend großen Differenzen, wie in Abschnitt 7.2 tatsächliche Zeiteinheiten auf der Ordinate abgetragen.

7.3.4.1 Verfahren auf der Basis Lateinischer Quadrate

Abb. 59 zeigt, daß das kombinatorische Zufallsverfahren C und die zugehörige Variante M mit pfadlängenreduzierter Phase I ab Problemgröße 10 tatsächlich effizienter arbeiten als das deterministische Verfahren D.

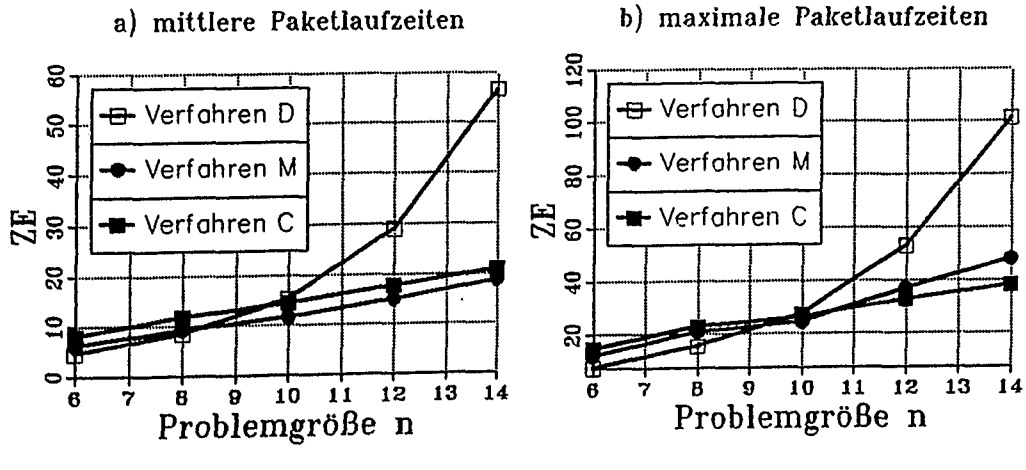


Abb. 59. Effizienz kombinatorischer Zufalls-Routing-Verfahren:
(Modell A / Priorität FIFO / Matrixtransposition)

7.3.4.2 Verfahren auf der Basis von Differenzmengen

Abb. 60 zeigt, daß das kombinatorische Zufallsverfahren P und die zugehörige Variante Q mit pfadlängenreduzierter Phase I für massiv-parallele Kommunikationsaufgaben deutlich effizienter arbeiten als das deterministische Verfahren D.

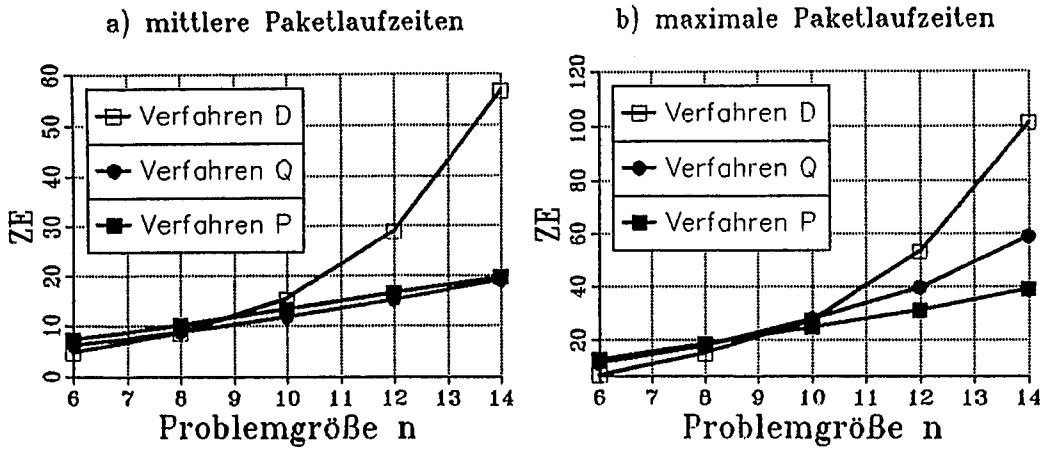


Abb. 60. Effizienz kombinatorischer Zufalls-Routing-Verfahren:
(Modell A / Priorität FIFO / Matrixtransposition)

Unter Beachtung der Ergebnisse für mittlere Paketlaufzeiten ist festzustellen, daß Zufallsverfahren mit pfadlängenreduzierter Phase I durchaus eine hinreichende Entkopplung von Start- und Zielknoten gewährleisten können, so daß sie sicherlich für viele Kommunikationsaufgaben zumindest im Bereich asynchroner Algorithmen den vollzufälligen Verfahren vorzuziehen sind.

Weitere Untersuchungen ergaben, daß die zufälligen Verfahren im Modell B mit *n-port-communication* erst bei größeren Kommunikationsproblemen das deterministische Verfahren übertreffen, was auf die höhere Bandbreite des Modells und die daraus resultierende geringere Anfälligkeit für Verstopfungen zurückzuführen ist [Bur,93].

8 Zusammenfassung und Ausblick

Die Gültigkeit des Karpschen Postulats über die strukturelle Verwandschaft von parallelen Strukturen und optimalen Versuchsplänen, dessen Verifikation als Kernprojekt der vorliegenden Arbeit aufzufassen ist, wurde mit dem vorgestellten Ansatz zu einer kombinatorischen Theorie von Multiprozessorrechnern mit lokalem Speicher und statischem Verbindungsnetzwerk, die sich über die Kapitel 3 bis 6 erstreckt, nachweislich bestätigt. Der Ansatz umfaßt eine einheitliche Beschreibungsmethodik für Parallelrechner und parallele Algorithmen und entspricht somit den Anforderungen an ein Integriertes Konzept; er erlaubt, basierend auf den Versuchplandarstellungen paralleler Strukturen, die kombinatorische Formulierung der wichtigsten Aufgaben Mapping, Partitionierung und Einbettung. Basierend auf der kombinatorischen Klassifizierung paralleler Strukturen und ausgehend von der Behandlung der Kernbereiche liegen mit der Verwendung der in Abschnitt 2.3 vorgestellten Transformationstechnik die wesentlichen Voraussetzungen für ein kombinatorisches Tragwerk des noch zu konstruierenden methodischen Gebäudes der Parallelverarbeitung vor.

Die Überlegenheit des Versuchsplankonzeptes gegenüber dem Graphentheoriekonzept resultiert aus der höheren Mächtigkeit der zugrundeliegenden Basisstruktur und der dadurch begründeten größeren Flexibilität oder gar universellen Einsetzbarkeit. Hier seien drei charakteristische Beispiele genannt: Mit Hilfe kombinatorischer Versuchspläne können auch Rechnerarchitekturen in ein konsistentes Schema integriert werden, die auf Bussen aufbauen, was im Falle der Graphen nicht möglich ist, da sie einen 2-uniformen Spezialfall der auch als Hypergraphen bezeichneten Versuchspläne darstellen. Auf Algorithmenseite betrachte man den Fall, daß innerhalb einer bestimmten Phase eines Algorithmus eine Teilmenge der dort vorliegenden Module jeweils paarweise miteinander kommunizieren müssen. Mit der Versuchsplanstruktur kann man diese Situation durch einen einzelnen Block beschreiben, während bei der Beschreibung mittels Graphen nur die Einzelbeziehungen modellierbar sind. Das Versuchsplankonzept bietet hier also gegenüber dem Graphenkonzept die Möglichkeit, sogenannte *cluster* zu erkennen, welches beim Mapping eine wertvolle Hilfe darstellt. Ein weiterer Aspekt zielt auf die Automatisierung und die damit zusammenhängende Implementierung eines Modells. Die Versuchspläne bilden hierfür eine wesentlich detailliertere Informationsbasis als Graphen. Die Notwendigkeit automatischer Programmoptimierung und -parallelisierung wird in der Literatur vielfach hervorgehoben, so daß man für zukunftsträchtige Modellierungskonzepte eine automatische Umsetzung ermöglichen sollte. Andererseits kann man aufgrund der Obermengensituation die gewohnte und als Quasistandard akzeptierte Modellierung mit Graphen in das Versuchsplankonzept integrieren. Eine vergleichbare Konstellation wird in der Informatik auch mit Aufwärtskompatibilität bezeichnet und stellt eine wesentliche Voraussetzung für die Einführung bzw. die hinreichende Akzeptanz eines neuen Konzeptes dar.

Über die reine Modellbildung hinaus war es Anliegen dieser Arbeit, die ausgereifte Theorie der Kombinatorik optimaler Versuchspläne für Problemstellungen aus der Parallelverarbeitung nutzbar zu machen. Dazu wurden die Problemstellungen der Parallelverarbeitung auf Operationen auf Versuchsplänen transformiert und so offene Probleme aus dem Bereich paralleler Strukturen auf bereits allgemein gelöste Probleme aus der Kombinatorik zurückgeführt. Die wichtigsten Transformationspaare sind (Mapping, Isomorphie), (Partitionierung, Resolution) und (Einbettung, Emulation). Darüber hinaus bietet die Fülle der Werkzeuge, die in der Theorie kombinatorischer

Versuchpläne bereitgestellt werden, eine Reihe weiterer Anwendungsmöglichkeiten, die teilweise nur angedeutet werden konnten, wie die Behandlung des Zugriffs auf parallele Speicher, der Entwurf neuer Topologien für Rechner mit statischem Verbindungsnetzwerk sowie Ansätze zur orginären Parallelisierung vieler Algorithmen. Zukünftige Anwendungsfelder kombinatorischer Versuchspläne könnten auf Ersetzungsheuristiken für Cache-Inhalte, die Organisation verteilter Datenstrukturen und die Modellierung fehlertoleranter Systeme abzielen.

Die statistischen Eigenschaften der Versuchspläne, und damit ihre ursprüngliche praktische Anwendung, wurden für den Entwurf neuer Routing-Verfahren genutzt, wobei sich diese Arbeit auf eine einzige Topologie beschränken mußte. Bereits für deterministische Verfahren konnte gezeigt werden, daß die schon für andere Bereiche der Parallelverarbeitung wertvolle Balance der Versuchspläne sich auch auf die Lastverteilung in einem Kommunikationssystem auswirken kann und je nach Kommunikationsaufgabe zu mehr oder minder großen Komplexitätsreduktionen führt. Für die Zukunft von größerem Interesse sind jedoch die Verfahren des Randomized Routing, für die ebenso teilweise beachtliche Effizienzgewinne mit kombinatorischen Techniken erzielt werden konnten.

Im Rahmen diese Arbeit konnten insbesondere auch bei den Untersuchungen der Routing-Verfahren nur prinzipielle Überlegungen angestellt werden; hier bieten sich sicherlich neue Anknüpfungspunkte. So wäre es sinnvoll, analoge Untersuchungen auch für andere Topologien als den hier gewählten Hypercube anzustellen. Dabei könnte man sich zunächst, in Reaktion auf die gegenwärtigen Entwicklungen auf dem Architektursektor, den maschenverbundenen Strukturen widmen. Längerfristig werden auch Topologien mit dynamischen Verbindungsnetzwerken wieder an Bedeutung gewinnen, so daß die Weiterentwicklung des gesamten Modells für diesen Sektor naheliegt; das erfordert aber die dafür notwendige Klassifizierung dynamischer Netzwerke durch kombinatorische Versuchspläne.

Als interessantester, zugleich aber schwieriger Schritt in einer Weiterentwicklung der vorliegenden Arbeit muß die Implementierung des vorgestellten kombinatorischen Modells der Parallelverarbeitung angesehen werden. Hier sind noch sehr komplexe Probleme zu bewältigen, zu denen die Bestimmung der Phasen eines parallelen Algorithmus und die damit zusammenhängende Festlegung separat ausführbarer Module zählen. So schwierig diese Probleme auch sein mögen, so lohnend ist ihre Bewältigung: Nur dann, wenn es gelingt, den inhärenten Parallelismus eines gegebenen Algorithmus hinreichend gut durch automatische Umsetzung zu nutzen, nur dann wird der Durchbruch der Parallelrechner in allgemeinen Anwendungen gelingen. Der durchschnittliche Anwender ist vielleicht noch in der Lage, parallele Programmierung auf einem System mit gemeinsamem Speicher zu bewältigen, wobei ihm auch das oft Mühe bereitet, die Kommunikation mittels Austausch von Nachrichten, die für die massiv-parallelen Systeme der Zukunft unumgänglich ist, überfordert ihn in der Regel. Hier ist eine Hilfestellung des Rechnersystems langfristig unverzichtbar, sei es durch Erweiterungen des Betriebssystems, sei es durch zusätzliche Oberflächen, die etwa ein Konzept mit virtuell gemeinsamem Speicher unterstützen, sei es durch Entwicklung von effizienten Datenflußarchitekturen oder vielleicht durch Implementierung eines ganzheitlichen Modells ähnlich dem hier vorgestellten. Ohne vergleichbare benutzerunterstützende Konzepte wird man

die großen Chancen, die der Übergang zum massiven Parallelismus bietet, nicht nutzen können.

