



FORSCHUNGSZENTRUM JÜLICH GmbH

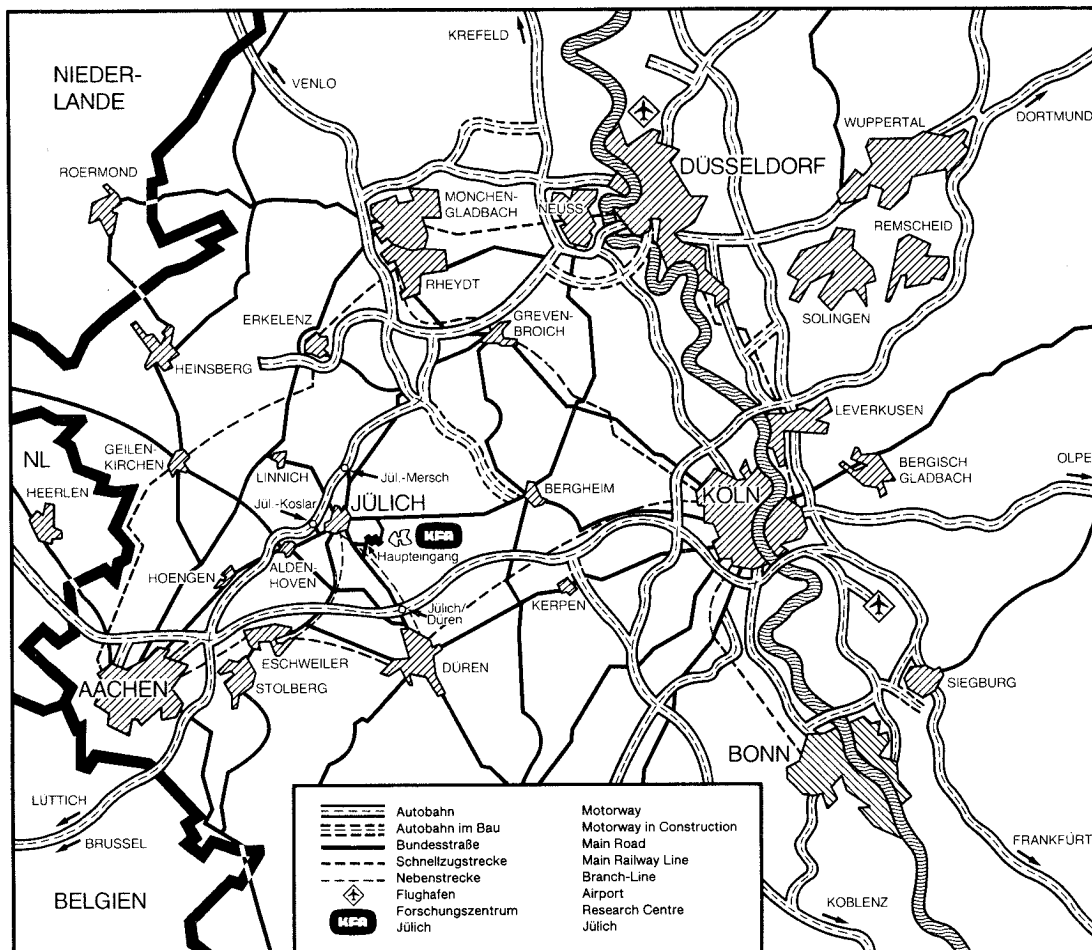
Zentralinstitut für Angewandte Mathematik

**Eine Programmierungsumgebung zur
Messung der wechselseitigen Einflüsse
von Hintergrundlast und
parallelem Programm**

von

Markus A. Linn

Jül-2416
Dezember 1990
ISSN 0366-0885



Als Manuskript gedruckt

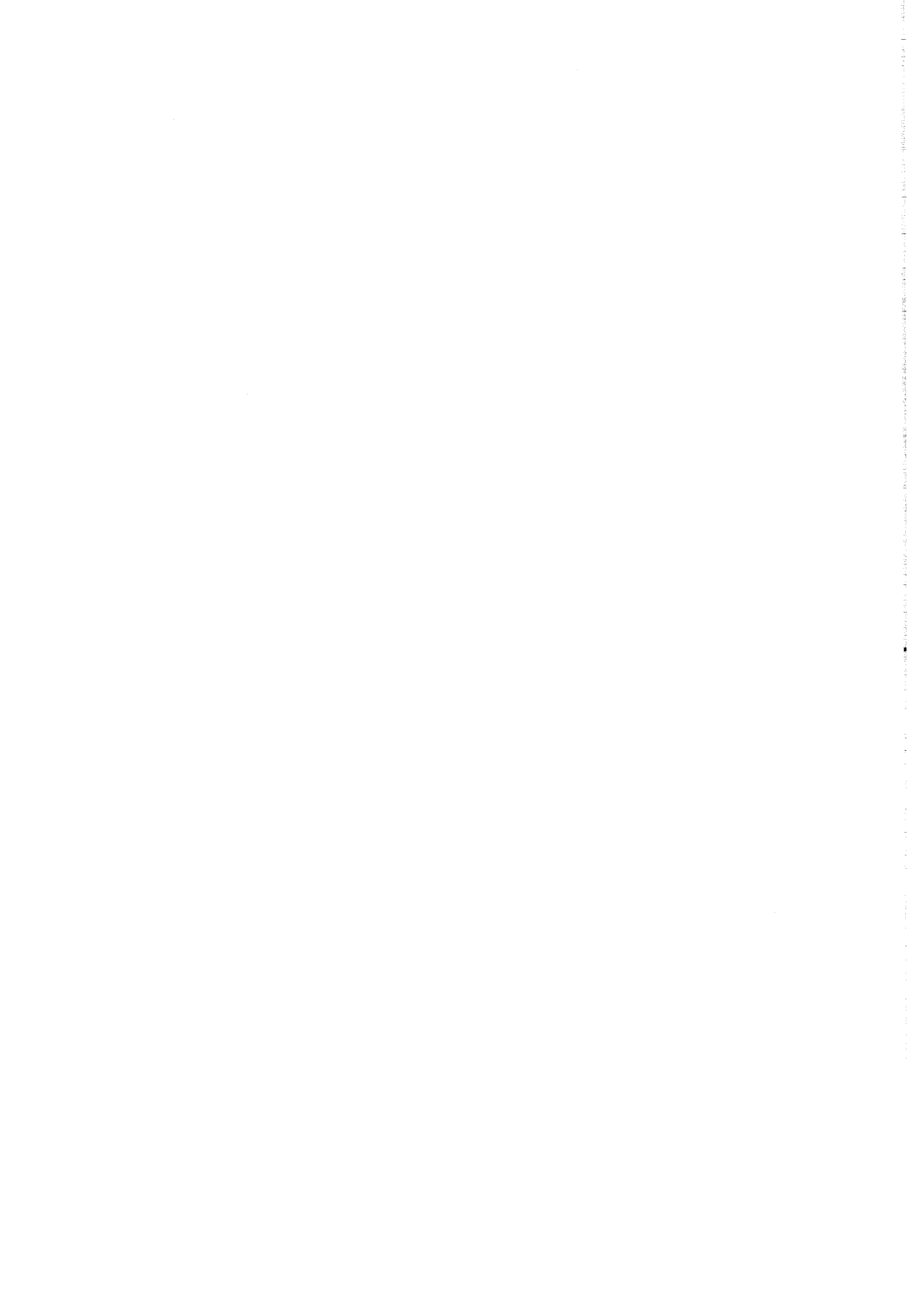
Forschungszentrum Jülich: Berichte Nr. 2416
 Zentralinstitut für Angewandte Mathematik Jül-2416

Zu beziehen durch: ZENTRALBIBLIOTHEK · Forschungszentrum Jülich GmbH
 Postfach 19 13 · D-5170 Jülich (Bundesrepublik Deutschland)
 Telefon: 02461/61-0 · Telex: 833 556-70 kfa d · Telefax: 02461/61-61 03

Eine Programmierumgebung zur Messung der wechselseitigen Einflüsse von Hintergrundlast und parallelem Programm

von

Markus A. Linn



Zusammenfassung

Die meisten Mehrprozessor-Supercomputer werden heute im Mehrbenutzerbetrieb eingesetzt, wo die einzelnen Prozessoren unabhängige Jobs ausführen, die um die vorhandenen Ressourcen konkurrieren. Mit Hilfe von Multitasking kann dabei ein Programm auch parallel mit mehreren Prozessoren ausgeführt werden. Die Effizienz von parallelen Programmen im Mehrbenutzerbetrieb und der Einfluß dieser Programme auf die gesamte Systemlast sind wesentliche Bewertungskriterien für den Einsatz von Multitasking.

Während die Effizienz von parallelen Programmen in dedizierter Umgebung bereits vielfach untersucht wurde, wird in dieser Arbeit ein neuer Ansatz zur Bewertung von Multitasking gemacht. Die Arbeit beschreibt das Benchmark-System PAR-Bench, das auf den Systemen Cray X-MP und Cray Y-MP unter den Betriebssystemen COS und UNICOS implementiert ist und Effizienzuntersuchungen von parallelen Programmen im Mehrbenutzerbetrieb ermöglicht. In einem ersten Schritt werden mit Hilfe von PAR-Bench synthetische Benchmark-Programme aufgrund von Parametervorgaben generiert. Bei der Generierung werden die Benchmark-Programme jedoch nur bezüglich der zu verrichtenden Arbeit festgelegt. Im zweiten Schritt können die Benchmark-Programme als Arbeitslast ausgeführt werden, wobei die vorher festgelegte Arbeit in verschiedenen Benchmark-Läufen auf unterschiedliche Arten erbracht werden kann. Die Art der Ausführung wird wiederum durch Parametervorgaben spezifiziert, wobei insbesondere auch der Parallelitätsgrad der Programme verändert werden kann.

Mit Hilfe von PAR-Bench können die wechselseitigen Einflüsse von parallelen Programmen und Hintergrundlast ermittelt werden, was die Bewertung verschiedener Multitasking-Implementationen, verschiedener Betriebssysteme und unterschiedlicher Hardware erlaubt. Aufgrund der großen Menge der für diese Bewertung relevanten Daten wurde zur Unterstützung von PAR-Bench das Benchmark-Analysesystem GRANSYS entwickelt, das eine automatische Auswertung der Benchmarks mit Hilfe von Graphiken und Tabellen unterstützt.

Inhaltsverzeichnis

1. Einleitung	1
2. Vektor-Supercomputer als Parallelrechner	5
2.1. Die Cray Y-MP - ein shared-memory-Rechner	6
2.2. Parallelverarbeitung	6
2.3. Die verwendeten Meßinstrumente	10
3. Leistungsbewertung durch Benchmarking	11
3.1. Methoden zur Leistungsbewertung	11
3.1.1. Meßumgebungen	12
3.1.2. Meßgrößen	14
3.2. Ein Klassifikationsansatz für Benchmark-Programme	15
3.3. Supercomputer-Benchmarks	17
3.3.1. Einbenutzerbetrieb	17
3.3.1.1. Ein universeller Supercomputer-Benchmark	17
3.3.1.2. Der Synthetic Standard Benchmark	18
3.3.1.3. Ein Benchmark vom PERFECT Club	19
3.3.1.4. PARC-Bench - ein systemorientierter Ansatz	20
3.3.2. Ein Benchmark im Mehrbenutzerbetrieb	21
3.3.3. Mehrbenutzerbetrieb mit parallelen Programmen	23
3.3.3.1. Der Einfluß des Multiprogramming	23
3.3.3.2. Ein parametergesteuerter Benchmark	25
4. Das Benchmark-System PAR-Bench	29
4.1. Das Funktionsprinzip von PAR-Bench	29
4.1.1. Der Generate-Modus	30
4.1.1.1. Die statischen Parameter	31
4.1.1.2. Die dynamischen Parameter	31
4.1.2. Der Execute-Modus	33
4.1.3. Die Simulation einer vollständigen Arbeitslast	34
4.1.4. Der Newdata-Modus	35
4.2. Die Realisierung von PAR-Bench	35
4.2.1. Die Erzeugung eines Benchmark-Programms	36
4.2.1.1. Der Parameterraum einer Cray Y-MP8/832	36
4.2.1.2. Die Programmkerne und das Parametergebiet	37
4.2.2. Die Parallelisierung eines Benchmark-Programms	42
4.2.3. Die Ermittlung der Kerndaten	49
4.3. Die Steuerung des Systems PAR-Bench	50
4.4. Portabilität	55
4.5. Die Schnittstelle zum FORTRAN 77 Standard	56

5. Das Benchmark-Analysesystem GRANSYS	59
5.1. Die Auswertung von PAR-Bench	59
5.1.1. Die Analysen der PAR-Bench-Informationen	61
5.1.1.1. Der Benchmark-Plot	61
5.1.1.2. Die PAR-Bench-Tabellen	64
5.2. Die Auswertung der SAR-Informationen	65
5.2.1. Die Analysen der SAR-Informationen	67
 6. Erfahrungen mit der Programmierungsumgebung	 71
6.1. Die Speedup-Messungen	71
6.2. Parallele Programme und Hintergrundlast	76
6.2.1. Parallele Programme mit geringer Parallelität	77
6.2.2. Parallele Programme mit hoher Parallelität	80
6.2.3. Die Behebung von Speicherengpässen	84
6.2.4. Die Portierung von Benchmarks auf eine Cray X-MP	88
 7. Ausblick	 93
 Literatur	 95
 Anhang A. Die Daten aller Programmkerne	 99
 Anhang B. Die Ausgabedatei eines Benchmark-Programms	 101

1. Einleitung

Bei der Forschung und Entwicklung im naturwissenschaftlich-technischen Bereich treten vermehrt Probleme auf, deren Lösung den Einsatz numerischer Verfahren erfordert. Die Nutzung von Rechnern ist für realistische Aufgabenstellungen in diesem Bereich unumgänglich. Aufgrund der Komplexität der Zusammenhänge ist jedoch die benötigte Rechenzeit in vielen Fällen so groß, daß die Leistungsfähigkeit der heute verfügbaren Rechner der begrenzende Faktor in bezug auf die Realisierbarkeit ist; die Lösung vieler Probleme aus diesem Bereich erfordert auch mit Hilfe modernster Supercomputer zum Teil viele Jahre an Rechenzeit.

Die Leistungsfähigkeit von Supercomputern kann zum einen durch Verbesserungen im technologischen Bereich erhöht werden. Da zum Beispiel die Lichtgeschwindigkeit eine Obergrenze für die Signalausbreitung darstellt und bei der Platzierung von Bauteilen auf Platinen mit heutiger Technologie Integrationsgrenzen erreicht werden, stößt man permanent an physikalische Grenzen, und zur Zeit werden auch mit erheblichem Aufwand nur noch geringe Fortschritte erzielt.

Im Gegensatz dazu werden momentan bei der Entwicklung neuer Architekturen entscheidende Leistungssteigerungen verzeichnet. Seit einiger Zeit konzentrieren sich die Aktivitäten auf das Gebiet der Mehrprozessorrechner. Einerseits ermöglichen diese Rechner im Mehrbenutzerbetrieb aufgrund der mehrfach vorhandenen Prozessoren eine effizientere Nutzung der Systemressourcen und damit auch eine Verkürzung der Ausführungszeiten. Andererseits können Leistungssteigerungen gegenüber Einprozessorsystemen durch die parallele Abarbeitung von Programmteilen erzielt werden, d.h. ein Programm kann mehrere Prozessoren gleichzeitig nutzen. Bei der Entwicklung von Mehrprozessorrechnern werden drei grundsätzlich verschiedene Konzepte verfolgt:

- In *shared-memory*-Supercomputern arbeiten relativ wenige Hochleistungsprozessoren auf einem gemeinsamen Speicher. Da die zur Parallelverarbeitung einzelner Programme notwendige Kommunikation zwischen den Prozessoren über den gemeinsamen Speicher erfolgt, ist die Kommunikationsgeschwindigkeit vergleichsweise hoch. Deswegen ist diese Klasse von Rechnern für Parallelisierungsstrategien geeignet, die eine häufige Kommunikation der Prozessoren erfordern [HwBr86]. In dieser Klasse der *shared-memory*-Rechner sind heute Systeme verschiedener Hersteller kommerziell verfügbar, z.B. die Y-MP der Firma Cray Research und die 3090J der Firma IBM. Weitere Rechner dieser Klasse sind bereits angekündigt, z.B. die Mehrprozessorversionen der VP 2000 der Firma Fujitsu und der NEC SX-3 der Firma NEC [Com89].
- *Distributed-memory*-Rechner (Rechner mit verteiltem Speicher) bestehen aus einer sehr großen Anzahl von Prozessoren relativ geringer Leistung, die jeweils auf einem eigenen Speicher arbeiten. In diesen massivparallelen Systemen erfolgt die Kommunikation der Prozessoren über spezielle Verbindungsnetzwerke (Bus, Hypercube) durch den Austausch von Nachrichten. Da diese Art der Kommunikation relativ zeitaufwendig ist, arbeiten *distributed-memory*-Rechner nur dann effizient, wenn die Kommunikation zwischen den Prozessoren nur selten erforderlich ist. Zu der Klasse der *distributed-memory*-Rechner gehören z.B. der iPSC der Firma Intel, die

Connection Machine der Thinking Machines Corporation [AlGo89] und der SUPRENUM-Rechner, eine gemeinsame Entwicklung mehrerer Firmen und Forschungseinrichtungen [Gil88].

- Bei bestimmten, sehr rechenzeitintensiven Aufgabenstellungen kann es von Vorteil sein, spezielle Rechner zu entwickeln, die an die Anforderungen dieser bestimmten Problemklasse angepaßt sind. Dabei werden z.B. die entsprechenden Algorithmen direkt auf die Hardware abgebildet. Dadurch wird die Möglichkeit des universellen Einsatzes dieser Spezialrechner stark eingeschränkt. Als Beispiel für einen Spezialrechner mit mehreren Prozessoren kann der GF11 der Firma IBM genannt werden, der für Anwendungen aus dem Bereich der Quanten-Chromodynamik entwickelt worden ist [BDW85].

Die unterschiedliche Eignung der Rechner aus den verschiedenen Klassen für bestimmte Anwendungen, das wachsende Angebot und die steigenden Kosten dieser Systeme machen Leistungsvergleiche notwendig. Aufgrund der Komplexität von Supercomputern kann mit Hilfe von strukturellen und technischen Daten nur eine erste Einordnung vorgenommen werden. Für Bewertungen aufgrund von Leistungsmessungen ist die Durchführung von Benchmarks die beste Methode. Benchmarks bestehen aus einer Menge von Programmen, die auf den betreffenden Rechnern zu Meßzwecken ausgeführt werden. Ein entscheidender Faktor bei der Leistungsbewertung ist die Ausführungszeit der Benchmark-Programme.

Es existieren mehrere Benchmarks, die auf einer Vielzahl von Rechnern durchgeführt worden sind und die zu Vergleichen herangezogen werden können. Da die Ergebnisse jedoch stark von den jeweiligen Benchmark-Programmen abhängen, favorisieren die verschiedenen Benchmarks nicht immer den gleichen Rechner. Deswegen führen potentielle Anwender zusätzlich eigene Benchmarks durch, die auf eigenen Programmen basieren. Diese Programme haben zum Teil sehr große Ausführungszeiten. Um den für Vergleichsmessungen akzeptablen zeitlichen Rahmen nicht zu verlassen, werden die Programme nur in gekürzten Versionen ausgeführt, d.h. es wird z.B. die Problemgröße reduziert. Auswahl und Modifikation der Programme für einen Benchmark sind mit großem Aufwand verbunden, der jedoch entscheidend ist für die Aussagekraft der Ergebnisse.

Mit der steigenden Verfügbarkeit der Mehrprozessor-Supercomputer wächst auch die Zahl der Anwender, die die Ausführungszeiten ihrer Programme durch die Parallelverarbeitung innerhalb der Programme verkürzen wollen. Zur Zeit wird die Nutzung der Parallelverarbeitung von Rechenzentren fast ausschließlich im Einbenutzerbetrieb angeboten. Da im Mehrbenutzerbetrieb im allgemeinen eine effizientere Nutzung der Hardware gewährleistet ist, bei den existierenden Systemen in der Regel jedoch kein flexibler Wechsel zwischen Einbenutzer- und Mehrbenutzerbetrieb möglich ist, wird der Einbenutzerbetrieb nur in Ausnahmefällen ermöglicht. Deshalb gewinnt die Ausführung von parallelen Programmen im Mehrbenutzerbetrieb zunehmend an Bedeutung.

Neben der Beschleunigung einzelner Programme kann diese Strategie eine wesentliche Steigerung des Systemdurchsatzes bewirken. Dies gilt insbesondere für *shared-memory*-Rechner, die auf dem Konzept der Realspeicheradressierung basieren. Dort können durch Programme mit hohem Speicherbedarf Engpässe im Speicher auftreten, d.h. es

passen nur wenige Programme gleichzeitig in den Speicher, so daß eine effiziente Nutzung aller Prozessoren nur durch Parallelverarbeitung innerhalb von Programmen möglich ist. Die Parallelisierung von Programmen im Mehrbenutzerbetrieb kann also eine effektivere Ausführung der Programme bei einer effizienteren Nutzung der Systemressourcen bewirken. Die quantitative Untersuchung dieser beiden Effekte ist Gegenstand dieser Arbeit.

Im Bereich der installierten Supercomputersysteme sind heute die *shared-memory*-Rechner am häufigsten vertreten. Im Rahmen dieser Arbeit soll ausschließlich diese Rechnerkategorie betrachtet werden. Kapitel 2 beschreibt am Beispiel einer Cray Y-MP die Architektur eines solchen Rechners. Darüber hinaus werden die grundlegenden Strategien für den Betrieb eines solchen Rechners erläutert.

In Kapitel 3 wird ein Überblick über bisherige Aktivitäten im Bereich des Benchmarking gegeben. Dabei werden grundsätzliche Überlegungen angestellt, und die grundlegenden Begriffe und Größen werden erläutert. Die bisherigen Benchmark-Ansätze werden kategorisiert, und es werden mehrere Benchmarks im Detail beschrieben. Darüber hinaus wird die Rolle dieser Arbeit innerhalb der beschriebenen Benchmark-Aktivitäten dargelegt.

Kapitel 4 beschreibt das im Rahmen dieser Arbeit erstellte Benchmark-System *PAR-Bench* (*PARallel-Benchmark*). Dabei wird auf das Funktionsprinzip, auf die wichtigsten Realisierungen und auf die Steuerung des Systems eingegangen. Mit diesem System können synthetische Benchmarks erstellt und ausgeführt werden, die insbesondere die Untersuchung von speziellen Systemkomponenten wie zum Beispiel der Multitasking-Implementationen, die die Parallelisierung von Programmen unterstützen, oder des Job-Schedulers des Betriebssystems erlauben.

Das in Kapitel 5 beschriebene Benchmark-Analysesystem *GRANSYS* (*GRaphical ANalyzing SYStem*) ermöglicht eine automatische Auswertung der mit dem System *PAR-Bench* durchgeführten Benchmarks. Dabei werden die relevanten Daten sowohl graphisch als auch tabellarisch dargestellt. Es werden die Anwendung von *GRANSYS* sowie die Bedeutung der jeweils dargestellten Informationen erläutert.

Kapitel 6 stellt anhand von Beispielen erste Ergebnisse vor, die durch die Nutzung dieser Programmierungsumgebung gewonnen wurden. In einem Teil der beschriebenen Messungen wird die Funktionalität des Systems *PAR-Bench* getestet, alle weiteren durchgeführten Benchmarks untersuchen das Zeitverhalten bei der Ausführung von parallelen Programmen im Mehrbenutzerbetrieb. Dabei wird neben Untersuchungen auf einer Cray Y-MP8/832 auch ein Vergleich dieses Rechners mit einer Cray X-MP/416 durchgeführt.

2. Vektor-Supercomputer als Parallelrechner

Ein Vektorprozessor unterstützt die Ausführung von Vektoroperationen. Bei einer Vektoroperation wird eine Operation auf eine Menge von vielen Operanden (Vektor) angewendet. Durch das bei der Vektorverarbeitung implizit angewandte Prinzip des *pipelining* [HoJe88, HwBr86] wird gegenüber der Skalarverarbeitung eine erhebliche Steigerung der Verarbeitungsgeschwindigkeit erzielt. Die jeweilige Steigerung hängt von der Vektorlänge, d.h. der Anzahl der zu verarbeitenden Operandenpaare, ab.

Bei den heute verfügbaren Vektor-Supercomputern sind häufig mehrere Hochleistungsvektorprozessoren zu einer Mehrprozessorkonfiguration zusammengefaßt. Diese Rechner gehören in die Kategorie der *shared-memory*-Rechner, d.h. alle Prozessoren arbeiten auf einem gemeinsamen Speicher. Im Mehrbenutzerbetrieb wird der gemeinsame Speicher auf mehrere Programme aufgeteilt. Bei der parallelen Abarbeitung einzelner Programme verarbeiten mehrere Prozessoren Teile von denselben Datensätzen. Da sich diese Daten im gemeinsamen Speicher befinden, ist kein Transfer von Daten zwischen den Prozessoren notwendig. Es findet lediglich eine Kommunikation zwischen den Prozessoren statt, um die Synchronisation der Verarbeitung zu gewährleisten. Der bei der Parallelverarbeitung entstehende Overhead, d.h. die zusätzlichen Aktivitäten für Synchronisation und Kommunikation und der damit verbundene Zeitaufwand, ist gering im Vergleich zu notwendigen Datentransfers bei Rechnern mit verteiltem Speicher.

Ein wesentliches Konzept zur Verbesserung der Effektivität dieser Rechner ist die Speicherverschränkung (*interleaving*), d.h. der Speicher wird in mehrere unabhängige Speicherbänke aufgeteilt. Dadurch sind einerseits gleichzeitig mehrere Speicherzugriffe eines Prozessors möglich, sofern dabei unterschiedliche Bänke angesprochen werden. Andererseits können mehrere Prozessoren gleichzeitig auf den Speicher zugreifen. Wenn eine Speicherbank während eines noch nicht beendeten Zugriffs erneut angesprochen wird, tritt ein Speicherzugriffskonflikt (*memory contention*) auf, der durch ein Prioritätenverfahren aufgelöst wird.

Auf *shared-memory*-Rechnern sind zwei verschiedene Speicherkonzepte realisiert. Beim Konzept des Realspeichers (Cray Y-MP) muß sich ein auszuführendes Programm vollständig im Hauptspeicher befinden, da nur der tatsächlich vorhandene Speicherbereich adressiert werden kann. Beim Konzept des virtuellen Speichers (IBM 3090) können dagegen auch Speicheradressen genutzt werden, die außerhalb des Hauptspeicherbereichs liegen. Die Daten mit diesen Adressen befinden sich im Hintergrundspeicher und werden nur bei Bedarf in den Hauptspeicher geladen. Die auszuführenden Programme müssen sich also nur teilweise im Hauptspeicher befinden. Der Vorteil dieses Konzeptes liegt in der Größe des Adreßraumes. Ein entscheidender Nachteil in bezug auf die Effektivität ist jedoch die im Vergleich zum Konzept des Realspeichers geringere Verarbeitungsgeschwindigkeit, da die benötigten Daten gegebenenfalls zunächst aus dem Hintergrundspeicher geladen werden müssen, bevor sie verarbeitet werden können.

In diesem Kapitel werden am Beispiel einer Cray Y-MP die Architekturmerkmale eines *shared-memory*-Rechners beschrieben. Darüber hinaus werden die zur Parallelisierung von Programmen relevanten Begriffe erläutert. Am Ende des Kapitels erfolgt eine kurze Beschreibung der im Rahmen dieser Arbeit verwendeten Meßinstrumente.

2.1. Die Cray Y-MP - ein shared-memory-Rechner

Die Cray Y-MP ist ein *shared-memory*-Rechner mit realer Adressierung; die folgende Beschreibung bezieht sich auf die im Zentralinstitut für Angewandte Mathematik des Forschungszentrums Jülich installierte Cray Y-MP8/832. In dieser Konfiguration verfügt das System über 8 Prozessoren und einen Hauptspeicher von 32 Megaworten, der in 256 Speicherbänke aufgeteilt ist. Die Wortlänge beträgt 64 bit. Abb. 1 zeigt die Architektur dieses Rechners.

Jeder Prozessor verfügt zusätzlich zu den skalaren Funktionseinheiten und Registern über mehrere Vektorfunktionseinheiten und Vektorregister. Damit gehört die Cray zu den registerorientierten Maschinen, d.h. die Operanden und Ergebnisse werden bei der Vektorverarbeitung in Registern gehalten. Mit Hilfe der Vektorregister wird das *chaining* realisiert, eine Verkettung von Vektoroperationen, die unnötige Speicherzugriffe vermeidet und so eine höhere Verarbeitungsgeschwindigkeit ermöglicht [HoJe88].

Die 8 identischen Prozessoren sind über den gemeinsamen Speicher und über gemeinsame Register (*shared-registers*) miteinander verbunden. Über diese beiden Wege kommunizieren die Prozessoren miteinander. Der Zugang zum Rechner erfolgt über das Ein-/Ausgabesystem, das neben der direkten Verbindung zum Hauptspeicher optional den Erweiterungsspeicher (*SSD = Solid-state Storage Device*) als Puffer benutzen kann.

Die Cray Y-MP ist der Nachfolger der Cray X-MP, die eine ähnliche Architektur mit maximal 4 Prozessoren und 16 Megaworten Speicher aufweist. Beide Rechner können unter den Betriebssystemen COS und UNICOS betrieben werden. Diese erlauben sowohl die Ausführung von Programmen im Einbenutzerbetrieb als auch im Mehrbenutzerbetrieb. Im Einbenutzerbetrieb spricht man von einer dedizierten Umgebung, d.h. einem Benutzer stehen alle Systemressourcen zur Verfügung. Im Mehrbenutzerbetrieb werden die Programme mehrerer Benutzer in einer Multiprogramming-Umgebung ausgeführt (siehe Kapitel 2.2).

Das UNIX-nahe Betriebssystem UNICOS erlaubt im Gegensatz zu COS neben der Batch-Verarbeitung auch den interaktiven Zugang und hat inzwischen das Betriebssystem COS weitgehend abgelöst.

2.2. Parallelverarbeitung

Der Begriff Parallelverarbeitung bezeichnet die gleichzeitige bzw. überlappende Ausführung von Jobs, Programmen oder Teilen davon. Im folgenden werden die in diesem Zusammenhang benutzten Begriffe erläutert:

Multiprogramming

In einer Multiprogramming-Umgebung sind mehrere Programme zur Ausführung bereit. Wenn ein Programm seine Ausführung auf einem Prozessor unterbricht, wird dem Prozessor sofort ein anderes Programm zugeordnet. Dabei vergibt der Scheduler die Prozessoren jeweils nur für festgelegte Zeitscheiben (*time slices*), um die Ausführung von kurzen Programmen nicht übermäßig durch vorher gestartete lange Programme zu behindern. Auf diese Weise wird eine effiziente Nutzung der vorhandenen Prozessoren gewährleistet.

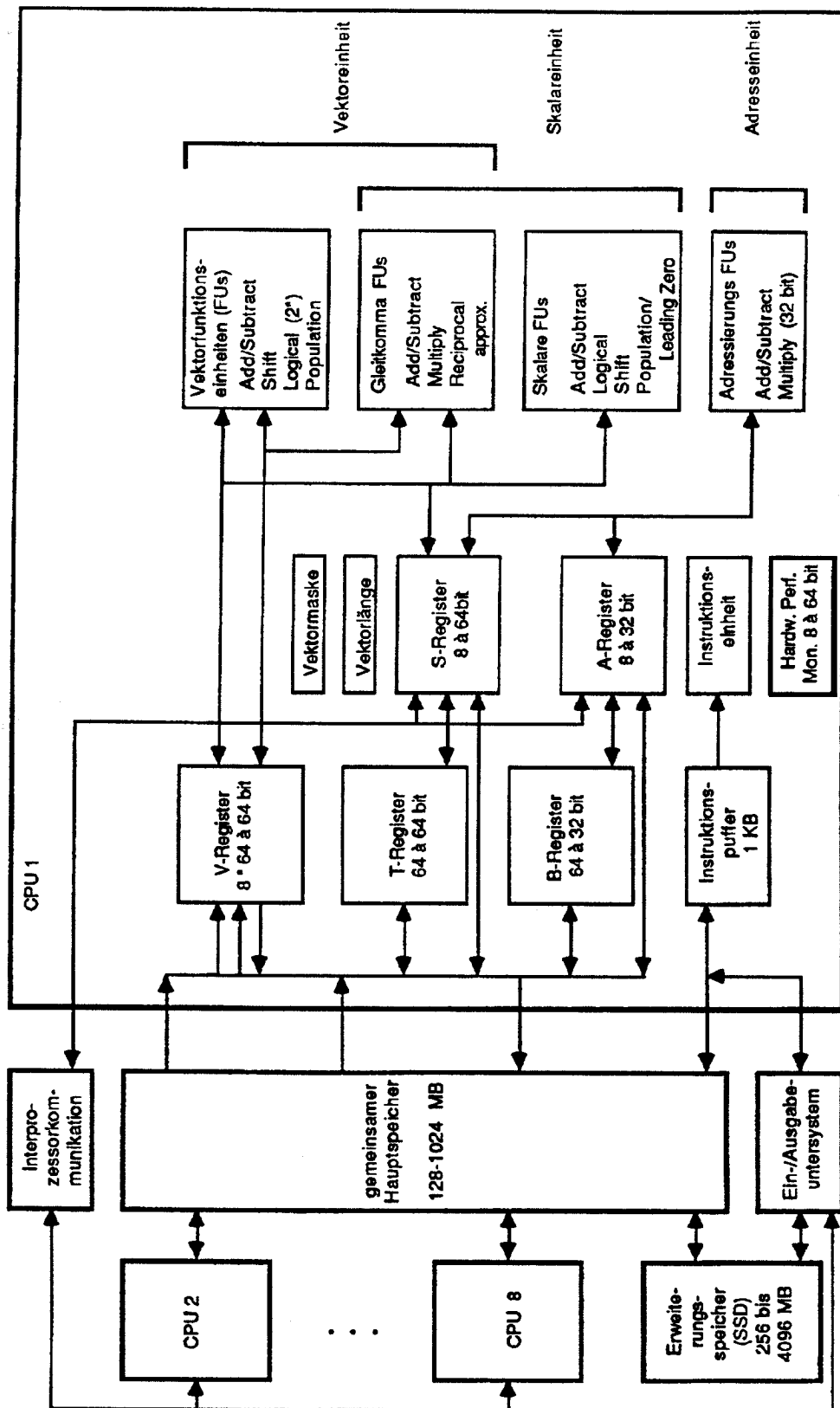


Abb. 1. Die Architektur der Cray Y-MP (siehe [BEK90])

Multiprocessing

Multiprocessing wird in einer Multiprogramming-Umgebung bei Systemen mit mehr als einem Prozessor realisiert. Dabei verarbeitet jeder Prozessor ein unabhängiges Programm, d.h. es findet eine Parallelverarbeitung auf Programmebene statt. Auf diese Weise wird die Ausführungszeit der Programme gegenüber Einprozessorsystemen reduziert, wenn jeweils dieselbe Arbeitslast zugrundegelegt wird.

