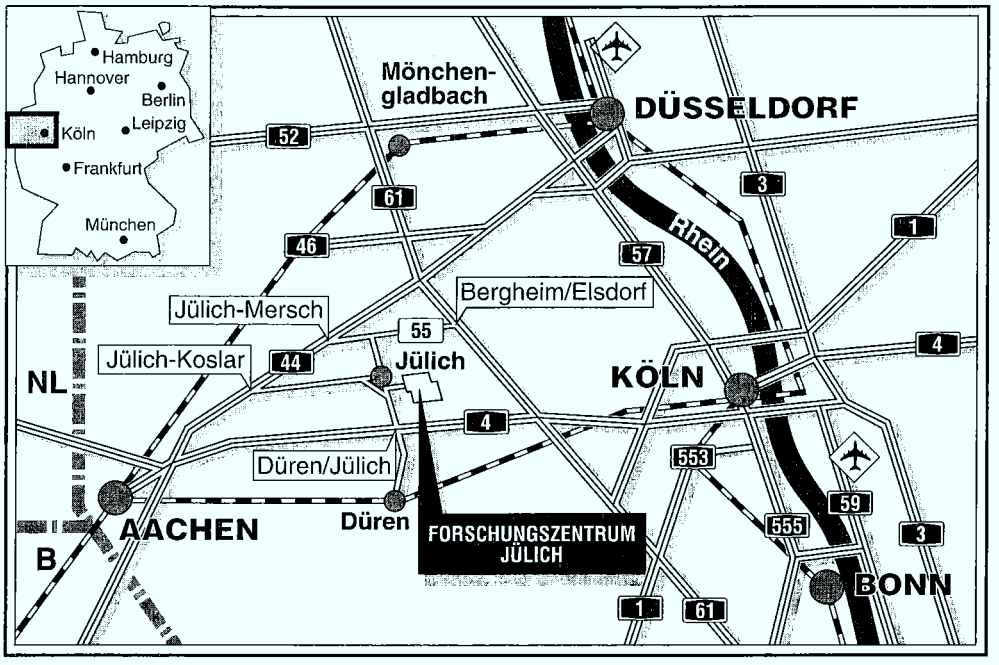
FORSCHUNGSZENTRUM JÜLICH GmbH

KFA

Institut für Sicherheitsforschung und Reaktortechnik

Parallelisierung des Finite-Element-Codes SMART-Konvektion

Astrid Watermann



Berichte des Forschungszentrums Jülich ; 3122
ISSN 0944-2952
Institut für Sicherheitsforschung und Reaktortechnik Jül-3122
D 294 (Diss. Universität Bochum)

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

Parallelisierung des Finite-Element-Codes SMART-Konvektion

Astrid Watermann

Abstract

Many science and engineering problems can be described by systems of partial differential equations. Due to the fact that analytical solutions of those systems exist only for some special cases, it is necessary to apply numerical solution strategies such as finite difference or finite element methods. Massively parallel computers allow to reduce the computing times of the comprehensive numerical calculations by subdividing the calculation into parts and assigning them to different processors.

In this thesis, the finite element code "SMART-Konvektion" is taken to develop and implement a concept which enables a complex computer code system to be ported to a massively parallel computer system. This concept is based on a decomposition of the considered finite element grid, which allows a parallel organization of the data and the solution of the systems of equations by an efficient parallelly executable conjugate gradient method.

Three examples as there are the Rayleigh-Bénard-convection, a convective flow with phase change, and a forced convection flow are analyzed with the computer code in cosideration to valuate its power increase on the parallel computer Intel/Paragon. A good speedup can be recognized for all examples.

Kurzfassung

Viele natur- und ingenieurwissenschaftliche Vorgänge sind durch Systeme partieller Differentialgleichungen beschreibbar. Da analytische Lösungen dieser Systeme nur für Spezialfälle bekannt sind, müssen numerische Lösungsverfahren wie beispielsweise Finite-Differenzen- oder Finite-Element-Verfahren angewendet werden. Massiv parallele Rechner erlauben, die Ausführungszeiten dieser umfangreichen numerischen Berechnungen zu verkürzen, indem einzelne Berechnungsabschnitte auf verschiedenen Rechenknoten ausgeführt werden.

In dieser Arbeit wird anhand des Finite-Element-Produktionscodes "SMART-Konvektion" ein Konzept entwickelt und implementiert, das es ermöglicht, ein komplexes Programmsystem auf einen Parallelrechner zu portieren. Dieses Konzept basiert auf einer Zerlegung des betrachteten Finite-Element-Netzes, wodurch eine parallele Datenverwaltung und die Lösung der entstehenden Gleichungssysteme durch ein effizient parallel ausführbares, konjugiertes Gradienten-Verfahren ermöglicht wird.

Für das betrachtete Programmsystem wird anhand von drei Beispielen, der Rayleigh-Bénard-Konvektion, einer Konvektionsströmung mit Phasenwechsel und einer erzwungenen Konvektionsströmung der Leistungszuwachs auf dem Parallelrechner Intel/Paragon analysiert und ein guter Leistungszuwachs verzeichnet.

Inhaltsverzeichnis

Seite

Bildverzeichnis

Tabellenverzeichnis

1 Einleitung und Problemstellung

Das Verhalten komplexer natur- und ingenieurwissenschaftlicher Vorgänge ist meist durch Systeme partieller Differentialgleichungen beschreibbar. Da analytische Lösungen dieser Systeme nur für Spezialfälle bekannt sind, müssen numerische Lösungsverfahren wie beispielsweise Finite-Differenzen- oder Finite-Element-Verfahren angewendet werden. Viele der zu berechnenden Probleme stellen hohe Anforderungen an Leistung und Speicherkapazität der eingesetzten Rechner. Beispielsweise wird für die Berechnung turbulenter Strömungen in [NSF94] die erforderliche Rechenleistung auf 1 TFLOPs und die Speicherkapazität auf 80 GByte geschätzt. Skalar- und Vektorrechner können diese Leistungsanforderungen bislang nicht erfüllen; so ermöglicht der in der KFA installierte Vektorrechner Cray M94 mit 4 Prozessoren eine maximale Leistung von 1,3 GFLOPs bei 2 GByte Hauptspeicher.

Massiv parallele Rechner bilden dagegen eine Möglichkeit, den oben aufgeführten Anforderungen an Leistung und Speicherkapazität gerecht zu werden, indem viele Prozessoren mit lokalem Speicher über ein schnelles Netzwerk miteinander verbunden werden. Der in der KFA vorhandene Rechner Intel/Paragon bietet mit 140 Rechenknoten eine maximale Leistung von 10,5 GFLOPs und einen Speicherbereich von 4,5 GByte. Diese Architektur ist bis zu etwa 300 GFLOPs skalierbar und kann als ein Schritt in Richtung einer Leistungsstärke im TFLOPs-Bereich gewertet werden.

Während die Entwicklung der Parallelrechner fast Marktreife erreicht hat, stehen dem Anwender nur wenige Programme zur Verfügung, die diese Architekturen nutzen können.

In der vorliegenden Arbeit wird untersucht, ob sich der durch parallele Architekturen erzielbare Leistungszuwachs für die im Institut für Sicherheitsforschung und Reaktortechnik des Forschungszentrums Jülich durchzuführenden numerischen Berechnungen nutzen läßt. Dazu wird beispielhaft das Finite-Element System SMART-Konvektion betrachtet, für das der Quellcode vorliegt. SMART ermöglicht insbesondere die Berechnung komplexer Wärmeübertragungsprobleme. Langjährige Erfahrungen bei der Anwendung von SMART und dessen intensive Verifikation erlauben es nicht, dieses etwa 300 000 Anweisungen umfassende System in absehbarer Zeit durch ein von Grund auf neu entwickeltes paralleles Programm zu ersetzen und begründen die Untersuchungen zur Parallelisierung des bestehenden Systems. Dabei sollen neben der Erstellung des parallelen Programmes auch Erfahrungen bei der Parallelisierung umfangreicher Produktionsprogramme gesammelt werden.

Mit ParSmart wird eine Parallelversion entwickelt, implementiert und verifiziert, welche das bislang serielle Programmsystem SMART auf den Parallelrechner Intel/Paragon abbildet. Unter Berücksichtigung der vorgegebenen Programmstruktur und Rechnerarchitektur wird ein Konzept gleichberechtigter Knoten entworfen. Dazu wird das zu berechnende Gesamtmodell in Teilmodelle zerlegt, die jeweils den einzelnen Rechenknoten zugeordnet werden. Die parallel arbeitenden Rechenknoten behandeln lokal die Teilmodelle. Bei der Berechnung des Gesamtmodells sind jedoch die Abhängigkeiten der Teilmodelle untereinander zu berücksichtigen, d.h., daß Daten zwischen den Teilmodellen ausgetauscht werden müssen.

Dieser Datentransfer wird durch ein für ParSmart neu entwickeltes Kommunikationsschema ausgeführt. Dazu sind die im seriellen System vorhandenen Datenstrukturen um ein erweitertes Hypermatrizenschema und Zuordnungstabellen zu ergänzen.

Da etwa 80–90% der Rechenzeit einer SMART-Rechnung zur Lösung der bei Finite-Element-Berechnungen aufgestellten, dünnbesetzten Gleichungssysteme benötigt werden, muß die Parallelisierungsstrategie insbesondere für diesen Berechnungsabschnitt ein effizient parallel ausführbares Lösungsverfahren berücksichtigen. Für das parallele Programmsystem ParSmart wird deshalb eine algorithmische Änderung vorgenommen, indem das bislang implementierte, direkte Cholesky-Verfahren durch ein iteratives konjugiertes Gradienten-Verfahren ersetzt wird. Dazu wird das Speicherschema der Hypermatrizen durch ein zeilenorientiertes Speicherschema ergänzt.

Die Rechenzeit einer parallelen Anwendung nimmt zunächst mit wachsender Anzahl der Rechenknoten ab, da diese die Berechnung untereinander aufteilen. Während im Idealfall beliebig viele Rechenknoten für eine Rechnung genutzt werden können, ist es auch möglich, daß die Rechenzeiten wieder ansteigen, wenn zuviele Rechenknoten ein kleines Problem bearbeiten. Aus diesem Grund wird zur Erzielung einer minimalen Rechenzeit für das Programmsystem ParSmart die optimale Anzahl der Rechenknoten abgeschätzt.

Die Verifikation von ParSmart sowie die Bewertung des Parallelrechnereinsatzes werden anhand von drei Anwendungen durchgeführt. Neben dem Beispiel der Rayleigh-Bénard-Konvektion erfolgt die Versuchsnachrechnung einer Wasserstoffausbreitung auf einer Wasseroberfläche und die Umströmung eines Laminarflügels.

Heterogene Rechnersysteme kombinieren Skalar-, Vektor- und Parallelrechner miteinander und bilden somit eine Alternative zur der homogenen Architektur des Intel/Paragon. Da diese sogenannten Metacomputer optimale Architekturen für verschiedenartige Berechnungsabschnitte bereitstellen, wird untersucht, wie diese die Rechenzeiten einer SMART-Rechnung weiter verkürzen können.

In Kapitel 2 werden die zum Verständnis des Programmsystems SMART erforderlichen Grundlagen bereitgestellt. Diese beinhalten die Beschreibung von Konvektionsströmungen sowie die Grundlagen der Parallelverarbeitung. Ein separater Abschnitt erläutert den Parallelrechner Intel/Paragon, auf dem ParSmart zunächst installiert wird. Das Kapitel schließt mit einem Überblick über bisherige Arbeiten zu Berechnungen nach der Methode der Finiten Elemente auf Parallelrechnern.

Ein Überblick über das serielle Programmsystem SMART wird in Kapitel 3 gegeben. Es werden die mathematischen Modelle und Algorithmen und deren Umsetzung in SMART erläutert. Weitere Abschnitte behandeln die Anwenderseite sowie die Systemebene dieses Programmsystems. Dadurch wird ein Einblick in die berechenbaren Probleme und die Programm- und Datenstrukturen gegeben. Abschließend wird der Ablauf einer seriellen Berechnung mit SMART vorgestellt.

Das neu entwickelte Konzept gleichberechtigter Knoten für die Parallelversion ParSmart wird in Kapitel 4 beschrieben. Dieses Gesamtkonzept wird unter Beachtung der von der Parallelversion zu erfüllenden Anforderungen und der Einsatzmöglichkeiten parallel arbei-

tender Prozessoren aufgestellt. Die Beschreibung einer parallel ablaufenden Berechnung mit ParSmart ergänzt das Kapitel.

Die notwendigerweise neu eingeführten Algorithmen und Datenstrukturen für die Parallelversion werden in Kapitel 5 vorgestellt. Insbesondere sind dieses die Verfahren zur Modellzerlegung, Kommunikation und Lösung der Gleichungssysteme sowie die Zuordnungstabellen, das erweiterte Hypermatrizen– und das zeilenorientierte Speicherschema. Da ein Problem gegebener Größe nicht auf einer beliebigen Anzahl von Rechenknoten effizient lösbar ist, wird eine Abschätzung zur Ermittlung der optimalen Anzahl der Rechenknoten für das betrachtete Problem hergeleitet.

In Kapitel 6 werden Rechnungen mit der Parallelversion ParSmart für ausgewählte Anwendungen vorgestellt. Die Beispiele dienen zur Verifikation der Parallelversion und decken einen repräsentativen Bereich der mit ParSmart berechenbaren Probleme ab. Anhand der Ergebnisse dieser Berechnungen wird der durch den Parallelrechnereinsatz erreichbare Leistungszuwachs bewertet.

Die Ergebnisse aus Kapitel 6 zeigen, daß die homogene Architektur des Intel/Paragon nicht ohne Einschränkungen für Finite-Element-Berechnungen mit Programmsystemen wie SMART geeignet ist. In Kapitel 7 werden deshalb Einsatzmöglichkeiten heterogener Rechnersysteme für SMART diskutiert.

Die Arbeit schließt mit einer Zusammenfassung der neuen Erkenntnisse und einem Ausblick auf zukünftige Arbeiten ab.

2 Grundlagen

Die Grundlagen, die zum Verständnis der mit SMART berechenbaren Konvektionsströmungen und der neu zu entwickelnden Parallelversion erforderlich sind, werden in diesem Kapitel beschrieben. Die physikalischen Eigenschaften einer Konvektionsströmung werden zusammengefaßt sowie Beispiele für die praktische Relevanz dieser Strömungsart im technischen Bereich aufgezeigt. Der folgende Abschnitt über Parallelverarbeitung klassifiziert die Rechnerarchitekturen und Programmiermodelle und beschreibt Bewertungsmaßstäbe zur Beurteilung der parallelen Berechnungen. Da die Parallelversion von SMART zunächst für den in der KFA vorhandenen Rechner Intel/Paragon entwickelt werden soll, wird in einem weiteren Abschnitt die Architektur und die Software dieses Parallelrechners mit verteiltem Speicher vorgestellt.

Abschließend wird ein Überblick über der Stand der Parallelisierungsmethoden für Finite-Element-Berechnungen und dafür genutzte Algorithmen gegeben, der es ermöglicht, die Untersuchungen zur Parallelisierung des Programmsystems SMART einzuordnen.

2.1 Konvektionsströmungen

Mit SMART können unter anderem Konvektionsströmungen berechnet werden [LS91], [Mer87]. Die Rechenzeiten für die Lösung dieser Probleme sind besonders hoch, da Temperatur- und Geschwindigkeitsfelder miteinander gekoppelt sind. Bei instationären Berechnungen oder Anfangswertproblemen sind dann in jedem Zeitschritt Gleichungssysteme für die Geschwindigkeitskomponenten und für die Temperaturen zu bestimmen, deren Lösungen sich gegenseitig beeinflussen und ein iteratives Prädiktor-Korrektor-Verfahren erfordern [LS91]. Die Parallelisierung des Programmsystems SMART bezieht sich im ersten Schritt nur auf den Programmteil zur Berechnung der Konvektionsströmungen. In diesem Abschnitt werden die physikalischen Eigenschaften der Konvektionsströmungen und praxisrelevante Anwendungen beschrieben.

2.1.1 Klassifizierung

Konvektionsströmungen lassen sich in natürliche (freie) und erzwungene Strömungen einteilen [ZO82]. In natürlichen Konvektionsströmungen wird die Bewegung des Fluids nur durch die Auftriebskräfte, d.h. durch Temperatur– oder Dichteunterschiede hervorgerufen. Bei erzwungener Konvektion wird die Strömung durch eine Druckdifferenz erzeugt [GH92], [Mer87], [Tri88], [ZO82], [Hic93], beispielsweise durch Pumpen oder Ventilatoren. Das Strömungsfeld kann dann unabhängig vom Temperaturfeld berechnet werden, wenn die Temperaturunterschiede die Strömung nicht oder nur in geringem Maße beeinflussen. Die Simulation dieser erzwungenen Strömungen vereinfacht sich gegenüber der freien Konvektion, da nur der Einfluß des Geschwindigkeitsfeldes auf das Temperaturfeld berücksichtigt werden muß.

Dimensionslose Kennzahlen für die erzwungene Konvektion sind [Mer87]:

Reynoldszahl	$Re = \frac{ul}{\nu}$,
Pécletzahl	$Pe = \frac{ul}{a}$,
Nußeltzahl	$Nu = \frac{\alpha l}{k_T}$.

Dabei kennzeichnet u die Geschwindigkeit, l die charakteristische Länge, ν die kinematische Viskosität, a die Temperaturleitfähigkeit, α die Wärmeübergangszahl und k_T die Wärmeleitfähigkeit.

Bei der freien Konvektion sind Strömung und Wärmeübertragungsmechanismen eng miteinander verbunden, die Lösungen für Temperatur- und Geschwindigkeitsfelder beeinflussen sich gegenseitig. Kennzahlen für die natürliche oder freie Konvektion sind [Mer87]:

Grashofzahl	$Gr = \frac{gl^3}{\nu^2}\beta\vartheta$,
Prandtlzahl	$Pr = \frac{\nu}{a}$,
Nußeltzahl	$Nu = \frac{\alpha l}{k_T}$,

mit den zuvor genannten Bezeichnungen. Zusätzlich kennzeichnet g die Gravitationskonstante, β den thermischen Ausdehnungskoeffizienten, sowie $\vartheta = |T_W - T_\infty|$ den charakteristischen Wert für den Temperaturunterschied zwischen der Wandtemperatur T_W und der mittleren Fluidtemperatur T_∞. Statt der Grashofzahl wird auch die Rayleighzahl

$$Ra = \frac{g\beta l^3 |T_W - T_\infty|}{\nu a} = Gr \cdot Pr \tag{2.1.1}$$

verwendet.

Neben der Unterscheidung freier und erzwungener Konvektionsströmungen werden diese wie auch andere Strömungsarten in laminare und turbulente Strömungen eingeteilt [GH92], [Mer87], [Ung88]. Zur Zeit können mit SMART nur Probleme gelöst werden, die sich durch das laminare Modell formulieren lassen [Szi89]. Laminares Verhalten tritt bei freien Konvektionsströmungen im Bereich für Rayleighzahlen $Ra \leq 10^8$ auf. Der Übergangsbereich zur turbulenten Strömung liegt etwa bei $10^8 \leq Ra \leq 10^{10}$, während für $Ra > 10^{10}$ vollständige Turbulenz vorliegt [POW90]. Für erzwungene Konvektionsströmungen tritt laminares Verhalten beispielsweise an einer längsangeströmten Platte bei Reynoldszahlen bis etwa $3 \cdot 10^5$ auf [BS94]. An dieser Stelle sollen nur laminare Strömungen oder Strömungen im Grenzbereich zur Turbulenz, die sich noch durch das laminare Modell formulieren lassen, betrachtet werden [Lan91]. Da praktische Problemstellungen oft auch turbulente Vorgänge beinhalten, ist für die Zukunft geplant, das Programmsystem SMART um ein Turbulenzmodell, beispielsweise das k-ϵ–Modell, zu erweitern [GH92], [Lan91], [Mer87].

2.1.2 Relevanz in technischen Anwendungen

Konvektionsströmungen sind in den technischen Anwendungen bedeutsam, in denen Temperaturunterschiede zwischen einzelnen Komponenten auftreten [Zie77].

Problemstellungen kommen aus den Bereichen Energietechnik, Verbrennungstechnik, Materialwissenschaften und der Luft- und Raumfahrttechnik [ZO82]. Beispielsweise sind Konvektionsströmungen bei Konstruktion und Betrieb von Heizkesseln und Hochöfen, der Wärmeübertragung in Kernreaktoren sowie in großen Wärmespeichersystemen zu berücksichtigen. Ferner sind sie in Sonnenkollektoren, bei Isolierungen, sowie in Klimaanlagen von Bedeutung. Auch bei Kristallzuchtverfahren für Mikroprozessoren [Tri88], elektrischen Schaltkreisen [Sch88], sowie der Mikrostrukturtechnik treten diese Strömungen auf.

Bereiche umwelttechnischer Anwendungen sind Biologie, Geophysik, Astrophysik, Meteorologie und Ozeanographie [ZO82]. Konvektionsströmungen müssen hier unter anderem bei der Bestimmung der atmosphärischen Zirkulation aufgrund von Temperaturunterschieden und der Berechnung der atmosphärischen Struktur der Planeten beachtet werden sowie beispielsweise bei der Untersuchung des freien Aufsteigens erwärmter Luft über einem Feuer.

Mit SMART kann derjenige Teil der oben aufgeführten Anwendungen berechnet werden, bei dem laminares Strömungsverhalten auftritt und der durch die noch in Abschnitt 3.1.1 zu erläuternden, vereinfachten Grundgleichungen beschrieben werden kann.

2.2 Parallelverarbeitung

In diesem Abschnitt werden die für eine Parallelverarbeitung wichtigen Grundlagen zusammengefaßt. Zunächst erfolgt ein Überblick über die Anforderungen an Rechenzeit und Speicherkapazität zur Lösung derzeit wichtiger Problemstellungen. Anschließend werden parallele Rechnerarchitekturen und Programmiermodelle beschrieben und Bewertungskriterien für den Einsatz von Parallelrechnern diskutiert.

2.2.1 Motivation

Derzeit verfügbare Skalar- und Vektorrechner der Hochleistungsklasse können für die Praxis interessante Aufgabenstellungen aus Gründen unzureichender Rechen- und Speicherplatzkapazität nicht oder nur unter erheblichen Vereinfachungen lösen. Während beispielsweise für die Berechnung turbulenter Strömungsvorgänge, wie in Bild 2.2.1 gezeigt, die erforderliche Rechenleistung auf 1 TFLOPs und die Speicherkapazität auf 80 GByte (1 Wort entspricht 8 Byte) geschätzt wird, ermöglichen Vektorrechner wie die in der KFA installierte Cray M94 mit 4 Prozessoren zur Zeit eine maximale Leistung von 1,3 GFLOPs bei 2 GByte Hauptspeicher. Demgegenüber bietet der Parallelrechner Intel/Paragon in einer Konfiguration mit etwa 2000 Rechenknoten, die jeweils zwei Rechenprozessoren und einen Kommunikationsprozessor umfassen, eine Leistung von etwa 300 GFLOPs und einen Hauptspeicherbereich von 256 GByte.

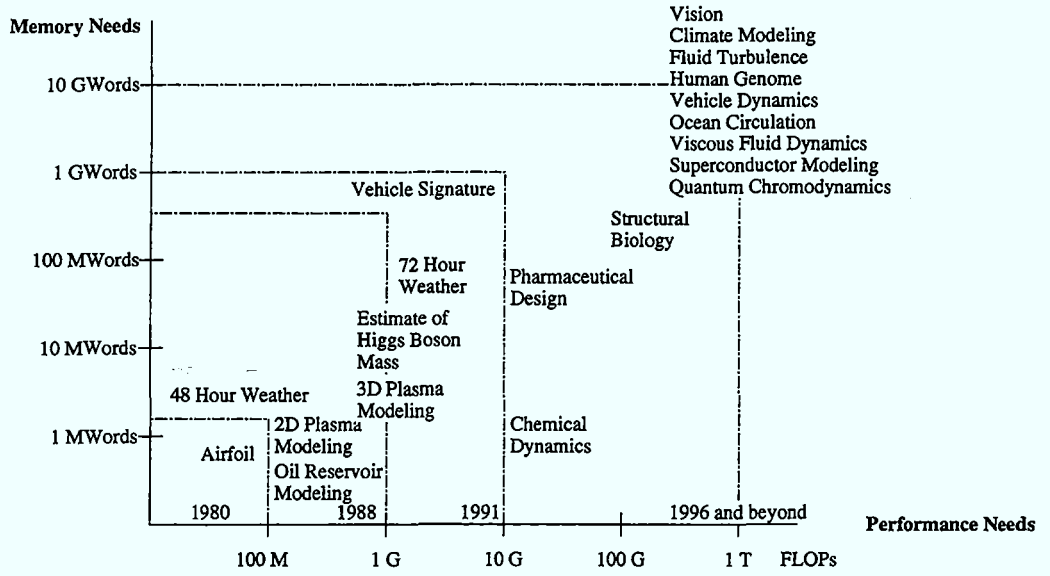


Bild 2.2.1: Anforderungen an Rechenleistung und Speicherkapazität zur Lösung der Grand Challenges, [NSF94]

Weltweite Initiativen bewerten die Entwicklung massiv paralleler Rechnersysteme und fordern deren Einsatz. In [BMF93] sieht das BMFT (jetzt: BMBF) durch den Einsatz massiv paralleler Rechner unter anderem Fortschritte in den Bereichen der Umwelt- und Klimaforschung, der Polymerphysik und der Quantenchemie, der Automobil-, Luft- und Raumfahrttechnik sowie der Energietechnik. Für die Europäische Union werden Perspektiven für zu lösende Problemstellungen in [EEC91] und [EG92] sowie für die USA in dem Projekt der *Grand Challenges* der National Science Foundation [NSF94] formuliert. Die allgemeinen Anforderungen an Speicherplatz und Leistungsfähigkeit zur Lösung aktueller Problemstellungen werden in Bild 2.2.1 aufgezeigt.

Massiv parallele Rechner können zukünftig die Anforderungen an Rechenleistung und Speicherkapazität erfüllen. Dazu werden Rechenknoten (derzeit etwa 100–2000) über ein schnelles Netzwerk miteinander verbunden. Neben der erreichbaren Leistung und der Hauptspeichergröße spielen jedoch auch die Kosten für die Rechnersysteme eine wichtige Rolle. Während eine Cray Y-MP mit 8 Prozessoren und ein Intel/Paragon mit 128 Rechenknoten jeweils eine Leistung von etwa 1,4 GFLOPs aufweisen, betragen die Kosten pro GFLOP bei der Cray 13,33 Mio $, beim Intel/Paragon jedoch nur 2,38 Mio $.

Ziele des Einsatzes der parallelen Systeme sind zum einen die Berechnung von Problemstellungen, die bislang nicht durchgeführt werden konnten, und zum anderen die Reduzierung der Rechenzeiten.

Kurze Rechenzeiten sind besonders dort von Interesse, wo die Ergebnisse nur für einen bestimmten Zeitbereich gültig sind. Sehr deutlich wird das am Beispiel der Wettervorhersage. Benötigt die Berechnung der Wettervorhersage für 48 Stunden etwa die gleiche Rechenzeit oder sogar länger, so ist das Ergebnis der Berechnung überholt. Auch die Simulation von Unfällen in großtechnischen Anlagen (z.B. Schadstoffausbreitung) muß deutlich schneller

erfolgen, um rechtzeitig erforderliche Gegenmaßnahmen einleiten zu können [TBC+94]. Für die Bilderkennung im Rahmen der automatischen Fahrzeugsteuerung muß hingegen eine Echtzeitverarbeitung erfolgen [BMF93].

2.2.2 Parallele Rechnerarchitekturen und Programmiermodelle

Parallelrechner können hinsichtlich der Daten- und Anweisungslokalität unterteilt werden. Diese von Flynn [Fly66] eingeführte Klassifizierung unterscheidet SISD (Single Instruction Single Data), SIMD (Single Instruction Multiple Data), MISD (Multiple Instruction Single Data) und MIMD (Multiple Instruction Multiple Data) Architekturen. Rechner mit SISD Architektur führen eine Anweisungsfolge auf einer Datenmenge aus und entsprechen den herkömmlichen, seriellen Rechnern. Bei der SIMD-Architektur wird dieselbe Anweisungsfolge auf unterschiedlichen Daten ausgeführt, sie entspricht den Vektorrechnern. Bei Rechnern der Klasse MISD wären auf jedem der parallel arbeitenden Rechenknoten andere Routinen, jedoch auf denselben Daten ausführbar; diese Klasse ist praktisch nicht relevant. Die Klasse der MIMD Rechner umfaßt Architekturen, die auf verschiedenartigen Datenmengen unterschiedliche Anweisungsfolgen ausführen können. Parallelrechner entsprechen dieser Architektur [JS92].

Architekturen	Anweisungen		Daten	
	SI	MI	SD	MD
SISD	X		X	
SIMD	X			X
MISD		X	X	
MIMD		X		X
S: Single, M: Multiple, D: Data, I: Instruction				

Tabelle 1: Klassifizierung der Rechnerarchitekturen

Eine weitere Einteilung der Parallelrechner kann in Systeme mit verteiltem oder mit gemeinsamem Speicher erfolgen. In Systemen mit verteiltem Speicher besitzt jeder Rechenknoten seinen eigenen lokalen Hauptspeicher, auf den er exklusiv zugreifen kann. Müssen Daten mit anderen Rechenknoten ausgetauscht werden, so geschieht dieses, indem Nachrichten untereinander verschickt werden *(Message-passing)*. Dieses Modell hat den Vorteil, daß es bei gleichen Hauptspeicherzugriffszeiten beliebig skalierbar ist [JS92]. Parallelrechner mit verteiltem Speicher werden weiterhin durch die Art des Verbindungsnetzwerkes zwischen den Prozessoren charakterisiert, da diese die Kommunikationszeiten beeinflussen. Mögliche Netzwerke haben eine Gitter-, Ring-, Torus-, Baum- oder Hypercube-Struktur [Hwa93].

In Systemen mit gemeinsamem Speicher greifen alle Rechenknoten auf einen Hauptspeicher zu. Probleme entstehen dabei primär durch Zugriffskonflikte, da die parallelen Prozesse

über gemeinsame Variablen kommunizieren; beispielsweise verhindern Semaphore Zugriffskonflikte und Verklemmungen [Bab88], [Hwa93]. Der Benutzer muß somit programmtechnisch sicherstellen, daß verschiedene Rechenknoten nicht gleichzeitig auf dieselben Daten zugreifen [GB95], [GO93].

Es ist zu hoffen, daß sich zukünftig Rechnerarchitekturen mit einem virtuellen gemeinsamen Speicher durchsetzen werden [Her93]. Dieses sind Systeme mit verteiltem Speicher, die dem Benutzer einen globalen Adreßraum zur Verfügung stellen. Dabei wird der Vorteil der beliebigen Skalierbarkeit des Hauptspeichers mit dem Vorteil für den Programmierer, kein explizites *Message-passing* und entsprechende Datenaufteilung programmieren zu müssen, verbunden.

Der Programmierung paralleler Rechnerarchitekturen können verschiedenartige Programmiermodelle zugrunde liegen. In Systemen mit verteiltem Speicher werden Nachrichten durch *Message-passing* zwischen den Rechenknoten verschickt [McB94]. Die meisten Rechnerarchitekturen bieten wie auch der Intel/Paragon systemspezifische Routinen für das *Message-passing* an. Diese haben den Vorteil, die Systemarchitektur sehr gut zu nutzen, weisen allerdings den Nachteil auf, daß sie nicht auf andere Parallelrechnersysteme übertragbar sind. Aus diesem Grunde wurden systemunabhängige Programme als Schnittstelle zwischen Parallelrechner und Anwenderprogramm eingeführt. Diese bieten dem Anwendungsprogrammierer eine *Message-passing*–Umgebung, die auf den verschiedenen Parallelrechnersystemen lauffähig ist. Solche Schnittstellen zwischen Rechnerarchitekturen und Anwenderprogrammen stellen z.B. die Systeme PARMACS [HHS92], PVM [GBD+93] und MPI [Hem94], [Uni94] dar.

Für Systeme mit gemeinsamem oder virtuell gemeinsamem Speicher wird das datenparallele Programmiermodell HPF (High Performance Fortran) [For94] angeboten. Parallelität wird dabei durch Feld-Operationen ausgedrückt, und Direktiven beschreiben, wie diese Felder auf die Knoten des Parallelrechners verteilt werden sollen. Explizite Kommunikationsanweisungen werden vom Compiler und nicht vom Programmierer eingesetzt. HPF bietet somit ein einfaches Programmiermodell und Portabilität auf andere Systeme.

Die Wahl des Programmiermodells ist von der Rechnerarchitektur und der Anforderung an Portabilität auf andere Systeme abhängig. Muß das parallele Programm auf verschiedenen Parallelrechnern laufen, so ist eine Schnittstelle zwischen Architektur und Anwendungsprogramm erforderlich, um nur eine parallele Programmversion entwickeln und pflegen zu müssen. Die so erreichbare Portabilität bedeutet jedoch eine Leistungseinschränkung, da gegenüber dem maschinenspezifischen *Message-passing* zusätzlicher Verwaltungsaufwand erforderlich wird. Soll ein Anwendungsprogramm nur auf einem bestimmten Parallelrechner implementiert werden, so sind die systemabhängigen *Message-passing*–Routinen vorzuziehen, da diese die Rechnerarchitektur optimal nutzen und somit die höchste Leistung erzielen.

Die vorangehend zusammengefaßten Programmiermodelle erfordern von dem Programmierer Eingriffe in das parallel auszuführende Programm. Dieses erscheint notwendig, da eine

automatische, beispielsweise vom Compiler durchgeführte Parallelisierung für allgemeine Anwendungscodes bislang nur ungenügende Ergebnisse liefert [DPR94].

2.2.3 Bewertung paralleler Berechnungen

Werden parallele Programme entwickelt, so muß neben den Aspekten der seriellen Programmentwicklung [Gri87] insbesondere beachtet werden, daß alle genutzten Rechenknoten hinsichtlich Rechenzeit und Speicherbelegung gleichmäßig ausgelastet werden und daß das parallele Programm skalierbar ist, d.h. auf einer beliebigen Anzahl von Rechenknoten mit gleicher Effizienz ausgeführt werden kann [Pet92].

Die Güte paralleler Berechnungen wird durch Speedup und Effizienz beurteilt, die nach folgenden Definitionen gebildet werden [GO93], [Hwa93]:

Definition: *(Speedup)*

Der Speedup wird als Quotient zwischen dem Zeitverbrauch des seriellen zum parallelen Algorithmus

$$\text{Speedup } S_p = \frac{\text{Rechenzeit auf einem Knoten}}{\text{Rechenzeit auf } p^* \text{ Knoten}} \tag{2.2.2}$$

auf p^* Knoten definiert.

Neben der oben genannten Definition werden in der Literatur auch andere Definitionen eingeführt [GO93], [Hwa93]. Bei der Betrachtung von Speedup-Faktoren ist deshalb stets die genaue Definition zu berücksichtigen.

Die Auslastung der genutzten Rechenknoten wird durch die Effizienz bestimmt:

Definition: *(Effizienz)*

Die Effizienz E_p kennzeichnet den Quotienten zwischen Speedup S_p und Anzahl p^* der verwendeten Rechenknoten:

$$\text{Effizienz } E_p = \frac{S_p}{p^*}. \tag{2.2.3}$$

Die Anforderung der Skalierbarkeit bedeutet eine konstante Effizienz E_p und somit linearen Speedup. Für praktische Anwendungen gilt jedoch Amdahl's Law, nach dem der erreichbare Speedup unter dem linearen Speedup liegt, wenn in dem parallelen Code ein serieller Anteil existiert.

Satz *(Amdahl's Law, [Amd67])*

Liegt ein serieller Anteil α^* in dem parallelen Algorithmus vor, so ist maximal der Speedup

$$S_{p,\alpha} = \frac{p^*}{1 + (p^* - 1)\alpha^*} \tag{2.2.4}$$

erreichbar.

Amdahl's Law kann graphisch entsprechend Bild 2.2.2 dargestellt werden [GO93], [Hwa93].

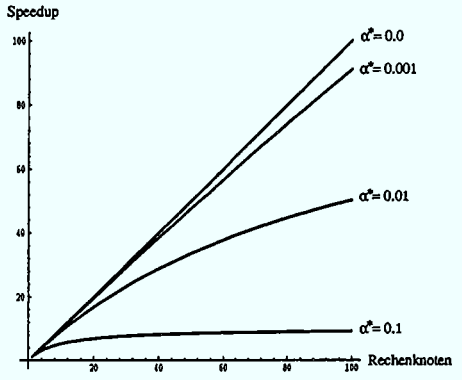


Bild 2.2.2: Amdahl's Law

Der Parameter α^* gibt den vorhandenen seriellen Anteil an. Nur für $\alpha^* = 0$ ist ein linearer Speedup erreichbar. Existiert ein serieller Anteil $\alpha^* > 0$, so liegt der maximale Speedup bei $\lim_{p^* \to \infty} \frac{p^*}{1+(p^*-1)*\alpha^*} = \frac{1}{\alpha^*}$, unabhängig von der Anzahl der verwendeten Rechenknoten. Enthält ein paralleler Code z.B. 5% Anweisungen, die nur seriell ausführbar sind, so ist asymptotisch ein maximaler Speedup von 20 erreichbar. Auch der Einsatz beliebig vieler Rechenknoten kann dann diese Situation nicht verbessern. Wird die Anzahl der Rechenknoten weiter erhöht, so wächst der Speedup nicht mehr und die Effizienz nimmt ab.

Bei der Neuentwicklung paralleler Programme sowie bei der Parallelisierung bestehender serieller Codes muß insbesondere angestrebt werden, jeden seriellen Anteil zu minimieren ($\alpha^* \to 0$), um Skalierbarkeit gewährleisten zu können. Leistungsverluste treten jedoch nicht nur durch serielle Anweisungsabschnitte auf, sondern werden auch durch die Hardware bedingt, insbesondere durch Kommunikation und IO.

In der Praxis wächst in der Regel die Problemgröße mit der Anzahl der verfügbaren Rechenknoten, d.h. es wird nicht ein festes Problem auf einer wachsenden Anzahl von Rechenknoten betrachtet, sondern es werden größere Probleme berechnet. Ein solches Verhalten kann wie in [GO93] und [Gus88] als *scaled Speedup* definiert werden.

Neben den zuvor genannten Bewertungskriterien können vom Anwender weitere Anforderungen gestellt werden. So kann beispielsweise unter Vernachlässigung der entstehenden Kosten nicht die Auslastung der Prozessoren, sondern die Rechenzeit entscheidend sein, in der ein Problem gelöst werden kann. Unter diesem Aspekt kann auch die Nutzung paralleler Prozessoren mit geringer Effizienz erwünscht sein, wenn dadurch die Rechenzeiten entsprechend reduziert werden.

Werden für die Programmläufe Kosten für die Belegung der Rechenknoten angerechnet, so ist die Effizienz, d.h. die Auslastung der Rechenknoten, ein wichtiges praktisches Kriterium zur Bewertung einer parallelen Berechnung. Optimaler, d.h. linearer Speedup bedeutet eine minimale Rechenzeit und maximale Effizienz, so daß linearer Speedup stets Ziel der

Parallelisierungsarbeiten sein muß. Weitere Bewertungsmaßstäbe für parallele Berechnungen werden beispielsweise in [JS92] diskutiert.

2.3 Intel/Paragon XP/S

Der im Zentralinstitut für Angewandte Mathematik des Forschungszentrums Jülich installierte massiv parallele Rechner Intel/Paragon [EK93] dient im vorliegenden Fall zur Entwicklung und Implementation der parallelen Version ParSmart. Dieser Supercomputer weist eine homogene Architektur mit verteiltem Speicher und *Message-passing*–Modell auf und wird nach der in Abschnitt 2.2.2 gegebenen Klassifizierung in die Klasse der MIMD-Rechner eingeordnet. Im folgenden werden Architektur und Software dieses Parallelrechners erläutert.

2.3.1 Architektur

Prozessoren des Intel/Paragon sind i860 XP-RISC–General-Purpose Prozessoren. Diese werden mit einer Taktfrequenz von 50 MHz betrieben und leisten eine Peak-Performance von 75 MFLOPs bei 64–bit Arithmetik [KM89]. Für ein XP/S-10 System mit 140 Knoten liegt die Peak-Performance bei 10,5 GFLOPs [Bem92], [EK93].

Die i860 Prozessoren werden mit leichten Modifikationen als Compute-, IO- und Service-Knoten eingesetzt [EK93]. Compute-Knoten führen parallele Benutzer-Applikationen aus, während IO-Knoten die Kommunikation zum Hintergrundspeicher gewährleisten. Service-Knoten unterstützen die Programmentwicklung, indem sie Compiler und Analyseprogramme bereitstellen.

Der Hauptspeicher der Rechenknoten kann je nach Konfiguration bis zu 128 MByte betragen. Der in der KFA installierte Rechner Intel/Paragon ist mit jeweils 32 MByte Hauptspeicher ausgestattet und bietet somit einen Gesamtspeicher von 4,5 GByte, hinzu kommt je Rechenknoten ein Daten- und Instruktionscache, der jeweils 16 KByte groß ist.

Das Verbindungsnetzwerk besteht aus einem zweidimensionalen Gitter, entsprechend Bild 2.3.1. Ein Rechenknoten umfaßt neben dem Prozessor i860 als Rechenprozessor

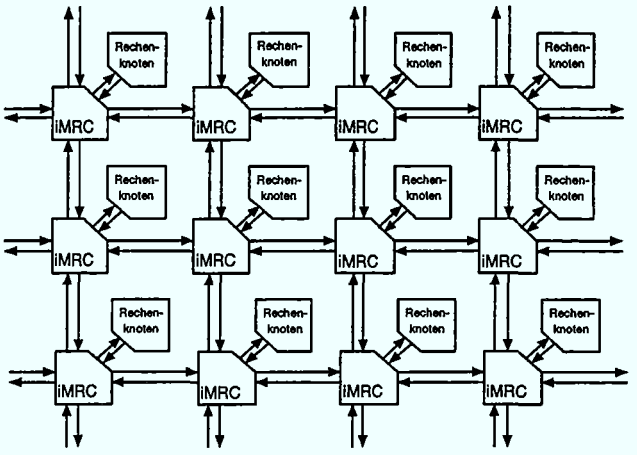


Bild 2.3.1: Verbindungsnetzwerk des Intel/Paragon, [EK93]

einen iMRC (Intel Message Routing Chip), der den Nachrichtentransfer über das Netzwerk organisiert und ebenfalls ein i860–Prozessor ist. Dieser iMRC stellt einerseits die Schnittstelle zwischen Netzwerk und Rechenknoten dar und verbindet andererseits die Rechenknoten des Gesamtsystems miteinander. Der iMRC arbeitet parallel zum Rechenknoten. Beide teilen sich den gemeinsamen Hauptspeicher sowie den Cache und kommunizieren über gemeinsame Variablen.

Für das Netzwerk wird eine Übertragungsbandbreite von 200 MByte/s bei einer Start-up Zeit von 30 μs spezifiziert. Die derzeitigen Werte liegen mit einer Bandbreite von 91 MByte/s und einer Start-up Zeit von 42 μs unter diesen angestrebten Werten [BBD95].

Der Hintergrundspeicher ist über die IO-Knoten mit dem Netzwerk verbunden. Die Konfiguration des Rechners Intel/Paragon in der KFA besteht aus 5 RAID-Systemen mit je 1,56 GByte Plattenplatz.

Die hohen Werte für die Start–up Zeit und die geringe Übertragungsbandbreite bedeuten, daß Kommunikation zwischen den Rechenknoten zeitaufwendig ist und somit möglichst minimiert werden muß. Ebenso ist der Datentransfer zum Sekundärspeicher zu minimieren, da die 5 RAID-Systeme überlastet werden, wenn 140 Rechenknoten gleichzeitig auf den Plattenspeicher zugreifen [BBD95], [ZB95].