Der Trend zu massiver Parallelität aber, der die Parallelverarbeitung zur Herausforderung der Informatik in den achtziger und neunziger Jahren hat werden lassen, ist ungebrochen und erklärt sich aus seiner Eigendynamik, die von Upfal und Wigderson in folgende Worte gekleidet wird [U&W,84]: „...in parallel computation, appetite increases with eating: the more processors we can have in parallel computers, the larger the problems we want to solve".

Literaturverzeichnis

[AGR,68] Abraham, C. T. / Ghosh, S. P. / Ray-Chaudhuri, D. K.
File Organization Schemes Based on Finite Geometries
Information and Control 12 (1968), 143 – 163

[A&B,86] Adiga, A. K. / Browne, J. C.
A Graph Model for Parallel Computations Expressed in the Computation Structures Language
Proceedings of the 1986 International Conference on Parallel Processing, 880 – 886

[Agr,83] Agrawal, D. P.
Graph Theoretical Analysis and Design of Multistage Interconnection Analysis
IEEE Transactions on Computers C-32 (1983), 637 – 648

[AHU,74] Aho, A. V. / Hopcroft, J. E. / Ullman, J. D.
The Design and Analysis of Computer Algorithms
Addison-Wesley
Reading, MA 1974

[Ahr,18] Ahrens, W. (ed.)
Mathematische Unterhaltungen und Spiele II
B. G. Teubner
Leipzig 1918

[A&&,91] Aiello, W. A. et al.
Fast Algorithms for Bit-Serial Routing on a Hypercube
Mathematical Systems Theory 24 (1991), 253 – 271

[A&K,86] Akers, S. B. / Krishnamurthy, B.
A Group Theoretic Model for Symmetric Interconnection Networks
Proceedings of the 1986 International Conference on Parallel Processing, 216 – 223

[Ale,82] Aleliunas, R.
Randomized Parallel Communication
Proceedings of the ACM Symposium on Principles of Distributed Computing, Ottawa 1982, 60 – 72

[ABM,87] Alon, N. / Barak, A. / Manber, U.
On Disseminating Information Reliably without Broadcasting
Proceedings of the 7th International Conference on Distributed Computing Systems, Berlin (1987), 74 – 81

[BST,89] Bal, H. E. / Steiner, J. G. / Tanenbaum, A. S.
Programming Languages for Distributed Computing Systems
ACM Computing Surveys 21 (1989), 261 – 322

[BCT,83] Barlotti, A. / Ceccherini, P. V. / Tallini, G. (eds.)
Combinatorics '81
Proceedings of the International Conference on Combinatorial Geometrics and their Applications, Rome 1981
Annals of Discrete Mathematics **18** (1983)

[Bat,68] Batcher, K. E.
Sorting Networks and Their Applications
Proceedings of AFIPS 1968 Spring Joint Computer Conference, 307 – 314

[Bat,86] Batten, L. M.
Combinatorics of Finite Geometries
Cambridge University Press
Cambridge 1986

[Bec,64] Beckenbach, E. F. (ed.)
Applied Combinatorial Mathematics
John Wiley & Sons
New York, NY 1964

[Ben,65] Benes, V. E.
Mathematical Theory of Connecting Networks and Telephone Traffic
Academic Press
New York, NY 1965

[BKS,91] Berg, T. B. / Kim, S.-D. / Siegel, H. J.
Impact of Temporal Juxtaposition on the Isolated Phase Optimization Approach to Mapping an Algorithm to Mixed-Mode Architectures
Proceedings of the 1991 International Conference on Parallel Processing I, 110 – 118

[Ber,73] Berge, C.
Graphs and Hypergraphs
North Holland
Amsterdam 1973

[B&S,84] Berman, F. / Snyder, L.
On Mapping Parallel Algorithms into Parallel Architectures
Proceedings of the 1984 International Conference on Parallel Processing, 307 – 309

[Ber,76] Berman, G.
The Application of Difference Sets to the Design of a Balanced Multiple-Valued Filing Scheme
Information and Control **32** (1976), 128 – 138

[B&&,83] Bermond, J.-C. et al.
Graphs and Interconnection Networks: Diameter and Vulnerability
in [Llo,83], 1 – 30

[BBB,89] Bermond, J.-C. / Berrada, K. / Bond, J.
Extensions of Networks with Given Diameter
in [Bol,89], 31 – 40

[BBS,84] Bermond, J.-C. / Bond, J. / Saclé, J. F.
Large Hypergraphs of Diameter 1
in [Bol,84], 19 – 28

[B&&,91] Bertsekas, D. P. et al.
Optimal Communication Algorithms for Hypercubes
Journal of Parallel and Distributed Computing 11 (1991), 263 – 275

[B&H,91] Beth, T. / Hatz, V.
A Restricted Crossbar Implementation and its Applications
ACM Computer Architecture News 19 (1991), 12 – 16

[B&H,92a] Beth, T. / Hatz, V.
Design Machines: Algebraically well Described Interconnection Networks
Designs, Codes and Cryptography 2 (1992), 287 – 298

[B&J,83] Beth, T. / Jungnickel, D.
On the Codes Generated by Certain Divisible Designs
in [BCT,83], 87 – 93

[BJL,85] Beth, T. / Jungnickel, D. / Lenz, H.
Design Theory
Bibliographisches Institut
Mannheim 1985

[Beu,82] Beutelspacher, A.
Einführung in die endliche Geometrie
Band 1. Blockpläne
Bibliographisches Institut
Mannheim 1982

[Bhu,84] Bhuyan, L. N.
A Combinatorial Analysis of Multibus Multiprocessors
Proceedings of the 1984 International Conference on Parallel Processing,
225 – 227

[B&L,89] Bier, T. / Loe, K.-F.
Embedding of Binary Trees into Hypercubes
Journal of Parallel and Distributed Computing 6 (1989), 679 – 691

[Big,89] Biggs, N. L.
Discrete Mathematics
Clarendon Press
Oxford 1989

[Bok,81] Bokhari, S. H.
On the Mapping Problem
IEEE Transactions on Computers **C-30** (1981), 207 – 214

[Bok,88] Bokhari, S. H.
Partitioning Problems in Parallel, Pipelined and Distributed Computing
IEEE Transactions on Computers **C-37** (1988), 48 – 57

[Bol,84] Bollobás, B. (ed.)
Graph Theory and Combinatorics
Academic Press
London 1984

[Bol,89] Bollobás, B. (ed.)
Graph Theory and Combinatorics 1988
Proceedings of the Cambridge Combinatorial Conference in Honour of Paul Erdös
Annals of Discrete Mathematics **43**
North Holland
Amsterdam 1989

[B&H,82] Borodin, A. / Hopcroft, J. E.
Routing, Merging and Sorting on Parallel Models of Computation
Proceedings of the 14th Annual ACM Symposium on the Theory of Computing (1982), 338 – 344

[BAG,69] Bose, R. C. / Abraham, C. T. / Ghosh, S. P.
File Organization of Records with Multiple-Valued Attributes for Multi-Attribute Queries
in [B&D,69], 277 – 295

[B&D,69] Bose, R. C. / Dowling, T. A. (eds.)
Combinatorial Mathematics and its Applications
The University of North Carolina Press
Chapel Hill, NC 1969

[BSP,60] Bose, R. C. / Shrikhande, S. S. / Parker, E. T.
Further Results on the Construction of Mutually Orthogonal Latin Squares and the Falsity of Euler's Conjecture
Canadian Journal of Mathematics **12** (1960), 189 - 203

[Bra,88] Brandes, T.
Formale Methoden zur Spezifizierung automatischer Parallelisierung
Hüthig
Heidelberg 1988

[B&H,83] Broomell, G. / Heath, J. R.
Classification Categories and Historical Development of Circuit Switching Topologies
ACM Computing Surveys **15** (1983), 95 – 133

[Bro,85] Browne, J. C.
Formulation and Programming of Parallel Computations: A Unified Approach
Proceedings of the 1985 International Conference on Parallel Processing, 624 – 631

[B&K,71] Budnik, P. / Kuck, D. J.
The Organization and Use of Parallel Memories
IEEE Transactions on Computers **C-20** (1971), 1566 – 1569

[Bur,89] Burg, H. C.
Implementierung und Untersuchung paralleler Algorithmen zur Schnellen Fourier-Transformation auf CRAY-Mehrprozessorrechnern
Spezielle Berichte der KFA Jülich **Jül-Spez-520**
Forschungszentrum Jülich
Jülich 1989

[Bur,93] Burg, H. C.
A Program for the Simulation of Deterministic and Randomized Routing on the Hypercube
Interner Bericht des Zentralinstituts für Angewandte Mathematik **93-06**
Forschungszentrum Jülich
Jülich 1993

[B&H,92b] Burg, H. C. / Helin, J.
1991 International Conference on Supercomputing. Conference Report
Parallel Computing **18** (1992), 467 – 471

[C&L,75] Cameron, P. J. / Lint, J. H. van
Graph Theory, Coding Theory and Block Designs
London Mathematical Society Lecture Note Series **19**
Cambridge University Press
Cambridge 1975

[Cam,76] Cameron, P. J.
Parallelisms of Complete Designs
London Mathematical Society Lecture Note Series **23**
Cambridge University Press
Cambridge 1976

[Cat,79] Cattermole, K. W.
Graph Theory and Communications Networks
in [W&B,79], 17 – 57

[C&C,88] Chan, M. Y. / Chin, F. Y. L.
On Embedding Rectangular Grids in Hypercubes
IEEE Transactions on Computers **C-37** (1988), 1285 – 1288

[C&L,89] Chen, G.-I. / Lai, T.-H.
Virtual Subcubes and Job Migration in a Hypercube.
Proceedings of the 1989 International Conference on Parallel Processing II, 73 – 76

[Che,88] Chen, M. C.
Mapping Parallel Algorithms onto General Purpose Parallel Machines
Proceedings of the 21st Annual Hawaii International Conference on System Sciences (1988), 131 – 141

[Che,58] Chew, V. (ed.)
Experimental Designs in Industry
John Wiley & Sons
New York, NY 1958

[C&F,83] Chiang, Y. P. / Fu, K. S.
Matching Parallel Algorithms and Architecture
Proceedings of the 1983 International Conference on Parallel Processing, 374 – 380

[Clo,53] Clos, C.
A Study of Non-Blocking Switching Networks
The Bell System Technical Journal **32** (1953), 406 – 424

[C&O,89] Colbourn, C. J. / Oorschot, P. C. van
Applications of Combinatorial Designs in Computer Science
ACM Computing Surveys **21** (1989), 223 – 250

[Col,85] Colbourn, M. J.
Algorithmic Aspects of Combinatorial Design: A Survey
Annals of Discrete Mathematics **26** (1985), 67 – 136

[C&V,86] Cole, R. / Vishkin, U.
Approximate and Exact Parallel Scheduling with Applications to List, Tree and Graph Problems
Proceedings of 27th IEEE International Symposium on the Foundations of Computer Science (1986), 478 – 491

[Cra,91] Cray Research (ed.)
Cray Standard C Programmer's Reference Manual **SR-2074**
Cray Research, Inc.
Mendota Heights, MN 1991

[Cra,92] Cray Research (ed.)
CRAY Y-MP M90 Series Functional Description Manual **HR-04030**
Cray Research, Inc.
Mendota Heights, MN 1992

[D&G,79] Das, M. N. / Giri, N. C.
Design and Analysis of Experiments
Wiley Eastern
New Dehli 1979

[Dem,70] Dembowski, P.
Kombinatorik
Bibliographisches Institut
Mannheim 1970

[D&K,74] Denes, J. / Keedwell, A. D.
Latin Squares and their Applications
Academic Press
New York, NY 1974