Multitasking

Die Ausführung eines Programms mit mehr als einem Prozessor setzt voraus, daß das Programm in unabhängige Prozesse zerlegt werden kann, die parallel ausgeführt werden können. Beim Multitasking wird ein Programm von zwei oder mehr Prozessoren gleichzeitig ausgeführt, eine ausführliche Beschreibung der auf Cray-Rechnern verfügbaren Multitasking-Konzepte findet sich in [DeNe88, HKN89, Lie88, Nag90a, Reg89]. Voneinander unabhängige Prozesse werden verschiedenen Prozessoren zugeordnet, die diese ausführen. Im Rahmen dieser Arbeit werden die beiden Cray Multitasking-Implementationen Microtasking und Autotasking genutzt, bei denen die Parallelverarbeitung z.B. auf Schleifenebene erfolgt. Dabei werden die einzelnen Iterationen einer DO-Schleife als Prozesse definiert. Mehrere Prozessoren führen gleichzeitig jeweils eine Iteration (Prozeß) dieser Schleife aus. Wenn ein Prozessor die Ausführung einer Iteration beendet hat, wird eine weitere Iteration ausgeführt, die noch nicht bearbeitet worden ist. Nach der vollständigen Ausführung der DO-Schleife erfolgt eine Synchronisation der Prozessoren, bevor die Ausführung des Programms fortgesetzt wird. In diesem Zusammenhang stellt die DO-Schleife also eine Kontrollstruktur dar, die die Synchronisationspunkte festlegt. Während die Definition von Kontrollstrukturen beim Microtasking manuell vorgenommen werden muß, kann dies beim Autotasking automatisch erfolgen. Die existierenden Multitasking-Implementationen sind vom jeweiligen System abhängig, d.h. Cray Microtasking und Autotasking können nur auf Systemen der Firma Cray genutzt werden. Ähnliche Multitasking-Implementationen wurden z.B. von IBM entwickelt [Reg89].

Parallelitätsgrad

Das Ziel bei der Ausführung eines Programms mit mehreren Prozessoren ist die Verkürzung der Ausführungszeit des Programms im Vergleich zur Ausführung mit nur einem Prozessor. Als Maß für diese Verkürzung wird der Speedup S definiert als Verhältnis von Ausführungszeit T_1 bei der Nutzung eines Prozessors zur Ausführungszeit T_p bei der Nutzung von p Prozessoren.

$$S = \frac{T_1}{T_p}$$

Wenn ein Programm zu 100 Prozent parallel ausgeführt wird, dann gilt für die Ausführungszeit T_p bei der Nutzung von p Prozessoren.

$$T_p = \frac{T_1}{p}$$

Der dabei erzielte Speedup S_{100} entspricht der Anzahl der genutzten Prozessoren p .

$$S_{100}(p) = p$$

Da jedoch im allgemeinen nur ein Teil eines Programms parallelisierbar ist, wird meist nur ein niedrigerer Speedup erzielt. Ein Maß für die tatsächliche Parallelität ist der Anteil f (*fraction*) der Zeit T_1 , in dem die parallelisierbaren Programmteile ausgeführt werden. Damit setzt sich die Zeit T_1 aus einem nicht parallelisierbaren Anteil $(1-f) T_1$ und einem parallelisierbaren Anteil $f T_1$ zusammen.

$$T_1 = (1-f) T_1 + f T_1$$

Bei paralleler Ausführung gilt damit für die Zeit T_p :

$$T_p = (1-f) T_1 + f \frac{T_1}{p}$$

Damit ergibt sich der Speedup als Funktion von f und p .

$$S(f,p) = \frac{T_1}{T_p} = \frac{1}{(1-f) + \frac{f}{p}}$$

Diese Formel wird als *Amdahl's law* bezeichnet. Da bei der parallelen Ausführung von Programmen jedoch durch zusätzliche Aktivitäten Overhead entsteht, der die Ausführungszeit erhöht, gibt diese Formel nur eine Obergrenze für den erreichbaren Speedup an. Um eine einheitliche Bezeichnungsweise für die in den folgenden Kapiteln relevanten Größen zu gewährleisten, wird die Variable f abweichend von der in der Literatur üblichen Bezeichnung im folgenden als Parallelitätsgrad PG bezeichnet.

Load-Balance-Faktor

Als *load balancing* bezeichnet man die Verteilung der Prozesse auf die Prozessoren. Eine Verteilung wird balanciert genannt, wenn in den parallelen Programmteilen alle Prozessoren ständig an der Ausführung beteiligt sind, d.h. wenn zum Beispiel alle Prozesse einer Kontrollstruktur aus denselben Anweisungen bestehen und jeder Prozessor die gleiche Anzahl von Prozessen ausführt. Im unbalancierten Fall sind die Prozessoren nicht gleichmäßig ausgelastet, so daß bei der Synchronisation der Prozessoren am Ende einer Kontrollstruktur für einen Teil der Prozessoren Wartezeiten (*time waiting on semaphores*) auftreten. Da während dieser Wartezeiten keine parallele Ausführung durch alle Prozessoren erfolgt, erhöht sich die Ausführungszeit in den parallelen Programmteilen unabhängig vom Parallelitätsgrad, und der Speedup wird kleiner. Als Maß für die Verteilung der Prozesse auf die Prozessoren wird der Load-Balance-Faktor LB eingeführt. Der Load-Balance-Faktor gibt das Verhältnis von genutztem Prozessorpotential zu vorhandenem Prozessorpotential an. Wenn z.B. 3 Prozesse einer DO-Schleife auf 4 Prozessoren verteilt werden, dann können 3 Prozessoren parallel arbeiten, während ein Prozessor warten muß. Der Load-Balance-Faktor beträgt in diesem Fall 0.75¹. Die genaue Definition des Load-Balance-Faktors erfolgt in Kapitel 4 im Zusammenhang mit den verwendeten Programmen.

¹ In den folgenden Kapiteln werden die Werte von Load-Balance-Faktor und Parallelitätsgrad in gleicher Bedeutung auch in Prozent angegeben, d.h. die Angabe 75% soll der Angabe 0.75 entsprechen.

2.3. Die verwendeten Meßinstrumente

Zur Leistungsbestimmung werden im Rahmen dieser Arbeit zwei verschiedene Instrumente verwendet, die im Betriebssystem UNICOS auf CRAY Rechnern zur Verfügung stehen. Jeder der Prozessoren verfügt über einen speziellen Satz von Registern, in denen ohne zusätzlichen Zeitaufwand bestimmte Ereignisse wie z.B. Anzahl der Gleitpunktoperationen, Anzahl der Speicherzugriffe oder Anzahl der Speicherzugriffskonflikte gezählt werden. Der Funktionsaufruf HPM (*Hardware Performance Monitor*, [Cra2040]) steuert die Nutzung dieser Register und stellt Auswerteroutinen bereit, die eine flexible Nutzung der dort gesammelten Informationen gestatten. Mit diesem Hilfsmittel können in Verbindung mit einem prozessoreigenen Taktzählregister (*real time clock*) bestimmte Zeitgrößen wie z.B. Wartezeiten, die aufgrund von Konfliktsituationen entstehen, auf Prozeßebene akkumuliert werden.

Im Betriebssystem UNICOS werden alle Systemaktivitäten durch *system calls* (Systemroutinen-Aufrufe) realisiert. Der System Monitor SAR (*System Activity Report*, [Cra2011]) liefert Informationen über Systemaktivitäten, indem er kostenlos die Anzahl der *system calls* in entsprechenden Zählern akkumuliert und darüber hinaus systemspezifische Größen wie z.B. Daten über den Dateitransfer oder die Länge von systeminternen Warteschlangen liefert.

3. Leistungsbewertung durch Benchmarking

Ein ständig wachsendes Angebot an Rechenanlagen für unterschiedliche Einsatzbereiche, die zunehmende Komplexität und die steigenden Kosten dieser Systeme machen Leistungsvergleiche und damit eine Leistungsbewertung notwendig. In diesem Kapitel wird die Leistungsbewertung von Supercomputern behandelt [Gel89, Mar88]. Dabei wird die Leistung dieser Rechner bei Problemlösungen im naturwissenschaftlich-technischen Bereich betrachtet, die im allgemeinen rechen- und speicherintensive Programme erfordern. Die Vielfalt der Architekturen von Supercomputern und die unterschiedlichen Anforderungen, die an diese Systeme gestellt werden, erschweren eine universelle Leistungsbewertung. In der Vergangenheit gab es eine Vielzahl von unterschiedlichen Ansätzen. Es gibt jedoch einige Versuche, die Leistungsbewertung durch Methodik zu systematisieren [GuNo89, PER89]. In diesem Kapitel werden beispielhaft mehrere Arbeiten aus diesem Bereich vorgestellt.

3.1. Methoden zur Leistungsbewertung

Die Leistung von Rechnern wird von unterschiedlichen Einflußfaktoren bestimmt. Außer von der Hardware des Rechners hängt die Leistung auch vom Betriebssystem ab, da dieses die Vergabe der Betriebsmittel steuert. Darüber hinaus sind der Compiler - insbesondere aufgrund der Vektorisierungseigenschaften - und die Parallelisierungsmöglichkeiten relevant. Diese Einflußfaktoren werden bei der Leistungsbewertung von Rechnern häufig separat betrachtet. Zur Leistungsbewertung von Rechnern sind unterschiedliche Methoden gebräuchlich (Abb. 2).

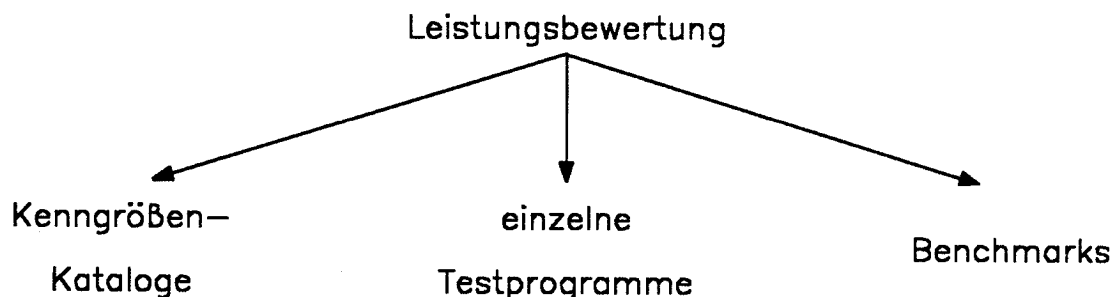


Abb. 2. Leistungsbewertung von Rechnern

Mit Hilfe der strukturellen und technischen Daten kann eine erste Kategorisierung von Rechnern vorgenommen werden. Dazu stehen für viele Rechner Kenngroößenkataloge [BEGK88, BEK90, Com89, HoJe88, HwBr86, Laz86] zur Verfügung, die einen detaillierten Vergleich ermöglichen. Solche und ähnliche Vergleiche werden in dieser Arbeit nicht weiter betrachtet.

Entsprechend der Abb. 2 beruhen alle weiteren Bewertungen von Rechnern auf Leistungsmessungen. Eine einfache Methode ist die Messung der Ausführungszeit eines einzelnen Testprogramms. Dieses kann zum Beispiel aus Programmkernen (*kernels*), d.h. aus den rechenzeitintensiven Schleifen eines Programms, oder auch aus einem vollständigen Anwendungsprogramm bestehen.

Im Gegensatz zu diesem Verfahren, bei dem nur ein einzelnes Programm betrachtet wird, werden bei einem Benchmark mehrere Programme ausgeführt. Ein Benchmark besteht aus einer Menge von Benchmark-Programmen, die eine definierte Belastung eines Systems repräsentieren sollen ohne dabei den für Tests akzeptablen Zeitrahmen zu verlassen. Während einerseits häufig die reale Belastung eines Systems von Interesse ist, werden andererseits auch solche Programme benötigt, die eine in der Zukunft erwartete Belastung repräsentieren. Die einzelnen Benchmark-Programme können dabei den oben erwähnten Testprogrammen entsprechen. Eine ausführliche Klassifikation der gebräuchlichen Benchmark-Programme wird in Kapitel 3.2 vorgenommen. Bei geeigneter Auswahl der einzelnen Benchmark-Programme ist so im Gegensatz zur Messung eines einzelnen Programms eine Leistungsbewertung für ein System unter tatsächlicher Last möglich.

Es besteht ein Unterschied zwischen Benchmarks mit dem Ziel der Leistungsbewertung eines Rechners für einen bestimmten Problembereich und dem der Leistungsbewertung relativ zu anderen Rechnern unter der jeweils gleichen anwenderspezifischen Arbeitslast [Ste88]. An der zuletzt genannten Klasse von Benchmarks sind besonders potentielle Käufer oder Benutzer interessiert, die verschiedene Rechner für ihre spezifischen Arbeitslasten vergleichen wollen. Bei Vergleichen favorisieren verschiedene Benchmarks nicht immer den gleichen Rechner. Dies liegt zum Teil daran, daß die Hersteller zum Beispiel ihre Compiler günstig in bezug auf bekannte Benchmarks auslegen. Solche Benchmarks können für erste Vergleiche herangezogen werden. Sie machen jedoch selten Aussagen über die Leistung eines Rechners in bezug auf die spezifische Arbeitslast des jeweiligen Benutzers. Aus diesem Grund führt man notwendigerweise zusätzlich eigene Benchmarks durch.

Bevor in den folgenden Kapiteln Benchmarks und Benchmark-Programme im Detail betrachtet werden, sollen noch zwei wichtige Begriffe erläutert werden:

3.1.1. Meßumgebungen

Die Ausführung von Benchmarks kann in unterschiedlichen Meßumgebungen erfolgen, die zu jeweils anderen Bewertungskriterien führen können. Im Rahmen dieser Arbeit werden nur solche Meßumgebungen betrachtet, die ausschließlich Batch-Verarbeitung zulassen, d.h. interaktive Lasten werden nicht einbezogen.

- **Einbenutzerbetrieb:**

Die einzelnen Benchmark-Programme können sequentiell im Einbenutzerbetrieb ausgeführt werden, d.h. jedem Benchmark-Programm stehen immer alle Ressourcen zur Verfügung. Dabei wird zur Leistungsbewertung der Mittelwert aller Ausführungszeiten gebildet. Diese Bewertung bezieht sich jedoch nur auf die Ausführung von jeweils einem Programm zu einer Zeit und macht weder eine Aussage über das Multiprogramming-Verhalten eines Systems noch über das Verhalten bei bestimmten Job-Profilen. Benchmarks dieser Art werden in Kapitel 3.3.1 näher beschrieben.

- **Mehrbenutzerbetrieb:**

Zur Beurteilung der Leistungsfähigkeit von Supercomputern im Rechenzentrumsbetrieb können Benchmarks im Mehrbenutzerbetrieb ausgeführt werden (siehe Ka-

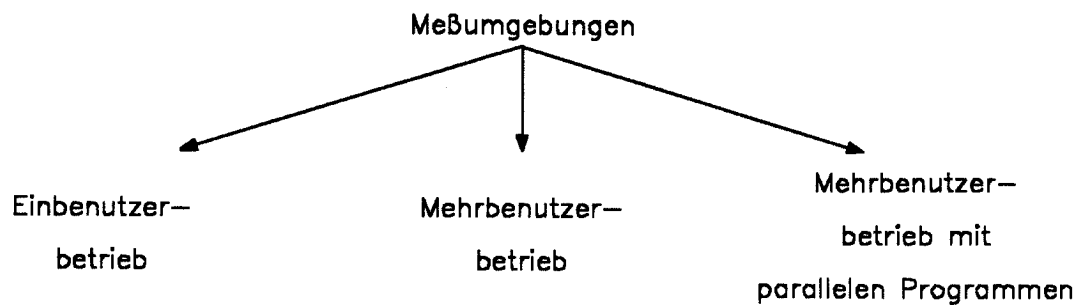


Abb. 3. Meßumgebungen

pitel 3.3.2). Dafür gibt es zwei grundsätzlich verschiedene Methoden. Zum einen besteht die Möglichkeit, ein Benchmark-Programm im normalen Produktionsbetrieb auszuführen. Solche Messungen sind jedoch im allgemeinen nicht reproduzierbar, da die Arbeitslast im normalen Produktionsbetrieb, die als Hintergrundlast für diese Messungen dient, häufig starken Schwankungen unterworfen ist. Aus diesem Grund sind solche Messungen von geringer Aussagekraft. Bei der zweiten Methode werden die Messungen in kontrollierter Umgebung durchgeführt, d.h. alle Benchmark-Programme werden zusammen auf einer dedizierten Maschine ausgeführt. Dabei ist neben den Ergebnissen für die einzelnen Benchmark-Programme die Ausführungszeit für den gesamten Benchmark von Interesse. Die so gewonnene Reproduzierbarkeit der Ergebnisse muß allerdings durch erheblich größeren Aufwand bei der Durchführung der Benchmarks bezahlt werden.

- **Mehrbenutzerbetrieb mit parallelen Programmen:**

Bei der Benutzung von Rechnern im naturwissenschaftlich-technischen Bereich zeichnen sich zwei entscheidende Entwicklungen ab. Zum einen werden Forschungsprojekte und Anwendungen und damit auch die zugehörigen Rechenprogramme immer umfangreicher, zum anderen steigt die Leistungsfähigkeit und Komplexität der verwendeten Mehrprozessorrechner. Diese Systeme ermöglichen eine Parallelverarbeitung von Programmen, die in bestimmten Fällen eine signifikante Verkürzung der Ausführungszeiten bewirkt. Dabei kann die Leistung der einzelnen Programme theoretisch um einen Faktor gesteigert werden, der der Anzahl der benutzten Prozessoren entspricht; der tatsächlich erzielbare Speedup hängt jedoch stark vom jeweiligen Programm ab. Mit der Zahl der verfügbaren Mehrprozessorrechner steigt auch die Zahl der Anwender, die die hohen Ausführungszeiten ihrer Programme durch Parallelverarbeitung verkürzen wollen. Die Parallelverarbeitung wird zur Zeit fast ausschließlich im Einbenutzerbetrieb, d.h. auf dedizierten Maschinen, angewendet. Da die Ausführung von Programmen in dedizierter Umgebung jedoch aus betriebstechnischen Gründen nur in Ausnahmefällen möglich ist und da die vorhandenen Systeme keinen flexiblen Wechsel zum Multiprogramming-Betrieb erlauben, gewinnt die Ausführung von parallelen Programmen in Multiprogramming-Umgebungen zunehmend an Bedeutung. Diese Strategie kann neben der Beschleunigung einzelner Programme eine wesentliche Verbesserung des Systemdurchsatzes bewirken. Bei Mehrprozessorrechnern mit gemeinsamem Speicher und realer Adressierung können durch große Programme Speicherengpässe auftreten, die eine effiziente Nutzung der übrigen Ressourcen nur durch Parallelverarbeitung innerhalb

der Programme zulassen [Bie87, Bie88]. Zur Zeit gibt es erst einige wenige Veröffentlichungen, die sich mit dieser aktuellen Entwicklung beschäftigen [DiIy89, DiIy90, Nag90b]. Darin wird sowohl die Leistung der parallelen Programme als auch der Durchsatz des Gesamtsystems untersucht. Die Parallelverarbeitung im Mehrbenutzerbetrieb kann also eine effektivere Ausführung von Programmen bei einer effizienteren Nutzung der Systemressourcen bewirken.

3.1.2. Meßgrößen

Als Ergebnis von Benchmark-Programmen erhält man Meßwerte von Größen, die eine Leistungsbewertung unter verschiedenen Gesichtspunkten erlauben. Als Meßgrößen sind zum Beispiel die verbrauchte Prozessorzeit (*cpu time*) und die Realzeit (*real time*) oder Ausführungszeit (*wallclock time* oder *turnaround time*) von Bedeutung. Die Ausführungszeit eines Programms ist der Zeitraum vom Start bis zum Ende des Programms, d.h. vom Beginn der Ausführung der ersten Anweisung bis zum Ende der letzten Anweisung. Die Ausführungszeit eines Benchmarks, bei dem alle Benchmark-Programme zusammen auf einer dedizierten Maschine ausgeführt werden, ist ein Maß für den Durchsatz eines Systems. Der Durchsatz eines Systems, das ist die pro Zeiteinheit vom System geleistete Menge an Arbeit, ist bei Messungen im Mehrbenutzerbetrieb von besonderem Interesse. Dabei kann als Einheit für die Arbeit z.B. die Anzahl der ausgeführten Instruktionen oder die Anzahl der bearbeiteten Jobs eingesetzt werden. In Abhängigkeit von der Meßumgebung und dem jeweiligen Benchmark sind zum Teil weitere Zeitgrößen für die Leistungsbestimmung relevant. Diese Größen werden im Verlauf dieser Arbeit im Zusammenhang mit den jeweiligen Benchmarks erläutert.

Außerdem werden die Verarbeitungsraten von Rechnern gemessen, welche häufig in Einheiten von MIPS (*Million Instructions Per Second*) angegeben werden. Dabei ist zu beachten, daß bei diesen Angaben auch solche Instruktionen berücksichtigt sind, die nur Adreßrechnungen ausführen. Deswegen ist die Einheit MIPS für die Angabe einer Rechenleistung nur bedingt geeignet. Dies gilt im besonderen für Vektorrechner, da diese mit einer Instruktion sowohl ein Resultat (Skalarinstruktion) als auch viele Resultate (Vektorinstruktion) produzieren können. Aus diesen Gründen werden die Verarbeitungsraten häufig auf der Basis von ausgeführten Operationen auf Daten angegeben. Es werden die Verarbeitungsraten für Gleitpunktop operanden in MFLOPS (*Million Floating-point Operations Per Second*), für ganzzahlige Operanden und für logische Operanden gemessen. Der *Hardware Performance Monitor* der Cray X-MP und Cray Y-MP Systeme ([Cra2040], siehe Kapitel 2), der die Anzahl der ausgeführten Operationen in speziellen Hardware-Registern ohne zusätzlichen Aufwand zählt, gibt zum Beispiel neben den Verarbeitungsraten in MIPS und MFLOPS die Vektorverarbeitungsrate von ganzzahligen und logischen Operationen an. Die Algorithmen in naturwissenschaftlich-technischen Anwendungen bestehen vorwiegend aus Gleitpunktop operationen. Da Supercomputer bislang primär für solche Anwendungen eingesetzt werden, ist die Mehrzahl der Anwender an der Gleitpunktverarbeitungsrate in MFLOPS interessiert. Als ein Maß wird häufig - vor allem durch Systemarchitekten und Hersteller - die theoretische Maximalverarbeitungsrate oder Spitzenleistung (*peak rate*) angegeben, welche bei simultaner Nutzung aller Gleitpunkt funktionseinheiten erzielt würde. Da diese theoretische Spitzenleistung praktisch nicht erreicht werden kann, sind die in Anwendungen bzw. in Benchmark-Programmen erzielten spezifischen Verarbeitungsraten

von größerer Aussagekraft. Auch die Effektivität der Maschine, die definiert ist als das Verhältnis von tatsächlich erreichter Leistung für eine bestimmte Operation oder für eine Kombination von Operationen zur theoretischen Spitzenleistung, ist eine wichtigere Größe [StPa89], insbesondere in bezug auf den Durchsatz eines Systems.

In einigen Benchmarks werden weitere Meßgrößen definiert, um die Leistungsbewertung in speziellen Fällen zu erweitern. Auch diese Größen werden im Verlauf dieser Arbeit im Zusammenhang mit den jeweiligen Benchmarks erklärt.

3.2. Ein Klassifikationsansatz für Benchmark-Programme

Einzelne Benchmark-Programme können von sehr unterschiedlichem Komplexitätsgrad sein. Dementsprechend beziehen sich auch die Ergebnisse auf verschiedene Teilsysteme eines Rechners. Der folgende Klassifikationsansatz baut auf einer Arbeit von J. Martin [Mar87] auf und klassifiziert die Benchmark-Programme bezüglich ihrer Komplexität.

- **Hardware-Tests:**

Den einfachsten Typ stellen Hardware-Tests dar. Sie bestehen zum Beispiel aus einer Vektorinstruktion mit zwei, drei oder vier Operanden und charakterisieren jeweils bestimmte Teile der Hardware. Zum Beispiel gehört Hockneys $(r_{\infty}, n_{1/2})$ -Benchmark [HoJe88] in diese Gruppe, der die maximale Leistung einer Vektorpipeline r_{∞} und die für die Hälfte der maximalen Leistung erforderliche Vektorlänge $n_{1/2}$ ermittelt. Darüber hinaus können mit diesem Typ eines Tests Speichereigenschaften und Ein-/Ausgabeleistungen eines Systems isoliert betrachtet werden [Det89, Hof90].

- **Programmkerne:**

Die inneren Schleifen eines Programms, in denen die meiste Rechenzeit verbraucht wird, werden Programmkerne genannt. Im Gegensatz zu Hardware-Tests bestehen sie aus Programmanweisungen, die aus realen Programmen entnommen wurden, und geben deshalb ein realistischeres Maß für die Leistung an. Da sie im Vergleich zum Originalprogramm eine größere Lokalität aufweisen, können im Originalprogramm auftretende Speicherengpässe bei separater Berechnung der Kerne entfallen, was die Leistung künstlich verbessert. Die erhaltenen Ergebnisse müssen ständig aktualisiert werden, da sie sehr stark von der jeweiligen Version des Compilers abhängig sind². Die *Livermore FORTRAN Kernels* [KLS86, McM86] gehören zu dieser Gruppe. Sie bestehen aus den rechenintensiven Kernen von Anwendungsprogrammen aus der Arbeitslast des *Lawrence Livermore National Laboratory*. Dabei wurden ausschließlich Kerne ausgewählt, die Gleitpunktberechnungen durchführen, zum Beispiel das Skalarprodukt oder die Elimination bei tridiagonalen Matrizen. Ein weiterer häufig verwendeter Benchmark aus dieser Gruppe ist der *Whetstone-Benchmark* [CuWi76], der ebenfalls aus ausgewählten Programmkernen besteht.

- **Grundroutinen:**

Oft wird das Zeitverhalten von Programmen von sehr häufig durchlaufenen mathematischen Grundroutinen wie z.B. Matrixmultiplikation oder Lösung von

² Aufgrund dieser Abhängigkeit sind Programmkerne auch für den Vergleich von Compilern geeignet [CDL88, Det87].

Gleichungssystemen bestimmt. Diese Routinen sind im Vergleich zu vollständigen Anwendungsprogrammen einfacher zu handhaben, charakterisieren aber wiederum jeweils nur Teile des zu testenden Systems. Dongarras *LINPACK-Benchmark* gehört in diese Gruppe [Don89]. Dongarra hat wiederholt Ergebnisse dieses Benchmarks für eine Vielzahl von Rechnern veröffentlicht, vom *Personal Computer* bis zum Supercomputer. Dieser Benchmark ist eine schnell verfügbare Referenz, die jedoch auf einen speziellen Problembereich begrenzt und somit nur bedingt verwendbar ist [KLS86]. Dongarra selbst sagt von diesem Benchmark: "Die hier wiedergegebenen Zeiten sollten in keiner Weise dafür verwendet werden, die gesamte Systemleistung zu beurteilen. Die Ergebnisse beziehen sich nur auf ein Problem: Die Lösung von dicht besetzten linearen Gleichungssystemen mittels *LINPACK-Programmen* in einer FORTRAN-Umgebung"³.

Grundroutinen unterscheiden sich von Programmkernen insofern, als sie nicht nur aus einer Schleife, sondern aus kompletten Problemlösungen bestehen. Der Aufwand zur Durchführung und die Aussagekraft der Ergebnisse sind im wesentlichen gleich.

- **Gekürzte Anwendungsprogramme:**

Vollständige Anwendungsprogramme sind oft wegen ihrer Ausführungszeit und ihrer Größe schlecht zu handhaben. Um insbesondere die Handhabung in Bezug auf die Portabilität zu erleichtern, werden die Programme gekürzt. Dabei ist darauf zu achten, daß das charakteristische Zeitverhalten eines Programms weitgehend erhalten bleibt. So kann zum Beispiel das Herausstreichen von Ein-/Ausgaberroutinen dieses Verhalten völlig verändern.

- **Vollständige Anwendungsprogramme:**

Häufig werden vollständige Anwendungsprogramme als Benchmark-Programme verwendet. Diese sind am besten geeignet, um eine reale Arbeitslast zu repräsentieren. Die für Vergleiche von Rechnern notwendige Portierung der Programme ist jedoch im allgemeinen mit erheblichem Aufwand verbunden. Ein weiterer Nachteil ist die geringe Flexibilität der Programme in Bezug auf charakteristische Programmparameter wie z.B. die Ausführungszeit oder den Hauptspeicherbedarf. Es ist zum Beispiel im allgemeinen nicht möglich, dasselbe Programm mit unterschiedlichem Speicherbedarf auszuführen, ohne dabei die Ausführungszeit zu verändern.

- **Synthetische Programme:**

Synthetische Benchmark-Programme bestehen aus Modulen, die unterschiedliche Teile eines Rechners belasten. Einige Module sind sehr rechenintensiv, während andere den Speicher beanspruchen oder Ein-/Ausgabeoperationen durchführen. Durch Kombinationen dieser Module werden synthetische Programme erstellt, die das Verhalten realistischer Programme nachbilden sollen. Synthetische Benchmark-Programme haben gegenüber Anwendungsprogrammen den Vorteil, daß sie flexibel in Bezug auf eine darzustellende Arbeitslast sind. Dabei erfordert die Modellierung der nachzubildenden Arbeitslast den größten Aufwand.

Häufig werden Kombinationen der hier aufgeführten Benchmark-Programme benutzt, die gemeinsam eine künstlich zusammengestellte Arbeitslast bilden. Dies ist ein Kompromiß zwischen einfacher Handhabbarkeit und Aussagekraft der Ergebnisse. Dabei

³ Übersetzung des Autors aus [Don89].

können die Ausführungszeiten von einzelnen Benchmark-Programmen zu einem Mittelwert zusammengefaßt werden, wofür es viele unterschiedliche Methoden gibt, die zu unterschiedlichen Ergebnissen führen können [HaLu88].

Bei der Auswahl von Benchmark-Programmen sollte die Arbeitslast in einem Rechenzentrum sorgfältig analysiert werden, da diese Arbeitslast durch die Benchmark-Programme möglichst gut repräsentiert werden soll. Deswegen sollten für die Erstellung von Benchmark-Programmen die jeweils meistbenutzten Programme einer Arbeitslast zugrundegelegt werden. Die Ausführung einer Arbeitslast als Benchmark kann sowohl im Einbenutzerbetrieb als auch im Mehrbenutzerbetrieb erfolgen.

3.3. Supercomputer-Benchmarks

Ein wichtiges Kriterium für die Aussagekraft von Benchmarks ist die Meßumgebung, in der die Ergebnisse ermittelt werden. Im Gegensatz zu Benchmarks im Einbenutzerbetrieb (*single user environment*) können Messungen im Mehrbenutzerbetrieb (*multi user environment*) Aussagen über den Durchsatz eines Systems machen. Die Gliederung der folgenden Benchmarks wurde bezüglich der unterschiedlichen Meßumgebungen vorgenommen.

Die Erstellung einer vollständigen Liste aller Benchmarks ist nahezu unmöglich. Deshalb sollen die hier beschriebenen Benchmarks als Beispiele für die einzelnen Kategorien dienen.

3.3.1. Einbenutzerbetrieb

Die Benchmarks in diesem Kapitel werden im Einbenutzerbetrieb durchgeführt, d.h. dem jeweiligen Benchmark-Programm stehen immer alle Systemressourcen zur Verfügung. Alle Ergebnisse sind nur für diese Betriebsart gültig.

Der Benchmark von J. M. van Kats et al. ist ein typischer Supercomputer-Benchmark. Eine Weiterentwicklung stellt der *Synthetic Standard Benchmark* von A. J. van der Steen und R. J. van der Pas dar. Der Benchmark vom *PERFECT Club* (*PERformance Evaluation for Cost-effective Transformations*) ist der Versuch einer Standardisierung, d.h. ein Benchmark, der für eine Vielzahl von Architekturen geeignet sein soll. In *PARC-Bench* (*Palo Alto Research Center Benchmark*) wird ein systemorientierter Ansatz gemacht. Dieser Benchmark ist durch die Definition geeigneter Meßgrößen insbesondere für Mehrprozessorsysteme geeignet.

3.3.1.1. Ein universeller Supercomputer-Benchmark

Der folgende Benchmark wurde für unterschiedliche Gruppen von Supercomputern bzw. Minisupercomputern durchgeführt [KaSt87, KLS86, StPa88]. Es war die Intention, reale Programme zu verwenden, dennoch wurden bekannte Benchmark-Programme, die aus Programmkernen bestehen, in den Benchmark einbezogen, um eine Referenz bezüglich anderer Systeme zu ermöglichen.

Der Benchmark setzt sich aus den drei folgenden Tests zusammen:

- bekannte Benchmark-Programme (Programmkerne),
- schwer vektorisierbare Programmkerne,
- Anwendungsprogramme.

Als bekannte Benchmark-Programme wurden die *Livermore FORTRAN Kernels* (siehe oben) und die Kerne der Deutschen Forschungs- und Versuchsanstalt für Luft- und Raumfahrt (*DFVLR-Programmkerne*) verwendet [KLS86]. Die *DFVLR-Programmkerne* bestehen aus teilweise sehr einfachen Operationen wie der Addition, der Multiplikation und der Division in Schleifen der Länge $N=100$ und $N=1000$, aber auch aus Grundroutinen wie der Nullstellenbestimmung von Polynomen neunten Grades. Damit geben diese Programme Informationen über Geschwindigkeit und Abstimmung von Prozessor und Speicher. Messungen zu Speicherbankkonflikten und einige einfache Rechnungen aus der linearen Algebra ergänzen diesen Teil des Benchmarks.

Die schwer vektorisierbaren Programmkerne bestehen aus verschachtelten DO-Schleifen mit komplizierten Datenabhängigkeiten und Schleifen mit indirekter Adressierung. Diese testen die Vektorisierungs- und gegebenenfalls die Parallelisierungseigenschaften der Compiler. Da die Güte der Compiler auch die Ergebnisse der anderen Tests beeinflusst und in bezug auf die Systemleistung nur eine Einflußgröße von vielen darstellt, sollten die Ergebnisse dieses Tests nur mit geringer Wichtung in die Bewertung des Gesamtsystems einbezogen werden, damit der Einfluß auf die Systemleistung nicht überbewertet wird. Dieser Test ermöglicht eine Beurteilung der Compiler, die bei der Interpretation der Ergebnisse aus den anderen Tests von Nutzen sein kann.

Im dritten Teil werden Tests mit Anwendungsprogrammen durchgeführt, welche das Verhalten der Systeme unter realer Last widerspiegeln. Dabei ist anzumerken, daß diese Programme im allgemeinen jeweils für eine bestimmte Maschine optimiert wurden, was bei der Interpretation der Ergebnisse zu beachten ist.

Dieser Benchmark liefert neben Vergleichsmöglichkeiten mit anderen Rechnern durch Einbeziehung bekannter Benchmarks Leistungsvergleiche unter verschiedenen Aspekten. Alle Messungen wurden im Einbenutzerbetrieb gemacht, d.h. dem Meßprogramm stehen immer alle Ressourcen zur Verfügung.

3.3.1.2. Der Synthetic Standard Benchmark

Der *Synthetic Standard Benchmark* ist aufbauend auf dem oben beschriebenen Benchmark von J. M. van Kats et al. systematischer zusammengestellt [StPa89]. Er besteht aus Modulen mit hierarchisch steigendem Komplexitätsgrad, welche sich jeweils aus acht bis zwölf Programmen zusammensetzen. Bei dieser Konzeption ermöglichen die Ergebnisse aus einem Modul die Erklärung der Ergebnisse aus hierarchisch höheren Modulen. Modul 1 besteht aus einfachen Hardware-Tests wie etwa dem $(r_{\infty}, n_{1/2})$ -Benchmark und aus den *DFVLR-Kernen* (siehe Kapitel 3.3.1.1). Diese Programme sind sehr prozessorintensiv und sollen die Leistungsstärken und -schwächen der Hardware im Einklang mit der Software offenlegen.

Einfache Grundroutinen aus der linearen Algebra, wie z.B. die Lösung von voll und schwach besetzten linearen Systemen, die Schnelle Fourier-Transformation [Bri88, Bur89] oder die Erzeugung von Zufallszahlen, bilden Modul 2. Die hier gemessenen Leistungen lassen sich mit den Ergebnissen aus Modul 1 zumindest teilweise erklären.

In Modul 3 befinden sich kleine Anwendungsprogramme, die meist mehr als eine der Grundroutinen aus Modul 2 benutzen. Unter anderem werden durch Programme mit

sehr hohem Speicherbedarf die Ein-/Ausgabefähigkeiten der Systeme getestet. Da die Grundroutinen in diesen Programmen in Modul 2 schon separat betrachtet wurden, ist wiederum eine weitgehende Erklärung der hier erhaltenen Ergebnisse möglich. Auch hier werden alle Messungen im Einbenutzerbetrieb durchgeführt, d.h. es kann keine Aussage über den Systemdurchsatz gemacht werden.