2.3.2 Software

Das Betriebssystem des Intel/Paragon ist OSF/1 AD (Advanced Development), eine Erweiterung von OSF/1 und damit kompatibel. Es basiert auf einem Mach 3 Mikrokernel, der auf jedem Rechenknoten läuft und grundlegende System-Funktionen unterstützt. Dieser Kernel benötigt jedoch etwa 7 MByte Hauptspeicher auf jedem Rechenknoten, so daß der Benutzer nur über 25 MByte des 32 MBytes umfassenden Hauptspeichers verfügen kann. Bild 2.3.2 zeigt die Architektur des Betriebssystems, d.h. die Schnittstelle zwischen der

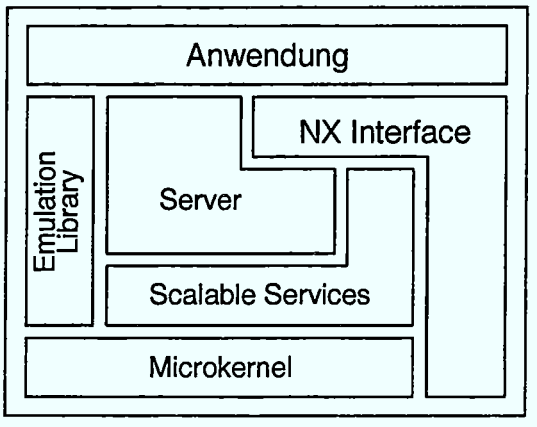


Bild 2.3.2: Architektur des Betriebssystems, [EK93]

Anwendung und dem Rechner. Sie umfaßt neben dem Microkernel das NX-*Message-passing*-Interface [Pie94], welches den Datentranfer zwischen den Rechenknoten ermöglicht. Ferner

wird dem Benutzer durch die skalierbaren Dienste, die Emulationsbibliothek und den Server der Eindruck vermittelt, daß er auf einem zusammengehörigen System arbeitet und nicht auf verteilten Rechenknoten [BBD95].

Die Kommunikation wird über das *NX-Interface* durchgeführt. Um Verklemmungen bei dieser Kommunikation im Gitter zu vermeiden, wird ein *wormhole*-Routing-Algorithmus angewendet [EK93]. Nachrichten werden zunächst in horizontaler, anschließend in vertikaler Richtung zum Zielknoten geleitet, d.h. es liegt nur eine Richtungsänderung vor. Dieser Algorithmus ist erforderlich, da die Übertragungskanäle reserviert bleiben, bis eine Nachricht ihr Ziel erreicht hat.

Für den Intel/Paragon stehen neben dem systemabhängigen, NX-basierten *Message-passing* die in Abschnitt 2.2.2 beschriebenen Programmierumgebungen MPI, PARMACS und PVM zur Verfügung. Auch das durch HPF unterstützte datenparallele Programmiermodell ist seit Anfang 1995 verfügbar. Eine Programmierumgebung zur Nutzung eines gemeinsamen virtuellen Speichers (SVM) befindet sich in der Testphase [GB95].

Dem Anwender stehen auf dem Intel/Paragon weiterhin Compiler, mathematische Bibliotheken und Produkte zur graphischen Leistungsanalyse zur Verfügung. Für die Parallelisierung des Programmsystems SMART werden der FORTRAN 77 Compiler, die Basic Linear Algebra (BLAS) Routinen sowie die Programme 'paragraph' und 'PARvis' [NA94] zur graphischen Analyse des Ablaufs der parallelen Berechnungen genutzt.

2.4 Überblick: Parallele Finite-Element-Berechnungen

Im folgenden wird ein Überblick über mögliche Parallelisierungsstrategien für Finite-Element-Programme gegeben. Bis zum Jahre 1988 wird eine Zusammenstellung der Arbeiten in dem Übersichtsartikel [WA88] gegeben und in [SDD94] erfolgt ein Überblick über den Stand des Parallelrechnereinsatzes im Bereich CFD (Computational Fluid Dynamics) aus dem Jahr 1994. In [Bus93] werden die zukünftigen Anforderungen an die Rechnerarchitekturen für die Berechnung erzwungener Konvektionsströmungen beschrieben. Dabei wird darauf hingewiesen, daß die für die Praxis wichtigen Problemstellungen nur bei Verbesserung der physikalischen Modellierung, der Algorithmen, der Programme und der Rechnerarchitekturen berechenbar sind.

Finite-Element-Rechnungen können in drei rechenzeitintensive Abschnitte unterteilt werden. Dieses sind die Berechnungen für die einzelnen Elemente, die Assemblierung der Systemmatrix aus den Elementbeiträgen sowie die anschließende Lösung der aufgestellten Gleichungssysteme [AM88]. Parallelisierungsstrategien müssen insbesondere diese drei Abschnitte berücksichtigen. In Abschnitt 2.4.1 werden dafür verschiedenartige parallele Berechnungsmöglichkeiten vorgestellt.

Bei parallelen Berechnungen ist die starke Wechselwirkung zwischen Programm- und Datenlokalität zu berücksichtigen. Eine effiziente Programmimplementation für Rechner mit verteiltem Speicher muß daher stets eine Datenaufteilung auf die Rechenknoten durchführen. Dies geschieht z.B. durch Modellzerlegungsverfahren, die die Finite-Element-Daten auf die

Rechenknoten aufteilen. Da etwa 80–90% der Rechenzeit einer Finite-Element-Rechnung zur Lösung der dünnbesetzten Gleichungssysteme benötigt werden, müssen Parallelisierungsstrategien insbesondere effiziente Lösungsverfahren berücksichtigen [Man94]. Deshalb werden alternative Konzepte zur Lösung der Gleichungssysteme in einem weiteren Abschnitt beschrieben. Abschließend erfolgt die Zusammenstellung bislang durch Parallelrechner erreichter Leistungszuwächse für ausgewählte Beispiele.

2.4.1 Alternative Parallelisierungskonzepte

Konzepte paralleler Finite-Element-Programme lassen sich entweder dem Master-Slave Konzept oder dem Konzept gleichberechtigter Knoten zuordnen.

Bei dem **Master-Slave Konzept** wird ein Prozessor des Parallelrechners als Masterknoten ausgewählt. Nur dieser speichert den gesamten ausführbaren Code. Abschnitte parallel berechenbarer Operationen werden von dem Master-Knoten erkannt und zur Verarbeitung an die Slave-Knoten übermittelt. Die Slave-Knoten empfangen die Arbeitsanweisung vom Master und führen diese aus. Nach Abschluß der Bearbeitung melden die Slave-Knoten dieses an den Master-Knoten zurück. In einer Finite-Element-Rechnung werden so beispielsweise die auf Elementebene ausführbaren Operationen an die Slave-Knoten verteilt.

Während im allgemeinen die Kommunikation zwischen Master– und Slave-Knoten überwiegt, kann die verteilte Organisation der Daten auf den Slave-Knoten Datentransfer zwischen diesen erfordern. Bei einer Finite-Element-Rechnung tritt diese Situation auf, wenn ein Rechenknoten die Anweisung erhält, ein bestimmtes Element zu bearbeiten, die erforderlichen Elementdaten jedoch nicht in seinem lokalen Speicher stehen. Eine schematische Darstellung zeigt Bild 2.4.1. Das Finite-Element-System PERMAS wird nach dieser Strategie parallelisiert [Man94].

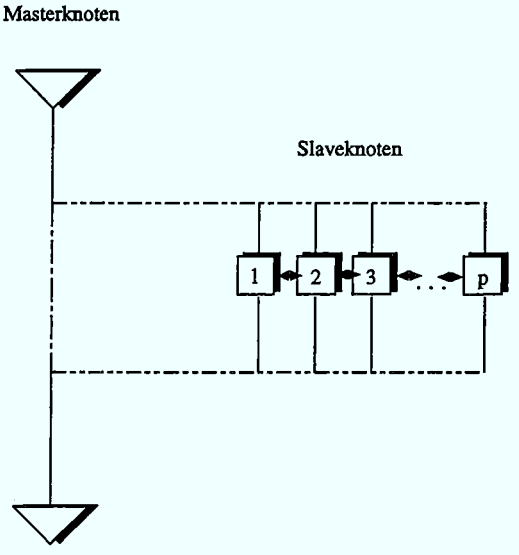


Bild 2.4.1: Master-Slave-Konzept

Dem Master-Slave Konzept steht das Konzept **gleichberechtigter Knoten** gegenüber. Dabei wird ein zu berechnendes Gesamtmodell in Teilmodelle zerlegt, die jeweils den Rechenknoten zugeordnet werden. Anschließend werden für diese Teilmodelle lokale Anweisungsfolgen ausgeführt, beispielsweise die Bestimmung der Elementmatrizen. Da die Aufgabe, das Gesamtmodell zu lösen, jedoch die Abhängigkeiten der Teilmodelle untereinander berücksichtigen muß, ist Datentransfer zwischen den Rechenknoten erforderlich. Bei Finite-Element-Berechnungen ist dieses bei der Assemblierung der Systemmatrix und der Lösung der Gleichungssysteme notwendig. Der Kommunikationsaufwand für eine parallele Berechnung

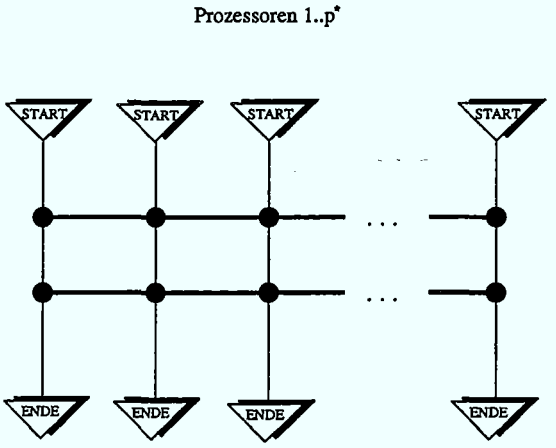


Bild 2.4.2: Konzept gleichberechtigter Knoten

auf vorgegebener Anzahl an Rechenknoten wird bei dieser Parallelisierungsstrategie folglich von der Art der Zerlegung des Gesamtproblems in Teilprobleme bestimmt.

Da die Architektur der betrachteten Parallelrechner durch Prozessorleistung und Kommunikationszeiten den Leistungszuwachs einer parallelen Berechnung bestimmt, muß sie bei der Wahl einer Parallelisierungsstrategie berücksichtigt werden [Far90]. Bei der Parallelisierung bestehender Programme wird die Strategie zusätzlich durch die bereits vorhandenen Programm- und Datenstrukturen beeinflußt, da diese Strukturen oftmals nur geringfügig modifiziert werden dürfen.

Während bei dem Konzept gleichberechtigter Knoten jeder Rechenknoten einen kompletten Programmlauf auf einem Teilmodell ausführt und somit den gesamten Code speichern muß, wird bei dem Master-Slave Konzept eine bessere Nutzung des lokalen Speicherplatzes gewährleistet, da nur diejenigen Programmabschnitte, die von Slave-Knoten ausgeführt werden können, lokal gespeichert werden müssen. Das Master-Slave Konzept bietet zusätzlich die Vorteile einer flexiblen Datenverwaltung, da der Masterknoten die Prozeß- und Datenverteilung während der Laufzeit vornimmt. Demgegenüber steht die feste Datenverteilung bei einem Konzept gleichberechtigter Knoten, bei dem die Teilmodelle bereits zu Beginn einer parallelen Berechnung aufgestellt und den Rechenknoten zugewiesen werden. Allgemein weist das Master-Slave-Konzept eine flexiblere Programm- und Datenstruktur auf, was jedoch mit einem nichtdeterministischen Programmablauf erkauft wird. Insbesondere die Fehleranalyse und Korrektur wird in diesem Fall schwierig und muß besonders berücksichtigt werden.

Demgegenüber steht das deterministische Konzept gleichberechtigter Knoten, welches eine leichte Fehleranalyse und Korrektur ermöglicht, da Datenverteilung und Kommunikationsschema bei wiederholten Programmläufen für ein Problem gleich sind.

Beispiele aus der Literatur für die Parallelisierung von Finite-Element-Rechnungen konzentrieren sich weitgehend auf parallele Lösungsverfahren für die aufgestellten Gleichungssysteme [HRN+94].

In [JH92] werden für Finite-Element-Berechnungen auf Parallelrechnern eine datenparallele Strategie, die dem Konzept gleichberechtigter Knoten entspricht, und zugehörige Gleichungslösungsverfahren diskutiert. Auch in [BJK+93] wird ein datenparalleles Konzept zur parallelen Berechnung kompressibler Strömungen für verschiedenartige Finite-Element-Formulierungen betrachtet.

Ein Konzept gleichberechtigter Knoten wird ebenfalls in [Mal88] untersucht, dabei werden verschiedenartige Verfahren zur Aufteilung der Daten auf die Rechenknoten gegenübergestellt.

Die parallele Implementation von Finite-Element-Rechnungen auf einem Parallelrechner mit Hypercube Netztopologie nach dem Konzept gleichberechtigter Knoten wird in [WC89] beschrieben.

Ein Master-Slave Konzept wird in [YYH93] betrachtet und für die Parallelisierung des Produktionscodes PERMAS [Man94] zugrundegelegt.

Das Konzept für eine parallele Datenbank zur Organisation der bei Finite-Volumen- und Finite-Element-Rechnungen zu verwaltenden Daten wird in [EAK+94] angegeben.

Weitere Beispiele für Parallelisierungsstrategien werden für spezielle Problemstellungen in [BE89], [BDF94], [DHRF94], [MT94] und [TAB+93] diskutiert.

2.4.2 Modellzerlegungsverfahren

Bei Parallelrechnern mit verteiltem Speicher ist es wichtig, die Daten eines Programmlaufs gleichmäßig auf die Rechenknoten zu verteilen, um zum einen den verfügbaren Speicher optimal zu nutzen und zum anderen Operationen auf lokalen Daten zu ermöglichen. Benötigt ein Rechenknoten Daten, die nicht in seinem lokalen Speicher stehen, so ist Kommunikation mit denjenigen Rechenknoten erforderlich, auf denen sich die benötigten Daten befinden. Die in 2.2.2 beschriebenen Kommunikationsbibliotheken ermöglichen diesen Datenaustausch. Aus den Leistungsdaten des in Abschnitt 2.3 vorgestellten Parallelrechners Intel/Paragon wird jedoch deutlich, daß die hohen Kommunikationszeiten erheblichen Einfluß auf die Gesamtrechenzeit einer Anwendung ausüben können. Für parallele Algorithmen ist folglich die Datenlokalität entscheidend für den Leistungszuwachs, da sie den Kommunikationsaufwand bestimmt. Die Datenaufteilung auf die parallel arbeitenden Rechenknoten wird für Finite-Element-Modelle durch Modellzerlegungsverfahren durchgeführt. Die Anforderungen, die an ein solches Verfahren gestellt werden, sind nach [Far88]:

- Anwendbarkeit für beliebige zwei- und dreidimensionale Geometrien und Diskretisierungen,

- gleichhohe Rechenzeit und Speicherplatzkomplexität der Teilmodelle für jeden Rechenknoten,
- minimale Anzahl der inneren Randpunkte, d.h. Teilmodelle miteinander verbindender Finite-Element-Knoten.

Bei dieser Aufgabe handelt es sich um ein *NP*–hartes Problem, d.h. es ist nur mit nichtdeterministischen Algorithmen in polynomieller Rechenzeit lösbar [FLS95]. In der Praxis werden deshalb meist Verfahren eingesetzt, die die genannten Kriterien nur näherungsweise erfüllen.

In [Mal88] wird ein Algorithmus beschrieben, der auf einer Bandbreitenminimierung der Systemmatrix einer Finite-Element-Rechnung nach dem *Reverse Cuthill McKee*–Verfahren basiert. Die Matrixzeilen werden dabei zeilenweise auf die Rechenknoten verteilt, und es wird gezeigt, daß für eine kleine Bandbreite die rechenknotenübergreifende Kommunikation geringer ist als bei einer größeren Bandbreite.

Ein anderes Verfahren wird in [Far88] als *Greedy* Algorithmus beschrieben. Dabei wird für jeden Rechenknoten schrittweise eine Menge möglichst benachbarter Elemente bestimmt, bis für jeden Rechenknoten gleich viele Elemente zusammengefaßt worden sind.

Weitere Modellzerlegungsverfahren basieren auf Methoden des Simulated Annealing und der Graphentheorie [Wil94]. Mit TOP/DOMDEC [FLS95] ist eine Programmbibliothek verfügbar, die verschiedene Modellzerlegungsverfahren anbietet.

Für die Parallelversion ParSmart wird der in Abschnitt 5.1.1 näher erläuterte *Greedy*–Algorithmus verwendet, der in relativ kurzen Rechenzeiten eine befriedigende Lösung bietet. Bei Bedarf kann dieser durch Algorithmen ergänzt werden, die ausgehend von der bestimmten Zerlegung weitere Verbesserungen vornehmen.

2.4.3 Parallele Lösungsverfahren für Gleichungssysteme

Etwa 80–90% einer Finite-Element-Rechnung werden zur Lösung der aufgestellten, dünnbesetzten Gleichungssysteme benötigt [Man94], [INS95]. Ein paralleles Programm muß folglich insbesondere für diesen Teil der Berechnung effiziente Algorithmen ausführen. Eine umfassende Übersicht über parallele direkte oder iterative Lösungsverfahren wird in [OV85] gegeben.

In seriellen Produktionsprogrammen für Finite-Element-Berechnungen wie SMART oder PERMAS wird oft das Cholesky-Verfahren zur Lösung der Gleichungssysteme $Ax = b$ mit dünnbesetzter und symmetrischer, positiv definiter Koeffizientenmatrix eingesetzt, da es in der Praxis gute Stabilität gewährleistet. Werden verschiedene Lastfälle betrachtet, d.h. sind Systeme $Ax = b_i, i = 1, ..., m$ zu lösen, so muß der aufwendigste Teil des Verfahrens, die Triangularisierung, nur einmal durchgeführt werden. Theoretische Untersuchungen zur Parallelisierung des Cholesky-Verfahrens wurden in [Stp92] und bereits in [GHL85] und [GKF89] durchgeführt.

Auf Parallelrechnern werden iterative Verfahren bevorzugt [OV85], da sie weniger Datenabhängigkeiten und somit einen höheren Anteil an parallel ausführbaren Operationen aufweisen. Unter den iterativen Verfahren kommt insbesondere dem konjugierten Gradienten-Verfahren besonderes Interesse zu [AÖS90], [Bas95], [Chi92], da das seriell von Hestenes und Stiefel [HS52] vorgeschlagene Verfahren für Gleichungssysteme der Dimension n theoretisch nach maximal n Iterationsschritten konvergiert.

In [HRN+94] werden Methoden für die parallele Berechnung der Navier-Stokes-Gleichungen durch Finite-Elemente angegeben. Dabei werden vorrangig Verfahren zur Lösung der aufgestellten Gleichungssysteme diskutiert und für moderat parallele Rechnersysteme direkte Verfahren bevorzugt.

Für ParSmart wird ein iteratives konjugiertes Gradienten-Verfahren eingesetzt, da dieses eine gute Leistungssteigerung durch den Parallelrechnereinsatz zeigt und bereits erfolgreich zur Lösung von Gleichungssystemen aus Finite-Element-Berechnungen eingesetzt wurde [Bas95].

2.4.4 Leistungszuwachs paralleler Berechnungen

Bei der Bewertung paralleler Rechencodes ist grundsätzlich zu unterscheiden, ob ein existierender Code parallelisiert oder ein neues System für einen Parallelrechner entwickelt werden soll.

Ziele der **Parallelisierung bestehender Produktionscodes** sind es, die Rechenzeiten bekannter Anwendungen zu verkürzen und größere Anwendungen, die beispielsweise durch mehr Freiheitsgrade charakterisiert werden, zu ermöglichen. Dabei sollen die bei der Codeentwicklung gewonnenen Erfahrungen genutzt und die Verifikationszeiten durch weitgehende Übernahme von Algorithmen und Datenstrukturen verkürzt werden. Während der langjährigen Entwicklungszeiten solcher Produktionsprogramme sind Algorithmen und Datenstrukturen jedoch meist für die vorhandenen seriellen Rechner konzipiert und optimiert worden und nicht ohne Änderungen oder Leistungseinbußen auf parallele Architekturen übertragbar. Der Verkürzung der Implementations- und Verifikationszeiten bei der Parallelisierung bestehender Programmsysteme stehen folglich Leistungseinbußen durch nicht optimale Programm- und Datenstrukturen gegenüber. Inwieweit diese gegenüber einer Neuentwicklung vertretbar sind, hängt von der Komplexität der Programme ab. Produktionscodes wie PERMAS und MSC/NASTRAN mit etwa 1,5 Mio. Anweisungen werden nicht in vertretbarer Zeit durch parallele Neuentwicklungen zu ersetzen sein. Im EUROPORT I Projekt werden daher u.a. die Systeme LS-DYNA3D, MSC/NASTRAN, PERMAS aus dem Bereich der Strukturmechanik sowie STAR-CD aus dem Bereich der Fluiddynamik parallelisiert [CLMT94].

Ziele der **Neuimplementation** paralleler Finite-Element-Berechnungen betreffen meist die Entwicklung optimaler Programm- und Datenstrukturen bzw. die Entwicklung oder Verwendung geeigneter paralleler Algorithmen. Diese Programmsysteme orientieren sich an der Architektur des betrachteten Parallelrechners und erlauben eine bessere Nutzung der

verfügbaren Leistung. Nachteilig ist oft die Spezialisierung dieser Codes auf bestimmte Problemstellungen.

Autor	Algorithmen	Rechner	Unbekannte	Anzahl Rechenknoten	Speedup	Effizienz	Literatur
Basermann	konjugiertes Gradienten-Verfahren mit Vorkonditionierung	Paragon	25222	140	83	0,59	[Bas95]
Crone	konjugiertes Gradienten-Verfahren mit Vorkonditionierung	Parsytec (Transputer)	16384	64	25,38	0,39	[Cro94]
			692224	64	56,58	0,88	[Cro94]

Tabelle 2: Leistungszuwachs für parallele Berechnungen

In Tabelle 2 werden Werte für Speedup und Effizienz für die Lösung von Gleichungssystemen durch parallele konjugierte Gradienten-Verfahren angegeben. Als Problemgröße wird hier die Dimension des gelösten Gleichungssystems betrachtet. Eine Berechnung auf Parallelrechnern wird jedoch nicht nur von dieser Dimension beeinflußt, sondern auch von dem Besetzungsgrad der Matrix, d.h. dem Verhältnis von Nichtnull- zu Nulleinträgen. Dieses beeinflußt wiederum die Anzahl der lokal ausführbaren Operationen und die erforderliche Kommunikation. In den vorangehend genannten Fällen handelt es sich um die Lösung von Gleichungssystemen mit dünnbesetzter Koeffizientenmatrix. Dabei wird inbesondere in [Cro94] deutlich, daß die Effizienz des Parallelrechnereinsatzes von der Dimension des gelösten Gleichungssystems abhängt.

In [ZB95] wird das von Intel für den Rechner Intel/Paragon angebotene Gleichungslösungsverfahren untersucht. Dabei werden Gleichungssysteme mit dichtbesetzter Koeffizientenmatrix durch eine LR-Zerlegung gelöst. Für eine Matrixdimension von 2500 wird bei Einsatz von bis zu 64 Rechenknoten ein maximaler Speedup von etwa 9 erreicht.

Für die Parallelisierung des Finite-Element-Codes MSC/NASTRAN werden in [INS95] Werte für den bisher erreichten Speedup angegeben. Dabei wird die Lösung eines Gleichungssystems mit 299604 Freiheitsgraden durch ein paralleles konjugiertes Gradienten-Verfahren bewertet. Auf 8 Rechenknoten des Rechners IBM SP 1 ergibt sich eine Verkürzung der Rechenzeiten um etwa einen Faktor 6.

3 Programmsystem SMART

Das Finite-Element System SMART ermöglicht unter anderem die Berechnung instationärer Wärmeübertragungsprobleme durch Wärmeleitung, Konvektion und Strahlung [LS91] sowie von inkompressiblen Strömungen [Szi89]. Das Programm ist eine Erweiterung von ASKA, einem an der Universität Stuttgart entwickelten General-Purpose Finite-Element-System [AFR+73], [AWW77]. Daten- und Programmstruktur stimmen mit ASKA, aber auch weitgehend mit PERMAS (Version 4) [INT87], einem kommerziell vertriebenen System, das ebenfalls aus ASKA hervorging, überein. SMART ist ein FORTRAN-Code und umfaßt etwa 300 000 Anweisungen, die in 4000 Routinen und Funktionen strukturiert sind.

Die in der seriellen SMART-Version angewendeten mathematischen Modelle und Algorithmen werden im folgenden zusammengefaßt. Eine Beschreibung der Anwenderebene gibt einen kurzen Einblick in die Problemformulierung und die breite Anwendbarkeit dieses Systems. Beispielsweise können Probleme mit nichtlinearem Materialverhalten und zeitabhängigen Randbedingungen berechnet werden. Die anschließende Beschreibung der Systemebene stellt die wichtigsten Grundlagen über Programm- und Datenaufbau, die für die Parallelisierung erforderlich sind, zusammen.

3.1 Mathematisches Modell — Algorithmen

Eine mathematische Beschreibung thermisch gekoppelter Strömungsvorgänge erfolgt durch die Kontinuitäts- und die Wärmebilanzgleichung sowie die Navier-Stokes-Gleichungen [Mer87]. Die Geschwindigkeit $\mathbf{v}$, der Druck p und die Temperatur T sind die lokalen Unbekannten des Systems. In diesem Abschnitt werden die durch die Boussinesq-Oberbeck Approximation vereinfachten Grundgleichungen beschrieben, in denen alle Stoffwerte je Zeitschritt mit Ausnahme der Dichte im Auftriebsterm als konstant betrachtet werden [Mer87]. Anschließend erfolgt die Beschreibung der räumlichen Diskretisierung der Navier-Stokes- und der Wärmebilanzgleichung durch Finite-Elemente, um die unbekannten Geschwindigkeiten und Temperaturen zu bestimmen [Szi89], [LS91]. Der Druck wird durch ein Penalty-Verfahren eliminiert [Wüs86], so daß dieser nicht aus dem System der Grundgleichungen ermittelt werden muß. Die Gewichtsfunktionen in der Finite-Element-Formulierung werden im konvektiven Term und im Auftriebsglied der Navier-Stokes Gleichung sowie im konvektiven Term der Wärmebilanzgleichung entsprechend einem Streamline-Upwind/Petrov-Galerkin-Verfahren gewählt [ALS92], [BH82]. Die verfügbaren Zeitintegrationsverfahren werden in Abschnitt 3.1.3 erläutert. Nach der Beschreibung der Lösung der aufgestellten Gleichungssysteme mit dem Cholesky-Verfahren wird abschließend die schematische Darstellung einer SMART-Konvektion Rechnung angegeben.

3.1.1 Grundgleichungen

Für inkompressible Fluide lauten die Grundgleichungen zur Beschreibung von Konvektionsströmungen in der Eulerschen Betrachtungsweise [LS91]:
Kontinuitätsgleichung:

$$\nabla^t(\mathbf{v}) = 0\,, \tag{3.1.1}$$

Navier-Stokes-Gleichungen:

$$\rho\frac{\partial \mathbf{v}}{\partial t} + \rho\big(\mathbf{v}\nabla^t\big)\mathbf{v} = \mathbf{R_a} + \rho\mathbf{X} - \nabla p + \mu\Delta\mathbf{v} \tag{3.1.2}$$

und Wärmebilanzgleichung:

$$\rho\frac{\partial}{\partial t}(c_T T) + \rho(\nabla(c_T T))^t\mathbf{v} = \nabla^t(k_T(\nabla T)) + Q_D + Q. \tag{3.1.3}$$

Die lokalen Unbekannten sind die Geschwindigkeit $\mathbf{v}$, der Druck p und die Temperatur T. Die Dichte wird mit ρ bezeichnet und zunächst in allen Termen als konstant angenommen, $\mathbf{R_a}$ mit

$$\mathbf{R_a} = \{R_x, R_y, R_z\} \tag{3.1.4}$$

ist der äußere Lastvektor und $\mathbf{X}$

$$\mathbf{X} = \{g_x, g_y, g_z\} \tag{3.1.5}$$

der Vektor der Erdbeschleunigung [LS91]. Ferner bezeichnet μ die dynamische Viskosität, c_T die spezifische Wärmekapazität, k_T die Wärmeleitfähigkeit, Q_D die Energiedissipationsrate und Q die volumenspezifische Leistung [LS91].

Die Boussinesq-Oberbeck Approximation vereinfacht die Grundgleichungen, indem die Dichte in allen Termen mit Ausnahme des Auftriebsglieds als konstant angenommen wird. Die approximierten Grundgleichungen lauten dann [LS91], [Mer87]:

Kontinuitätsgleichung:

$$\nabla^t(\mathbf{v}) = 0, \tag{3.1.6}$$

Navier-Stokes-Gleichungen:

$$\begin{aligned}\frac{\partial \mathbf{v}}{\partial t} + \big(\mathbf{v}\nabla^t\big)\mathbf{v} &= \frac{\mathbf{R_a}}{\rho_0} + \left(1 + \frac{\delta\rho}{\rho_0}\right)\mathbf{X} + \frac{1}{\rho_0}\big(\nabla^t\sigma\big)^t \\ &= \frac{\mathbf{R_a}}{\rho_0} + [1 - \alpha_V(T - T_0)]\mathbf{X} - \frac{1}{\rho_0}\nabla p + \nu_0\Delta\mathbf{v},\end{aligned} \tag{3.1.7}$$

Wärmebilanzgleichung:

$$\frac{\partial T}{\partial t} + (\nabla T)^t\mathbf{v} = a\Delta T + Q. \tag{3.1.8}$$

Die lineare Approximation der Dichte in (3.1.7) folgt aus dem isobaren Volumenausdehnungskoeffizienten α_V nach Integration und Taylorentwicklung [Wüs86]. T_0, ρ_0 und ν_0 bezeichnen Bezugswerte für die Temperatur, Dichte und kinematische Viskosität. Für kleine Temperaturunterschiede $\Delta T < 10K$ ergibt sich mit ausreichender Genauigkeit

$$\rho = \rho_0[1 - \alpha_V(T - T_0)] \tag{3.1.9}$$

mit $\rho_0 = \rho(T)$. Die Wärmebilanzgleichung wird durch $a = \frac{k_T}{\rho_0 c_T}$ sowie die Annahme konstanter Stoffwerte vereinfacht. Ferner wird die Energiedissipation Q_D vernachlässigt.

3.1.2 Räumliche Diskretisierung

Nach Abschnitt 3.1.1 sind die Navier-Stokes-Gleichungen mit den dort vereinbarten Bezeichnungen entsprechend

$$\rho \frac{\partial \mathbf{v}}{\partial t} + \rho(\mathbf{v}\nabla^t)\mathbf{v} = \mathbf{R_a} + \rho\mathbf{X} + \nabla\phi \tag{3.1.10}$$

mit $\nabla\phi = -\nabla p + \mu\Delta\mathbf{v}$ zu schreiben. $\nabla\phi$ bezeichnet den Cauchy'schen Spannungstensor, der für Newtonsche Fluide symmetrisch ist [Szi89].

Zur Aufstellung der räumlich diskretisierten Gleichungen werden die äußeren Lasten $\mathbf{R_a}$ zunächst vernachlässigt. Nach der Methode der gewichteten Residuen lautet die integrale Form von (3.1.10):

$$\int_V (\delta\mathbf{v})^t \rho \frac{\partial \mathbf{v}}{\partial t} dV + \int_V (\delta\mathbf{v})^t \rho(\mathbf{v}\nabla^t)\mathbf{v}\, dV = \int_V (\delta\mathbf{v})^t \rho\mathbf{X} dV + \int_V (\delta\mathbf{v})^t (\nabla^t\phi)^t dV. \tag{3.1.11}$$

In 3.1.11 ist $\delta\mathbf{v}$ eine allgemeine Gewichtsfunktion im Sinne einer virtuellen Geschwindigkeit [Szi89].

Indem der Druck durch ein Penalty-Verfahren aus dem Spannungsvektor σ eliminiert wird, wird die Lösung darauf reduziert, die Geschwindigkeiten zu bestimmen. Die Wahl des Penalty-Parameters $\tilde{\kappa}$ wird in [Wüs86] diskutiert.

Die räumliche Diskretisierung der Navier-Stokes-Gleichungen durch Finite Elemente erfolgt nun durch den Galerkin-Ansatz für das Geschwindigkeitsfeld $\mathbf{v}$

$$\mathbf{v}(x,t) = w_V(x)\mathbf{v_e}(t), \tag{3.1.12}$$

für die virtuelle Geschwindigkeit $\delta\mathbf{v}$ in den viskosen Termen und im Trägheitsglied durch

$$\delta\mathbf{v}(x,t) = w_V(x)\delta\mathbf{v_e}(t) \tag{3.1.13}$$

sowie für die virtuelle Geschwindigkeit $\delta\mathbf{v}$ im konvektiven Term und im Auftriebsglied mittels

$$\begin{aligned} \delta\mathbf{v}(x,t) &= \tilde{w}_V(x)\delta\mathbf{v_e}(t) \\ &= (w_V(x) + UPWD\vartheta_V)\delta\mathbf{v_e}(t) \end{aligned} \tag{3.1.14}$$

wobei ein Petrov-Galerkin Ansatz gewählt wird und $UPWD$ den *Upwind*–Faktor bezeichnet [Szi89]. Die w_V bzw. $\tilde{w}_V$ sind Gewichtsfunktionen und $\mathbf{v_e}(t)$ entspricht der Geschwindigkeit in den Knoten des Elementes e zum Zeitpunkt t.

Mit der Element-Massenmatrix

$$\mathbf{m} = \int_V \rho\, w_V^t\, w_V\, dV, \tag{3.1.15}$$

der Element-Konvektionsmatrix

$$\mathbf{n} = \int_V \rho\, \tilde{w}_V^t \left(w_V \mathbf{v_e} \nabla^t\right) w_V\, dV, \tag{3.1.16}$$

dem hydrostatischen Anteil der Element-Viskositätsmatrix

$$\mathbf{f_H} = \int_V (Dw_V)^t\, 2\mu\tilde{\kappa}\, \mathbf{E_{33}} \left(D^t w_V\right) dV, \tag{3.1.17}$$

dem deviatorischen Anteil der Element-Viskositätsmatrix

$$\mathbf{f_D} = \int_V (Dw_V)^t\, 2\mu \left[\mathbf{I_6} - \frac{1}{3}\mathbf{E_{33}}\right] \left(D^t w_V\right) dV \tag{3.1.18}$$

und dem Element-Lastvektor

$$\mathbf{r} = \int_V \rho \tilde{w}_V^t\, \mathbf{X}\, dV + \int_F w_V^t \mathbf{f_n}\, dF \tag{3.1.19}$$

lautet (3.1.11) auf Elementebene:

$$\mathbf{m}\,\dot{\mathbf{v}} + \mathbf{nv} = -\mathbf{fv} + \mathbf{r} \tag{3.1.20}$$

mit $\mathbf{f} = \mathbf{f_H} + \mathbf{f_D}$. In (3.1.17) und (3.1.18) bezeichnet $\mathbf{E_{33}}$= $\{111000\} \cdot \{111000\}^T$, D den Differentialoperator

$$D = \begin{bmatrix} \frac{\partial}{\partial x} & 0 & 0 \\ 0 & \frac{\partial}{\partial y} & 0 \\ 0 & 0 & \frac{\partial}{\partial z} \\ \frac{1}{2}\sqrt{2}\frac{\partial}{\partial y} & \frac{1}{2}\sqrt{2}\frac{\partial}{\partial x} & 0 \\ 0 & \frac{1}{2}\sqrt{2}\frac{\partial}{\partial z} & \frac{1}{2}\sqrt{2}\frac{\partial}{\partial y} \\ \frac{1}{2}\sqrt{2}\frac{\partial}{\partial z} & 0 & \frac{1}{2}\sqrt{2}\frac{\partial}{\partial x} \end{bmatrix} \tag{3.1.21}$$

und $\mathbf{I_6} = \underline{1}$ den Vektor $\{111111\}$.

Nach Übergang auf Strukturebene durch Assemblierung der Elementbeiträge ergibt sich [Szi89]:

$$\mathbf{M}\dot{\mathbf{V}} + (\mathbf{N}+\mathbf{F})\mathbf{V} = \mathbf{R}. \tag{3.1.22}$$

Die Matrizen $\mathbf{M}, \mathbf{N}, \mathbf{F}$, sowie der Vektor $\mathbf{R}$ ergeben sich durch Assemblierung der Elementbeiträge $\mathbf{m}, \mathbf{n}, \mathbf{f}$ und $\mathbf{r}$. In (3.1.22) bleiben die in Kapitel (3.1.1) vereinbarten Stoffwertdefinitionen zu berücksichtigen.

Nach Abschnitt 3.1.1 lautet die Wärmebilanzgleichung für ein inkompressibles Medium mit den dort eingeführten Bezeichnungen:

$$\rho c_T \frac{\partial}{\partial t}(T) + \rho c_T (\nabla(T))^t \mathbf{v} = k_T \nabla^t (\nabla T) + Q, \tag{3.1.23}$$

wenn der dissipative Term vernachlässigt wird, $Q_D = 0$.

Die integrale Form von (3.1.23) lautet

$$\begin{aligned} &\int_V \delta T\, \rho\, c_T \frac{\partial T}{\partial t}\, dV + \int_V \delta T \rho c_T\, \mathbf{v}^t (D_T T)\; dV \\ &= \int_V \delta T D_T^t (\mathbf{k}_T (D_T T)) dV + \int_V \delta T Q_T dV + \int_F \delta T Q_n dF \end{aligned} \tag{3.1.24}$$

nachdem die Gewichtsfunktion δT im Sinne eines virtuellen Temperaturfeldes eingeführt worden ist. D_T entspricht dem Differentialoperator $D_T = \{\frac{\partial}{\partial x} \frac{\partial}{\partial y} \frac{\partial}{\partial z}\}$, $Q_n = \alpha(T - T_\infty)$ ist der Wärmeübergang an der Oberfläche in ein umgebendes Medium mit der Temperatur T_∞ und α der Wärmeübergangskoeffizient.

Zur Diskretisierung wird nun zunächst das Temperaturfeld approximiert:

$$T(x,t) = w_T(x)\, T_e(t) \tag{3.1.25}$$

wobei der Vektor T_e die Temperaturen an den Knotenpunkten des Elementes e kennzeichnet [LS91].

Das virtuelle Temperaturfeld in den kapazitativen, konduktiven und Quelltermen wird durch einen Galerkin-Ansatz [LS91]

$$\delta T(x,t) = w_T(x) \delta T_e(t) \tag{3.1.26}$$

beschrieben, während der konvektive Term durch einen Petrov-Galerkin Ansatz formuliert wird [LS91]:

$$\begin{aligned} \delta T &= \tilde{w}_T\, \delta T_e(t) \\ &= (w_T + UPWD\; \vartheta_T) \delta T_e(t) \end{aligned} \tag{3.1.27}$$

Für das Geschwindigkeitsfeld gilt der Ansatz

$$\mathbf{v}(x,t) = w_V(x)\mathbf{v}_e(t) \tag{3.1.28}$$

Zusammenfassend folgt aus 3.1.23 auf Elementebene [LS91]:

$$\mathbf{c}\dot{\mathbf{T}} + (\mathbf{n}+\mathbf{k})\mathbf{T} = \mathbf{q} \tag{3.1.29}$$

mit der Element-Kapazitätsmatrix

$$\mathbf{c} = \int_V \rho\, c_T\, w_T^t\, w_T\, dV, \tag{3.1.30}$$

Element-Konvektionsmatrix

$$\mathbf{n} = \int_V \rho c_T\, \tilde{w}_V^t\, (w_V \mathbf{v_e})^t\, (D_T w_T)\, dV, \tag{3.1.31}$$

Element-Konduktivitätsmatrix

$$\begin{aligned}\mathbf{k} = &\int_V (D_T w_T) k_T (D_T w_T) dV \\ &- \int_V w_T^t w_T Q_1 dV + \int_F \alpha\, w_T^t w_T dF,\end{aligned} \tag{3.1.32}$$

Element-Lastvektor

$$\mathbf{q} = \int_V w_T^t\, Q_0\, dV + \int_F \alpha\, w_T^t\, T_\infty dF \tag{3.1.33}$$

und analog auf Strukturebene [LS91]:

$$\mathbf{C}\dot{\mathbf{T}} + (\mathbf{N}+\mathbf{K})\mathbf{T} = \mathbf{Q}. \tag{3.1.34}$$

Dabei bezeichnen $\mathbf{C}, \mathbf{N}$ und $\mathbf{K}$ die Matrizen auf Systemebene, die sich durch Assemblierung der Elementmatrizen $\mathbf{c}, \mathbf{n}$ und $\mathbf{k}$ ergeben sowie $\mathbf{Q}$ die assemblierte Rechthandseite aus den Elementbeiträgen $\mathbf{q}$.

3.1.3 Zeitliche Diskretisierung

Die Berechnung instationärer Probleme erfolgt in SMART durch das allgemeine ζ–Verfahren, welches die Geschwindigkeit $\mathbf{v}$ und die Temperaturen T durch einen linearen Ansatz im Zeitablauf diskretisiert [AFR+73]. Im folgenden werden die verfügbaren Zeitintegrationsverfahren erläutert und die zeitliche Diskretisierung der Navier-Stokes– und der Wärmebilanzgleichung angegeben.

Die Geschwindigkeit v aus Gleichung (3.1.20) werden im Integrationspunkt $\mathbf{v}^\zeta$ durch

$$\mathbf{v}^\zeta = \frac{(1+\zeta)}{2}\mathbf{v}^{\mathbf{n+1}} + \frac{(1-\zeta)}{2}\mathbf{v}^{\mathbf{n}} \tag{3.1.35}$$

linear approximiert, wobei $\mathbf{v}^{\mathbf{n}}$ die bekannte Geschwindigkeit zu Beginn des Zeitschritts $\Delta t = t^{n+1} - t^n$ und $\mathbf{v}^{\mathbf{n+1}}$ die unbekannte Geschwindigkeit am Ende des Zeitschritts bezeichnet. Für die zeitliche Ableitung $\dot{\mathbf{v}}^\zeta$ folgt

$$\dot{\mathbf{v}}^\zeta = \frac{1}{\Delta t}\left(\mathbf{v}^{\mathbf{n+1}} - \mathbf{v}^{\mathbf{n}}\right). \tag{3.1.36}$$

Aus diesem allgemeinen $\zeta-$Verfahren lassen sich die folgenden, bekannten Zeitintegrationsverfahren ableiten [AWW77]:

$\zeta = -1$	Vorwärtsdifferenzenverfahren
$\zeta = 0$	Trapezverfahren
$\zeta = \frac{1}{3}$	Galerkin-Verfahren
$\zeta = 1$	Rückwärtsdifferenzenverfahren

Bei der Wahl des Integrationspunktes ζ sind neben Stabilitätseigenschaften der Einfluß von Rundungsfehlern und das Auftreten von Oszillationen zu berücksichtigen. Ausführliche Diskussionen über die Eigenschaften der einzelnen Verfahren wurden in der Finite-Differenzen-Literatur geführt; zusammenfassend wird eine Analyse für die in SMART bereitgestellten Verfahren in [AWW77] gegeben.

Im allgemeinen liefert das Vorwärtsdifferenzenverfahren bei gleichem Zeitaufwand den größten Fehler und ist nur bedingt stabil. In der Praxis hat sich das Rückwärtsdifferenzenverfahren bewährt. Es bedingt zwar größere numerische Fehler als Galerkin- oder Trapezverfahren, verhält sich jedoch stabil und liefert oszillationsfreie Lösungen. Werden die Ansätze (3.1.35) und (3.1.36) in (3.1.20) eingesetzt, ergibt sich folgende Iterationsvorschrift auf Elementebene

$$\left[\frac{1}{\Delta t}\mathbf{m}^\zeta + \frac{1+\zeta}{2}\left(\mathbf{n}^\zeta + \mathbf{f}^\zeta\right)\right]\mathbf{v}_{\mathbf{i+1}}^{\mathbf{n+1}} = \mathbf{r}^\zeta - \frac{1}{\Delta t}\mathbf{m}^\zeta\mathbf{v}^{\mathbf{n}} - \left[\frac{1-\zeta}{2}\left(\mathbf{n}^\zeta + \mathbf{f}^\zeta\right)\right]\mathbf{v}^{\mathbf{n}} \tag{3.1.37}$$

bei der das Superskript ζ die Abhängigkeit der Massenmatrix $\mathbf{m}^\zeta$, der Konvektionsmatrix $\mathbf{n}^\zeta$ der Viskositätsmatrix $\mathbf{f}^\zeta$ und des Lastvektors $\mathbf{r}^\zeta$ vom Integrationspunkt ζ bezeichnet. Der Index $i, i = 1, \ldots$ kennzeichnet den Iterationsschritt.