[Den,74] Dennis, J. B.
First Version of a Data Flow Procedure Language
Springer Lecture Notes in Computer Science 19 (1974), 362 – 376

[DJL,85] Deshpande, S. R. / Jenevein, R. M. / Lipovski, G. J.
TRAC: An Experience with a Novel Architectural Prototype
Proceedings of the 1985 AFIPS National Computer Conference, 247 – 258

[D&D,89] Dongarra, J. J. / Duff, I. S.
Advanced Architecture Computers
Argonne National Laboratory Report ANL/MCS-TM-57
Argonne, IL 1989

[Dot,84] Doty, K. W.
New Designs for Dense Processor Interconnection Networks
IEEE Transactions on Computers C-33 (1984), 477 – 450

[Duf,86] Duff, M. J. B. (ed.)
Intermediate Level Image-Processing
Academic Press
New York, NY 1986

[Dun,90] Duncan, R.
A Survey of Parallel Computer Architectures
Computer 23 (1990), 5 – 16

[D&H,88] Dutt, S. / Hayes, J. P.
On Allocating Subcubes in a Hypercube Multiprocessor
Proceedings of the 3rd Conference on Hypercube Computers and Applications, Pasadena (1988), 801 – 810

[E&K,93] Esser, R. / Knecht, R.
Intel Paragon XP/S – Architecture and Software Environment
Interner Bericht des Zentralinstituts für Angewandte Mathematik **93-05**
Forschungszentrum Jülich
Jülich 1993

[E&T,63] Estrin, G. / Turn, R.
Automatic Assignment of Computations in a Variable Structure Computer System
IEEE Transactions on Electronic Computers **EC-12** (1963), 755 – 773

[Eul,82] Euler, L.
Recherches sur une nouvelle espèce de quarrees magiques
Verhandelingen uitgegeven door het Zeeuwsch Genootschap der Wetenschappen te Vlissingen **9** (1782), 85 – 239

[F&C,91] Fang, M.-Y. / Chen, W.-T.
Embedding Large Binary Trees to Hypercube Multiprocessors
Proceedings of the 1991 International Conference on Parallel Processing I, 714 – 715

[Fen,81] Feng, T.-y.
A Survey of Interconnection Networks
IEEE Computer **14** (1981), 12 – 27

[Fid,92] Fiduccia, C. M.
Bused Hypercubes and other Pin-Optimal Networks
IEEE Transactions on Parallel and Distributed Systems **3** (1992), 14 – 24

[Fis,26] Fisher, R. A.
The Arrangement of Field Experiments
Journal of the Ministry of Agriculture 33 (1926), 503 – 513

[Fis,60] Fisher, R. A.
The Design of Experiments
Oliver and Boyd
Edinburgh 1960

[Fla,69] Flachsmeyer, J.
Kombinatorik
VEB Deutscher Verlag der Wissenschaften
Berlin 1969

[Fly,66] Flynn, M. J.
Very High Speed Computing Systems
Proceedings of the IEEE **54** (1966), 1901 – 1909

[F&M,85] Fortes, J. A. B. / Moldovan, D. I.
Parallelism Detection and Transformation Techniques Useful for VLSI Algorithms
Journal of Parallel and Distributed Computing **2** (1985), 277 – 301

[F&F,88] Fox, G. C. / Furmanski, W.
Optimal Communication Algorithms for Regular Decompositions on the Hypercube
Proceedings of the 3rd Conference on Hypercube Concurrent Computers and Applications (1988), 648 – 713

[FCS,91] Fulgham, M. / Cypher, R. / Sanz, J.
A Comparison of SIMD Hypercube Routing Strategies
Proceedings of the 1991 International Conference on Parallel Processing **III**, 236 – 243

[Gen,78] Gentleman, W. M.
Some Complexity Results for Matrix Computations on Parallel Processors
Journal of the ACM **25** (1978), 112 – 115

[Ger,87] Gerstein, L. J.
Discrete Mathematics and Algebraic Structures
W. H. Freeman
New York, NY 1987

[Gil,81] Giloi, W. K.
Rechnerarchitektur
Springer
Berlin 1981

[Gil,88] Giloi, W. K.
SUPRENUM: A Trendsetter in Modern Supercomputer Development
Parallel Computing **7** (1988), 283 – 296

[GKL,88] Goldreich, O. / Krawczyk, H. / Luby, M.
On the Existence of Pseudorandom Generators
Proceedings of the 29th IEEE International Symposium on the Foundations of Computer Science (1988), 12 – 24

[Gol,67] Golomb, S. W.
Shift Register Sequences
Holden Day
San Francisco, CA 1967

[G&S,81] Goodman, J. R. / Sequin, C. H.
Hypertree: A Multiprocessor Interconnection Topology
IEEE Transactions on Computers **C-30** (1981), 923 – 933

[G&&,83] Gottlieb, A. et al.
The NYU-Ultracomputer – Designing an MIMD Shared Memory Parallel Computer
IEEE Transactions on Computers C-32 (1983), 175 – 189

[G&B,91] Greenberg, D. S. / Bhatt, S. N.
Routing Multiple Paths in Hypercubes
Mathematical Systems Theory 24 (1991), 295 – 329

[G&L,85] Greenberg, R. I. / Leiserson, C. E.
Randomized Routing on Fat Trees
Proceedings of the 26th IEEE International Symposium on the Foundations of Computer Science (1985), 241 – 249

[G&&,69] Guy, R. et al.
Combinatorial Structures and their Applications
Gordon and Breach
New York, NY 1969

[Hag,91] Hagemann, J.
Combinatorial Structures for Multiprocessor-Systems
Parallel Computing 17 (1991), 699 – 706

[Hal,86] Hall, M.
Combinatorial Theory
John Wiley & Sons
New York, NY 1986

[H&L,75] Hall, M. / Lint, J. H. van (eds.)
Combinatorics
Proceedings of the NATO Advanced Studies Institute, Breukelen 1974
D. Reidel
Dordrecht 1975

[Ham,87] Hamming, R. W.
Information und Codierung
VCH
Weinheim 1987

[H&F,88] Han, Y. / Finkel, R.
An Optimal Scheme for Disseminating Information
Proceedings of the 1988 International Conference on Parallel Processing II, 198 – 203

[Hän,75] Händler, W.
On Classification Schemes for Computers in the Post-von-Neumann-Era
Springer Lecture Notes in Computer Science 24 (1975), 439 – 452

[Hat,92] Hatz, V.
Interconnection Networks Based on Block Designs .
Proceedings of the Second Joint Conference on Vector and Parallel Processing CONPAR 92 – VAPP V, Lyon, September 1992
Springer Lecture Notes in Computer Science 634, 37 – 42

[Hed,77] Hedayat, A.
A Complete Solution to the Existence and Nonexistence of Knut Vik Designs and Orthogonal Knut Vik Designs
Journal of Combinatorial Theory (A) 22 (1977), 331 – 337

[H&F,75] Hedayat, A. / Federer, W. T.
On the Nonexistence of Knut Vik Designs for All Even Orders
The Annals of Statistics 3 (1975), 445 – 447

[HKT,92] Hiranandani, S. / Kennedy, K. / Tseng, C.-W.
Evaluation of Compiler Optimizations for FORTRAN D on MIMD Distributed Memory Machines
Proceedings of the 6th ACM International Conference on Supercomputing (1992), 1 – 14

[H&J,86] Ho, C.-T. / Johnsson, S. L.
Distributed Routing Algorithms for Broadcasting and Personalized Communication in Hypercubes
Proceedings of the 1986 International Conference on Parallel Processing 640 – 648

[H&J,90a] Ho, C.-T. / Johnsson, S. L.
Embedding Meshes in Boolean Cubes by Graph Decomposition
Journal of Parallel and Distributed Computing 8 (1990), 325 – 339

[Hoa,85] Hoare, C. A. R.
Communicating Sequential Processes
Prentice Hall
London 1985

[H&J,88] Hockney, R. W. / Jesshope, C. R.
Parallel Computers 2
Adam Hilger
Bristol 1988

[Hoß,83] Hoßfeld, F.
Parallele Algorithmen
Informatik Fachberichte 64
Springer
Heidelberg 1983

[H&A,70] Hoßfeld, F. / Amadori, R.
On Pseudorandom and Markov Sequences Optimizing Correlation Time-of-Flight Spectrometry
Bericht Jül-684-FF
Kernforschungsanlage Jülich – Institut für Festkörperforschung
Jülich 1970

[H&W,83] Hoßfeld, F. / Weidner, P.
Parallele Algorithmen
Informatik-Spektrum 6 (1983), 142 – 154

[H&J,90b] Huang, C.-H. / Juang, J.-Y.
A Partial Compaction Scheme for Processor Allocation in Hypercube Multiprocessors
Proceedings of the 1990 International Conference on Parallel Processing I, 211 – 217

[H&P,85] Hughes, D. R. / Piper, F. C.
Design Theory
Cambridge University Press
Cambridge 1985

[H&X,90] Hwang, K. / Xu, J.
Mapping Partitioned Program Modules onto Multicomputer Nodes Using Simulated Annealing
Proceedings of the 1990 International Conference on Parallel Processing II, 292 – 293

[IOW,90] Ibrahim, R. L. / Ogden, J. A. / Williams, S. A.
Should Concurrency Be Specified?
in [Rat,90], 247 – 271

[Jac,83] Jacobs, K.
Einführung in die Kombinatorik
Walter de Gruyter
Berlin 1983

[Jam,86] Jamieson, L. H.
The Mapping of Parallel Algorithms to Reconfigurable Parallel Architectures
in [Duf,86], 53 – 63

[Jam,87] Jamieson, L. H.
Characterizing Parallel Algorithms
in [JGD,87], 66 – 93

[JGD,87] Jamieson, L. H. / Gannon, D. B. / Douglass, R. J. (eds.)
The Characteristics of Parallel Algorithms
MIT Press
Cambridge, MA 1987

[J&H,89] Johnsson, S. L. / Ho, C.-T.
Optimum Broadcasting and Personalized Communication in Hypercubes
IEEE Transactions on Computers **C-38** (1989), 1249 – 1268

[K&L,88] Kadar, I. / Liebman, E. J.
Robust Texture Extractors for Real-Time Pyramidal Architectures
Proceedings of the SPIE 1988 Conference on Hybrid Image and Signal Processing **939**, 48 – 57

[K&M,66] Karp, R. M. / Miller, R. E.
Properties of a Model for Parallel Computations: Determinacy, Termination, Queueing
SIAM Journal of Applied Mathematics **14** (1966), 1390 – 1411

[K&W,85] Karp, R. M. / Wigderson, A.
A Fast Parallel Algorithm for the Maximal Independent Set Problem
Journal of the ACM **32** (1985), 762 – 773

[KKL,87] Kilian, J. / Kipnis, S. / Leiserson, C. E.
The Organization of Permutation Architectures with Bussed Interconnections
Proceedings of the 28th IEEE International Symposium on the Foundations of Computer Science (1987), 305 – 315

[K&R,88] Kim, C.-k. / Reed, D. A,
Adaptive Packet Routing in a Hypercube
Proceedings of the 3rd Conference on Hypercube Concurrent Computers and Applications (1988), 625 – 630

[K&B,88] Kim, S. J. / Browne, J. C.
A General Approach to Mapping of Parallel Computations upon Multiprocessor Architectures
Proceedings of the 1988 International Conference on Parallel Processing **III**, 1 – 8

[Kli,92] Klingenberg, W.
Lineare Algebra und Geometrie
Springer
Berlin 1992

[Kne,90] Knecht, R.
Parallelizing Divide-and-Conquer Algorithms – Microtasking versus Autotasking
Springer Lecture Notes in Computer Science **457**, 548 – 558

[K&M,89] Krämer, O. / Mühlenbein, H.
Mapping Strategies in Message-Based Multiprocessor Systems
Parallel Computing **9** (1989), 213 – 225

[KNU,75] Krayl, H. / Neuhold, E. J. / Unger, C.
Grundlagen der Betriebssysteme
Walter de Gruyter
Berlin 1975

[KHI,91] Krishnakumar, N. / Hegde, V. / Iyengar, S. S.
Fault Tolerant Based Embeddings of Quadtrees into Hypercubes
Proceedings of the 1991 International Conference on Parallel Processing III, 244 – 249

[Kri,89] Krishnamurthy, E. V.
Parallel Processing. Principles and Practice
Addison Wesley
Sidney 1989