3.3.1.3. Ein Benchmark vom PERFECT Club

Vom *PERFECT Club* (*PERFormance Evaluation for Cost-effective Transformations*) wird bemängelt, daß die existierenden Benchmarks in keiner festen Beziehung zueinander stehen und daß sie außerhalb ihrer speziellen Problemumgebung keine Aussagekraft haben [PER89]. Aus diesem Grund wird eine Standardisierung des Benchmarking angestrebt. Dabei geht es nicht um einen Benchmark zur Bestimmung des Systemdurchsatzes, sondern um die Schaffung einer Vergleichsmöglichkeit für eine große Spanne verschiedener Architekturen von verschiedenen Herstellern. Durch die Verwendung von Anwendungsprogrammen haben hier die Ergebnisse im Vergleich zu Dongarras *LINPACK-Benchmark* (siehe Kapitel 3.2) eine größere Aussagekraft. Grundsätzlich werden drei Ziele verfolgt:

1. Die Entwicklung einer Methode, die die Erzeugung einer Sammlung von FORTRAN-Benchmarks aus portablen Anwendungsprogrammen ermöglicht.

Die Idee, Anwendungsprogramme als Benchmark zu verwenden, ist nicht neu. Allerdings besteht oft das Problem, daß Anwendungsprogramme nur begrenzt portabel sind und somit nur auf einer geringen Anzahl von Rechnern ausgeführt werden können. Aus diesem Grund werden bei diesem Ansatz alle Programme in den *PERFECT FORTRAN Standard* umgeschrieben. Der *PERFECT FORTRAN Standard* wurde speziell für diesen Benchmark entwickelt und macht Einschränkungen bezüglich des FORTRAN 77 Standards. Auf diese Art wird eine größere Portabilität gewährleistet. Aus demselben Grund werden Ein- und Ausgabe aus den Programmen gestrichen. Dadurch wird das Zeitverhalten der Programme unter Umständen völlig verändert, was bei der Interpretation der Ergebnisse beachtet werden muß.

2. Die Erstellung dieser Sammlung und die Durchführung der Benchmarks auf einer Vielfalt von Supercomputern.

Da diese Sammlung nicht vollständig sein kann, muß eine repräsentative Auswahl gemacht werden. Die aktuelle Sammlung besteht aus Anwendungsprogrammen aus dem naturwissenschaftlich-technischen Bereich, die auf einer Vielzahl von verschiedenen Rechnern erstellt wurden.

3. Die Analyse von Optimierungsstrategien für häufig vorkommende Algorithmen und deren Einfluß auf die Leistung des Anwendungsprogramms.

Die Testprogramme werden jeweils für verschiedene Optimierungsstrategien durchgeführt, welche zusammen mit den korrespondierenden Ergebnissen festgehalten werden. Dabei werden die Optimierungen auf drei verschiedenen Ebenen vorgenommen. Auf der einfachsten Ebene werden nur Direktiven für die Compiler in den

Quelltext eingefügt. Diese steuern zum Beispiel die Parallelisierung der Programme. Die zweite Optimierungsebene beinhaltet Modifikationen des Quelltextes. Ein einfaches Beispiel ist eine Änderung innerhalb einer DO-Schleife, welche die Vektorisierung oder die Parallelisierung der Schleife ermöglicht. Auf der dritten Ebene werden Optimierungen bezüglich der Algorithmen vorgenommen. Dabei werden zum Beispiel serielle Algorithmen durch parallele Algorithmen ersetzt, um die parallele Ausführung zu ermöglichen.

Da die Anwendungsprogramme im Original jeweils für eine spezielle Maschine geschrieben worden sind, können die Ergebnisse für die einzelnen Optimierungsstrategien nicht in Relation zu einer optimalen seriellen Version interpretiert werden. Sie sollen vielmehr die Auswirkungen der verschiedenen Optimierungsstrategien bei unterschiedlichen Architekturen aufzeigen und damit das Verständnis dieser Architekturen vertiefen.

Eine sinnvolle Erweiterung dieses Benchmarks wäre die Definition einer ähnlichen Vorgehensweise für Messungen im Mehrbenutzerbetrieb, welche Vergleiche einer Vielzahl von Rechnern bezüglich des Systemdurchsatzes ermöglichen würde.

3.3.1.4. PARC-Bench - ein systemorientierter Ansatz

Die Ergebnisse von Benchmarks können besonders bei Mehrprozessorarchitekturen häufig unerklärlich erscheinen, da die Einflüsse der Prozessoren aufeinander nur schwer zu erfassen sind. Deswegen wird in [GuNo89] ein systemorientierter Ansatz gemacht: *PARC-Bench* (*Palo Alto Research Center Benchmark*). Dieser Benchmark soll als Vergleichsgrundlage für ein Spektrum von Architekturen der Klasse der universellen Mehrprozessorarchitekturen dienen. Zu dieser Klasse gehören die *MIMD*-Mehrprozessorsysteme⁴ mit gemeinsamem Speicher [HwBr86].

Von den unterschiedlich komplexen Ebenen der Architektur wird jeweils nur die einfachste betrachtet. Die bisher fertiggestellten Komponenten von *PARC-Bench* untersuchen zum Beispiel den Einfluß von speicherintensiven Anwendungsprogrammen auf die jeweiligen Mehrprozessorsysteme. Dazu wird in einem Testprogramm von mehreren Prozessoren auf eine gemeinsame Variable zugegriffen, was zu Zugriffskonflikten führt und so eine Sperreffizienz (*lock contention efficiency*) ermittelt. Diese Sperreffizienz E^s ist definiert als das Verhältnis der Ausführungszeit T_1 des Programms mit einem Prozessor zum Produkt aus Ausführungszeit T_p^s mit mehreren Prozessoren und Anzahl der Prozessoren p :

$$E^s = \frac{T_1}{p T_p^s}$$

Der Index s soll dabei den speziellen Test von speicherintensiven Programmen kennzeichnen. Die anderen Komponenten arbeiten in ähnlicher Weise. So wird zum Beispiel die Erzeugung von neuen Prozessen und der daraus resultierende Abfall der Systemleistung untersucht. Dabei wird die Abarbeitung eines Jobs in Abhängigkeit von der Anzahl der dafür erzeugten Prozesse insbesondere bei Systemschwächung (*system degradation*) untersucht. Systemschwächung liegt dann vor, wenn die Zahl der Prozesse größer

⁴ *MIMD* = *Multiple Instruction stream Multiple Data stream*

ist als die Zahl der Prozessoren. Für diese Untersuchung wird ein Maß für die System-schwächung definiert. Außerdem wird der Overhead von virtuellen Speichern, die Leistung von Plattenspeicher und Netzwerk sowie das Systemverhalten bei Kontextwechsel untersucht. Diese Definitionen von neuen systemorientierten Größen ermöglichen eine Analyse der Systemleistung in einem mehrdimensionalen Parameterraum, die besonders bei Multiprozessorarchitekturen die sonst verborgenen Leistungseigenschaften offenlegt.

Die in diesem Kapitel beschriebenen Benchmarks erlauben keine Leistungsbewertung von Rechnern bezüglich des Systemdurchsatzes. Sie ermöglichen vielmehr Leistungsvergleiche auf niedrigeren Ebenen, die jeweils andere Systemkomponenten betreffen. Dieser Vergleich von einzelnen Systemkomponenten fördert das Verständnis der Leistungsfähigkeit von verschiedenen Architekturen. Die Ergebnisse der Benchmarks im Einbenutzerbetrieb sind damit insbesondere bei der Erstellung von Benchmarks im Mehrbenutzerbetrieb von Nutzen, da hier der Einfluß von charakteristischen Benchmark-Parametern auf einzelne Systemkomponenten im Zusammenhang mit der Leistung des Gesamtsystems beachtet werden muß.

3.3.2. Ein Benchmark im Mehrbenutzerbetrieb

Bei der Ausführung von typischen Produktionsprogrammen findet meist ein wiederholter Wechsel zwischen der Ein-/Ausgabe und den Berechnungen statt. Deswegen können einzelne Programme im allgemeinen nicht ständig alle Betriebsmittel eines Rechners nutzen. Dies gilt insbesondere für Supercomputer. Um eine effiziente Nutzung zu gewährleisten, werden diese Rechner im Mehrbenutzerbetrieb betrieben. Deswegen macht auch eine Messung mit einer der realen Last sehr nahe kommenden Arbeitslast keine Aussage über die Leistung im realen Betrieb, wenn diese Messung im Einbenutzerbetrieb durchgeführt wird.

Die Beurteilung eines Supercomputers im realen Betrieb erfordert Messungen im Mehrbenutzerbetrieb. In einem Benchmark von [HeKa89] werden Benchmark-Programme in verschiedenen Meßumgebungen ausgeführt. Dabei sind die Durchsatztests im Mehrbenutzerbetrieb von besonderem Interesse. Der gesamte Benchmark ist in die vier folgenden Tests gegliedert:

- Hardware-Tests, einfache Programmkerne und Grundroutinen;
- Vollständige Anwendungsprogramme im Einbenutzerbetrieb (*single user environment*);
- Durchsatztests mit Anwendungsprogrammen im Mehrbenutzerbetrieb (*multiprogramming environment*);
- Interaktiver Multiuser-Durchsatztest und Antwortzeittest.

Da in dieser Arbeit ausschließlich Batch-Umgebungen betrachtet werden, wird auf die Untersuchungen des letzten Tests nicht näher eingegangen.

- **Hardware-Tests, einfache Programmkerne und Grundroutinen:**
Hier sind zum Beispiel Addition, Quadratwurzel und Matrixmultiplikation vertreten, also Elemente aus den einfachen Gruppen der oben angegebenen Klassifikation. Diese Tests sind für Skalar-, Vektor- und Parallelverarbeitung vorgesehen und wer-

den mit unterschiedlichen Vektorlängen durchgeführt, so daß neben der Maximalleistung für bestimmte Operationen auch eventuelle Engpässe sichtbar werden. Darüber hinaus wird die Ein-/Ausgabe untersucht. Alle Tests werden auf einer dedizierten Maschine durchgeführt (siehe Kapitel 2) und geben keine Information über die Leistung im realen Betrieb, sind also in diesem Benchmark von geringerer Bedeutung. Ihr Vorteil gegenüber den anderen Tests liegt in der guten Portabilität.

- **Vollständige Anwendungsprogramme im Einbenutzerbetrieb:**

Auf dieser Ebene werden vollständige oder gekürzte Anwendungsprogramme ausgeführt, jeweils ein Programm zu einer Zeit auf einer dedizierten Maschine. Im Idealfall würde man alle Programme einer Arbeitslast einmal ausführen, was aber nicht praktikabel ist, da manche Programme nicht portabel sind. Deswegen wurde eine repräsentative Auswahl getroffen, wobei die Aussagekraft der Ergebnisse sehr stark von dieser Auswahl abhängt. Bei diesem Test wird der Mittelwert aus den gewichteten Ausführungszeiten der einzelnen Benchmark-Programme gebildet.

- **Durchsatztests mit Anwendungsprogrammen im Mehrbenutzerbetrieb:**

Dieser Test gibt ein Maß für den Durchsatz eines Systems bei einer bestimmten Arbeitslast an. Der Durchsatz ist sowohl ein Maß für die Leistungsfähigkeit der Hardware, welche Multiprogramming unterstützt, als auch für den gesamten System-Overhead, der zum Beispiel durch Kontextwechsel, *paging* und *swapping* erzeugt wird. Für diesen Test werden auf einem dedizierten System mehrere (gekürzte) Anwendungsprogramme simultan ausgeführt. Dabei treten Abhängigkeiten von mehreren Parametern wie z.B. von der Priorität der einzelnen Jobs oder von der Größe der Zeitscheiben (siehe Kapitel 2) auf. Verbraachte Prozessorzeit und Realzeit für die gesamte Arbeitslast sind hier die interessantesten Größen. Aus der Realzeit der gesamten Arbeitslast wird der Durchsatz ermittelt. Da bei diesem Test das Betriebssystem eine entscheidende Rolle in bezug auf die Leistung spielt, sind für die Interpretation der Ergebnisse weitere Zeitgrößen von Interesse, z.B. die vom Betriebssystem verbrauchte Prozessorzeit (*system time*), die Wartezeit der einzelnen Jobs (*wait time*: Zeit, in der kein Prozessor zur Verfügung steht), oder die Zeit, in der die Prozessoren nicht genutzt werden (*idle time*). Die Bestimmung dieser Größen ist nur auf einigen Systemen möglich und wird in dieser Arbeit nicht durchgeführt.

Die Ergebnisse dieser Tests werden durch gewichtete Addition zu einem Gesamtergebnis zusammengefaßt. Der so erhaltene Zahlenwert soll einen Vergleich von mehreren Systemen ermöglichen. Da in diesem Zahlenwert alle Einzelaspekte eines Vergleichs bereits bewertet und zusammengefaßt sind, sollen die Ergebnisse des Vergleichs nur im Zusammenhang mit der speziellen Bewertung interpretiert werden. Ein Vergleich aufgrund eines einzigen Zahlenwertes für jedes System ist hier jedoch nicht sinnvoll, da die Komplexität der bewerteten Systeme eine differenzierte Betrachtungsweise erfordert. Im Gegensatz dazu ermöglichen die Einzelergebnisse der Tests einen Vergleich, der differenzierte Aussagen über jeweils andere Teilsysteme erlaubt.

Die im nächsten Kapitel betrachtete Ausführung von parallelen Programmen in einer Multiprogramming-Umgebung ist besonders in bezug auf die derzeit verfügbaren Supercomputer eine wichtige Erweiterung der in diesem Benchmark im dritten Test durchgeführten Untersuchungen.

3.3.3. Mehrbenutzerbetrieb mit parallelen Programmen

Die Parallelisierung von Programmen ist in bezug auf die effiziente Nutzung eines Mehrprozessorrechners immer dann sinnvoll, wenn ein Programm beschleunigt werden soll oder wenn der Grad des Multiprocessing (siehe Kapitel 2) nicht ausreichend ist, weil zum Beispiel ein Programm eine Ressource vollständig benötigt.

Bei der Entwicklung von universellen Supercomputern werden zur Zeit zwei grundsätzlich verschiedene Wege beschritten. Den massivparallelen Systemen mit verteiltem Speicher auf der eine Seite stehen die Mehrprozessorsysteme mit gemeinsamem Speicher gegenüber. Bei dieser zuletzt genannten Gruppe von Rechnern findet das Konzept der realen Adressierung eine wachsende Verbreitung⁵. Durch das Konzept des Realspeichers kann in einer Multiprogramming-Umgebung bei Programmen mit großem Speicherplatzbedarf der Durchsatz des Systems sinken. Diese Durchsatzverminderung tritt immer dann auf, wenn die Anzahl der Programme im Speicher kleiner ist als die Anzahl der Prozessoren und jedes dieser Programme jeweils nur einen Prozessor nutzt, d.h. wenn die Anzahl der aktiven Prozesse kleiner ist als die Anzahl der im System verfügbaren Prozessoren. Durch die Parallelisierung der Programme kann die Anzahl der Prozesse erhöht und damit der Systemdurchsatz unter Umständen signifikant verbessert werden [Bie88].

Mit der Notwendigkeit von parallelen Programmen in einer Multiprogramming-Umgebung steigt auch das Interesse an einer Leistungsbewertung von Rechnern in dieser Betriebsart. Für diese Leistungsbewertung muß zum einen der Einfluß der Multiprogramming-Umgebung auf die Leistung von parallelen Programmen untersucht werden. In Kapitel 3.3.3.1 wird nur dieser Aspekt betrachtet. Zum anderen haben die parallelen Programme Auswirkungen auf die übrige Arbeitslast in einer Multiprogramming-Umgebung. Mit dem System *PAR-Bench* (siehe Kapitel 3.3.3.2) ist die Leistungsbewertung für diese Betriebsart unter beiden Aspekten möglich.

3.3.3.1. Der Einfluß des Multiprogramming

In Arbeiten von R. T. Dimpsey und R. K. Iyer [Dily89, Dily90] wird der Einfluß des Overheads durch Multiprogramming auf die Leistung von parallelen Anwendungsprogrammen untersucht. Zu diesem Zweck werden der Multiprogramming-Overhead und der gesamte System-Overhead eines parallelen Anwendungsprogramms in einer Multiprogramming-Umgebung gemessen.

Alle Messungen beziehen sich auf den Minisupercomputer Alliant FX/8. Der Alliant FX/8 ist ein *shared-memory*-Rechner und verfügt bei diesen Messungen über acht Prozessoren, die entweder unabhängig (*detached*) serielle Jobs oder aber alle acht gemeinsam (*clustered*) einen parallelen Job bearbeiten. Desweiteren stehen drei Prozessoren für interaktive Arbeiten, Ein-/Ausgabe und Betriebssystemaufgaben zur Verfügung. Aufgrund der sehr speziellen Architektur dieses Rechners lassen sich die Ergebnisse dieser Messungen nicht verallgemeinern und sind somit für andere Mehrprozessorarchitekturen

⁵ Nachdem im Bereich der Supercomputer die Firma Cray in der Vergangenheit der einzige Anbieter von Mehrprozessorsystemen mit gemeinsamem Speicher und realer Adressierung gewesen ist, hat jetzt auch die Firma NEC mit der NEC SX-3 das Konzept des Realspeichers verwirklicht. Darüber hinaus sieht auch die VP2000 der Firma Fujitsu für die Vektorarithmetik reale Adressierung vor.

von geringer Aussagekraft. Die Autoren führen alle Messungen für jeweils drei verschiedene Anwendungsprogramme durch, die der Benchmark-Sammlung des *PERFECT-Clubs* angehören (siehe Kapitel 3.3.1.3).

In der ersten Meßreihe werden nur parallele Jobs betrachtet, die in der Konfiguration *clustered* bearbeitet werden. Mit Messungen in kontrollierter Umgebung, d.h. auf einer dedizierten Maschine, wird versucht, eine untere Grenze⁶ für den Multiprogramming-Overhead bei parallelen Jobs zu ermitteln. Dazu wird ein paralleles Anwendungsprogramm zusammen mit zusätzlichen parallelen Jobs auf einem dedizierten System ausgeführt. Dabei erzeugen die zusätzlichen Jobs keinen System-Overhead. In Abhängigkeit von der Anzahl der zusätzlichen Jobs wird die Ausführungszeit des Anwendungsprogramms gemessen. Da diese Ausführungszeit im Multiprogramming-Betrieb gemessen wird, enthält sie auch den durch Multiprogramming erzeugten Overhead. Dann wird diese Ausführungszeit unter der Annahme, daß im Multiprogramming-Betrieb kein zusätzlicher Multiprogramming-Overhead erzeugt wird, approximativ berechnet. Der gesuchte Multiprogramming-Overhead ergibt sich dann aus der Differenz der gemessenen und der approximierten Ausführungszeit. Das Ergebnis von 4% als Untergrenze für den Multiprogramming-Overhead⁷ wird durch eine methodisch andere Messung bestätigt [DiIy89].

In einer weiteren Meßreihe werden die gleichen Untersuchungen mit zusätzlichen seriellen Jobs gemacht. Hier steigt die Untergrenze für den Multiprogramming-Overhead auf 5,3% an.

Im Gegensatz zu den bisherigen Messungen wird im folgenden der Multiprogramming-Overhead im realen Produktionsbetrieb für jeweils andere Hintergrundlasten bestimmt [DiIy90]. Dabei bestehen die Hintergrundlasten für die Messungen vorwiegend aus parallelen Programmen. Der Multiprogramming-Overhead wird wiederum aufgrund einer Differenz aus Messung und Approximation ermittelt. In dieser Betriebsart sind sowohl die gemessenen Ausführungszeiten der Anwendungsprogramme als auch die ermittelten Werte für den Multiprogramming-Overhead von der jeweiligen Hintergrundlast abhängig und schwanken deshalb in weiten Bereichen. Die Approximation wird hier auf der Grundlage von Meßwerten durchgeführt, die einen statistischen Fehler enthalten, der bei der Approximation durch eine Zufallsgröße dargestellt wird. Aus diesem Grund werden mehrere Messungen durchgeführt, die bezüglich Mittelwert und Standardabweichung statistisch ausgewertet werden. Für den Multiprogramming-Overhead wird ein Mittelwert von 16%, für den gesamten System-Overhead werden 24% ermittelt. Außerdem wird eine schwache Korrelation zwischen der Ausführungszeit des parallelen Anwendungsprogramms und dem jeweiligen Multiprogramming-Overhead bzw. System-Overhead festgestellt.

Aus diesen Ergebnissen kann gefolgert werden, daß der Multiprogramming-Overhead eine signifikante Leistungsverringerung für parallele Anwendungsprogramme bewirkt.

⁶ Da bei diesen Messungen keine Speicherbankkonflikte durch Multiprogramming auftreten, ist ein entsprechender Overhead im gemessenen Multiprogramming-Overhead nicht enthalten, und es wird nur eine untere Grenze ermittelt.

⁷ Die Angaben für den Multiprogramming-Overhead erfolgen hier bezogen auf die gesamte Ausführungszeit in Prozent.

Dabei ist anzumerken, daß diese Ergebnisse nur in den Umgebungen gültig sind, die während der Untersuchungen vorherrschten. Es können keine Aussagen über die Leistung von parallelen Programmen in anderen Umgebungen gemacht werden. Insbesondere wurden diese Untersuchungen nur in solchen Multiprogramming-Umgebungen durchgeführt, die selbst im wesentlichen aus parallelen Programmen bestanden. Deswegen können keine Aussagen über die Leistung von parallelen Programmen in Multiprogramming-Umgebungen mit vorwiegend sequentiellen Programmen gemacht werden. Die festgestellte Leistungsverringerung bei parallelen Programmen in Multiprogramming-Umgebungen mit vorwiegend parallelen Programmen läßt eine Leistungsverringerung für die gesamte Arbeitslast und damit eine Verringerung des Systemdurchsatzes vermuten. Mit dem im folgenden Kapitel vorgestellten System kann insbesondere auch dieser Sachverhalt untersucht werden.

3.3.3.2. Ein parametergesteuerter Benchmark

Die bisher vorgestellten Benchmarks unterscheiden sich grundsätzlich bezüglich der Ziele, die sie verfolgen. Die Benchmarks im Einbenutzerbetrieb ermöglichen eine grobe Einordnung von Rechnern und können durch die gezielte Untersuchung von Einzelaspekten dem Verständnis der unterschiedlichen Architekturen dienen, welches bei der Erstellung von Benchmarks im Mehrbenutzerbetrieb eine wichtige Voraussetzung ist. Nur Untersuchungen im Mehrbenutzerbetrieb bieten ein Maß für die Systemleistung im realen Betrieb. Der Aufwand für die Erstellung und Durchführung dieser Benchmarks ist entscheidend höher als im Einbenutzerbetrieb.

Da die parallele Verarbeitung von Programmen insbesondere auch in Multiprogramming-Umgebungen notwendig geworden ist, werden Benchmarks benötigt, die sowohl das Verhalten der parallelen Programme als auch das Systemverhalten untersuchen. Da es auf diesem Gebiet zur Zeit erst einige wenige Ansätze gibt, die jedoch keine umfassenden Untersuchungen ermöglichen, wurde das Benchmark-System *PAR-Bench* entwickelt. Bei diesem System handelt es sich um ein Benchmark-System, welches die oben beschriebenen Ansätze integriert und darüber hinaus deutlich erweitert. Als Basissystem für diesen Benchmark wurde ein am Zentralinstitut für Angewandte Mathematik des Forschungszentrums Jülich entwickeltes System verwendet [Nag90c]. Das Benchmark-System *PAR-Bench* wurde im Rahmen dieser Arbeit auf das Betriebssystem UNICOS portiert und dabei deutlich erweitert.

PAR-Bench erlaubt Messungen sowohl im Einbenutzerbetrieb als auch im Mehrbenutzerbetrieb für vorgebbare Arbeitslasten, die bezüglich der relevanten Parameter in weiten Bereichen eingestellt werden können. Es können sowohl synthetische als auch beliebige andere Benchmark-Programme, wie z.B. Anwendungsprogramme oder Programmkerne, in eine solche Arbeitslast eingebunden werden. Die synthetischen Benchmark-Programme werden vom System *PAR-Bench* automatisch generiert und sind innerhalb der Grenzen des jeweiligen Rechners bezüglich der Belastung der Systemressourcen weitgehend uneingeschränkt konfigurierbar.

Auf diese Weise kann allein durch die synthetischen Benchmark-Programme eine beliebige Arbeitslast nachgebildet werden, wobei die charakteristischen Daten dieser Arbeitslast als Vorgabe gemacht werden. Das System *PAR-Bench* erzeugt die synthetischen Benchmark-Programme einer Arbeitslast und speichert sie. Auf diese Weise

können die Programme identisch wiederholt werden, d.h. sie sind als jeweils identische Hintergrundlast für weitere Messungen mit zusätzlichen Programmen geeignet. Dabei werden sowohl die Programme der Hintergrundlast als auch die zusätzlichen Programme erzeugt, die unabhängig von der Hintergrundlast variiert werden können.

Insbesondere können bei *PAR-Bench* parallele Programme in eine Arbeitslast eingebunden werden. Grundsätzlich kann für jedes der bereits erzeugten synthetischen Benchmark-Programme die Anzahl der zu benutzenden Prozessoren vorgegeben werden, ohne jedoch die von den Programmen verrichtete Arbeit zu verändern. In Verbindung mit der Reproduzierbarkeit der Hintergrundlast besteht damit die Möglichkeit, in zwei separaten Messungen mit jeweils identischer Hintergrundlast dasselbe Programm einmal sequentiell mit einem Prozessor und dann parallel mit variabler Anzahl von Prozessoren auszuführen. Auf diese Weise kann die Leistung des parallelen Programms und damit der Einfluß der Multiprogramming-Umgebung auf diese Leistung bestimmt werden. Durch Vergleich der beiden Messungen kann auch der Einfluß des parallelen Programms auf die Hintergrundlast und damit auf den Systemdurchsatz ermittelt werden.

Damit die parallelen Programme realistisches Verhalten zeigen, kann für jedes der synthetischen Benchmark-Programme analog zur Anzahl der Prozessoren auch der Parallelitätsgrad und der Load-Balance-Faktor (siehe Kapitel 2) vorgegeben werden. Damit läßt sich nicht nur das Verhalten realer Programme nachbilden, sondern es kann auch der Einfluß dieser Faktoren auf die Leistung des parallelen Programms und auf den Systemdurchsatz ermittelt werden.

Die Ausführung von parallelen Programmen in einer Multiprogramming-Umgebung kann insbesondere bei einer Überbelastung des Hauptspeichers eine signifikante Verbesserung des Systemdurchsatzes bewirken [Bie88]. Um insbesondere diesen Aspekt zu untersuchen, kann auch der Speicherbedarf der einzelnen Programme nachträglich verändert werden, ohne die von den Programmen verrichtete Arbeit zu beeinflussen. Da auch die Priorität eines Programms innerhalb einer Multiprogramming-Umgebung sowohl einen Einfluß auf die Ausführungszeit des jeweiligen Programms als auch auf den Systemdurchsatz haben kann, wurde bei dem System *PAR-Bench* in analoger Weise auch die Einstellung dieses Parameters realisiert.

Aufgrund der Verwendung von synthetischen Benchmark-Programmen kann einerseits das charakteristische Verhalten realer Programme exakt nachgebildet werden. Andererseits wird mit den synthetischen Komponenten eine große Flexibilität gewährleistet:

Es können mit geringem Aufwand weitgehend beliebige Arbeitslasten realisiert werden. Dabei kann eine Arbeitslast für die Untersuchung von einzelnen Einflußparametern - insbesondere von denen, die die Parallelität betreffen - bezüglich dieser Parameter gezielt variiert werden, ohne die jeweils verrichtete Arbeit zu verändern.

Da zusätzlich zur weitgehenden Wahlfreiheit bezüglich der Arbeitslasten auch die Portabilität der Benchmarks gewährleistet ist, kann das Benchmark-System *PAR-Bench* als universelles System zum Vergleich von Rechnern mit weitgehend beliebigen anwenderspezifischen Arbeitslasten benutzt werden. Dabei werden die in diesem Kapitel gemachten Ansätze nicht nur integriert, sondern in bezug auf die aktuellen Strategien beim Betrieb von Supercomputern entscheidend erweitert. Da mit dem System *PAR-Bench*

alle Messungen in dedizierter Umgebung durchgeführt werden, ist die Reproduzierbarkeit der Ergebnisse weitgehend gewährleistet.

Die durch dieses System implementierten Meßmethoden eröffnen neue Möglichkeiten im Bereich des Benchmarking, und die Aussagekraft der Ergebnisse ist größer als die der bisher publizierten Resultate anderer Benchmark-Aktivitäten.

4. Das Benchmark-System PAR-Bench

Die Auswahl von Benchmark-Programmen hat einen starken Einfluß auf die Ergebnisse von Benchmarks. Um die Leistung von Rechnern im realen Betrieb zu ermitteln, muß ein Benchmark als vollständige Arbeitslast ausgeführt werden. Damit die Ergebnisse einen realen Bezug haben, sollten die Programme der aktuellen oder, je nach Motivation, der projizierten Arbeitslast einer Rechenzentrums Umgebung entsprechen. Aufgrund dieser Anforderungen wurde das Benchmark-System *PAR-Bench* entwickelt:

- Dieses System erlaubt die Modellierung weitgehend beliebiger Arbeitslasten und die Ausführung dieser Lasten als Benchmark.
- Die mit diesem System erstellten Benchmarks eignen sich insbesondere auch für die Untersuchung von ausgewählten Systemkomponenten wie zum Beispiel der Multitasking-Implementation oder des Job-Schedulers.

Bei derartigen Untersuchungen ist es notwendig, einzelne Testprogramme für verschiedene Messungen bezüglich einiger Parameter zu variieren. Dabei sind die Parameter charakteristisch, die die Arbeit der übrigen Prozessoren und damit die anderen Jobs im System beeinflussen. Zu dieser Gruppe von Parametern gehören der Speicherbedarf, die Speicherzugriffsrate und die Ein-/Ausgabehäufigkeit. Weitere Parameter wie die benötigte Prozessorzeit, die Priorität eines Jobs und die Gleitpunktverarbeitungsrate haben keinen unmittelbaren Einfluß auf die übrigen Jobs, die gerade von anderen Prozessoren bearbeitet werden. In Anwendungsprogrammen sind diese Parameter im allgemeinen nicht unabhängig voneinander, d.h. sie können nicht flexibel eingestellt werden. Der Speicherbedarf zum Beispiel kann über die Problemgröße verändert werden, die aber gleichzeitig die benötigte Prozessorzeit beeinflusst. Aus diesem Grund werden in dem hier vorgestellten System synthetische Benchmark-Programme benutzt, die eine weitgehend beliebige und unabhängige Einstellung dieser Parameter ermöglichen. Durch entsprechende Parametervorgaben kann dabei das Verhalten tatsächlicher Programme exakt nachgebildet werden.

Das Benchmark-System *PAR-Bench* ist bisher auf den Systemen Cray X-MP und Cray Y-MP für die Betriebssysteme COS und UNICOS (siehe Kapitel 2) implementiert. Die erzeugten Benchmarks können ohne größeren Aufwand auf andere Systeme portiert werden. Die folgende Beschreibung des Systems *PAR-Bench* bezieht sich jedoch ausschließlich auf das Betriebssystem UNICOS.

4.1. Das Funktionsprinzip von PAR-Bench

In dieser Arbeit sollen mittels *PAR-Bench* insbesondere die wechselseitigen Einflüsse von parallelen Programmen und Hintergrundlast untersucht werden. Darüber hinaus soll dieses System einen Vergleich verschiedener Rechner im Mehrbenutzerbetrieb ermöglichen. Deswegen wurden beim Entwurf des Systems *PAR-Bench* zwei unterschiedliche Zielsetzungen verfolgt:

- Um verschiedene Rechner im Mehrbenutzerbetrieb vergleichen zu können, ist es notwendig, alle Untersuchungen mit der jeweils gleichen Arbeitslast durchzuführen. Bei den Untersuchungen der wechselseitigen Einflüsse von parallelen Programmen

und Hintergrundlast sollen verschiedene Versionen von Testprogrammen mit identischer Hintergrundlast ausgeführt werden. Um sowohl die Reproduzierbarkeit der Arbeitslasten als auch die der Hintergrundlasten zu gewährleisten, dürfen nur solche Programme verwendet werden, die bereits vor der Ausführung in jeder Beziehung festgelegt sind, d.h. alle relevanten Parameter müssen bei wiederholter Ausführung unveränderte Werte aufweisen.

- Um den Einfluß der Vorgabeparameter zu ermitteln, werden Vergleichsmessungen durchgeführt, bei denen einzelne Programme bezüglich dieser Parameter variiert werden. Solche Vergleiche sind nur dann sinnvoll, wenn die von den Programmen verrichtete Arbeit bei allen Messungen gleich ist. Deswegen dürfen bei diesen Vergleichen nur solche Programme verwendet werden, die bezüglich der Arbeit festgelegt sind, unabhängig von der Variation der Vorgabeparameter.

Das System *PAR-Bench* erlaubt grundsätzlich die Einbeziehung beliebiger Programme in eine Arbeitslast. Eine wesentliche Möglichkeit besteht jedoch darin, synthetische Programme zu erzeugen, die den obigen Forderungen genügen. Zu diesem Zweck wurden im System *PAR-Bench* zwei Betriebsarten vorgesehen:

- Im Generate-Modus werden synthetische Benchmark-Programme erzeugt, die dann bezüglich der durchzuführenden Arbeit festgelegt sind.
- Im Execute-Modus werden diese Benchmark-Programme ausgeführt, wobei charakteristische Parameter variiert werden können, ohne die Arbeit zu verändern.

In Erweiterung dazu ist eine zur Initialisierung des Benchmark-Systems benötigte Funktion im Newdata-Modus realisiert (siehe Kapitel 4.1.4).

4.1.1. Der Generate-Modus

Im Generate-Modus werden die synthetischen Benchmark-Programme erzeugt. Jedem Benchmark-Programm ist ein korrespondierender Job zugeordnet. Zunächst wird nur die Generierung eines Benchmark-Programms betrachtet. Dabei werden Vorgaben bezüglich der Programmparameter gemacht, die folgendermaßen realisiert werden:

Die Parameter Priorität, Speicherbedarf und benötigte Prozessorzeit sind statisch, d.h. die entsprechenden Vorgaben werden einmal eingestellt und bleiben konstant. Sie werden zum Teil in der jeweiligen Version der JCL-Anweisungen (*Job Control Language*) oder innerhalb des Programms voreingestellt.

Im Gegensatz dazu hängen Parameter wie die Speicherzugriffsrate in MMREFS (*Million Memory REferences per Second*), die Ein-/Ausgabehäufigkeit in MIOS (*Million array elements transferred to or from an I/O device per Second*) und die Gleitpunktverarbeitungsrate in MFLOPS (*Million Floating-point Operations per Second*) stark von der jeweiligen Systembelastung ab. Zur Realisierung dieser Parameter werden während des Programmlaufs dynamische Entscheidungen getroffen, die deswegen als dynamische Parameter bezeichnet werden.

4.1.1.1. Die statischen Parameter

In den meisten Betriebssystemen hat die Priorität eines Jobs einen Einfluß auf die Zuweisung von Prozessorzeit an diesen Job. Dabei werden Jobs mit hoher Priorität bevorzugt, d.h. sie erhalten anteilig mehr Prozessorzeit als Jobs mit niedriger Priorität. Auch bei der Zuteilung des Speicherplatzes werden im allgemeinen Jobs mit hoher Priorität bevorzugt. Dies gilt zum Beispiel für das Betriebssystem UNICOS. Dort können die Jobs, die sich bereits im Speicher befinden, nur von anderen Jobs mit höherer oder gleicher Priorität aus dem Speicher verdrängt werden.