Die inkrementelle Vorgehensweise ermöglicht es, die Geschwindigkeitsänderung

$$\Delta\mathbf{v}_{\mathbf{i+1}}^{\mathbf{n+1}} = \mathbf{v}_{\mathbf{i+1}}^{\mathbf{n+1}} - \mathbf{v}_{\mathbf{i}}^{\mathbf{n+1}} \tag{3.1.38}$$

je Iterationsschritt direkt auf Konvergenz zu untersuchen [Szi89].

Weiterhin wird (3.1.37) vereinfacht, indem der asymmetrische Term $\frac{1+\zeta}{2}\mathbf{n}^{\zeta}\mathbf{v}_{i+1}^{n+1}$ auf der Linkhandseite eliminiert und durch $\frac{1+\zeta}{2}\mathbf{n}^{\zeta}\mathbf{v}_{i}^{n+1}$ auf der Rechthandseite berücksichtigt wird [Szi89].

Mit

$$\mathbf{v}_{i}^{\zeta} = \frac{1+\zeta}{2}\mathbf{v}_{i}^{n+1} + \frac{1-\zeta}{2}\mathbf{v}^{n} \tag{3.1.39}$$

ergibt sich das durch

$$\left[\frac{1}{\Delta t}\mathbf{m}^{\zeta} + \frac{1+\zeta}{2}\mathbf{f}^{\zeta}\right]\left(\mathbf{v}_{i+1}^{n+1} - \mathbf{v}_{i}^{n+1}\right) = \mathbf{r}^{\zeta} - \frac{1}{\Delta t}\mathbf{m}^{\zeta}\left(\mathbf{v}_{i}^{n+1} - \mathbf{v}^{n}\right) - \mathbf{n}^{\zeta}\mathbf{v}_{i}^{\zeta} - \mathbf{f}\mathbf{v}_{i}^{\zeta} \tag{3.1.40}$$

formulierte inkrementelle iterative Verfahren auf Elementebene zur Lösung der Navier-Stokes-Gleichungen [Szi89].

Bei der Vorgehensweise in SMART ist zu berücksichtigen, daß diese Iterationsvorschrift bereits auf Elementebene angewendet wird, bevor die so aufgestellten Elementmatrizen anschließend zur Systemmatrix assembliert werden.

Gelöst wird das System

$$\mathbf{K}^{*}\left(\mathbf{V}_{i+1}^{n} - \mathbf{V}_{i}^{n}\right) = \mathbf{R}^{\zeta} - \mathbf{P}^{\zeta} \tag{3.1.41}$$

mit

$$\mathbf{K}^{*} = \frac{1}{\Delta t}\mathbf{M}^{\zeta} + \frac{1-\zeta}{2}\mathbf{F}^{\zeta} \tag{3.1.42}$$

und

$$\mathbf{P}^{\zeta} = \frac{1}{\Delta t}\mathbf{M}^{\zeta}\left(\mathbf{V}_{i}^{n+1} - \mathbf{V}^{n}\right) - \mathbf{N}^{\zeta}\mathbf{V}_{i}^{\zeta} - \mathbf{F}^{\zeta}\mathbf{V}_{i}^{\zeta} \tag{3.1.43}$$

Pro Iterationsschritt erfolgt die Lösung von (3.1.41) durch das Cholesky-Verfahren.

Analog zu (3.1.35) und (3.1.36) gilt für die Temperatur T^{ζ} im Integrationspunkt ζ

$$T^{\zeta} = \frac{1+\zeta}{2}T^{n+1} + \frac{1-\zeta}{2}T^{n} \tag{3.1.44}$$

sowie für die zeitliche Ableitung

$$\dot{T}^{\zeta} = \frac{1}{\Delta t}\left(T^{n+1} - T^{n}\right). \tag{3.1.45}$$

Mit (3.1.44) und (3.1.45) lautet nach der Aufspaltung des konvektiven Terms in einen symmetrischen $\mathbf{n}^{s}$ und einen asymmetrischen Anteil $\mathbf{n}^{a}$ mit

$$\mathbf{c}\dot{T} + \mathbf{n}^{s}T + \mathbf{k}T = \mathbf{q} - \mathbf{n}^{a}T \tag{3.1.46}$$

die zeitliche Diskretisierung der Wärmebilanzgleichung

(3.1.47)
$$\frac{1}{\Delta t}\mathbf{c}\left(T_{i+1}^{n+1}-T^{n}\right)+\mathbf{n}^{\mathbf{s},\zeta}\left(\frac{1+\zeta}{2}T_{i+1}^{n+1}+\frac{1-\zeta}{2}T^{n}\right)+\mathbf{k}\left(\frac{1+\zeta}{2}T_{i+1}^{n+1}+\frac{1-\zeta}{2}T^{n}\right)$$
$$=\frac{1+\zeta}{2}\mathbf{q}_{\mathbf{n+1}}+\frac{1-\zeta}{2}\mathbf{q}_{\mathbf{n}}-\mathbf{n}^{\mathbf{a},\zeta}\left(\frac{1+\zeta}{2}T_{i}^{n+1}+\frac{1-\zeta}{2}T^{n}\right)$$

Das Superskript ζ bezeichnet auch hier die Abhängigkeit der Terme vom Integrationspunkt ζ, i kennzeichnet den Iterationsindex. Wird Gleichung (3.1.47) umgeordnet, ergibt sich die Iterationsvorschrift [LS91]

(3.1.48)
$$\left(\frac{1}{\Delta t}\mathbf{c}+\frac{1+\zeta}{2}\mathbf{n}^{\mathbf{s},\zeta}+\frac{1+\zeta}{2}\mathbf{k}\right)T_{i+1}^{n+1}=\frac{1+\zeta}{2}\mathbf{q}^{\mathbf{n+1}}+\frac{1-\zeta}{2}\mathbf{q}^{\mathbf{n+1}}$$
$$+\left(\frac{1}{\Delta t}\mathbf{c}-\frac{1-\zeta}{2}\mathbf{n}^{\mathbf{s},\zeta}-\frac{1-\zeta}{2}\mathbf{k}\right)T^{n}$$
$$-\left(\frac{1+\zeta}{2}\mathbf{n}^{\mathbf{a},\zeta}T_{i}^{n+1}+\frac{1-\zeta}{2}\mathbf{n}^{\mathbf{a},\zeta}T^{n}\right)$$

bzw. in zusammenfassender Schreibweise

(3.1.49)
$$\mathbf{K}^{*}T_{i+1}^{n+1}=\mathbf{R}^{*}\left(T^{i}\right)$$

Auch bei der zeitlichen Diskretisierung der Wärmebilanzgleichung wird die iterative Vorgehensweise bereits in den Elementansätzen berücksichtigt.

3.1.4 Lösung der Gleichungssysteme: Cholesky-Verfahren

In SMART erfolgt die Lösung der aufgestellten Gleichungssysteme durch das Cholesky-Verfahren. Dafür wird jedoch eine symmetrische und positiv definite Gradientenmatrix der Linkhandseite vorausgesetzt. Durch den konvektiven Term $\mathbf{n}$ wird diese Bedingung verletzt, so daß es erforderlich wird, diesen in einen symmetrischen $\mathbf{n}^{\mathbf{s}}$ und einen asymmetrischen $\mathbf{n}^{\mathbf{a}}$ Anteil aufzuspalten [LS91]

(3.1.50)
$$\mathbf{n}=\mathbf{n}^{\mathbf{s}}+\mathbf{n}^{\mathbf{a}}$$

mit

(3.1.51)
$$\mathbf{n}^{\mathbf{s}}=\frac{1}{2}\left(\mathbf{n}+\mathbf{n}^{\mathbf{t}}\right)\qquad\text{und}\qquad\mathbf{n}^{\mathbf{a}}=\frac{1}{2}\left(\mathbf{n}-\mathbf{n}^{\mathbf{t}}\right)$$

und den asymmetrischen Anteil auf der Rechthandseite zu berücksichtigen. Gleichung 3.1.29 geht in

(3.1.52)
$$\mathbf{c}\dot{T}+\mathbf{n}^{\mathbf{s}}T+\mathbf{k}T=\mathbf{q}-\mathbf{n}^{\mathbf{a}}T$$

über, welche ein iteratives Lösungsverfahren erfordert, da nun auch die Rechthandseite von der unbekannten Temperatur abhängt [Szi89], [LS91].

Anschließend erfolgt die Lösung der nun symmetrischen Gleichungssysteme (3.1.41) und (3.1.49) pro Iterationsschritt durch das direkte Cholesky-Verfahren [DH91]. Sei $Ax = b$ das zu lösende Gleichungssystem. Die Lösung x wird durch Faktorisierung, Vorwärts- und Rückwärtssubstitution wie folgt ermittelt.

- Faktorisierung $A = L\tilde{D}L^t = LU$:

$$U_{ij} = A_{ij} - \sum_{k=1}^{i-1} L_{ik}U_{ki} \qquad j > i$$
$$L_{ij} = U_{ij}/\tilde{D}_{ii} \tag{3.1.53}$$

- Vorwärtssubstitution $Lu = b \rightarrow u$:

$$u_j = b_j - L_{ij}u_i \tag{3.1.54}$$

- Rückwärtssubstitution $L^t x = u \rightarrow x$:

$$x_i = \left(Lu_i - \sum L_{ji}u_i\right)/L_{ii} \tag{3.1.55}$$

3.2 Anwenderebene

Bei einer SMART Rechnung muß zunächst die Geometrie des zu berechnenden Modells durch Finite Elemente diskretisiert werden. Dazu ist vom Anwender eine topologische Beschreibung aufzustellen, das Finite-Element-Netz. In SMART kann ein Netz bis zu $2^{15} - 1 = 32767$ Freiheitsgrade aufweisen. Für größere Problemstellungen ist die in [ADPW84] beschriebene Unterstrukturtechnik anzuwenden, die es erlaubt, 1000 Netze miteinander zu verbinden, und so eine maximale Problemgröße mit

$$32767 * 1000 = 32\,767\,000 \tag{3.2.56}$$

Freiheitsgraden/Unbekannten ermöglicht. Für Konvektionsprobleme wird diese Unterstrukturtechnik jedoch noch nicht unterstützt, so daß die Topologie nur durch ein Netz beschrieben werden kann. Da im Geschwindigkeitsfeld drei Freiheitsgrade je Elementknoten berücksichtigt werden müssen, können nur Finite-Element-Netze mit maximal 10922 Elementknoten aufgestellt werden. Nach der Diskretisierung des zu berechnenden Problems muß der Anwender die Stoffwerte des Fluids sowie Anfangs- und Randbedingungen angeben.

Neben der Beschreibung der Eingabedatensätze ist ein Steuerprogramm zu formulieren, welches den Programmablauf bestimmt. Darin können dann nichtlineares Materialverhalten und zeitabhängige Randbedingungen berücksichtigt werden. Die nachfolgende Beschreibung der topologischen und physikalischen Problembeschreibung sowie des Steuerprogramms ermöglichen einen Einblick in die mit SMART formulierbaren Probleme.

3.2.1 Topologische und physikalische Problembeschreibung

Vor einer SMART-Rechnung müssen die Modelldiskretisierung und die Stoffwerte in einer Datei zusammengestellt werden, die dann als Eingabedatei für das SMART-Steuerprogramm genutzt wird. Bei Konvektionsströmungen sind zwei Eingabedateien zu erstellen, eine für das Geschwindigkeits- und eine für das Temperaturnetz.

In diesen Eingabedateien wird zunächst die Modelldiskretisierung durch Finite Elemente beschrieben, wobei die Elemente aus der in Anhang C angegebenen Elementbibliothek ausgewählt werden müssen. Die topologische Beschreibung umfaßt die Netzdaten, d.h. die Art, Anzahl und Verknüpfung der Elemente und insbesondere die Numerierung der Knotenpunkte. Zusätzlich sind die Freiheitsgrade festzulegen. Diese werden in vier Klassen unterteilt:

SUPPRESSED	Freiheitsgrade, an denen Null-Werte vorgeschrieben werden
PRESCRIBED	Freiheitsgrade, die vorgeschriebene Werte aufweisen
EXTERNAL	Freiheitsgrade, die Teilnetze mit dem Hauptnetz verbinden (nur bei der Unterstrukturtechnik)
LOCAL	Freiheitsgrade, die als primäre Unbekannte auftreten

Neben der Diskretisierung der Gesamtgeometrie müssen weitere Daten für die Finiten Elemente und Knotenpunkte angegeben werden. Dazu gehören Knotenpunktkoordinaten und Informationen über lokale Koordinatensysteme sowie die Anfangszustände für Geschwindigkeits- und Temperaturfeld und vorgeschriebene Geschwindigkeiten und Temperaturen. Hinzu kommen die Stoffdaten für Temperaturausdehnung (ALFA), Konduktivität (CONI), Kapazität (CAPA), Dichte (DENS) und dynamische Viskosität (VISK) sowie der Penalty-Parameter (BULK), die Lasten (BQIN) und Elementströme (FLOW).

In Anhang D werden die Eingabedatensätze für ein Beispiel aufgezeigt.

3.2.2 Steuerprogramm

In einem Steuerprogramm, dem SPC (SMART Procedure Call), formuliert der Anwender den Problemlösungsweg. Dabei werden Parameter wie Anzahl der Zeitschritte, Zeitschrittweite, UPWIND-Faktor, Zahl der Iterationen, Konvergenzkriterien etc. festgelegt. Weiterhin werden Routinen aufgerufen, die den Problemlösungsweg beschreiben. Dieses sind beispielsweise Routinen, die lineares bzw. nichtlineares Materialverhalten berücksichtigen oder das Zeitintegrationsverfahren festlegen. Ferner können instationäre Randbedingungen angegeben werden. Der Benutzer wird durch ein interaktives Programm dabei unterstützt, dieses Steuerprogramm aufzustellen. Er benötigt so keine Information über die mathematischen Algorithmen, sondern ruft Routinen auf, die gesamte Problemlösungsabschnitte durchführen. Andererseits ermöglicht die umfangreiche Dokumentation einen weitgehenden Einblick in Programm- und Datenstrukturen, so daß bei Bedarf Problemlösungsabschnitte ergänzt oder verändert werden können.

Einige der Routinen, die vom Anwender aufzurufen sind, werden in den nachfolgenden Tabellen zusammengestellt:

INIV	Initialisierung des Anfangszustands
LNAVLI	Aufstellen der Gradientenmatrix, Linkhandseite, linear
LNAVNL	Aufstellen der Gradientenmatrix, Linkhandseite, nichtlinear
RNAVLI	Lastvektor Rechthandseite, linear
RNAVNL	Lastvektor Rechthandseite, nichtlinear
SOLVNA	Lösung des Gleichungssystems, linear
SOLVNL	Lösung des Gleichungssystems, nichtlinear

Tabelle 3: Routinen zur Berechnung der Navier-Stokes-Gleichungen

INIC	Initialisierung des Anfangszustands
LCONLI	Aufstellen der Gradientenmatrix, Linkhandseite, linear
LCONNL	Aufstellen der Gradientenmatrix, Linkhandseite, nichtlinear
RECONV	Lastvektor Rechthandseite, linear und nichtlinear
SOLVNL	Lösung der Gleichungssysteme, linear und nichtlinear

Tabelle 4: Routinen zur Berechnung der Wärmebilanzgleichung

Diese problemorientierten Routinen werden durch Verwaltungsroutinen ergänzt, welche die Daten aus der Eingabedatei einlesen und aufbereiten sowie die Daten organisieren. Für eine Beispielrechnung ist ein SPC im Anhang E aufgeführt.

3.3 Systemebene

In diesem Abschnitt werden die Programm- und Datenstrukturen von SMART zusammengefaßt, die für die Parallelisierung wichtig sind, bzw. die für die Implementation der Parallelversion ergänzt werden müssen [Ham74], [Sch71], [Str71]. Insbesondere ist die Hierarchie der Routinen zu nennen, die es erleichtert, einzelne Problemlösungsschritte zu verändern bzw. zu parallelisieren. Wichtig ist auch die Organisation der bei Finite-Element-Berechnungen entstehenden großen Matrizen in einem Hypermatrizen-Speicherschema. In SMART ist zur Organisation der umfangreichen Datenmengen, ein Datenverwaltungssystem (DRS, Data Retrieval System) implementiert [Ham74].

3.3.1 Programmstruktur

Die etwa 4000 Routinen, aus denen das System SMART besteht, können, wie in Tabelle 5 gezeigt, strukturiert werden. Ebene 1 zeigt die Routinen, die aus dem SPC aufgerufen werden.

1	Problemorientierte Routinen, die vom Anwender aufzurufen sind (SOLVNL, SOLVNA,...)
2	mathematische Lösungsalgorithmen Cholesky: TRIA, SOLV Zeitintegration: BACKBV (Rückwärtsdifferenzen-Verfahren) ... Gaussintegration: GAUSS
3	Formulierung mathematischer Operationen auf Hypermatrizenebene Matrixaddition: ADDH Matrixmultiplikation: MULTH ....
4a	logische Datenorganisation: Verwaltung der Daten in Büchern Bücher anlegen: NEWBUK Bücher speichern: SAVBUK Bücher löschen: REFBUK ...
4b	physikalische Datenorganisation: Datentransporte Einlesen von Seiten Ändern von Seiten Speichern von Seiten

Tabelle 5: Strukturierung der SMART-Routinen (Ebenen 1 bis 4)

Dieses sind Routinen zur Berechnung einzelner Lösungsschritte, wie bereits in Abschnitt 3.2.2 erläutert. Informationen über die verwendeten mathematischen Algorithmen und die Datenorganisation benötigt der Anwender nicht, eine ausführliche Dokumentation kann jedoch bei Bedarf den SMART Handbüchern [Ham74], [HK74], [Sch71], [Str71] entnommen werden. Auf Ebene 2 werden die mathematischen Algorithmen zusammengefaßt, wie z.B. die Lösung der Gleichungssysteme oder die Integration. Die Operationen werden auf dem Datentyp *Buch* durchgeführt, welcher in Abschnitt 3.3.2 erklärt wird. Die Ebene 3 stellt die mathematischen Operationen auf der Datenstruktur *Buch* zusammen, so daß diese Operationen auf einer hohen Abstraktionsebene ausgeführt werden können. Die Ebenen 4a und 4b führen die Datenorganisation durch das DRS aus. Es werden *Bücher* angelegt und verwaltet sowie die Datentransporte zum Hintergrundspeicher organisiert.

3.3.2 Datenstrukturen

Das Programmsystem SMART bzw. das System ASKA ist bereits Anfang der 70er Jahre entwickelt worden. Die damals verfügbaren Hauptspeicher waren auf einige KByte beschränkt. Um komplexe Probleme nach der Finite-Element-Methode zu lösen, müssen große Datenmengen wie z.B. die Systemmatrix gespeichert werden. Das DRS (Data Retrieval System) organisiert deshalb den verfügbaren Hauptspeicherplatz und ermöglicht es, Daten auf Platten auszulagern [Ham74]. Dem Anwender bleibt diese Organisation verborgen und der Programmierer, der z.B. mathematische Algorithmen implementiert, kann dieses auf abstrakten Datentypen durchführen, ohne Speicherplatzprobleme lösen zu müssen, indem er Operationen auf den *Büchern* formuliert.

Bevor der Datentyp *Buch* beschrieben wird, soll ein kurzer Einblick in die Arbeitsweise des DRS gegeben werden. Das DRS unterteilt den verfügbaren Hauptspeicherbereich 'physikalisch' in *Seiten*, die zwischen Hauptspeicherbereich und Plattenspeicher transportiert werden. Auf eine Seite können mehrere *Paragraphen* abgebildet werden, die die Daten logisch zusammenfassen. Ein *Buch* ist eine Zusammenfassung mehrerer *Paragraphen.* Programmiertechnisch besteht es aus einer mnemonischen Bezeichnung, die auf den Speicherplatz im Hintergrundspeicher zeigt.

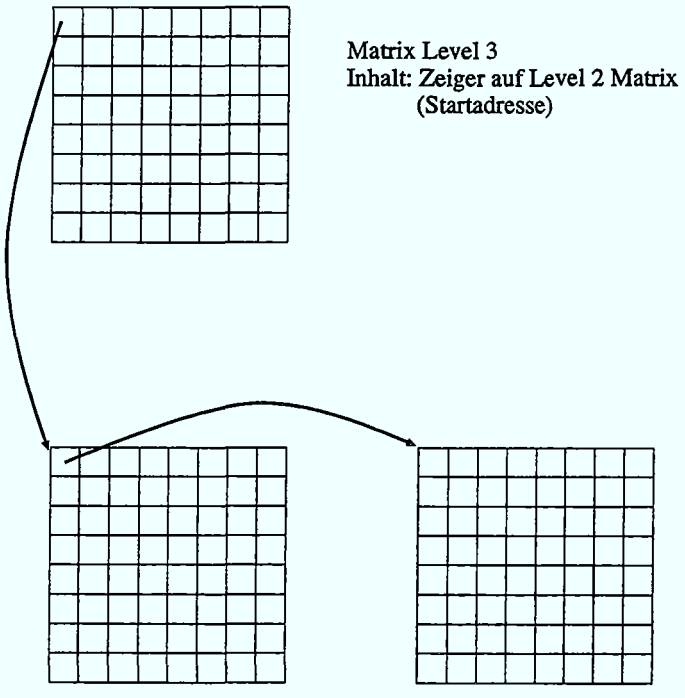


Bild 3.3.1: Dreistufiges Hypermatrizenschema in SMART

In Büchern gibt es zwei sich ergänzende Speicherschemata, das Hypermatrizenschema und die Kontrollisten. Das Hypermatrizenschema wird in Bild 3.3.1 dargestellt. Die Level 3 Matrix, die vom Benutzer durch einen mnemonischen Namen adressiert werden kann, besteht aus Zeigern auf Level 2 Matrizen. Die Matrixeinträge in der Level 2 Matrix stellen wiederum Zeiger auf Matrizen der Ebene 1 dar. Eine solche Speichertechnik erlaubt eine Partitionierung der Systemmatrix, die somit abschnittsweise in den Hauptspeicher gelagert, bzw. aus diesem ausgelagert werden kann. Der Speicherplatzbedarf wird in diesem Schema verringert, indem nur Matrizen gespeichert werden, die mindestens ein Nichtnull-Element enthalten. In dem Hypermatrizenschema werden die Matrizen und Vektoren auf Element- sowie auf Systemebene gespeichert [Str71].

Eine weitere Datenstruktur bildet die Kontrolliste, die in zweidimensionalen Feldern die weiteren erforderlichen Informationen organisiert. Dazu gehören Daten über Elementeigenschaften, Freiheitsgrade der Elemente, Elemente mit lokalen Koordinatensystemen sowie Informationen über den Zugriffspfad auf die numerischen Daten einer Hypermatrix [Str71].

3.4 Schematische Darstellung einer Rechnung mit SMART-Konvektion

Im folgenden wird eine Berechnung mit SMART-Konvektion vorgestellt. Dabei werden die Schritte, die der Anwender durchführen muß, sowie der Programmablauf beschrieben.

Vor einer Berechnung mit SMART sind die Eingabedatensätze mit der Geometriediskretisierung, den Anfangs- und Randbedingungen sowie den Stoffwerten zu erstellen, vergl. Abschnitt 3.2.1 und Anhang D. Anschließend sind diese Datensätze durch Preprozessoren, die das Programmsystem SMART ergänzen, aufzubereiten. Diese Preprozessoren haben die Aufgabe, die vom Benutzer angegebene problemorientierte Eingabedatei in ein für SMART lesbares Datenformat zu konvertieren.

In der anschließenden SMART-Rechnung wird das in Abschnitt 3.2.2 erläuterte SPC ausgeführt. Dazu werden zunächst die Variablen für Zeitschrittgröße und –anzahl, UPWIND-Faktor etc. initialisiert. Nachfolgend werden die durch die Preprozessoren aufbereiteten Datensätze eingelesen und durch das DRS, wie in Abschnitt 3.3.2 beschrieben, in den Kontrollisten auf dem Hintergrundspeicher organisiert. Nach dieser Initialisierungsphase kann die Hintergrundspeicherdatei als Datenbank betrachtet werden, aus der Daten durch verschiedenartige Zugriffsmechanismen ausgelesen, bearbeitet und in diese zurückgeschrieben werden können. Der Speicherplatz für später aufzustellende Systemmatrizen wird zu diesem Zeitpunkt bereits reserviert.

Nach dieser nur einmal auszuführenden Initialisierungsphase werden die Anweisungen der Zeitschleife bearbeitet. Bei der in Bild 3.4.1 beispielhaft dargestellten Berechnung einer voll gekoppelten Konvektionsströmung wird zunächst das Geschwindigkeitsfeld bestimmt.

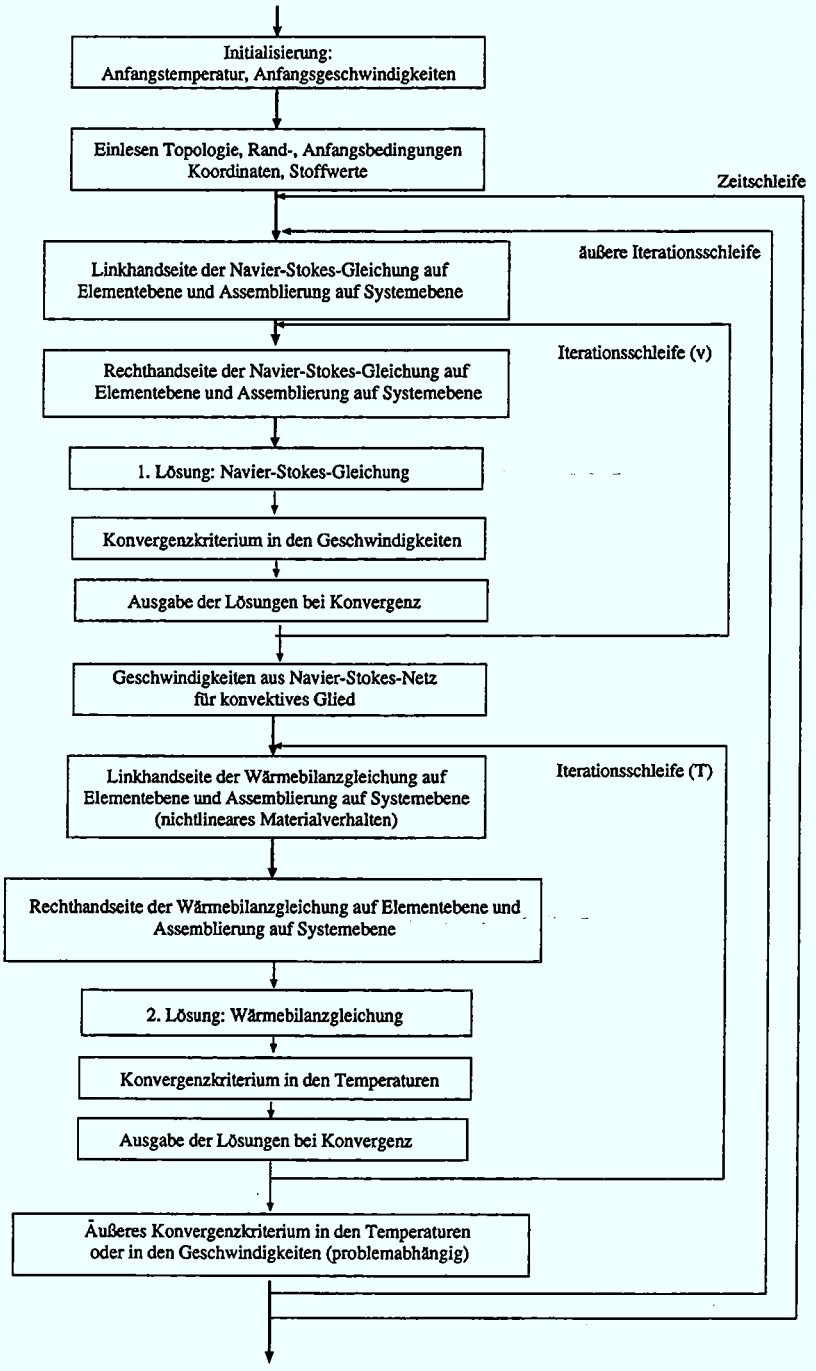


Bild 3.4.1: SMART Rechnung mit nichtlinearen Materialeigenschaften

Dazu wird die Linkhandseite auf Elementebene nach der in Abschnitt 3.1.2 und 3.1.3 beschriebenen Vorgehensweise bestimmt und auf Systemebene assembliert. Es folgt die Aufstellung der Rechthandseite sowie die Lösung des Gleichungssystems durch das Cholesky-

Verfahren. Durch das konvektive Glied wird die Matrix der Linkhandseite asymmetrisch, so daß nach Abschnitt 3.1.4 ein iteratives Verfahren durchgeführt werden muß. Ist das Konvergenzkriterium erfüllt, wird das Temperaturfeld analog bestimmt, indem zunächst die Elementmatrizen bestimmt und zur Systemmatrix assembliert werden. Anschließend wird die Rechthandseite ermittelt und das aufgestellte, asymmetrische Gleichungssystem durch ein iteratives Vorgehen gelöst. Bei der Temperaturfeldberechnung muß beachtet werden, daß bei temperaturabhängigen Materialdaten, die Linkhandseite der Wärmebilanzgleichung für jeden Iterationsschritt neu zu bestimmen ist.

Nachdem Geschwindigkeits- und Temperaturfeld zunächst unabhängig voneinander bestimmt wurden, werden sie durch ein Prädiktor-Korrektor-Verfahren für den betrachteten Zeitschritt aufeinander abgestimmt. Dieses wird in Bild 3.4.1 als äußere Iterationsschleife bezeichnet. Ist für einen Zeitschritt eine konsistente Lösung für Geschwindigkeits- und Temperaturfeld ermittelt worden, wird der nächste Zeitschritt berechnet. Dabei ist zu beachten, daß das in Abschnitt 3.1.3 erläuterte Zeitintegrationsverfahren bereits auf Elementebene durchgeführt wird.

4 Konzept gleichberechtigter Knoten für die Parallelversion ParSmart

Das neu entwickelte Konzept für die Parallelversion ParSmart wird in diesem Abschnitt vorgestellt. Dieses Konzept muß verschiedenartigen Anforderungen genügen, die zunächst beschrieben werden. Anschließend wird analysiert, wie sich die Rechenzeiten einer seriellen SMART-Rechnung den einzelnen Problemlösungsabschnitten zuordnen lassen. Anhand dieser Zuordnung der Rechenzeiten werden Einsatzmöglichkeiten für den Parallelrechner Intel/Paragon diskutiert.

Unter Beachtung der zu erfüllenden Anforderungen und der Verteilung der Rechenzeiten auf die einzelnen Problemlösungsschritte wird das Gesamtkonzept gleichberechtigter Rechenknoten für ParSmart aufgestellt und der Ablauf einer parallelen Berechnung beschrieben.

4.1 Anforderungen an die Parallelversion

ParSmart soll wie auch SMART als Produktionsprogramm eingesetzt werden. Es muß daher den gleichen Leistungsumfang anbieten wie das serielle System. Hinzu kommen Anforderungen aus Anwendersicht, sowie aus Sicht der Softwarepflege und der Parallelverarbeitung.

Dabei ist anzumerken, daß diese Anforderungen durchaus Zielkonflikte erzeugen. So kann beispielsweise bei weitgehender Übernahme der für serielle Architekturen optimierten Algorithmen die Skalierbarkeit nicht uneingeschränkt gewährleistet werden.

4.1.1 Anforderungen aus Anwendersicht

Um dem Anwender zu ermöglichen, die durch den langjährigen Einsatz von SMART gewonnenen Erfahrungen weiterhin zu nutzen, sollte die Anwenderschnittstelle unverändert bleiben. Das bedeutet, daß die Eingabedatensätze wie bislang für einen seriellen Lauf aufgestellt werden und auch keine Veränderungen in dem in Abschnitt 3.2.2 beschriebenen Steuerprogramm vorzunehmen sind. Bereits erstellte Modelle können damit auch auf dem Parallelrechner verwendet werden.

Da der Anwender in SMART einzelne Problemlösungsschritte verändern kann, muß dieses auch in der Parallelversion ParSmart möglich sein. Insbesondere bedeutet das, die bisherigen Programm- und Datenstrukturen zu übernehmen und sie gegebenenfalls nur zu ergänzen.

Ferner sollte das Parallelkonzept dem Anwender weitgehend verborgen bleiben, d.h. er sollte keine zusätzlichen Arbeiten, wie beispielsweise die Zerlegung in Teilmodelle, übernehmen müssen. Ausgehend von der Problemgröße sollte automatisch eine Näherung für die optimale Anzahl von Rechenknoten ermittelt werden, auf der das parallele Programm effizient ausgeführt wird.

4.1.2 Anforderungen zur 'Softwarepflege'

Das Programmsystem SMART ist über einen langen Zeitraum entwickelt, optimiert und verifiziert worden. Insbesondere die gute Verifikation spricht für die Parallelisierung des bestehenden Programms, so daß die getesteten problemspezifischen Routinen deshalb beibehalten werden sollten.

SMART bietet ein umfangreiches Fehleranalysesystem, welches sowohl fehlerhafte Installationsparameter als auch Eingabe- und Laufzeitfehler dokumentiert. Diese Fehleranalyse muß auch in der parallelen Version erhalten bleiben.

Auch sollen die bislang gewonnenen Erfahrungen bei der Softwarepflege und –erweiterung genutzt werden können, das bedeutet Programm- und Datenstrukturen der seriellen Version beizubehalten und das Parallelkonzept als Erweiterung dieser seriellen Version aufzufassen.

4.1.3 Anforderungen zur Skalierbarkeit

Die optimale Skalierbarkeit des Programmsystems für eine wachsende Anzahl von Rechenknoten setzt entsprechend Abschnitt 2.2.3 voraus, daß kein serieller Anteil im parallelen Programm enthalten ist. Diese Anforderung ist praktisch nicht zu erfüllen, allerdings sollte die Minimierung des seriellen Anteils eine hohe Priorität besitzen. Voraussetzung für diese Minimierung ist die Parallelisierung aller Programmabschnitte.

Um die Architektur mit verteiltem Speicher des Rechners Intel/Paragon weitgehend zu nutzen, ist eine Verteilung der Daten auf die an einer Rechnung beteiligten Rechenknoten erforderlich. Dieses ermöglicht dann auch lokale Berechnungen und dient somit der Kommunikationsminimierung.

4.2 Analyse einer SMART Rechnung

In diesem Abschnitt wird untersucht, wie sich die Rechenzeiten einer seriellen SMART-Rechnung auf die einzelnen Berechnungsabschnitte verteilen. Anhand dieser Analyse werden nachfolgend die Einsatzmöglichkeiten parallel arbeitender Rechenknoten diskutiert.

4.2.1 Analyse der Rechenzeiten

Um ein Konzept zur Parallelisierung des bestehenden Programms zu entwickeln, muß die Verteilung der Rechenzeiten auf die einzelnen Programmabschnitte analysiert werden [MOT86]. Ausgehend von dieser Verteilung müssen dann insbesondere diejenigen Abschnitte effizient parallel ausführbar sein, für die die meiste Rechenzeit benötigt wird.

Tabelle 6 zeigt die Aufteilung der Rechenzeiten auf die einzelnen Programmabschnitte für eine serielle Berechnung mit SMART. Dabei ist das in Abschnitt 6.1 betrachtete Beispiel der Rayleigh-Bénard Konvektion zugrundegelegt worden; für andere Beispiele ergibt sich eine sehr ähnliche Aufteilung der Rechenzeiten. Es wird der Zeitverbrauch in Sekunden, der CPU-, System- und IO-Zeiten umfaßt, angegeben. Diese Rechenzeiten beziehen sich nur auf

einen Zeitschritt. Da die Dateninitialisierung nur einmal zu Beginn ausgeführt werden muß, die Zeitschleife aber wiederholt durchlaufen wird, wird der prozentuale Zeitverbrauch der einzelnen Routinen innerhalb einer Zeitschleife bestimmt.

	Sekunden	Prozent
Dateninitialisierung	160	
Anfangswerte für Geschwindigkeiten	44.2	
Anfangswerte für Temperaturen	21	
Aufstellung der Linkhandseite der Navier-Stokes-Gleichung, bestehend aus: Berechnung der Elementmatrizen (46 s) Assemblierung auf Systemebene (148 s) Triangularisierung (415 s)	611.1	
Linkhandseite der Wärmebilanzgleichung, bestehend aus: Berechnung der Elementmatrizen (53.3 s, 1.8%) Assemblierung auf Systemebene (118.7 s, 4.0%) Triangularisierung (2350.9 s, 80.1%)	2530	86.2%
Rechthandseite der Wärmebilanzgleichung	6.5	0.2%
Rechthandseite der Navier-Stokes-Gleichung	81	2.8%
Lösung der Wärmebilanzgleichung	115	3.9%
Lösung der Navier-Stokes-Gleichung	187	6.4%
Datenverwaltung	17.2	0.6%
Zeitschleife:	2936.7	100%
Gesamtzeitverbrauch	3 773	

Tabelle 6: Analyse der Rechenzeiten einer SMART-Rechnung (Rechner Intel/Paragon)

Bei der betrachteten Berechnung ist weiterhin zu beachten, daß die Linkhandseite des Gleichungssystems zur Lösung der Navier-Stokes-Gleichungen aufgrund konstanter Zeitschrittweite während der Berechnung nicht verändert werden muß und somit nur einmal vor der Zeitschleife aufzustellen ist. Sind hingegen variable Zeitschrittweiten vorhanden, muß auch diese Linkhandseite in jedem Zeitschritt neu bestimmt werden. In diesem Fall erhöht die

Triangularisierung der Linkhandseite den Anteil der Rechenzeit innerhalb der Zeitschleife, der zur Lösung der Gleichungssysteme benötigt wird.

Die Angaben in Tabelle 6 zeigen, daß etwa 90% der Rechenzeit auf die Lösung beider Gleichungssysteme durch das in Abschnitt 3.1.4 beschriebene Cholesky-Verfahren (Triangularisierung und Lösung) entfallen.

Aus diesem Grund muß insbesondere für die Lösung der Gleichungssysteme ein effizient parallel ausführbares Verfahren gewählt werden. Dieses wird auch bei der in Abschnitt 2.4 erwähnten Parallelisierung der Programmsysteme PERMAS und NASTRAN berücksichtigt.

Wird jedoch nur das etwa 90% der Gesamtrechenzeit beanspruchende Lösungsverfahren parallelisiert, so bleibt ein serieller Anteil von 10% bestehen. Auch unter der Annahme, daß die Gleichungssysteme mit linearem Speedup gelöst werden können, ist nach Abschnitt 2.2.3 für die gesamte Berechnung asymptotisch nur ein maximaler Speedup von 10 erreichbar. Das bedeutet, daß auch die anderen Berechnungsabschnitte in das Parallelisierungskonzept einzubeziehen sind, also auch die Berechnung der Integrale auf Elementebene und deren Assemblierung sowie der Verwaltungsaufwand.

4.2.2 Einsatzmöglichkeiten für den Parallelrechner Intel/Paragon

Im folgenden werden die Einsatzmöglichkeiten des Parallelrechners Intel/Paragon für Finite-Element-Berechnungen mit SMART diskutiert, wobei die vorangehend beschriebenen Anforderungen und die Analyse der Rechenzeiten berücksichtigt werden.

Der umfassendste Anteil einer SMART Rechnung ist die Lösung der aufgestellten Gleichungssysteme. Dazu wird bislang das direkte Cholesky-Verfahren eingesetzt, welches für die Triangularisierung, den aufwendigsten Rechenabschnitt, serielle Anweisungen erfordert und somit nur unter erheblichen Modifikationen parallel ausführbar ist. Hier erscheint eine algorithmische Änderung in SMART geboten, welche das direkte Cholesky-Verfahren durch ein iteratives Lösungsverfahren ersetzt und somit eine größere Leistungssteigerung verspricht.

Für die parallele Gleichungslösung ist es erforderlich, die Daten, das sind in diesem Fall die Matrizen der Linkhandseiten sowie die Vektoren der Rechthandseiten, auf den verteilten Speicher des Rechners Intel/Paragon aufzuteilen, um weitgehend lokale Operationen zu ermöglichen und den Kommunikationsaufwand zu minimieren.

Neben der Gleichungslösung sind auch die anderen Abschnitte einer Finite-Element-Berechnung zu parallelisieren. Die Berechnung der Integrale (3.1.15) – (3.1.19) und (3.1.30) – (3.1.33) auf Elementebene ist dabei lokal für jedes Element, d.h. unabhängig von den benachbarten Elementen durchführbar und kann somit parallel erfolgen. Hier wird die MIMD-Architektur des Paragon benötigt, da bei Finite-Element-Diskretisierungen verschiedenartige Elemente genutzt werden können, die unterschiedlich behandelt werden müssen.

Die Assemblierung der Elementmatrizen zu den Systemmatrizen kann in zwei Stufen erfolgen. Während in der ersten Stufe die benachbarten Elemente, die sich auf einem Rechenknoten befinden, lokal zusammengefaßt werden, ist in der zweiten Stufe die Assemblierung der Elementbeiträge durchzuführen, die nicht auf einem Rechenknoten gespeichert

werden. Dazu müssen Elementdaten über das Kommunikationsnetzwerk des Parallelrechners ausgetauscht werden. Die Kommunikation geschieht dabei stets zwischen benachbarten Teilmodellen und wird am schnellsten durchgeführt, wenn diese Teilmodelle auf direkt miteinander verbundenen Rechenknoten gespeichert werden. In der Praxis ist das jedoch nur für wenige Modelle möglich.

Die Wahl zwischen einem Master-Slave Konzept und einem Konzept gleichberechtigter Knoten wird im folgenden diskutiert. Dazu ist zunächst anzumerken, daß die homogene Architektur des Rechners Intel/Paragon ein Konzept gleichberechtigter Knoten nahelegt. Auch das gitterförmige Kommunikationsnetzwerk dieses Rechners spricht gegen ein Master-Slave Konzept, welches eine gleichwertige Kommunikation des Master-Knotens mit allen Slave-Knoten erfordert. Diese kann von dem vorliegenden Kommunikationsnetz nicht ermöglicht werden.

Das Master-Slave Konzept würde auf einem Rechenknoten eine zentrale Prozeß– und Datenverwaltung durchführen. Dieses bedingt weitgehende Eingriffe in das bestehende Programm. Das nichtdeterministische Modell dieser Strategie erfordert zusätzlich erhebliche Erweiterungen zur Analyse von Fehlern, da ein Berechnungslauf nicht mit gleicher Daten- und Prozeßverteilung wiederholbar ist.

In Anbetracht der zuvor aufgestellten Anforderungen an die Parallelversion ParSmart und der Architektur des Rechners Intel/Paragon ist das Konzept gleichberechtigter Knoten zu bevorzugen. Es ermöglicht, weitgehend Programmabschnitte zu übernehmen und so die Verifikationszeiten zu verringern. Dabei wird ein zu berechnendes Modell in Teilmodelle zerlegt, die jeweils in einer lokalen SMART-Rechnung behandelt werden. Die Abhängigkeiten der Teilmodelle voneinander werden durch Datentransfer zwischen den Rechenknoten berücksichtigt. Der Nachteil einer unflexiblen Datenstruktur ist vertretbar, da SMART die Datenstrukturen für das Gesamtsystem zu Beginn einer Berechnung festlegt und dann unverändert beibehält. Berechnungen, bei denen Diskretisierungen nachträglich verfeinert werden können (adaptive Verfahren [DLY89], [PVMZ87]), sind somit derzeit nicht einsetzbar.