[KRT,88] Krizanc, D. / Rajasekaran, S. / Tsantilas, T.
Optimal Routing Algorithms for Mesh-Connected Processor Arrays
Proceedings of the 3rd Aegean Workshop on Computing (AWOC 88) VLSI Algorithms and Architectures (1988), 411 – 422

[K&&,85] Kuehn, J. T. et al.
The Use and Design of the PASM
Proceedings of the 1985 Conference on Integrated Technology for Parallel Image Processing, 133 – 152

[K&P,92] Kumar, J. M. / Patnaik, L. M.
Extended Hypercube: A Hierarchical Interconnection Network of Hypercubes
IEEE Transactions on Parallel and Distributed Systems 3 (1992), 45 – 57

[Kun,74] Kung, H. T.
New Algorithms and Lower Bounds for the Parallel Evaluation of Certain Rational Expressions
Proceedings of the 6th Annual ACM Symposium on the Theory of Computing (1974), 323 – 333

[Kun,80] Kung, H. T.
The Structure of Parallel Algorithms
in [Yov,80], 65 – 112

[Kun,82] Kung, H. T.
Why Systolic Architectures?
Computer 15 (1982), 37 – 46

[KLJ,87] Kung, S. Y. / Lewis, P. S. / Jean, S. N.
Canonic and Generalized Mapping from Algorithms to Arrays – A Graph Based Methodology
Proceedings of the Twentieth Annual Hawaii International Conference on System Sciences (1987), 124 – 133

[LJD,93] Lakshmivarahan, S. / Jwo, J.-S. / Dhall, S. K.
Symmetry in Interconnection Networks Based on Cayley-Graphs of Permutation Groups: A Survey
Parallel Computing 19 (1993), 361 – 407

[L&L,90] Lamport, L. / Lynch, N.
Distributed Computing: Models and Methods
in [Lee,90b], 1157 – 1199

[Law,75] Lawrie, D. H.
Access and Alignment of Data in an Array Processor
IEEE Transactions on Computers C-24 (1975), 1145 – 1155

[Lei,90] Leighton, T.
The Role of Randomness in the Design of Parallel Architectures
Proceedings of the 6th MIT Conference on Advanced Research in VLSI, (1990), 177 – 178

[Lei,92a] Leighton, F. T.
Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes
Morgan Kaufmann
San Mateo, CA 1992

[Lei,92b] Leighton, T.
Randomness in the Design of Interconnection Networks
1. Internationales Heinz-Nixdorf-Symposium, Paderborn, 11. November 1992, Vortrag

[Lei,92c] Leiserson, C. E.
The Network Architecture of the Connection Machine CM-5
1. Internationales Heinz-Nixdorf-Symposium, Paderborn, 11. November 1992, Vortrag

[Lee,90a] Leeuwen, J. van (ed.)
Handbook of Theoretical Computer Science
Vol. A: Algorithms and Complexity
Elsevier
Amsterdam 1990

[Lee,90b] Leeuwen, J. van (ed.)
Handbook of Theoretical Computer Science
Vol. B: Formal Models and Semantics
Elsevier
Amsterdam 1990

[Lev,85] Levitan, S. P.
Evaluation Criteria for Communication Structures in Parallel Architectures
Proceedings of the 1985 International Conference on Parallel Processing, 147 – 154

[L&M,85] Lin, T.-c. / Moldovan, D. I.
Tradeoffs in Mapping Algorithms to Array Processors
Proceedings of the 1985 International Conference on Parallel Processing, 719 – 726

[Liu,68] Liu, C. L.
Introduction to Combinatorial Mathematics
McGraw-Hill
New York, NY 1968

[Liu,85] Liu, C. L.
Elements of Discrete Mathematics
McGraw-Hill
New York, NY 1985

[L&H,91] Liu, H. / Huang, S.-H. S.
Dilation-6 Embeddings of 3-Dimensional Grids into Optimal Hypercubes
Proceedings of the 1991 International Conference on Parallel Processing **III**, 250 – 254

[Llo,83] Lloyd, E. K. (ed.)
Surveys in Combinatorics
Invited Papers for the Ninth British Combinatorial Conference 1983
London Mathematical Society Lecture Note Series **82**
Cambridge University Press
Cambridge 1983

[Loo,89] Loogen, R.
Parallele Implementierung funktionaler Programmiersprachen
Informatik-Fachbericht **232**
Springer
Heidelberg 1989

[Lub,85] Luby, M.
A Simple Parallel Algorithm for the Maximal Independent Set Problem
Proceedings of the 17^{th} Annual ACM Symposium on the Theory of Computing (1985), 1 – 10

[Lün,79] Lüneburg, H.
Galoisfelder, Kreisteilungskörper und Schieberegisterfolgen
Bibliographisches Institut
Mannheim 1979

[Mcg,92] McGeoch, C.
Analyzing Algorithms by Simulation: Variance Reduction Techniques and Simulation Speedups
ACM Computing Surveys 24 (1992), 195 – 212

[Mea,90] Mead, R.
The Non-Orthogonal Design of Experiments
Journal of the Royal Statistical Society A 153 (1990), Part 2, 151 – 201

[M&R,82] Memmi, G. / Raillard, Y.
Some New Results about the (d,k)-Graph Problem
IEEE Transactions on Computers C-31 (1982), 784 – 791

[M&K,91] Mendelsohn, B. / Koren, I.
Using Simulated Annealing for Mapping Algorithms onto Data Driven Arrays
Proceedings of the 1991 International Conference on Parallel Processing I, 123 – 127

[Men,82] Mendelsohn, E. (ed.)
Algebraic and Geometric Combinatorics
Annals of Dicrete Mathematics 15
North Holland
Amsterdam 1982

[Mic,80] Mickunas, M. D.
Using Projective Geometry to Design Bus Connection Networks
Proceedings of the Workshop on Interconnection Networks for Parallel and Distributed Processing, West Lafayette 1980, 47 – 55

[M&S,76] Mies, P. / Schütt, D.
Feldrechner
Bibliographisches Institut
Mannheim 1976

[Mil,78] Miller, G. L.
On the $n^{\log n}$ Isomorphism Technique
Proceedings of the 10th Annual ACM Symposium on the Theory of Computing (1978), 51 – 58

[M&C,86] Mitra, D. / Cieslak, R. A.
Randomized Parallel Communications
Proceedings of the 1986 International Conference on Parallel Processing, 224 – 230

[M&C,87] Mitra, D. / Cieslak, R. A.
Randomized Parallel Communications on an Extension of the Omega Network
Journal of the ACM 34 (1987), 802 – 824

[Mok,85] Mok, A. K.
A Graph-Based Computation Model for Real-Time Systems
Proceedings of the 1985 International Conference on Parallel Processing, 619 – 623

[M&S,88] Monien, B. / Sudborough, H.
Comparing Interconnection Networks
Proceedings of the 13th Symposium on the Mathematical Foundations of Computer Science, Carlovy Vary (CSFR) (1988), 138 – 153

[M&E,92] Morris, D. / Evans, D. G.
Modelling Distributed and Parallel Computer Systems
Parallel Computing **18** (1992), 793 – 806

[M&&,84] Mudge, T. N. et al.
Analysis of Multiple Bus Interconnection Networks
Proceedings of the 1984 International Conference on Parallel Processing, 228 – 232

[OHW,87] Oberschelp, W. / Hinsen, G. / Wagner, F.
Kombinatorik. Eine Einführung
Schriften zur Informatik und Angewandten Mathematik **129**
RWTH Aachen
Aachen 1987

[O&V,86] Oberschelp, W. / Vossen, G.
Rechneraufbau und Rechnerstrukturen
R. Oldenbourg
München 1986

[O&W,76] Oberschelp, W. / Wille, D.
Mathematischer Einführungskurs für Informatiker
Leitfäden der Angewandten Mathematik und Mechanik 35
B. G. Teubner
Stuttgart 1976

[O&B,86] O'Dell, R. / Browne, J. C.
Parallel Structuring of Control and Resources Management Systems for Parallel Programs
Proceedings of the 1986 International Conference on Parallel Processing, 153 – 156

[Oed,93] Oed, W.
Das FORTRAN-Programmiermodell des CRAY-Research massiv-parallelen Prozessorsystems
3. GI-ITG Workshop über Parallele Systeme und Algorithmen (PASA), Bonn, 1. April 1993, Vortrag

[P&Y,88] Papadimitriou, C. H. / Yannakakis, M.
Towards an Architecture-Independent Analysis of Parallel Algorithms
Proceedings of the 20th Annual ACM Symposium on the Theory of Computing (1988), 510 – 513

[Par,87] Parberry, I.
Some Practical Simulations of Impractical Computers
Parallel Computing 4 (1987), 93 – 101

[P&&,84] Patt, Y. N. et al.
A Comparison of Several Evolving (University) Supercomputer Architectures
Proceedings of the 4th Jerusalem Conference on Information Technology (1984), 15 – 26

[P&S,85] Peterson, J. L. / Silberschatz, A.
Operating System Concepts
Addison-Wesley
Reading, MA 1985

[Phe,79] Phelps, K. T.
Parallelism in Designs and Strongly Regular Graphs
Proceedings of the 9th Manitoba Conference on Numerical Mathematics and Computing (1979), 365 – 384

[Pip,84] Pippenger, N.
Parallel Communication with Limited Buffers
Proceedings of the 25th IEEE International Symposium on the Foundations of Computer Science (1984), 127 – 136

[Pól,18] Pólya, G.
Über die „doppelt-periodischen" Lösungen des n-Damen-Problems
in [Ahr,18], 364 – 374

[Pop,88] Poplawski, D. A.
Mapping Rings and Grids onto the FPS T-Series Hypercube
Parallel Computing 7 (1988), 1 – 10

[P&R,87] Pountain, D. / Rudolph, R.
OCCAM – Das Handbuch
Heinz Heise
Hannover 1987

[P&V,81] Preparata, F. P. / Vuillemin, J.
The Cube-Connected Cycles: A Versatile Network for Parallel Computation
Communications of the ACM 24 (1981), 300 – 309

[Pre;84] Preparata, F. P.
VLSI Algorithms and Architectures
Proceedings of the 11th Symposium on the Mathematical Foundations of Computer Science (1984), 149 – 161

[Prz,92] Przybylski, F.
Scheduling-Verfahren für Rechner mit verteiltem Speicher: Eine vergleichende Übersicht
Berichte des Forschungszentrums Jülich **Jül-2598**
Forschungszentrum Jülich
Jülich 1992

[Qui,88] Quinn, M. J.
Algorithmenbau und Parallelcomputer
McGraw-Hill
Hamburg 1988

[RLH,88] Raabe, U. / Lobjinski, M. / Horn, M.
Verbindungsstrukturen für Multiprozessoren
Informatik-Spektrum 11 (1988), 195 – 206

[Rab,76] Rabin, M. O.
Probabilistic Algorithms
in [Tra,76], 21 – 39

[Rab,89] Rabin, M. O.
Efficient Dispersal of Information for Security, Load Balancing, and Fault Tolerance
Journal of the ACM 36 (1989), 335 – 348

[Rag,71] Raghavarao, D.
Constructions and Combinatorial Problems in Design of Experiments
John Wiley & Sons
New York, NY 1971

[RLS,88] Raghupathy, S. / Leuze, M. R. / Schach, S. R.
Message Routing Schemes in a Hypercube Machine
Proceedings of the 3rd International Conference on Hypercube Concurrent Computers (1988), 640 – 646

[R&R,87] Rajasekaran, S. / Reif, J. H.
Randomized Parallel Computation
Proceedings of the 1987 International Conference on the Fundamentals of Computation Theory (FCT87), 364 – 376

[R&T,87] Rajasekaran, S. / Tsantilas, T.
An Optimal Randomized Routing Algorithm for the Mesh and a Class of Efficient Mesh-like Routing Networks
Proceedings of the 7th Conference on the Foundations of Software Technology and Theoretical Computer Science (1987), 226 – 241

[Rat,90] Rattray, C. (ed.)
Specification and Verification of Concurrent Systems
Springer
London 1990

[R&S,89] Reif, J. H. / Sen, S.
Randomization in Parallel Algorithms and its Impact on Computational Geometry
Proceedings of the International Symposium on Optimal Algorithms
Varna 1989
Springer Lecture Notes in Computer Science **401** (1989), 1 – 8

[Rei,82] Reisig, W.
Petrinetze
Springer
Berlin 1982

[Rob,84] Roberts, F. S.
Applied Combinatorics
Prentice Hall
Englewood Cliffs, NJ 1984