Auf dem in dieser Arbeit benutzten System⁸ haben alle Benutzerjobs Prioritäten zwischen 20 und 39⁹. Dabei entsprechen die kleineren Werte den höheren Prioritäten. Die Priorität eines Jobs wird bei *PAR-Bench* über den *nicevalue* eingestellt, der in einer JCL-Anweisung des Jobs angegeben wird. Dieser bewirkt eine Verschlechterung der Priorität des Jobs ausgehend von 20 um den *nicevalue*. Dadurch kann die relative Prioritätenverteilung innerhalb einer Arbeitslast beliebig vorgegeben werden.

Der Speicherbedarf eines Jobs wird zu Beginn des Benchmark-Programms eingestellt. Dabei kann eine Vorgabe nur dann realisiert werden, wenn der Mindestbedarf nicht unterschritten wird. Deswegen wird zunächst der aktuell zugeteilte Speicherplatz abgefragt. Wenn dieser größer ist als die Vorgabe, erfolgt die Ausführung des Programms ohne Änderung des aktuellen Wertes. Andernfalls kann dem Programm entsprechend der Vorgabe zusätzlicher Speicherplatz zugeteilt werden.

Um die benötigte Prozessorzeit des Programms einstellen zu können, wird die verbrauchte Prozessorzeit mit Hilfe des HPM (*Hardware Performance Monitor*, siehe Kapitel 2) gemessen. Sobald der Zeitvorgabewert für diesen Parameter erreicht ist, wird die Programmgenerierung beendet.

4.1.1.2. Die dynamischen Parameter

Die dynamischen Parameter werden durch eine Kombination von verschiedenen Programmkernen (*kernels*) realisiert. Dazu existiert ein Satz von mehr als 100 verschiedenen Programmkernen. Jeder dieser Kerne repräsentiert bezüglich der dynamischen Parameter einen Punkt $\mathbf{p} = (\text{MMREFS}, \text{MIOS}, \text{MFLOPS})$ in einem dreidimensionalen Parameterraum, die mit den Kernen korrespondierenden Kerndaten sind in einer Datei abgelegt (siehe Kapitel 4.1.4). Diese Datei dient als Eingabedatei für den Generate-Modus. Innerhalb der konvexen Hülle, die durch diese Punkte aufgespannt wird, kann durch eine geeignete Folge der Kerne jeder Punkt erreicht werden.

Im Generate-Modus wird je nach Vorgabe der dynamischen Parameter eine geeignete Kernfolge generiert und ausgeführt. Dabei wird zur Laufzeit dynamisch entschieden, welcher Kern als nächstes ausgeführt wird. Die aktuellen Werte der Parametergrößen werden zur Laufzeit mit Hilfe des HPM gemessen. Diese Werte ergeben den aktuellen Punkt $\mathbf{a}^{(j)}$ im Parameterraum nach der Ausführung von j Kernen. Außerdem liefert der HPM die bisher verbrauchte Prozessorzeit T . Aus diesen Meßwerten und aus den gespeicherten Kerndaten (die mit den Kernen K_i korrespondierenden Punkte $\mathbf{p}_i$ und die

⁸ Es wurde eine Cray Y-MP8/832 unter dem Betriebssystem UNICOS verwendet.

⁹ Die Prioritäten 0 bis 19 sind für Systemjobs vorgesehen.

Ausführungszeiten der Kerne t_i) wird dann für jeden Kern K_i ein hypothetischer Ergebnispunkt $\mathbf{h}_i^{(j+1)}$ berechnet, der nach hypothetischer Ausführung des Kernes K_i im Parameterraum erreicht würde:

$$\mathbf{h}_i^{(j+1)} = \frac{T \mathbf{a}^{(j)} + t_i \mathbf{p}_i}{T + t_i}$$

Dann wird aus allen Kernen ein Kern $K_i^{(j+1)}$ so ausgewählt, daß der euklidische Abstand¹⁰ $e_i^{(j+1)}$ des hypothetischen Punktes $\mathbf{h}_i^{(j+1)}$ vom vorgegebenen Zielpunkt $\mathbf{z}$ minimal wird.

$$e_i^{(j+1)} = \| \mathbf{z} - \mathbf{h}_i^{(j+1)} \|$$

Der Index i des so bestimmten Kernes $K_i^{(j+1)}$ wird als $(j+1)$ -te Entscheidung abgespeichert und der Kern wird ausgeführt. Dieser Vorgang wird solange wiederholt, bis die vorgegebene Prozessorzeit des Benchmark-Programms erreicht ist.

Das Struktogramm in Abb. 4 beschreibt die Arbeitsweise von *PAR-Bench* in der Betriebsart Generate.

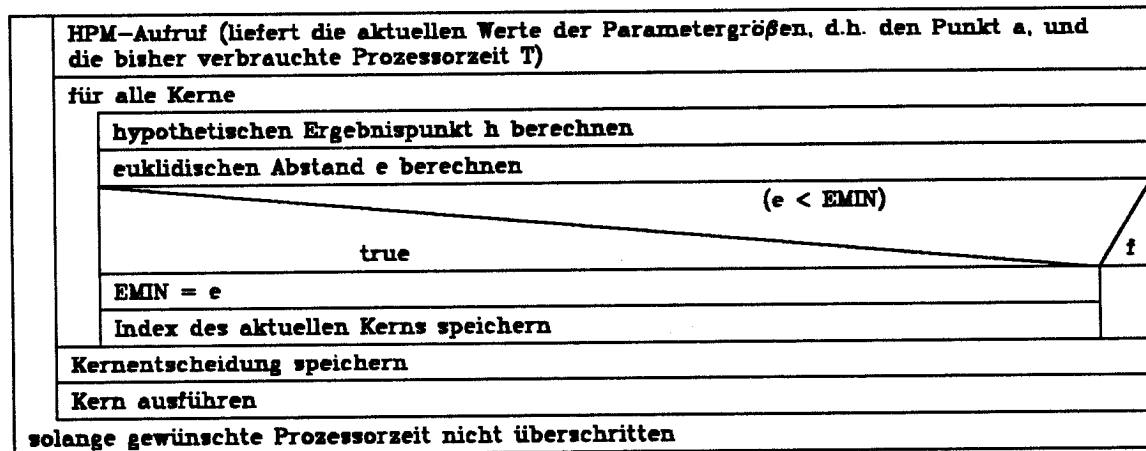


Abb. 4. Der Generate-Modus des Systems PAR-Bench

Die Folge der ausgeführten Kerne wird in einer Datei gespeichert. Diese Datei steuert die Ausführung des Benchmark-Programms; die durch diese Datei definierte Arbeit kann dann identisch wiederholt werden, da jedesmal dieselbe Folge von Kernen ausgeführt wird.

Bei wiederholter Generierung eines Programms mit jeweils identischen Vorgabeparametern werden aufgrund der komplexen Abhängigkeit des Zeitverhaltens von der Umge-

¹⁰ Für den euklidischen Abstand ist innerhalb einer Dimension die relative Abweichung vom Vorgabewert in bezug auf die Größe des Wertebereichs des entsprechenden Parameters relevant. Deswegen wurden alle Parameterwerte auf das Intervall (0,1) normiert.

bung¹¹ keine identischen Programme erzeugt, d.h. die jeweiligen Kernfolgen sind nicht identisch. Eine Realisierung der Vorgaben ist also durch verschiedene Kernfolgen möglich, d.h. die Abbildung einer Kernfolge auf die zugehörigen Vorgabeparameter ist nicht eindeutig. Um die Kernfolge und damit die verrichtete Arbeit eines Programms bei verschiedenen Messungen konstant zu halten, wird im Execute-Modus jeweils dieselbe Kernfolge benutzt.

4.1.2. Der Execute-Modus

Im Execute-Modus wird die im Generate-Modus erstellte Kernfolge als Benchmark-Programm ausgeführt, unabhängig von den aktuellen Werten der dynamischen Parameter. Aus diesem Grund ist eine Messung der aktuellen Werte im Execute-Modus nicht mehr notwendig, d.h. der HPM wird im Execute-Modus nicht mehr benötigt.

Die im Generate-Modus generierte Kernfolge wird aus einer Datei geladen. Die Entscheidung für den nächsten auszuführenden Kern muß nur gelesen werden, der jeweilige Kern wird dann ausgeführt. Dieser Vorgang wird solange wiederholt, bis alle Entscheidungen gelesen wurden. Das folgende Struktogramm beschreibt die Arbeitsweise von *PAR-Bench* im Execute-Modus.

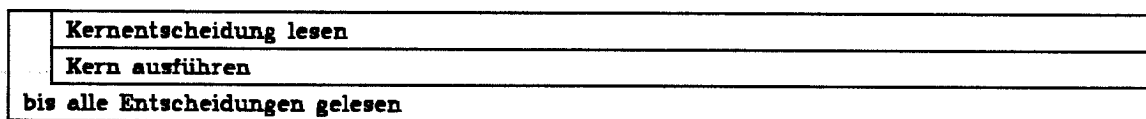


Abb. 5. Der Execute-Modus des Systems PAR-Bench

Aus den durchgeführten Messungen geht hervor, daß sich das Zeitverhalten eines Benchmark-Programms im Generate-Modus von dem im Execute-Modus nur um ca. 2 bis 5 Prozent unterscheidet. Dieser Unterschied wird durch die Aufrufe des HPM im Generate-Modus hervorgerufen. Da die Messung der Benchmark-Programme nur im Execute-Modus erfolgt, ist dieser Unterschied für die Aussagekraft der Ergebnisse ohne Bedeutung.

Aufgrund der dynamischen Parameter und der Prozessorzeit wurde im Generate-Modus die Folge der Programmkerne ermittelt. Diese Folge legt die Arbeit fest, die vom Programm verrichtet wird. Da diese Arbeit für alle Vergleichsmessungen mit dem jeweiligen Programm konstant sein muß, wird die Folge der Programmkerne nicht verändert. Deswegen sind für ein einmal generiertes Programm keine expliziten Änderungen bezüglich der dynamischen Parameter und der Prozessorzeit möglich.

Um den Einfluß aller weiteren charakteristischen Programmparameter eines so festgelegten Programms zu untersuchen, wurde eine explizite Einstellung dieser Parameter ermöglicht, ohne dabei die Folge der Programmkerne zu verändern. Diese Einstellung

¹¹ Im Generate-Modus können schon geringe Zeitunterschiede andere Kernentscheidungen bewirken.

wird erst nach der Generierung des Programms, d.h. im Execute-Modus, vorgenommen. Die betroffenen Parameter werden als "variable Parameter" bezeichnet:

- Der Speicherbedarf eines Programms kann bei einer Überbelastung des Hauptspeichers einen signifikanten Einfluß auf den Systemdurchsatz haben. In [Bie87, Bie88] wird zum Beispiel ein Benchmark beschrieben, in dem ein Programm mit hohem Speicherbedarf die Ausführung von anderen Jobs im System behindert, so daß ein Teil der Prozessoren ohne Arbeit ist, d.h. eine *idle*-Schleife ausführt. Desweiteren kann die Priorität eines Jobs sowohl die Ausführungszeit dieses Jobs als auch den Systemdurchsatz beeinflussen. Deswegen wurden diese beiden Parameter als variable Parameter realisiert, d.h. sie werden auch im Execute-Modus entsprechend Kapitel 4.1.1 wiederum statisch eingestellt.
- Ein wesentliches Ziel dieser Arbeit ist die Bereitstellung eines Systems zur Untersuchung des Einflusses von parallelen Programmen in einer Multiprogramming-Umgebung. Zu diesem Zweck können die im Generate-Modus sequentiell erzeugten Programme durch Nutzung von Multitasking-Primitiven parallel ausgeführt werden. Die Art und Weise der Parallelisierung wird durch die Multitasking-Parameter des Systems *PAR-Bench* festgelegt. Es ist möglich den Parallelitätsgrad, den Load-Balance-Faktor und die Anzahl der anzufordernden Prozessoren als variable Parameter für die Ausführung zu spezifizieren. Auf diese Weise können die erzeugten Programme in Abhängigkeit von den Multitasking-Parametern sowohl sequentiell als auch parallel ausgeführt werden, wobei die verrichtete Arbeit unabhängig von den Multitasking-Parametern jeweils unverändert bleibt. Da für die Realisierung der Multitasking-Parameter einige Vorüberlegungen notwendig sind, wird diese in Kapitel 4.2.2 getrennt beschrieben.

Die vollständige Bestimmung eines Programms ist also vor der Ausführung zum einen durch die Folge der Programmkerne und zum anderen durch die Spezifikation der variablen Parameter gegeben.

4.1.3. Die Simulation einer vollständigen Arbeitslast

Um bei der Simulation einer vollständigen Arbeitslast aussagekräftige und reproduzierbare Ergebnisse zu erhalten, werden Generate-Modus und zugehöriger Execute-Modus auf einer dedizierten Maschine ausgeführt. Da die Generierung und damit die Messung und Einstellung der Benchmark-Parameter (Generate-Modus) nicht zur Zeit der Ausführung des Benchmarks (Execute-Modus) erfolgt, ist nicht unbedingt gewährleistet, daß die gemessenen Werte der Parameter mit den tatsächlich relevanten Werten bei der Programmausführung übereinstimmen. Auch bei Ausführung von identischen Programmen im Generate-Modus bzw. im Execute-Modus kann durch den Einfluß der jeweiligen Umgebung das Zeitverhalten signifikant unterschiedlich sein. Um das zu vermeiden, müssen die korrespondierenden Programme in der jeweils gleichen Umgebung ausgeführt werden. Deswegen werden alle Benchmark-Programme eines Benchmarks jeweils nach identischem Zeitplan gestartet. Auf diese Weise ist gewährleistet, daß die Benchmark-Programme im Generate-Modus in einer Umgebung generiert und ausgeführt werden, die zumindest ähnlich auch in den verschiedenen Läufen im Execute-Modus herrscht. Dadurch ist der Einfluß der Umgebung auf die einzelnen Bench-

mark-Programme einer Arbeitslast und damit das Zeitverhalten in beiden Betriebsarten im wesentlichen gleich.

Zum Vergleich verschiedener Rechner werden die Programme einer Arbeitslast auf einem Rechner generiert, und die ermittelten Programmkernfolgen werden mit unveränderten Parametervorgaben auch für die Messungen auf den anderen Rechnern benutzt. Für weitere Untersuchungen bietet das System *PAR-Bench* die folgenden Möglichkeiten:

- Die Änderung einer Arbeitslast für verschiedene Messungen bezüglich der variablen Parameter, wobei die verrichtete Arbeit konstant bleibt. Dabei können die variablen Parameter der gesamten Arbeitslast variiert werden.
- Die Änderung von einzelnen Programmen innerhalb einer Arbeitslast bezüglich der variablen Parameter. Diese Möglichkeit wird insbesondere bei der Untersuchung der wechselseitigen Einflüsse von parallelem Programm und Hintergrundlast genutzt. Dabei ist entscheidend, daß nur die parallelen Programme verändert werden, während die Hintergrundlast bei allen Messungen identisch ist.
- Das Einfügen von zusätzlichen Testprogrammen in eine Arbeitslast.

Durch diese Möglichkeiten kann das Benchmark-System *PAR-Bench* bei einer Vielzahl von verschiedenen Untersuchungen als flexibles Werkzeug eingesetzt werden.

4.1.4. Der Newdata-Modus

Im Newdata-Modus werden die mit den Programmkernen korrespondierenden Punkte im Parameterraum und die Ausführungszeiten der Kerne ermittelt und als Eingabedaten für den Generate-Modus in einer Datei abgelegt. Dazu werden die Kerne ausgeführt, und die Werte der dynamischen Parameter und die Ausführungszeiten werden mit Hilfe des HPM (*Hardware Performance Monitor*) gemessen.

Der Newdata-Modus ist insbesondere auch für die Einbeziehung zusätzlicher Programmkerne in das Benchmark-System geeignet, d.h. in diesem Fall wird ein Programmlauf im Newdata-Modus durchgeführt, der die neuen Kerndaten ermittelt.

4.2. Die Realisierung von PAR-Bench

In den drei Betriebsarten des Benchmark-Systems *PAR-Bench*, d.h. im Generate-, Execute- und Newdata-Modus, werden jeweils gleiche Daten- und Programmstrukturen verwendet. Um Unterschiede bezüglich dieser Strukturen und damit Fehler bei der Programmerstellung zu vermeiden, wurden alle drei Betriebsarten in einem Programm realisiert. Dabei wird die jeweilige Betriebsart aufgrund eines Eingabeparameters bestimmt. Dieses Kapitel beschreibt die wesentlichen Realisierungen für die Erzeugung eines Benchmark-Programms im Generate-Modus, für die Parallelisierung eines Benchmark-Programms im Execute-Modus und die Ermittlung der Kerndaten im Newdata-Modus. Dabei werden Beispiele für die jeweils relevanten Daten angegeben.

4.2.1. Die Erzeugung eines Benchmark-Programms

Die synthetischen Benchmark-Programme von *PAR-Bench* bestehen aus Programmkernen, die sich bezüglich der dynamischen Parameter Speicherzugriffsrate, Ein-/Ausgabehäufigkeit und Gleitpunktverarbeitungsrate und der Ausführungszeit unterscheiden. Dabei werden die Vorgaben für diese Parameter durch die Verwendung einer Kombination von verschiedenen Programmkernen realisiert. Jeder dieser Programmkerne repräsentiert bezüglich der dynamischen Parameter einen Punkt innerhalb eines dreidimensionalen Parameterraumes. Diese Punkte spannen eine konvexe Hülle auf, die das Parametergebiet von *PAR-Bench* begrenzt. Innerhalb dieses Gebietes kann durch eine geeignete Folge von Programmkernen jeder Punkt erreicht werden. Das Parametergebiet liegt innerhalb eines quaderförmigen maschinenspezifischen Parameterraumes, wobei die Eckpunkte des Quaders durch den Koordinatenursprung und die Maximalwerte der drei Dimensionen bestimmt werden. Die Maximalwerte hängen von der Architektur und der Taktzeit des jeweiligen Rechners ab.

Die dynamischen Parameter können also innerhalb eines Parametergebietes gewählt werden, das wiederum durch den maschinenspezifischen Parameterraum begrenzt wird. Im folgenden wird als Beispiel der Parameterraum einer Cray Y-MP8/832 beschrieben. Aufgrund der prinzipiellen Eigenschaften eines Parameterraumes werden die Programmkerne erstellt, die das Parametergebiet von *PAR-Bench* definieren. Die Erzeugung eines Benchmark-Programms in der Form einer Folge von Programmkernen wird durch ein Beispiel verdeutlicht, in dem die einzelnen Iterationsschritte der Erzeugung dargestellt sind.

4.2.1.1. Der Parameterraum einer Cray Y-MP8/832

Um die Eigenschaften eines Parameterraumes zu untersuchen, wird als Beispiel der Parameterraum einer Cray Y-MP8/832 betrachtet. Dabei berechnen sich die Maximalwerte¹² für die drei Dimensionen des Parameterraumes folgendermaßen:

- **Speicherzugriffsrate:**

Für Speicherzugriffe stehen jedem Prozessor drei *ports* zur Verfügung, zwei zum Laden und einer zum Speichern. Über jeden *port* kann pro Maschinentakt ein Speicherzugriff (MREF) durchgeführt werden. Damit ergibt sich die maximale Speicherzugriffsrate in MMREFS (*Million Memory REferences per Second*)¹³:

$$MM_{\max} = \frac{\text{Anzahl der Ports} \times 1 \text{ MREF}}{\text{Taktzeit}} = 500 \text{ MMREFS}$$

- **Ein-/Ausgabehäufigkeit:**

Für die Ein-/Ausgabe existiert ein spezieller *I/O-port* je Prozessor, der pro Maschinentakt eine Ein- oder Ausgabe (IO) durchführen kann. Daraus errechnet sich die maximale Ein-/Ausgabehäufigkeit in MIOS (*Million elements transferred to or from an I/O device per Second*):

¹² Diese Werte gelten jeweils für einen Prozessor.

¹³ Die Taktzeit der Cray Y-MP8/832 beträgt 6 ns.

$$IO_{\max} = \frac{\text{Anzahl der Ports} \times 1 IO}{\text{Taktzeit}} = 167 MIOS$$

- **Gleitpunktverarbeitungsrate:**

Die maximale Gleitpunktverarbeitungsrate wird durch drei Vektorfunktionseinheiten (Additions-, Multiplikations- und Reziprokeinheit) bestimmt, die pro Maschinentakt jeweils eine Gleitpunktoperation (FLOP) ausführen können. Es ergibt sich die maximale Gleitpunktverarbeitungsrate in MFLOPS (*Million Floating-point Operations per Second*):

$$FL_{\max} = \frac{\text{Anzahl der Funktionseinheiten} \times 1 FLOP}{\text{Taktzeit}} = 500 MFLOPS$$

In Abb. 6 ist der daraus resultierende Parameterraum der Cray Y-MP8/832 dargestellt.

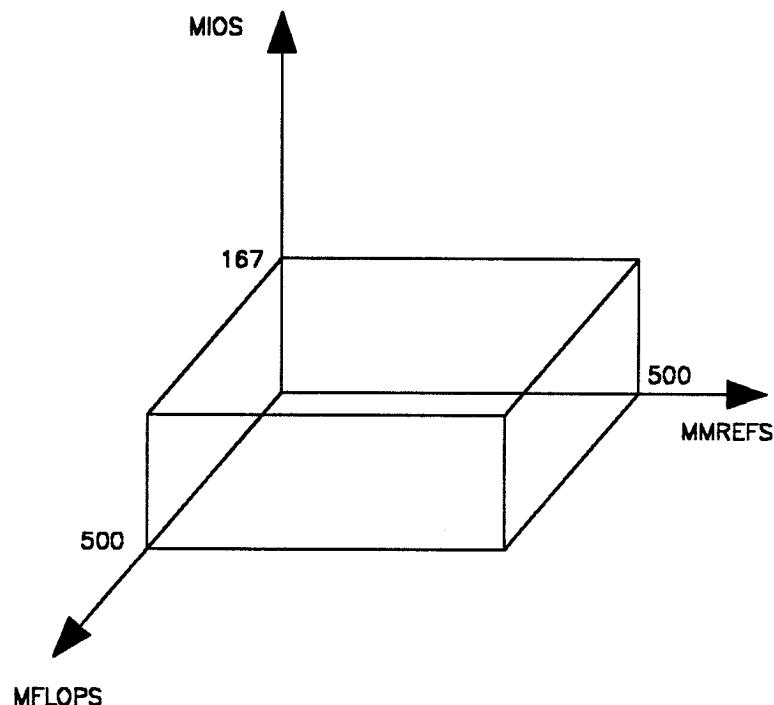


Abb. 6. Der Parameterraum der Cray Y-MP8/832

Die Maximalwerte geben theoretische Spitzenleistungen an, die praktisch nicht erreicht werden können. Sie stellen deshalb nur Obergrenzen für den tatsächlich nutzbaren Teil des maschinenspezifischen Parameterraums dar, der durch die Programmkerne bestimmt wird.

4.2.1.2. Die Programmkerne und das Parametergebiet

Für die Erstellung der Programmkerne wurden vier Forderungen aufgestellt:

1. Die Benchmark-Programme sollen portabel sein, deshalb sollen die Kerne in der Programmiersprache FORTRAN erstellt werden.

2. Da für die Realisierung weitgehend beliebiger Parameterwerte unter Umständen eine Kombination von sehr vielen Programmkernen benötigt wird, soll die Ausführungszeit der einzelnen Kerne nicht zu groß sein.
3. Um den durch das Abspeichern bzw. das Einlesen von Kernentscheidungen erzeugten Overhead klein zu halten, soll die Anzahl der Entscheidungen pro Zeiteinheit nicht zu groß bzw. die Ausführungszeit der Kerne nicht zu klein sein.
4. Die von den Programmkernen aufgespannte konvexe Hülle soll den maschinenspezifischen Parameterraum möglichst gut ausfüllen.

Durch die Verwendung von FORTRAN Programmen wird die Forderung nach guter Ausnutzung des maschinenspezifischen Parameterraumes eingeschränkt, da dieser durch Assembler-Kerne besser ausgenutzt werden kann. Die beiden Forderungen in bezug auf die Ausführungszeit ergeben eine Ober- bzw. Untergrenze für die Ausführungszeit. Die Schleifeninitialisierung innerhalb der Programmkerne bewirkt eine Leistungsverringering, die umso größer wird, je kürzer die Ausführungszeit der Kerne ist. Deswegen stellt die Obergrenze für die Ausführungszeit eine weitere Einschränkung bezüglich der Ausnutzung des Parameterraumes dar.

Um den quaderförmigen Parameterraum möglichst gut auszunutzen, wurden Programmkerne entwickelt, deren korrespondierende Punkte möglichst nahe bei den Kanten bzw. bei den Eckpunkten des Parameterraumes liegen. Zunächst wird nur die Ebene $IO=0$ des Parametergebietes betrachtet, die in Abb. 7 dargestellt ist.

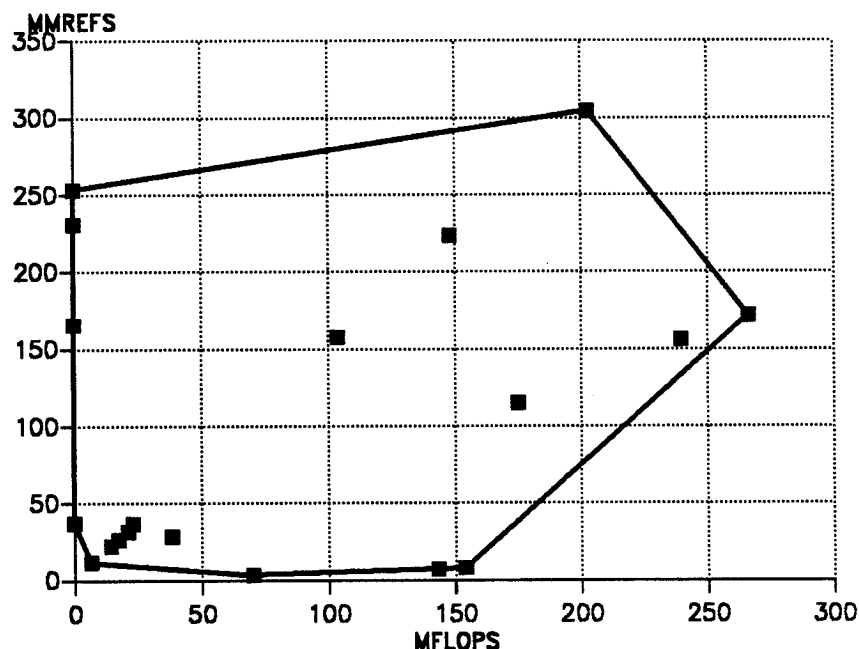


Abb. 7. Die Ebene $IO = 0$ des Parametergebietes

Die Punkte auf der MMREFS-Achse werden zum Beispiel durch einen Programmkern C realisiert, der nahezu ausschließlich Speicherzugriffe durchführt. Dieser Kern besteht aus den folgenden Schleifen:

```

DO 500 ITER=1,NITER
  DO 400 IPROC=1,CHUNKS
    DO 300 REST=1,(840/CHUNKS)
      DO 200 I=1,IKERN
        DO 100 LAUF=1,LENGTH
          FELDB(LAUF,IPROC)          = FELDC(LAUF,1)
          FELDB(LAUF,IPROC+CHUNKS)   = FELDC(LAUF,2)
          FELDB(LAUF,IPROC+2*CHUNKS) = FELDC(LAUF,3)
          FELDB(LAUF,IPROC+3*CHUNKS) = FELDC(LAUF,4)
          FELDB(LAUF,IPROC+4*CHUNKS) = FELDC(LAUF,5)
          FELDB(LAUF,IPROC+5*CHUNKS) = FELDC(LAUF,6)
          FELDB(LAUF,IPROC+6*CHUNKS) = FELDC(LAUF,7)
100      CONTINUE
200      CONTINUE
300      CONTINUE
400      CONTINUE
500      CONTINUE

```

Abb. 8. Die Schleifen des Programmkerns C

Da innerhalb der Schleife 100 keine Datenabhängigkeiten bestehen, wird diese vektorisiert. Dabei wird mit der Variablen 'LENGTH' die Vektorlänge vorgegeben. Da die Speicherzugriffsrate von der Vektorlänge abhängig ist, können mit verschiedenen Vektorlängen verschiedene Punkte auf der MMREFS-Achse realisiert werden. Die vier Punkte auf der Achse entsprechen den Vektorlängen 5, 32, 64 und 448, wobei die größte Vektorlänge mit der größten Speicherzugriffsrate korrespondiert. Die beiden mittleren Punkte vergrößern das Parametergebiet nicht, sie verbessern aber die Iterationsgeschwindigkeit bei der Ermittlung einer Kernfolge.

Die Schleife 200 soll die Ausführungszeit des Kerns vergrößern, die Variable 'IKERN' hat den Wert 50. Die beiden Schleifen 300 und 400 dienen der Parallelisierung der Kerne und werden in Kapitel 4.2.2 erläutert. Im sequentiellen Fall hat die Variable 'CHUNKS' den konstanten Wert '4'.

Um in Programmen mit großer Ausführungszeit die Anzahl der Kernentscheidungen zu verringern, wird mit der Schleife 500 die Ausführungszeit der einzelnen Kerne verlängert. Dabei wird die Variable 'NITER' proportional zur Anzahl der bereits getroffenen Kernentscheidungen erhöht.

Vier weitere Kerne sind analog aufgebaut, d.h. mit jeweils vier Versionen für verschiedene Vektorlängen ergeben sich die insgesamt zwanzig Punkte in Abb. 7.

Die Erweiterung dieser fünf Programmkerne auf die Dimension IO wird über Ein-/Ausgabeschleifen realisiert, die in die äußere Schleife (Schleife 400 in Abb. 8) des jeweiligen Kernes eingeschlossen sind. Die Ein-/Ausgabeschleifen bestehen aus Anweisungen für formatiertes und unformatiertes Lesen bzw. Schreiben, wobei in impliziten oder expliziten Schleifen jeweils Felder von Gleitpunktzahlen transferiert werden. Ana-

log zur Vektorlänge wird über eine Variable gesteuert, ob und wieviele Elemente ein- bzw. ausgegeben werden sollen. Für jeden der fünf Programmkerne können fünf verschiedene Ein-/Ausgabeverversionen ausgewählt werden, von denen jeweils eine keine Ein-/Ausgaben durchführt, so daß die korrespondierenden Punkte in der Ebene $IO=0$ liegen. Die vollständigen Schleifen des Kernes C sind in Abb. 9 dargestellt.

```

DO 500 ITER=1,NITER

    (Eingabeschleife)

    DO 400 IPROC=1,CHUNKS
        DO 300 REST=1,(840/CHUNKS)
            DO 200 I=1,IKERN
                DO 100 LAUF=1,LENGTH
                    FELDB(LAUF,IPROC)          = FELDC(LAUF,1)
                    FELDB(LAUF,IPROC+CHUNKS)    = FELDC(LAUF,2)
                    FELDB(LAUF,IPROC+2*CHUNKS)  = FELDC(LAUF,3)
                    FELDB(LAUF,IPROC+3*CHUNKS)  = FELDC(LAUF,4)
                    FELDB(LAUF,IPROC+4*CHUNKS)  = FELDC(LAUF,5)
                    FELDB(LAUF,IPROC+5*CHUNKS)  = FELDC(LAUF,6)
                    FELDB(LAUF,IPROC+6*CHUNKS)  = FELDC(LAUF,7)
                100      CONTINUE
            200      CONTINUE
        300      CONTINUE
    400      CONTINUE

    (Ausgabeschleife)

500  CONTINUE

```

Abb. 9. Die vollständigen Schleifen des Programmkerns C

Bei vier Versionen für verschiedene Vektorlängen mit jeweils fünf verschiedenen Ein-/Ausgabeverversionen ergeben sich je Programmkern zwanzig verschiedene Versionen. Die bisher fünf Programmkerne mit jeweils zwanzig Versionen werden durch einen Kern ergänzt, der nur Ein-/Ausgabeoperationen durchführt. Dieser besteht ausschließlich aus den Ein-/Ausgabeschleifen des Kernes C und kann in vier verschiedenen Versionen ausgeführt werden¹⁴.

Die einzelnen Benchmark-Programme werden durch die Erzeugung einer Folge dieser Programmkerne realisiert. Dabei werden die Vorgaben für die dynamischen Parameter iterativ angenähert. Die Untersuchungen haben gezeigt, daß Vorgabewerte, die innerhalb der konvexen Hülle liegen, mit einer relativen Abweichung von ca. 1 Prozent realisiert werden können. Ein Vorgabepunkt außerhalb der konvexen Hülle wird durch den

¹⁴ Die Ausführung dieses Kernes mit verschiedenen Vektorlängen ist nicht sinnvoll, da die Ein-/Ausgabehäufigkeit unabhängig von der Vektorlänge ist. Da weiterhin die Version ohne Ein-/Ausgabe wegfällt, bleiben noch vier Versionen übrig.

Punkt innerhalb der konvexen Hülle approximiert, dessen euklidischer Abstand vom Vorgabepunkt am kleinsten ist.

In Abb. 10 sind für ein Beispielprogramm die Vorgaben, die einzelnen Iterationsschritte und die Ergebnisse auszugsweise dargestellt. Die angegebenen Werte sind mit Hilfe des HPM ermittelt worden. Dieser wird bei Programmbeginn initialisiert und liefert bei jedem Aufruf die für die gesamte bereits verrichtete Arbeit relevanten Verarbeitungsraten. In diesem Beispiel werden in 50 Sekunden Prozessorzeit 2081 Entscheidungen getroffen.

	#	Kern	MMREFS	MFLOPS	MIOS	Zeit
Vorgabe			250.00	200.00	0.00	50.00000
	1	1	289.21	190.87	0.00	0.01578
	2	6	258.58	192.59	0.00	0.01955
	3	6	239.82	195.38	0.00	0.02313
	4	2	233.34	186.38	0.00	0.02656
	5	1	244.37	181.52	0.00	0.04386
	6	1	247.77	178.41	0.00	0.06150
	7	5	222.38	193.84	0.00	0.08379
	8	1	230.01	191.26	0.00	0.10067
	9	1	237.18	190.81	0.00	0.11669
	10	1	245.16	192.47	0.00	0.13133
	...					
	95	5	248.57	199.52	0.00	1.30080
	96	1	249.24	199.59	0.00	1.31542
	97	1	249.91	199.67	0.00	1.33000
	...					
	611	1	250.04	199.97	0.00	13.94126
	612	6	249.98	200.00	0.00	13.95072
	613	2	249.96	199.96	0.00	13.95942
	614	2	249.94	199.93	0.00	13.96820
	615	1	250.13	199.95	0.00	14.01228
	616	6	250.06	199.97	0.00	14.02174
	...					
	2079	7	249.98	199.99	0.00	49.96051
	2080	2	249.97	199.97	0.00	49.98043
	2081	6	249.93	199.99	0.00	50.00217
Ergebnis			249.93	199.99	0.00	50.00217

Abb. 10. Die Erzeugung einer Folge von Programmkernen

In der Tabelle ist abzulesen, daß Vorgabe und Ergebnis sehr gut übereinstimmen. Dabei sind die Vorgabewerte bereits nach wenigen Sekunden gut approximiert.

Die Parallelisierung der Programmkerne bezieht sich nur auf die inneren Schleifen, jedoch nicht auf die Ein-/Ausgabeoperationen. Deswegen können die Multitasking-Parameter nur dann realisiert werden, wenn die betroffenen Programmkerne keine Ein-/Ausgabe durchführen. Insbesondere für die Programme, die parallelisiert werden sollen, wird deswegen die Ein-/Ausgabehäufigkeit IO mit '0' vorgegeben. Trotz dieser Vorgabe können Kerne mit $IO \neq 0$ ausgewählt werden, wenn sich für diese der jeweils kleinste euklidische Abstand ergibt (siehe Kapitel 4.1.1). Um das zu verhindern, wurde ein Bewertungsfaktor eingeführt, der für die Vorgabe $IO = 0$ den euklidischen Abstand

bei Abweichungen in der Dimension IO künstlich vergrößert. Dadurch erklärt sich die strikte Einhaltung der Vorgabe für IO in Abb. 10. In Abb. 11 ist das Programmverhalten für die ersten 20 Sekunden graphisch dargestellt.