4.3 Gesamtkonzept

Das Gesamtkonzept für die Parallelversion ParSmart ist ein Konzept gleichberechtigter Knoten, wie in Bild 2.4.2 dargestellt.

Eine ParSmart-Rechnung beginnt mit einer Modellzerlegung, welche das vom Anwender beschriebene Gesamtmodell in Teilmodelle zerlegt und sie jeweils den Rechenknoten des Parallelrechners zuordnet [WA94]. Die so ermittelte Datenaufteilung bleibt während einer parallelen Berechnung unverändert. Die Rechenknoten bearbeiten dann weitgehend nur die Daten, die ihnen durch die Teilmodelle zugeordnet werden. So stellt jeder Rechenknoten bei der Berechnung der Elementbeiträge nur die Elementmatrizen für sein Teilmodell auf. Für die anschließende Assemblierung der Gesamtsystemmatrix ist jedoch Kommunikation zwischen den Rechenknoten erforderlich, um die Abhängigkeiten der Teilmodelle untereinander zu berücksichtigen. Die Lösung der Gleichungssysteme erfolgt dann rechenknotenübergreifend.

Unhabhängig von dem Lösungsverfahren ist ergänzend zu den lokal ausführbaren Berechnungen stets Datentransfer erforderlich.

Nachfolgend werden die einzelnen Programmabschnitte einer parallelen Berechnung mit ParSmart erläutert und die vorzunehmenden Ergänzungen beschrieben. Die speziell für ParSmart ausgewählten und implementierten Algorithmen werden anschließend in Kapitel 5.1 vorgestellt.

Datenvorbereitung *(preprocessing)*:

In der parallelen Version wird vor dem Aufruf der Preprozessoren zunächst das vom Anwender erstellte Modell durch ein Modellzerlegungsverfahren in p^* Teilmodelle zerlegt, die den Rechenknoten zugeordnet werden. Dabei kann für p^* die Anzahl der Rechenknoten bestimmt werden, die die kürzeste Rechenzeit ermöglicht. Für die ermittelten Teilmodelle werden SMART-Eingabedatensätze aufgestellt, die anschließend durch die entsprechenden Preprozessoren aufbereitet werden. Durch die Zerlegung des Modells vor dem Aufruf der Preprozessoren ermöglicht ParSmart die Berechnung von Finite-Element-Modellen mit mehr als 32767 Freiheitsgraden, vgl. Abschnitt 3.2. Die Größe der berechenbaren Modelle hängt nach dieser Änderung von der Anzahl der Rechenknoten ab. So muß in der Parallelversion nur noch jedes Teilmodell weniger als 32767 Freiheitsgrade aufweisen. Werden beispielsweise 140 Rechenknoten eingesetzt, so können Modelle mit bis zu etwa 4500000 Freiheitsgraden berechnet werden.

Vor Beginn der ParSmart-Berechnung sind die Teilmodelle den Rechenknoten des Parallelrechners bereits zugeordnet. Während der Laufzeit des Programms ändert sich diese Zuordnung nicht. Die ergänzende Modellzerlegung wird in Abschnitt 5.1.1 erläutert.

Programminitialisierung:

Das Konzept gleichberechtigter Knoten sieht die Programminitialisierung auf jedem Rechenknoten vor. Der ausführbare ParSmart-Code muß somit auf jedem Rechenknoten gespeichert werden.

Dateninitialisierung:

Nach dem Start einer parallelen ParSmart-Rechnung bearbeitet jeder Rechenknoten das ihm in der Datenvorbereitungsphase zugeordnete Teilmodell. Das Einlesen der Datensätze geschieht parallel auf den Rechenknoten und jeweils für das zugeordnete Teilmodell. In diesem Schritt werden wie auch in der seriellen SMART-Version die Speicherplätze für die nachfolgenden Berechnungen festgelegt. Dabei ist jedoch zu beachten, daß jeder Rechenknoten nur ein Teilmodell verwaltet, d.h. auch nur einen Teil der Gesamtdaten organisiert. Die für Parallelrechner mit verteiltem Speicher geforderte Datenaufteilung liegt somit vor.

Berechnungen auf Elementebene:

Die Berechnung der Integrale (3.1.15)-(3.1.19) und (3.1.30)-(3.1.33) kann für jedes Element unabhängig von den benachbarten Elementen durchgeführt werden. Die parallel arbeitenden Rechenknoten stellen die Elementmatrizen nur für die Elemente der von ihnen bearbeiteten Teilmodelle auf. In diesem Abschnitt arbeiten die Rechenknoten lokal und benötigen keine Informationen über die benachbarten Teilmodelle. Somit ist keine Kommunikation erforderlich und theoretisch linearer Speedup erreichbar, wenn die Rechenzeit der Teilmodelle gleich hoch ist.

Assemblierung der Systemmatrix:

Die Gesamtsystemmatrix wird in der parallelen Version in zwei Schritten aufgestellt. In dem ersten Schritt werden jeweils lokal die Elementmatrizen für die Teilmodelle auf den Rechenknoten assembliert. Anschließend erfolgt in dem zweiten Schritt die rechenknotenübergreifende Zusammenfassung der lokalen Systemmatrizen der Teilmodelle.

In dem zweiten Schritt müssen auch Knotendaten der Finiten Elemente, die mit Elementen auf anderen Rechenknoten verbunden sind, assembliert werden. Dieser Schritt ist durchzuführen, um die Abhängigkeiten der Teilmodelle untereinander zu berücksichtigen.

Für diese rechenknotenübergreifende Kommunikation wird in ParSmart das in Abschnitt 5.1.2 beschriebene Kommunikationsschema implementiert.

Bei der Kommunikation zur globalen Assemblierung werden die Daten zugleich so verteilt, daß die Gesamtmatrix anschließend zeilenweise verteilt auf den Rechenknoten gespeichert wird. Dieser Datenaustausch erfolgt, indem sendende Rechenknoten Matrixzeilen aus dem Hypermatrizenschema auslesen und an die Empfängerknoten verschicken. Diese Empfängerknoten ergänzen die lokal verwalteten Matrixzeilen um die erhaltenen Werte.

Lösung der Gleichungssysteme:

Die aufgestellten Gleichungssysteme für die Navier-Stokes-Gleichungen und für die Wärmebilanzgleichung werden durch ein konjugiertes Gradienten-Verfahren gelöst. Dazu werden die SMART-Routinen zur Triangularisierung und Lösung mit dem Cholesky-Verfahren durch den Aufruf des parallelen konjugierten Gradienten-Verfahrens ersetzt. Dieses Iterationsverfahren wird parallel ausgeführt, indem jeder Rechenknoten die erforderlichen Operationen auf den von ihm verwalteten Daten ausführt. Nach der Kommunikation zur Assemblierung der Gesamtmatrix liegt diese zeilenweise verteilt auf den Rechenknoten vor, so daß bei der Lösung der Gleichungssysteme jeder Rechenknoten jeweils die ihm zugeordneten Zeilen der Gesamtmatrix bearbeitet.

Während jeder Iteration ist Kommunikation zwischen den Rechenknoten erforderlich, um die globale Näherung für den Lösungsvektor des Iterationsschrittes aufzustellen und auf die beteiligten Rechenknoten zu verteilen. Hinzu kommt eine globale Operation zur Überprüfung der Konvergenz. Diese in jedem Iterationsschritt ausgeführte Kommunikation wirkt sich negativ auf den erreichbaren Leistungszuwachs aus.

Das für ParSmart implementierte parallele konjugierte Gradienten-Verfahren wird in Abschnitt 5.1.3 beschrieben.

Datenausgabe:

Nach der verteilten Berechnung müssen die Teilergebnisse der Rechenknoten je Zeitschritt zusammengefaßt und in eine gemeinsame Datei geschrieben werden.

Die Parallelisierungsstrategie besteht, wie beschrieben, in der Zerlegung des zu berechnenden Gesamtmodells in Teilmodelle, die zunächst jeweils unabhängig von den Rechenknoten behandelt werden können. Für den Anwender bedeutet es, daß er die Eingabedatensätze für das Gesamtmodell und das Steuerprogramm wie bislang aufstellen kann und keine Änderungen durchführen muß. Die parallele Berechnung beginnt dann mit einer lokalen SMART-Rechnung für jedes der aufgestellten Teilmodelle [Wat95]. Das bedeutet, daß der SMART-Code auf jedem Rechenknoten gespeichert und ausgeführt werden muß. Dieses Vorgehen erlaubt, daß Programmteile weitgehend unverändert übernommen werden können, und erfüllt somit die Forderung, Programm- und Datenstrukturen beizubehalten. Somit können die Verifikationszeiten verkürzt werden. Da jeder Rechenknoten zunächst eine unabhänge SMART-Rechnung ausführt, kann auch das vorhandene umfangreiche Fehleranalysesystem weiterhin genutzt werden. Allerdings ist zu beachten, daß es sich auf die Teilmodelle bezieht. Sollten bei der parallelen Berechnung Fehler auftreten, so können direkt die Rechenknoten bestimmt werden, auf denen keine korrekte Programmausführung stattfindet.

Das Kommunikationsschema, wie auch das Verfahren zur Lösung der Gleichungssysteme ergänzen den seriellen SMART-Quellcode. Die neu implementierten Routinen können in die bestehende Programmstruktur eingefügt und bei Bedarf durch andere Algorithmen ersetzt werden. Die in SMART implementierten Datenstrukturen werden in ParSmart unverändert beibehalten und durch neue Strukturen ergänzt.

Das Konzept gleichberechtigter Knoten ermöglicht theoretisch die Skalierbarkeit für eine steigende Anzahl der Rechenknoten, wenn die Teilmodelle der einzelnen Rechenknoten gleiche Rechenzeit und Speicherkapazität benötigen und zusätzlich die ergänzenden Algorithmen für die Assemblierung der Gesamtmatrix und die Lösung der Gleichungssysteme skalierbar sind.

Obige Betrachtung zeigt, daß das gewählte Konzept den Anforderungen aus Abschnitt 4.1 gerecht werden kann.

4.4 Darstellung einer parallelen Berechnung mit ParSmart

Als Beispiel für eine parallele Berechnung wird der bereits in Abschnitt 3.4 beschriebene Ablauf einer seriellen Rechnung zugrundegelegt. Die parallele Version weicht nun dadurch ab, daß das Gesamtmodell auf die Rechenknoten aufgeteilt wird. In Bild 4.4.1 wird der Einsatz paralleler Rechenknoten durch die parallel angeordneten Pfeile in vertikaler Richtung angedeutet. Die neu eingeführten Algorithmen sind links neben dem Ablaufplan eingetragen.

Weiterhin verdeutlicht Bild 4.4.1, daß an dem Problemlösungsweg keine Änderungen vorgenommen werden und die neu implementierten Algorithmen das bisherige Konzept ergänzen. Diese neu eingeführten Algorithmen sind zum einen die Modellzerlegung vor der parallelen Berechnung, das Kommunikationsschema zur Assemblierung der Teilmatrizen und das parallele konjugierte Gradienten-Verfahren zur Lösung der Gleichungssysteme. Die zusätzlich erforderliche globale Kommunikation zur Bestimmung der Konvergenz kann durch die rechnerabhängigen *Message-passing*–Routinen des Rechners Intel/Paragon durchgeführt werden.

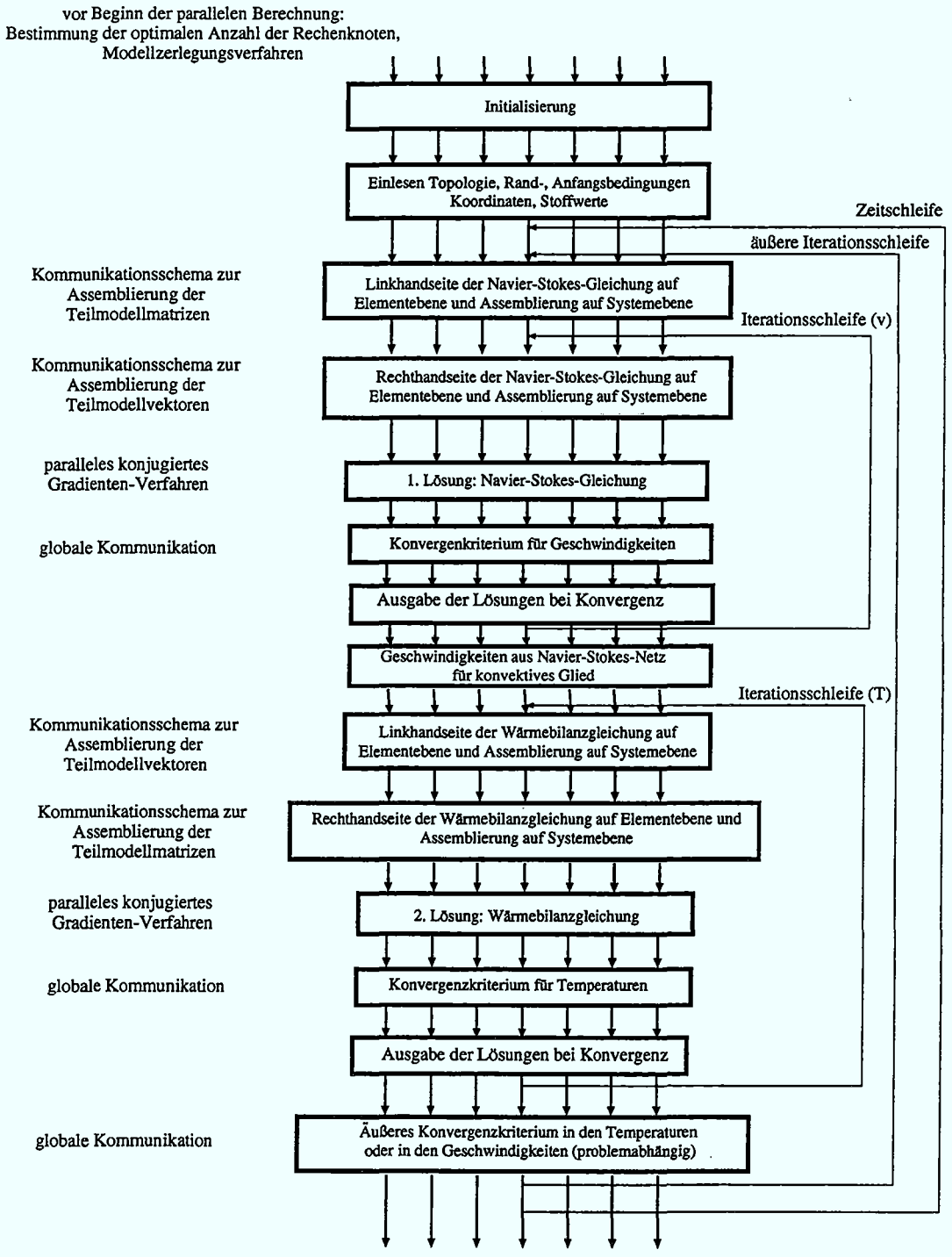


Bild 4.4.1: Ablaufdiagramm einer parallelen Berechnung mit ParSmart

5 Algorithmen und Datenstrukturen in ParSmart

Das entwickelte Programmsystem ParSmart erweitert den seriellen Code SMART um Routinen, die eine Parallelverarbeitung ermöglichen. Die ausgewählten und neu implementierten Algorithmen und Datenstrukturen werden nachfolgend beschrieben.

Das Konzept gleichberechtigter Knoten erfordert eine Modellzerlegung, die die Finite-Element-Daten auf die Rechenknoten aufteilt und somit eine Datenpartitionierung durchführt. Die in ParSmart implementierte Modellzerlegung wird in Abschnitt 5.1.1 beschrieben. Während die Elementbeiträge parallel für die Teilmodelle berechnet werden können, ist zur Assemblierung der Systemmatrix für das Gesamtmodell Kommunikation zwischen den Rechenknoten notwendig. Das in Abschnitt 5.1.2 vorgestellte Kommunikationsschema beschreibt den zur Assemblierung erforderlichen Datentransfer. Die Lösung der Gleichungssysteme erfolgt durch ein konjugiertes Gradienten-Verfahren, das in Abschnitt 5.1.3 erläutert wird.

Da in der Parallelversion zusätzliche Informationen, beispielsweise für die Vernetzung der Teilmodelle untereinander, verwaltet werden müssen, sind die Datenstrukturen aus SMART zu erweitern. Dieses geschieht durch Einführung von Zuordnungstabellen (Abschnitt 5.2.1), einem erweiterten Hypermatrizenschema (Abschnitt 5.2.2) sowie einem zeilenorientierten Speicherschema (Abschnitt 5.2.3).

In Abschnitt 5.3 werden die Implementierungsaspekte der Parallelversion ParSmart auf dem Rechner Intel/Paragon beschrieben und die Leistungszuwächse für die einzelnen Algorithmen diskutiert.

In der Praxis zeigt sich, daß ein Problem gegebener Größe nicht effizient auf einer beliebig hohen Anzahl von Rechenknoten berechnet werden kann. Dabei ist es möglich, daß die Ausführungszeiten ab einer gewissen Anzahl von Rechenknoten wieder ansteigen. Gründe dafür liegen beispielsweise in einem zu hohen Kommunikationsaufwand. Für ParSmart wird deshalb in Abschnitt 5.4 eine einfache Abschätzung für die Anzahl der Rechenknoten gegebenen, auf der ein maximaler Leistungszuwachs erreichbar ist.

5.1 Algorithmen für die entwickelte Parallelversion ParSmart

Nachfolgend werden die für ParSmart ausgewählten und implementierten Algorithmen beschrieben. Dieses sind insbesondere die Modellzerlegung, das Kommunikationsschema und das konjugierte Gradienten-Verfahren.

Neben der Beschreibung und Diskussion der Algorithmen wird erläutert, wie diese in das Programmsystem integriert werden. Dabei ist zu beachten, daß der strukturelle Aufbau von SMART auch in ParSmart erhalten bleibt und die Algorithmen in weiterführenden Arbeiten leicht verbessert oder ausgetauscht werden können.

5.1.1 Modellzerlegung

Die Parallelversion ParSmart basiert auf der Zerlegung des zu berechnenden Modells in Teilmodelle, die den einzelnen Rechenknoten des Parallelrechners zugeordnet werden. Ein ähnliches Vorgehen ist von der Unterstrukturtechnik bekannt, die es ermöglicht, ein komplexes Modell in Unterstrukturen zu zerlegen und zu berechnen, falls Speicher- und Rechenleistung unzureichend sind, um das Modell insgesamt zu berechnen [AM88].

Abschnitt 2.4.2 zeigt, daß eine Zerlegung für komplexe Geometrien von dem Anwender nicht einfach zu bestimmen ist, wenn sie die genannten Anforderungen erfüllen soll. In ParSmart wird deshalb ein Modellzerlegungsverfahren implementiert, welches das zu berechnende Gesamtmodell automatisch in Teilmodelle zerlegt. Jeder Rechenknoten bearbeitet dann ein solches Teilmodell. Durch die Zuordnung zu den Rechenknoten werden die Daten des Gesamtmodells auf den verteilten Speicher des Rechners Intel/Paragon abgebildet. Der Anwender muß diese nicht selbst verteilen.

Für ParSmart wird die Modellzerlegung durch den in [Far88] beschriebenen *Greedy*–Algorithmus durchgeführt. Der Vorteil dieses Algorithmus ist, daß er mit geringem Rechenaufwand für viele zwei- und dreidimensionale Diskretisierungen eine akzeptable Zerlegung bestimmt [Far88]. Diese kann dann als Ausgangspunkt für weitere Verbesserungen genutzt werden.

Der *Greedy*-Algorithmus zerlegt die durch Finite Elemente diskretisierte Geometrie, indem für jedes Teilmodell gleich viele, möglichst benachbarte Elemente zusammengefaßt werden. Somit erfolgt eine annähernd gleichmäßige Verteilung der Daten sowie der anschließend darauf auszuführenden Operationen [Far88]. Der Ablauf kann wie folgt beschrieben werden [Far88]:

for $i = 1$ to p^* do

(1) wähle einen Knoten n_i, für den gilt:

(1a) n_i gehört zum inneren Rand von D_{i-1}

(1b) n_i hat ein minimales Gewicht $\neq 0$

(2) initialisiere D_i mit allen nicht markierten Elementen, die mit Knoten n_i verbunden sind

(3) führe für jedes Element e_k aus D_i rekursiv durch:

(3a) markiere e_k

(3b) reduziere das aktuelle Gewicht jedes mit e_k verbundenen Knoten um Eins

(3c) verbinde alle nicht markierten Elemente mit e_k in D_i

(4) Abbruch des Verfahren für:
$[\#D_i = |N_e|/p^*]$ (+1 falls Rest $\neq 0$)

(5) markiere alle Knoten, die zum inneren Rand von D_i gehören

end

Das globale Modell D wird in p^* Teilmodelle $D_i, 1 \leq i \leq p^*$, zerlegt. p^* entspricht der Anzahl der Rechenknoten, auf denen das parallele Programm ausgeführt werden soll. D ist durch die Menge der Finite-Element-Knoten $N_n, n_i(x,y,z) \in N_n$ und der Menge der Finiten Elemente N_e mit $e_j(n_1, n_2, ..., n_k) \in N_e$ gegeben. Innere Randpunkte sind Finite-Element-Knoten n_i, die zu einem Teilmodell D_i und einem benachbarten Teilmodell D_j gehören. Das Gewicht eines Knotens wird definiert als die Anzahl der Elemente, zu denen dieser Knoten zugeordnet ist. Ein Knoten, der im Inneren des Teilmodells D_i liegt, besitzt das Gewicht Null, während Knoten auf dem inneren Rand ein von Null verschiedenes Gewicht aufweisen.

Die Schritte 3a-c werden listenweise abgearbeitet; sie beginnen jeweils mit dem ersten markierten Element des durchgeführten rekursiven Schrittes. Die Implementation des beschriebenen Algorithmus in FORTRAN wird ebenfalls in [Far88] angegeben. Für die Implementation in ParSmart ist das Verfahren für den Fall modifiziert worden, in dem ein Teilmodell aus nicht zusammenhängenden Elementmengen besteht.

Anhand nachfolgender Beispiele wird die Problematik dieser Modellzerlegung diskutiert. Dabei ist zu beachten, daß die Anzahl der die Teilmodelle miteinander verbindenden Finite-Element-Knoten die Höhe des Kommunikationsaufwandes für nachfolgend auszuführende Algorithmen bestimmt.

Bild 5.1.1 zeigt die Zerlegung eines regelmäßigen Modells in 2, 20 und 45 Teilmodelle. Die Diskretisierung wird hier durch 99x100 quadratische Elemente vorgenommen, die der Übersicht halber nicht eingezeichnet werden. Die Vorgehensweise des Algorithmus sichert

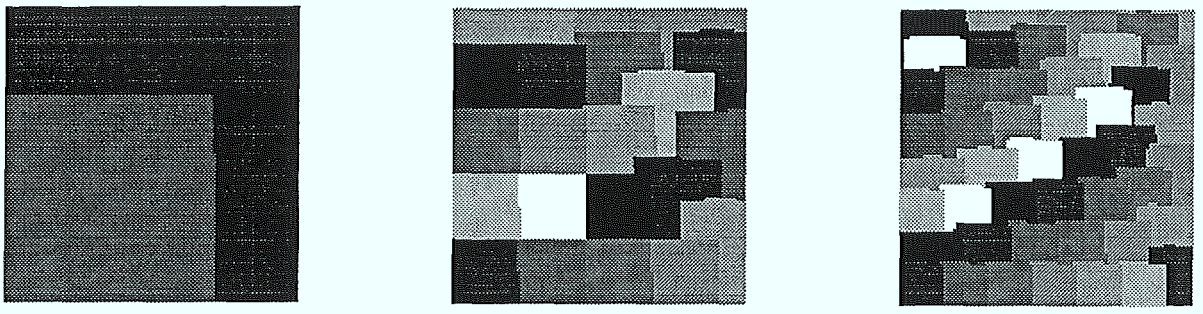


Bild 5.1.1: Zerlegung eines zweidimensionalen Modells

eine nahezu gleichmäßige Aufteilung der Elemente auf die Rechenknoten. Anhand dieser Beispiele wird jedoch die Problematik bei der Minimierung der inneren Randpunkte deutlich. So würde bei dem in zwei Teilmodelle zerlegten Gesamtmodell, eine waagerechte oder senkrechte Zerlegung eine kleinere Anzahl innerer Randpunkte ergeben. Wird das Gesamtmodell in 20 bzw. 45 Teilmodelle zerlegt, so wird deutlich, daß einige der Teilmodelle besonders viele Randpunkte aufweisen. In Extremfällen wird einem Rechenknoten ein Teilmodell zugeordnet, welches nicht aus einer zusammenhängenden Menge von Finiten Elementen besteht. Dieses erhöht die Menge der inneren Randpunkte und somit den Kommunikationsaufwand. Bei dem dargestellten, einfachen Modell könnte eine intuitive Zerlegung Teilmodelle mit weniger inneren Randpunkten bilden. Rechenzeit und Speicherbelegung werden jedoch nahezu gleichmäßig verteilt. Der Vorteil des Algorithmus besteht darin, daß er auch für komplexe

Modelle anwendbar ist, wie beispielsweise für das in Bild 5.1.2 gezeigte, dreidimensionale Modell und eine geringe Rechenzeit benötigt.

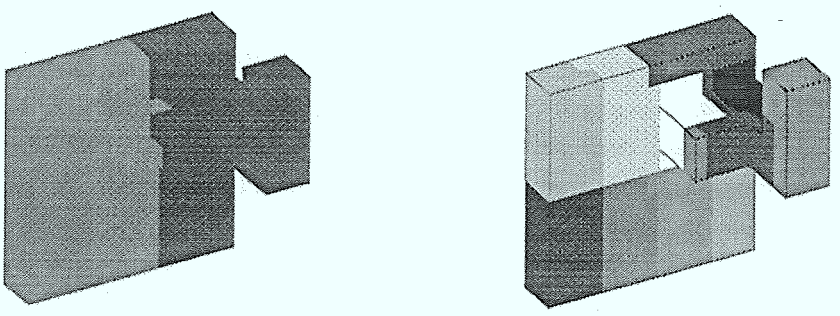


Bild 5.1.2: Zerlegung eines dreidimensionalen Modells

Für das noch in Abschnitt 6.3 zu betrachtende Modell des umströmten Laminarflügels wird für zehn Rechenknoten die folgende Zerlegung ermittelt:

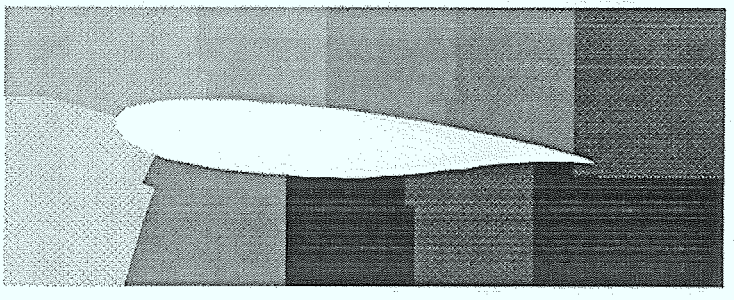


Bild 5.1.3: Zerlegung eines Tragflügelmodells

In ParSmart wird dieses Verfahren integriert, indem zunächst die Element-Diskretisierung aus den SMART-Eingabedateien eingelesen und in eine für dieses Modellzerlegungsverfahren geeignete Datenstruktur umgespeichert wird. Auf dieser Datenstruktur erfolgt die Partitionierung in Teilmodelle. Anschließend werden für diese Teilmodelle automatisch die verteilten ParSmart Eingabedatensätze erstellt. Diese Datensätze können dann durch die Preprozessoren parallel für die ParSmart-Rechnung aufbereitet werden.

Für die parallele Berechnung ist zu beachten, daß diese Modellzerlegung durch die Anzahl der inneren Randpunkte bereits die Komplexität der Kommunikation für die später auszuführenden Algorithmen bestimmt. Durch die inneren Randpunkte, die Verknüpfung der

Teilmodelle und die Zuordnung dieser Teilmodelle zu den Rechenknoten wird für die Assemblierung der Gesamtsystemmatrix festgelegt, welche Rechenknoten welche Daten untereinander austauschen müssen.

5.1.2 Kommunikationsschema

Die Assemblierung der Systemmatrix einer Finite-Element-Diskretisierung wird in SMART durchgeführt, indem für jeden Knotenpunkt die Einträge der entsprechenden Elementmatrizen addiert werden. Im Rahmen der Parallelversion erfolgt diese Assemblierung in zwei Stufen. Dabei assembliert zunächst jeder Rechenknoten diejenigen Elementmatrizen, die er zuvor aufgestellt hat. In dieser ersten Phase wird auf jedem Rechenknoten der serielle SMART-Code ausgeführt. Es ist keine Kommunikation erforderlich, d.h. jeder Rechenknoten arbeitet auf seinen lokalen Daten. In der zweiten Stufe der Assemblierung müssen Daten zwischen den Rechenknoten ausgetauscht werden, um die Abhängigkeiten der Teilmodelle untereinander zu berücksichtigen. Dieser Datentransfer wird dem nachfolgenden Kommunikationsschema entsprechend durchgeführt.

In dem Gesamtkonzept der Parallelversion ParSmart ist vorgesehen, daß jeder Rechenknoten diejenigen Zeilen der Gesamtmatrix speichert, die sich den Knotennummern der von ihm verwalteten Elemente zuordnen lassen. Um eine redundante Speicherung und Berechnung für Matrixzeilen zu vermeiden, die zu inneren Randpunkten gehören und somit auf verschiedenen Rechenknoten vorhanden sind, werden diese Zeilen jeweils dem Rechenknoten mit der kleinsten Nummer zugeordnet.

Für eine solche Matrixpartitionierung, bei der die Zeilen auf die Rechenknoten verteilt sind, läßt sich ein paralleles konjugiertes Gradienten-Verfahren effizient ausführen, wie im nachfolgenden Abschnitt gezeigt wird.

Die Kommunikation zur globalen Assemblierung und zeilenweisen Aufteilung der Systemmatrix auf die Rechenknoten wird nachfolgend an einem einfachen Beispiel erläutert. Dazu wird ein regelmäßiges Finite-Element-Netz entsprechend Bild 5.1.4 betrachtet.

In der linken Bildhälfte wird die Finite-Element-Diskretisierung des Gesamtmodells mit den globalen Knotenpunktnummern und in der rechten die Aufteilung der Teilmodelle auf die Rechenknoten mit den zugehörigen Datenströmen für die Assemblierung der Systemmatrix und zeilenweisen Speicherung für das Gesamtmodell dargestellt. Dabei bildet $R_i = \{3, 8, 11, 12, 13, 14, 15, 18, 23\}$ die Menge der inneren Randpunkte, für die im zweiten Schritt der Assemblierung Kommunikation erforderlich ist.

In ParSmart werden die Zeilen der Gesamtmatrix jeweils auf dem Rechenknoten mit kleinster Nummer gespeichert, der einen Eintrag in diese Zeile liefert. Für das betrachtete Beispiel bedeutet dies, daß der Rechenknoten 1 die Zeilen $\{1, 2, 3, 6, 7, 8, 11, 12, 13\}$ der Gesamtmatrix speichern soll. Dazu werden Werte für die Elementknoten $\{3, 8, 11, 12, 13\}$ von den benachbarten Teilmodellen benötigt.

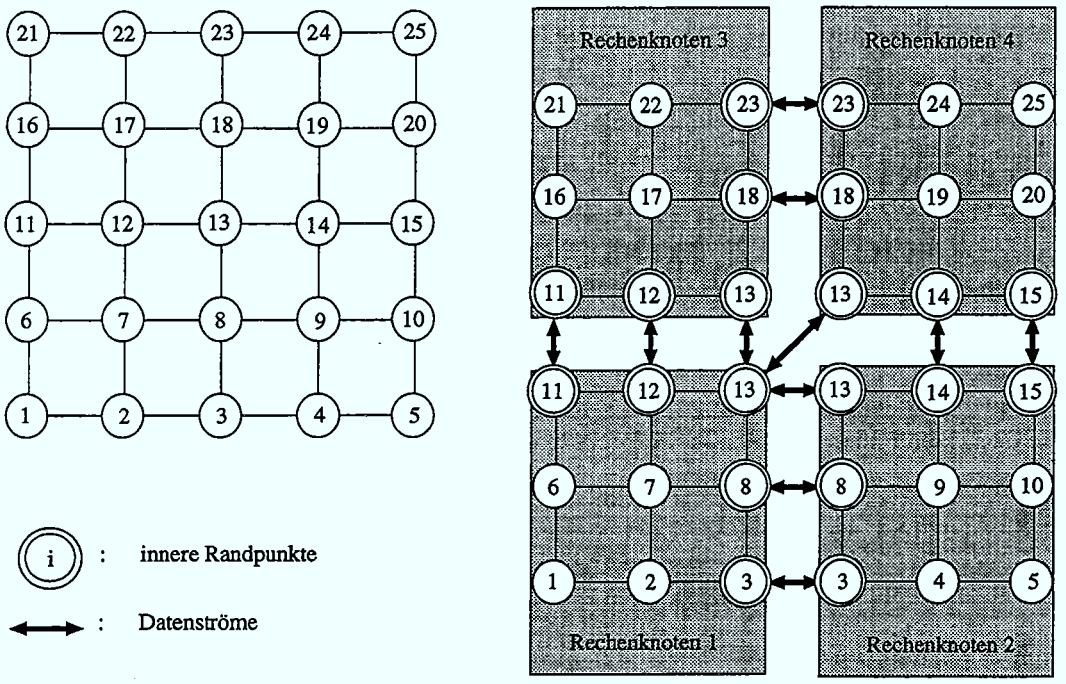


Bild 5.1.4: Diskretisierung und Partitionierung

Die Kommunikation wird zunächst so durchgeführt, daß jeder Rechenknoten Zeilen seiner lokalen, assemblierten Matrix an die Rechenknoten mit der kleinsten Nummer verschickt, auf dem Einträge für diese Zeilen gespeichert werden:

- Rechenknoten p_s $(1 < p_s \leq p^*)$ sendet die zu inneren Randpunkten zuzuordnenden Zeilen der lokal assemblierten Matrix an den Rechenknoten mit kleinster Nummer, auf dem Einträge in diese Zeilen gespeichert werden.

- Rechenknoten p_e $(1 \leq p_e < p_s)$ erwartet Matrixzeilen für innere Randpunkte von Rechenknoten mit höherer Nummer, um die lokalen Matrixzeilen zu berechnen.

Anschließend werden die assemblierten Einträge an die Rechenknoten mit größeren Nummern zurückgeschickt. Bei dieser Vorgehensweise ist zu beachten, daß die Empfängerknoten nach der Kommunikation Zeilen der Gesamtmatrix verwalten.

- Rechenknoten p_s $(1 < p_s \leq p^*)$ sendet die zu inneren Randpunkten zuzuordnenden Zeilen der lokal assemblierten Matrix an die Rechenknoten mit größeren Nummern zurück.

- Rechenknoten p_e $(1 < p_e < p_s)$ erwartet Matrixzeilen für innere Randpunkte vom Rechenknoten mit kleinster Nummer, der diese Zeilen verwaltet, um die lokalen Matrixzeilen zu berechnen.

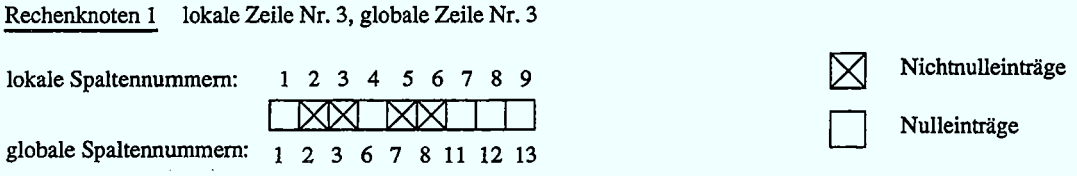
In Bild 5.1.5 wird die Kommunikation veranschaulicht, die erforderlich ist, um die Zeile 3 der Gesamtmatrix auf Rechenknoten 1 zu speichern. Dazu wird zunächst die Besetzungsstruktur dieser Zeile nach der Assemblierung der Elemente des Teilmodells gezeigt. Es sind jeweils lokale und globale Zeilen- und Spaltennummern angegeben.

Damit die bislang lokal aufgestellte Zeile 3 auf Rechenknoten 1 der Zeile 3 der Gesamtmatrix entspricht, müssen Daten von Rechenknoten 2 erhalten werden. Dieses sind die lokalen Einträge von Rechenknoten 2 für Zeile 3 der Gesamtmatrix. Nachdem Rechenknoten 1 diese Einträge bekommen hat wird die bislang lokale Zeile so ergänzt, daß sie der Zeile 3 der Gesamtmatrix entspricht. Dazu werden die Spalteneinträge in der lokalen Hypermatrix durch Addition assembliert, im Beispiel die Spalten 3 und 8. Spalten, die bislang nicht in der lokalen Hypermatrix gespeichert wurden (4 und 9) werden in einem Feld, das Zeilen- und Spaltennummer, sowie den numerischen Wert speichert, abgelegt. Dieses erweiterte Hypermatrizenschema wird erforderlich, da neue Spalten nur unter erheblichem Aufwand in das bestehende Hypermatrizenschema eingefügt werden können.

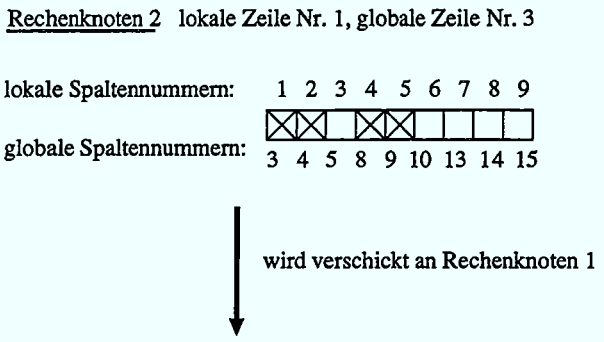
Da in dem SMART-Hypermatrizenschema nur die obere Dreiecksmatrix gespeichert wird, bedeutet die Addition der Zeileneinträge auf Rechenknoten 1 zugleich die vollständige Ergänzung der lokal gespeicherten Matrixzeilen $\{2, 3, 7\}$ für die Gesamtmatrix auf Rechenknoten 1.

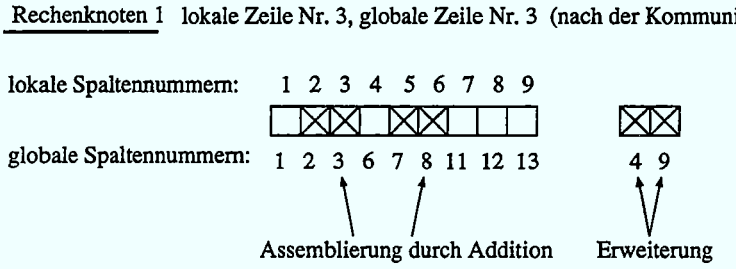
In Bild 5.1.5 werden jeweils die lokalen und die globalen Spaltennummern angegeben. Dieses ist erforderlich, da die Numerierung der Finite-Element-Knoten der Teilmodelle in ParSmart lokal jeweils mit 1 beginnend erfolgt. Die Abbildung der lokalen auf die globalen Spaltennummern erfolgt wie in Abschnitt 5.2.1 erläutert, durch Zuordnungstabellen.

Nach der beschriebenen zweistufigen Kommunikation sind die Zeilen der Systemmatrix des Gesamtmodells auf die Rechenknoten verteilt und werden dort zunächst in dem erweiterten Hypermatrizenschema gespeichert.



Um auf Rechenknoten 1 die Zeile 3 der Gesamtmatrix zu speichern, ist folgender Datentransfer von Rechenknoten 2 zu Rechenknoten 1 erforderlich:





Besetzungsstruktur der Matrixzeile 3 der globalen Systemmatrix:

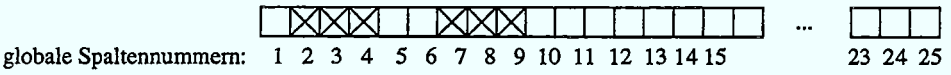


Bild 5.1.5: Besetzungsstruktur einer globalen und einer lokalen Matrixzeile

5.1.3 Lösung der Gleichungssysteme

Das Lösungsverfahren zur Berechnung der etwa 90% der Gesamtrechenzeit beanspruchenden Gleichungssysteme, muß effizient auf dem massiv parallelen Rechner Intel/Paragon ausführbar sein. Um die in Abschnitt 4.1 zusammengestellten Anforderungen näherungsweise erfüllen zu können, wird das Cholesky-Verfahren durch ein iteratives konjugiertes Gradienten-Verfahren (CG, conjugate gradients) ersetzt. Dieses wurde erstmals in [HS52] vorgeschlagen und beispielsweise in [AÖS90] bezüglich der Ausführung auf Parallelrechnern untersucht.

Für ParSmart wird ein konjugiertes Gradienten-Verfahren für **symmetrische** Gleichungssysteme eingesetzt, da bei der Parallelisierung bestehender Programmsysteme die Wechselwirkungen zwischen Programm- und Datenstrukturen besonders zu berücksichtigen sind. Für die Parallelisierung des Programmsystems SMART würde der Einsatz eines Lösungsverfahrens für **asymmetrische** Gleichungssysteme [HS52] bedeuten, daß das Speicherschema der Hypermatrizen nicht übernommen werden kann. In diesem Speicherschema wird aus Symmetriegründen nur die obere Dreiecksmatrix gespeichert. Der Einsatz eines Lösungsverfahrens für asymmetrische Gleichungssysteme würde somit einen höheren Speicherplatzbedarf erfordern, da in diesem Fall eine asymmetrische Matrix zu speichern wäre. Dieser zusätzliche Speicherbedarf kann dann zu erhöhtem Datentransfer zum Hintergrundspeicher führen.

Der Einsatz eines Lösungsverfahrens für asymmetrische Gleichungssysteme würde jedoch nicht nur ein neues Speicherkonzept, sondern auch andere Elementansätze erfordern, Abschnitt 3.1.2, und das Gesamtkonzept des Programmablaufes erheblich verändern. Die Bedingungen zur Parallelisierung von SMART, bewährte und verifizierte Programmteile weiterhin zu verwenden, könnten bei der genannten Vorgehensweise nicht erfüllt werden.

Durch den Wechsel vom direkten Cholesky-Verfahren zum iterativen konjugierten Gradienten-Verfahren wird ein effizient parallel ausführbares Verfahren eingesetzt, welches ermöglicht, die bisherigen, gut verifizierten Programm- und Datenstrukturen beizubehalten.

In Tabelle 7 ist das **serielle** Verfahren nach [AÖS90] angegeben, zusätzlich werden die **parallelen** Berechnungschritte bezüglich Lokalität und Kommunikationsgehalt gekennzeichnet. Um den Algorithmus auf das globale System anzuwenden, müssen für die Berechnung der Skalarprodukte globale Summationen durchgeführt werden, ebenso für die Bestimmung des globalen Vektors d_i zur Matrix-Vektor-Multiplikation. Die globalen Summationen, die erforderlich sind, um die Skalarprodukte zu bestimmen und die Konvergenz zu überprüfen, werden durch eine Umstellung der Rechenschritte so zusammengefaßt, daß je Iteration nur eine globale Summation erforderlich ist. In der Praxis zeigt sich, daß hinreichende Fehlerschranken bereits nach deutlich weniger Iterationen erreicht werden. Maximal werden, bei Vernachlässigung von Rundungsfehlern n Iterationsschritte bis zur Konvergenz des Verfahrens benötigt [HS52].