[Röd,91] Rödder, S.
Aufbau und Eigenschaften von Hypercube-Rechnern und Untersuchung von Datentransferalgorithmen
Berichte des Forschungszentrums Jülich **Jül-2428**
Forschungszentrum Jülich
Jülich 1991

[Rod,69] Rodriguez, J. E.
A Graph Model for Parallel Computations
MIT Technical Report **TR-64**
Cambridge, MA 1969

[Rys,63] Ryser, H. J.
Combinatorial Mathematics
John Wiley & Sons
New York, NY 1963

[S&S,88] Saad, Y. / Schultz, M. H.
Topological Properties of Hypercubes
IEEE Transactions on Computers **C-37** (1988), 867 – 872

[S&S,89] Saad, Y. / Schultz, M. H.
Data Communication in Parallel Architectures
Parallel Computing **11** (1989), 131 – 150

[Sch,91] Scherson, I. D.
Orthogonal Graphs for the Construction of a Class of Interconnection Networks
IEEE Transactions on Parallel and Distributed Systems **P-2** (1991), 3 – 19

[Sha,78] Shapiro, H. D.
Generalized Latin Squares on the Torus
Discrete Mathematics **24** (1978), 63 – 77

[S&T,85] Shen, C.-C. / Tsai, W.-H.
A Graph Matching Approach to Optimal Task Assignment in Distributed Computing Systems Using a Minimax Criterion
IEEE Transactions on Computers **C-34** (1985), 197 – 203

[Sie,80] Siegel, H. J.
The Theory Underlying the Partitioning of Permutation Networks
IEEE Transactions on Computers **C-29** (1980), 791 – 800

[S&R,86] Skvarcius, R. / Robinson, W. B.
Discrete Mathematics with Computer Science Applications
Benjamin/Cummings
Menlo Park, CA 1986

[Sny,82] Snyder, L.
Introduction to the Configurable, Highly Parallel Computer
Computer **15** (1982), 47 – 56

[S&S,77] Solovay, R. / Strassen, V.
A Fast Monte-Carlo Test for Primality
SIAM Journal on Computing **6** (1977), 84 – 85

[Sri,73] Srivastava, J. N. (ed.)
A Survey of Combinatorial Theory
North Holland
Amsterdam 1973

[S&H,84] Steinberg, D. M. / Hunter, W. G.
Experimental Design: Review and Comment
Technometrics **26** (1984), 71 - 130

[S&S,86] Sterling, L. / Shapiro, E.
The Art of Prolog: Advanced Programming Techniques
Addison Wesley
Reading, MA 1986

[S&W,90] Stout, Q. F. / Wagar, B.
Intensive Hypercube Communication. Prearranged Communication in Link-Bound Machines
Journal of Parallel and Distributed Computing **10** (1990), 167 – 181

[S&S,87] Street, A. P. / Street, D. J.
Combinatorics of Experimental Design
Clarendon Press
Oxford 1987

[Szy,91] Szymanski, T.
$O(\log N/\log\log N)$ Randomized Routing in Degree $\log N$ 'Hypermeshes'
Proceedings of the 1991 International Conference on Parallel Processing
I, 443 – 450

[S&F,91] Szymanski, T. / Fang, C.
Randomized Routing of Virtual Connections in Extended Banyans
Proceedings of the 1991 International Conference on Parallel Processing
I, 451 – 455

[T&D,90] Tan, S. T. / Du, D. H. C.
On Subcube Allocation and Relinquishment Schemes for Hypercube Connected Multiprocessors
Proceedings of the 1990 International Conference on Parallel Processing
I, 571 – 572

[Tar,00] Tarry, G.
Le probleme des 36 officiers
Compte rendu. Association française pour l'Avancement des Sciences 1 (1900), 122 – 123

[Tar,01] Tarry, G.
Le probleme des 36 officiers
Compte rendu. Association française pour l'Avancement des Sciences 2 (1901), 170 – 203

[Tom,85] Tomescu, I.
Problems in Combinatorics and Graph Theory
John Wiley & Sons
New York, NY 1985

[Tra,76] Traub, J. F. (ed.)
Algorithms and Complexity. New Directions and Recent Results
Academic Press
New York, NY 1976

[TBK,91] Tsai, W. K. / Bagherzadeh, N. / Kim, Y. C.
Hypermesh: A Combined Quad Tree and Mesh Network for Parallel Processing
Proceedings of the 1991 International Conference on Parallel Processing
I, 696 – 697

[Upf,84] Upfal, E.
Efficient Schemes for Parallel Communication
Journal of the ACM 31 (1984), 507 – 517

[U&W,84] Upfal, E. / Wigderson, A.
How to Share Memory in a Distributed System
Proceedings of the 25th IEEE International Symposium on the Foundations of Computer Science (1984), 171 – 180

[Val,80] Valiant, L. G.
Experiments with a Parallel Communication Scheme
Proceedings of the 18th Allerton Conference on Communication, Control, and Computing, Monticello (1980), 802 – 811

[Val,82a] Valiant, L. G.
A Scheme for Fast Parallel Communication
SIAM Journal of Computing 11 (1982), 350 – 361

[Val,82b] Valiant, L. G.
Parallel Computation
Foundations of Computer Science 4, Part I
Mathematical Centre Tracts. 158
Amsterdam 1982

[Val,83] Valiant, L. G.
Optimality of a Two-Phase Strategy for Routing in Interconnection Networks
IEEE Transactions on Computers C-32 (1983), 861 – 863

[Val,88] Valiant, L. G.
Optimally Universal Parallel Computers
Philosophical Transactions of the Royal Society of London A 326 (1988), 373 – 376

[Val,90a] Valiant, L. G.
General Purpose Parallel Architectures
in [Lee,90a], 945 – 971

[Val,90b] Valiant, L. G.
A Bridging Model for Parallel Computation
Communications of the ACM 33 (1990), 103 – 111

[V&B,81] Valiant, L. G. / Brebner, G. J.
Universal Schemes for Parallel Communication
Proceedings of the 13th Annual ACM Symposium on Theory of Computing (1981), 263 – 277

[Wag,89] Wagner, A.
Embedding Arbitrary Binary Trees in a Hypercube
Journal of Parallel and Distributed Computing 7 (1989), 503 – 520

[Wal,88] Wallis, W. D.
Combinatorial Designs
Marcel Dekker
New York, NY 1988

[W&Y,91] Wang, H. / Yang, Q.
Prime Cube Graph Approach for Processor Allocation in Hypercube Multiprocessors
Proceedings ot the 1991 International Conference on Parallel Processing I, 25 – 32

[Web,93] Weberpals, H.
Struktur und Leistung paralleler Algorithmen für den Parallelrechner KSR I
3. GI-ITG Workshop über Parallele Systeme und Algorithmen (PASA), Bonn, 1. April 1993, Vortrag

[Wij,89] Wijshoff, H. A. G.
Data Organization in Parallel Computers
Kluwer
Boston, MA 1989

[Wil,72] Wilkov, R. S.
Analysis and Design of Reliable Computer Networks
IEEE Transactions on Communications **20** (1972), 660 – 678

[W&B,79] Wilson, R. J. / Beineke, L. W. (eds.)
Applications of Graph Theory
Academic Press
London 1979

[Wit,81] Wittie, L. D.
Communication Structures for Large Networks of Microcomputers
IEEE Transactions on Computers **C-30** (1981), 264 – 273

[Wu_,85] Wu, A. Y.
Embedding of Tree Networks into Hypercubes
Journal of Parallel and Distributed Computing **2** (1985), 238 – 249

[W&S,90] Wu, M.-Y. / Shu, W.
A Dynamic Program Partitioning Strategy on Distributed Memory Systems
Proceedings of the 1990 International Conference on Parallel Processing I, 551 – 552

[Y&&,88] Yang, C. S. et al.
Graph Theoretic Reliability Analysis for the Boolean n-Cube Networks
IEEE Transactions on Circuits and Systems 35 (1988), 1175 – 1179

[YBN,91] Yang, J. / Bic, L. / Nicolau, A.
A Mapping Strategy for MIMD Computers
Proceedings of the 1991 International Conference on Parallel Processing I, 102 – 109

[Y&S,82] Yau, S. S. / Shatz, S. M.
On Communication in the Design of Software Components of Distributed Computer Systems
Proceedings of the 3rd International Conference on Distributed Computing Systems (1982), 280 – 287

[Yov,80] Yovits, M. C. (ed.)
Advances in Computers Vol. 19
Academic Press
New York, NY 1980

Anhang

Anhang A. Einführung in die Kombinatorik der optimalen Versuchspläne

Das hier zusammengetragene Material stützt sich auf die Arbeiten [BJL,85], [C&O,89], [Col,85], [Dem,70], [Hed,77], [H&F,75], [H&P,85], [OHW,87], [S&S,87] und [Wal,88]. Um ein anwendungsorientiertes Suchen zu ermöglichen, werden bei den jeweiligen Abschnitten die Anwendungsfelder in der Parallelverarbeitung kurz angedeutet. Für die zugehörigen Beweise wird auf die hier angegebene Literatur, für weiterführende Dinge wie Existenztheorien und Zusammenhänge mit anderen Bereichen der Mathematik wird auf folgende Arbeiten verwiesen: [Bat,86], [Ber,73], [Bol,84], [Bol,89], [D&G,79], [G&&,69], [H&L,75], [Liu,68], [Liu,85], [Men,82], [O&W,76], [Rag,71], [Rob,84], [Rys,63], [Tom,85]. Die hier aufgeführten Werke stellen mehrheitlich Monographien und Tagungsbände dar; die Zahl verwandter (Einzel-)Arbeiten ist unüberschaubar.

A.1 Blockpläne

Zunächst erfolgt die Definition des zentralen Begriffs, des kombinatorischen Versuchsplans, auf welchem die gesamte Theorie aufbaut, und die Erläuterung der damit zusammenhängenden wichtigsten Spezifikationsparameter. Der Abschnitt wird ergänzt durch wichtige Erweiterungen und Spezialfälle von Versuchsplänen, die beispielsweise im Bereich des Mapping paralleler Algorithmen auf parallele Architekturen ihre Anwendung finden.

Def. 1: Sei V eine endliche Menge von v Elementen; ein **Mengensystem** $\mathcal{B}$ auf V ist eine Kollektion von Teilmengen von V. Das Mengensystem heißt **einfach**, falls $\mathcal{B}$ keine dieser (Teil-)Mengen mehr als einmal enthält. Elemente von $\mathcal{B}$ (Mengen) werden als **Blöcke** bezeichnet; die Anzahl der Blöcke wird mit $b = |\mathcal{B}|$ bezeichnet.

Def. 2: Sei V endliche Menge, $\mathcal{B}$ Mengensystem auf V, dann heißt das Paar $(V,\mathcal{B})$ ein **kombinatorischer Versuchsplan.**

Def. 3: Die Zahl $v = |V|$ heißt **Kardinalität** des Versuchsplans.

Def. 4: K bezeichnet die **Menge der Blockgrößen** $\{|B| : B \in \mathcal{B}\}$. Falls K nur eine einzige Blockgröße, k, enthält, so wird das Mengensystem als **k-uniform** bezeichnet.

Def. 5: Jedes Element $x \in V$ tritt in einer bestimmten Anzahl von Blöcken auf. $\mathcal{B}_x \subseteq \mathcal{B}$ bezeichne diejenige Teilmenge von $\mathcal{B}$, die alle Blöcke enthält, die x enthalten. Der **Wiederholungsfaktor** r_x ist definiert durch $r_x := |\mathcal{B}_x| > 0$ und die **Menge R der Wiederholungsfaktoren** ist definiert durch $R := \{r_x \mid x \in V\}$.

Def. 6: Besitzen in einem k-uniformen Versuchsplan $(V,\mathcal{B})$ alle Elemente von V den gleichen Wiederholungsfaktor r, so heißt der Versuchsplan **(r-)regulär.**

Def. 7: Falls ein Block des Versuchsplans alle Elemente von V enthält, heißt dieser Block **vollständig**. Falls dies in einem regulären Versuchsplan auftritt, so handelt es sich um einen **vollständigen Versuchsplan**. Ein Versuchsplan heißt **unvollständig**, wenn wenigstens ein Block nicht vollständig ist.