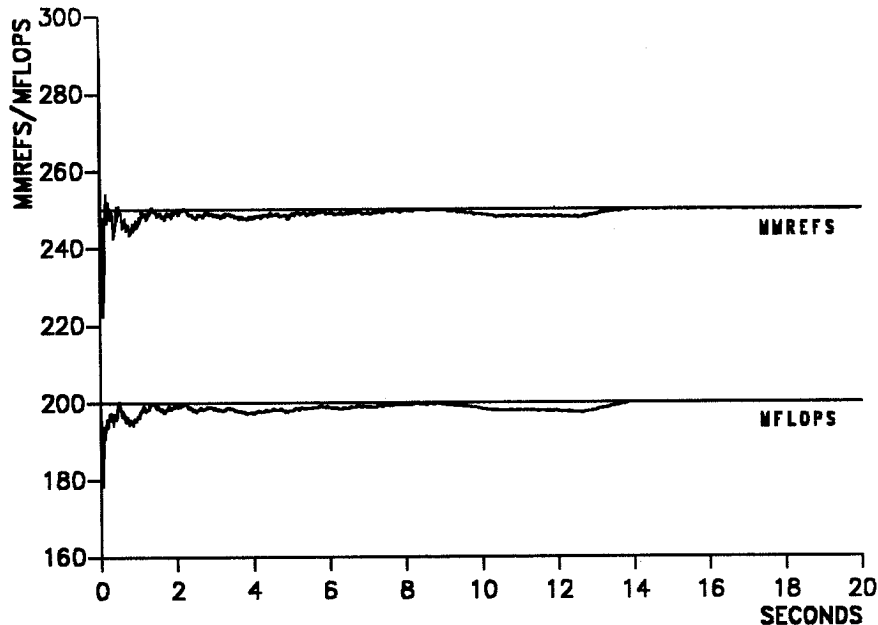


Abb. 11. Die Iteration der Vorgaben (MMREFS,MFLOPS)

In diesem Beispiel sind nach 14 Sekunden keine wesentlichen Abweichungen mehr vorhanden. Da der Einfluß eines einzelnen Programmkerns auf die Parameterwerte mit zunehmender Zeit abnimmt, werden die möglichen Abweichungen aufgrund eines Programmkerns mit zunehmender Zeit kleiner. Deswegen treten auch im weiteren Verlauf der Kurven keine größeren Abweichungen auf.

4.2.2. Die Parallelisierung eines Benchmark-Programms

Die Parallelisierung eines Benchmark-Programms wird durch die Multitasking-Parameter spezifiziert. Diese gehören zu den variablen Parametern, d.h. es können im Execute-Modus bei wiederholter Ausführung einer Kernfolge jeweils unterschiedliche Vorgaben gemacht werden, ohne die verrichtete Arbeit zu verändern.

Die Multitasking-Parameter

In *PAR-Bench* ist die Parallelisierung der Programme durch Cray Multitasking (siehe Kapitel 2) realisiert. Dabei wurde sowohl Cray Microtasking als auch Autotasking implementiert. Diese beiden Multitasking-Strategien wurden ausgewählt, da sie im Ver-

gleich zu Cray Macrotasking portabel¹⁵ und einfach zu integrieren sind, darüber hinaus führen sie zu einem niedrigeren Overhead. Aufgrund dieser Wahl ergeben sich die relevanten Multitasking-Parameter, der Parallelitätsgrad PG , der Load-Balance-Faktor LB und die Anzahl der anzufordernden Prozessoren $NPROC$.

Der Parameter $NPROC$ gibt die Anzahl der maximal vom Programm parallel nutzbaren Prozessoren an. Diese maximale Anzahl steht dem Programm im allgemeinen nicht ständig zur Verfügung¹⁶. Der Job-Scheduler kann einem Programm zum Beispiel einen Teil der Prozessoren entziehen und zu einem späteren Zeitpunkt wieder zuteilen. In der Zwischenzeit setzt das Programm seine Ausführung mit der reduzierten Anzahl von Prozessoren fort. Auf diese Weise ist die Anzahl der tatsächlich verfügbaren Prozessoren für keinen Zeitpunkt vorhersagbar. Für Cray Microtasking und Autotasking wird dieser Parameter in die Direktive 'CMICS GETCPUS NPROC' eingesetzt [Cra0222, Cra2088].

Um das reale Verhalten eines beliebigen parallelen Programms nachzubilden, sollten der Parallelitätsgrad und der Load-Balance-Faktor beliebig vorgebbar sein. Die beiden implementierten Multitasking-Strategien unterstützen die Parallelisierung auf Anweisungsebene. Dabei wird zum Beispiel eine DO-Schleife gleichzeitig von mehreren Prozessoren abgearbeitet, d.h. ein Prozessor bearbeitet jeweils einen *chunk*, das ist eine feste Anzahl von Schleifendurchläufen der parallelisierten DO-Schleife.

Der Parallelitätsgrad eines Programms wird definiert als der Anteil an der Ausführungszeit bei sequentieller Ausführung, in der die parallelisierbaren Programmteile ausgeführt werden. Bei *PAR-Bench* wird der Parallelitätsgrad auf Programmebene eingestellt, d.h. ein vorgegebener Wert wird durch eine Kombination von Programmkernen mit jeweils anderem Parallelitätsgrad realisiert. Dazu werden die Programmkerne entweder sequentiell oder zu 100% parallel ausgeführt, d.h. es existieren jeweils zwei Versionen von jedem Kern. Mit

$PTIME$ = Ausführungszeit der parallelen Kerne des Programms bei sequentieller Ausführung

und

$TTIME$ = Ausführungszeit aller Kerne des Programms bei sequentieller Ausführung

ergibt sich der Parallelitätsgrad des Programms:

$$PG = \frac{PTIME}{TTIME}$$

Nach der Ausführung von j Programmkernen berechnet sich der Parallelitätsgrad PG ⁶⁾ im Einzelnen aus

$KTIME(i)$ = Ausführungszeit des Kernes i bei sequentieller Ausführung,

$SKERN(i)$ ⁶⁾ = Anzahl der sequentiellen Kerne i

¹⁵ Die Portabilität wird insofern gewährleistet, als alle für die Parallelisierung notwendigen Erweiterungen des Programmtextes von den FORTRAN-Compilern als Kommentar überlesen werden.

¹⁶ Dies gilt insbesondere in Multiprogramming-Umgebungen.

und

$PKERN(i)^{(j)} = \text{Anzahl der parallelen Kerne } i$

$$PG^{(j)} = \frac{PTIME^{(j)}}{TTIME^{(j)}} = \frac{\sum_i (PKERN(i)^{(j)} \times KTIME(i))}{\sum_i (PKERN(i)^{(j)} \times KTIME(i)) + \sum_i (SKERN(i)^{(j)} \times KTIME(i))}$$

Der so realisierte Parallelitätsgrad charakterisiert die Eigenschaften eines parallelen Programms nur zum Teil. Der parallele Anteil eines Programms ist bei realen Programmen häufig unbalanciert, d.h. die *chunks* können nicht gleichmäßig auf die verfügbaren Prozessoren verteilt werden. Wenn z.B. für 27 *chunks* einer Schleife 8 Prozessoren zur Verfügung stehen, dann werden in drei Verarbeitungsschritten 24 *chunks* und im vierten Schritt nur 3 *chunks* bearbeitet¹⁷. Bei einem Parallelitätsgrad von 100% ergibt sich in diesem Beispiel bei Nutzung von 8 Prozessoren nur ein maximaler Speedup (Geschwindigkeitszuwachs) von 6.75. Der erzielte Speedup kann also unabhängig vom Parallelitätsgrad verringert werden [Esc89].

Als Maß für diese Verringerung wird in *PAR-Bench* der Load-Balance-Faktor *LB* eingeführt. Der Load-Balance-Faktor eines Programmkerns wird definiert als Quotient aus genutztem Prozessorpotential und vorhandenem Prozessorpotential. Wenn jeder *chunk* des Kerns die gleiche Prozessorzeit benötigt, dann entspricht *CHUNKS*, die Anzahl der *chunks* eines Kerns, dem genutzten Prozessorpotential. Mit der Anzahl der Prozessoren *NPROC* ergibt sich der Load-Balance-Faktor *LB* eines Programmkerns:

$$LB = \frac{CHUNKS}{STEPS \times NPROC}$$

mit der Anzahl der Verarbeitungsschritte

$$STEPS = \lceil \frac{CHUNKS}{NPROC} \rceil$$

Der Load-Balance-Faktor wird also bei vorgegebener Prozessorzahl durch die Anzahl der *chunks* bestimmt. Um diesen Faktor zu steuern, muß die Variable *CHUNKS* in der parallelen Schleife des Programmkerns durch das System *PAR-Bench* entsprechend vorgegeben werden:

```
DO 400 IPROC=1,CHUNKS
```

(innere Schleifen des Programmkerns)

```
400 CONTINUE
```

¹⁷ Dabei wird davon ausgegangen, daß ständig alle 8 Prozessoren zur Verfügung stehen.

Damit die vom Kern verrichtete Arbeit unabhängig vom Wert der Variablen *CHUNKS* wird, wird eine innere Schleife programmiert, deren Endparameter umgekehrt proportional zur Variablen *CHUNKS* ist:

```

DO 400 IPROC=1,CHUNKS
  DO 300 REST=1,(MAX/CHUNKS)

      (innere Schleifen des Programmkerns)

300  CONTINUE
400  CONTINUE

```

Dabei muß der Wert von *MAX* den folgenden Forderungen genügen:

1. Damit die vom Kern verrichtete Arbeit konstant bleibt, muß auch der Wert von *MAX* konstant sein.
2. Da die Ausführungszeit der Kerne nicht zu groß werden darf (siehe Kapitel 4.2.1), muß auch der Wert von *MAX* nach oben begrenzt werden.
3. Für alle gültigen Werte von *CHUNKS* muß der Quotient $MAX/CHUNKS$ ganzzahlig sein.

Durch diese Forderungen wird der Wertebereich der Variablen *CHUNKS* und damit auch der des Load-Balance-Faktors eines Programmkerns stark eingeschränkt. Deswegen wird der Load-Balance-Faktor analog zum Parallelitätsgrad auf Programmebene eingestellt, d.h. ein vorgegebener Wert wird durch eine Kombination von Programmkernen mit jeweils anderem Load-Balance-Faktor realisiert.

Um für eine beliebige Anzahl von Prozessoren eine zu 100% balancierte Ausführung der Kerne zu ermöglichen, muß der Wert der Variablen *CHUNKS* mindestens so groß sein wie die jeweilige Prozessorzahl *NPROC*. Im System *PAR-Bench* besteht diese Möglichkeit für 1 bis 8 Prozessoren. Mit diesem Wertebereich für die Variable *CHUNKS* ($1 \leq CHUNKS \leq NPROC$ mit $NPROC \leq 8$) ergibt sich entsprechend den Forderungen der Mindestwert für *MAX* so, daß *MAX* durch alle Zahlen von 1 bis 8 ganzzahlig teilbar ist:

$$MAX = 1 \times 2 \times 2 \times 2 \times 3 \times 5 \times 7 = 840$$

Mit $1 \leq CHUNKS \leq NPROC$ gilt außerdem für die Anzahl der Verarbeitungsschritte immer $STEPS = 1$. Der Load-Balance-Faktor eines Programmkerns ergibt sich für $STEPS = 1$ zu:

$$LB = \frac{CHUNKS}{NPROC}$$

Damit sind zum Beispiel für $NPROC = 8$ die Werte $LB = n/8$ mit $1 \leq n \leq 8$ realisierbar. Auf Programmebene wird der Load-Balance-Faktor bzw. die Variable *CHUNKS* über den Speedup eingestellt. Bei der Ausführung eines Programmkerns ist der load balance-abhängige Speedup *LBS* (mit $CHUNKS \leq NPROC$) gleich der Anzahl der *chunks*.

$$LBS = CHUNKS$$

Wenn der Load-Balance-Faktor eines Programmkerns mit der Vorgabe übereinstimmt, d.h. wenn $CHUNKS = LB \times NPROC$ den entsprechenden Wert hat, dann ergibt sich der *load balance*-abhängige Speedup eines Programmkerns zu:

$$LBS = LB \times NPROC$$

Der tatsächliche Speedup LBS ist also ein Maß für den tatsächlichen Load-Balance-Faktor. Um diesen Speedup auch auf Programmebene einzustellen, wird der *load balance*-abhängige Speedup $LBS^{(j)}$ nach der Ausführung von j parallelen Programmkernen ermittelt.

$$LBS^{(j)} = \frac{\sum_{i=1}^n \sum_{k=1}^{m(i)} KTIME(i)}{\sum_{i=1}^n \sum_{k=1}^{m(i)} \frac{KTIME(i)}{CHUNKS(i,k)}}$$

Dabei ist $CHUNKS(i,k)$ die Anzahl der *chunks* des k -ten Kernes i , und es gilt $j = n \cdot m(i)$. Da in dieser Formel der Zähler gerade der Größe $PTIME$ entspricht, läßt sich analog zum Parallelitätsgrad ein Quotient von zwei Zeiten definieren:

$$LBS^{(j)} = \frac{PTIME^{(j)}}{LBTIME^{(j)}}$$

Die beiden Multitasking-Parameter Parallelitätsgrad und Load-Balance-Faktor werden also ähnlich wie die dynamischen Parameter realisiert (siehe Kapitel 4.1.1), d.h. durch eine Kombination von Programmkernen mit jeweils anderen Parameterwerten. Im Gegensatz dazu findet hier jedoch keine Auswahl von Kernen durch dynamische Entscheidungen statt, sondern es werden die bereits ausgewählten Kerne bezüglich der Multitasking-Parameter gezielt entweder parallel oder sequentiell ausgeführt, bei dynamischer Wahl des Load-Balance-Faktors.

- Für die Realisierung des Parallelitätsgrades existiert von jedem Programmkern eine sequentielle und eine parallele Version. Dabei wird für jeden Kern innerhalb der Kernfolge mit obiger Formel der hypothetische Parallelitätsgrad bei sequentieller bzw. bei paralleler Ausführung berechnet. Entsprechend dem besten hypothetischen Wert in bezug auf die Vorgabe wird die sequentielle oder die parallele Version ausgewählt.
- Für die parallelen Kerne wird der Load-Balance-Faktor eingestellt. Dabei muß bei vorgegebener Prozessorzahl die Anzahl der *chunks* in der parallelen Schleife für jeden Kern festgelegt werden. Für $NPROC = 8$ sind zum Beispiel jeweils 8 Werte für $CHUNKS$ möglich, aus denen der jeweils beste Wert ausgewählt werden kann. Bei dieser Strategie wird ein Vorgabewert fast ausschließlich durch den nächstgrößeren und den nächstkleineren der möglichen Werte realisiert, d.h. bei $NPROC = 8$ und einer Vorgabe $LB = 0.80$ wird entweder $CHUNKS = 6$ ($LB = 6/8 = 0.75$) oder $CHUNKS = 7$ ($LB = 7/8 = 0.875$) ausgewählt. Es werden also fast nie alle Prozessoren genutzt, da bei diesen Werten für $CHUNKS$ ständig 1 bzw. 2 Prozessoren *idle* sind. Im Gegensatz dazu können reale Programme zumindest zeitweise alle angeforderten Prozessoren nutzen, d.h. reale Programme sind zeitweise balanciert. Wenn

```

SUBROUTINE PARSEQ

(Deklarationsteil)

LBS    = LB*NPROC
TTIME = TTIME+KTIME(i)
IF(ABS(((PTIME+KTIME(i))/TTIME)-PG).LT.
>  ABS((PTIME/TTIME)-PG)) THEN
  PFLAG = 1
  PTIME = PTIME+KTIME(i)
  IF(CHOICE.LT.1.OR.CHOICE.GT.CHOICEMAX) THEN
    CHOICE = CHOICEMAX
  ENDIF
  IF(CHOICE.GT.CHOICEMAX/2) THEN
    CHUNKS = NPROC
  ELSE
    LBSNBEST = 10
    DO 100 N=1,NPROC
      LBSN = PTIME/(LBTIME+KTIME(i)/N)
      IF(ABS(LBSN-LBS).LT.ABS(LBSNBEST-LBS)) THEN
        LBSNBEST = LBSN
        CHUNKS = N
      ENDIF
    100 CONTINUE
  ENDIF
  LBTIME = LBTIME+KTIME(i)/CHUNKS
  CHOICE = CHOICE-1
ELSE
  PFLAG = 0
  CHUNKS = 4
ENDIF
END

```

Abb. 12. Das Unterprogramm PARSEQ

z.B. 25 *chunks* von 8 Prozessoren abgearbeitet werden, dann sind 3 Verarbeitungsschritte balanciert, d.h. es können jeweils alle 8 Prozessoren genutzt werden, und der 4. Schritt ist unbalanciert (nur noch ein *chunk* bei 8 Prozessoren). Um dieses Verhalten nachzubilden, wird die Anzahl der *chunks* für einen Teil der Kerne mit $CHUNKS = NPROC$ vorgegeben. Für alle übrigen Kerne wird der jeweils beste Wert ausgewählt, d.h. es wird für alle möglichen Werte von $CHUNKS$ der hypothetische Load-Balance-Faktor berechnet, um dann den besten Wert auszuwählen. Da in realen Programmen im allgemeinen ein häufiger Wechsel zwischen balancierter und unbalancierter Ausführung stattfindet, wird auch bei *PAR-Bench* die Anzahl der *chunks* abwechselnd vorgegeben bzw. ausgewählt.

Für diese Berechnungen werden gemäß der oben angestellten Überlegungen nur die sequentiellen Ausführungszeiten der Kerne ($KTIME(i)$) und die Kernfolge ($SKERN(i)$ und $PKERN(i)$) benötigt. Da diese Daten nach dem Generate-Modus zur Verfügung stehen, können die jeweiligen Parameterwerte schon vor der Ausführung der Benchmark-Programme berechnet werden, und die Ergebnisse können als Eingabedaten für den Execute-Modus bereitgestellt werden. Um die Anzahl der Dateien zu begrenzen und damit die Handhabbarkeit von *PAR-Bench* zu verbessern, wurde die Erstellung dieser

Eingabedatensätze vermieden. Statt dessen werden die entsprechenden Berechnungen während der Ausführung der Programme im Execute-Modus durchgeführt. Dabei werden die Daten für jeden Kern unmittelbar vor dessen Ausführung ermittelt. Das Unterprogramm PARSEQ in Abb. 12 wird vor der Ausführung jedes Kerns aufgerufen und berechnet die relevanten Daten. Mit der Variablen *CHOICE* wird gesteuert, ob die Anzahl der *chunks* vorgegeben bzw. ausgewählt werden soll. Die Variable *PFLAG* enthält die Entscheidung für eine parallele oder eine sequentielle Kernversion, und die Variable *CHUNKS* ist identisch mit der gleichnamigen Variable in den Programmkernen. Mit den Variablen *TTIME*, *PTIME* und *LBTIME* werden entsprechend der obigen Formeln die jeweils aktuellen Werte des Parallelitätsgrades und des Load-Balance-Faktors berechnet. Da die erstellten Benchmark-Programme auch solche Programmteile enthalten, die nicht parallelisiert werden können, wird die Variable *TTIME* mit der Dauer dieser sequentiellen Programmteile vorbesetzt. Deswegen ist der maximal realisierbare Parallelitätsgrad auf 99,8% begrenzt¹⁸. In Abb. 13 ist die iterative Realisierung der Vorgabewerte dargestellt. Die angegebenen Werte beziehen sich jeweils auf alle bereits ausgeführten Programmkerne.

	#	PFLAG	CHUNKS	PG	LB	Zeit
Vorgabe				0.8500	0.7000	
	1	1	8	0.2755	1.0000	0.2760
	2	1	8	0.4773	1.0000	0.3826
	3	1	8	0.5639	1.0000	0.4587
	4	1	3	0.6260	0.7254	0.5347
	5	1	5	0.6881	0.6983	0.6413
	6	1	6	0.7212	0.7054	0.7173
	7	1	8	0.7573	0.7428	0.8239
	8	1	8	0.7778	0.7642	0.8999
	9	1	8	0.7951	0.7822	0.9760
	10	1	3	0.8152	0.6915	1.0825
	11	1	7	0.8274	0.7032	1.1586
	12	1	5	0.8380	0.6968	1.2346
	13	1	8	0.8509	0.7171	1.3412
	14	1	8	0.8589	0.7300	1.4172
	15	1	8	0.8687	0.7462	1.5233
	16	1	3	0.8781	0.6904	1.6409
	17	0	4	0.8246	0.6904	1.7475
	18	1	8	0.8319	0.7013	1.8236
	...					
	107	1	8	0.8560	0.7039	7.0945
	108	0	4	0.8469	0.7039	7.1705
	109	1	8	0.8486	0.7065	7.2466
	...					
	734	1	6	0.8504	0.7000	116.6319
	735	1	6	0.8507	0.7001	116.8600
	736	1	8	0.8511	0.7008	117.1798
Ergebnis				0.8511	0.7008	

Abb. 13. Die Realisierung von PG und LB

¹⁸ Dieser Wert geht aus Speedup-Messungen hervor, die in Kapitel 6 beschrieben sind.

Die Vorgaben für den Parallelitätsgrad und für den Load-Balance-Faktor stimmen mit den Ergebnissen sehr gut überein. Dabei sind die Vorgabewerte bereits nach wenigen Sekunden gut approximiert. Abb. 14 stellt die ersten 70 Sekunden der Iteration graphisch dar.

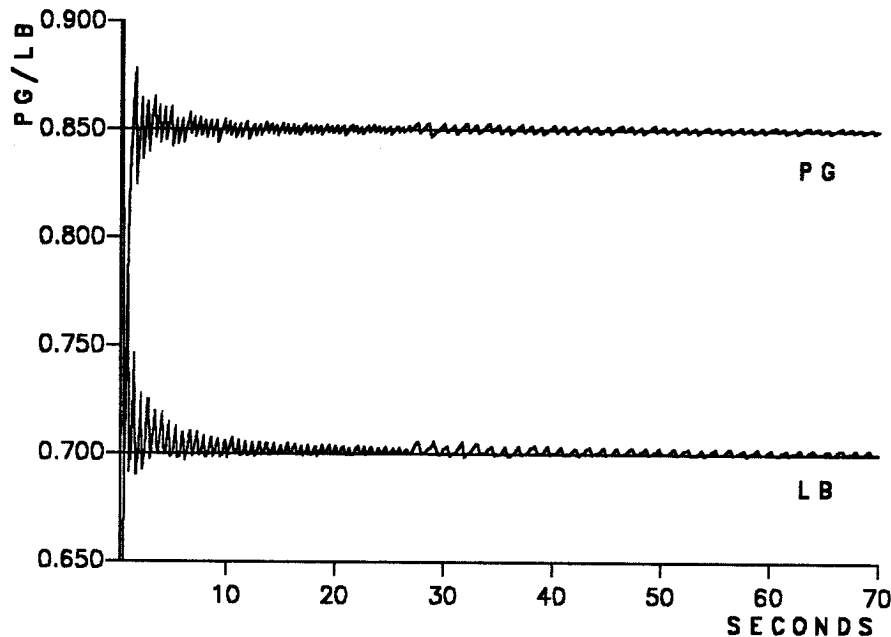


Abb. 14. Die Iteration der Vorgaben (PG, LB)

Der Wert des Parallelitätsgrades ist ungefähr mit gleicher Häufigkeit größer bzw. kleiner als der Vorgabewert. Im Gegensatz dazu wird der Wert des Load-Balance-Faktors nur selten kleiner als der Vorgabewert, da die wiederholte Festlegung der Anzahl der *chunks* mit $CHUNKS = NPROC$ den Load-Balance-Faktor vergrößert. In diesem Beispiel sind bereits nach 10 Sekunden keine wesentlichen Abweichungen mehr vorhanden.

Die Einstellung von Parallelitätsgrad und Load-Balance-Faktor wird also nur aufgrund der Kerndaten vorgenommen, d.h. es erfolgt keine Kontrolle der tatsächlichen Werte dieser Größen. Als Nachweis der Genauigkeit dieser Methode werden in Kapitel 6 die tatsächlichen Werte von Parallelitätsgrad und Load-Balance-Faktor durch Speedup-Messungen ermittelt.

4.2.3. Die Ermittlung der Kerndaten

Bei der Erzeugung der Benchmark-Programme werden die Koordinaten der mit den Kernen korrespondierenden Punkte im Parameterraum und die Ausführungszeiten der Kerne benötigt. Diese Daten werden im Newdata-Modus ermittelt. Dazu werden die

Kerne ausgeführt, und der HPM wird jeweils vor bzw. nach der Ausführung eines Kerns aufgerufen. In Abb. 15 sind die so ermittelten Daten der verwendeten Programmkerne auszugsweise dargestellt¹⁹.

#	MMREFS	MFLOPS	MIOS	KTIME
1	0.3042749E+03	0.2026991E+03	0.0000000E+00	0.7426182E+00
2	0.3016307E+03	0.1987915E+03	0.1590330E+01	0.7572162E+00
3	0.3060782E+03	0.2038969E+03	0.1213671E-02	0.7382560E+00
4	0.3049481E+03	0.2024189E+03	0.5397835E+00	0.7436463E+00
5	0.2880145E+03	0.1913155E+03	0.3416171E-02	0.7868458E+00
6	0.2233510E+03	0.1481289E+03	0.0000000E+00	0.1451709E+00
7	0.2173632E+03	0.1340732E+03	0.7508021E+01	0.1603917E+00
8	0.2262922E+03	0.1500619E+03	0.6252532E-02	0.1433020E+00
9	0.2221540E+03	0.1437296E+03	0.2682945E+01	0.1496147E+00
10	0.1727531E+03	0.1123171E+03	0.1403454E-01	0.1915274E+00
11	0.1575439E+03	0.1039462E+03	0.0000000E+00	0.1034382E+00
12	0.1570095E+03	0.9010829E+02	0.1009192E+02	0.1193255E+00
13	0.1560846E+03	0.1029575E+03	0.8579636E-02	0.1044333E+00
14	0.1569285E+03	0.9861825E+02	0.3681716E+01	0.1090274E+00
...				

Abb. 15. Die Daten der verwendeten Programmkerne (auszugsweise)

4.3. Die Steuerung des Systems PAR-Bench

Zur Ausführung der Benchmarks steht das Programm BENCHGEN zur Verfügung. Dieses ermöglicht sowohl die Ausführung einzelner Benchmark-Programme als auch die Ausführung einer vollständigen Arbeitslast²⁰. Das Programm BENCHGEN generiert entsprechend einer Eingabedatei eine vorgegebene Anzahl von Jobs, die wiederum die Benchmark-Programme ausführen. Die Eingabedatei für das Programm BENCHGEN enthält also die Daten für alle Jobs einer Arbeitslast, ein Beispiel ist in Abb. 16 dargestellt.

```

MODE= 'EXE'
#,DEL, PR, C,SEC,VE, MM, FL,IO, SIZE, PRNAME, PG, LB,NPROC
5, 20, 12, 0, 80, 0, 100, 100, 0, 1000, 'bench', 100, 100, 1
3, 20, 12, 0, 80, 0, 150, 120, 0, 1000, 'bench', 100, 100, 1
2, 50, 12, 0, 50, 0, 70, 80, 0, 1000, 'bench', 100, 100, 1
6, 50, 12, 0, 50, 0, 150, 120, 0, 1000, 'bench', 100, 100, 1
4, 70, 12, 0, 30, 0, 70, 80, 0, 1000, 'bench', 100, 100, 1

```

Abb. 16. Beispiel einer Eingabedatei für BENCHGEN

¹⁹ Die Daten aller verwendeten Programmkerne sind in Anhang A wiedergegeben.

²⁰ Die Simulation einer Arbeitslast ist nur auf einer dedizierten Maschine sinnvoll.

Diese Eingabedatei wird vom Programm BENCHGEN zeilenweise eingelesen und verarbeitet, d.h. eine Zeile enthält alle Parameter, die einen Job spezifizieren. Die erste Zeile bestimmt die Betriebsart, die die Arbeitsweise für den gesamten Benchmark festlegt. Im folgenden werden die Parameterbeschreibung und gegebenenfalls der Wertebereich der Parameter angegeben. Dabei werden die Parameter bezüglich ihrer Aufgabenstellung in Gruppen zusammengefaßt.

- **Programmablaufsteuerung:**

Da die Ausführung von Benchmark-Programmen in verschiedenen Betriebsarten innerhalb eines Benchmarks nicht sinnvoll ist, wird die Betriebsart über den Parameter **MODE** für den gesamten Benchmark global eingestellt. Dabei kann dieser Parameter die folgenden Werte annehmen:

NEW Dieser Wert spezifiziert den Newdata-Modus, d.h. die mit den Programmkernen korrespondierenden Punkte im Parameterraum und die Ausführungszeiten der Kerne werden ermittelt und als Eingabedaten für den Generate-Modus in einer Datei abgelegt. Da die Kerndaten bereits vor der Erzeugung bzw. Ausführung eines Benchmarks zur Verfügung stehen müssen, werden in dieser Betriebsart alle weiteren Eingabedaten ignoriert, d.h. es wird nur ein Job generiert, der die Ermittlung der Kerndaten durchführt, ohne jedoch Benchmark-Programme zu erzeugen bzw. auszuführen.

Da die Kerndaten sowohl von den verwendeten Programmkernen als auch von der Umgebung abhängen, wird eine erneute Ermittlung der Kerndaten bei jeder Änderung bezüglich einer dieser beiden Faktoren notwendig, also entweder bei der Einbeziehung zusätzlicher Programmkerne in den Benchmark oder bei einem Wechsel auf einen anderen Rechner, auf dem der Generate-Modus ausgeführt werden soll.

GEN Mit *GEN* wird der Generate-Modus gestartet, d.h. alle Benchmark-Programme eines Benchmarks werden aufgrund der Vorgaben generiert und in Form von Programmkernfolgen in den entsprechenden Dateien abgespeichert.

Da die Datei mit den Kerndaten als Eingabedatei benötigt wird, muß sichergestellt werden, daß diese Datei bereits existiert. Dabei wird geprüft, ob die in der Datei gespeicherten Daten mit den aktuell verwendeten Programmkernen korrespondieren.

EXE Dieser Wert spezifiziert den Execute-Modus, d.h. ein Benchmark wird aufgrund der bereits erzeugten Kernfolgen und der entsprechenden Eingabedaten ausgeführt. Dabei muß gewährleistet sein, daß die Dateien mit den entsprechenden Kernfolgen bereits existieren.

- **Parameter, die die Arbeitslast beschreiben:**

Aufgrund der Parameter Prozessorzeit, Speicherzugriffsrate, Gleitpunktverarbeitungsrate und Ein-/Ausgabehäufigkeit wird die jeweilige Folge von Programmkernen ermittelt, die die Arbeit eines Programms festlegt. Da diese Folge im

Generate-Modus bestimmt wird, haben die Parameter im Execute-Modus keine Auswirkung.

SEC Dieser Parameter gibt die Prozessorzeit (*cpu time*) des Jobs in Sekunden an. Im Generate-Modus stellt diese Angabe die Abbruchbedingung für die Generierung dar. Es werden solange weitere Kerne ausgewählt und ausgeführt, bis die vorgegebene Prozessorzeit erreicht ist. Im Execute-Modus kann die Prozessorzeit nicht explizit modifiziert werden, da unabhängig von der verbrauchten Prozessorzeit alle Kerne der jeweiligen Folge ausgeführt werden.

Der Wertebereich für die Prozessorzeit ist nur nach unten begrenzt, d.h. es wird zumindest ein Programmkern ausgeführt. Schon bei sehr kurzen Prozessorzeiten können die Vorgaben für die dynamischen Parameter mit großer Genauigkeit realisiert werden (siehe Kapitel 4.2.1).

MM Die Speicherzugriffsrate wird in MMREFS (*Million Memory REferences per Second*) angegeben.

FL Die Gleitpunktverarbeitungsrate wird in MFLOPS (*Million Floating-point Operations per Second*) angegeben.

IO Die Ein-/Ausgabehäufigkeit wird in MIOS (*Million array elements transferred to or from an I/O device per Second*) angegeben.

Diese Parameter können innerhalb eines dreidimensionalen Parametergebietes realisiert werden (siehe Kapitel 4.2.1). Das Parametergebiet hängt von der Maschine und von den verwendeten Programmkernen ab. In Abb. 17 ist das Parametergebiet für eine Cray Y-MP8/832 und die in Kapitel 4.2.1 beschriebenen Programmkerne dargestellt.

VE Mit *VE* kann eine spezielle Betriebsart eingestellt werden. Dabei gibt *VE* die Versionsnummer eines bestimmten Kerns an, der dann ausschließlich ausgeführt wird. Der Wert '0' bedeutet Normalbetrieb, d.h. die Vorgabeparameter werden durch eine Kombination verschiedener Kerne approximiert. Auch dieser Parameter ist nur für den Generate-Modus relevant.

- **Die variablen Parameter:**

Die variablen Parameter können nach der Festlegung einer Kernfolge vor der Ausführung der Benchmarks variiert werden. Dies ermöglicht die Ausführung von Benchmark-Programmen, die sich nur bezüglich dieser Parameter unterscheiden. Zu dieser Gruppe gehören die statischen Parameter Speicherbedarf und Priorität sowie die Multitasking-Parameter. Die Parameter Speicherbedarf und Priorität werden sowohl im Generate-Modus als auch im Execute-Modus eingestellt.

SIZE Die Vorgabe für den Speicherbedarf des Jobs wird in Kilo Worten angegeben. Diese kann nur dann realisiert werden, wenn der Mindestbedarf nicht unterschritten wird. Sonst wird das Benchmark-Programm mit dem Mindestbedarf an Speicherplatz ausgeführt.

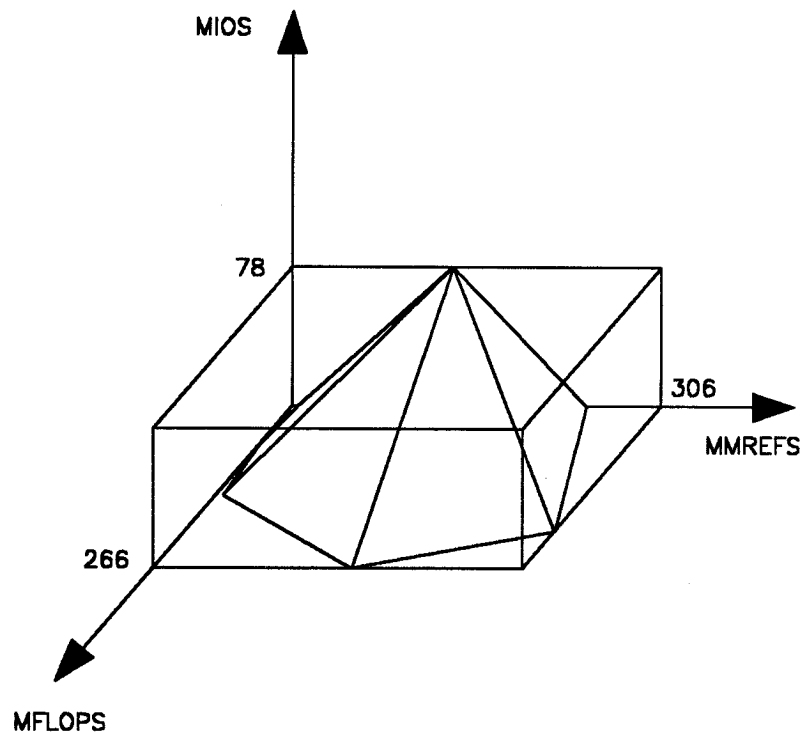


Abb. 17. Das 3-dimensionale Parametergebiet von PAR-Bench

PR Dieser Parameter gibt den *nicevalue* eines Jobs an, der zur Einstellung der Priorität benötigt wird. Der Wertebereich erstreckt sich von 0 (höchste Priorität) bis 19 (niedrigste Priorität).