Für eine iterative Lösung der Gleichungssysteme auf einem Parallelrechner muß ein verteiltes Speicherschema vorliegen, welches schnelle Zugriffe auf die einzelnen Matrixeinträge ermöglicht [Bas93]. Im Hypermatrizenschema muß in jedem Iterationsschritt über drei Stufen

auf die einzelnen numerischen Einträge in der Matrix zugegriffen werden, so daß eine aufwendige Adressierungsberechnung erforderlich ist. Dabei müssen für den Zugriff auf einen Eintrag drei Adressen bestimmt werden, hinzu kommt weiterer Aufwand zur Ermittlung der Matrixpositionen in der Datenbasis. In einem zeilenorientierten Speicherschema kann hingegen direkt auf die Matrixeinträge zugegriffen werden. Deshalb wird hier ergänzend ein zeilenorientiertes Speicherschema eingeführt, das in Abschnitt 5.2.3 erläutert wird [Bas93].

seriell [AÖS90]	parallel	
	lokale Anweisungen	Kommunikation
$g_0 = Ax_0 - b$ $d_0 = -g_0$	$g_0^p = A_0^p x_0 - b^p$ $d_0^p = -g_0^p$	x_0 bestimmen
$i = 0, 1....$ $\alpha_i = \frac{g_i^T g_i}{d_i^T A d_i}$	$h_{i,p}^1 := (g_i^p)^t g_i^p;$ $h_{i,p}^2 := (d_i^p)^t (A^p d_i)$ $\alpha_i = h_i^1 / h_i^2$	d_i bestimmen $h_i^1 := \sum_{p=0}^{n-1} h_{i,p}^1$ $h_i^2 := \sum_{p=0}^{n-1} h_{i,p}^2$
$\beta_i = \frac{\alpha_i (Ad_i)^T A d_i}{d_i^T A d_i} - 1$	$h_{i,p}^3 := (A^p d_i)^t A^p d_i$ $h_{i,p}^4 := d_i^t A^p d_i$ $\beta_i = \alpha_i h_i^3 / h_i^4 - 1$	$h_i^3 := \sum_{p=0}^{n-1} h_{i,p}^3$ $h_i^4 := \sum_{p=0}^{n-1} h_{i,p}^4$
$g_{i+1}^t g_{i+1} = \beta_i g_i^T g_i$	$h_{i,p}^5 := (g_i^p)^t g_i^p$ $g_{i+1}^T g_{i+1} = \beta_i h_i^5$	$h_i^5 := \sum_{p=0}^{n-1} h_{i,p}^5$
$x_{i+1} = x_i + \alpha_i d_i$ $g_{i+1} = g_i + \alpha_i A d_i$ $d_{i+1} = -g_{i+1} + \beta_i d_i$	$x_{i+1}^p = x_i^p + \alpha_i d_i$ $g_{i+1}^p = g_i^p + \alpha_i A d_i$ $d_{i+1}^p = -g_{i+1}^p + \beta_i d_i$	
Anmerkung:	$h_{i,p}^k, k = 1, 5$: lokale Hilfsvariablen $h_i^k, k = 1, 5$: globale Hilfvariablen	

Tabelle 7: Serielles und paralleles CG-Verfahren

Den schnelleren Zugriffszeiten auf einzelne Matrixeinträge in dem zeilenorientierten Speicherschema steht der Nachteil gegenüber, daß veränderte Linkhandseiten in das neue Speicherschema zu konvertieren sind. Der zusätzliche Aufwand des Umspeicherns ist bei dem Einsatz eines iterativen Lösungsverfahrens vertretbar, da er die aufwendigen Zugriffsmechanismen auf die Einträge der Hypermatrizen erheblich reduziert. Weiterhin kann dieses lokal auf den Rechenknoten durchgeführt werden. Das erweiterte Hypermatrizenschema sowie das zeilenorientierte Speicherschema werden in Abschnitt 5.2.2 bzw. 5.2.3 erläutert.

Zur Verbesserung der Konvergenz bei der Lösung des Gleichungssystems $Ax = b$ wird eine Vorkonditionierung

$$\underbrace{(M^{-1}AM^{-1})}_{\tilde{A}}\underbrace{(Mx)}_{\tilde{x}} = \underbrace{M^{-1}b}_{\tilde{b}} \tag{5.1.1}$$

mit geeigneter Matrix M durchgeführt [DH91]. In ParSmart wird dazu eine Diagonalskalierung implementiert, bei der die Diagonaleinträge der Matrix $\tilde{A}$ zu Eins normiert werden, d.h.

$$M = \tilde{D} := \begin{cases} \sqrt{a_{ij}} & i = j \\ 0 & i \neq j \end{cases} \tag{5.1.2}$$

Diese Vorkonditionierung wird auf jedem Rechenknoten lokal ausgeführt. Weitere Verbesserungen der Matrixkondition sind beispielsweise durch die in [Bas95] untersuchten Vorkonditionierungsverfahren möglich.

5.2 Ergänzende Datenstrukturen

Die in SMART implementierten Datenstrukturen, insbesondere die in Abschnitt 3.3.2 beschriebenen Hypermatrizen und Kontrollisten, werden für die Parallelversion ParSmart übernommen. Die neu implementierten Algorithmen erfordern zur effizienten Funktionalität Erweiterungen, die in diesem Abschnitt vorgestellt werden.

Die Systemmatrix für das Gesamtmodell ist in ParSmart zeilenweise auf die Rechenknoten aufgeteilt. Diese Vorgehensweise erfordert es, die Hypermatrizen, welche die lokalen Teilmatrizen verwalten, geeignet zu erweitern. Ferner ist es zur effizienten Anwendung eines iterativen Lösungsverfahrens erforderlich, von dem Speicherschema der Hypermatrizen zu einem zeilenorientierten Speicherschema überzuwechseln.

5.2.1 Zuordnungstabellen

Die auf einer Modellzerlegung basierende Parallelisierungsstrategie erfordert den Austausch numerischer Daten zwischen denjenigen Finite-Element-Knoten, die Teilnetze miteinander verbinden. Da SMART in den Systemmatrizen nur Elementknoten mit lokalen Freiheitsgraden berücksichtigt, wird intern die Numerierung der lokalen Freiheitsgrade der Finite-Element-Knoten für jedes Teilmodell von Eins beginnend aufwärts durchgeführt. Um

den Datenaustausch zwischen den Rechenknoten zu ermöglichen, ist somit eine rechenknotenübergreifende Numerierung erforderlich. In ParSmart werden dafür ergänzend zu den SMART-Kontrollisten Zuordnungstabellen angelegt, welche diese Abbildung zwischen der lokalen und der globalen Numerierung ermöglichen. Diese Tabellen werden für das Geschwindigkeits- und für das Temperaturnetz als Felder angelegt. Sie werden von den Kommunikationsroutinen benötigt, die für die Zeilen- und Spaltennummern die globalen Werte versenden.

5.2.2 Erweitertes Hypermatrizenschema

Zur Aufstellung der prozessorübergreifenden Gesamtmatrix müssen die Zeilen, die lokal auf den Rechenknoten gespeichert werden und nur eine Zeile der lokalen Systemmatrix beinhalten, um Einträge der benachbarten Rechenknoten erweitert werden. Die lokale Systemmatrix ist wie im seriellen SMART-System im Hypermatrizenschema gespeichert. Dabei ist anzumerken, daß unabhängig von der Numerierung der Finite-Element-Knoten diese Hypermatrizen nur Zeilen/Spalten mit mindestens einem Nichtnulleintrag speichern. In der Kommunikationsroutine werden diese lokalen Matrixzeilen nun so ergänzt, daß sie einer Zeile der Gesamtmatrix entsprechen. Muß eine Zeile einen Spalteneintrag speichern, der nicht in der lokalen Matrix vorhanden ist, so geschieht das in einem Feld, welches die lokalen Hypermatrizen ergänzt. Dazu werden globale Zeilen- und Spaltennummern sowie die entsprechenden numerischen Werte gespeichert. Grund dafür ist der hohe, nicht vertretbare Aufwand zum Einfügen neuer Spalten in die Hypermatrix, insbesondere, da diese für die Lösung der Gleichungssysteme in das zeilenorientiertes Speicherschema umgespeichert werden sollten.

5.2.3 Zeilenorientiertes Speicherschema

Die Vorteile eines iterativen Verfahrens zur Lösung der Gleichungssysteme, die insbesondere in guten Parallelisierungseigenschaften bestehen, können nur genutzt werden, wenn ein effizientes Speicherschema vorliegt [Bas95]. Testrechnungen, in denen ein paralleles CG-Verfahren auf Hypermatrixbasis zugrundegelegt wurde, zeigten eine schlechte Effizienz auf dem Parallelrechner, da in jedem Iterationsschritt die aufwendige Adreßrechnung zur Bestimmung der numerischen Einträge in der Hypermatrix durchgeführt werden mußte.

Für eine iterative Lösung in ParSmart werden die Linkhandseiten deshalb von dem Hypermatrizenschema in ein zeilenorientiertes Speicherschema umgespeichert. Dieses verkürzt die Zugriffszeiten auf die Matrixeinträge bei der iterativen Lösung. In dem neuen Schema wird die Matrix in einem zweidimensionalen Feld verwaltet, in dem in einer Zeile die Nichtnullelemente aufeinanderfolgend abgespeichert werden. In einer weiteren Indexmatrix werden die entsprechenden Spalteneinträge gespeichert [Bas95]. Dieses Schema ermöglicht eine weitere Speicherplatzreduzierung für dünnbesetzte Matrizen gegenüber dem Hypermatrizenschema, da nur wenige Nullelemente gespeichert werden.

Bild 5.2.1 zeigt ein Beispiel für die Speicherung einer dünnbesetzten Matrix im zeilenorientierten Schema.

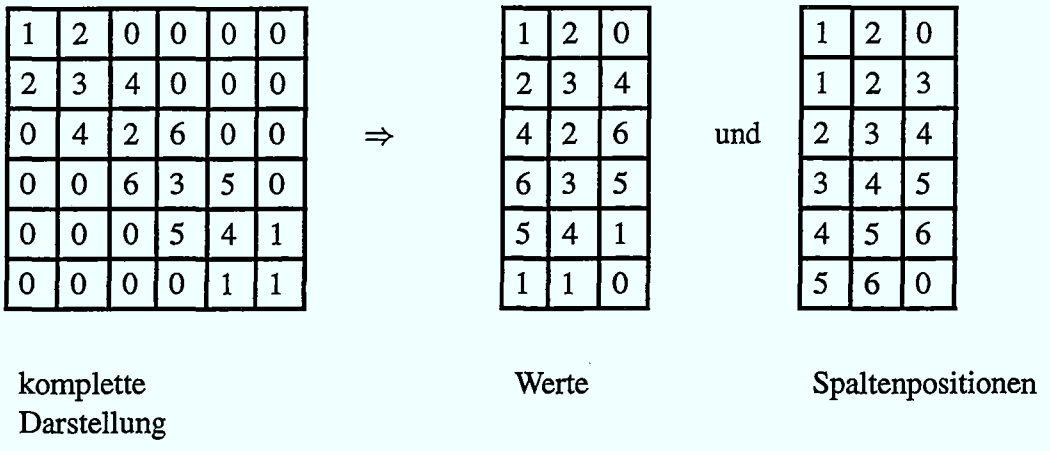


Bild 5.2.1: Zeilenorientiertes Speicherschema

Die Konvertierung der Speicherung erfolgt dabei lokal auf jedem Rechenknoten des Parallelrechners, indem die Matrixzeilen aus dem erweiterten Hypermatrizenschema ausgelesen und in das zeilenorientierte Format gespeichert werden. Theoretisch ist dabei linearer Speedup erreichbar. Eine Minderung des maximalen Speedups kann sich jedoch durch eine verschiedenartige Besetzungsstruktur der Hypermatrizen für die Teilmodelle und somit durch unterschiedliche Zugriffszeiten auf Matrixeinträge ergeben. Der Speicherbedarf, und damit die Zugriffszeit auf einzelne Einträge der lokalen Hypermatrizen wird durch die Numerierung der Elementknoten und somit auch durch die Bandbreite der Matrizen bestimmt. In dem zeilenorientierten Speicherschema hingegen wirkt sich die Bandbreite weder auf die Anzahl der zu speichernden Einträge noch auf das darauf auszuführende konjugierte Gradienten-Verfahren aus.

5.3 Implementierungsaspekte

Für die Parallelversion ParSmart ist der serielle SMART-Code um Routinen zur parallelen Berechnung erweitert worden. Um die systemspezifischen Eigenschaften des Rechners Intel/Paragon weitgehend auszunutzen, wurden für diesen Rechner spezifische Optimierungen vorgenommen, wobei Einschränkungen bezüglich der Portabilität auf andere Parallelrechnersysteme entstehen können. Insbesondere wird das in Abschnitt 2.3 beschriebene, NX-basierte *Message-passing* angewendet.

Das in SMART implementierte Datenverwaltungssystem erfordert bei der gewählten Parallelisierungsstrategie für jeden Rechenknoten eine Datenbasis auf dem Hintergrundspeicher, auf die während der Laufzeit lesend und schreibend zugegriffen wird. Das unzureichende Verhältnis von 140 Rechen- zu 5 IO-Knoten des Rechners Intel/Paragon führt hier zu massiven Problemen [ZB95]. Deshalb wurde für SMART ein erweitertes Konzept zur Datenverwaltung implementiert, das im folgenden vorgestellt wird.

Der Leistungszuwachs durch den Parallelrechnereinsatz wird weitgehend durch das Kommunikationsschema und das Lösungsverfahren beeinflußt. Neben der Konvertierung der Systemmatrizen werden diese deshalb hinsichtlich Leistungszuwachs und Skalierbarkeit untersucht.

5.3.1 Erweitertes Konzept zur Datenverwaltung und IO-Reduktion

Bei der Entwicklung des Programmsystems SMART ist insbesondere berücksichtigt worden, daß viele für die Praxis interessante Probleme nicht während der gesamten Laufzeit im Hauptspeicher zu verwalten und zu lösen sind. Es ist daher das in Abschnitt 3.3.2 beschriebene, umfangreiche Datenverwaltungssystem konzipiert worden, welches es ermöglicht, die erforderlichen Daten während eines Programmlaufes in einer Hintergrundspeicherdatei zu verwalten. In dem für ParSmart entwickelten Konzept wird vorgesehen, daß jeder Rechenknoten seine eigene Datenbasis verwaltet. Massiv parallele Supercomputer wie der Intel/Paragon erweitern zwar den Hauptspeicherbereich, der sich aus den Speichern der einzelnen Rechenknoten zusammensetzt, viele Probleme können jedoch auch bei diesem erweiterten Hauptspeicherbereich noch nicht vollständig im Hauptspeicher gelöst werden. Für das in Abschnitt 6.1 beschriebene Beispiel der Rayleigh-Bénard-Konvektion ergeben sich DRS (Data Retrieval System) –Dateien der in Bild 5.3.1 dargestellten Größe je Rechenknoten. Ab etwa 4 Rechen-

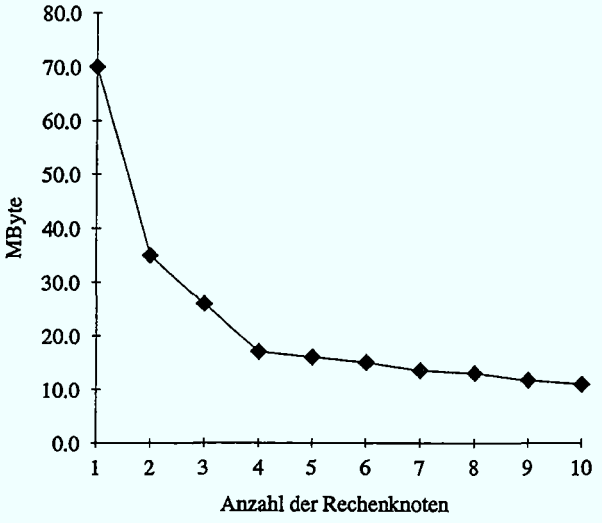


Bild 5.3.1: Größe der DRS-Dateien

knoten nimmt die Größe der den Rechenknoten zugeordneten Dateien nur noch geringfügig ab.

Bei Nutzung des auf dem Intel/Paragon vorhandenen parallelen Dateisystems (PFS, Parallel File System), vgl. Abschnitt 2.3, entsteht bei den Zugriffen auf die Datenbasis ein solcher Engpaß, daß bereits auf 45 Knoten für die Dateninitialisierung erheblich mehr Zeit

als im seriellen Fall benötigt wird. In Tabelle 8 wird der Zeitverbrauch für die Dateninitiali-

Rechenknoten	Zeitverbrauch	
	Minimum (s) (Rechenknoten mit dem geringsten Zeitverbrauch)	Maximum (s) (Rechenknoten mit dem höchsten Zeitverbrauch)
1	234	-
5	91	93
10	65	89
15	79	106
20	78	122
25	70	176
30	40	206
35	27	227
40	31	377
45	31	402

Tabelle 8: Zeiten zur Dateninitialisierung

sierung paralleler Berechnungen zusammengefaßt. Da jeder Rechenknoten seine Datenbasis auf dem Hintergrundspeicher aufstellt, entsteht ein Engpaß duch die 5 IO Knoten des Intel/Paragon. Der minimale Zeitverbrauch gibt an, wann der schnellste Rechenknoten die Initialisierungsphase abgeschlossen hat. Da maximal 5 Rechenknoten parallel auf den Hintergrundspeicher zugreifen können, entstehen hier Wartezeiten, wenn zuviele Rechenknoten Datentransfer durchführen müssen. Der maximale Zeitverbrauch gibt nun an, wann alle an einer parallelen Berechnung beteiligten Rechenknoten den Datentranfer zum Hintergrundspeicher abgeschlossen haben. Diese Zeitspanne muß der Anwender für die Dateninitialisierung einrechnen.

Für ParSmart wurde aus diesem Grund zu dem bestehenden Konzept der **Datenverwaltung auf dem Hintergrundspeicher** zusätzlich ein weiteres Konzept entwickelt und implementiert, welches alternativ zu dem Standard-SMART-Verwaltungssystem eingesetzt werden kann und insbesondere die Nutzung der verfügbaren 140 Rechenknoten ermöglicht.

Das alternative Konzept verzichtet auf die Speicherung der Daten in explizit angelegten Dateien auf dem Sekundärspeicher und **simuliert quasi eine vollständige Rechnung im Hauptspeicher.** Der vom Programmierer verfügbare Hauptspeicher je Rechenknoten beträgt nach Abschnitt 2.3 ca. 25 MByte, so daß die Datenbasis bereits bei 4 Rechenknoten im Hauptspeicher gehalten werden kann. Die Datenorganisation bleibt in der bekannten Form erhalten, jeder Lese-/Schreibbefehl auf den Hintergrundspeicher wird durch einen entsprechenden Zugriff auf ein im Hauptspeicher angelegtes Feld ersetzt. Da beispielsweise bei einem seriellen Lauf die Datei der Größe 70 MByte nicht im Hauptspeicher verwaltet werden kann,

wird ein Paragon *Paging* genutzt, welches die Daten über Betriebssystemfunktionen auslagert. Auch hier ist Datentransport vom und zum Hintergrundspeicher erforderlich. Er ist jedoch wesentlich geringer als der in SMART implementierte Datentransfer, bei dem stets *Seiten* mit einer maximalen Größe von 32KByte transportiert werden können. Dieses neue Konzept ermöglicht Berechnungen auf 140 Rechenknoten, bei denen nur noch das erforderliche Einlesen der Dateien mit den Eingabedaten zu Engpässen aufgrund unzureichender IO-Leistung führt. Bei instationären Berechnungen ist die Dateninitialisierungsphase vernachlässigbar, da der Zeitverbrauch in der Zeitschleife nach Tabelle 6 überwiegt. Bei einer Konfiguration des Rechners Intel/Paragon mit 128 MByte Hauptspeicher je Rechenknoten würde auch für größere Beispiele kein Paragon *Paging* mehr benötigt.

In Bild 5.3.2 wird der Zeitverbrauch für die durchschnittliche Dateninitialisierung im Standard-SMART System und für die alternative Initialisierung in ParSmart gegenübergestellt. Der Vorteil der Berechnung im Hauptspeicher mit *Paging* des Intel/Paragon ist offensichtlich.

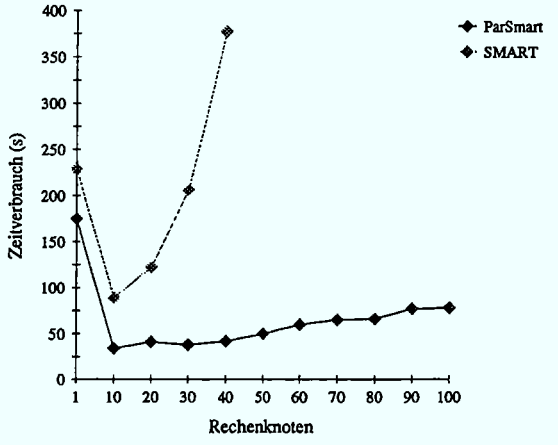


Bild 5.3.2: Vergleich der Dateninitialisierung in Standard-SMART mit dem neu entwickelten Konzept für ParSmart

Daß auch bei dem für ParSmart entwickelten Konzept ein Anstieg der Rechenzeiten zur Dateninitialisierung auftritt, ist darauf zurückzuführen, daß sich der Datentransfer vom Hintergrundspeicher nicht vollständig vermeiden läßt, da die Eingabedatensätze für die Teilmodelle stets von diesem eingelesen werden müssen.

Bild 5.3.3 zeigt die Größe der einzulesenden Teilmodellbeschreibungen für die Rechenknoten. Neben der Größe der Dateien je Rechenknoten wird die Gesamtmenge der zu übertragenden Daten gezeigt. Diese Datenmenge steigt mit der Anzahl der Teilmodelle, so daß auch der gesamte Zeitverbrauch zur Initialisierung ansteigt.

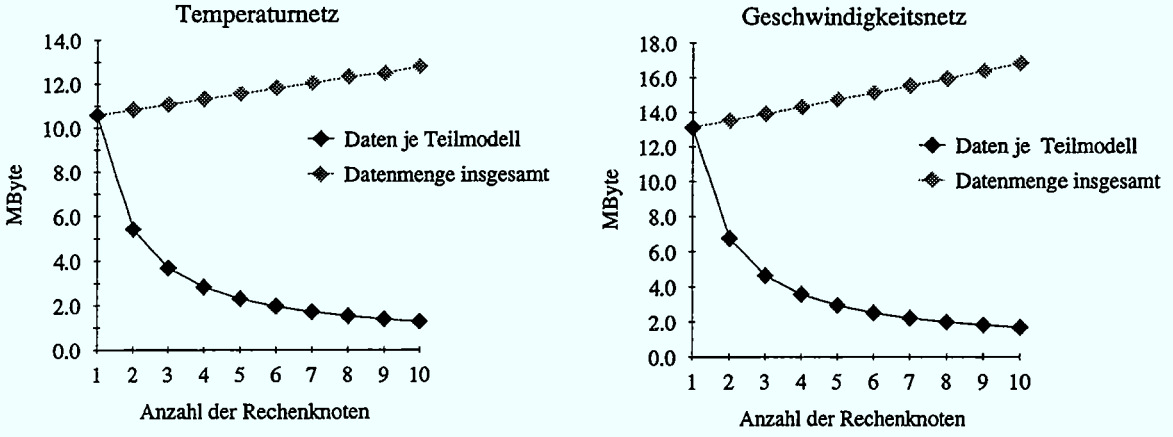


Bild 5.3.3: Größe der Eingabedateien für Temperatur- und Geschwindigkeitsnetz

Es ist anzumerken, daß in Bild 5.3.2 nur der Zeitverbrauch zur Dateninitialisierung, also der IO-intensivste Abschnitt dargestellt worden ist. Der Unterschied zwischen ParSmart– und SMART–Konzept läßt sich jedoch in analoger Weise auf die nachfolgenden Berechnungen übertragen, bei denen stets auf die Daten der Datenbasis zugegriffen wird.

5.3.2 Implementation der Algorithmen

ParSmart ergänzt den vorliegenden SMART-Code um die Routinen zur Ausführung auf dem Parallelrechner Intel/Paragon. Zur Portierung wurden zunächst einige systemabhängige Routinen angepaßt sowie der Datentransfer über das Parallele Datei System (PFS) durch die verfügbaren Lese-/Schreiboperationen optimiert.

Die in Abschnitt 5.1.1 beschriebene Modellzerlegung wird vor Beginn der parallelen Berechnung seriell durchgeführt, die anschließende Aufstellung der Eingabedatensätze für die Teilmodelle wird auf den jeweiligen Rechenknoten, also parallel, durchgeführt.

Für das in Abschnitt 5.1.2 beschriebene Kommunikationsschema wird das asynchrone, NX-basierte *Message-passing* verwendet, welches den verfügbaren Coprozessor auf den Rechenknoten nutzt, vgl. Abschnitt 2.3 und so ermöglicht, daß sich Abschnitte von Berechnung und Kommunikation weitgehend überlappen. Hier ist jedoch teilweise Synchronisation erforderlich, da empfangende Rechenknoten erst dann mit der Berechnung fortfahren können, wenn die erwarteten Daten verfügbar sind.

Das konjugierte Gradienten-Verfahren aus Abschnitt 5.1.3 wird unter Anwendung der BLAS-Routinen implementiert. Diese werden bei der Berechnung der lokalen Skalarprodukte und der Skalar-Vektormultiplikationen angewendet [Kuc92]. Die Matrix-Vektormultiplikation wird in FORTRAN durchgeführt, da die BLAS-Routinen keine geeigneten Speicherstrukturen für dünnbesetzte Matrizen anbieten. Die Lösung der Gleichungssysteme erfolgt auf der 32–bit-Architektur des Rechners Intel/Paragon stets mit doppelter Genauigkeit, um numerische Fehler gering zu halten.

Die globale Kommunikation je Iterationsschritt des konjugierten Gradienten-Verfahrens sowie innerhalb der SMART-Iterationen zur Überprüfung der Konvergenz wird über die maschinenspezifischen, NX-basierten Funktionen durchgeführt. Diese *broadcast*–Operationen erlauben den effizienten globalen Datenaustausch zwischen allen beteiligten Rechenknoten, zeigen jedoch Skalierungsprobleme bei der Anwendung für größere Knotenzahlen.

Für die ergänzenden Datenstrukturen werden dynamisch verwaltete Speicherplätze angelegt, die nach Gebrauch wieder freigegeben werden. Die Größe der zu verwaltenden Datenmengen erfordert dieses Vorgehen, damit das ausführbare Programm in den Hauptspeicher des Parallelrechners geladen werden kann. Der Speicherplatz wird dann erst zur Laufzeit reserviert, wobei *Paging* durch das Betriebssystem zu erhöhtem Zeitverbrauch führen kann.

5.3.3 Bewertung und Skalierbarkeit der Parallelversion

In diesem Abschnitt werden die implementierten Routinen bezüglich der Skalierbarkeit bewertet. Insbesondere wird der erreichbare Speedup für einzelne Berechnungsabschnitte analysiert. Die angegebenen Werte beziehen sich auf das Beispiel der Rayleigh-Bénard-Konvektion, sie weichen in den anderen Beispielen jedoch nur geringfügig davon ab. Die absoluten Zeitangaben erfassen den Zeitverbrauch in Sekunden, der sich jeweils aus CPU-, System- sowie Datentransferzeiten zusammensetzt. Die Messungen sind aus einem Programmlauf herausgenommen, so daß auch Synchronisationszeiten berücksichtigt werden. Die Wartezeiten ergeben sich, wenn beispielsweise ein Rechenknoten Teilergebnisse von Rechenknoten benötigt, die diese noch nicht versendet haben. Ferner handelt es sich um Messungen im Einbenutzerbetrieb des Rechners. Laufen gleichzeitig Anwendungen anderer Benutzer auf dem Intel/Paragon, so können diese die Ausführungszeiten erheblich beeinflussen.

Dateninitialisierung:

Das Konzept gleichberechtigter Knoten, welches jedem Rechenknoten ein Teilmodell zuordnet, erfordert die Dateninitialisierung, d.h. die Aufstellung der Datenbasis und das Einlesen der Teilmodellbeschreibung auf jedem Rechenknoten. Da der Rechner Intel/Paragon nur über 5 IO-Knoten verfügt, die den gesamten Datentransfer zum PFS für die 140 Rechenknoten des Parallelrechners durchführen, ist hier aufgrund dieser Hardware-Beschränkungen kein linearer Speedup erreichbar. Um diesen Engpaß zu überwinden, ist das in Abschnitt 5.3.1 beschriebene IO-Konzept neu eingeführt worden, welches eine effizientere Initialisierung ermöglicht. Es ist zu beachten, daß die Dateninitialisierung nur einmal durchgeführt wird und mit zunehmender Anzahl der zu berechnenden Zeitschritte vernachlässsigt werden kann.

Kommunikationsschema:

Das Kommunikationsschema dient in ParSmart zur Assemblierung der Gesamtsystemmatrix und teilt diese gleichzeitig zeilenweise auf die Rechenknoten auf. Dazu werden die Zeilen, die auf benachbarten Rechenknoten zu inneren Randpunkten gehören, entsprechend

Abschnitt 5.1.2 aus dem lokalen Hypermatrizenschema ausgelesen, versendet und auf dem Empfängerknoten ergänzt.

Bei der Implementation konnte die Kommunikation weitgehend mit der Berechnung überlappt werden. So wird beispielsweise eine Matrixzeile aus dem Hypermatrizenschema ausgelesen und, während sie verschickt wird, kann die nächste Matrixzeile ausgelesen werden. Bei den Empfängerknoten können Wartezeiten nicht vollständig vermieden werden, da hier eine Zeile erst abgespeichert werden kann, wenn sie auf dem Rechenknoten eingetroffen ist.

Der Aufwand für die Kommunikation wird durch die Anzahl der inneren Randpunkte beeinflußt, die wiederum durch das in Abschnitt 5.1.1 beschriebene Modellzerlegungsverfahren vorgegeben werden.

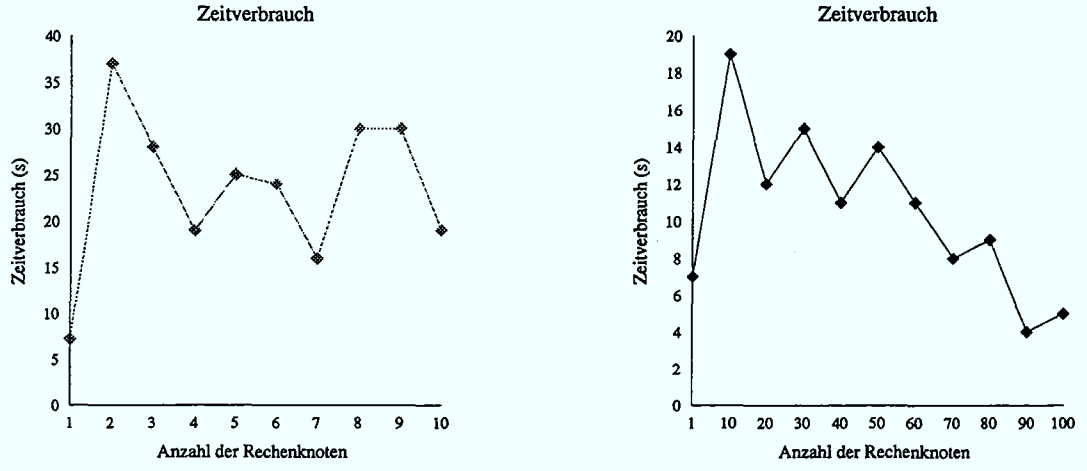


Bild 5.3.4: Zeitverbrauch zur Kommunikation, Rayleigh-Bénard-Konvektion

Bild 5.3.4 zeigt den Gesamtzeitverbrauch innerhalb der Kommunikationsroutine. Dabei werden CPU-, System- und Kommunikations- sowie Wartezeiten eingeschlossen. Diese Zeiten weisen starke Schwankungen auf, die zum einen durch die Abbildung der Teilmodelle auf die Rechenknoten entstehen. Werden benachbarte Teilmodelle auf direkt über das Kommunikationsnetzwerk miteinander verbundener Rechenknoten abgebildet, so sind die Kommunikationszeiten kürzer, als wenn Kommunikation zwischen physikalisch weiter entfernten Rechenknoten erforderlich ist, vgl. Abschnitt 2.3.

Zum anderen bildet die Größe der lokalen Hypermatrizen einen weiteren Einflußfaktor, da Werte aus den Zeilen dieser Matrizen ausgelesen bzw. in diese zurückgeschrieben werden müssen. Von der Größe dieser Hypermatrizen hängt der Aufwand zum Auslesen/Speichern einzelner Zeilen ab.

Der Anteil der Kommunikation je Rechenknoten nimmt tendenziell mit wachsender Knotenanzahl ab, da kleinere Teilmodelle entstehen und lokal weniger Matrixzeilen versendet bzw. empfangen werden müssen. Mit wachsender Anzahl der Teilmodelle steigt jedoch die

Anzahl der inneren Randpunkte, so daß für das Gesamtmodell die gesamte Netzauslastung zunimmt.

Für den Einsatz in dem Programmsystem ParSmart bedeutet die vorliegende Situation, daß durch die erforderliche Kommunikation der erreichbare Leistungszuwachs verringert wird.

Umspeichern der Systemmatrix:

Bei der Anwendung des iterativen konjugierten Gradienten-Verfahrens zur Lösung des Gleichungssystems $Ax = b$ muß in jedem Iterationsschritt auf die Einträge der Matrix A zugegriffen werden. Für ParSmart wird deshalb eine Konvertierung des Speicherschemas vorgenommen, indem die Matrix A aus dem Hypermatrizenschema in das zeilenorientierte Speicherschema umgespeichert wird.

Bild 5.3.5 zeigt die Rechenzeit zur Konvertierung der Linkhandseite der diskretisierten Navier-Stokes-Gleichung. Teilweise liegt die Rechenzeit unter der bei linearem Speedup erreichbaren, was wie folgt zu begründen ist: Bei der Konvertierung des Speicherschemas werden jeweils Zeilen aus der Hypermatrix ausgelesen und nur die Nichtnulleinträge in das zeilenorientierte Speicherschema geschrieben. Unter der Annahme, daß alle Einträge, also auch die Nullen, in Matrix A umgespeichert werden, nimmt die Anzahl der zu lesenden Einträge je Rechenknoten quadratisch ab, so daß in diesem Fall auch eine quadratische statt einer linearen Abnahme der Rechenzeiten zu erwarten wäre. Da im Hypermatrizenschema aber nur Teilmatrizen gespeichert werden, die mindestens einen Nichtnulleintrag aufweisen, ist keine quadratische Abnahme der Rechenzeit erreichbar, jedoch eine superlineare.

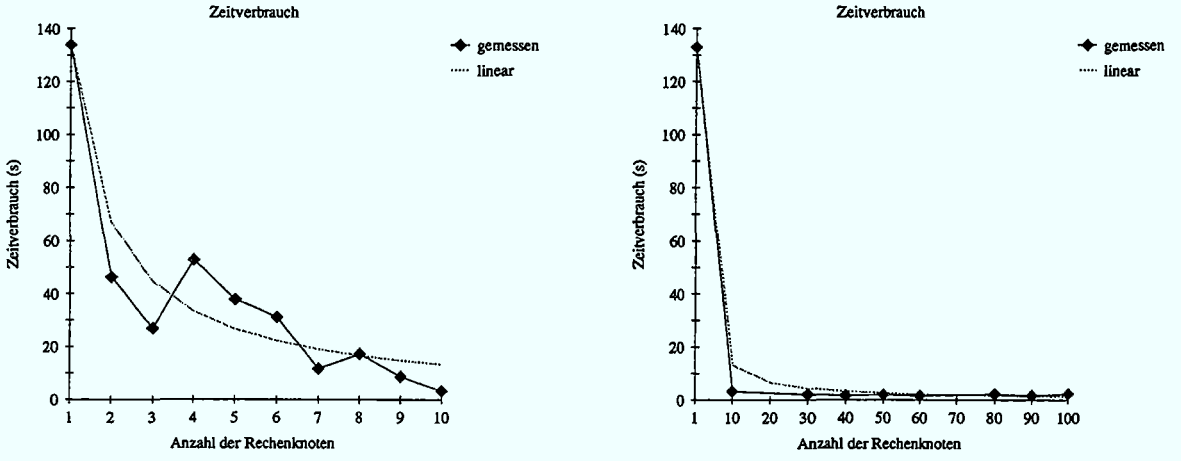


Bild 5.3.5: Zeitverbrauch zur Konvertierung des Speicherschemas

Bei den Testläufen auf 4, 5 und 6 Rechenknoten treten extrem hohe Rechenzeiten auf. Das läßt sich in diesem Fall auf das zur Laufzeit erfolgende, dynamische Anlegen der Speicherplätze für das zeilenorientierte Speicherschema zurückführen. Generell treten bei der Matrixkonvertierung Probleme durch Datentransfer zum Hintergrundspeicher auf, da

Speicherplatz für die Systemmatrix im Hypermatrizenschema, wie auch im zeilenorientierten Speicherschema benötigt wird.

Ferner sind die in Bild 5.3.5 sichtbaren Schwankungen auf die Numerierung der Teilmodelle zurückzuführen. Im Beispiel bleibt die Knotenumerierung für das Gesamtmodell stets gleich, während die Teilmodelle jeweils von Eins beginnend aufwärts numeriert werden. Die Bandbreite und damit die Anzahl der auf einer Ebene zu speichernden Teilhypermatrizen hängt dann von der Topologie der lokalen Modelle ab.

Lösungsverfahren:

Zur Lösung der Gleichungssysteme ist das in Abschnitt 5.1.3 beschriebene konjugierte Gradienten-Verfahren implementiert worden. Im folgenden wird anhand der Berechnung des Geschwindigkeitsfeldes aus Beispiel 6.1 die Leistungssteigerung analysiert. Die Dimension des Gleichungssystems beträgt 20025. Mit maximal 18 Nichtnullelementen je Matrixzeile liegt ein Besetzungsgrad von etwa 0.09% vor. Als Konvergenzkriterium wird $\epsilon = 10^{-10}$ gewählt. Bei der ersten Bestimmung des Geschwindigkeitsfeldes werden 1783 Iterationen durchlaufen. Der Lösungsvektor einer Berechnung wird als Startvektor für die nachfolgende Berechnung verwendet, so daß bereits eine gute Näherung für den Lösungsvektor vorliegt und dadurch die Zahl der Iterationen verringert wird. Bei den durchgeführten Rechnungen und dem genannten Konvergenzkriterium werden in ParSmart in den nachfolgenden Lösungsabschnitten zwischen 100 und 300 Iterationen benötigt. Diese Anzahl der Iterationen hängt jedoch von der Zeitschrittgröße und damit der Kondition der Matrix ab.

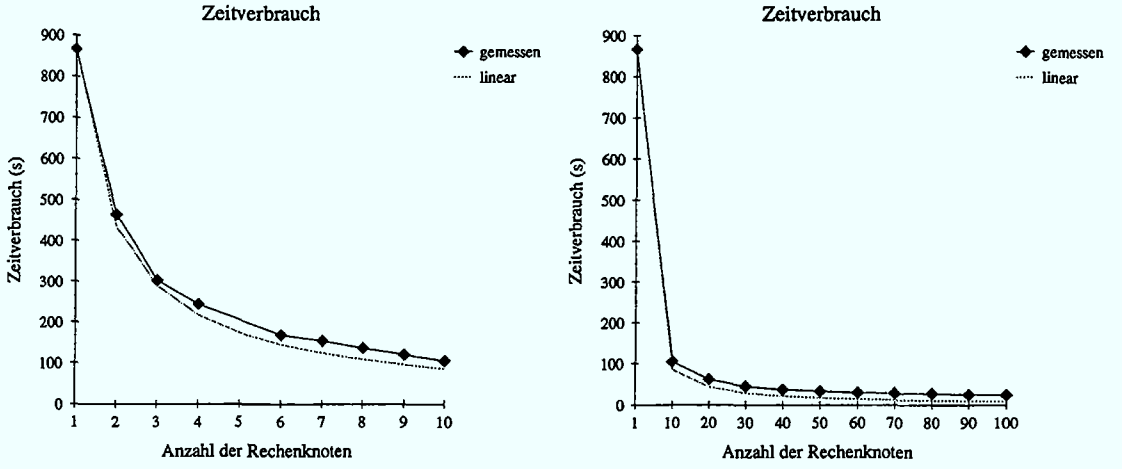


Bild 5.3.6: Zeitverbrauch des CG-Verfahrens, Rayleigh-Bénard-Konvektion

In Bild 5.3.6 wird der Gesamtzeitverbrauch, bestehend aus CPU-, System- und Kommunikationszeiten, für die Lösung mit dem konjugierten Gradienten-Verfahren angegeben.

Auf bis zu zehn Rechenknoten zeigt sich eine gute Skalierbarkeit, wie in der linken Hälfte von Bild 5.3.6 deutlich wird. Die Rechenzeiten liegen gering über dem Optimum, das aus dem

Zeitverbrauch der seriellen Berechnung unter Annahme linearen Speedups bestimmt wird. In der rechten Bildhälfte werden Rechnungen auf bis zu 100 Rechenknoten betrachtet.

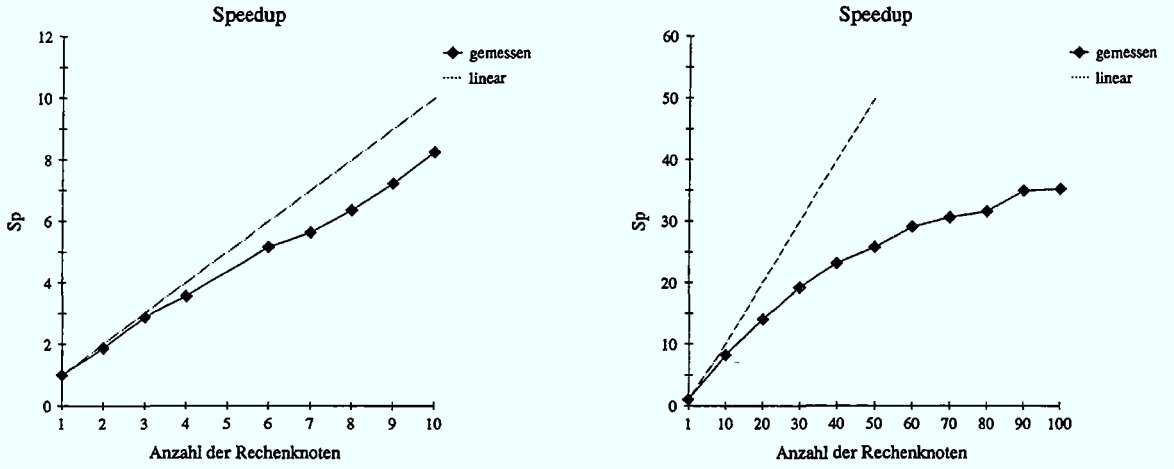


Bild 5.3.7: Speedup des CG-Verfahrens, Rayleigh-Bénard-Konvektion

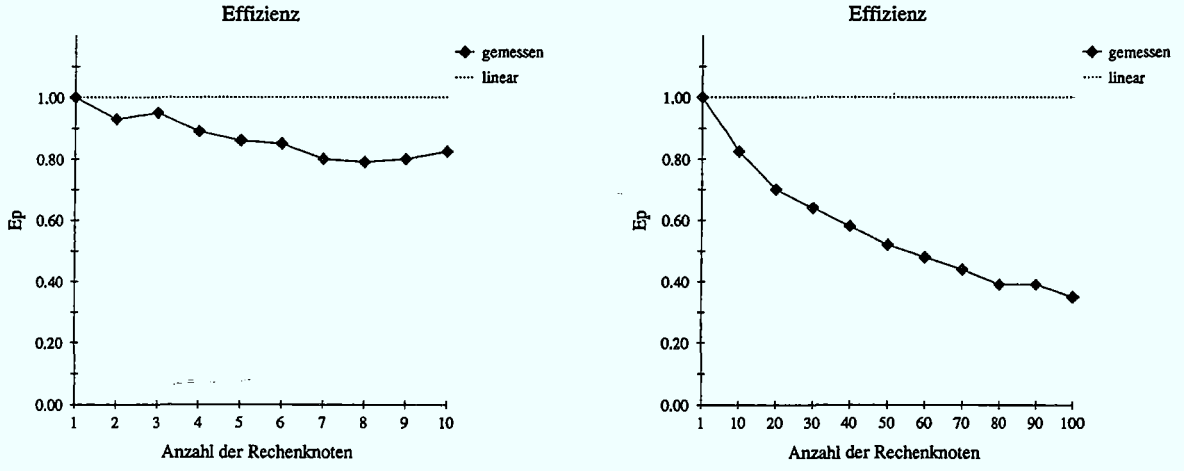


Bild 5.3.8: Effizienz des CG-Verfahrens, Rayleigh-Bénard-Konvektion

Die Werte für den Speedup zeigen nach Bild 5.3.7 für bis zu zehn Rechenknoten eine gute Leistungssteigerung. Abweichungen vom linearen Speedup entstehen durch die Kommunikation in der Iterationsschleife des konjugierten Gradienten-Verfahrens. In der rechten Bildhälfte wird der Speedup auf bis zu 100 Rechenknoten angegeben.