In vollständigen Versuchplänen gilt $k = v$ und $r = b$. An derartigen Versuchsplänen besteht wenig Interesse, wenn nicht eine weitergehende Struktur, wie z. B. ein Lateinisches Quadrat, betrachtet wird.

Jede Teilmenge $S \subseteq V$ ist eine (nicht notwendigerweise echte) Teilmenge einer Anzahl von Blöcken aus $\mathscr{B}$.

Def. 8: Die Anzahl der Blöcke in $\mathscr{B}$, die S enthalten, wird mit $\lambda(S)$ bezeichnet und heißt der **Index von S in $\mathscr{B}$**.

Def. 9: Die **t-Index-Menge** eines Mengensystems $(V,\mathscr{B})$ ist definiert als die Menge $\Lambda_t := \{\lambda(S): |S| = t, S \subseteq V\}$; $\Lambda := \Lambda_2$.

Def. 10: Ein Mengensystem heißt **t-balanciert**, falls die t-Index-Menge nur einen einzigen Wert $\lambda_t > 0$ enthält.

Ein 1-balanciertes Mengensystem zeichnet sich lediglich dadurch aus, daß es einen einzigen Wiederholungsfaktor r mit $r = \lambda_1$ gibt. Jedes Mengensystem ist 0-balanciert mit $\lambda_0 = b$. Ein t-balanciertes k-uniformes Mengensystem ist ebenso $(t-1)$-balanciert.

Def. 11: Ein **balancierter unvollständiger Blockplan (balanced incomplete block design, BIBD)** ist ein Paar $(V,\mathscr{B})$, wobei $\mathscr{B}$ ein k-uniformes Mengensystem auf V darstellt, das 2-balanciert mit Index λ ist. $(V,\mathscr{B})$ wird dann als (v,k,λ)-Plan bezeichnet.

Die übrigen Parameter, r und b, können aus den spezifizierten Parametern durch Beachtung der Identitäten $vr = bk$ und $r(k-1) = \lambda(v-1)$ berechnet werden. Diese Identitäten erhält man durch Abzählen der Anzahl der Paare, in denen ein gegebenes Element auftritt, und der Gesamtzahl der Elemente des Plans in je zwei Richtungen. Blockpläne sind Mengensysteme, in denen ungeordnete Paare uniform auftreten; die natürliche Ausweitung sind t-balancierte Mengensysteme.

Def. 12: Ein **t-Plan** $(V,\mathscr{B})$ oder genauer ein **t-(v,k,λ)-Plan** enthält ein k-uniformes Mengensystem $\mathscr{B}$ auf V mit $|V| = v$, welches t-balanciert ist mit der t-Index-Menge $\{\lambda\}$.

Triviale t-Pläne ergeben sich durch Setzen von $v = k$ oder $k = t$.

Eine notwendige Bedingung für die Existenz von Blockplänen ergibt sich durch Betrachtung der Zahl der Blöcke.

Lemma 1: Die Zahl der Blöcke b in einem Blockplan beträgt:

$$b = \lambda \binom{v}{2} / \binom{k}{2}$$

Satz 1: **(Fishers Ungleichung)**
In jedem Blockplan (mit $v > k$) gilt:

$$b \geq v$$

Kor. 1: Damit ergibt sich:

$$\lambda (v-1) \geq k (k-1)$$

Def. 13: Sei $(V,\mathscr{B})$ Versuchsplan, X eine beliebige, feste Teilmenge von V; sei $\mathscr{B}_x$ die Teilmenge der Blöcke von $\mathscr{B}$, die X enthalten. Sei $\mathscr{B}_x \backslash X$ die Menge von Blöcken, die man erhält, wenn aus jedem Block in $\mathscr{B}_x$ alle Elemente von X entfernt werden. Dann heißt der Plan $(V\backslash X, \mathscr{B}_x \backslash X)$ der **nach X abgeleitete Plan.**

Falls $(V,\mathscr{B})$ ein t-Plan ist, so ist der abgeleitete Plan $(V\backslash X, \mathscr{B}_x \backslash X)$ ein $(t-|X|)$-Plan.

Def. 14: Ein **Hyperplan** $(V, \{\mathscr{B}_i\}, I)$ (I Indexmenge) ist eine Familie von $i = |I|$ kombinatorischen Versuchsplänen $(V, \mathscr{B}_i)$, die über der gleichen Grundmenge V von Elementen operieren.

Def. 15: Eine **Versuchsplansequenz** $(V^x, \mathscr{B}^x, \Phi, l)$ umfaßt eine Folge von Versuchsplänen $(V,\mathscr{B})_1 \ldots (V,\mathscr{B})_l$, wobei l die Länge der Folge festlegt, und eine Folge von Relationen $\Phi = (\varphi_{12}, \ldots, \varphi_{l-1,l})$, die die Beziehungen von Blöcken in verschiedenen Versuchsplänen festlegt.

A.2 Symmetrische Pläne

Eine Ursache für das besondere Interesse an symmetrischen Plänen besteht darin, daß sie die Zahl der Blöcke in einem Plan minimieren. Überträgt man diesen Zusammenhang auf Verbindungen in einem Netzwerk so wird die Optimierungseigenschaft deutlich. Einen zweiten Grund liefert das nachfolgende Lemma.

Def. 16: Ein Plan mit $b = v$ heißt **symmetrischer Plan.**

Bem. 1: In einem symmetrischen Plan gilt $\lambda(v-1) = k(k-1)$ und daher haben die Parameter des symmetrischen Plans die Form $(((k^2-k)/\lambda)+1, k, \lambda)$. Man beachte, daß $b = v$ impliziert, daß $k = r$.

Def. 17: Die Zahl $\mu = k - \lambda$ heißt **Ordnung** des symmetrischen Plans.

Def. 18: Symmetrische Pläne mit $\lambda = 2$ werden als **Bi-Ebenen (biplanes)** bezeichnet.

Def. 19: Aus einem symmetrischen Plan kann man einen **Residuen-Plan** generieren, indem man einen Block auswählt, diesen Block entfernt und ebenso alle seine Elemente aus den übrigen Blöcken entfernt.

Lemma 2: Zwei Blöcke in einem symmetrischen (v,k,λ)-Plan schneiden sich immer in genau λ Elementen, d. h. sie haben genau λ identische Elemente.

Der Fall $\lambda = 1$ hat eine besondere Aufmerksamkeit erlangt.

Def. 20: Ein symmetrischer Plan mit den Parametern $(k^2-k+1,k,1)$ wird als **(endliche) projektive Ebene** bezeichnet. Die Parameter können alternativ geschrieben werden als $(\mu^2+\mu+1, \mu+1, 1)$ und die Ebene heißt dann **von der Ordnung μ.**

Def. 21: Residuen projektiver Ebenen sind $(\mu^2, \mu, 1)$-Pläne und werden üblicherweise als **affine Ebenen** bezeichnet.

Projektive Ebenen treten in vielen Bereichen, insbesondere der darstellenden Geometrie, auf – daher auch der Name; nur selten werden sie als Teil der kombinatorischen Theorie der optimalen Versuchspläne betrachtet bzw. identifiziert. Dieser Abschnitt weist die Zugehörigkeit zu der hier betrachteten Theorie nach, wobei die Motivation für die Einbeziehung der projektiven Ebenen in diese Arbeit im wesentlichen durch ihre Verwendbarkeit als Konstruktionsprinzip neuer Netzwerktopologien gegeben ist (vgl. Kapitel 2). Projektive Ebenen kann man aus einer allgemeinen Klasse von Strukturen gewinnen, die symmetrische Pläne hervorrufen.

Def. 22: Sei S ein $(m+1)$-dimensionaler Vektorraum über $GF(q)$, dem endlichen Körper mit q Elementen, wobei q eine Primzahl oder eine Primzahlpotenz ist. Die Menge aller Teilräume von S heißt **projektive Geometrie der Dimension m über $GF(q)$** und wird mit $PG(m,q)$ bezeichnet.

Def. 23: Die eindimensionalen Teilräume von S heißen **Punkte**, die $(m-1)$-dimensionalen Teilräume heißen **Hyperebenen.**

Lemma 3: Für jede Hyperebene H bezeichne B_H die Menge der Punkte, die in H enthalten sind. Benutzt man nun die eindimensionalen Teilräume von S als Punkte, so definiert die Menge der Blöcke $\{B_H \mid H \subset S\}$ einen symmetrischen Plan mit den Parametern:

$$v=\frac{q^{m+1}-1}{q-1}, \quad k=\frac{q^m-1}{q-1}, \quad \lambda=\frac{q^{m-1}-1}{q-1}.$$

Für $m=2$ ergeben sich projektive Ebenen.

Kor. 2: Es folgt, daß projektive Ebenen für alle Werte $\mu=q$ existieren, die Primzahlpotenzen sind.

A.3 Verallgemeinerungen von Blockplänen

Da die Verbindungsnetzwerke in der Parallelverarbeitung, aus technischen Gründen oder kostenbedingt, im allgemeinen nicht die idealen Formen besitzen, die den Grundstrukturen optimaler Versuchspläne entsprechen, können sie in der Mehrheit der Fälle nur mittels Versuchsplänen kombinatorisch beschrieben werden, die gegenüber balancierten unvollständigen Blockplänen hinsichtlich Uniformität oder Balance nur reduzierten Anforderungen genügen müssen.

Def. 24: Sei $(V,\mathcal{B})$ ein Mengensystem, welches k-uniform auf v Elementen ist und jede t-Teilmenge erscheine in *höchstens* [*mindestens*] λ Blöcken von $\mathcal{B}$.

Dann wird $(V,\mathscr{B})$ als eine t-(v,k,λ)-**Verdichtung** [t-(v,k,λ)-**Überdeckung**] bezeichnet.

Lemma 4: Eine t-(v,k,λ)-Verdichtung kann maximal

$$b(t,v,k,\lambda) = \lambda\binom{v}{t}/\binom{k}{t}$$

Blöcke haben, während eine t-(v,k,λ)-Überdeckung mindestens diese Zahl von Blöcken besitzen muß. Die Gleichheit gilt genau dann, wenn die Verdichtung (Überdeckung) ein t-(v,k,λ)-Plan ist.

Def. 25: Ein **partiell balancierter unvollständiger Blockplan (PBIBD)** ist eine Verallgemeinerung eines BIBD. In diesem Fall ist es nicht notwendig, daß jedes Paar von Elementen gleichhäufig auftritt. Falls $\Lambda = \{\lambda_1,\ldots,\lambda_m\}$ eine Menge positiver ganzer Zahlen ist, dann ist ein PBIBD mit den Parametern (v,k,Λ) eine Anordnung von v Elementen in k-Teilmengen derart, daß jedes Paar insgesamt in λ_i Blöcken auftritt für irgendein λ_i aus Λ.

Verdichtungen und Überdeckungen entbinden von der Forderung, daß der Index konstant sein muß. Nun wird stattdessen die Forderung aufgegeben, daß die Blockgröße konstant sein muß:

Def. 26: Ein **paarweise balancierter Plan** (pairwise balanced design, PBD) mit den Parametern (v,K,λ) ist ein Mengensystem auf v Elementen, mit Blockgrößen aus K, welches 2-balanciert mit Index λ ist.

Ein PBD hat im allgemeinen keinen eindeutigen Wiederholungsfaktor. Blockpläne sind PBDs mit einer einzigen Blockgröße. Es existiert eine umfassende Theorie über paarweise balancierte Pläne. Fishers Ungleichung gilt auch für PBDs; daher sind symmetrische Pläne wiederum PBDs mit einer Minimalzahl von Blöcken.

Schließlich kann man sowohl die Restriktion betreffend die Blockgrößen aufgeben, als auch nur eine obere Grenze für den Index vorgeben.

Def. 27: Ein **partieller PBD** ist ein Versuchsplan $(V,\mathscr{B})$ mit den Parametern (v,K,Λ), es existiert also weder für die Blockgröße noch für die Balance ein einheitlicher Wert. Ein partieller PBD wird auch als **nicht-uniforme Verdichtung** bezeichnet.

A.4 Zerlegbarkeit

Die Zerlegbarkeit kombinatorischer Versuchspläne findet ihre direkte Korrespondenz in der Partitionierung statischer Netzwerke.

Def. 28: Eine **partiell-parallele Klasse (PPC)** eines Plans $(V,\mathscr{B})$ ist eine Sammlung disjunkter Blöcke von $\mathscr{B}$.