Im Gegensatz zu diesen Parametern haben die Multitasking-Parameter im Generate-Modus keine Auswirkung, da nur sequentielle Programme generiert werden. Die Parallelisierung der Programme erfolgt erst im Execute-Modus und wird durch die Multitasking-Parameter spezifiziert. Sie geben an, ob ein Programm parallel oder sequentiell ausgeführt werden soll und charakterisieren das Verhalten der parallelen Programme durch die Anzahl der anzufordernden Prozessoren *NPROC*, den Parallelitätsgrad *PG* und den Load-Balance-Faktor *LB*.

NPROC Aufgrund der Anzahl der anzufordernden Prozessoren wird die Programmversion ausgewählt. Für *NPROC* = 1 wird eine sequentielle Programmversion ausgeführt, d.h. eine Version, die ohne Multitasking-Präprozessor übersetzt wurde. Für alle anderen Werte von *NPROC* wird eine parallele Programmversion ausgeführt. Die Anzahl der anzufordernden Prozessoren kann zwischen '1' und der Anzahl der physikalisch verfügbaren Prozessoren gewählt werden²¹.

PG, LB Der Parallelitätsgrad *PG* und der Load-Balance-Faktor *LB* werden in Prozent angegeben und können im Bereich '0' bis '100' gewählt werden. Da die Benchmark-Programme auch Programmteile enthalten, die nicht

²¹ Die Cray Y-MP8/832 verfügt über 8 Prozessoren.

parallelisiert werden können, wird auch bei der Vorgabe $PG = 100$ nur ein maximaler Parallelitätsgrad von 99.8% realisiert.

Diese beiden Parameter haben nur bei Ausführung einer parallelen Programmversion eine Auswirkung (siehe auch bei *NPROC*). In diesem Fall wird für $PG = 0$ die Anzahl der Prozessoren gleich 1 gesetzt, d.h. es wird eine parallele Programmversion mit einem Prozessor ausgeführt.

Tab. 1 beschreibt das Zusammenwirken der Multitasking-Parameter.

NPROC	PG	Version
1	beliebig	sequentiell
2-8	0	parallel mit $NPROC = 1$
2-8	ungleich 0	entsprechend der <i>NPROC</i> -Vorgabe parallel

Tab. 1. Das Zusammenwirken der Multitasking-Parameter

Die Parallelisierung der Programmkerne bezieht sich nur auf die inneren Schleifen, jedoch nicht auf die Ein-/Ausgabeoperationen (siehe Abb. 9 auf Seite 40). Deswegen können der Parallelitätsgrad und der Load-Balance-Faktor nur dann realisiert werden, wenn die betroffenen Programmkerne keine Ein-/Ausgabe durchführen. Diese Forderung wird erfüllt, wenn im zugehörigen Programmablauf im Generate-Modus die Vorgabe $IO = 0$ gemacht wurde.

- **Weitere Parameter zur Modellierung der Benchmarks:**

Die weiteren Parameter haben zum Teil Einfluß auf die Konfiguration einer Arbeitslast oder initialisieren spezielle Betriebsarten wie z.B. die Einbeziehung von Anwendungsprogrammen in einen Benchmark.

Dieser Parameter gibt die Anzahl der zu generierenden Jobs mit den Daten dieser Zeile an.

DEL Mit *DEL* (*delay*) wird eine Verzögerungszeit angegeben, nach der das Benchmark-Programm gestartet wird. Damit läßt sich die Startzeit der einzelnen Benchmark-Programme relativ zueinander innerhalb einer Arbeitslast vorgeben. Die Verzögerungszeit sollte zumindest so groß gewählt werden, daß die Benchmark-Programme erst dann beginnen, wenn alle Jobs bereits generiert sind.
Unter UNICOS wird diese Verzögerung durch das Kommando 'sleep' realisiert.

C Der Parameter *C* gibt die Job-Klasse der einzelnen Jobs an. In der Testphase werden die Programme im normalen Rechenbetrieb ausgeführt und die Priorität eines Jobs wird vom Betriebssystem aufgrund der Job-Klasse ermittelt. Bei den relevanten Messungen dagegen wird die Einstellung der Priorität der Benchmark-Programme explizit vorgenommen. Deshalb werden die Programme in der speziellen Job-Klasse 'MULTIX' gestartet, in der die übliche Ermittlung der Prioritäten durch das Betriebssystem abgeschaltet ist. Der Wert '1' startet

den generierten Job in der Job-Klasse 'MULTIX', '0' in der Job-Klasse 'HIGH'. Die Job-Klasse wird in einer JCL-Anweisung (*Job Control Language*) des Jobs spezifiziert.

PRNAME gibt den Namen des auszuführenden Programms an. Über diese Variable können zusätzliche Programme, z.B. reale Produktionsprogramme, in den Benchmark einbezogen werden.

Aufgrund dieser Eingabedaten generiert das Programm BENCHGEN die entsprechenden Jobs, die wiederum die Benchmark-Programme ausführen. Dazu schreibt das Programm die JCL-Anweisungen und die Eingabedaten für das jeweilige Benchmark-Programm in eine Datei und submittiert diese als neuen Job. In Abb. 18 ist ein Beispiel einer solchen Datei dargestellt.

```
# USER=zdv202 PW=$CRAY          # Benutzernummer und Password
# QSUB -r    be020                # Jobidentifikation
# QSUB -lT   330                  # Zeitlimit des Jobs
# QSUB -lM   4Mw                  # Speicherplatzlimit des Jobs
# QSUB -q    multix               # Job-Klasse
# QSUB -ln   15                   # Nicevalue
# QSUB
sleep      70                    # Delay
cd $HOME/exe
cat > benin020 << 'EOF'          # Eingabedatei des Benchmark-Programms
'UNICOS','BE020'
 30.,0,0.,70.,80.,2000000,0,0
 1,0
100.,100.,6
EOF
$HOME/bin/bench.par < benin020   # Aufruf des Benchmark-Programms
```

Abb. 18. Die JCL-Anweisungen eines von BENCHGEN erzeugten Jobs

Jeder dieser Jobs erzeugt eine Ausgabedatei, die einen Teil der für den Benchmark relevanten Meßdaten enthält. Die graphische Auswertung dieser Daten ist in Kapitel 5 beschrieben.

4.4. Portabilität

Das Benchmark-System *PAR-Bench* erzeugt synthetische Benchmark-Programme, die weitgehend dem FORTRAN 77 Standard entsprechen. Deswegen können die erzeugten Benchmarks ohne großen Aufwand auf andere Systeme portiert werden. Zum Vergleich verschiedener Rechner werden die Benchmarks durch die Ermittlung der Kernfolge auf einem Rechner erzeugt. Diese Kernfolge kann dann auch auf anderen Rechnern ausgeführt werden. Die Erzeugung der Benchmarks erfolgt in dieser Arbeit auf den Systemen Cray X-MP oder Cray Y-MP unter den Betriebssystemen COS bzw. UNICOS. Die

Methode von separater Generierung und Ausführung gewährleistet die weitgehende Portabilität der Benchmarks.

Da die Benchmark-Programme auch im Execute-Modus einige Maschinen- und Betriebssystemabhängigkeiten wie Systemzeitabfrage, Maschinentypabfrage und Dateihandhabung enthalten, sind bei der Portierung einige Modifikationen notwendig. Bezüglich dieser Abhängigkeiten existiert eine wohldefinierte Schnittstelle, die in Kapitel 4.5 beschrieben wird.

Die Parallelisierung der Benchmark-Programme erfolgt auf den Cray-Systemen durch die beiden Multitasking-Strategien Microtaking bzw. Autotasking. Dazu wurden Präprozessor-Direktiven in die entsprechenden FORTRAN-Programme eingefügt. Diese Direktiven beginnen in der ersten Spalte mit einem 'C' und werden deswegen von anderen Compilern wie Kommentare behandelt. Dadurch ist die Portabilität des Benchmarks auch bezüglich dieser Direktiven gewährleistet. Für eine Parallelisierung der Benchmark-Programme auf anderen Systemen ist jedoch zusätzlicher Aufwand notwendig, der vom jeweiligen System abhängig ist.

Das Programm BENCHGEN generiert die JCL-Anweisungen (*Job Control Language*) der einzelnen Benchmark-Jobs. Da die JCL maschinenabhängig ist, muß auch das Programm BENCHGEN für ein anderes System erweitert werden.

4.5. Die Schnittstelle zum FORTRAN 77 Standard

Da auch bei der Ausführung der Benchmarks im Execute-Modus einige Systemroutinen benötigt werden, wurde für die Portierung der Benchmarks eine komfortable Schnittstelle definiert:

- Alle für den Execute-Modus relevanten Maschinenabhängigkeiten, d.h. alle Programmanweisungen, die nicht dem FORTRAN 77 Standard entsprechen, befinden sich in speziell benannten Unterprogrammen.
- Von allen Unterprogrammen, die Maschinenabhängigkeiten enthalten, wird für jeden Rechner bzw. Betriebssystem eine entsprechende Version erstellt. Dabei existiert nur ein Gesamtprogramm, das alle Versionen der entsprechenden Unterprogramme enthält. Die Versionen sind durch den ersten Buchstaben des Namens eindeutig einem bestimmten System zuzuordnen. Zum Beispiel beginnen die Namen der entsprechenden UNICOS-Versionen mit 'U' und die Namen der COS-Versionen mit 'C'.

Diese Vorgehensweise ermöglicht die Benutzung eines Quelltextes auf mehreren verschiedenen Systemen. Der Vorteil dieser Lösung besteht darin, daß keine Programmodule ausgewechselt werden müssen und damit keine Gefahr besteht, die falschen Module aufzurufen.

- Die betroffenen Unterprogramme werden nur von speziellen Modulen aufgerufen, die ausschließlich die Auswahl der entsprechenden Version vornehmen. Dabei wird die Auswahl aufgrund einer eindeutigen Systemidentifikation getroffen, die den Benchmark-Programmen als Eingabe-Parameter zur Verfügung gestellt wird. Die

Namen dieser Module beginnen mit 'X' und unterscheiden sich nur durch den ersten Buchstaben von den Namen der korrespondierenden Unterprogramme.

Vom Unterprogramm 'PROLOG' existieren zum Beispiel die beiden Versionen 'CPROLOG' und 'UPROLOG', und im Modul 'XPROLOG' wird die jeweilige Version ausgewählt. Im folgenden sind die Maschinenabhängigkeiten aufgeführt, und am Beispiel des Betriebssystems UNICOS werden die einzelnen Realisierungen erläutert.

Datum, Uhrzeit und Maschinenidentifikation

Um die jeweilige Benchmark-Messung eindeutig identifizieren zu können, wird von jedem Benchmark-Programm das Datum, die Uhrzeit und die Maschinenidentifikation ermittelt. Die Ausführung der Benchmarks kann jedoch auch ohne diese Daten erfolgen. Die Ermittlung der Uhrzeit ist außerdem für die Auswertung der Benchmarks relevant, weil dabei die Start- und Endzeitpunkte der einzelnen Benchmark-Programme betrachtet werden. Stellvertretend für die entsprechenden Unterprogramme werden im folgenden die Namen der jeweils aufrufenden Module angegeben:

XPROLOG und XEPILOG:

In den zugehörigen Unterprogrammen werden das Datum und der Start- bzw. Endzeitpunkt eines Benchmark-Programms ermittelt. Dazu werden unter UNICOS die Systemroutinen *TIMEF* [Cra2079] und *gettimeofday* [Cra2012] verwendet. Desweiteren wird die Maschinenidentifikation ermittelt. Dazu wird unter UNICOS die Routine *UNAME* [Cra2079] benutzt.

XAUSGTIM:

Da die Ermittlung von Datum und Uhrzeit maschinenabhängig ist, kann zur Darstellung dieser Daten die Benutzung von jeweils anderen Formaten bzw. Variablen von Vorteil sein. Dies trifft zum Beispiel bezüglich COS und UNICOS zu, wo für die Ausgabe dieser Daten jeweils andere Versionen des Unterprogramms 'AUSGTIM' benutzt werden.

Der Speicherplatzbedarf

Der Speicherplatzbedarf gehört zu den variablen Parametern. Die Einstellung dieses Parameters erfolgt zur Laufzeit des Programms.

XAQUIREM:

Im Unterprogramm 'AQUIREM' sind die Abfrage des belegten Speicherplatzes und die Vergrößerung des Speicherplatzes auf den vorgegebenen Wert realisiert. Unter UNICOS wird dazu die Systemroutine *MEMORY* [Cra2079] verwendet.

Dateihandling

Im Execute-Modus führen die Benchmark-Programme Zugriffe auf die Dateien durch, in der die jeweiligen Kernfolgen gespeichert sind. Da die Art des Zugriffs (Lesezugriff bzw. Schreib- und Lesezugriff) auf eine benötigte Eingabedatei von der Betriebsart (Generate- bzw. Execute-Modus) abhängt, kann das Bekanntmachen der entsprechenden Datei erst nach dem Einlesen der Parameter, d.h. zur Laufzeit der Programme erfolgen. Diese Bekanntmachung einer Datei ist nicht in allen Systemen nur aufgrund von FORTRAN-Anweisungen möglich. Unter COS z.B. muß eine Datei dem Programm durch eine Systemroutine explizit bekannt gemacht werden.

XACCESS und XOPEN:

Eine entsprechende Version des Unterprogramms 'ACCESS' wird nur dann benötigt, wenn im jeweiligen System die Bekanntmachung von Eingabedateien erforderlich ist. Da unter COS als Folge dieser Methode auch die Zuweisung einer FORTRAN-Einheit an eine Datei durch eine Systemroutine erfolgt, ergibt sich eine weitere Maschinenabhängigkeit. Diese Zuweisung wird im Unterprogramm 'OPEN' durchgeführt. Unter UNICOS wird keine Version des Unterprogramms 'ACCESS' benötigt, und die Zuweisung einer FORTRAN-Einheit an eine Datei wird mit der FORTRAN-Anweisung *OPEN* durchgeführt.

5. Das Benchmark-Analysesystem GRANSYS

Das Benchmark-System *PAR-Bench* liefert Informationen über die einzelnen Benchmark-Programme und über die wechselseitigen Einflüsse dieser Programme untereinander. Jedes Benchmark-Programm erzeugt eine Ausgabedatei mit den Informationen zu diesem Programm, sowohl über die Parametervorgaben als auch über die tatsächlich erreichten Parameterwerte, z.B. für den Speicherbedarf, die Prozessorzeit und die Multi-tasking-Parameter. Darüber hinaus werden Informationen über den Start- und Endzeitpunkt des Programms und über die vom System verbrauchte Prozessorzeit angegeben. Da eine Benchmark-Messung aus mehreren Benchmark-Programmen besteht und da für die Untersuchung einzelner Einflußgrößen mehrere Benchmark-Messungen miteinander verglichen werden müssen, ist die Gesamtheit der relevanten Daten so umfangreich, daß eine manuelle Auswertung der Untersuchungen praktisch nicht möglich ist. Deswegen wurde zur automatischen Auswertung der Benchmarks das Benchmark-Analysesystem *GRANSYS*²² entwickelt. Dieses System führt eine automatische Analyse durch, die die Interpretation, die graphische Visualisierung und erste Bewertungen umfaßt. Dabei kann sowohl die Analyse einzelner Benchmarks und die Erstellung entsprechender Graphiken als auch der Vergleich mehrerer Benchmark-Messungen mit Hilfe von Tabellen auf der Grundlage von allen zur Verfügung stehenden Daten durchgeführt werden.

Die von den Benchmark-Programmen erzeugten Daten sind aus der Sicht eines Benutzers interessant in bezug auf das Verhalten seiner Programme. Im Gegensatz dazu werden Informationen über die Systemaktivitäten benötigt, um z.B. den Einfluß der parallelen Programme aus der Sicht eines Rechenzentrums zu untersuchen. Zu diesem Zweck wird der Systemmonitor *SAR*²³ (siehe Kapitel 2) verwendet, der die Informationen über die Systemaktivitäten bezüglich der gesamten Arbeitslast ermittelt. Auch die Auswertung dieser Daten erfolgt mit Hilfe von *GRANSYS*.

Die Auswertung der Benchmarks kann auf beliebigen Rechnern durchgeführt werden. Da die bei den Untersuchungen in dieser Arbeit verwendeten Systeme Cray X-MP/416 und Cray Y-MP8/832 über einen Front-End-Rechner, eine IBM 3090 unter dem Betriebssystem VM/CMS, betrieben werden, erfolgt auch die Auswertung der Daten auf diesem System. Das Benchmark-Analysesystem *GRANSYS* ist in der Programmiersprache REXX²⁴ [IBM24] implementiert und benutzt das Graphiksystem GDDM²⁵ [IBM33] zur Visualisierung. Daneben werden Tabellen erzeugt, die durch das Textverarbeitungssystem GML²⁶ [IBM20] weiterverarbeitet werden können.

5.1. Die Auswertung von PAR-Bench

Bei der Ausführung einer Arbeitslast im Mehrbenutzerbetrieb mit Hilfe von *PAR-Bench* werden mehrere Benchmark-Programme simultan auf einer dedizierten Maschine ausgeführt. Bei einer solchen Benchmark-Messung entspricht jedem Benchmark-Programm ein

²² *GRaphical ANalyzing SYstem*

²³ *System Activity Report*

²⁴ *Restructured EXtended eXecutor language*

²⁵ *Graphical Data Display Manager*

²⁶ *Generalized Markup Language*

korrespondierender Job, und jeder Job erzeugt eine Ausgabedatei mit einer Menge von Meßdaten. Dabei wird für die jeweilige Ausgabedatei die zugehörige Benchmark-Messung durch den Dateityp (*filetype*) und das Benchmark-Programm durch den Dateinamen (*filename*) eindeutig identifiziert. Jede dieser Dateien enthält eine Vielzahl von Informationen. Neben Betriebssystem- und Maschinenidentifikation werden die Parametervorgaben und die relevanten Meßdaten ausgegeben. In Abb. 19 ist eine solche Datei auszugsweise wiedergegeben²⁷.

```
*****
* JOB      = BE008                      *
* MACHINE  = CRAY Y-MP8/832            *
* SYSTEM   = UNICOS                     *
*****
```

EINGABE DATEN =====

BETRIEBSART.....MODE	EXE
MEMORY-GROESSE.....TSIZE	4000000
PARALLELITAETSGRAD.....PG	100.00000000
LOAD BALANCE.....LB	78.12500000
ANZAHL CPUS.....NPROC	6

...

AUSGABE DATEN =====

ZEITEN:	DATUM.....DATUM	06/18/90
-----	STARTZEIT.....CLOCKST	12:13:48.332720
	ENDZEIT.....CLOCKEND	12:16:31.445597
	VERBRAUCHTE REALZEIT.....ZEIT	163.11288
	MEMORY-GROESSE.....SIZE	4000000
	PARALLELITAETSGRAD.....PGEFF	100.00000000
	LOAD BALANCE.....LBEFF	78.12438109
	ANZAHL CPUS.....NPROCEFF	6

...

```
*****
Job report of all user commands
*****
Group 0: CPU seconds      :   92.32    CP executing      : 15387429630
```

...

Abb. 19. Ausgabedatei eines Benchmark-Programms (auszugsweise)

Darin sind die Betriebsart, die Vorgaben für den Speicherplatzbedarf und die Multitasking-Parameter und die entsprechenden tatsächlichen Werte dieser Größen sowie der

²⁷ Anhang B enthält eine nahezu vollständige Ausgabedatei.

Start- bzw. Endzeitpunkt des Programms angegeben. Die Daten in dieser Datei werden zum Teil von den Benchmark-Programmen selbst erzeugt, z.B. werden Start- und Endzeitpunkt durch den Aufruf von Systemroutinen ermittelt. Andere Informationen wie die verbrauchte Prozessorzeit oder die Priorität eines Jobs werden üblicherweise vom Betriebssystem in der Ausgabedatei des Jobs angegeben.

5.1.1. Die Analysen der PAR-Bench-Informationen

Das Benchmark-Analysesystem *GRANSYS* bietet eine Vielzahl von Auswertungsmöglichkeiten der von *PAR-Bench* erzeugten Daten. Die Spezifikation der gewünschten Analyse ist abfragegesteuert. In Abb. 20 ist die erste Auswahlenebene dargestellt.

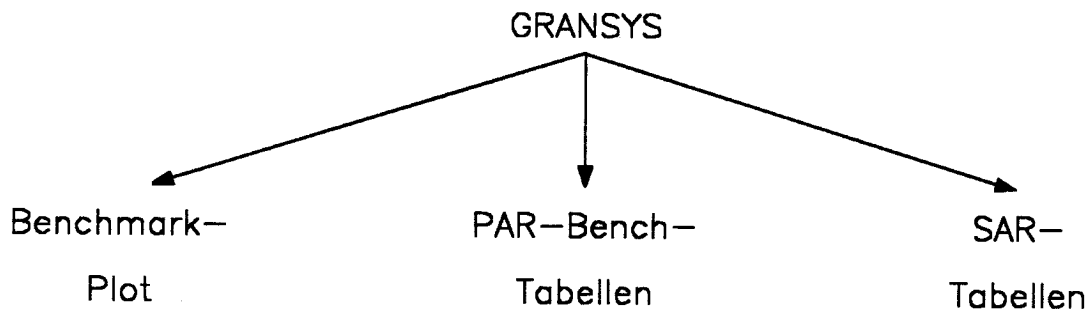


Abb. 20. Die erste Auswahlenebene von GRANSYS

Für die Analyse eines einzelnen Benchmarks steht der *Benchmark-Plot* zur Verfügung. Ein Vergleich verschiedener Messungen kann mit Hilfe der entsprechenden Plots erfolgen. Die *PAR-Bench-Tabellen* verarbeiten die Daten mehrerer Messungen und unterstützen so einen direkten Vergleich. Die *SAR-Tabellen* werden in Kapitel 5.2 beschrieben.

5.1.1.1. Der Benchmark-Plot

Um den Einfluß von einzelnen Testprogrammen auf den Systemdurchsatz zu untersuchen, muß das Zeitverhalten aller Programme der jeweiligen Arbeitslast betrachtet werden. Der *Benchmark-Plot* stellt das Zeitverhalten der Programme von jeweils einer Benchmark-Messung graphisch dar und liefert bestimmte Tabelleninformationen zu dieser Messung.

Nach der Auswahl des *Benchmark-Plots* in der ersten Ebene der Abfragesteuerung kann das Layout des Plots aus mehreren Mustern ausgewählt werden. Die Einstellung des jeweiligen Musters wird durch die Aktivierung einer entsprechenden *Profile-Datei* realisiert. In einer weiteren Auswahlenebene wird die gewünschte Messung mit Hilfe einer eindeutigen Kennung spezifiziert. Für die Analyse werden alle Ausgabedateien der Messung benötigt. Diese werden innerhalb der *filelist* aufgrund des Dateityps (*filetype*) identifiziert, der mit der eindeutigen Kennung der Messung übereinstimmt. Die für die Analyse relevanten Daten werden eingelesen und in Variablen abgespeichert, um eine Weiterverarbeitung zu ermöglichen. Dabei wird zum Beispiel die für den Durchsatz des Systems relevante Dauer des Benchmarks *Bench-time* aufgrund der Start- und Endzeit-

punkte aller Benchmark-Programme ermittelt. Außerdem werden innerhalb einer Benchmark-Messung spezielle Testjobs lokalisiert, die parallel oder mit erhöhter Priorität ausgeführt wurden. Die für den gesamten Benchmark und für die speziellen Testjobs ermittelten Daten werden in einer Datei BENCHDAT abgelegt. Diese Datei kann bei allen weiteren Analysen mit GRANSYS verwendet werden, so daß die Analyse der Ausgabedateien in diesem Fall eingespart werden kann²⁸. Auf diese Weise wird der zur Datenhaltung benötigte Speicherplatz erheblich verringert. Die Datei BENCHDAT dient als Eingabedatei (*import file*) für das IBM Graphiksystem GDDM. Dieses System wird von GRANSYS aufgerufen und erzeugt den *Benchmark-Plot*. Die Graphik und die Tabellen des *Benchmark-Plots* visualisieren sowohl das gesamte Systemverhalten als auch die speziellen Testjobs. Abb. 21 zeigt eine solche Visualisierung für eine Benchmark-Messung.

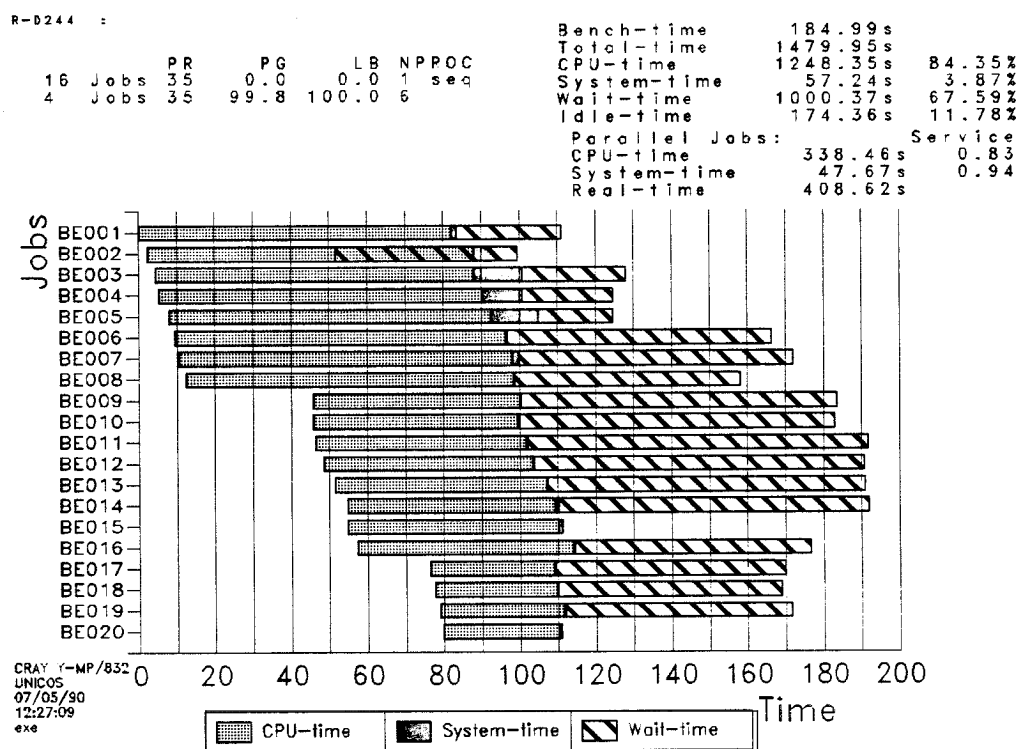


Abb. 21. Beispiel für einen Benchmark-Plot

In der Graphik wird das Zeitverhalten der einzelnen Benchmark-Programme wiedergegeben. Dabei geben die Anfangs- bzw. Endpunkte der Balken die Start- und Endzeitpunkte der jeweiligen Jobs relativ zum Startzeitpunkt des ersten Jobs an. Die verschiedenen Balken stellen die *CPU-time* (vom Benchmark-Programm verbrauchte Prozessorzeit), die *System-time* (vom System verbrauchte Prozessorzeit) und die *Wait-*

²⁸ Die Datei BENCHDAT wird insbesondere auch für die Erstellung der *PAR-Bench-Tabellen* benötigt (siehe Kapitel 5.1.1.2).

time (Wartezeit) der Benchmark-Programme dar²⁹. Wenn der Balken für die *Wait-time* über den beiden anderen liegt, dann war die Ausführungszeit des Jobs kleiner als die verbrauchte Prozessorzeit. Dabei gibt das linke Ende des überlappenden Balkens den Endzeitpunkt des Jobs an.

Eine genaue Erläuterung dieser Zeitgrößen wird in der folgenden Beschreibung der Tabellen gegeben. In den Tabellen werden die für den gesamten Benchmark relevanten Zeitgrößen und die Zeitgrößen spezieller Jobs separat angegeben. Die einzelnen Zeitgrößen haben die folgenden Bedeutungen:

- **Bench-time:**
Hier wird die mittlere Dauer des gesamten Benchmarks angegeben. Die *Bench-time* wird bestimmt durch den Startzeitpunkt des ersten Jobs, der in das System gelangt, und durch die Endzeitpunkte der N letzten Jobs³⁰, die das System verlassen, wobei N die Anzahl der Prozessoren des Systems ist. Die Differenz aus der mittleren Endzeit und der Startzeit ergibt die *Bench-time*, die ein Maß für den Systemdurchsatz ist.
- **Total-time:**
Die *Total-time* ist die gesamte zur Verfügung stehende Prozessorzeit. Diese ergibt sich aus dem Produkt der *Bench-time* und der Anzahl der Prozessoren des Systems.
- **CPU-time:**
Die *CPU-time* ist die Summe aller Zeitintervalle, in denen ein Benchmark-Programm die Kontrolle über einen Prozessor hat. Die *CPU-time* eines Jobs wird vom Betriebssystem UNICOS in der Ausgabedatei des Jobs angegeben.
Für die *CPU-time* des gesamten Benchmarks wird die *CPU-time* aller Jobs addiert. Für die Tabelle 'Parallel Jobs' wird die *CPU-time* aller parallelen Jobs addiert.
- **System-time:**
Die *System-time* ist die Summe aller Zeitintervalle, in denen das Betriebssystem die Kontrolle über einen Prozessor hat. Die *System-time* eines Jobs wird vom Betriebssystem UNICOS in der Ausgabedatei des Jobs angegeben.
Für die *System-time* des gesamten Benchmarks wird die *System-time* aller Jobs addiert. Für die Tabelle 'Parallel Jobs' wird die *System-time* aller parallelen Jobs addiert.
- **Wait-time:**
Die Bedeutung der *Wait-time* unterscheidet sich bei sequentiellen und parallelen Jobs. Im sequentiellen Fall ist die *Wait-time* die Summe aller Zeitintervalle, in denen ein Job keine Kontrolle über einen Prozessor hat. In dieser Zeit wartet der Job auf die Zuteilung eines Prozessors, oder er führt Ein-/Ausgabeoperationen durch. Die

²⁹ Da diese Zeitgrößen zum Teil sowohl vom System *PAR-Bench* als auch vom Systemmonitor SAR ermittelt werden, erfolgt eine Unterscheidung aufgrund der Schreibweise. Während z.B. mit *CPU-time* die von *PAR-Bench* ermittelte Größe bezeichnet wird, ist mit *cpu time* die allgemeine Bezeichnung gemeint. Die entsprechenden Bezeichnungen bei den SAR-Informationen werden in Kapitel 5.2 erläutert.

³⁰ Dabei werden parallele Jobs entsprechend der Anzahl der angeforderten Prozessoren mehrfach gezählt.

Wait-time ergibt sich aus der Differenz der *Real-time* und der Summe aus *CPU-time* und *System-time*. Bei parallelen Jobs wird die *Wait-time* auf die gleiche Art ermittelt. Da jedoch mehrere Prozessoren gleichzeitig genutzt werden können, ist keine Aussage über die Zeit möglich, in der ein paralleler Job keine Kontrolle über einen Prozessor hatte. In Verbindung mit der *CPU-time* und der *System-time* gibt die *Wait-time* ein Maß für die mittlere Anzahl der zugeteilten Prozessoren an. Wenn die *Wait-time* z.B. negativ ist (Balkenüberlappung in Abb. 21, d.h. das linke Ende des überlappenden Balkens gibt den Endzeitpunkt des Jobs an), dann ist die mittlere Anzahl der zugeteilten Prozessoren größer als 1.

Für die *Wait-time* des gesamten Benchmarks wird die *Wait-time* aller Jobs addiert.

- **Idle-time:**

Die *Idle-time* ist die Summe aller Zeitintervalle, in denen ein Prozessor nicht benutzt wird.

Die *Idle-time* des gesamten Benchmarks ergibt sich aus der Differenz der *Total-time* und der Summe der *CPU-time* des gesamten Benchmarks und der *System-time* des gesamten Benchmarks:

$$Idle - time = Total - time - (CPU - time_{gesamt} + System - time_{gesamt})$$

- **Real-time:**

Die *Real-time* ist die Ausführungszeit eines Benchmark-Programms.

Für die Tabelle 'Parallel Jobs' wird die *Real-time* aller parallelen Jobs addiert.

Die beiden Angaben in der Spalte *Service* geben den Servicefaktor an, d.h. das Verhältnis von *CPU-time* zu *Real-time* (oberer Wert) bzw. von der Summe aus *CPU-time* und *System-time* zu *Real-time* (unterer Wert).

In einer weiteren Tabelle werden Informationen zu den Jobs des Benchmarks gegeben. Dabei wird für verschiedene Jobgruppen die Anzahl der Jobs, die Priorität (*PR*), der Parallelitätsgrad (*PG*), der Load-Balance-Faktor (*LB*) und die Anzahl der Prozessoren (*NPROC*) angegeben. Für den Fall *NPROC* = 1 wird zusätzlich die Programmversion angegeben. Dabei bedeutet 'seq', daß eine sequentielle Programmversion verwendet wurde, während 'par' mit einer parallelen Programmversion korrespondiert.

Nach der Darstellung des *Benchmark-Plots* erlaubt *GRANSYS* die Ausgabe auf verschiedenen Plotttern. Zusätzlich wird der Plot in einer *Plot-Datei* gespeichert, um eine Weiterverarbeitung - zum Beispiel das Einbinden in Texte - zu ermöglichen.

Nach der Beendigung eines solchen Programmlaufes erfolgt die Rückkehr in die erste Auswahlebene und es kann eine weitere Analyse gestartet werden.

5.1.1.2. Die PAR-Bench-Tabellen

Während der *Benchmark-Plot* nur die Daten einer Benchmark-Messung visualisiert, werden in den *PAR-Bench-Tabellen* Daten aus mehreren Messungen in Tabellenform gegenübergestellt. Dabei werden zunächst die gewünschten Messungen aufgrund ihrer Kennung ausgewählt. Da diese Auswahl in einer Datei gespeichert wird, existiert vor jedem Programmlauf bereits eine Default-Auswahl, die der jeweils letzten Auswahl entspricht. Die Erstellung der Tabellen erfolgt auf der Grundlage der entsprechenden

BENCHDAT-Dateien, die wiederum mit Hilfe der Kennung identifiziert werden. Falls die benötigten Dateien nicht existieren, werden sie automatisch erstellt und gespeichert.

Im nächsten Schritt können die gewünschten Daten ausgewählt werden. Dabei können alle Daten spezifiziert werden, die in den Tabellen der *Benchmark-Plots* aufgeführt werden. Auch diese Auswahl wird in einer Datei gespeichert, und kann so für die Erstellung weiterer Tabellen wiederverwendet werden.

Die so spezifizierten Daten werden aus den BENCHDAT-Dateien extrahiert und in einer temporären Datei gespeichert. Dabei werden die Daten so formatiert, daß sie vom IBM Textverarbeitungssystem GML als Tabelle dargestellt werden können. Dieses System kann von GRANSYS aufgerufen werden und visualisiert dann die Tabelle. Dabei stehen alle Funktionen des Systems GML zur Verfügung, d.h. die Tabelle kann zum Beispiel im Editor beliebig modifiziert werden, sie kann abgespeichert und ausgedruckt werden. In Tab. 2 ist eine solche Tabelle dargestellt.

Benchid	Bench-time	Total-time	CPU-time	System-time	Wait-time	Idle-time
R-D241	170.41	1363.29	1299.94	15.47	931.80	47.88
R-D242	187.26	1498.05	1265.43	11.31	1160.44	221.31
R-D243	171.95	1375.59	1298.09	17.11	963.85	60.39
R-D244	184.99	1479.95	1248.35	57.24	1000.37	174.36
R-D245	174.11	1392.87	1229.47	49.23	963.64	14.17

Tab. 2. Beispiel einer PAR-Bench-Tabelle

Darin entspricht die *Benchid* der eindeutigen Kennung einer Benchmark-Messung. Sie ist identisch mit dem Dateityp der entsprechenden BENCHDAT-Datei. Bei den beiden Benchmarks R-D242 und R-D244 sind die Werte der *Bench-time* gegenüber den übrigen Benchmarks erhöht, d.h. der Systemdurchsatz ist bei diesen beiden Benchmarks kleiner. Die Ursache dafür findet sich in den ebenfalls erhöhten Werten der *Idle-time* und der *Wait-time*. Die Abweichungen in den Spalten *CPU-time* und *System-time* sind wesentlich kleiner, d.h. diese beiden Größen haben einen geringeren Einfluß auf den verminderten Systemdurchsatz.