Dabei wird deutlich, daß ein maximaler Speedup von etwa 35 erreicht werden kann. Auf hohen Knotenanzahlen überwiegt der Synchronisationsaufwand in jeder Iterationsschleife, da die Anzahl der lokal ausführbaren Operationen abnimmt. Für 50 Rechenknoten sind beispielsweise nur etwa 400 Zeilen der Gesamtmatrix je Rechenknoten mit maximal 18 Einträgen zu bearbeiten.

Während der Anwender, der an kurzen Rechenzeiten interessiert ist, bereit ist, 100 Rechenknoten zur Lösung des Gleichungssystems einzusetzen, zeigt die in Bild 5.3.8 dargestellte Effizienz, daß diese bereits bei dem Einsatz von mehr als 50 Rechenknoten unter 50% liegt.

5.4 Optimale Anzahl der Rechenknoten

Die Betrachtung der Skalierbarkeit paralleler Algorithmen zeigt, daß ein Problem gegebener Größe nicht auf einer beliebig großen Anzahl von Rechenknoten effizient ausgeführt werden kann. In der Praxis zeigt sich, daß die Rechenzeiten wieder ansteigen, wenn zu viele Rechenknoten ein kleines Modell berechnen. Deshalb wird im folgenden eine Abschätzung für die Anzahl der Rechenknoten angegeben, auf denen ein Problem gegebener Größe mit dem höchsten Leistungszuwachs durch ParSmart berechnet werden kann.

Der Leistungszuwachs einer Berechnung mit ParSmart wird durch die folgenden Faktoren beinflußt:

- die Dimension des Gesamtproblems,
- die Zerlegung des Gesamtmodells in Teilmodelle:
  - das Verhältnis der inneren Randpunkte zu der Größe eines Teilmodells,
  - die Zuordnung der Teilmodelle zu den Rechenknoten,
- der Anteil der Kommunikation in den Algorithmen:
  - Kommunikationsschema
  - paralleles CG-Verfahren
- die Kommunikationszeiten und die Netzauslastung sowie
- die Zeiten für den Datentransfer zum Hintergrundspeicher.

Um eine praktische Abschätzung für den Zeitverbrauch einer ParSmart Berechnung zu erhalten, werden die vorangehend zusammengefaßten Einflußfaktoren nicht einzeln berücksichtigt, sondern das grundlegende Verhalten einer ParSmart Rechnung betrachtet.

Die Rechenzeit setzt sich theoretisch aus einem seriellen Anteil α^* und einem linear skalierbaren Anteil β^*/p^* zusammen. Dabei sind α^* und β^* von der Anwendung abhängige Parameter und p^* bezeichnet die Anzahl der Rechenknoten. In der Praxis tritt jedoch noch ein weiterer Anteil auf, der mit zunehmender Anzahl der Rechenknoten wächst. Für ParSmart läßt sich hier aus den Zeitmessungen verschiedener Testläufe ein linearer Anteil der Form $\gamma^* p^*$ ermitteln. Der Faktor γ^* charakterisiert insbesondere den Einfluß der Netzauslastung und der Synchronisationszeiten, die beide mit wachsender Knotenanzahl ansteigen.

In Bild 5.4.1 werden die Anteile β^*, γ^* der Gesamtrechenzeit dargestellt.

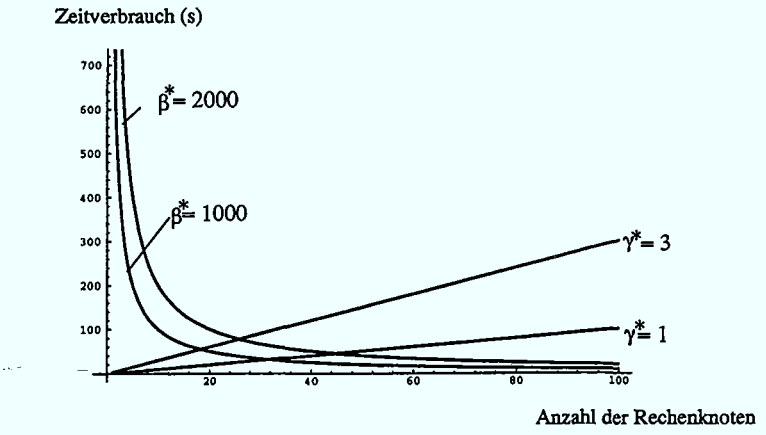


Bild 5.4.1: Einflußfaktoren auf die Rechenzeiten paralleler Berechnungen

Die Rechenzeit in der Zeitschleife kann dann nach Formel (5.4.3)

$$t^*(p^*) \approx \alpha^* + \frac{\beta^*}{p^*} + \gamma^* p^* \tag{5.4.3}$$

angenähert werden.

Soll die Rechenzeit minimal und entsprechend der Leistungszuwachs maximal werden, so gilt für die Anzahl der optimalen Rechenknoten p^*_{opt}:

$$p^*_{opt} \approx \sqrt{\frac{\beta^*}{\gamma^*}} \tag{5.4.4}$$

wobei β^* etwa der Rechenzeit des seriellen Programmes entspricht und γ^* maschinenabhängige Parameter zusammenfaßt.

Für das in Abschnitt 6.1 vorgestellte Beispiel der Rayleigh-Bénard-Konvektion läßt sich der Zeitverbrauch in der Zeitschleife bei einer Iteration durch die in Bild 5.4.2 gezeigte Funktion nach der Methode der kleinsten Quadrate approximieren. Für die Parameter α^*, β^* und γ^* lassen sich die Werte $\alpha^* = 33$, $\beta^* = 1549$ und $\gamma^* = 2$ ablesen, die optimale Anzahl der Rechenknoten liegt dann bei 28.

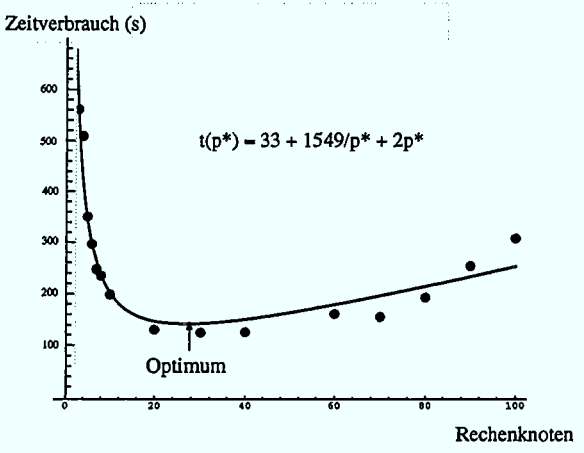


Bild 5.4.2: Approximation der Rechenzeit für einen Zeitschritt

Bei der beschriebenen Vorgehensweise muß der Parameter β^*, der etwa der Ausführungszeit einer Zeitschrittberechnung entspricht, ermittelt werden. Dies kann zum einen über eine Vorabrechnung geschehen, in der ein Zeitschritt für das aufgestellte Modell auf einem Rechenknoten berechnet wird. Ausgehend von dieser Rechenzeit kann anschließend die Anzahl der Rechenknoten bestimmmt werden, die eine minimale Ausführungszeit der Problemlösung ermöglicht.

Da die Rechenzeit bei einer Lösung mit dem konjugierten Gradienten-Verfahren von der Anzahl der Iterationen, d.h. von Startwert, Matrixkondition, etc. abhängt, wird für die nachfolgende Abschätzung das Cholesky-Verfahren der Standard-SMART-Version betrachtet.

Die Rechenzeit kann dann durch den Ansatz

$$\beta^* = c_1 n^2 + c_2 n + c_3 \tag{5.4.5}$$

angenähert werden. Aus Testläufen lassen sich die Werte $c_1 = 1.758 \cdot 10^{-6} B$, $c_2 = 0.013 B$ sowie $c_3 = 10.1$ auf dem Rechner Intel/Paragon bestimmen. Dabei bezeichnet B die Bandbreite der Matrix A des zu lösenden Gleichungssystems $Ax = b$ der Dimension n; für das betrachtete Beispiel gilt $B = 9$. Durch die Bandbreite B werden dabei unter anderem die Besetzungsstruktur der Systemmatrix sowie die Knotenanzahl der verwendeten Finiten Elemente berücksichtigt.

6 Beispiele paralleler Berechnungen mit ParSmart

Im folgenden werden drei Beispielrechnungen mit ParSmart vorgestellt, anhand derer der Leistungszuwachs durch den Parallelrechnereinsatz bewertet wird. Wie in Abschnitt 3.2 erläutert, können mit SMART bislang nur Netze mit maximal 10922 Finite-Element-Knoten, das entspricht 32766 Freiheitsgraden im Geschwindigkeitsnetz, berechnet werden. Dieses sind im Vergleich zu den in Abschnitt 2.4 betrachteten Anwendungen 'relativ kleine' Beispiele.

Bei den diskutierten Beispielen liegt der Schwerpunkt auf der Bewertung der Skalierungseigenschaften der Parallelversion. Sie sollen die Einsatzmöglichkeiten paralleler Rechenknoten für Finite-Element-Berechnungen veranschaulichen. Für die untersuchten Beispiele werden jeweils Rechenzeiten, Speedup und Effizienz innerhalb einer Zeitschleife analysiert.

Neben der Bewertung des Leistungszuwachses sollen die Beispiele jedoch auch einer ersten Verifikation des parallelen Programms ParSmart dienen. Aus diesem Grunde werden für ParSmart nur solche Beispiele vorgestellt, für die serielle Vergleichsläufe mit SMART durchgeführt werden können.

In Abschnitt 6.1 wird als Beispiel für eine freie Konvektionsströmung die Rayleigh-Bénard Konvektion betrachtet. Aufgrund geringer Temperaturunterschiede werden dabei konstante Stoffwerte betrachtet.

Als weiteres Beispiel wird in Abschnitt 6.2 die Ausbreitung von flüssigem Wasserstoff auf Wasser untersucht. Es handelt sich dabei um eine Versuchsnachrechnung eines am Institut für Sicherheitsforschung und Reaktortechnik durchgeführten Experiments. Dabei werden aufgrund der hohen Temperaturunterschiede variable Stoffwerte betrachtet. Die Berechnung berücksichtigt einen Phasenwechsel von Wasser zu Eis, der bei dem Experiment auftritt.

Als Beispiel für eine erzwungene Konvektionsströmung wird in Abschnitt 6.3 die Umströmung eines Laminarflügels betrachtet.

6.1 Rayleigh-Bénard Konvektion

Im folgenden wird das Beispiel der Rayleigh-Bénard Konvektion betrachtet. Für dieses Beispiel ist die analytische Lösung bekannt, und es liegen experimentelle Untersuchungen vor [Tri88], [ZO82]. Da diese freie Konvektionsströmung außerdem auch im SMART-Handbuch [LS91] diskutiert wird, kann anhand dieses Beispiels eine erste Verifikation des parallelisierten Programms durchgeführt werden.

6.1.1 Problembeschreibung

Im folgenden wird Wasser betrachtet, das sich zwischen zwei zur Bildebene horizontal unendlich ausgedehnten Platten befindet. An der unteren Platte wird eine Temperatur von $20,5°C$ vorgegeben, während die obere Platte eine Temperatur von $20,0°C$ aufweist. Die Anfangstemperatur des Fluids beträgt ebenfalls $20,0°C$. Zwischen den beiden Platten liegt

damit eine Temperaturdifferenz von $0,5K$ vor. Bild 6.1.1 zeigt die Diskretisierung durch $QUAM4$ Elemente, sowie die Randbedingungen, wobei die vertikalen Ränder adiabat angenommen werden. Um mit SMART ein möglichst großes Beispiel zu erstellen, wurden hier 400×25 Elemente für die Diskretisierung verwendet. Für eine Höhe $h = 0,01m$, eine Länge

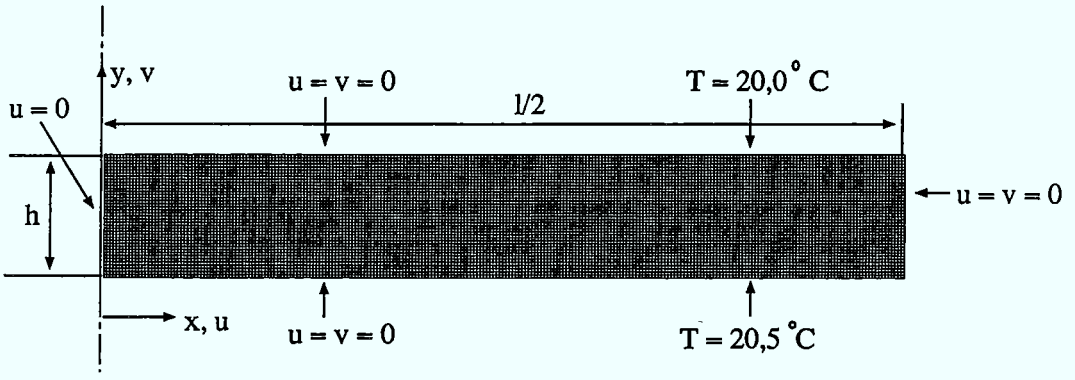


Bild 6.1.1: Diskretisierung und Randbedingungen, Rayleigh-Bénard-Konvektion

$l = 0,12m$ und eine Temperaturdifferenz von $\Delta T = 0,5K$ ergibt sich eine Rayleighzahl von $Ra = \frac{g\beta\Delta T H^3}{(\frac{\mu}{\rho})(\frac{k_T}{\rho c_T})} = 7240$, die zu einem Einsetzen der Konvektionsströmung führt. Die Stoffwerte für Wasser bei $20,0°C$ betragen dabei $\mu = 9,93 \cdot 10^{-4}$ Pa s, $\rho = 998,2$ kg/m^3, $c_T = 4182,5$ J/kgK, $k_T = 0,597$ W/mK, $\beta = 2,1 \cdot 10^{-4}$ 1/K und $g = 9,81$ m/s^2. Geometrie, Randbedingungen und Stoffwerte entsprechen dem in [LS91] diskutierten Beispiel.

In Bild 6.1.2 werden Temperatur- und Geschwindigkeitsfelder zu vier Zeitpunkten dargestellt. Während nach dem ersten Zeitschritt noch eine gleichförmige Temperaturschichtung vorliegt, bilden sich nach etwa 100 Sekunden leichte Konvektionsrollen aus. Nach etwa 200 Sekunden liegt ein quasi stationärer Zustand vor, die Konvektionszellen sind deutlich erkennbar. Die noch im Fluid vorhandenen Schwankungsbewegungen können numerisch nicht mehr erfaßt werden.

In den Bildern 6.1.2, 6.2.3 und 6.3.2 werden zur besseren Übersicht, Modelle mit weniger Freiheitsgraden dargestellt, als tatsächlich in den Rechnungen betrachtet wurden. Diese Vorgehensweise ermöglicht insbesondere einen deutlicheren Eindruck über die Geschwindigkeitsfelder der betrachteten Modelle.

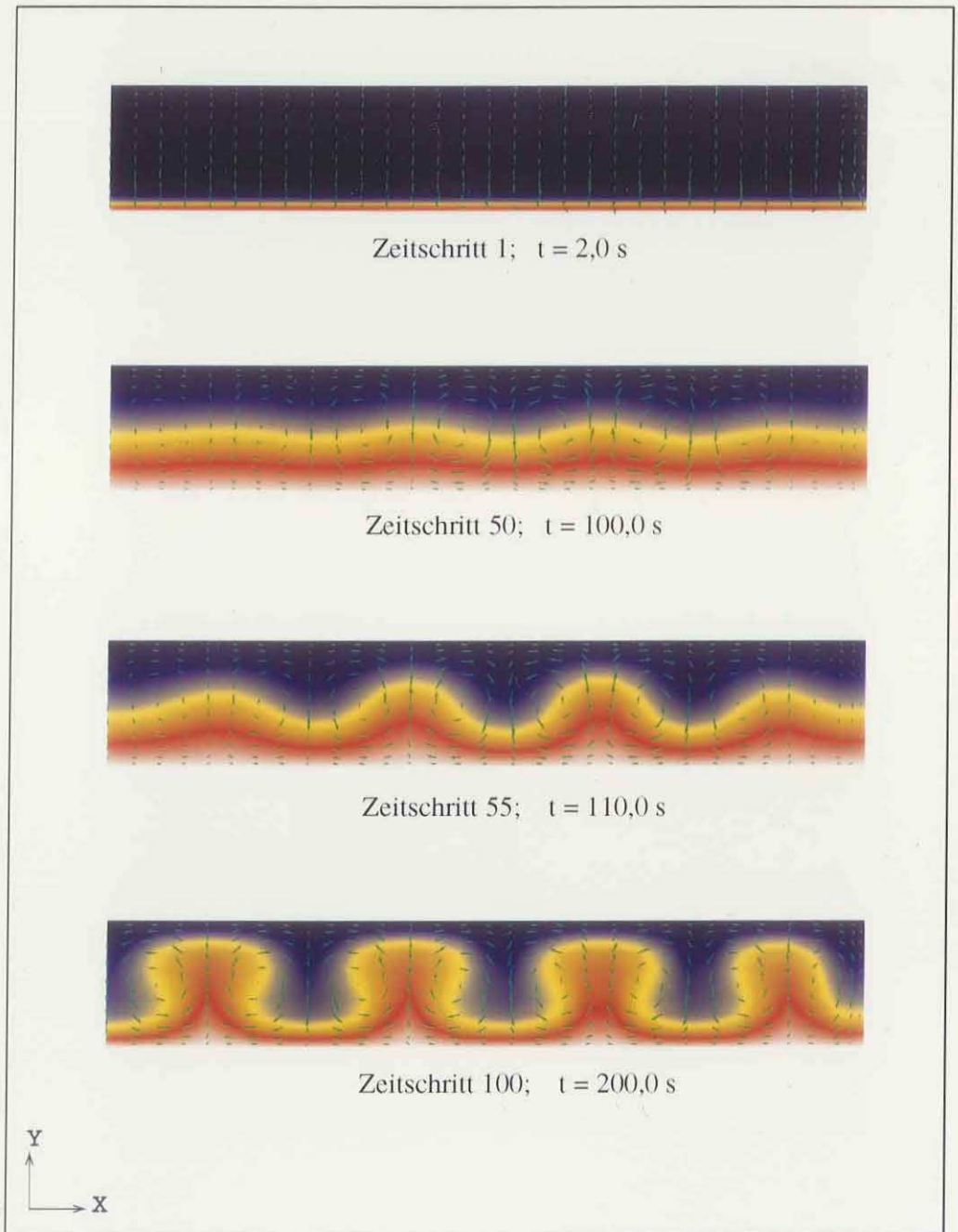


Bild 6.1.2: Temperatur- und Geschwindigkeitsfelder bei der Rayleigh-Bénard-Konvektion

6.1.2 Speedup und Effizienz

Im folgenden wird der Leistungszuwachs durch den Einsatz des Rechners Intel/Paragon für die Berechnung der Rayleigh-Bénard Konvektion analysiert. Einzelne Berechnungsabschnitte wurden bereits in Abschnitt 5.3.3 diskutiert, so daß hier nur noch der Zeitverbrauch des Gesamtsystems und innerhalb der Zeitschleife betrachtet wird.

Das Beispiel bildet mit 10000 Finiten Elementen ein Modell, welches mit SMART ohne Unterstrukturtechnik noch berechenbar ist. Die Dimension der Matrix zur Lösung der Wärmebilanzgleichung ist 9600, die für das Geschwindigkeitsfeld 20025.

Die Rechenzeiten für die hier betrachtete Berechnung, d.h. für die Dateninitialisierung und die Bestimmung der Lösung für einen Zeitschritt werden in Bild 6.1.3 dargestellt. Neben den Meßwerten wird die bei linearem Speedup erreichbare Rechenzeit gezeigt. Die Rechenzeiten umfassen dabei CPU-, System- und Datentransferzeiten.

Es ist zu beachten, daß ParSmart auf einem Rechenknoten mit $1720s$ bereits deutlich schneller ausgeführt wird als SMART ($2936s$, vgl. Tabelle 6). Dieses ist auf das neue Konzept zur IO-Reduktion sowie den Austausch des Lösungsverfahrens zurückzuführen.

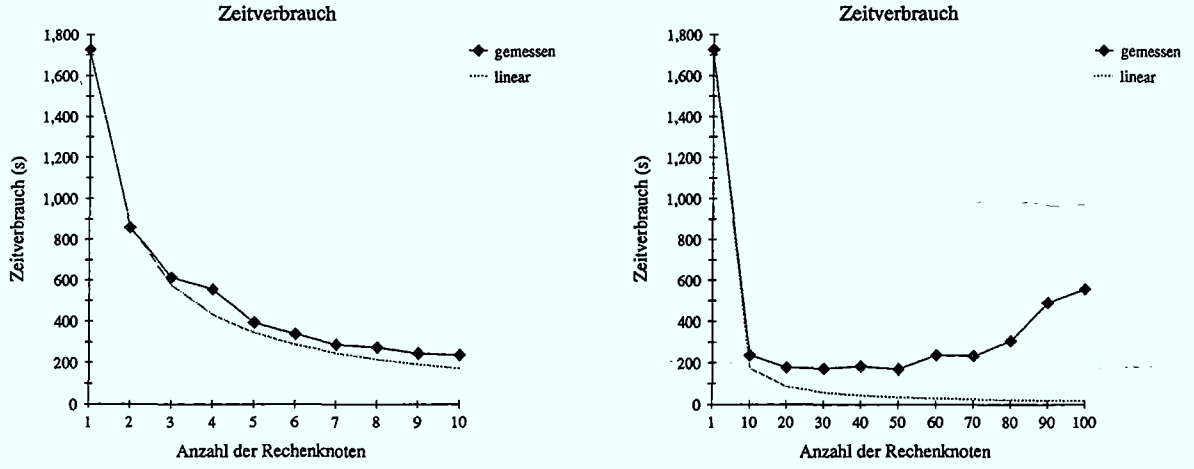


Bild 6.1.3: Zeitverbrauch der Gesamtberechnung, Rayleigh-Bénard Konvektion

Bild 6.1.3 zeigt, daß auf bis zu zehn Rechenknoten eine nahezu lineare Verkürzung der Rechenzeiten eintritt. Bei Verwendung von 20, 30 oder 40 Rechenknoten sind kaum Veränderungen der Rechenzeiten festzustellen, während die Rechenzeiten ab 50 Rechenknoten wieder ansteigen. Dieses ist auf die kleine Problemgröße sowie *Paging*–Effekte zurückzuführen.

Obige Bilder geben die Gesamtrechenzeiten für die Dateninitialisierung und die Berechnung *eines* Zeitschrittes an. Bei instationären Berechnungen wird die Zeitschleife wiederholt durchlaufen, so daß die Phase der Dateninitialisierung weitgehend vernachlässigt werden kann. Deshalb wird in den nachfolgenden Bildern der Rechenzeitverbrauch innerhalb der

Zeitschleife dargestellt. Während in der linken Bildhälfte wieder die Werte für ein bis zehn Rechenknoten angegeben werden, sind rechts die Werte für bis zu 100 Rechenknoten dargestellt. Die Rechenzeiten steigen auch hier ab etwa 50 Rechenknoten wieder an, so daß die Leistung des Parallelrechners keine weitere Verkürzung des Zeitverbrauchs ermöglicht. Die Angabe der Rechenzeiten für größere Anzahlen von Rechenknoten erfolgt, um auch dafür die Lauffähigkeit der Parallelversion zu demonstrieren.

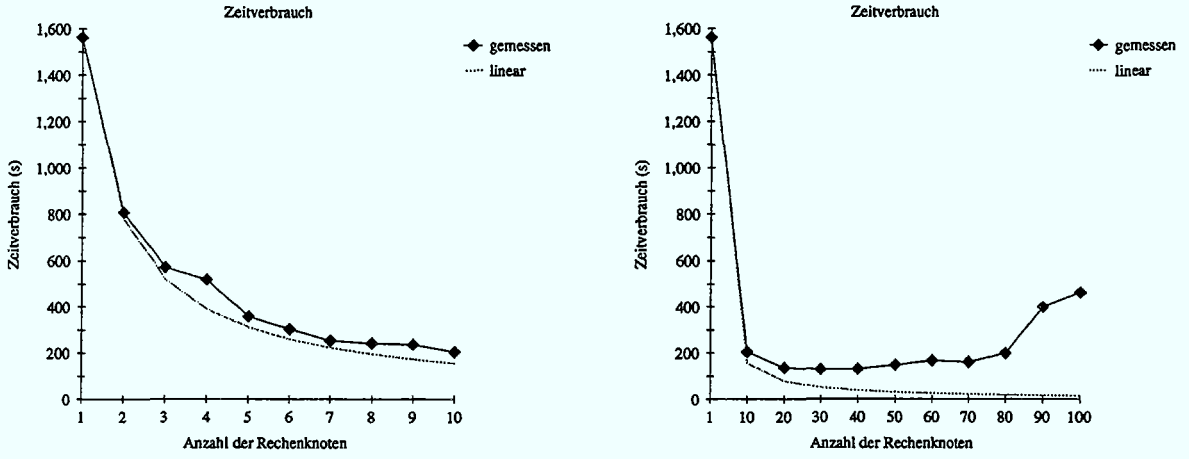


Bild 6.1.4: Zeitverbrauch innerhalb einer Zeitschleife, Rayleigh-Bénard-Konvektion

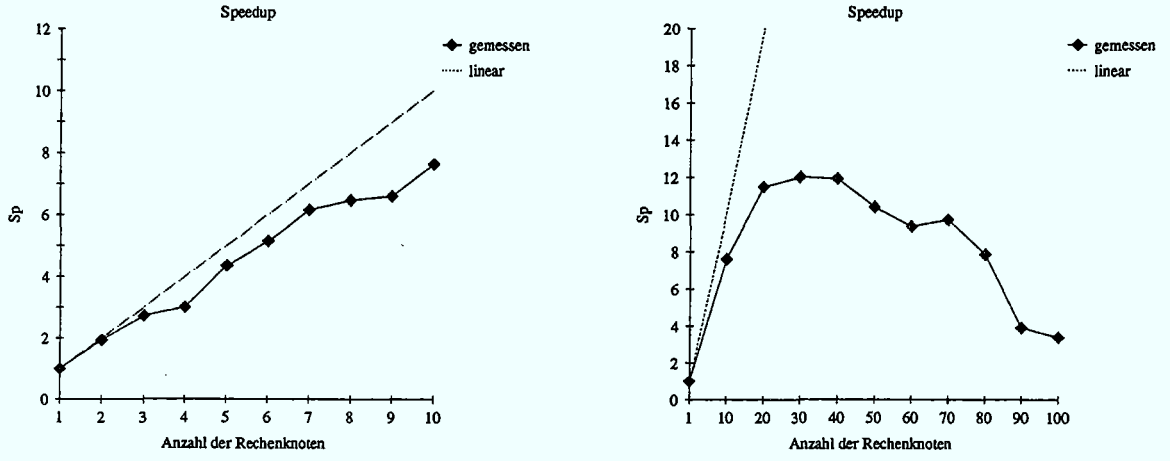


Bild 6.1.5: Speedup innerhalb einer Zeitschleife, Rayleigh-Bénard-Konvektion

Bild 6.1.5 zeigt den Leistungszuwachs innerhalb der Zeitschleife. Während auf zehn Rechenknoten ein maximaler Speedup von 8 erreicht wird können die Rechenzeiten durch den Einsatz von 30 Rechenknoten um den Faktor 12 verkürzt werden.

Der Anwender, der an einer schnellen Berechnung interessiert ist, erhält die kürzesten Rechenzeiten durch den Einsatz von 28 Rechenknoten, vgl. Abschnitt 5.4.

Der Einsatz der 28 Rechenknoten ermöglicht zwar die maximale Verkürzung der Rechen-

zeiten, die in Bild 6.1.6 dargestellte Effizienz zeigt jedoch, daß die Rechenknoten in diesem Fall nur mit einer Effizienz von 40% genutzt werden können.

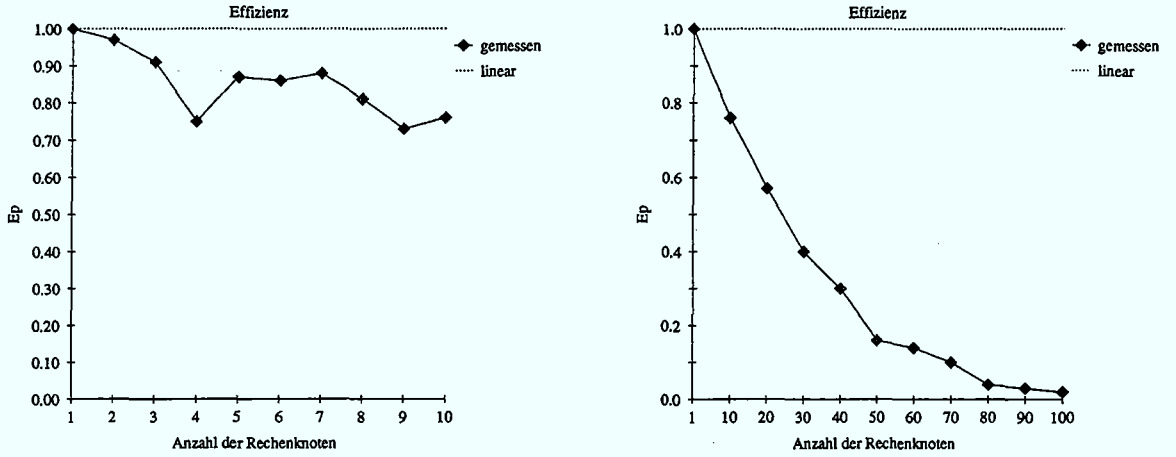


Bild 6.1.6: Effizienz innerhalb einer Zeitschleife, Rayleigh-Bénard Konvektion

Zusammenfassend zeigen die Ergebnisse für das betrachtete Modell eine gute Verkürzung der Rechenzeiten. Es ist jedoch zu beachten, daß für einzelne Testläufe unterschiedliche Zeitspannen gemessen werden. Gründe dafür liegen beispielsweise in der Abbildung der Teilmodelle auf die Rechenknoten.

Für die Verifikation des parallelen Programmsystems wurden jeweils die mit ParSmart berechneten Lösungsvektoren für Temperatur- und Geschwindigkeitsfelder verschiedener Zeitschritte mit den entsprechenden, mit Standard-SMART berechneten Lösungsvektoren verglichen und Übereinstimmung festgestellt.

6.2 Wasserstoffausbreitung

Dem folgenden Beispiel liegen Versuche am Institut für Sicherheitsforschung und Reaktortechnik der KFA zur Ausbreitung von flüssigem Wasserstoff auf einer Wasseroberfläche zugrunde. Wasserstoff stellt einen zukunftsträchtigen Energieträger dar, der auf verschiedenen Wegen vom Herstellungsort zum Verbraucher gelangen kann. Ein Transportmittel sind dabei Schiffe [GG93]. Bei der störfallbedingten Freisetzung kann sich der flüssige Wasserstoff auf der Wasseroberfläche ausbreiten. In den durchgeführten Experimenten wurde untersucht, wie sich der flüssige Wasserstoff in einem solchen Fall ausbreitet.

Mit SMART kann das Zusammentreffen von flüssigem Wasserstoff und Wasser modelliert werden. Dabei muß aufgrund der hohen Temperaturdifferenzen mit variablen Stoffwerten gerechnet werden. Bei der durchgeführten Berechnung soll die Frage geklärt werden, ob sich bei einem solchen Vorgang, wie im Versuch auch, Eis bildet.

6.2.1 Problembeschreibung

Bei den durchgeführten Versuchen wurde flüssiger Wasserstoff auf ein mit Wasser gefülltes Becken gegeben. Die Ausbreitung des flüssigen Wasserstoffs wurde durch Temperaturmessungen emittelt.

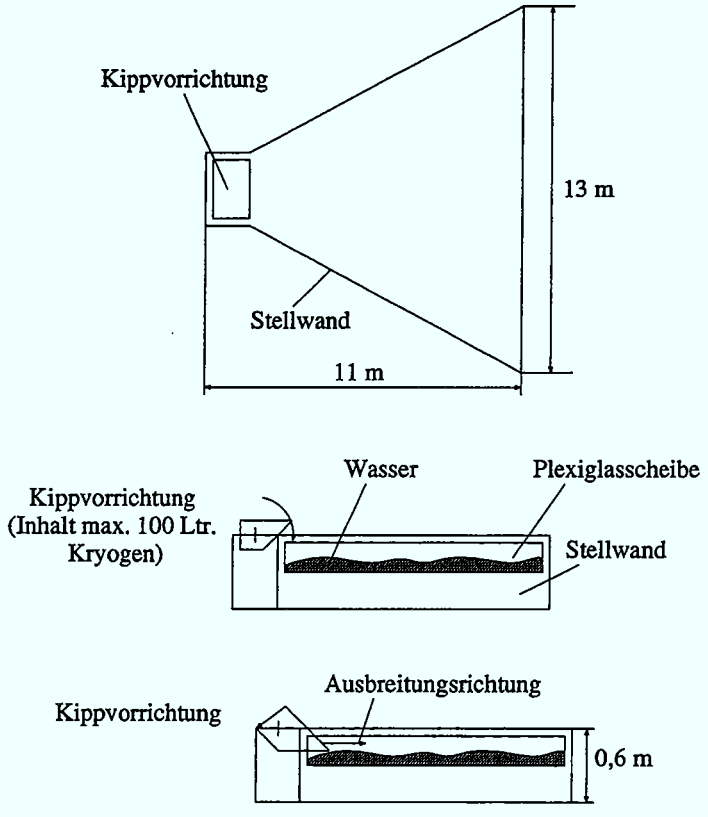


Bild 6.2.1: Versuchsaufbau zur Wasserstoffausbreitung

Bild 6.2.1 zeigt den Versuchsaufbau zur Simulation der Wasserstoffausbreitung. In der Kippvorrichtung befinden sich maximal 100 Liter flüssiger Wasserstoff, die in das mit Wasser gefüllte Becken gegeben werden. Die Temperaturen werden an und unterhalb der Wasseroberfläche durch Thermoelemente gemessen. Während der Ausbreitung bildet sich Eis auf der Wasseroberfläche.

Zur Berechnung des beschriebenen Experimentes wird das Auftreffen des flüssigen Wasserstoffes simuliert, indem an einem Teil der Oberfläche ein Wärmestrom von $\dot{q} = 100kW/m^2$ entzogen wird. Dieses entspricht dem Auftreffen des flüssigen Wasserstoffes mit einer Temperatur von $-253°C$ [Zab67]. Die Ausbreitung des flüssigen Wasserstoffes wird bei den Berechnungen vernachlässigt. Der mit SMART berechnete Phasenwechsel erfordert eine besonders feine Diskretisierung sowie Zeitschritte von $\Delta t = 0,01$s. In der nachfolgenden Berechnung wird deshalb nur die Anfangsphase betrachtet, da diese bereits die Bewertung des Leistungszuwachses durch den Parallelrechner Intel/Paragon ermöglicht.

Die Diskretisierung erfolgt wie in Bild 6.2.2 gezeigt, durch 100×100 Rechteckselemente.

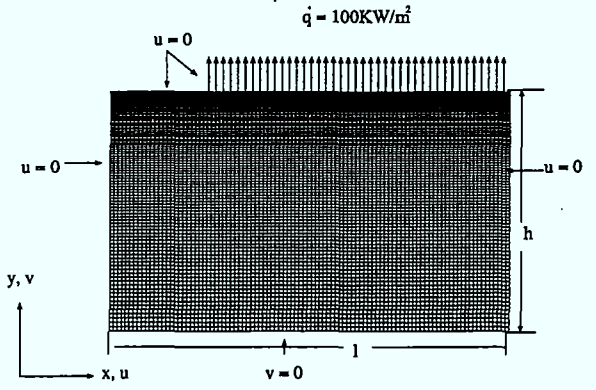


Bild 6.2.2: Diskretisierung und Randbedingungen, Wasserstoffausbreitung

Die Abmessungen betragen $h = 0,1cm$ und $l = 0,2cm$. Für den Anfangszustand werden die Stoffwerte von Wasser bei $20,0°C$ betrachtet, d.h. $\mu = 9,93 \cdot 10^{-4}$ Pa s, $\rho = 998,2$ kg/m^3, $c_T = 4182,5$ J/kgK, $k_T = 0,597$ W/mK, $\beta = 2,1 \cdot 10^{-4}$ 1/K und $g = 9,81$ m/s^2 [Ing88]. Die Randbedingungen werden in Bild 6.2.2 angegeben.

Während der Berechnung sind aufgrund der hohen Temperaturdifferenzen temperaturabhängige Stoffwerte zu beachten.

In Bild 6.2.3 werden Temperatur- und Strömungsfelder zu ausgewählten Zeitpunkten dargestellt. Das erste Bild zeigt Temperatur- und Geschwindigkeitsfelder nach einem Zeitschritt von $0,01s$. Hier liegt nur eine geringe Veränderung des Ausgangszustandes vor. Im weiteren Zeitverlauf sinken die Temperaturen an der Wasseroberfläche in dem Bereich, in dem der Wärmestrom entzogen wird, auf $0,0°C$ ab. Wird weiterhin Wärme entzogen, so bildet sich eine Eisschicht aus, die dicker wird. Der Phasenübergang wird mit SMART berechnet. In den Bildern werden Wasser und Eis durch die eingezeichnete Linie für eine Temperatur von Null Grad getrennt. Der schwarze Bereich kennzeichnet Temperaturen unter $0,0°C$.

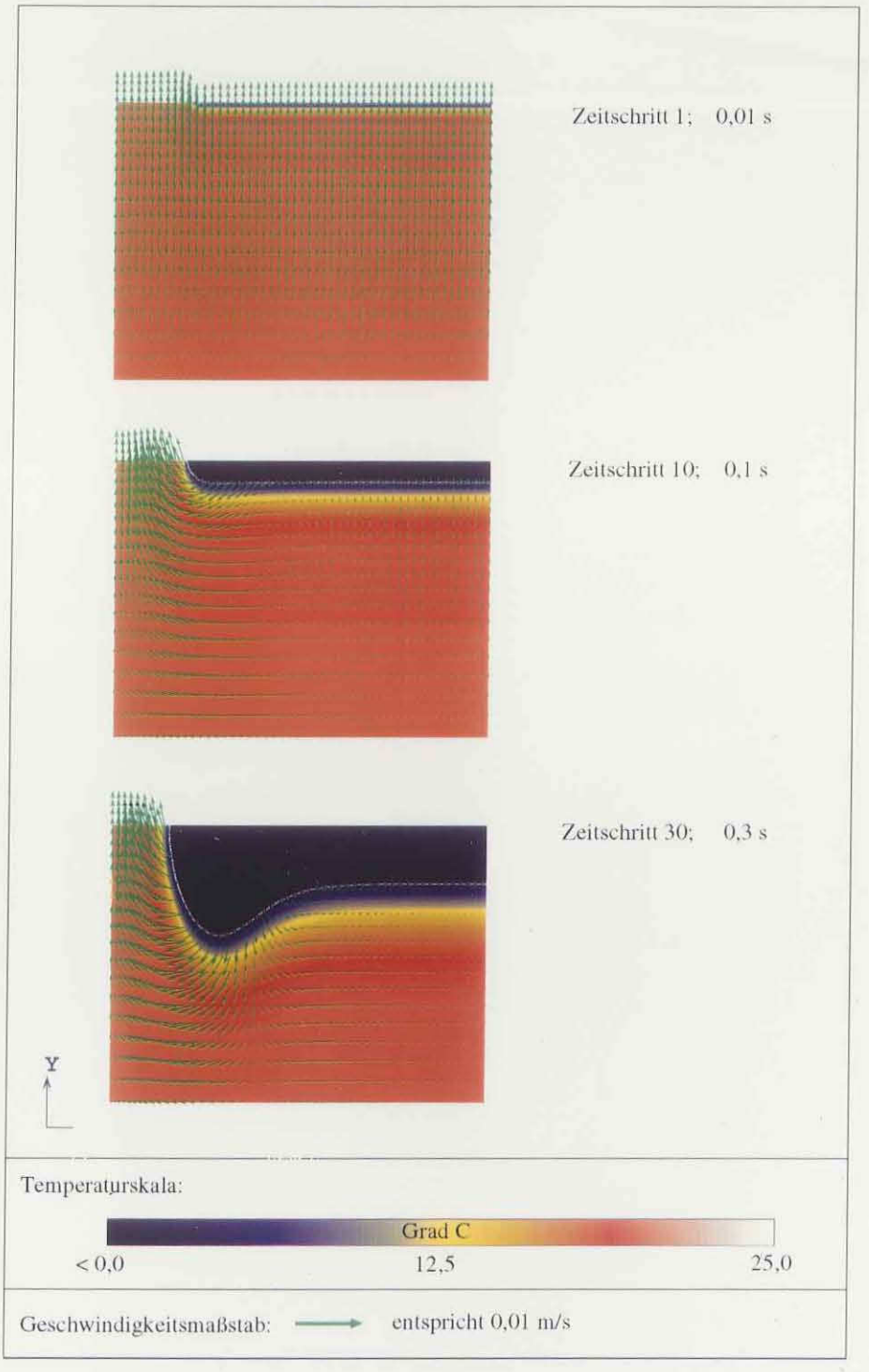


Bild 6.2.3: Auftreffen von flüssigem Wasserstoff auf Wasser

6.2.2 Speedup und Effizienz

Die Zeitbedarf für die Berechnung eines Zeitschrittes des vorangehend beschriebenen Beispiels wird in Bild 6.2.4 gezeigt. Dabei erfolgt in der linken Bildhälfte wieder die Darstellung der Rechenzeiten auf bis zu 10 Rechenknoten und in der rechten auf bis zu 100 Rechenknoten.