Def. 29: Für einen Plan $(V,\mathscr{B})$ ist eine **parallele Klasse (Zerlegungsklasse, PC)** von Blöcken $\wp \subseteq \mathscr{B}$ eine Menge von Blöcken derart, daß die Schnittmenge von je zwei Blöcken leer ist und die Vereinigung aller Blöcke von $\wp$ V ergibt.

Eine parallele Klasse ist demnach eine PPC, in der jedes Element von V exakt einmal auftritt; mit anderen Worten, eine PC enthält v/k Blöcke.

Def. 30: Eine **fast-parallele Klasse** ist eine abgeschwächte Form einer parallelen Klasse, bei der die Vereinigung alle bis auf ein Element von V enthalten muß.

Def. 31: Wenn $\mathscr{B}$ in parallele Klassen zerlegt werden kann, wenn also die b Blöcke in disjunkte parallele Klassen partitioniert werden können, wird diese Zerlegung als **Resolution** bezeichnet und $(V,\mathscr{B})$ heißt dann ein **resolvierbarer Plan.**

Def. 32: Ganz allgemein kann ein t-(v,k,λ)-Plan **zerlegbar** sein in (t')-(v,k,λ')-Pläne; Resolution ist dann der Fall $t' = 1$ und $\lambda' = 1$.

Def. 33: Die größte(n) PPC(s) in einem gegebenen Plan werden als **Maximum** bezeichnet.

Def. 34: Eine PPC ist **maximal**, wenn kein Block in dem verbleibenden Plan existiert, der wechselseitig disjunkt zu allen Blöcken innerhalb der PPC ist, d. h. die PPC kann nicht erweitert werden.

A.5 Isomorphie

Die Eigenschaft zweier Versuchpläne, zueinander isomorph zu sein, wird in der Parallelverarbeitung an der Stelle wesentlich, wo der eine Versuchsplan einen parallelen Algorithmus, der andere eine parallele Architektur repräsentiert.

Die Isomorphie weist in diesem Fall auf die Möglichkeit eines geeigneten Mapping hin.

Def. 35: Zwei Pläne $(V_1, \mathscr{B}_1)$ und $(V_2, \mathscr{B}_2)$ heißen **isomorph** genau dann, wenn eine Bijektion $\varphi: V_1 \rightarrow V_2$ existiert, so daß gilt:
$(b_1, \ldots, b_k) \in \mathscr{B}_1 \Leftrightarrow (\varphi(b_1), \ldots, \varphi(b_k)) \in \mathscr{B}_2$

Def. 36: Gilt darüber hinaus $V_A = V_B =: V$ (und existiert damit eine Permuation Π auf V, die ein Π auf $\mathscr{B}_A$ induziert, so daß $\Pi(\mathscr{B}_A) = \mathscr{B}_B$), so heißen die Versuchspläne $(V,\mathscr{B})_A$ und $(V,\mathscr{B})_B$ **äquivalent**, $(V,\mathscr{B})_A \approx (V,\mathscr{B})_B$.

Def. 37: Ein Versuchsplan $(V,\mathscr{B})_2$ **emuliert** einen Versuchsplan $(V,\mathscr{B})_1$, $(V,\mathscr{B})_2 > (V,\mathscr{B})_1$, $(V,\mathscr{B})_1 < (V,\mathscr{B})_2$, wenn gilt: $\exists (V,\mathscr{B})_x \approx (V,\mathscr{B})_2$ mit

1. $V_1 \subseteq V_x$
2. $\forall B_i \in \mathscr{B}_1 \mid \exists B_j \in \mathscr{B}_x$ mit $B_i \subseteq B_j$

Die Emulation heißt **strikt**, $(V,\mathscr{B})_2 \gtrsim (V,\mathscr{B})_1$, $(V,\mathscr{B})_1 \lesssim (V,\mathscr{B})_2$, wenn folgende Bedingungen gelten:

1. $V_1 \subseteq V_x$
2. $\forall B_i \in \mathscr{B}_1 \mid \exists B_j \in \mathscr{B}_x$ mit $B_i = B_j$

Def. 38: Gegeben sei ein Versuchsplan $(V,\mathscr{B})$; der zu $(V,\mathscr{B})$ **duale** Versuchsplan entsteht, indem man Blöcke als duale Elemente und Elemente als duale Blöcke auffaßt. Hierbei bezeichnen die Elemente von Mengen aus $\mathscr{B}$ mit welchen dualen Blöcken, das jeweilige duale Element inzidiert.

Def. 39: Ein **Automorphismus** eines Plans ist eine Bijektion der Elemente des Plans auf sich, wobei Blöcke auf Blöcke abgebildet werden und Teilmengen, die keine Blöcke bilden, auf Teilmengen abgebildet werden, die keine Blöcke bilden.

Def. 40: Die Menge aller Automorphismen bildet eine Gruppe mit der üblichen Komposition von Abbildungen als Verknüpfung und wird als **Automorphismen-Gruppe** bezeichnet.

Def. 41: Ein **zyklischer Automorphismus** ist eine zyklische Permutation, die alle Elemente des Plans einschließt.

Def. 42: Ein Plan der Kardinalität v heißt **zyklisch**, Bezeichnung: $t\text{-}CB(v,k,\lambda)$, wenn seine Automorphismen-Gruppe einen v-Zyklus enthält.

Ein $t\text{-}CB(v,k,\lambda)$-Plan kann repräsentiert werden als $t\text{-}(v,k,\lambda)$-Plan mit den Elementen $\{0,...,v-1\}$, für die, wenn $\{a_1, ..., a_k\}$ einen Block bildet, $\{a_1+1, ..., a_k+1\}$ ebenso einen Block bildet, wobei die Addition modulo v ausgeführt wird. Ein zyklischer Plan ist immer isomorph zu einem Plan $(V,\mathscr{B})$, für den $V = Z_v = \{0,1,...,v-1\}$ und die Abbildung $f: i \mapsto i+1 \pmod v$ ein Automorphismus ist.

A.6 Differenzmengen

Die gebräuchlichste Darstellung eines zyklischen Plans geschieht in Form einer Differenzmenge. Differenzmengen können vielseitig, auch mit Anwendungen in der Informatik, als Generatorsysteme genutzt werden.

Def. 43: Eine **(zyklische) (v,k,λ)-Differenzmenge** D, $D = \{d_1,..., d_k\}$, ist eine Sammlung von k Resten modulo v, so daß für jeden Rest $x \not\equiv 0 \pmod v$ die Kongruenz $d_i - d_j \equiv x \pmod v$ genau λ Lösungspaare (d_i, d_j) besitzt mit $d_i, d_j \in D$.

Eine notwendige Bedingung ist: $k(k-1) = \lambda(v-1)$. Dies entspricht der Gleichheit im Korollar zu Fishers Ungleichung (Korollar 1). Jede (v,k,λ)-Differenzmenge erzeugt einen zyklischen symmetrischen BIBD, dessen Blöcke durch $B(i) = \{d_1+i,..., d_k+i\} \pmod v$, $i = 0,...,v-1$ bestimmt werden. Die Differenzmenge, die für einen gegebenen Plan schwierig zu ermitteln ist, wird daher oft auch als **Anfangs-** oder **Basisblock** des symmetrischen Plans bezeichnet.

Def. 44: Ein $(v,k,2)$-Plan heißt **zyklisch**, wenn der Plan durch m Anfangsblöcke modulo v erzeugt werden kann, möglicherweise mit dem Extra-Basis-Block $(0,m',2m',...,(k-1)m')$, wobei $b = mv + m'$, $m' < v$.

Die Definition kann offensichtlich leicht auf größere Werte von t und λ verallgemeinert werden.

Def. 45: Gegeben seien zwei Differenzmengen D_1 und D_2 mit identischen Parametern. Falls $D_2 = tD_1 + s(\bmod v)$ für beliebige ganze Zahlen t und s, so sind die Differenzmengen D_1 und D_2 **äquivalent**.

Def. 46: Falls $D_1 = tD_1 + s(\bmod v)$ für ein s, so heißt t ein **Multiplikator** von D_1.

Die Abbildungen $x \mapsto tx + i(\bmod v)$, $i = 0,...,v-1$, sind Isomorphismen der zugehörigen symmetrischen Blockpläne.

Def. 47: Ein Plan heißt **r-rotational**, $r > 1$, wenn er einen Automorphismus besitzt, der ein Element festhält und die übrigen Elemente in r Zyklen jeweils der Länge $(v-1)/r$ permutiert.

A.7 Lateinische Quadrate

Lateinische Quadrate sind, neben der Grundstruktur als solcher, sicherlich die bekannteste Ausprägung kombinatorischer Versuchspläne. Fisher stellte 1926 erstmalig den Zusammenhang zwischen den Lateinischen Quadraten in ihrer historisch gewachsenen Notation und den kombinatorischen Versuchsplänen, in der Form wie sie auch hier eingeführt wurden, dar. Algebraisch gesehen ist ein Lateinisches Quadrat die Multiplikationstabelle einer Quasigruppe.

In der Parallelverarbeitung ergeben sich Anwendungsmöglichkeiten Lateinischer Quadrate und ihrer Sonderformen beispielsweise im Skewing und im Routing.

Def. 48: Sei $(V,\mathscr{B})$ ein vollständiger Versuchsplan. Fasse die Blöcke als **gerichtet** auf, d.h. die Reihenfolge der Elemente ist signifikant. (Blöcke sind hier also keine Mengen.)
Falls sich die Blöcke in einer $(b \times v)$-Matrix zeilenweise so anordnen lassen, daß jedes Element von V in jeder Spalte höchstens einmal auftritt, heißt der Versuchsplan **Lateinisches Rechteck**.

Def. 49: Sei $(V,\mathscr{B})$ ein vollständiger, symmetrischer Versuchsplan. Falls sich die gerichteten Blöcke in einer quadratischen $(v \times v)$-Matrix zeilenweise so anordnen lassen, daß jedes Element von V (in jeder Zeile genau einmal und) in jeder Spalte genau einmal auftritt, heißt der Versuchsplan **Lateinisches Quadrat** (der Kardinalität v).

Def. 50: Ein **Griechisch-Lateinisches Quadrat** entsteht durch Überlagerung zweier Lateinischer Quadrate derart, daß in der Überlagerungsstruktur jedes Paar von Objekten genau einmal auftritt.

Def. 51: Verhalten sich die beiden Lateinischen Quadrate so zueinander, daß sich durch Überlagerung ein Griechisch-Lateinisches Quadrat ergibt, so werden die Quadrate als **orthogonal** bezeichnet.

Satz 2: Eine Menge von paarweise orthogonalen Lateinischen Quadraten der Kardinalität v hat höchstens Mächtigkeit $v-1$.

Für (vollständig genannte) Mengen von $v-1$ paarweise orthogonalen Lateinischen Quadraten der Kardinalität v besteht ein Zusammenhang mit den projektiven Ebenen:

Satz 3: Eine endliche projektive Ebene der Ordnung v existiert genau dann, wenn es eine vollständige Menge von $v-1$ paarweise orthogonalen Lateinischen Quadraten der Kardinalität v gibt.

Eine wichtige Untersuchung der Struktur Lateinischer Quadrate zielt auf die Charakterisierung partieller Lateinischer Quadrate, die ohne Hinzunahme von Zeilen, Spalten oder Elementen zu Lateinischen Quadraten vervollständigt werden können.

Def. 52: Ein **partielles Lateinisches Quadrat** der Kardinalität v ist ein $v \times v$-Feld; jeder Eintrag ist entweder leer oder enthält ein Element der Menge $\{0,...,v-1\}$. Jede Zeile (Spalte) enthält jedes Element höchstens einmal.

Wichtig für viele Anwendungen sind Verfeinerungen Lateinischer Quadrate, die die gleichmäßige Verteilung der Elemente nicht nur über Zeilen und Spalten sondern auch über die Diagonalen fordern.

Def. 53: Ein Lateinisches Quadrat heißt **diagonal**, falls jedes Element i $(1 \leq i \leq n)$ auf der Hauptdiagonalen nur exakt einmal auftritt. Ein Lateinisches Quadrat heißt **doppelt diagonal (dd)**, falls dies sowohl für die Haupt- als auch für die Nebendiagonale gilt.

Satz 4: Ein beliebiges doppelt diagonales Lateinisches Quadrat der Kardinalität p $(p \geq 1)$, kann mit einem geeigneten diagonalen Lateinischen Quadrat der Kardinalität q ($q = 1 \vee q \geq 4$ falls $p = 1$, $q \geq 1$ sonst) kombiniert werden, um ein doppelt diagonales Lateinisches Quadrat der Kardinalität pq zu erhalten.