Nach der Rückkehr ins System GRANSYS kann die Spezifikation und die Erstellung einer weiteren Tabelle vorgenommen werden, oder es wird in die erste Auswahlenebene verzweigt.

5.2. Die Auswertung der SAR-Informationen

Alle bisherigen Auswertungen basieren auf Informationen, die sich auf einzelne Jobs beziehen. Im Gegensatz dazu werden während der Ausführung der Benchmarks mit dem Systemmonitor SAR Informationen über die Systemaktivitäten des gesamten Systems gesammelt. Im Betriebssystem UNICOS werden alle Systemaktivitäten durch *system calls* (Systemroutinen-Aufrufe) realisiert. Der Systemmonitor SAR zählt kostenfrei alle *system calls* und gibt so ein Maß für die Systemaktivitäten an. Weitere Informationen beziehen sich auf die Nutzung des Erweiterungsspeichers, der beim *swapping* als schneller Hintergrundspeicher verwendet wird, und auf die Größe der *run queue* und der *swap*

queue. Als Maß für die Auslastung der Prozessoren wird die Prozessorzeit der Benutzerprogramme (*cpu time*) und die des Betriebssystems (*system time*) gemessen. Desweiteren wird die *idle time* angegeben, das ist die Zeit, in der Prozessoren nicht genutzt werden.

```
sn1011 sn1011 5.1.10 hlrz.18 CRAY Y-MP      08/22/90

13:15:36 ...
      %usr    %sys    %wsem    #locks    %idle
      ...
      device    reads    writes    total
runq-sz %runocc swpq-sz %swpocc
bread/s lread/s %rcache bwrit/s lwrit/s %wcache pread/s pwrit/s
swpin/s bswin/s xswin/s swpot/s bswot/s xswot/s pswch/s
scall/s sread/s swrit/s  fork/s   exec/s rchar/s wchar/s
      ...

13:16:38 ...
      98        2        0        0        0
      ...
      SSD-VHSP    1144    335    1479
      40-2-22A      5      20      25
      40-2-23A     10     132     142
      16.5      116     5.1      39
      13       742      98     164     238      31      46      0
      4.90    839.7    0.68    11.12    850.9     0.36     278
      1915     171     124    18.60    20.34 1245537    41561
      ...

13:16:40 ...
      100        0        0        0        0
      ...
      SSD-VHSP    2239    275    2514
      49-1-20A     16      0      16
      40-2-22A     16      33      49
      40-2-23A     45     130     175
      S49-0123     64      0      64
      17.0      100     6.1      53
      34      1063     97     185     284      35      65      0
      4.50   1133.0    0.50     9.78   1232.7     0.67     400
      2499     211     124    28.49    34.99 1270923    37017
      ...
```

Abb. 22. Die vom Systemmonitor gesammelten Daten (auszugsweise)

Die Ermittlung dieser Daten wird für vorgebbare Zeitintervalle wiederholt. Dabei werden jeweils die Systemaktivitäten innerhalb eines Zeitintervalls angegeben. Im Rahmen dieser Arbeit werden die Informationen des SAR in bezug auf die gesamte Dauer eines Benchmarks betrachtet. Da Start- und Endzeitpunkt eines Benchmarks nicht genau vorhersagbar sind, werden die Zeitintervalle mit 2 Sekunden vorgegeben³¹, so daß die

³¹ Aus Vergleichsmessungen geht hervor, daß ein Auslesen der Informationen des SAR im Zweisekundentakt keinen meßbaren Einfluß auf das Zeitverhalten eines Benchmarks hat.

Auswertung in bezug auf Start- und Endzeitpunkt mit ausreichender Genauigkeit durchgeführt werden kann. In Abb. 22 sind diese Daten auszugsweise wiedergegeben. Die Bedeutung der Daten wird im nächsten Kapitel erläutert. Das Programm BENCHGEN initialisiert den Systemmonitor SAR, und die ermittelten Daten werden als Datei im Hintergrundspeicher der Cray abgelegt.

5.2.1. Die Analysen der SAR-Informationen

Da die relevanten Daten jeweils für aufeinanderfolgende Zeitintervalle ermittelt wurden, ist für eine Auswertung in bezug auf die gesamte Dauer eines Benchmarks eine Weiterverarbeitung der Daten notwendig. Zu diesem Zweck steht mit dem SAR-Kommando [Cra2011] eine UNICOS-Systemfunktion zur Verfügung, die eine Zusammenfassung der Daten für ein vorgebbares Zeitintervall erlaubt. Dabei werden Start- und Endzeitpunkt eines Benchmarks angegeben, und die resultierenden Daten der jeweiligen Messung werden in einer Datei gespeichert. Eine solche SARDAT-Datei ist in Abb. 23 auszugsweise dargestellt.

```

...
13:15:42      %usr      %sys      %wsem      #locks      %idle
13:18:42          93          2          0          0          4
...
13:15:42 device      reads      writes      total
13:18:42 SSD-VHSP 237273 405387 642660
          49-1-20A 60791 60207 120998
          40-2-22A 60506 59914 120420
          40-2-23A 60746 60230 120976
          S49-0123 60377 59904 120281

13:15:42 runq-sz %runocc swpq-sz %swpocc
13:18:42      16.4      100      4.7      51

13:15:42 bread/s lread/s %rcache bwrit/s lwrit/s %wcache pread/s pwrit/s
13:18:42          8      335      98      55      124      56      12      0

13:15:42 swpin/s bswin/s xswin/s swpot/s bswot/s xswot/s pswch/s
13:18:42      4.71 1435.8      0.75      4.83 1451.3      0.58      132

13:15:42 scall/s sread/s swrit/s fork/s exec/s rchar/s wchar/s
13:18:42      729      86      87      6.23      5.96 1039157      23513

...

```

Abb. 23. Beispiel einer SARDAT-Datei (auszugsweise)

Da die Erstellung der SARDAT-Dateien unter UNICOS erfolgt, müssen diese Dateien bereits vor dem Aufruf des Systems GRANSYS bereitgestellt werden.

Bei der Analyse der SAR-Informationen werden die Daten mehrerer Messungen in Form von Tabellen gegenübergestellt. Die Erstellung der *SAR-Tabellen* erfolgt analog zur Erstellung der *PAR-Bench-Tabellen*. Die zu vergleichenden Messungen werden aufgrund

ihrer Kennung ausgewählt, und die getroffene Auswahl steht als Default-Auswahl für weitere Analysen zur Verfügung. Als Eingabedateien dienen die SARDAT-Dateien, die aufgrund der Benchmark-Kennung identifiziert werden.

Auch die Auswahl der gewünschten Daten wird in einer Datei gespeichert und steht für die Erstellung weiterer Tabellen als Default zur Verfügung.

Die aus den SARDAT-Dateien extrahierten Daten werden wiederum als GML-Tabelle formatiert und können mit Hilfe des Systems GML weiterverarbeitet, gespeichert oder ausgedruckt werden. In Tab. 3 sind Beispiele für *SAR-Tabellen* mit den wichtigsten Daten dargestellt.

Benchid	scall	sread	swrit	fork	exec	ssd	lread	lwrit
EA250A2	131220	15480	15660	1121	1072	42660	60300	22320
EA250B2	115740	14040	14580	790	813	34157	54360	20160
EA250A5	128148	14691	15576	1127	1065	42971	59826	21948
EA250B5	118193	13575	15204	870	885	36453	56472	20996
EA250A10	130500	15300	16380	1105	1074	44823	63360	23040
EA250B10	121451	13394	16109	1015	1011	41142	60273	22444

Benchid	runq	%runq	swpq	%swpq	swpin	swpot	pswch
EA250A2	16.4	100	4.7	51	847	869	23760
EA250B2	16.9	100	4.9	52	869	912	21240
EA250A5	16.5	100	4.4	51	798	815	23541
EA250B5	16.8	100	4.2	55	821	847	21177
EA250A10	16.4	100	4.2	51	810	815	24120
EA250B10	16.4	100	4.1	49	816	821	21720

Benchid	usr	%usr	sys	%sys	idle	%idle
EA250A2	1339.20	93	28.80	2	57.60	4
EA250B2	1353.60	94	28.80	2	57.60	4
EA250A5	1345.20	95	28.32	2	42.48	3
EA250B5	1361.12	94	28.96	2	57.92	4
EA250A10	1339.20	93	28.80	2	72.00	5
EA250B10	1346.64	93	28.96	2	72.40	5

Tab. 3. Beispiele für SAR-Tabellen

Die Daten in den einzelnen Spalten beziehen sich auf die gesamte Dauer des jeweiligen Benchmarks und haben die folgenden Bedeutungen:

- **Benchid:**
Die *Benchid* entspricht der eindeutigen Kennung einer Benchmark-Messung. Sie ist identisch mit dem Dateityp (*filetype*) der entsprechenden SARDAT-Datei.
- **scall:**
Im Betriebssystem UNICOS werden alle Systemaktivitäten von Systemroutinen durchgeführt. Mit *scall* wird die Anzahl aller *System calls* (Systemroutinen-Aufrufe) angegeben.

- **sread, swrit, fork, exec:**
Diese Größen geben die Häufigkeit spezifischer *system calls* an. Die Angaben *sread* und *swrit* entsprechen der Anzahl der Lese- bzw. Schreibzugriffe mit Hilfe von *system calls*. Mit *fork* wird ein neuer Prozess erstellt, und mit *exec* wird ein Prozess gestartet.
- **ssd:**
Die Größe *ssd* ist ein Maß für die Nutzung des Erweiterungsspeichers der Cray. Es wird die Anzahl der Datenübertragungen angegeben, wobei je Übertragung 4096 Bytes gelesen oder geschrieben werden.
- **lread, lwrit:**
Diese beiden Größen geben die Anzahl der Lese- bzw. Schreibzugriffe auf den *system buffer* (Pufferspeicher) an. Dieser wird bei dem benutzten System als Pufferspeicher zwischen dem Hauptspeicher bzw. dem Erweiterungsspeicher und den Hintergrundspeichern genutzt.
- **runq, %runq:**
In diesen beiden Spalten wird die Größe der *run queue*, d.h. die mittlere Anzahl der Prozesse im Hauptspeicher, und der prozentuale Anteil der davon zur Ausführung bereiten Prozesse angegeben.
- **swpq, %swpq:**
swpq ist die Größe der *swap queue*, d.h. die mittlere Anzahl der Prozesse, die durch *swapping* aus dem Hauptspeicher verdrängt wurden und auf ein erneutes Laden in den Hauptspeicher warten. Zusätzlich wird der prozentuale Anteil der davon zur Ausführung bereiten Prozesse angegeben.
- **swpin, swpot:**
Diese beiden Größen geben die Anzahl der Prozesse an, die durch *swapping* in den Speicher geladen bzw. aus dem Speicher verdrängt wurden.
- **pswch:**
Die Größe *pswch* gibt die Anzahl der *process switches* an, das sind alle Wechsel in der Kontrollfunktion von Prozessen über Prozessoren.
- **usr, %usr, sys, %sys, idle, %idle:**
Die Angaben in dieser Tabelle entsprechen den Angaben in den *PAR-Bench-Tabellen* für die *CPU-time*, die *System-time* und die *Idle-time*³². Die Prozentangaben beziehen sich auf den jeweiligen Anteil an der gesamten Prozessorenzeit (*Total-time*).
Die Ermittlung dieser Zeitgrößen mit dem System *PAR-Bench* (siehe Kapitel 5.1.1.1) wurde für jeden Job separat durchgeführt, so daß die Endzeitpunkte der einzelnen Jobs berücksichtigt sind. Dadurch sind in der *Idle-time* des Systems *PAR-Bench*

³² Da diese Zeitgrößen sowohl vom System *PAR-Bench* als auch vom Systemmonitor SAR ermittelt werden, erfolgt eine Unterscheidung aufgrund der Schreibweise. Während z.B. mit *CPU-time* die von *PAR-Bench* ermittelte Größe bezeichnet wird, ist mit *usr* die entsprechende Größe des SAR gemeint. Die Schreibweise *cpu time* korrespondiert mit der allgemeinen Bezeichnung dieser Größe.

keine Anteile enthalten, die durch die leerlaufende *run queue* am Ende der Benchmarks hervorgerufen werden. Im Gegensatz dazu wird die Ermittlung der Zeitgrößen mit Hilfe des Systemmonitors SAR unabhängig von den Endzeitpunkten der einzelnen Jobs für einen mittleren Endzeitpunkt des Benchmarks durchgeführt, d.h. in der Angabe *idle* sind zusätzliche unerwünschte Anteile enthalten. Deswegen ist in den *SAR-Tabellen* der prozentuale Anteil der *idle time* größer als in den *PAR-Bench-Tabellen*.

6. Erfahrungen mit der Programmierumgebung

Das Benchmark-System *PAR-Bench* kann für eine Vielzahl von Untersuchungen verwendet werden: Aufgrund der Flexibilität bezüglich charakteristischer Parameter können weitgehend beliebige Benchmarks konfiguriert werden, die sowohl als reproduzierbare Arbeitslast zum Vergleich verschiedener Systeme als auch zur Untersuchung spezieller Systemkomponenten geeignet sind. In diesem Kapitel werden Tests in bezug auf die Genauigkeit des Systems *PAR-Bench* und beispielhaft erste Anwendungen des Systems beschrieben. Darin wird insbesondere die Cray Autotasking-Implementation auf einer Cray Y-MP8/832 untersucht:

- Um die tatsächlichen Werte von Parallelitätsgrad und Load-Balance-Faktor der vom System *PAR-Bench* erzeugten Benchmark-Programme zu untersuchen, werden Speedup-Messungen bei unterschiedlichen Vorgaben für diese Parameter durchgeführt. Durch Vergleich der gemessenen Werte für den Speedup mit den erwarteten Werten wird die Einhaltung der Parametervorgaben getestet.
- In den beschriebenen Anwendungen werden die wechselseitigen Einflüsse von parallelen Programmen und Hintergrundlast in einer Multiprogramming-Umgebung betrachtet. Um die Leistungsfähigkeit der Systeme Cray X-MP und Cray Y-MP zu vergleichen, wird ein Teil der Benchmarks auf beiden Systemen durchgeführt.

Bei allen durchgeführten Untersuchungen wurden dieselben Kernfolgen verwendet, es wurde also nur ein Programmablauf im Generate-Modus durchgeführt, der die Kernfolgen für 20 Benchmark-Programme erzeugt hat. Um einen Bezug auf die durchgeführten Benchmarks zu gewährleisten, wurde auch bei den Speedup-Messungen eines dieser Benchmark-Programme ('BP001') verwendet. Da das charakteristische Verhalten der benutzten Benchmark-Programme in bezug auf den Speedup gleich ist, gelten die erzielten Ergebnisse auch für die übrigen Benchmark-Programme. In den folgenden Beschreibungen der Messungen sind alle Zeitgrößen in Sekunden angegeben.

6.1. Die Speedup-Messungen

Bei parallelen Programmen können in Abhängigkeit vom Parallelitätsgrad und vom Load-Balance-Faktor unterschiedliche Speedups erzielt werden. Um die tatsächlichen Werte dieser beiden Parameter in den von *PAR-Bench* erzeugten Benchmark-Programmen zu ermitteln, wird der erzielte Speedup gemessen. Zu diesem Zweck wird jeweils ein Programm auf einer dedizierten Maschine ausgeführt, sowohl sequentiell als auch parallel mit jeweils anderen Vorgaben für den Parallelitätsgrad und den Load-Balance-Faktor. Zur Ermittlung des Speedups wird jeweils die Ausführungszeit (*real time*) gemessen. Die gemessenen Speedups werden mit den aufgrund der Parametervorgaben erwarteten Speedups verglichen. Der in Abhängigkeit vom Parallelitätsgrad PG erwartete Speedup mit $NPROC$ Prozessoren wird mit Hilfe von *Amdahl's law* ermittelt (siehe Kapitel 2).

$$S_{LB=100}(PG, NPROC) = \frac{T_1}{T_{NPROC}} = \frac{1}{(1 - PG) + \frac{PG}{NPROC}}$$

Der Index $LB = 100$ gibt an, daß diese Formel nur bei einem Load-Balance-Faktor von 100% gilt. Der Einfluß des Load-Balance-Faktors LB auf den erwarteten Speedup ist linear (siehe Kapitel 4.2.2). Für $PG = 100$ gilt bei paralleler Ausführung mit $NPROC$ Prozessoren:

$$S_{PG=100}(LB, NPROC) = LB \times NPROC$$

Zur Ermittlung des erwarteten Speedups in Abhängigkeit vom Parallelitätsgrad und vom Load-Balance-Faktor wird die Ausführungszeit für diesen Fall bestimmt. Dabei ist zu beachten, daß der Load-Balance-Faktor nur einen Einfluß auf den parallelen Teil des Programms hat. Im Unterschied zu *Amdahl's law* werden im parallelen Teil nicht $NPROC$ Prozessoren sondern im zeitlichen Mittel nur $(LB \times NPROC)$ Prozessoren genutzt. Damit ergibt sich die Ausführungszeit $T_{(LB \times NPROC)}$:

$$T_{(LB \times NPROC)} = (1 - PG) T_1 + PG \frac{T_1}{LB \times NPROC}$$

Es folgt für den erwarteten Speedup als Funktion von PG , LB und $NPROC$:

$$S(PG, LB, NPROC) = \frac{T_1}{T_{(LB \times NPROC)}} = \frac{1}{(1 - PG) + \frac{PG}{LB \times NPROC}}$$

Für die durchgeführten Messungen wird in Tab. 4 die Ausführungszeit (Real time) in Sekunden, der gemessene Speedup und der erwartete Speedup angegeben.

NPROC	PG	LB	Real time	gemessener Speedup	erwarteter Speedup
1	0	0	113.09	1.00	1.00
4	100	100	29.22	3.87	4.00
4	99	100	30.17	3.75	3.88
4	98	100	30.62	3.69	3.77
4	97	100	31.84	3.55	3.67
4	96	100	32.91	3.43	3.57
4	95	100	33.43	3.38	3.48
4	90	100	37.46	3.02	3.08
4	80	100	46.15	2.45	2.50

Tab. 4. Der Speedup in Abhängigkeit vom Parallelitätsgrad

Bei allen Messungen wurde das Benchmark-Programm 'BP001' verwendet. Bei sequentieller Ausführung betrug die Speicherzugriffsrate 70 MMREFS (*Million Memory REferences per Second*), die Gleitpunktverarbeitungsrate betrug 100 MFLOPS (*Million FLoating-point OPerations per Second*). Um eine weitgehend vollständige Parallelisierbarkeit zu gewährleisten, wurde ein Benchmark-Programm gewählt, das keine Ein-/Ausgaben durchführt. Die parallelen Programme wurden mit den Parametervorgaben $NPROC = 4$ und $LB = 100$ ausgeführt. Die Vorgabe für den Parallelitätsgrad wurde im Intervall von 80 bis 100% variiert.

Die Werte des gemessenen Speedups stimmen mit denen des erwarteten Speedups gut überein. Abb. 24 stellt diese beiden Größen in Abhängigkeit vom vorgegebenen Parallelitätsgrad graphisch dar. Die geringen Abweichungen von 2 bis 4% werden durch eine

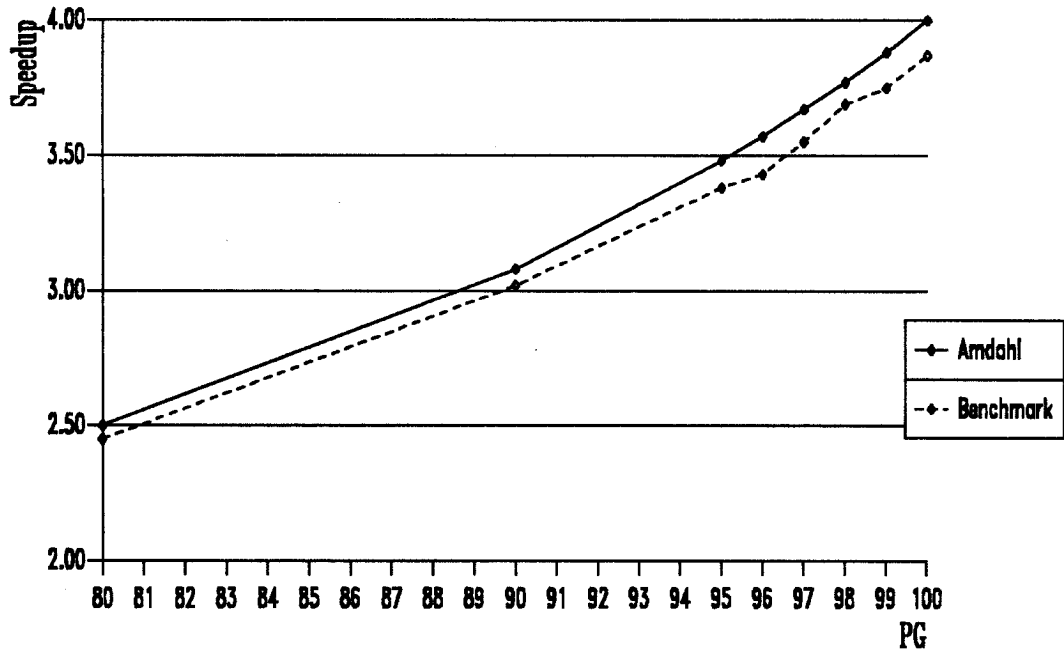


Abb. 24. Der Speedup des Benchmarks und der Speedup nach Amdahl

Erhöhung der benötigten Prozessorzeit verursacht. Die Parallelisierung des Programms mit Hilfe der Autotasking-Implementation erfordert zusätzlichen Aufwand. Außerdem bewirkt die verkürzte Ausführungszeit des Programms eine Erhöhung der Speicherzugriffsrate. Im Vergleich zur sequentiellen Ausführung treten dadurch mehr Speicherzugriffskonflikte (*memory contention*, siehe Kapitel 2) auf, d.h. die mittlere Dauer eines Speicherzugriffs ist größer. Der Einfluß der durch diese beiden Effekte erhöhten Prozessorzeit auf den erzielten Speedup wird offensichtlich, wenn man den Quotienten von genutzter Prozessorzeit T_C und Ausführungszeit T_R bildet, der im folgenden als Servicefaktor SF bezeichnet wird.

$$SF = \frac{\text{CPU time}}{\text{Real time}} = \frac{T_C}{T_R}$$

Mit der erwarteten Ausführungszeit

$$T_R = (1 - PG) T_C + PG \frac{T_C}{LB \times NPROC}$$

ist der erwartete Servicefaktor identisch mit dem erwarteten Speedup:

$$SF = \frac{T_C}{T_R} = S(PG, LB, NPROC) = \frac{1}{(1 - PG) + \frac{PG}{LB \times NPROC}}$$

In Tab. 5 werden die Ausführungszeit (Real time), die Prozessorzeit (CPU time), der gemessene Servicefaktor sowie der erwartete Servicefaktor angegeben.

NPROC	PG	LB	Real time	CPU time	gemessener Servicefaktor	erwarteter Servicefaktor
1	0	0	113.09	113.03	1.00	1.00
4	100	100	29.22	116.44	3.98	4.00
4	99	100	30.17	116.06	3.85	3.88
4	98	100	30.62	116.05	3.79	3.77
4	97	100	31.84	117.49	3.69	3.67
4	96	100	32.91	116.19	3.53	3.57
4	95	100	33.43	116.40	3.48	3.48
4	90	100	37.46	115.81	3.09	3.08
4	80	100	46.15	115.85	2.51	2.50

Tab. 5. Der Servicefaktor in Abhängigkeit vom Parallelitätsgrad

Die zum Teil sehr gute Übereinstimmung von gemessenem und erwartetem Servicefaktor beweist, daß der vorgegebene und der tatsächliche Parallelitätsgrad nahezu identisch sind. Die Abweichungen des gemessenen vom erwarteten Speedup beruhen also fast ausschließlich auf dem bei der Parallelisierung implizit vorhandenen Overhead und auf der erhöhten Anzahl der Speicherzugriffskonflikte. Unabhängig von der Erhöhung der Prozessorzeit, die in *Amdahl's law* nicht berücksichtigt wird, gibt der Servicefaktor ein Maß für den tatsächlichen Parallelitätsgrad an. Dabei kann der gemessene Servicefaktor vom erwarteten Servicefaktor sowohl geringfügig nach oben als auch nach unten abweichen. In beiden Fällen wird dies durch eine Abweichung des tatsächlichen Parallelitätsgrades von der Vorgabe hervorgerufen, was jedoch vernachlässigt werden kann. Abb. 25 stellt den gemessenen Servicefaktor und den erwarteten Servicefaktor in Abhängigkeit vom vorgegebenen Parallelitätsgrad dar.

Als zusätzliches Ergebnis dieser Messungen wurde eine Erhöhung der Prozessorzeit durch die Parallelisierung mit Autotasking um 2 bis 4% festgestellt. Diese Erhöhung kann jedoch in Abhängigkeit von den Speicherzugriffskonflikten und von den jeweiligen Parallelisierungs-Primitiven auch andere Werte annehmen.

In bezug auf den Load-Balance-Faktor wurden ebenfalls vergleichende Messungen durchgeführt. Bei einem Parallelitätsgrad von 100% wurde der Load-Balance-Faktor im Intervall von 80 bis 100% variiert. Tab. 6 stellt die relevanten Daten dar.

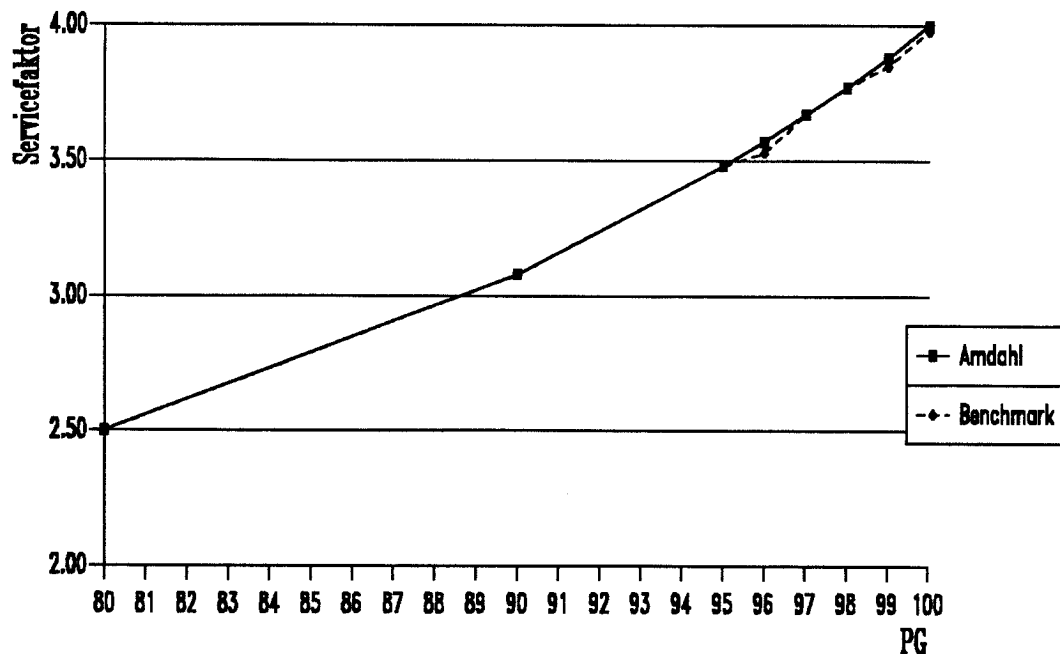


Abb. 25. Der Servicefaktor des Benchmarks im Vergleich zum Speedup nach Amdahl

NPROC	PG	LB	Real time	gemessener Speedup	erwarteter Speedup (Servicefaktor)	CPU time	gemessener Servicefaktor	wsem%
1	0	0	113.09	1.00	1.00	113.03	1.00	0.00
4	100	100	29.22	3.87	4.00	116.44	3.98	0.11
4	100	99	29.48	3.84	3.96	116.29	3.94	1.12
4	100	98	29.74	3.80	3.92	116.11	3.90	2.04
4	100	97	30.30	3.73	3.88	116.29	3.84	3.10
4	100	96	30.55	3.70	3.84	116.88	3.83	4.07
4	100	95	31.10	3.64	3.80	116.47	3.75	5.85
4	100	90	32.40	3.49	3.60	116.28	3.59	10.06
4	100	80	36.42	3.11	3.20	116.14	3.19	20.07

Tab. 6. Der Speedup, der Servicefaktor und die Wartezeit der Prozessoren in Abhängigkeit vom Load-Balance-Faktor

Auch hier ergeben sich Abweichungen des gemessenen vom erwarteten Speedup von 2 bis 4%. Aus den Werten für den Servicefaktor geht wiederum hervor, daß diese Abweichungen im wesentlichen durch die Parallelisierung hervorgerufen werden, nicht jedoch durch das System *PAR-Bench*. Ein weiteres Indiz für die Genauigkeit von *PAR-Bench* ist die gemessene Wartezeit der Prozessoren (*time waiting on semaphores*, siehe Kapitel 2), die bei diesen Messungen durch unbalancierte Programmteile hervor-

gerufen wird. Während dieser Wartezeit wird das Programm nur von einem Teil der Prozessoren ausgeführt, und die übrigen Prozessoren warten bis zur nächsten Synchronisation. In der Spalte 'wsem%' wird der prozentuale Anteil der Wartezeit an der gesamten Prozessorzeit angegeben³³. Ein Anteil der Wartezeit von z.B. 10% bedeutet, daß im zeitlichen Mittel 10% aller Prozessoren gewartet haben, und entspricht einem Load-Balance-Faktor von 90%. Der Vergleich der Wartezeit mit der jeweiligen Vorgabe für den Load-Balance-Faktor bestätigt die aufgrund des Servicefaktors gewonnenen Ergebnisse.

In einer weiteren Meßreihe wurden sowohl Parallelitätsgrad als auch Load-Balance-Faktor variiert. Tab. 7 zeigt die dabei ermittelten Daten.

NPROC	PG	LB	Real time	gemessener Speedup	erwarteter Speedup = Service-faktor	CPU time	Service-faktor
1	0	0	113.09	1.00	1.00	113.03	1.00
4	100	100	29.22	3.87	4.00	116.44	3.98
4	95	95	35.12	3.21	3.33	116.28	3.31
4	90	90	40.73	2.78	2.86	115.91	2.85
4	85	85	46.37	2.44	2.50	115.79	2.50
4	80	80	52.09	2.17	2.22	115.74	2.22
4	80	80	52.09	2.17	2.22	115.74	2.22

Tab. 7. Der Speedup und der Servicefaktor in Abhängigkeit von Parallelitätsgrad und Load-Balance-Faktor

Die Abweichungen des gemessenen Speedups vom erwarteten Speedup liegen wiederum bei 2 bis 4%, d.h. der durch die Parallelisierung erzeugte Overhead ist gleich geblieben. Wie der gemessene Servicefaktor bestätigt, stimmen die tatsächlichen Parameterwerte auch bei gleichzeitig verringertem Parallelitätsgrad und Load-Balance-Faktor sehr genau mit den Vorgaben überein.

6.2. Parallele Programme und Hintergrundlast

Die im folgenden beschriebenen Benchmarks stellen erste Anwendungen des Systems *PAR-Bench* dar. Darin werden die wechselseitigen Einflüsse von parallelen Programmen und Hintergrundlast in einer Multiprogramming-Umgebung untersucht.

In allen Benchmarks wurden dieselben Kernfolgen verwendet, d.h. es wurde nur ein Programmablauf im Generate-Modus durchgeführt. In diesem Lauf wurden 20 Benchmark-Programme generiert, die Parametervorgaben waren für jeweils zwei aufeinanderfolgende Programme identisch. Abb. 26 zeigt die entsprechende Eingabedatei für das Programm *BENCHGEN*.

³³ In der Angabe 'CPU time' in den Tabellen ist nur der Anteil der Prozessorzeit enthalten, in der die Prozessoren tatsächlich genutzt werden. Die gesamte Prozessorzeit enthält darüber hinaus die Wartezeit der Prozessoren.

MODE= 'GEN'													
#,DEL, PR, C, SEC,VE, MM, FL, IO, SIZE, PRNAME, PG, LB,NPROC													
2, 20, 10, 1, 120, 0, 70, 100, 0, 2000, 'bench', 0., 0., 1													
2, 20, 10, 1, 120, 0, 150, 80, 0, 2000, 'bench', 0., 0., 1													
2, 20, 10, 1, 120, 0, 70, 100, 0, 2000, 'bench', 0., 0., 1													
2, 20, 10, 1, 120, 0, 150, 80, 0, 2000, 'bench', 0., 0., 1													
2, 50, 10, 1, 90, 0, 100, 150, 0, 2000, 'bench', 0., 0., 1													
2, 50, 10, 1, 90, 0, 100, 150, 0, 2000, 'bench', 0., 0., 1													
2, 50, 10, 1, 90, 0, 120, 70, 0, 2000, 'bench', 0., 0., 1													
2, 50, 10, 1, 90, 0, 120, 70, 0, 2000, 'bench', 0., 0., 1													
2, 80, 10, 1, 60, 0, 100, 100, 0, 2000, 'bench', 0., 0., 1													
2, 80, 10, 1, 60, 0, 70, 80, 0, 2000, 'bench', 0., 0., 1													

Abb. 26. Die Eingabedatei für das Programm BENCHGEN im Generate-Modus

Als Referenzmessung wurde diese Arbeitslast im Execute-Modus mit unveränderten Parametervorgaben ausgeführt, d.h. alle Benchmark-Programme wurden sequentiell ausgeführt. Um die Auswirkungen von parallelen Programmen in einer Multiprogramming-Umgebung zu untersuchen, wurden verschiedene Meßreihen durchgeführt, bei denen jeweils ein Teil der Benchmark-Programme eines Benchmarks parallel ausgeführt wurde.

6.2.1. Parallele Programme mit geringer Parallelität

In der ersten Meßreihe wurde jedes zweite der 20 Benchmark-Programme parallel ausgeführt, jedoch mit geringer Parallelität³⁴. Zunächst wurde der Parallelitätsgrad mit 70% vorgegeben, und es wurden jeweils 4 Prozessoren angefordert. Abb. 27 stellt den sequentiellen Referenz-Benchmark und den Test-Benchmark mit den parallelen Programmen in Form von *Benchmark-Plots* gegenüber.

Die *Bench-time* bezeichnet die mittlere Dauer des gesamten Benchmarks und ist ein Maß für den Systemdurchsatz. In Relation zum Referenz-Benchmark wird beim Test-Benchmark, neben der extrem vergrößerten *Bench-time* sowohl bei den parallelen Programmen als auch bei der Hintergrundlast eine vergrößerte Ausführungszeit beobachtet. Die extreme Vergrößerung der *Bench-time* bedeutet eine ebenso starke Verschlechterung des Systemdurchsatzes. Die Ursachen dafür können durch Vergleich der übrigen Zeitgrößen ermittelt werden. Die Vergrößerung der *CPU-time*, die durch die parallelen Programme erzeugt wird (siehe Abb. 27), wird zum Teil durch den bei der Parallelisierung implizit vorhandenen Overhead verursacht (siehe Kapitel 6.1). Eine weitere Ursache ist der geringe Parallelitätsgrad der Benchmark-Programme. Die parallelen Programme können die vom Betriebssystem zugeteilten Prozessoren nur zeitweise nutzen, in den sequentiellen Programmteilen kann jeweils nur ein Prozessor genutzt werden.