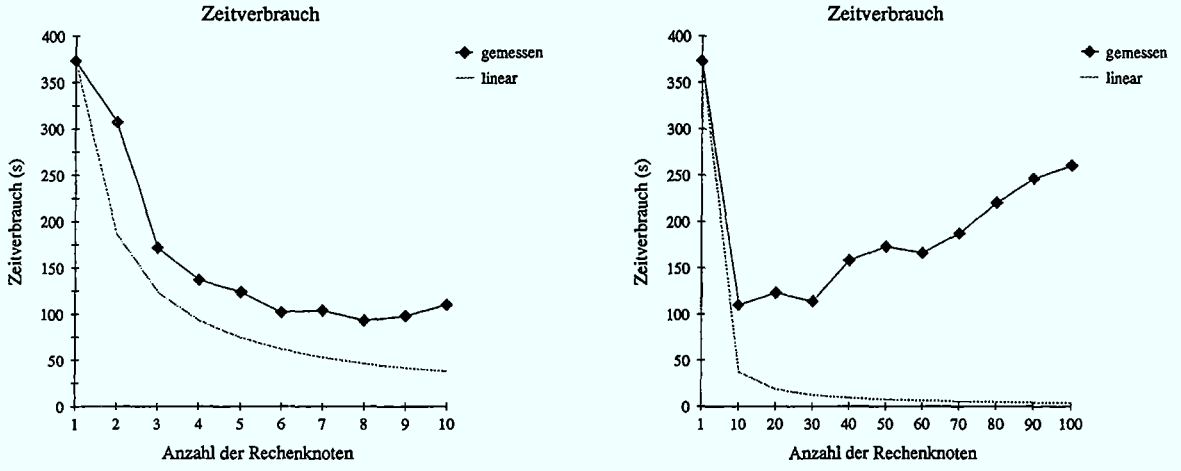


Bild 6.2.4: Zeitverbrauch innerhalb einer Zeitschleife, Wasserstoffausbreitung

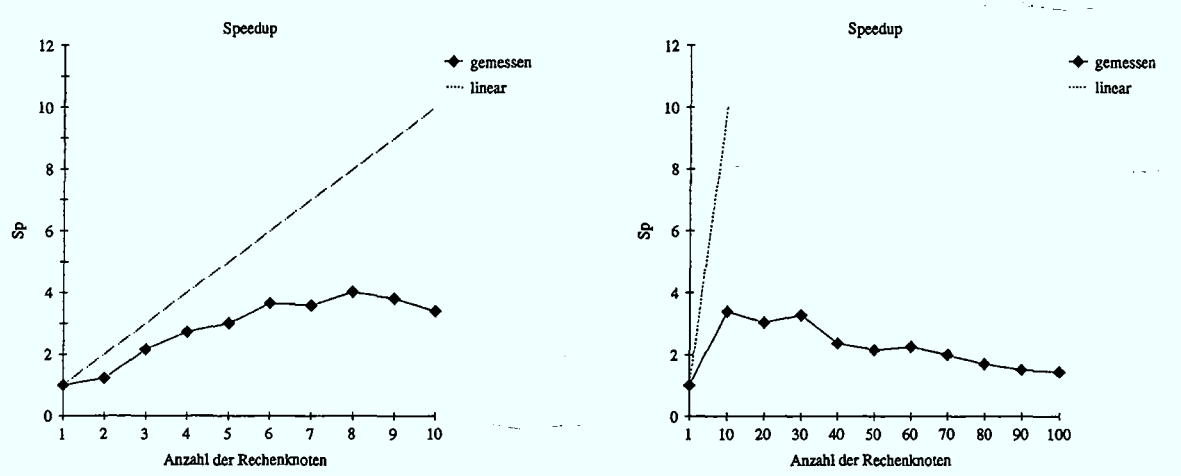


Bild 6.2.5: Speedup innerhalb einer Zeitschleife, Wasserstoffausbreitung

Daß die Rechenzeiten deutlich über den erreichbaren Werten liegen, wird in diesem Beispiel durch Datentransfer zum Hintergrundspeicher bedingt. Dieser Datentransfer ist zusätzlich erforderlich, um den Phasenübergang von Wasser zu Eis zu bestimmen. Insbe-

sondere wirken sich die Datentransferzeiten in diesem Beispiel auf den erreichbaren Leistungszuwachs aus. Dabei ist zu beachten, daß das Konzept gleichberechtigter Knoten eine Datei für jeden Rechenknoten auf dem Hintergrundspeicher erfordert. Die bereits in Abschnitt 5.3.1 beschriebene Problematik des so durchgeführten, parallelen Datentransfers erhöht die Ausführungszeiten bei dem Einsatz von mehr als 10 parallel arbeitenden Rechenknoten erheblich.

Bild 6.2.6 zeigt die Auslastung der Rechenknoten. Es wird deutlich, daß bereits ab 8 Rechenknoten nur noch eine Auslastung unter 50% erreicht wird. Ab zehn Rechenknoten sinkt die Auslastung weiter ab, so daß der Einsatz von mehr als zehn Rechenknoten nicht mehr sinnvoll erscheint, zumal auch keine weitere Verkürzung der Rechenzeiten möglich ist. Die Werte für Speedup und Effizienz werden auch für mehr als 10 Rechenknoten aufgezeigt, um die Problematik des Parallelrechnereinsatzes für dieses Beispiel zu demonstrieren.

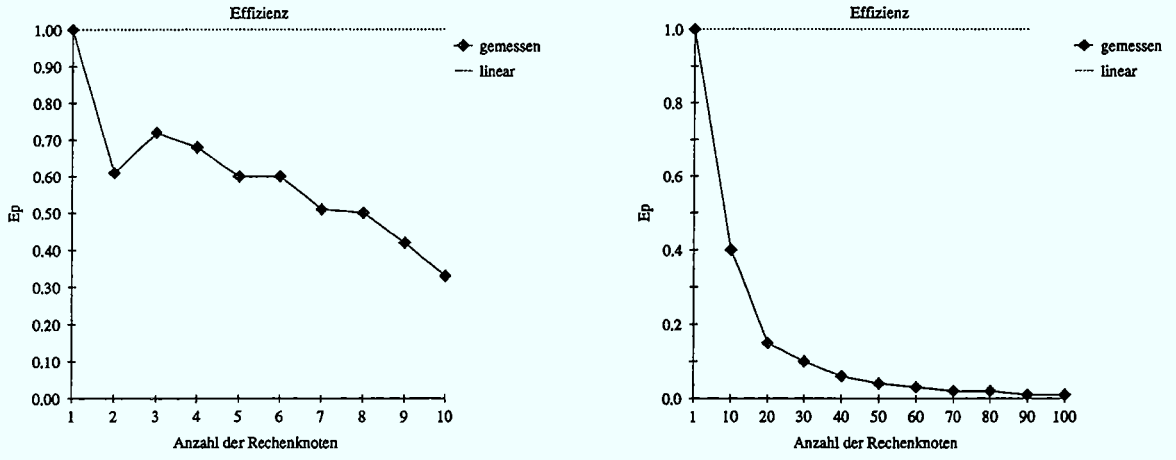


Bild 6.2.6: Effizienz innerhalb einer Zeitschleife, Wasserstoffausbreitung

Wird die Abschätzung für eine optimale Anzahl der Rechenknoten aus Abschnitt 5.4 auf dieses Beispiel angewendet, so ergibt sich für $\beta^* \approx 374s$ und den für Beispiel 6.1 bestimmten Parameter $\gamma^* = 2$ für die optimale Anzahl der Rechenknoten 13. Dabei bleibt jedoch der in diesem Beispiel zusätzlich erforderliche Datentransfer unberücksichtigt, der eine Verringerung dieser Rechenknotenanzahl bewirkt.

Für die Nachrechnung des vorangehend beschriebenen Experimentes ist im Institut für Sicherheitsforschung und Reaktortechnik ein weiteres Programmsystem eingesetzt worden. Dieses berechnet jedoch nur den Wärmeübergang unter Vernachlässigung der einsetzenden Konvektionsströmung und des Phasenübergangs. Dabei wird die Ausbreitung des flüssigen Wasserstoffes über einen Zeitraum von 500 Sekunden betrachtet.

Auch für dieses Beispiel wurde ein serieller Vergleichslauf mit SMART durchgeführt und die Ergebnisse mit denen des parallelen Laufs verglichen. Dabei liegt die Differenz der Ergebnisse unter 10^{-4}.

6.3 Laminarflügel

Im folgenden wird die Umströmung eines Laminarflügels betrachtet. Das Beispiel zeigt die Berechnung einer erzwungenen Konvektionsströmung mit ParSmart. Dabei wird eine Anströmgeschwindigkeit sowie eine Temperaturdifferenz vorgegeben.

6.3.1 Problembeschreibung

Als Beispiel für die erzwungene Konvektion wird im folgenden eine Umströmung eines Tragflügelprofils betrachtet. Die Flügelgeometrie wird entsprechend [KOS87] gewählt. Die Diskretisierung durch 8894 Viereckselemente und die Randbedingungen werden in Bild 6.3.1 gezeigt.

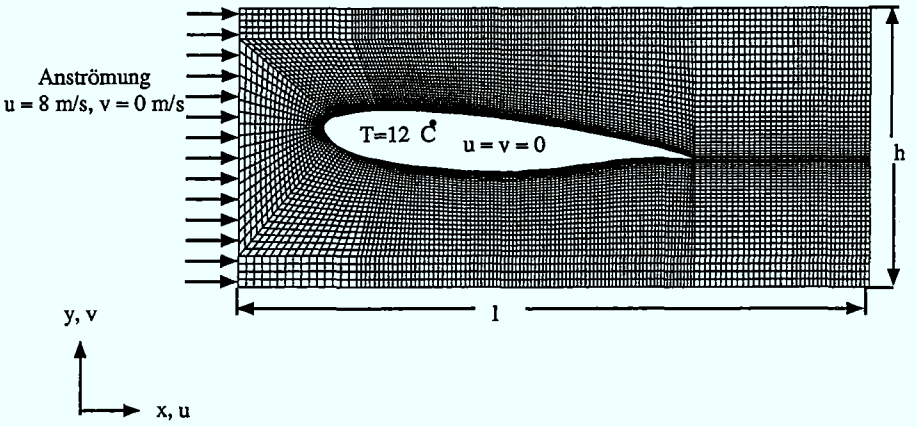


Bild 6.3.1: Diskretisierung und Randbedingungen des Tragflügelmodells

Die Länge beträgt in dem betrachteten Beispiel $l = 4,1m$ und die Höhe $h = 1,8m$. Die Geschwindigkeiten an der Flügeloberfläche sind Null. Die Temperatur an der Flügelkontur wird mit $12°C$ vorgegeben, während die Umgebungstemperatur im Anfangszustand $10,0°C$ beträgt. Der Tragflügel wird mit einer Geschwindigkeit von $8m/s$ unter einem Winkel von $0°$ angeströmt.

Als Anfangsbedingung wird ein Geschwindigkeitsfeld von $2m/s$ vorgegeben. Aufgrund der geringen Strömungsgeschwindigkeiten kann in diesem Fall inkompressibles Verhalten angenommen werden [BS94]. Um den Einfluß des Geschwindigkeitsfeldes auf das Temperaturfeld anschaulich zu demonstrieren wird für das nachfolgend betrachtete Beispiel ein Modellfluid angenommen: $\mu = 17,74 \cdot 10^{-4}$ Pa s, $\rho = 1,23$ kg/m^3, $c_T = 1,238$ J/kgK, $k_T = 24,94 \cdot 10^{-2}$ W/mK, $\beta = 3,543 \cdot 10^{-3}$ 1/K und $g = 9,81$ m/s^2.

Bild 6.3.2 zeigt die Geschwindigkeits- und Temperaturfelder bei der Umströmung des Laminarflügels zu vier ausgewählten Zeitpunkten. In der linken Bildhälfte werden jeweils Geschwindigkeits- und Temperaturfeld dargestellt. Um einen besseren Überblick über das Temperaturfeld zu erhalten, wird dieses noch einmal in der rechten Hälfte für den entsprechenden Zeitschritt abgebildet.

Nach einem Zeitschritt, d.h. $0,1s$, hat sich der vorgegebene Zustand nur wenig verändert. Nach $2,0$ Sekunden wird eine Erwärmung des Fluids um den Tragflügel sichtbar. Durch die vorgegebene Strömung wird das erwärmte Fluid entlang des Tragflügels in Strömungsrichtung transportiert. Nach etwa 9 Sekunden strömt das erwärmte Fluid hinter dem Flügel ab.

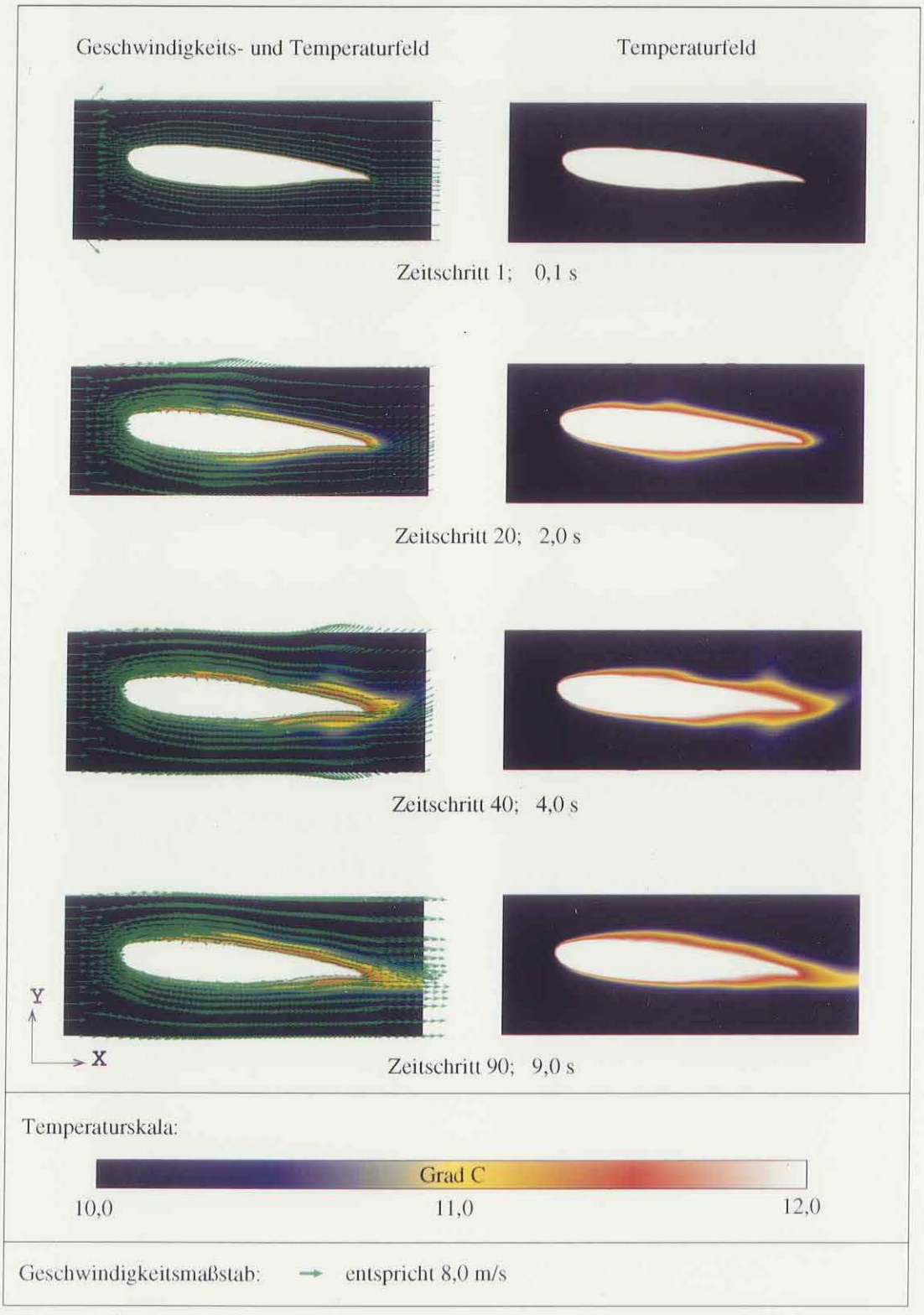


Bild 6.3.2: Geschwindigkeits- und Temperaturfelder der Flügelumströmung

6.3.2 Speedup und Effizienz

Im folgenden wird der Leistungszuwachs für die Berechnung der Flügelumströmung diskutiert. Dabei zeigt sich wie auch in den vorangehenden Beispielen auf bis zu zehn Rechenknoten ein guter Leistungszuwachs. Der Einsatz der verfügbaren 140 Rechenknoten kann auch für dieses Beispiel nicht effizient erfolgen, da die Problemgröße für diese Anzahl an Rechenknoten zu gering ist. Dieses ist, wie auch in den vorangehend beschriebenen Beispielen, auf die kleine Dimension (9087) der zu lösenden Matrix mit dem Besetzungsgrad 0, 09% für das Temperaturfeld in jedem Zeitschritt sowie den steigenden Kommunikationsaufwand zurückzuführen.

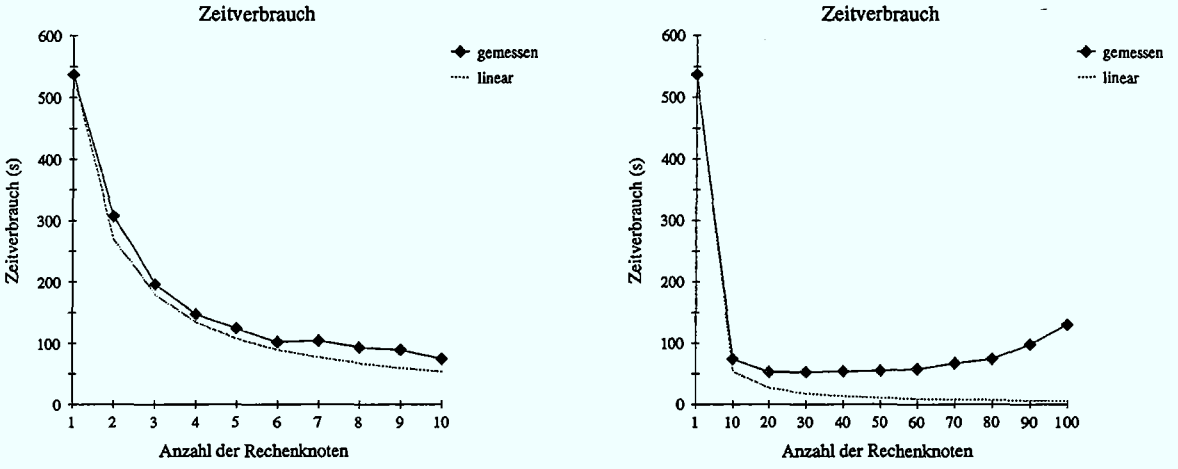


Bild 6.3.3: Zeitverbrauch innerhalb einer Zeitschleife, Flügelumströmung

Bild 6.3.3 zeigt die Rechenzeit bei einmaliger Ausführung der Zeitschleife.

Auch bei diesem Beispiel nehmen die Rechenzeiten ab etwa 30 Rechenknoten wieder zu. Der in Bild 6.3.4 gezeigte Leistungszuwachs beträgt etwa 10 bei Einsatz von 20 Rechenknoten.

In Bild 6.3.5 wird die Auslastung der eingesetzten Rechenknoten gezeigt. Diese liegt bei der Nutzung von 20 Rechenknoten bereits bei 50%. Der Anwender, der an kurzen Rechenzeiten interessiert ist, wird das Programm in diesem Fall auf etwa 20 Rechenknoten starten und dabei die Effizienz von 50% akzeptieren.

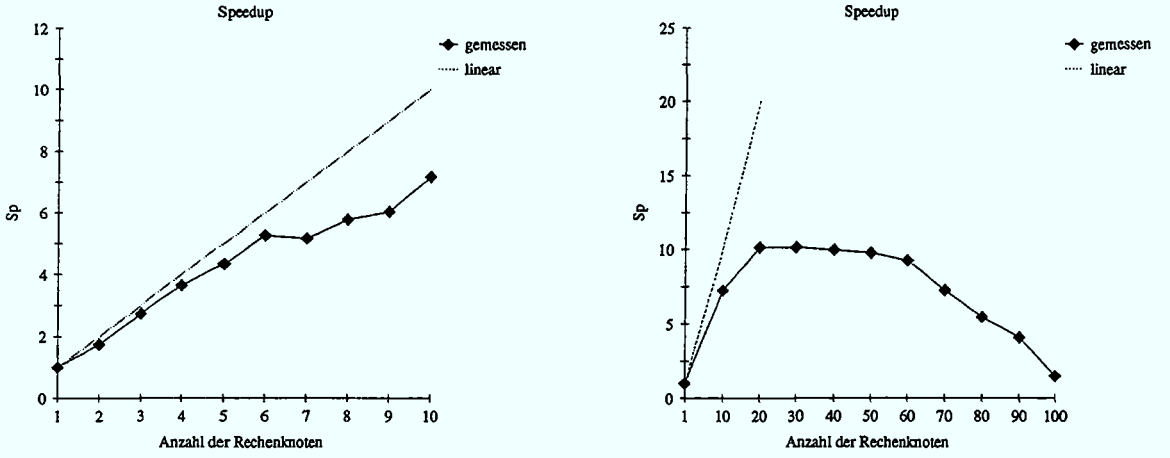


Bild 6.3.4: Speedup innerhalb einer Zeitschleife, Flügelumströmung

Nach Abschnitt 5.4 kann eine Anzahl von 17 Rechenknoten als optimal angenommen werden, wenn eine minimale Gesamtrechenzeit innerhalb der Zeitschleife angestrebt wird.

Auch für dieses Beispiel wurde ein serieller Vergleichslauf mit SMART durchgeführt und die Übereinstimmung der Ergebnisse festgestellt.

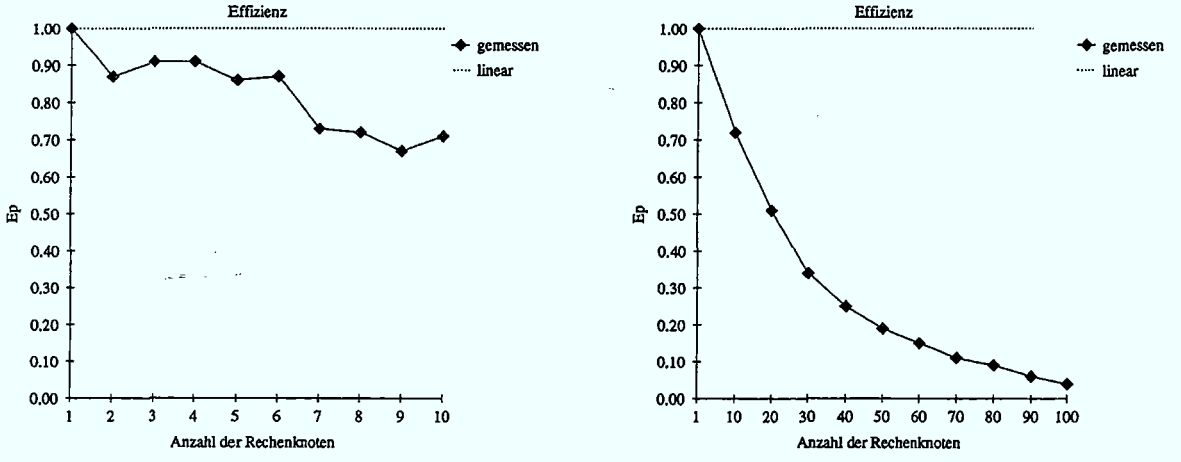


Bild 6.3.5: Effizienz innerhalb einer Zeitschleife, Flügelumströmung

6.4 Zusammenfassende Bewertung der Beispielrechnungen

Die vorangehend betrachteten Beispielrechnungen zeigen innerhalb der Zeitschleife ein ähnliches Verhalten für den Leistungszuwachs. Dabei ist eine maximale Beschleunigung um den Faktor 12 für das Gesamtsystem möglich. Dieses erscheint aufgrund der

Einsatzmöglichkeiten von 140 Rechenknoten zunächst unbefriedigend. Da mit SMART-Konvektion derzeit nur Finite-Element-Netze mit maximal 32767 Unbekannten, das sind bei dreidimensionalen Modellen Netze mit maximal 10922 Element-Knoten, berechnet werden können, sind nur 'relativ kleine' Beispiele betrachtet worden. Dabei zeigen sich für einzelne Berechnungsabschnitte die erwarteten Leistungszuwächse.

Prinzipiell wird der Zeitverbrauch für die parallelen Berechnungen durch den Datentransfer zum Hintergrundspeicher beeinflußt. Durch das IO-Konzept, welches die Datenbasis auf einen erweiterten Hauptspeicherbereich abbildet, kann der Datentransfer gegenüber einem Standard SMART-Lauf reduziert werden. Für die beschriebenen Probleme läßt sich das *Paging* jedoch nicht vollständig vermeiden. Mittels der dynamischen Speicherwaltung werden Speicherplätze erst während der Laufzeit angelegt. Dieses wirkt sich in ähnlicher Weise negativ auf den erreichbaren Leistungszuwachs aus, da zuvor andere Speicherbereiche aus dem Hauptspeicher ausgelagert werden müssen.

Einen weiteren Einflußfaktor auf den erreichbaren Leistungszuwachs bildet die Kommunikation in den implementierten Algorithmen. Insbesondere die Synchronisation der Rechenknoten, wie sie zur Überprüfung globaler Konvergenz erforderlich ist, wirkt sich negativ auf den Leistungszuwachs aus. Für diese Synchronisation kommt es zu Wartezeiten auf einzelnen Rechenknoten, da die Berechnung erst weitergeführt werden kann, wenn alle an der Rechnung beteiligten Rechenknoten diesen Synchronisationspunkt erreicht haben. Durch den aufwendigen Datentransfer durch das Betriebssystem benötigen die Rechenknoten auch bei gleichmäßiger Lastverteilung teilweise unterschiedliche Ausführungszeiten für die lokal durchzuführenden Anweisungen. Die Kommunikation zur Assemblierung der Matrizen wird in ParSmart hingegen asynchron durchgeführt und mit den Berechnungen überlappt.

Der Einsatz von bis zu zehn Rechenknoten zeigt für die betrachteten Beispiele einen guten Leistungszuwachs. Die bislang mit SMART berechenbaren Probleme ermöglichen jedoch keine effiziente Nutzung der 140 Rechenknoten des Intel/Paragon. Die Anzahl der effizient einsetzbaren Rechenknoten wird insbesondere von der Bestimmung der Temperaturen in jedem Zeitschritt beeinflußt. Bei einem Freiheitsgrad je Elementknoten liegt die Dimension des zu lösenden Gleichungssystems dann bei etwa 10000. Werden 30 Rechenknoten eingesetzt, so verwaltet jeder Rechenknoten etwa 335 Zeilen der Gesamtmatrix mit jeweils maximal 9 Einträgen. Die aufwendigste Anweisung in einer Iteration des konjugierten Gradienten-Verfahrens, die Matrix Vektor Multiplikation, benötigt dann noch 3015 Skalarmultiplikationen. Der Zeitverbrauch wird bei der Lösung dieser dünnbesetzten Matrizen kleiner Dimension durch die Kommunikation und Synchronisation der Rechenknoten innerhalb der Iterationsschleife bestimmt.

Werden die vorangehenden Aspekte bei der Bewertung des Leistungszuwachses berücksichtigt, so kann die erreichte Verkürzung der Rechenzeiten durchaus positiv bewertet werden. Das parallele Programmsystem ParSmart ist auf dem massiv parallelen Rechner Intel/Paragon einsetzbar. Allerdings scheint die effiziente Nutzung der verfügbaren 140 Rechenknoten nur für entsprechend große Modelle möglich.

7 Metacomputer für Finite-Element-Berechnungen

Komplexe Programmsysteme weisen im allgemeinen in den einzelnen Programmabschnitten unterschiedliche Daten- und Programmstrukturen auf [MOT86]. Manche Teilberechnungen lassen sich daher effizient auf massiv parallelen Rechnersystemen ausführen, während andere Abschnitte besser auf Vektorrechnern oder seriell ausführbar sind. Eine optimale Rechnerarchitektur würde für jeden Berechnungsabschnitt die geeigneten Komponenten bereitstellen [FS93], [DBP94], [SH93]. Diese Anforderung wird von heterogenen Systemen oder Metacomputern erfüllt [HN95], [Hos95], [KPSW93].

Im folgenden wird untersucht, wie heterogene Architekturen, die aus einer Vernetzung von Skalar-, Vektor- und Parallelrechnern bestehen, für Finite-Element-Berechnungen mit SMART eingesetzt werden können.

7.1 Heterogene Systeme

Ein heterogenes System besteht aus der Verbindung verschiedener Rechnerarchitekturen über ein leistungsfähiges Netzwerk [GWWL94]. So können beispielsweise Vektorrechner, wie Cray Y-MP, Workstation-Cluster, massiv parallele Systeme wie Intel/Paragon oder Connection Machine und herkömmliche Skalarrechner ein heterogenes System bilden. Dem Anwender wird eine Gesamtarchitektur zur Verfügung gestellt, die es ermöglicht, jeden Teilabschnitt eines Programmsystems auf der am besten dafür geeigneten Komponente auszuführen.

Die derzeit im Forschungszentrum Jülich installierten Rechner können ebenfalls als ein heterogenes System betrachtet werden. Eine mögliche Konfiguration würde beispielsweise den Vektorrechner Cray Y-MP M94, den massiv parallelen Rechner Intel/Paragon, die RISC-Prozessoren der IBM-SP2 und weitere lokale Workstations oder Workstationgruppen umfassen (vgl. Bild 7.1.1). Zu der Zusammenfassung von Rechnern verschiedener Hersteller bilden heterogene Systeme eines Herstellers eine Alternative. So werden beispielsweise von Cray Research Rechnersysteme angeboten, die Vektorrechner und massiv parallele Systeme verbinden. Eine mögliche Konfiguration bilden die Vektorrechner Y-MP und das massiv parallele System T3E [CRA94].

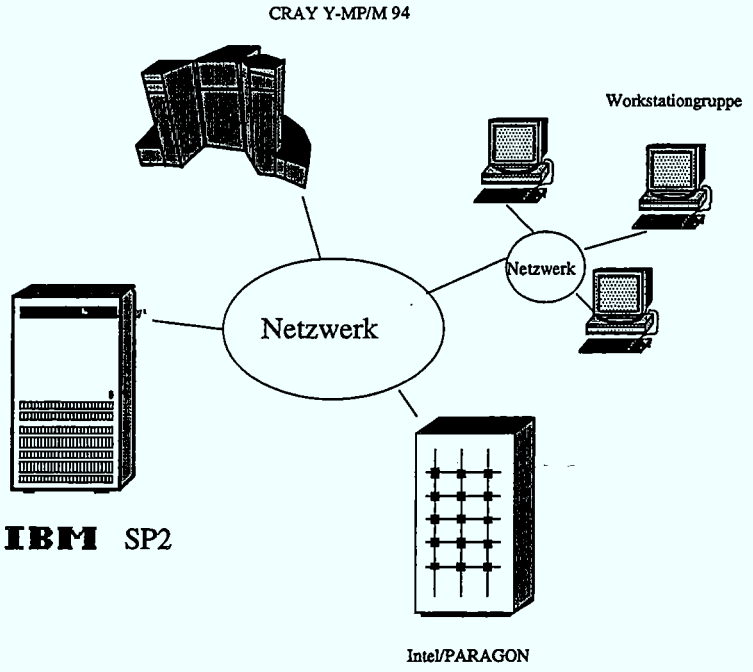


Bild 7.1.1: Heterogenes System aus einigen der in der KFA installierten Rechnern

7.2 Eine optimale Hardwareumgebung für ParSmart

Finite-Element-Berechnungen bestehen aus Berechnungsabschnitten, die unterschiedliche Anforderungen an die verfügbaren Rechner stellen. So wurden bereits in [Ful86] Untersuchungen über eine optimale Rechnerarchitektur für Finite-Element-Berechnungen durchgeführt.

Im folgenden soll speziell für den Finite-Element-Code SMART analysiert werden, wie eine Verkürzung der Rechenzeiten durch den Einsatz heterogener Architekturen erzielt werden kann.

Während bei der homogenen Architektur des Rechners Intel/Paragon für alle Programmabschnitte die Rechenknoten mit gleicher Leistung eingesetzt werden, wird bei heterogenen Rechnersystemen jeder Programmabschnitt auf der Architektur ausgeführt, die den größten Leistungszuwachs ermöglicht; das bedeudet die Verwirklichung eines die Problemlösung optimierenden Arbeitsteilungsprozeß etwa zwischen Skalar-, Vektor- und Parallelrechner. Nachfolgende Betrachtungen beziehen sich auf eine SMART-Berechnung nach dem in Bild 3.4.1 dargestellten Ablauf.

Dateninitialisierung:

Für die Dateninitialisierung muß bei SMART stets die Modellbeschreibung eingelesen werden. Für diesen Abschnitt sind Komponenten mit einer Übertragungsbandbreite zum Hintergrundspeicher erforderlich, die eine schnelle Dateninitialisierung gewährleisten. Bei heterogenen Systemen sollte die bei Berechnungen mit SMART aufgestellte Datenbasis

auf einer Kompontente mit großem Hauptspeicherbereich abgelegt werden, auf die jeder Rechner zugreifen kann, also ein virtueller gemeinsamer Speicher zwischen den heterogenen Rechnerkomponenten. Ausgehend von einer solchen Datenorganisation muß jeder Rechner des heterogenen Systemkomplexes auf diese Datenbasis zugreifen können.

Berechnungen auf Elementebene:

In SMART werden die Elementmatrizen durch die in (3.1.15)-(3.1.19) und (3.1.30)-(3.1.33) beschriebenen Ansätze aufgestellt. Diese Berechnungen können lokal ohne Kommunikation durchgeführt werden. Für diesen Abschnitt sind massiv parallele Rechner geeignet, da die Elemente jeweils von den einzelnen Rechenknoten bearbeitet werden können. Für diese Programmabschnitte ist aus algorithmischer Sicht keine Kommunikation erforderlich. Allerdings müssen die erforderlichen Daten jeweils auf den entsprechenden Rechenknoten verfügbar sein. Dieses erfordert schnelle Kommunikationsmöglichkeiten zwischen dem Rechner, auf dem die Datenbasis verwaltet wird, und den Prozessoren des Parallelrechners.

Assemblierung der Systemmatrix:

Bei der Assemblierung der Elementmatrizen müssen die Abhängigkeiten benachbarter Elemente berücksichtigt werden. Werden die im vorangehenden Schritt bestimmten Elementmatrizen verteilt gespeichert, so wird für diese Phase ein Kommunikationsnetzwerk benötigt, welches einen schnellen Datenaustausch ermöglicht.

Lösung der Gleichungssysteme:

Die Lösung der Gleichungssysteme wird im Original-SMART-Code durch das Cholesky-Verfahren durchgeführt. Wird dieses beibehalten, so können für die Lösung Rechner wie die Cray Y-MP genutzt werden. Eine höherer Leistungszuwachs ist durch den Einsatz massiv paralleler Systeme möglich. Diese Systeme setzen jedoch ein anderes Lösungsverfahren voraus, wie z.B. konjugierte Gradienten-Verfahren.

Datenausgabe:

Die Ausgabe der Lösungsvektoren für Geschwindigkeits- und Temperaturfelder erfordert, wie auch die Dateneingabe, eine hohe Übertragungsrate zu einem Hintergrundspeicher. Bei instationären Berechnungen mit SMART müssen in jedem Zeitschritt die Ergebnisvektoren für Temperatur- und Geschwindigkeitsfeld ausgegeben werden. Deshalb sind diese Lösungsvektoren auf Rechnerkomponenten mit hoher IO-Leistung zu speichern, um die Berechnungen nicht durch lange Zugriffszeiten zum Hintergrundspeicher zu unterbrechen.

In Tabelle 7.2.1 wird eine heterogene Systemarchitektur für Finite-Element-Berechnungen mit SMART zusammengefaßt. Es wird deutlich, daß massiv parallele Rechner für umfassende Berechnungsabschnitte eingesetzt werden können. Allerdings sind wegen der engen Kopplung der Probleme sehr schnelle Verbindungsnetzwerke erforderlich.

Programmabschnitt	Rechnerarchitektur
Dateninitialisierung:	Skalarrechner mit hoher Kommunikations- und IO-Leistung
Berechnungen auf Elementebene:	massiv parallele Rechner
Assemblierung der Systemmatrix:	massiv parallele Rechner
Lösung der Gleichungssysteme:	massiv parallele Rechner
Datenausgabe:	Skalarrechner mit hoher Kommunikations- und IO-Leistung
Ergebnisdarstellung:	Vektor- oder Parallelrechner

Bild 7.2.1: Optimale Rechnerarchitektur für Finite-Element-Berechnungen

Prinzipielle Anforderungen an die Architektur bestehen in hoher Rechen- **und** IO-Leistung. Zum Einlesen komplexer Problembeschreibungen und zur Ausgabe der berechneten Ergebnisse für eine graphische Aufbereitung ist eine Komponente mit großer Übertragungsrate zu Hintergrundspeichern mit einigen TByte Plattenkapazität erforderlich.

Die Interpretation der berechneten Ergebnisse ist meist nur noch über graphische Hilfsmittel möglich, so daß ein Hochleistungsgraphiksystem für Zwecke des Postprocessing verfügbar sein muß. Für diese Aufgabe lassen sich abhängig von der Datenmenge Vektor- oder Parallelrechner einsetzen.

8 Zusammenfassung und Ausblick

8.1 Zusammenfassung

Im Rahmen dieser Arbeit ist ein Konzept zur Portierung des seriellen Finite-Element-Codes SMART-Konvektion auf den massiv parallelen Rechner Intel/Paragon entwickelt, implementiert und bewertet worden. Unter dem Aspekt, Erfahrungen bei der Parallelisierung bestehender Produktionscodes zu erarbeiten, werden Wege, Möglichkeiten und Grenzen der Nutzung des Rechners Intel/Paragon für Finite-Element-Rechnungen mit SMART aufgezeigt.

Entscheidend für die Wahl des Konzepts gleichberechtigter Knoten für die Parallelversion ParSmart ist das Verfahren zur Lösung der Gleichungssysteme, das etwa 90% der Gesamtrechenzeit einer SMART-Rechnung ausmacht. Hier wurde eine algorithmische Änderung dahingehend vorgenommen, daß das direkte Cholesky-Verfahren durch ein iteratives und somit effizient parallel ausführbares konjugiertes Gradienten-Verfahren ausgetauscht wurde. Für die notwendige Datenaufteilung auf die Rechenknoten ist ein bekanntes Modellzerlegungsverfahren implementiert worden, das vor einer SMART Rechnung das Gesamtmodell in Teilmodelle zerlegt und somit die Zuordnung der Daten zu den Rechenknoten für den folgenden Programmablauf festlegt. Somit können mit ParSmart Modelle mit etwa 4500000 Freiheitsgraden, statt wie bisher 32767 Unbekannten berechnet werden.

Die Teilmodelle werden weitgehend lokal berechnet. Da die Abhängigkeiten der Teilmodelle untereinander zu berücksichtigen sind, ist Datentransfer zwischen den Rechenknoten notwendig. Dies geschieht durch ein für ParSmart entwickeltes Kommunikationsschema. Nach der Aktualisierung der Teilmodelle durch Daten benachbarter Modelle erfolgt die Lösung des Gleichungssystems für das Gesamtmodell durch das parallele konjugierte Gradienten-Verfahren.

Die ergänzend eingeführten Algorithmen erforderten eine Erweiterung der in SMART implementierten Datenstrukturen. Dazu sind Zuordnungstabellen, ein erweitertes Hypermatrizenschema sowie ein zeilenorientiertes Speicherschema eingeführt worden.

Aus einer allgemeinen Analyse der Rechenzeiten für parallele Finite-Element-Berechnungen mit ParSmart wurde eine Abschätzung für die optimale Anzahl von Rechenknoten entwickelt, die algorithmische und architektonische Einflüsse berücksichtigt.

Zur Bewertung der erreichbaren Leistung wurden drei Beispiele berechnet. Die Rayleigh-Bénard-Konvektion als ein Beispiel, für das die analytische Lösung bekannt ist, eine Simulation der Ausbreitung flüssigen Wasserstoffs auf Wasser, die durch Vergleiche mit Versuchsergebnissen und Berechnungen durch andere Codes überprüfbar ist, und die Umströmung eines Laminarflügels.

Der erreichbare Leistungszuwachs liegt für die berechneten Beispiele maximal bei 12, wobei der Zeitverbrauch betrachtet wird, den der Anwender auf die Bearbeitung warten muß. Dabei sind jedoch nur Beispiele betrachtet worden, für die auch Vergleichsläufe mit

dem seriellen SMART Code durchgeführt werden konnten. Für größere Beispiele verspricht ParSmart auch die effiziente Nutzung der 140 Rechenknoten.

Als Alternative zu der homogenen Rechnerarchitektur des Intel/Paragon verspricht der Einsatz heterogener Systeme eine weitere Leistungssteigerung.

8.2 Ausblick

Ausgehend von den Erfahrungen, die in dieser Arbeit hinsichtlich der Parallelisierung des Programmsystems SMART für den Parallelrechener Intel/Paragon erarbeitet wurden, kann der Einsatz massiv paralleler Systeme für Berechnungen mit SMART tendenziell positiv bewertet werden.

Zukünftige Arbeiten sollten weiterhin algorithmische Verbesserungen einzelner Programmabschnitte in ParSmart behandeln. So wäre beispielsweise der Einsatz eines konjugierten Gradienten-Verfahrens für asymmetrische Gleichungssysteme sowie die dazu erforderliche Änderung der Elementansätze zu untersuchen.

Neben der Verkürzung der Rechenzeiten durch algorithmische Veränderungen ist der Leistungszuwachs durch den Einsatz heterogener Rechnersysteme für Finite-Element-Berechnungen mit SMART zu analysieren.