Def. 54: Ein **Knut-Vik-Versuchsplan** der Kardinalität v ist ein $v \times v$-Feld von Elementen, so daß folgende Bedingungen gelten:

1. (im Hinblick auf Zeilen und Spalten) bildet der Versuchsplan ein Lateinisches Quadrat
2. Diagonalen und Gegendiagonalen enthalten für alle $j = 0,1,...v-1$. (ebenfalls) jedes Element exakt einmal; die j-te Diagonale [Gegendiagonale] ist dabei definiert durch die Elemente $(i, j+i \bmod v)$ $[(i, j-i-1 \bmod v)]$ für $i = 0,1,...,v-1$.

Satz 5: **(Hedayat, Federer 1975)**
Knut-Vik-Versuchspläne existieren für alle Kardinalitäten v mit $v \geq 1$ falls v weder durch 2 noch durch 3 teilbar ist.

A.8 Hypergraphen

In diesem Abschnitt soll deutlich gemacht werden, daß die Graphentheorie als Teil der kombinatorische Theorie der optimalen Versuchspläne aufgefaßt werden kann. Dieser Zusammenhang erlaubt die Integration der in vielen Teilbereichen der Parallelverarbeitung bereits vorhandenen graphentheoretischen Modelle in das in der vorliegenden Arbeit entwickelte kombinatorische Modell.

Def. 55: Ein **Graph** ist ein 2-uniformer Versuchsplan $(V,\mathscr{B})$.

Aufgrund dieses Zusammenhangs werden Versuchspläne auch als **Hypergraphen** angesprochen. Das Mengensystem $\mathscr{B}$ wird bei dieser Betrachtungsweise dann häufig als E (*edges*, Kantenmenge) notiert. r-Regularität des Hypergraphen (V,E) bedeutet dann, daß jede Ecke (*vertex*) $x \in V$ mit r Kanten inzidiert; k-Uniformität des Hypergraphen (V,E) bedeutet, daß jede (Hyper-)Kante $y \in E$ mit k Ecken inzidiert.

Def. 56: Sei $(V,\mathscr{B})$ Hypergraph (Versuchsplan) der Kardinalität v; die **Inzidenzmatrix** des Hypergraphen ist eine $(b \times v)$-Matrix, für deren Einträge a_{ij} gilt: $a_{ij} = 1$ falls $a_i \in B_j$, 0 sonst. (Dies setzt eine vorangegangene Numerierung der Blöcke $B_i \in \mathscr{B}$ und Elemente $a_i \in V$ voraus.)

Def. 57: Sei $(V,\mathscr{B})$ Hypergraph (Versuchsplan) der Kardinalität v; die **Adjazenzmatrix** des Hypergraphen ist eine $(v \times v)$-Matrix, für die gilt: Die Diagonaleinträge sind leer; falls für zwei Elemente $a,b \in V$ ein Block B im Mengensystem $\mathscr{B}$ existiert, der beide Elemente enthält, so wird in der Adjazenzmatrix an den Stellen (a,b) und (b,a) jeweils eine 1 eingetragen; ist kein solcher Block vorhanden, so wird an diesen Stellen eine 0 eingetragen.

Def. 58: Zwei Graphen G_1 und G_2 heißen zueinander **isomorph**, wenn zwischen ihren Ecken und ihren Kanten Bijektionen bestehen, derart, daß die Inzidenzbeziehungen erhalten bleiben. Formal: Für $G_1 = (V_1, E_1)$ und $G_2 = (V_2, E_2)$ mit $|V_1| = |V_2|$ existiert eine Bijektion $\varphi: V_1 \to V_2$, für die gilt: $(x,y) \in E_1 \Leftrightarrow (\varphi(x), \varphi(y)) \in E_2 \quad \forall x,y \in V_1$.

Def. 59: Sei $G = (V,E)$ Graph; ein **Faktor** $F = (V_F, E_F)$ von G ist ein spannender Teilgraph, d.h. $V_F = V$ und $\forall v \in V_F \mid \exists w \in V_F$ mit $(v,w) \in E_F$.

Def. 60: Ein Faktor heißt **r-Faktor**, wenn er r-regulär ist.

Def. 61: Eine [r-]**Faktorisierung** eines Graphen ist eine Kollektion kantendisjunkter [r-]Faktoren, deren Vereinigung den vollständigen Graphen ergibt.

Def. .62: Ein Graph $G = (V,E)$ ohne Schlingen, d.h. Kanten der Form (v,v), heißt **bipartit**, wenn es eine Zerlegung V_1, V_2 von V gibt, derart daß $\forall v_1 \in V_1, v_2 \in V_2 : v_1, v_2$ sind nicht adjazent.

Es ist möglich, einen kombinatorischen Versuchsplan $(V,\mathscr{B})$ als bipartiten Graphen $G = (V_1 \cup V_2, E)$ darzustellen mit der Setzung: $V_1 := V$, $V_2 := \mathscr{B}$, $\forall v \in V, B \in \mathscr{B}: (v,B) \in E \Leftrightarrow v \in B$.

Anhang B. Zusätzliche mathematische Hilfsmittel

In diesem Abschnitt werden mathematische Hilfsmittel definiert, die im Verlauf der Arbeit zur Anwendung gelangen, aber nicht Teil der kombinatorischen Theorie der optimalen Versuchspläne sind ([AHU,74], [Ger,87], [Big,89]).

B.1 Ordnungsrelationen

Def. 63: Eine **partielle Ordnung** auf einer Menge V ist eine Relation $\preccurlyeq$, so daß für alle $a, b \in V$ gilt:

1. $a \preccurlyeq a$ ($\preccurlyeq$ ist *reflexiv*)
2. $(a \preccurlyeq b \wedge b \preccurlyeq c) \Rightarrow a \preccurlyeq c$ ($\preccurlyeq$ ist *transitiv*)
3. $(a \preccurlyeq b \wedge b \preccurlyeq a) \Rightarrow a = b$ ($\preccurlyeq$ ist *antisymmetrisch*)

Def. 64: Eine **lineare Ordnung (totale Ordnung)** $\prec$ auf einer Menge V ist eine partielle Ordnung, so daß für jedes Paar (a,b) von Elementen von V entweder gilt $a \prec b$ oder $b \prec a$.

Def. 65: Sei $\mathcal{B}$ ein k-uniformes Mengensystem über einer Menge V. Auf V existiere eine lineare Ordnung $\prec$. Für alle $(x_1, ..., x_k) \in \mathcal{B}$ sei o.B.d.A. $x_1 \prec x_2 \prec ... \prec x_k$. Die lineare Ordnung $\prec$ auf V induziert eine **lexikographische Ordnung** auf dem Mengensystem $\mathcal{B}$ vermöge der Vorschrift: $\forall (a_1, ..., a_k), (b_1, ..., b_k) \in \mathcal{B}$:
$(a_1, ..., a_k) \prec (b_1, ..., b_k) \Leftrightarrow \exists j$ mit $a_j \prec b_j \wedge a_i = b_i \ \forall 1 \leq i < j$.

B.2 Gray-Codes

Def. 66: **Gray-Codes** sind Codierungen von Dezimalzahlen in Dualzahlen, in denen Codewörter aufeinanderfolgender Dezimalzahlen sich in genau einem Bit unterscheiden. Ein solcher Code kann rekursiv (induktiv) nach folgendem Schema entwickelt werden:

- für die Zahlenbereiche $0,...,N-1 = 2^n - 1$ hat die Gray-Code-Darstellung einer Dualzahl n Bits.
- für die Zahlen $\{0,1\}$ sind die Gray-Code-Dualdarstellungen $\{0,1\}$; für die Zahlen $\{0,1,2,3\}$ sind die Gray-Code-Darstellungen $\{00,01,11,10\}$.
- seien $a_{0,0}...a_{0,n-1}, ..., a_{n-1,0}...a_{n-1,n-1}$ die Gray-Code-Dualdarstellungen des Zahlenbereichs $0,...,N-1 = 2^n - 1$, dann ergibt sich die Gray-Code-Darstellung des Bereichs $0,...,2N-1 = 2^{n+1} - 1$ rekursiv daraus wie folgt:

  $0, ..., N-1 = 0a_{0,0}...a_{0,n-1}, ..., 0a_{n-1,0}...a_{n-1,n-1}$
  $N, ..., 2N-1 = 1a_{n-1,0}...a_{n-1,n-1}, ..., 1a_{0,0}...a_{0,n-1}$

  Für den Bereich $N,...,2N-1$ wird also auf die gegebene n-Bit-Darstellung des Bereichs $0,...,N-1$ in absteigender Form zugegriffen; dies kann auch als Spiegelung aufgefaßt werden, wobei anschließend der ersten Hälfte eine 0, der zweiten Hälfte eine 1 vorangestellt wird.

B.3 Permutationen

Def. 67: Eine **Permutation** einer Menge M ist eine bijektive Abbildung $\Pi: M \to M$.

Def. 68: Die Menge aller Permutationen von $\{1,2,...,n\}$ wird mit S_n bezeichnet. S_n bildet mit der Verkettung von Abbildungen als Verknüpfung eine Gruppe. Diese heißt **Symmetrische Gruppe vom Grad** n.

Lemma 5: S_n hat $n!$ Elemente.

Def. 69: $\Pi \in S_n$ heißt r-**Zyklus**, $\Pi = (i_1 ... i_r)$, wenn es $i_1, ..., i_r \in \{1,...,n\}$ gibt, mit $\Pi(i_1) = i_2$, $\Pi(i_2) = i_3$, ..., $\Pi(i_r) = i_1$ und $\Pi(j) = j \ \forall j \notin \{i_1, ..., i_r\}$.

Def. 70: Ein 2-Zyklus heißt **Transposition**. Der 1-Zyklus ist die identische Abbildung.

Satz 6: Jedes $\Pi \in S_n$ ist Produkt von disjunkten Zyklen, also von Zyklen mit disjunkten Ziffernmengen. Jeder r-Zyklus ist Produkt (Verkettung) von $(r-1)$ Transpositionen. Jedes $\Pi \in S_n$ ist Produkt von Transpositionen aus $\{(1\,2),(2\,3),(3\,4),...,(n-1\,n)\}$.

B.4 Restklassen

Def. 71: Sei M eine Menge ganzer Zahlen, n eine natürliche Zahl. Eine Zerlegung von M in **Restklassen modulo** n ist eine Zerlegung von M in n Mengen $R_0, ..., R_{n-1}$ derart, daß die Teilmenge R_i alle diejenigen Elemente von M enthält, die bei der ganzzahligen Division durch n den Rest i liefern.

Dank

Die vorliegende Arbeit entstand im Zentralinstitut für Angewandte Mathematik (ZAM) der Forschungszentrum Jülich GmbH (KFA). Dem Institutsdirektor des ZAM und Inhaber des Lehrstuhls für Technische Informatik und Computerwissenschaften an der RWTH Aachen, Herrn Prof. Dr. F. Hoßfeld, gilt mein herzlicher Dank für die Anregung des Themas, seine Betreuung und wichtige Impulse während der Ausarbeitung.

Herrn Prof. Dr. W. Oberschelp, dem Inhaber des Lehrstuhls für Angewandte Mathematik insbesondere Informatik an der RWTH Aachen, danke ich für die Übernahme des Korreferats.

Besonderer Dank gebührt Herrn Dr. P. Weidner, dem Leiter der Abteilung Mathematik des ZAM, für zahlreiche wertvolle Ratschläge im Verlauf der Arbeit.

Den Herren Dr. R. Esser, Dr. M. Gerndt, Dr. J. Grotendorst., Dr. E.M.E. Wermuth und Frau Dipl.-Inform. R. Knecht gilt mein Dank für hilfreiche Diskussionen und Anregungen. Herrn W. Frings danke ich für die tatkräftige Unterstützung bei der Erzeugung und Implementierung von Sonderzeichen.

Die kollegiale Arbeitsatmosphäre sowie die Hilfsbereitschaft der Mitarbeiter im ZAM haben wesentlich zum Gelingen dieser Arbeit beigetragen. Ihnen allen gilt an dieser Stelle mein Dank. Besonders fruchtbar war die Zusammenarbeit im Kreise der Diplomanden und Doktoranden, von denen ich insbesondere meinem damaligen Bürokollegen Herrn Dr.-Ing. S. Heydorn sehr verbunden bin.