Anhang A Symbolverzeichnis

Symbol	Erklärung
Griechische Buchstaben	
α	Wärmeübergangskoeffizient
α^*	serieller Anteil einer parallelen Berechnung
α_V	isobarer Volumenausdehnungskoeffizient
α_i	Hilfsvariable, CG-Verfahren
β	thermischer Ausdehnungskoeffizient
β^*	linear skalierbarer Anteil einer parallelen Berechnung
β_i	Hilfsvariable, CG-Verfahren
γ^*	linear ansteigender Anteil einer parallelen Berechnung
Δ	Differenz
$\delta\cdot$	Gewichtsfunktionen
ϵ	Dissipationsrate der Turbulenzenergie
ζ	Parameter zur Zeitintegration
$\dot{\epsilon}$	Vektor der virtuellen Verzerrungsgeschwindigkeiten
ϑ	charakteristischer Temperaturunterschied, $\lvert T_W - T_\infty \rvert$
κ	Kompressionsmodul
$\tilde{\kappa}$	Penalty-Parameter
ν	kinematische Viskosität
∇	Gradient, $\{\frac{\partial}{\partial x}\frac{\partial}{\partial x}\frac{\partial}{\partial x}\}$
$\nabla\phi$	Cauchy'scher Spannungstensor, $\nabla\phi = -\nabla p + \mu\Delta v$
$\frac{\partial}{\partial t}$	partielle Zeitableitung
μ	dynamische Viskosität
ρ	Dichte
σ	Spannungsvektor

Lateinische Buchstaben

A	Systemmatrix
$\tilde{A}$	vorkonditionierte Matrix
a	Temperaturleitfähigkeit
a_{ij}	Koeffizienten der Matrix A
B	Bandbreite
b, b_i	Rechthandseiten
$\tilde{b}$	Rechthandseite nach Vorkonditionierung
C	Kapazitätsmatrix auf Systemebene
c	Element-Kapazitätsmatrix
c_1, c_2, c_3	Koeffizienten zur Bestimmung der Rechenzeit
c_T	spezifische Wärmekapazität
D	Menge der Finiten Elemente des zu berechnenden Gesamtmodells
D	Differentialoperator, Seite 26
$\tilde{D}$	Diagonalmatrix
D_i, D_j	Menge der Finiten Elemente eines Teilmodells i
D_T	Differentialoperator (Temperaturfeld), $\{\frac{\partial}{\partial x}\frac{\partial}{\partial y}\frac{\partial}{\partial z}\}$
d_i	Defekt im CG-Verfahren, Iterationsschritt i
E_p	Effizienz auf p^* Rechenknoten
$\mathbf{E_{33}}$	Tensorprodukt, Seite 26
e_k, e_j	Finites Element, Modellzerlegung
F	Oberfläche
F	Viskositätsmatrix auf Systemebene
f	Element-Viskositätsmatrix, $f_H + f_D$
$\mathbf{f_n}$	Oberflächenkräfte pro Flächeneinheit
$\mathbf{f_D}$	deviatorischer Anteil der Element-Viskositätsmatrix
$\mathbf{f_H}$	hydrostatischer Anteil der Element-Viskositätsmatrix
Gr	Grashofzahl
g	Gravitationskonstante

g_i	Gradient im CG-Verfahren, Iterationsschritt i
h	Höhe
$h_{i,p}^k$	Hilfsvariable, lokal, Iterationsschritt i
h_i^k	Hilfsvariable, global, Iterationsschritt i
I_6	Vektor $\{111111\}$
K^*	Linkhandseite (Zusammenfassung)
K	Konduktivitätsmatrix auf Systemebene
$\mathbf{k}$	Element-Konduktivitätsmatrix
k	Turbulenzenergie
k_T	Wärmeleitfähigkeit
L	untere Dreiecksmatrix
l	charakteristische Länge
M	Matrix zur Vorkonditionierung
M	Massenmatrix auf Systemebene
$\mathbf{m}$	Element-Massenmatrix
m	Anzahl der Lastvektoren/Rechthandseiten
N	Konvektionsmatrix auf Systemebene
N_e	Menge der Finiten Elemente eines Modells
N_n	Menge der Finite-Element-Knoten eines Modells
Nu	Nußeltzahl
n	Dimension des Gleichungssystems
$\mathbf{n}$	Element-Konvektionsmatrix
n_i	Finite-Element-Knoten
$\mathbf{n^a}$	asymmetrischer Anteil der Element-Konvektionsmatrix
$\mathbf{n^s}$	symmetrischer Anteil der Element-Konvektionsmatrix
Pe	Pécletzahl
Pr	Prandtlzahl
p^*	Anzahl der Rechenknoten
p_{opt}^*	optimale Anzahl der Rechenknoten bzgl. minimaler Rechenzeit
p	Druck
p_e	Rechenknoten/Empfänger
p_s	Rechenknoten/Sender

Q	volumenspezifische Leistung
$\mathbf{Q}$	Lastvektor auf Systemebene, Temperaturen
Q_0, Q_1	Wärmequelle/-senke
Q_D	Energiedissipationsrate
Q_n	Wärmeübergang, $\alpha(T - T_\infty)$
$\mathbf{q}$	Element-Lastvektor, Temperaturen
$\dot{q}$	Wärmestrom
R^*	Rechthandseite (Zusammenfassung)
$\mathbf{R}$	Lastvektor auf Systemebene, Geschwindigkeiten
Ra	Rayleighzahl
$\mathbf{R_a}$	äußerer Lastvektor, $\mathbf{R_a}= (R_x, R_y, R_z)$
Re	Reynoldszahl
R_i	Menge der inneren Randpunkte
$\mathbf{r}$	Element-Lastvektor, Geschwindigkeiten
S_p	Speedup auf p^* Rechenknoten
$S_{p,\alpha}$	Speedup auf p^* Rechenknoten bei einem seriellen Programmanteil α^*
T	Temperatur
$\dot{T}$	zeitliche Ableitung der Temperatur
T_W	Wandtemperatur
T_∞	mittlere Fluidtemperatur
t	Zeit
t^*	Näherung fur die Rechenzeit innerhalb einer Zeitschleife
U	obere Dreiecksmatrix
u	Geschwindigkeitskomponente in x-Richtung
$\mathbf{V}$	Geschwindigkeitsvektor auf Systemebene
$\dot{\mathbf{V}}$	zeitliche Ableitung der Geschwindigkeit, Systemebene
v	Geschwindigkeitskomponente in y-Richtung
$\dot{\mathbf{v}}$	zeitliche Ableitung der Geschwindigkeit, Elementebene
$\mathbf{v}$	Geschwindigkeitsvektor, $\mathbf{v}= (u, v, w)$
w	Geschwindigkeitskomponente in z-Richtung
w_T, w_V	Gewichtsfunktionen

$\mathbf{X}$	Vektor der Erdbeschleunigung, $\mathbf{X} = \{\mathbf{g_x}, \mathbf{g_y}, \mathbf{g_z}\}$
$\tilde{x}$	Lösungsvektor des vorkonditionierten Gleichungssystems
x	Lösungsvektor des Systems $Ax = b$

tiefgestellte Indizes

e	Element
i	Iterationsschritt
ij	Zeilen-/Spaltenindex einer Matrix
k	Summationsindex
T	Temperatur
V	Geschwindigkeit
0	Bezugswert / Startwert

hochgestellte Indizes

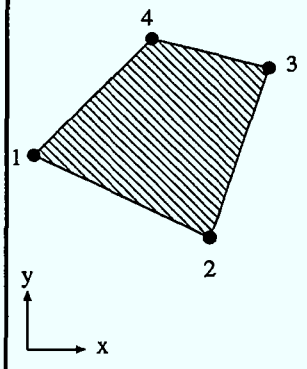
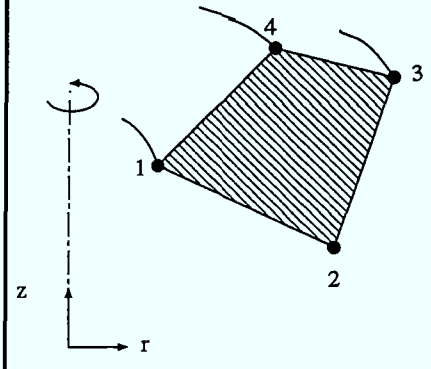
ζ	Integrationspunkt
n	Zeitschritt
p	lokal, je Rechenknoten
t	transponiert

Anhang B Abkürzungen

ASKA	Automatic System for Kinematic Analysis
BLAS	Basic Linear Algebra Subprograms
CFD	Computational Fluid Dynamics
CG	Conjugate Gradients
CPU	Central Processing Unit
DRS	Data Retrieval System
FLOPs	Floating Point Operations per second
HPF	High Performance FORTRAN
IO	Input/Output, Datentransfer zum Hintergrundspeicher
iMRC	Intel Message Routing Chip
KFA	Forschungszentrum Jülich GmbH
MPI	Message Passing Interface
$\mathcal{NP}$	Non-Polynomial, Klasse der nur nichtdeterministisch in polynomieller Rechenzeit lösbaren Probleme
NX	Node Execution
OSF/AD	Open System Foundation / Advanced Development
PARMACS	PARallel MACroS
PFS	Parallel File System
PVM	Parallel Virtual Machine
RAID	Redundant Array of Inexpensive Disks
RISC	Reduced Instruction Set Computer
SPC	SMART Procedure Call
SVM	Shared Virtual Memory
UPWD	Upwind-Faktor

Anhang C Elementbibliothek

Tabelle 9: Navier-Stokes-Elemente

	*QUAM4V*	ebenes Viereckelement mit geraden Kanten	
	Knoten	4	
	Kanten	8	u, v, z an allen Knoten
	Koordinaten		x, y an allen Knoten
	Materialdaten		
	Viskosität		μ an allen Knoten
	Penalty-Parameter		$\hat{k}$ an allen Knoten
	Dichte		ρ an allen Knoten
	Temperaturausdehnungs-koeffizient		β an allen Knoten
	QUAX4V	rotationssymmetrisches Viereckselement mit geraden Kanten	
	Knoten	4	
	Kanten	8	u_r, u_z an allen Knoten
	Materialdaten		
	Viskosität		μ an allen Knoten
	Penalty-Parameter		$\hat{k}$ an allen Knoten
	Dichte		ρ an allen Knoten
	Temperaturausdehnungs-koeffizient		β an allen Knoten

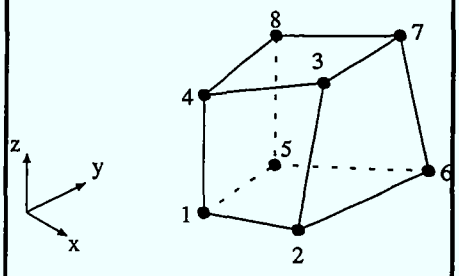
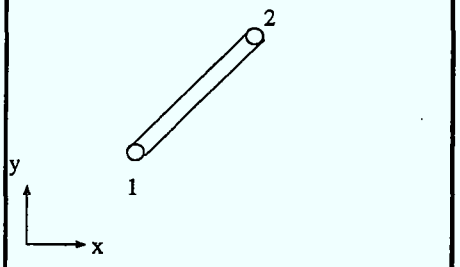
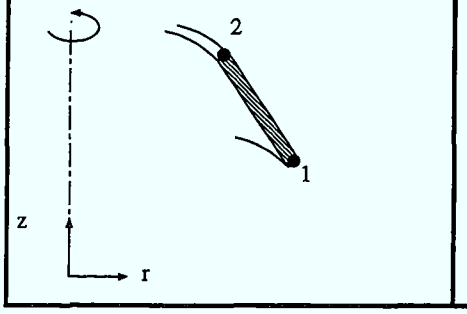
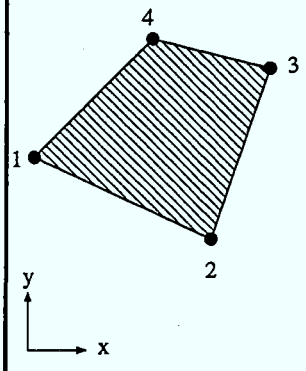
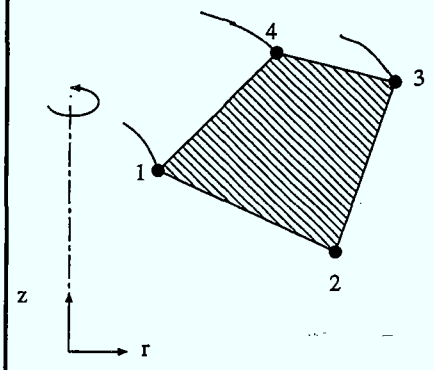
	*HEXE8V*	Hexaeder-Vollkörperelement mit geraden Kanten	
	Knoten	8	
	Kanten	24	u, v, w an allen Knoten
	Materialdaten		
	Viskosität		μ an allen Knoten
	Penalty-Parameter		$\hat{k}$ an ellen Knoten
	Dichte		ρ an allen Knoten
	Temperaturausdehnungs-koeffizient		β an allen Knoten
	*FLA2V*	gerades Stabelement	
	Knoten	2	
	Freiheitsgrade	4	u, v an allen Knoten
	Koordinaten		
	Materialdaten		entfallen
	Druck- und Linienlasten		p, l an allen Knoten
	*FLAX2V*	rotationssymmetrisches gerades Membranschalenelement	
	Knoten	2	
	Freiheitsgrade	4	u_r, u_z an allen Knoten
	Koordinaten		v, z an allen Knoten
	Materialdaten		entfallen
	Druck- und Schublasten		p, l an allen Knoten

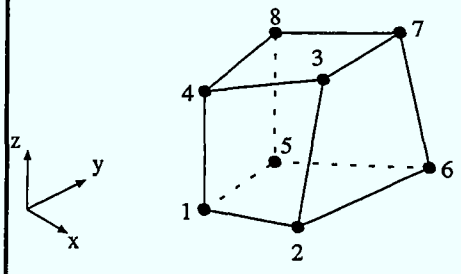
Tabelle 10: Diffusionselemente



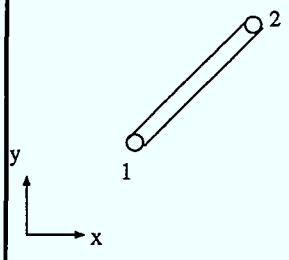
QUAM4D	ebenes Viereckselement mit geraden Kanten	
Knoten	4	
Kanten	4	T an allen Knoten
Koordinaten		x, y, z an allen Knoten
geometrische Daten		
Dicke		t an allen Knoten
Materialdaten		
Materialgesetz isotrop		k_T Konduktivität an allen Knoten
Temperaturausdehnungs-koeffizient		β an allen Knoten
Kapazität		$\rho \cdot c_T$ an allen Knoten
Lastarten		
Flächenströme		f an allen Knoten
Volumenströme		v an allen Knoten
Übergangskoeffizienten und Quellenkoeffizienten		α, β an allen Knoten



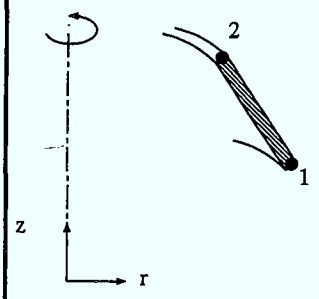
QUAX4D	rotationssymmetrisches Viereckselement mit geraden Kanten	
Knoten	4	
Freiheitsgrade	4	
Koordinaten		r, z an allen Knoten
Materialdaten		
Materialgesetz isotrop		k_T Konduktivität an allen Knoten
Temperaturausdehnungs-koeffizient		ϵ an allen Knoten
Kapazität		$\rho \cdot c_T$ an allen Knoten
Lastarten		
Volumenströme		v an allen Knoten
Quellenkoeffizienten		β an allen Knoten



HEXE8D	gerades Hexaeder-Vollkörperelement	
Knoten	8	
Freiheitsgrade	8	T an allen Knoten
Koordinaten		x, y, z an allen Knoten
Materialdaten		
Materialgesetz isotrop		k_T Konduktivität an allen Knoten
Temperaturausdehnungs-koeffizient		β an allen Knoten
Kapazität		$\rho \cdot c_T$ an allen Knoten
Lastarten		
Volumenströme		v an allen Knoten
Quellenkoeffizienten		β an allen Knoten



FLA2D	gerades Stabelement mit variabler Querschnittsfläche	
Knoten	2	
Freiheitsgrade	2	T an allen Knoten
Koordinaten		x, y an allen Knoten
geometrische Daten		
Dicke		t an allen Knoten
Materialdaten		
Materialgesetz isotrop		k_T Konduktivität an allen Knoten
Temperaturausdehnungs-koeffizient		β an allen Knoten
Kapazität		$\rho \cdot c_T$ an allen Knoten
Lastarten		
Flächenströme		f an allen Knoten
Volumenströme		v an allen Knoten
Übergangs-koeffizienten und Quellenterme		α, β an allen Knoten



FLA2XD	rotationssymmetrisches gerades Membranschalenelement mit variabler Dicke	
Knoten	2	
Freiheitsgrade	2	T an allen Knoten
Koordinaten		r, z an allen Knoten
geometrische Daten		
Dicke		t an allen Knoten
Materialdaten		
Materialgesetz isotrop		k_T Konduktivität an allen Knoten
Temperatur-Ausdehnungskoeffizient		β an allen Knoten
Kapazität		$\rho \cdot c_T$ an allen Knoten
Lastarten		
Flächenströme		f an allen Knoten
Volumenströme		v an allen Knoten
Übergangskoeffizienten und Quellenterme		α, β an allen Knoten

Anhang D SMART Eingabedatensätze

Beispiel eines Eingabedatensatzes für das Temperaturnetz [LS91]

```
NUMNET=  1     ROT   COOR COUP
TOPOLOGY
NET             2     1   22     1
QUAM4D          1     0   10
                1     3    4     2
INKR           10     2
ENDTOP
FREEDOMS
PRESCRIB 1      1     2     1
         1     21     2     1
ENDFRE
COOR       2
G          1          0.          0.          0.
R          2
           1        0.1
Z         11
           2        0.1
*
PVAL       2          1                      !vorgeschriebene Temperaturen
           1         0.
           2         0.
          21       100.
          22       100.
*
INIC       2    -1    1                      !Anfangszustand
           1         0.
*
CONI       2     1   -1    -1                !Konduktivitaet
           1     0.5683
*
CAPA       2     1   -1    -1                !Kapazitaet
           1    4.217E6
*
GEDA       2     1   -1    -1                !Elementdicke
           1         1.
*
FLOW       2     1   -1    -1     1          !Waermestrom
           1         0.
*
FIN
```

Beispiel eines Eingabedatensatzes für das Geschwindigkeitsnetz [LS91]

```
NUMNET=  1     ROT  COOR
TOPOLOGY
NET             1    1   22    1
QUAM4V          1    0   10
                1    3    4    2
INKR           10    2
ENDTOP
FREEDOMS
SUPPRESS 1      1   22    1
PRESCRIB 2      1   22    1
ENDFRE
COOR      1
G         1         0.         0.         0.
R         2
          1        0.1
Z        11
          2        0.1
*
INIV      1   -1    1                     ! Anfangszustand der Geschwindigkeiten
          1         0.         0.
*
USRP      1   -1    1                     ! vorgeschriebene Geschwindigkeiten
          1         0.  3.369E-6
*
VISK      1    1   -1   -1                ! Viskositaet
          1  1.793E-3
*
BULK      1    1   -1   -1                ! Penalty-Parameter
          1     1.0E6
*
DENS      1    1   -1   -1                ! Dichte
          1     1.0E3
*
ALFA      1    1   -1   -1                ! Temperaturausdehnung
          1    4.6E-4
*
GEDA      1    1   -1   -1                ! Elementdicke
          1         1.
*
BQIN      1    1   -1   -1    1           ! Schwerebeschleunigung
          1         0.
*
FIN
```

Anhang E SMART Steuerprogramm

Beispiel für ein SMART Steuerprogramm (SPC) [LS91]

```
C
      LOGICAL GENZ
      COMMON /  / U(98),NETLIS(50),IPAGES(1)
      REAL TIME
      DATA ZETA,UPWD,DELTAT / 1.0 , .5164 , 2.0 /
      DATA NZEIT /  100 /
      DATA ICOUP,ICRIT / 1 , 2 /
      DATA IT0MAX / 2 /
C
      REWIND 1
      REWIND 2
C**** TOPOLOGIE UND DATENEINLESEN
      CALL FSTART(1,-1)
      CALL SET(4HDIAS,2)
      CALL SET(4HPRNT,-1)
C**** NETZ-SCHLEIFE:  NAVIER-STOKES-NETZ - NET=1
C                     TEMPERATUR-NETZ    - NET=2
      NET = 2
      DO 99 I = 1,NET
       CALL INTOP(I)
       CALL FIXCON
       CALL BELDA
       CALL BSB
       CALL BSA
       CALL INDAT(I)
       IF(I.EQ.2) CALL SAMBUK(4HNPCO,1)
       CALL ELCO
   99 CONTINUE
C---------------------------------------------------------------------
C         NAVIER-STOKES-GLEICHUNG
C---------------------------------------------------------------------
C**** NAVIER-STOKES-NETZ
      CALL USENET(1)
      CALL TN
C**** ANFANGSBEDINGUNG FUER GESCHWINDIGKEIT
      CALL INIV
C**** GRADIENTENMATRIX LINKHANDSEITE
      CALL LNAVLI(DELTAT,ZETA,ICOUP)
C---------------------------------------------------------------------
C         WAERMEBILANZ-GLEICHUNG
C---------------------------------------------------------------------
C**** TEMPERATUR-NETZ
      CALL USENET(2)
      CALL INIC
C
C-------------------------------------------------------------
C            * * *  BEGINN ZEITSCHLEIFE  * * *
C-------------------------------------------------------------
      TIME=0.
      DO 500 KTIME = 1,NZEIT
        TIME = TIME + DELTAT
```

```
C
C-------------------------------------------------------------
C           * * *  AUESSERE ITERATIONSSCHLEIFE  * * *
C-------------------------------------------------------------
C
       DO 100 IT0 = 1,IT0MAX
C
C****     DIFFUSIONS-NETZ
         CALL USENET(2)
C****     ITERATIONS-PARAMETER (WAERMEBILANZ-GLEICHUNG)
         EPS  = .005
         MINIT = 1
         MAXIT = 5
         GENZ  = .FALSE.
C****     GESCHWINDIGKEITEN VON NAVIER-STOKES-NETZ
         CALL REFBUK (4HVOLD)
         CALL REFBUK (4HVNEW)
         CALL SAMBUK (4HVOLD,1)
         CALL SAMBUK (4HVNEW,1)
C***     LINKHANDSEITE
         CALL LCONLI (DELTAT,ZETA,UPWD)
C****     ITERATIONS-SCHLEIFE (WAERMEBILANZ-GLEICHUNG)
  200    CONTINUE
C***     RECHTHANDSEITE
         CALL RECONV (DELTAT,ZETA,UPWD)
C***     LOESUNG
         CALL SOLVNL (KTIME,TIME,EPS,MINIT,MAXIT,GENZ)
         IF(.NOT.GENZ) GOTO 200
C***     KNOTENTEMPERATUREN
         IF (MOD(KTIME,5).EQ.0 .AND. IT0.EQ.IT0MAX) THEN
            CALL TEMP
            CALL ALTLAB (4HTEMP,4HUSR )
            NETLIS(22)=0
            CALL DXUSR (20)
            CALL REFBUK(4HUSR )
         END IF
C***     ANFANSBEDINGUNGEN NAECHSTER ZEITSCHRITT
         CALL ALTLAB(4HSRL ,4HSRL0)
C
C-------------------------------------------------------------------
C                    NAVIER-STOKES-GLEICHUNG
C-------------------------------------------------------------------
C
C****     NAVIER-STOKES-NETZ
         CALL USENET(1)
C****     ITERATIONS-PARAMETER (NAVIER/STOKES-GLEICHUNG)
         EPS  = .005
         MINIT = 1
         MAXIT = 2
         GENZ  = .FALSE.
C****     TEMPERATUREN VON TEMPERATUR-NETZ
         CALL REFBUK (4HTOLD)
         CALL REFBUK (4HTNEW)
         CALL SAMBUK (4HTOLD,2)
         CALL SAMBUK (4HTNEW,2)
C****     ITERATIONSSCHLEIFE (NAVIER/STOKES-GLEICHUNG)
```

```
  300     CONTINUE
C****     LASTVEKTOR RECHTHANDSEITE
          CALL RNAVLI(DELTAT,ZETA,UPWD,ICOUP)
C****     GLEICHUNGSLOESUNG
          CALL SOLVNA(KTIME,TIME,EPS,MINIT,MAXIT,ICRIT,GENZ)
          IF(GENZ) GOTO 350
          GOTO 300
  350     CONTINUE
C
          IF (MOD(KTIME,5).EQ.0 .AND. IT0.EQ.IT0MAX) THEN
            CALL REFBUK(4HUSR )
            CALL USR
            NETLIS(22)=0
            CALL DXUSR (10)
          END IF
C
          CALL ALTLAB(4HSRL ,4HSRL0)
C
  100   CONTINUE
C-------------------------------------------------------------
C            * * *  ENDE AUESSERE ITERATIONSSCHLEIFE  * * *
C-------------------------------------------------------------
C
  500 CONTINUE
C-------------------------------------------------------------
C        * * *  ENDE ZEITSCHLEIFE  * * *
C-------------------------------------------------------------
      CALL REFBUK(4HUSR )
C
      CALL BREAK
      END
```

Literatur

[ADPW84] J.H. Argyris, J. St. Doltsinis, P.M. Pimenta, und H. Wüstenberg. Natural Finite Element Techniques for Viscous Fluid Motion. *Computer Methods in Applied Mechanics and Engineering*, (45):3–55, 1984.

[AFR+73] J.H. Argyris, G. Faust, J.R. Roy, J. Szimmat, E.P. Warnke, und K.J. Willam. Finite Elemente zur Berechnung von Spannbeton-Reaktordruckbehältern, Institut für Statik und Dynamik der Luft- und Raumfahrtkonstruktionen, Universität Stuttgart, 1973.

[ALS92] J. Argyris, A. Laxander, und J. Szimmat. Petrov-Galerkin Finite Element Approach to Coupled Heat and Fluid Flow. *Computer Methods in Applied Mechanics and Engineering*, (94):181–200, 1992.

[AM88] J. Argyris und H.P. Mlejnek. *Die Methode der Finiten Elemente, Band I, II, III.* Friedrich Vieweg & Sohn, Braunschweig/Wiesbaden, 1986, 87, 88.

[Amd67] G. Amdahl. Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities. *AFIPS Conference Proceedings 30*, 1967.

[AÖS90] C. Aykanat, F. Özgüner, und D.S. Scott. Vectorization and Parallelization of the Conjugate Gradient Algorithm on Hypercube-connected Vector Processors. *Microprocessing and Microprogramming*, 29:67–82, 1990.

[AWW77] J.H. Argyris, E.P. Warnke, und K.J. Willam. Berechnungen von Temperatur- und Feuchtefeldern in Massivbauten nach der Methode der Finiten Elemente, Institut für Statik und Dynamik der Luft- und Raumfahrtkonstruktionen, Universität Stuttgart, 1977.

[Bab88] R. G. Babb. *Programming Parallel Processors.* Addison-Wesley Publishing Company, Inc., 1988.

[Bas93] A. Basermann. Datenverteilungs- und Kommunikationsmodelle für parallele CG-Verfahren zur Lösung von Gleichungssystemen mit dünnbesetzter Koeffizientenmatrix aus FE —Anwendungen, Aachener Informatik-Berichte, 1993.

[Bas95] A. Basermann. *Iterative Verfahren für dünnbesetzte Matrizen zur Lösung technischer Probleme auf massiv-parallelen Systemen.* Dissertation, 1995. Berichte des Forschungszentrums Jülich, Jül-3015.

[BBD95] R. Berrendorf, H. Burg, und U. Detert. Leistungscharakteristika von Parallelrechnern: Fallstudie Intel Paragon. 1995. Interner Bericht: KFA-ZAM-IB-9501, erscheint in: it+ti 2/95, Leistung von Parallelrechnern, R. Oldenbourg Verlag.

[BDF94] R. Biswas, K. Devine, und J.E. Flatherty. Parallel, Adaptive Finite Element Methods for Conservation Laws. *Applied Numerical Mathematics*, 14:255–283, 1994.

[BE89] I. Babuska und H.C. Elman. Some Aspects of Parallel Implementation of the Finite-Element Method on Message Passing Architectures. *Journal of Computational and Applied Mathematics 27*, 1989.

[Bem92] T. Bemmerl. Paragon XP/S – The Road to TeraFLOPS. In H.-W. Meuer, Ed, *Supercomputer '92*, pp 82–86. Springer-Verlag, 1992.

[BH82] A.N. Brooks und T.J.R. Hughes. Streamline Upwind/Petrov-Galerkin Formulations for Convection Dominated Flows with Particular Emphasis on the Incompresssible Navier-Stokes Equations. *Computer Methods in Applied Mechanics and Engineering*, (32):199–259, 1982.

[BJK+93] M. Behr, A. Johnson, J. Kennedy, S. Mittal, und T. Tezduyar. Computation of Incompressible Flows with Implicit Finite Element Implementations on the Connection Machine. *Computer Methods in Applied Mechanics and Engineering*, 108:99–118, 1993.

[BMF93] BMFT. BMFT-Initiative zur Förderung des parallelen Höchstleistungsrechnens in Wissenschaft und Wirtschaft, 1993.

[BS94] H.D. Baehr und K. Stephan. *Wärme- und Stoffübertragung*. Springer-Verlag, 1994.

[Bus93] D.M. Bushnell. Impacts of Massive Parallelism upon Forced Convection Heat Tranfer Computations. In W. Nakayama und K. Yang, Eds, *Computers and Computing in heat transfer science and engineering*. CRC Press, 1993.

[Chi92] P. Chin. Preconditioned Conjugate Gradient Methods for the Incompressible Navier-Stokes Equations. *International Journal for Numerical Methods in Fluids*, 15:273–295, 1992.

[CLMT94] A. Colbrook, M. Lemke, H. Mierendorff, und C.-A. Thole. EUROPORT-ESPRIT European Porting Projects. In W. Gentzsch und U. Harms, Eds, *High-Performance Computing and Networking*, Volume 796 of *Lecture Notes in Computer Science*, pp 46–54, 1994.

[CRA94] CRAY. Cray Research Konzepte für massiv-parallele Prozessorsysteme, Cray Research GmbH, Riesstraße 25, 80992 München, 1994.

[Cro94] L.G.C. Crone. The Preconditioned Conjugate Gradient Method on Distributed Memory Systems. In *High Performance Computing and Networking*. Springer-Verlag, 1994.

[DBP94] V. Donaldson, F. Berman, und R. Paturi. Program Speed-up in a Heterogeneous Computing Network. *Journal of Parallel and Distributed Computing*, 21:316–322, 1994.

[DH91] P. Deuflhard und A. Hohmann. *Numerische Mathematik*. Walter de Gryter, Berlin, New York, 1991.

[DHRF94] L.C. Dutto, W.G. Habashi, M.P. Robichaud, und M. Fortin. A Method for Finite Element Parallel Viscous Compressible Flow Calculations. *International Journal for Numerical Methods in Fluids*, 19:275–294, 1994.

[DLY89] P. Deuflhard, P. Leinen, und H. Yserentant. Concepts of an Adaptive Hierachical Finite Element Code. *Impact of Computing in Science and Engineering*, 1:3–35, 1989.

[DPR94] M. Dion, J.-L. Philippe, und Y. Robert. Parallelizing Compilers: What Can Be Achieved? Volume II of *Lecture Notes in Computer Science*, pp 447–456. Springer-Verlag, 1994.

[EAK+94] A. Ecer, H.U. Akay, W.B. Kemle, H. Wang, D. Erciksun, und E.J. Hall. Parallel Computation of Fluid Dynamics Problems. *Computer Methods in Applied Mechanics and Engineering*, 112:91–108, 1994.

[EEC91] EEC. Report of the EEC Working Group on High Performance Computing, Commission of the European Communities, 1991.

[EG92] EG. The Ei^3 Report on the Future of High Performance Computing in Europe, 1992.

[EK93] R. Esser und R. Knecht. Intel Paragon XP/S — Architecture and Sofware Environment. In H.-W. Meuer, Ed, *Supercomputer '93*, pp 121–141. Springer-Verlag, 1993.

[Far88] C. Farhat. A Simple and Efficient Automatic FEM Domain Decomposer. *Computers & Structures*, 28(5):579–602, 1988.

[Far90] C. Farhat. Which Parallel Finite Element Algorithm for which Architecture and which Problem? *Engineering Computation*, 7:187–195, 1990.

[FLS95] C. Farhat, S. Lanteri, und H.D. Simon. TOP/DOMDEC — A Software Tool for Mesh Partitioning and Parallel Processing. *erscheint in: Journal of Computing Systems in Engineering*, 1995.

[Fly66] M.J. Flynn. Very High-Speed Computing Systems. *Prodeedings of the IEEE*, (54):1901–1909, 1966.

[For94] High Performance Fortran Forum. *High Performance Fortran — Language Specification*. Rice University, 1994.

[FS93] R. F. Freund und H. J. Siegel. Heterogeneous Processing. *Computer*, 26(6):13–17, 1993.

[Ful86] R. E. Fulton. The Finite Element Machine: An Assesment of the Impact of Parallel Computing on Future Finite Element Computations. *Finite Element in Analysis and Design*, pp 83–98, 1986.

[GB95] M. Gerndt und R. Berrendorf. Parallelizing Applications with SVM-Fortran, 1995. Interner Bericht, KFA-ZAM-IB-9505, erscheint in: Proceedings der HPCN Europe 1995, Lecture Notes in Computer Science, Springer-Verlag.

[GBD+93] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, und V. Sunderam. PVM 3 User's Guide and Reference Manual, Oak Ridge National Laboratory, Oak Ridge, Tennessee 37831, 1993.

[GG93] G. Giacomazzi und J. Gretz. Euro-Quebec Hydro-Hydorgen Project (EQHHPP): a Challenge to Cryogenic Technolgogy. *Cryogenics*, 33(8):767–771, 1993.

[GH92] K. Gersten und H. Herwig. *Strömungsmechanik*. Grundlagen und Fortschritte der Ingenieurwissenschaften. Vieweg, 1992.

[GHL85] A. George, M.T. Heath, und J. Liu. Parallel Cholesky Factorization on a Multiprocessor, Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, 1985.

[GKF89] D. Goehlich, L. Komzsik, und R.E. Fulton. Application of a Parallel Equation Solver To Static FEM Problems. *Computers & Structures*, 31(2):121–129, 1989.

[GO93] G. Goloub und J.M. Ortega. *Scientific Computing*. Academic Press, Inc., 1993.

[Gri87] D. Gries. *The Science of Programming*. Texts and Monographs in Computer Science. Springer-Verlag, 1987.

[Gus88] J.L. Gustafson. Reevaluating Amdahl's Law. *Communications of the ACM*, 31(5):532–533, 1988.

[GWWL94] A.S. Grimshaw, J.B. Weissman, E.A. West, und E.C. Loyot(jr). Metasystems: An Approach Combining Parallel Processing and Heterogeneous Distributed Computing Systems. *Journal of Parallel and Distributed Computing*, 21:257–270, 1994.

[Ham74] R.E. Hammer. ASKA — Data Retrieval System, Institut für Statik und Dynamik der Luft- und Raumfahrtkonstruktionen, Universität Stuttgart (ISD), 1974.

[Hem94] R. Hempel. The MPI Standard for Message Passing . Volume I, II of *Lecture Notes in Computer Science*, pp 247–252. Springer-Verlag, 1994.

[Her93] F. Hertweck. A Comparison of Some Current Parallel Computer Architectures. In H.-W. Meuer, Ed, *Supercomputer '93*, Informatik aktuell, pp 104–120. Springer-Verlag, 1993.

[HHS92] R. Hempel, H.-C. Hoppe, und A. Supalov. PARMACS 6.0 Library Interface Specification, Institut für Methodische Grundlage, Gesellschaft für Mathematik und Datenverarbeitung, Sankt Augustin, Germany, 1992.

[Hic93] E.F. Hicken. Thermohydraulik in Kernreaktoren. Vorlesungsunterlage, Universität Bochum, 1993.

[HK74] H.M. Hilber und M. König. Element-Processors in ASKA, Institut für Statik und Dynamik der Luft- und Raumfahrtkonstruktionen, Universität Stuttgart (ISD), 1974.

[HN95] F. Hossfeld und W.E. Nagel. Per Aspera ad Astra: On the Way to Parallel Processing. 1995. Erscheint in den Proceedings des Seminars: "Supercomputing '95" in Mannheim, 22–24. Juni 1995, K.G. Saur Verlag, München.

[Hos95] F. Hossfeld. On the Super-computational Background of the Research Centre Jülich. 1995. Interner Bericht, KFA-ZAM-IB-9502, erscheint in: Poceedings des HLRZ-Workshops: 'Supercomputers in Brain Research: From Tomography to Neural Networks'.

[HRN+94] W.G. Habashi, M. Ronichaud, V.-N. Nguyen, W.S. Ghaly, M. Fortin, und J.W.H. Liu. Large-scale Computational Fluid Dynamics by the Finite Element Method. *International Journal for Numerical Methods in Fluids*, 18:1083–1105, 1994.

[HS52] M.R. Hestenes und E. Stiefel. Methods of Conjugate Gradients for Solving Linear Systems. *Journal of Research of the National Bureau of Standards*, 49(6):409–436, 1952.

[Hwa93] K. Hwang. *Advanced Computer Architecture: Parallelism, Scalability, Programmability*. McGraw-Hill Series in Computer Science. 1993.

[Ing88] Verein Deutscher Ingenieure, Ed. *VDI-Wärmeatlas*. 1988.

[INS95] *EUROPORT insights — Industrial Application Codes on Parallel Computers Fluid Dynamics and Structural Mechanics*. 2. 95.

[INT87] INTES. PERMAS Theory Manual , INTES Ingenieurgesellschaft für technische Software mbH, 1987. INTES Publication No. 302.

[JH92] Z. Johan und T.J.R. Hughes. A Data Parallel Finite Element Method for Computational Fluid Dynamics on the Connection Machine System. *Computer Methods in Applied Mechanics and Engineering*, 99:113–134, 1992.

[JS92] D. Jungmann und H. Stange. *Einführung in die Rechnerarchitektur*. Hanser Studienbücher der Informatik, 1992.

[KM89] L. Kohn und N. Margulis. The i860 64–Bit Supercomputing Microprocessor. In *Proceedings Supercomputing '89*, 1989.

[KOS87] W. Kordulle, H. Oertel, und H. Sobieczky. Numerische Aerodynamik mit Großrechenanlagen. *DFVLR-Nachrichten*, (51), 1987.

[KPSW93] A.A. Khokhar, V.K. Prasanna, M.E. Shaaban, und C.-L. Wang. Heterogeneous Computing Challenges and Opportunities. *Computer*, 26(6):18–27, 1993.

[Kuc92] Kuck & Associates, 1906 Fox Drive, Chamaign, IL 61820; USA. *CLASSPACK, Basic Math Library; User's Guide*, 1992.

[Lan91] A. Lankhorst. *Laminar and Turbulent Convection in Cavities*. Dissertation, Technische Universität Delft, 1991.

[LS91] A. Laxander und J. Szimmat. SMART II,4 – Konvektion Benutzerhandbuch, ICA, Institut für Computer-Anwendungen, Universität Stuttgart, 1991.

[Mal88] J.G. Malone. Automated Mesh Decomposition and Concurrent Finite Element Analysis for Hypercube Multiprocessor Computers. *Computer Methods in Applied Mechanics and Engineering*, (70):27–58, 1988.

[Man94] H. Manz. Parallelisierung des FEM Programms PERMAS, INTES GmbH, 1994.

[McB94] O.A. McBryan. An Overview of Message Passing Environments. *Parallel Computing*, 20:417–444, 1994.

[Mer87] G.P. Merker. *Konvektive Wärmeübertragung*. Springer-Verlag, 1987.

[MOT86] S. McGrogan, R. Olson, und N. Toda. Parallelizing Large Existing Programs — Methodology and Experience. In *31^{th} IEEE Computer Society International Conference*, pp 458–66, 1986.

[MT94] S. Mittal und T.E. Tezduyar. Massively Parallel Finite Element Computation of Incompressible Flows Involving Fluid-Body Interactions. *Computer Methods in Applied Mechanics and Engineering*, 112:253–282, 1994.

[NA94] W.E. Nagel und A. Arnold. Performance Visualization of Parallel Programs: The PARvis Environment. *Proceedings 1994 Intel Supercomputer Users Group (ISUG) Conference*, pp 24–31, 1994. Technischer Bericht CCSF-47, Concurrent Supercomputing Consortium, Pasadena, California.

[NSF94] NSF. Grand Challenges 1993: High Performance Computing and Communications, Committee on Physical, Mathematical, and Engineering Sciences, U.S. Office of Science and Technology Policy, National Science Foundation, Washingston, DC, 1994.

[OV85] J.M. Ortega und R.G. Voigt. *Solution of Partial Differential Equations on Vector and Parallel Computers*. SIAM, Philadelphia, 1985.

[Pet92] J. Petersen. Introduction to Programming on Distributed Memory Multiprocessors. *Computer Physics Communications*, 73:72–92, 1992.

[Pie94] P. Pierce. The NX Message Passing Interface. *Parallel Computing*, 20:463–48–, 1994.

[POW90] L. Prandtl, K. Oswatitsch, und K. Wieghardt. *Führer durch die Strömungslehre*. Vieweg, 9 Auflage, 1990.

[PVMZ87] J. Peraire, M. Vahdati, K. Morgan, und O.C. Zienkiewicz. Adaptive Remeshing for Compressible Flow Computations. *Journal computational Physics*, 72:449–463, 1987.

[Sch71] E. Schrem. ASKA — Subroutines and Common-Blocks in the Root-System, Institut für Statik und Dynamik der Luft- und Raumfahrtkonstruktionen, Universität Stuttgart (ISD), 1971.

[Sch88] A. Schaller. Finite Element Analysis of Microelectronic Components — State of the Art. In *Fourth IEEE/CHMT european international electronic manufacturing technology symposium proceedings '88*, pp 57–61, 1988.

[SDD94] H.D. Simon, W.R. Van Dalsem, und L. Dagum. Parallel Computational Fluid Dynamics: Current Status and Future Requirements, NASA Ames Research Center, 1994.

[SH93] W. Schönauer und Häfner. Explaining the Gap Between Theoretical Peak Performance and Real Performance for Supercomputer Architectures. In H. Küsters und E. Stein, Eds, *Proceedings of the Joint International Conference on Mathematical Methods and Supercomputing in Nuclear Applications*, Volume 2, pp 75–88. Kernforschungszentrum Karlsruhe, 1993.

[Stp92] P. Stpiczyński. Parallel Cholesky Factorization on Orthogonal Multiprocessors. *Parallel Computing*, 18:213–219, 1992.

[Str71] P. Streiner. ASKA — Internal Data Structure, Institut für Statik und Dynamik der Luft- und Raumfahrtkonstruktionen, Universität Stuttgart (ISD), 1971.

[Szi89] J. Szimmat. SMART I,4. Inkompressible Strömumg mit Reibung (Navier-Stokes-Gleichung), Institut für Computer-Anwendungen, Universität Stuttgart, 1989.

[TAB+93] T. Tezduyar, S. Aliabadi, M. Behr, A. Johnsin, und S. Mittal. Parallel Finite-Element Computations of 3D Flows. *Computer*, 1993.

[TBC+94] A.M. Tentner, R.N. Blomquist, T.R. Canfield, P.L. Garner, E.M. Gelbard, K.C. Gross, M. Minkoff, und R.A. Valentin. Advances in Parallel Computing for Reactor Analysis and Safety. *Communications of the ACM*, 37(4):55–64, 1994.

[Tri88] D.J. Tritton. *Physical Fluid Dynamics*. Oxford Science Publications, 1988.

[Ung88] J. Unger. *Konvektionsströmungen*. Teubner Studienbücher, 1988.

[Uni94] University of Tennessee, Knoxville, Tennessee. *MPI: A Message-Passing Interface Standard*, 1994.

[WA88] D.W. White und J.F. Abel. Bibliography on Finite Elements and Supercomputing. *Communications in Applied Numerical Methods*, 4:279–294, 1988.

[WA94] A. Watermann und J. Altes. Experiences Concerning the Parallelization of the Finite Element Code SMART. In W. Gentzsch und U. Harms, Eds, *High Performance Computing and Networking*, pp 92–93, 1994.

[Wat95] A. Watermann. Berechnungen von Konvektionsströmungen nach der Methode der Finiten Elemente auf dem massiv parallelen Supercomputer Intel/Paragon. *ZAMM Zeitschrift für Angewandte Mathematik und Mechanik*, 75:511–512, 1995.

[WC89] K.H. Law W.T. Carter, T.-L. Sham. A Parallel Finite Element Method and its Prototype Implementation on a Hypercube. *Computers & Structures*, 31:921–934, 1989.

[Wil94] R. Williams. Parallel Load Balancing for Parallel Applications. Concurrent Supercomputing Facilities, California Institut of Technology, Pasadena, California, 1994.

[Wüs86] H. Wüstenberg. *Effektive numerische Verfahren zur Behandlung inkompressibler viskoser Strömungsvorgänge.* Dissertation, Institut für Computer-Anwendungen, Universität Stuttgart, 1986.

[YYH93] A. Yoshida, G. Yagawa, und S. Hamada. Parallel Finite Elements with Domain Decomposition and its Pre-Processing. *SMIRT 12*, pp 195–200, 1993.

[Zab67] M.G. Zabetakis. Safety with Cryogenic Fluids. *The International Cryogenics Monograph Series, Heywood Books London*, 1967.

[ZB95] G. Zavagli und H. Burg. Untersuchung des parallelen Gleichungslösers Paragon ProSolver-DES —slab solver — auf dem Rechner Intel/Paragon XP/S-10. März 1995. Interner Bericht: KFA-ZAM-IB-9506.

[Zie77] O.C. Zienkiewicz. *The Finite Element Method.* Mc Graw-Hill Book Company (UK) Limited, 1977.

[ZO82] J. Zierep und H. Oertel(jr.), Eds. *Convective Transport and Instability Phenomena.* G. Braun Karlsruhe, 1982.

Dank

Die vorliegende Arbeit entstand am Institut für Sicherheitsforschung und Reaktortechnik (ISR) des Forschungszentrums Jülich.

Herrn Prof. Dr.-Ing. E.F. Hicken, Direktor am Institut für Sicherheitsforschung und Reaktortechnik des Forschungszentrums Jülich und Inhaber der gleichnamigen Professur an der Ruhr-Universität Bochum, danke ich für die intensive Betreuung und das große Interesse an dieser Doktorarbeit.

Herrn Prof. Dr. F. Hoßfeld, Direktor des Zentralinstituts für Angewandte Mathematik (ZAM) des Forschungszentrums Jülich und Inhaber des Lehrstuhls für Technische Informatik und Computerwissenschaften an der RWTH Aachen danke ich herzlich für die Übernahme des Korreferats.

Herrn Dr.-Ing. J. Altes danke ich besonders für die fachliche Betreuung, die konstruktiven Diskussionen und auch für die Möglichkeit, erste Ergebnisse auf verschiedenen Veranstaltungen vorzutragen. Herrn D. Koschmieder danke ich für die technische Unterstützung bei der Anfertigung der Bilder mit dem von ihm entwickelten Graphikprogramm RAPS sowie für die Diskussionen programmiertechnischer Fragestellungen.

Ferner gilt mein Dank dem Intel-Team des Zentralinstituts für Angewandte Mathematik für die Unterstützung bei der Nutzung des Parallelrechners Intel/Paragon, stellvertretend möchte ich Frau J. Docter und Frau K. Faelski aus dem ZAM sowie Herrn H. Bast von der Firma Intel nennen.

Weiterhin möchte ich allen Mitarbeiterinnen und Mitarbeitern des Instituts für Sicherheitsforschung und Reaktortechnik danken, die durch ihre Kollegialität zum Gelingen dieser Arbeit beigetragen haben.

Last but not least danke ich meinen Eltern!

FORSCHUNGSZENTRUM JÜLICH GmbH

KFA

Jül-3122
September 1995
ISSN 0944-2952