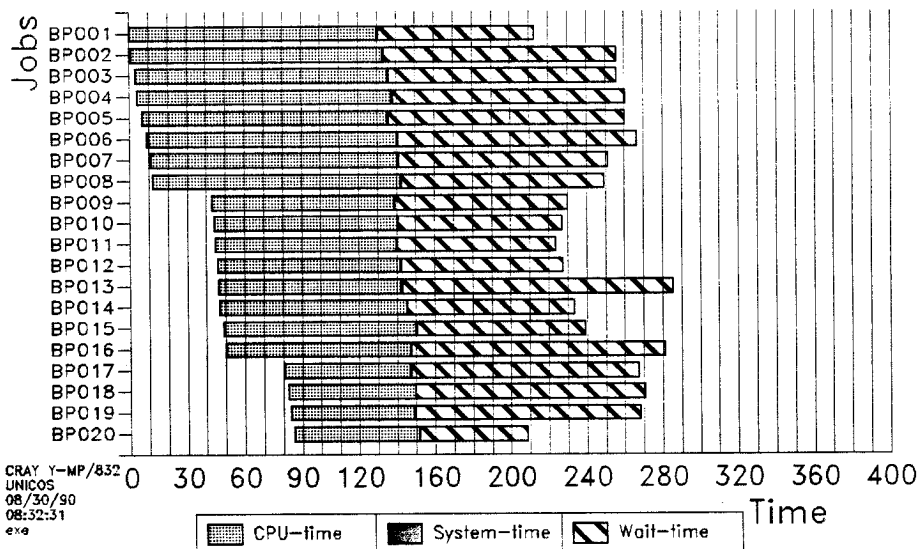
³⁴ Als 'Programme mit geringer Parallelität' werden hier sowohl Programme mit geringem Parallelitätsgrad als auch Programme mit geringem Load-Balance-Faktor bezeichnet, d.h., wenn z.B. eine der beiden Größen einen Wert von ca. 70% hat.

```

EX01 :
PR      PG      LB NPROC
20 Jobs 30      0.0  0.0  1 seq

Bench-time 269.70s
Total-time 2157.63s
CPU-time    2082.36s 96.51%
System-time 12.85s   0.60%
Wait-time   2126.10s 98.54%
Idle-time    62.42s  2.89%

```



CRAY Y-MP/832
UNICOS
08/30/90
08:32:31
exe

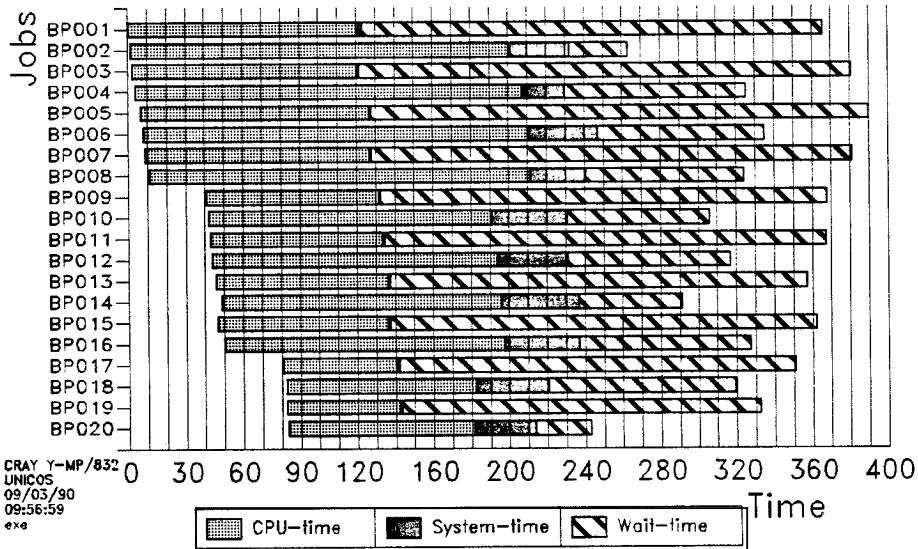
```

EA06 :
PR      PG      LB NPROC
10 Jobs 30      0.0  0.0  1 seq
10 Jobs 30      70.1 100.0 4

Bench-time 371.12s
Total-time 2968.98s
CPU-time    2553.38s 86.00%
System-time 359.63s 12.11%
Wait-time   3057.39s 102.98%
Idle-time    55.97s  1.89%

Parallel Jobs:
CPU-time    1596.31s 0.60
System-time 348.51s 0.73
Real-time   2675.43s

```



CRAY Y-MP/832
UNICOS
09/03/90
09:56:59
exe

Abb. 27. Der Referenz-Benchmark und ein Test-Benchmark mit 10 parallelen Programmen

Die nicht genutzten Prozessoren werden den Programmen jedoch nicht in jedem Fall entzogen³⁵, so daß jedes parallele Programm zusätzliche *CPU-time* benötigt.

Auch die Erhöhung der *System-time*, die ebenfalls durch die parallelen Programme verursacht wird, bewirkt eine Vergrößerung der *Bench-time*. In dieser Zeit werden die betreffenden Prozessoren vom Betriebssystem genutzt, um z.B. die Zuteilung der Prozessoren an die Programme zu verwalten. Aufgrund der parallelen Programme ist das Verhältnis von vorhandenen Prozessoren und aktiven Prozessen im Vergleich zum sequentiellen Referenz-Benchmark verkleinert, so daß der Verwaltungsaufwand steigt. Dadurch wird zumindest ein Teil der zusätzlichen *System-time* verursacht.

Die Verkürzung der *Idle-time* ist vergleichsweise gering und hat damit keinen großen Einfluß auf die *Bench-time*. Die Ursache für diese Verkürzung liegt wiederum bei den parallelen Programmen. Die mit Autotasking parallelisierten Programme können mit einer dynamisch wechselnden Anzahl von Prozessoren ausgeführt werden, die maximal der Anzahl der angeforderten Prozessoren entspricht (siehe Kapitel 4). Damit ist häufig die Nutzung von Prozessoren möglich, die von keinem anderen Programm benötigt werden und die somit *idle* wären.

Entsprechende Benchmarks wurden auch mit 6 bzw. 8 Prozessoren durchgeführt. In einem weiteren Benchmark mit 4 Prozessoren wurde anstelle des Parallelitätsgrads der Load-Balance-Faktor mit 70% vorgegeben. In Tab. 8 sind die relevanten Zeitgrößen von allen bisherigen Messungen zusammengestellt.

NPROC	PG	LB	Bench-time	CPU-time	System-time	Idle-time
1	0	0	269.70	2082.36	12.85	62.42
4	70	100	371.12	2553.38	359.63	55.97
6	70	100	439.84	2933.75	537.39	47.56
8	70	100	499.05	3443.83	506.07	42.49
4	100	70	375.11	2634.76	325.38	40.76

Tab. 8. Benchmarks mit parallelen Programmen geringer Parallelität im Vergleich mit dem Referenz-Benchmark

Bei einer Erhöhung der angeforderten Prozessorzahl verstärken sich die oben beschriebenen Effekte zum Teil erheblich. Die *Bench-time* erhöht sich in Relation zum sequentiellen Referenz-Benchmark im ungünstigsten Fall auf 185%. Bei geringer Parallelität aufgrund des Load-Balance-Faktors werden ähnliche Effekte beobachtet.

In einer weiteren Meßreihe wurden entsprechende Benchmarks mit leicht erhöhter Priorität bei den parallelen Programmen durchgeführt. Dabei traten die gleichen Effekte auf, in zum Teil noch leicht verstärktem Ausmaß.

Die durchgeführten Benchmarks haben gezeigt, daß die Ausführung einer verhältnismäßig großen Zahl von parallelen Programmen in einer Multiprogramming-Umgebung nicht unproblematisch ist. Dabei stellt insbesondere gerade die geringe Parallelität der Programme ein Problem dar. Neben einer extremen Verringerung des Systemdurch-

³⁵ Die Realisierung des Autotasking-Konzeptes und die entsprechende Einbettung ins Betriebssystem sind in der nächsten Version des UNICOS (Version 6.0) verändert und sollen in derartigen Situationen eine effizientere Nutzung der Prozessoren ermöglichen.

satzes wurde sowohl bei den parallelen Programmen als auch bei der Hintergrundlast eine Vergrößerung der Ausführungszeit beobachtet.

6.2.2. Parallele Programme mit hoher Parallelität

In den in diesem Kapitel beschriebenen Meßreihen werden alle parallelen Programme mit hoher Parallelität ausgeführt, d.h. sowohl der Parallelitätsgrad als auch der Load-Balance-Faktor werden mit 100% vorgegeben. Entsprechend der Tab. 4 auf Seite 72 wurde mit solchen Programmen in dedizierter Umgebung bei Nutzung von 4 Prozessoren ein Speedup von 3.87 erzielt; dies ist ein Wert, der auch bei vielen Anwendungsprogrammen erreicht werden kann, und damit sind die resultierenden Benchmarks durchaus typisch für reale Problemstellungen. In allen Benchmarks wird jeweils nur ein Testprogramm parallel ausgeführt, die übrigen 19 Programme werden in bezug auf den Referenz-Benchmark mit unveränderten Parametervorgaben, d.h. sequentiell, ausgeführt.

In der ersten Meßreihe haben alle Benchmark-Programme eines Benchmarks die gleiche Priorität. Das Testprogramm wird in drei verschiedenen Benchmark-Läufen mit 4, 6 bzw. 8 Prozessoren ausgeführt. Abb. 28 stellt den Referenz-Benchmark und den Benchmark, bei dem das Testprogramm 8 Prozessoren anfordert, gegenüber. In den darin enthaltenen Tabellen 'Serial Jobs' bzw. 'Parallel Jobs' sind die jeweiligen Daten des Testprogramms ('BP009') angegeben.

Die Parallelisierung des Programms 'BP009' hat eine sehr geringe Vergrößerung der *Bench-time* um ca. 1% und damit eine ebenso geringe Verschlechterung des Systemdurchsatzes bewirkt. Die Vergrößerung der *Bench-time* wird wiederum durch eine Vergrößerung der *CPU-time* und der *System-time* verursacht, d.h. es treten die schon in Kapitel 6.2.1 beobachteten Effekte auf, hier jedoch bei nur einem Programm. Der geringfügigen Durchsatzverschlechterung steht eine deutliche Beschleunigung des parallelen Testprogramms gegenüber. Tab. 9 gibt die relevanten Daten der drei Benchmarks sowie die Daten des Referenz-Benchmarks an.

NPROC	Benchmark				Job			
	Bench-time	CPU-time	System-time	Idle-time	CPU-time	System-time	Real-time	Real-time-Faktor
1	269.70	2082.36	12.85	62.42	95.66	0.50	186.78	1.00
4	267.71	2068.94	13.06	59.69	92.03	2.60	49.76	3.75
6	272.42	2074.59	30.81	73.97	94.41	19.03	43.25	4.31
8	272.00	2094.08	20.19	61.74	102.68	8.29	34.43	5.42

Tab. 9. Benchmarks mit einem parallelen Programm hoher Parallelität bei gleicher Priorität aller Programme und der Referenz-Benchmark

Die Daten in den vier rechten Spalten der Tabelle beziehen sich auf das Testprogramm. Der *Real-time*-Faktor ist der Quotient aus *Real-time* des Testprogramms bei sequentieller Ausführung und *Real-time* des Testprogramms bei paralleler Ausführung. Der *Real-time*-Faktor steigt mit der Anzahl der angeforderten Prozessoren auf einen Maximalwert von 5.42, während die Durchsatzverringerung für $NPROC = 6$ und $NPROC = 8$ nur ca. 1% beträgt. Für $NPROC = 4$ wird sogar eine verkürzte *Bench-time*

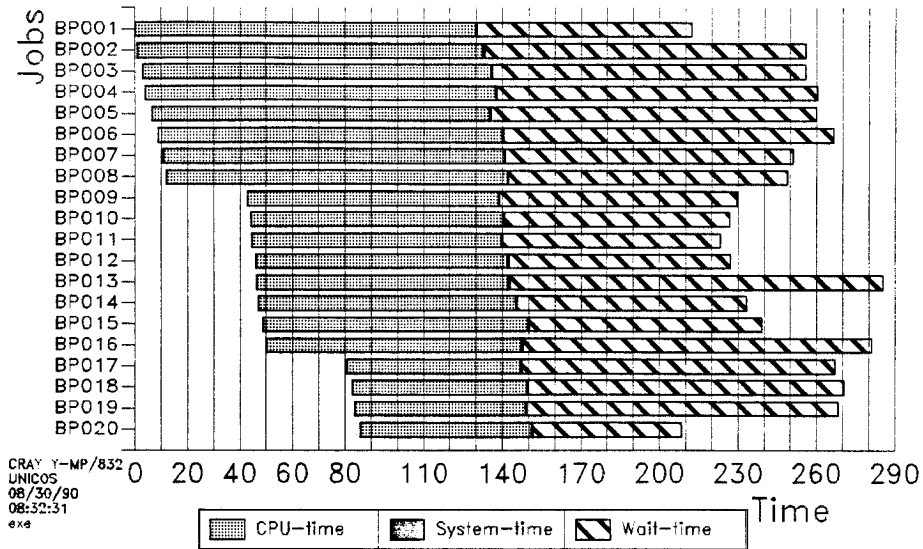
```

EX01 :
PR      PG      LB NPROC
20 Jobs 30      0.0  0.0  1 seq

Bench-time 269.70s
Total-time 2157.63s
CPU-time    2082.36s  96.51%
System-time 12.85s   0.60%
Wait-time   2126.10s  98.54%
Idle-time    62.42s   2.89%

Serial Jobs:
CPU-time    95.66s   0.51
System-time  0.50s   0.51
Real-time   186.78s

```



```

ED25 :
PR      PG      LB NPROC
19 Jobs 34      0.0  0.0  1 seq
1 Job  34  99.8 100.0  8

Bench-time 272.00s
Total-time 2176.01s
CPU-time    2094.08s  96.23%
System-time 20.19s   0.93%
Wait-time   2041.81s  93.83%
Idle-time    61.74s   2.84%

Parallel Jobs:
CPU-time    102.68s  2.98
System-time  8.29s   3.22
Real-time   34.43s

```

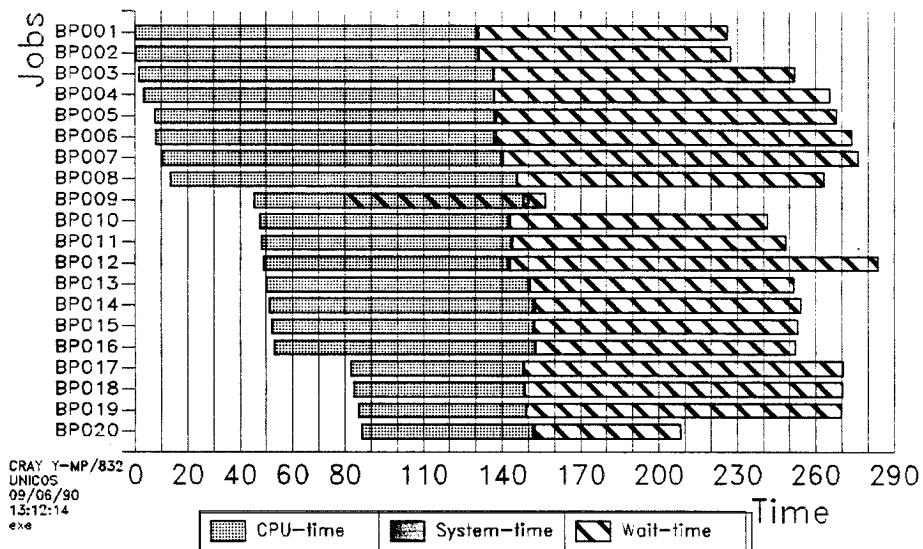


Abb. 28. Der Referenz-Benchmark und ein Test-Benchmark:
Ein paralleles Programm ('BP009') wird mit 8 Prozessoren ausgeführt.

und damit ein besserer Durchsatz erzielt. Diese Verkürzung wird durch den niedrigeren Wert der *CPU-time* bedingt. Die aufgrund der Parallelisierung erwartete Erhöhung der *CPU-time* wird offensichtlich von anderen, der Erhöhung entgegengesetzten Effekten übertroffen. Eine Ursache für die niedrigere *CPU-time* konnte aufgrund von zusätzlichen Messungen bestimmt werden. Dabei wurde die Anzahl der Speicherzugriffskonflikte der einzelnen Benchmark-Programme für den Referenz-Benchmark und für den Test-Benchmark mit $NPROC = 4$ mit Hilfe des HPM ermittelt. Beim Test-Benchmark traten insgesamt deutlich weniger Speicherkonflikte auf, was zumindest eine Ursache für die Verkürzung der *CPU-time* ist. Da die resultierende Verkürzung der *CPU-time* jedoch nicht bekannt ist, sind zur quantitativen Auswertung dieses Effekts weitere Untersuchungen notwendig.

Tab. 10 stellt die SAR-Informationen für die Benchmarks mit jeweils einem parallelen Programm bei gleicher Priorität aller Programme dar.

NPROC	usr	%usr	sys	%sys	idle	%idle
1	2073.60	96	21.60	1	64.80	3
4	2058.24	96	21.44	1	64.32	3
6	2067.20	95	43.52	2	65.28	3
8	2088.96	96	43.52	2	65.28	3

NPROC	runq	%runq	swpq	%swpq	swpin	swpot	pswch
1	13.8	100	4.0	69	291	315	17820
4	14.6	100	3.2	70	160	184	30284
6	14.8	100	3.2	69	182	201	146336
8	14.9	100	3.2	71	182	206	42976

NPROC	scall	sread	swrit	fork	exec	ssd	lread	lwrit
1	103140	13230	11610	866	872	238913	57240	21600
4	112024	14472	12596	903	919	197827	63248	23852
6	116960	14960	12784	957	979	199731	65280	24480
8	116144	14960	12784	946	968	224870	65008	24480

Tab. 10. Die SAR-Tabellen für die Benchmarks mit einem parallelen Programm und gleicher Priorität und der Referenz-Benchmark

Wie in Kapitel 5.2 erläutert, enthalten die mit Hilfe des SAR gemessenen Zeitgrößen *usr* (*cpu time*), *sys* (*system time*) und *idle* (*idle time*) einen systematischen Fehler, d.h. die mit Hilfe von *PAR-Bench* ermittelten Vergleichsgrößen sind genauer. Wie aus Tab. 10 hervorgeht, führen die mit Hilfe des SAR gemessenen Werte dieser Größen dennoch zu qualitativ ähnlichen Ergebnissen. Zum Beispiel ist die *cpu time* für $NPROC = 4$ leicht gesunken und steigt für $NPROC = 6$ und $NPROC = 8$ wieder an.

Die Anzahl der Datenübertragungen von und zum Erweiterungsspeicher (*ssd*) verhält sich qualitativ ähnlich wie die *cpu time*. Dasselbe Verhalten wird auch in bezug auf die Häufigkeit des *swapping* festgestellt (*swpin*, *swpot*). Da der Erweiterungsspeicher bei dem verwendeten System insbesondere beim *swapping* genutzt wird, entspricht die beobachtete Korrelation der Größen *ssd*, *swpin* und *swpot* den Erwartungen. In Verbindung mit der *cpu time* kann dieses Verhalten jedoch als eine weitere Ursache für die Verkürzung

der *cpu time* interpretiert werden. Diese Ursache tritt bei allen drei Test-Benchmarks auf, wird aber insbesondere bei $NPROC = 8$ von der Erhöhung der *cpu time* aufgrund der Parallelisierung überdeckt.

Die Anzahl der *process switches* (*pswch*) ist für $NPROC = 6$ deutlich erhöht. Der häufige Wechsel in der Kontrolle über die Prozessoren kann eine Ursache für die erhöhten Werte von *idle time* und *system time* bei diesem Benchmark sein³⁶.

Die übrigen Größen in Tab. 10 geben zum Teil ein Bild von der Konfiguration der Benchmarks wieder. Die Größen *fork* und *exec* geben z.B. die Anzahl der erstellten und ausgeführten Prozesse an. Wie erwartet steigt die Anzahl der Prozesse durch die Einbeziehung eines parallelen Programms in den Benchmark.

In weiteren Benchmarks wurden die Testprogramme mit erhöhter Priorität ausgeführt. Tab. 11 zeigt die Ergebnisse dieser Messungen in Relation zum Referenz-Benchmark. Um den Einfluß der erhöhten Priorität auch im sequentiellen Fall zu untersuchen, wurde außerdem ein Benchmark mit $NPROC = 1$ bei erhöhter Priorität des Testprogramms durchgeführt.

		Benchmark				Job			
NPROC	PR	Bench-time	CPU-time	System-time	Idle-time	CPU-time	System-time	Real-time	Real-time-Faktor
1	E	269.70	2082.36	12.85	62.42	95.66	0.50	186.78	1.00
1	H	268.77	2073.04	12.20	64.95	94.06	0.55	112.72	1.66
4	H	268.50	2069.83	15.60	62.53	92.36	4.32	36.59	5.10
6	H	271.03	2080.93	19.54	67.75	95.22	7.18	31.12	6.00
8	H	273.58	2085.28	24.92	78.40	102.19	12.77	29.45	6.34

Tab. 11. Benchmarks mit einem parallelen Programm hoher Parallelität und erhöhter Priorität und der Referenz-Benchmark

In der Spalte *PR* wird die Priorität des Testprogramms in Relation zu den übrigen Programmen angegeben, *E* (*Equal*) bedeutet gleiche Priorität, *H* (*High*) bedeutet erhöhte Priorität. Die *Bench-time* hat sich durch die Erhöhung der Priorität nicht wesentlich verändert. Auch *CPU-time*, *System-time* und *Idle-time* haben sich nur leicht verändert und führen zu keiner entscheidenden Änderung des Durchsatzes. Der *Real-time*-Faktor ist dagegen nochmals gestiegen und erreicht bei $NPROC = 8$ den Maximalwert von 6.34. Dabei ist jedoch zu beachten, daß die Erhöhung der Priorität auch für $NPROC = 1$ einen *Real-time*-Faktor von 1.66 bewirkt.

Die Ausführung eines parallelen Programms mit hoher Parallelität bei einer sequentiellen Hintergrundlast kann also im Vergleich zur sequentiellen Ausführung dieses Programms mit derselben Hintergrundlast zu einer wesentlichen Verkürzung der Ausführungszeit des parallelen Programms führen, bei einer vernachlässigbaren Verringerung des Systemdurchsatzes. Für die Programme der Hintergrundlast bedeutet das eine Vergrößerung der Ausführungszeit, d.h. die beim parallelen Programm eingesparte Ausführungszeit verteilt sich auf die Ausführungszeiten der Programme der Hintergrundlast. Solange die Anzahl

³⁶ Dabei wurden die Werte von *Idle-time* und *System-time* aus Tab. 9 zugrundegelegt, da die entsprechenden Angaben in den SAR-Tabellen aufgrund der Meßmethode systematische Fehler aufweisen (siehe Kapitel 5.2).

der parallelen Programme in Relation zur Anzahl der Programme der Hintergrundlast klein ist, wird die Ausführungszeit eines einzelnen Programms jedoch nur geringfügig vergrößert. Bei den durchgeführten Benchmarks wird die Ausführungszeit der Programme der Hintergrundlast bei einem *Real-time*-Faktor von 6.34 im Mittel um ca. 4% erhöht.

Die in diesem Kapitel durchgeführten Benchmarks haben die Möglichkeit der Leistungssteigerung durch die Ausführung eines parallelen Programms in einer Multiprogramming-Umgebung aufgezeigt. Da die auftretende Verringerung des Systemdurchsatzes vernachlässigbar ist, sind auch die Ausführungszeiten der Programme der Hintergrundlast nur geringfügig erhöht. Der höchste *Real-time*-Faktor wurde bei Ausführung des parallelen Programms mit 8 Prozessoren und erhöhter Priorität erzielt. Die Untersuchungen zeigen, daß es problemlos möglich ist, ein hochgradig paralleles Programm in einer Multiprogramming-Umgebung auszuführen, ohne dabei Overhead zu erzeugen.

6.2.3. Die Behebung von Speicherengpässen

In den bisherigen Messungen hatten alle Benchmark-Programme einen Speicherbedarf von 2 MW (Mega Worte). Im Gegensatz dazu werden die Benchmark-Programme in den folgenden Meßreihen mit erhöhtem Speicherbedarf ausgeführt. Abb. 29 enthält die Eingabedatei eines neuen Referenz-Benchmarks, bei dem wiederum alle Benchmark-Programme sequentiell ausgeführt werden.

MODE= 'EXE'														
#	DEL	PR	C	SEC	VE	MM	FL	IO	SIZE	PRNAME	PG	LB	NPROC	
2	20	10	1	120	0	70	100	0	3000	'bench'	0.	0.	1	
2	20	10	1	120	0	150	80	0	12000	'bench'	0.	0.	1	
2	20	10	1	120	0	70	100	0	9000	'bench'	0.	0.	1	
2	20	10	1	120	0	150	80	0	2000	'bench'	0.	0.	1	
2	50	10	1	90	0	100	150	0	6000	'bench'	0.	0.	1	
2	50	10	1	90	0	100	150	0	12000	'bench'	0.	0.	1	
2	50	10	1	90	0	120	70	0	3000	'bench'	0.	0.	1	
2	50	10	1	90	0	120	70	0	6000	'bench'	0.	0.	1	
2	80	10	1	60	0	100	100	0	9000	'bench'	0.	0.	1	
2	80	10	1	60	0	70	80	0	2000	'bench'	0.	0.	1	

Abb. 29. Die Eingabedatei für das Programm BENCHGEN:

Ein Benchmark mit Speicherengpaß (der Speicherbedarf der Programme ist in der Spalte 'SIZE' in Kilo Worten angegeben).

Bei einem nutzbaren Hauptspeicherbereich von 27 MW tritt bei diesem Referenz-Benchmark aufgrund des erhöhten Speicherbedarfs ein Speicherengpaß auf, d.h. die Anzahl der Programme, die gleichzeitig in den Hauptspeicher passen, ist kleiner als die Anzahl der verfügbaren Prozessoren. Zu Beginn des Referenz-Benchmarks werden z.B. 8 Benchmark-Programme gestartet, von denen sich jedoch höchstens 5 gleichzeitig im Hauptspeicher befinden können:

$$2 \times 2 \text{ MW} + 2 \times 3 \text{ MW} + 1 \times 9 \text{ MW} = 19 \text{ MW}$$

Ein weiteres Programm mit 9 bzw. 12 MW Speicherbedarf kann nicht zusätzlich in den Hauptspeicher geladen werden. Um diesen Speicherengpaß zu beheben, wird in den Test-Benchmarks jedes zweite Benchmark-Programm parallel ausgeführt, mit Parallelitätsgrad und Load-Balance-Faktor von jeweils 100%. In drei Benchmark-Läufen werden von den parallelen Programmen 4, 6 bzw. 8 Prozessoren angefordert. Abb. 30 zeigt den Referenz-Benchmark mit Speicherengpaß und den Test-Benchmark mit $NPROC = 6$.

Im Test-Benchmark wurde die *Bench-time* um 94 Sekunden verkürzt, das entspricht einer Verbesserung des Systemdurchsatzes um 32%. Außerdem wurde neben einer deutlichen Verkürzung der Ausführungszeit der parallelen Programme auch die Ausführungszeit der Programme der Hintergrundlast verkürzt.

Die im wesentlichen durch die parallelen Programme hervorgerufene leichte Erhöhung von *CPU-time* und *System-time* wird von der entscheidenden Verringerung der *Idle-time* überdeckt. Die Verringerung der *Idle-time* wird durch die parallelen Programme verursacht. Während beim Referenz-Benchmark die Anzahl der aktiven Prozesse der Anzahl der im Hauptspeicher befindlichen Programme entspricht, die zumindest zeitweise kleiner ist als die Anzahl der verfügbaren Prozessoren, sind beim Test-Benchmark mehr Prozesse aktiv, d.h. die parallelen Programme werden jeweils von bis zu 6 Prozessen ausgeführt. Beim Referenz-Benchmark wird dadurch *Idle-time* erzeugt, die beim Test-Benchmark um ca. 79% verringert ist, da die größere Anzahl von aktiven Prozessen eine effizientere Nutzung der Prozessoren ermöglicht. Die aufgrund der verkürzten *Idle-time* gewonnene *cpu time* von 870 Sekunden entspricht ca. 43% der gesamten benötigten *CPU-time*. In Tab. 12 sind die relevanten Daten des Referenz-Benchmarks mit Speicherengpaß und die Daten der Test-Benchmarks zusammengestellt.

NPROC	Benchmark				Job			
	Bench-time	CPU-time	System-time	Idle-time	CPU-time	System-time	Real-time	Real-time-Faktor
1	383.98	1954.75	17.38	1099.72	975.71	9.45	2222.14	1.00
4	320.70	1968.37	42.53	554.72	1012.24	31.62	1206.21	1.84
6	289.90	2004.01	84.92	230.26	1048.49	73.95	1016.09	2.18
8	379.39	2150.87	119.53	764.73	1189.59	104.28	1443.86	1.53

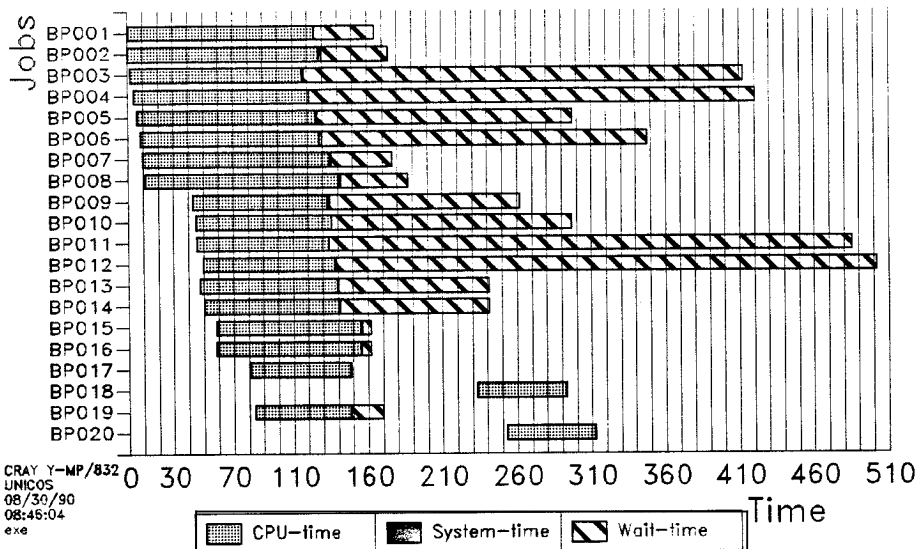
Tab. 12. Benchmarks mit hohem Speicherbedarf bei gleicher Priorität aller Programme

Die Angaben in den vier rechten Spalten beziehen sich jeweils auf alle 10 parallelen Programme. In den Test-Benchmarks mit 4 bzw. 8 Prozessoren ergibt sich im Vergleich zum Referenz-Benchmark wiederum eine geringere *Bench-time* und damit ein besserer Systemdurchsatz. In Relation zum Test-Benchmark mit 6 Prozessoren fällt diese Verbesserung jedoch deutlich geringer aus. Die wesentliche Ursache für die vergleichsweise hohe *Bench-time* bei $NPROC = 4$ bzw. 8 liegt in der erhöhten *Idle-time*. In allen 3 Test-Benchmarks haben die 3 zuletzt beendeten Programme einen Speicherplatzbedarf von jeweils 9 bzw. 12 MW, d.h. es passen immer nur 2 dieser Programme gleichzeitig in den Hauptspeicher. In allen 3 Test-Benchmarks werden also in der Endphase zumindest zeitweise nur 2 Prozessoren genutzt, die *Bench-time* wird jedoch jeweils anders berechnet.

EX01M :

PR PG LB NPROC
20 Jobs 30 0.0 0.0 1 seq

Bench-time 383.98s
Total-time 3071.85s
CPU-time 1954.75s 63.63%
System-time 17.38s 0.57%
Wait-time 2390.35s 77.81%
Idle-time 1099.72s 35.80%
Serial Jobs:
CPU-time 975.71s Service 0.44
System-time 9.45s 0.44
Real-time 2222.14s



EA12M :

PR PG LB NPROC
10 Jobs 30 0.0 0.0 1 seq
10 Jobs 30 99.7 100.0 6

Bench-time 289.90s
Total-time 2319.19s
CPU-time 2004.01s 86.41%
System-time 84.92s 3.66%
Wait-time 1221.85s 52.68%
Idle-time 230.26s 9.93%
Parallel Jobs:
CPU-time 1048.49s Service 1.03
System-time 73.95s 1.10
Real-time 1016.09s

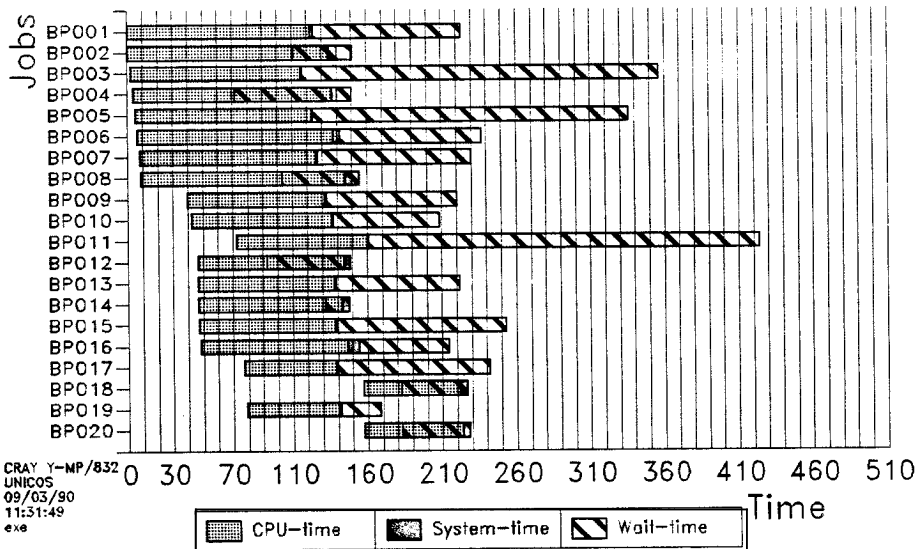


Abb. 30. Der Referenz-Benchmark mit Speicherengpaß und der Test-Benchmark mit NPROC = 6

Im Test-Benchmark mit $NPROC = 6$ werden zuletzt 3 sequentielle Programme beendet, d.h. es werden 3 Prozessoren zur *Bench-time* gerechnet, von denen immer einer *idle* ist und somit zur Erhöhung der *Idle-time* beiträgt. Wie aus den entsprechenden Benchmark-Plots hervorgeht, werden bei den Test-Benchmarks mit $NPROC = 4$ bzw. 8 zwei sequentielle Programme und ein paralleles Programm zuletzt beendet, d.h. es werden 2 + 4 bzw. 2 + 6 Prozessoren zur *Bench-time* gerechnet, von denen zumindest teilweise 4 bzw. 6 Prozessoren zur Erhöhung der *Idle-time* beitragen. Trotz gleicher Systembelastung - es werden jeweils nur 2 Prozessoren genutzt - wird in diesem Sonderfall die *Bench-time* unterschiedlich berechnet³⁷. Dabei sind die ermittelten Werte der *Bench-time* für $NPROC = 4$ bzw. 8 größer als die tatsächlich relevanten Werte; die in der Tabelle angegebenen Werte geben also nur ein ungenaues Maß für den Systemdurchsatz an.

In einer weiteren Meßreihe wurden die entsprechenden Benchmarks mit erhöhter Priorität bei den parallelen Programmen durchgeführt. Tab. 13 enthält die relevanten Daten des Referenz-Benchmarks und der Test-Benchmarks mit erhöhter Priorität der parallelen Programme.

NPROC	PR	Benchmark				Job			
		Bench-time	CPU-time	System-time	Idle-time	CPU-time	System-time	Real-time	Real-time-Faktor
1	E	383.98	1954.75	17.38	1099.72	975.71	9.45	2222.14	1.00
4	H	292.00	1981.16	75.90	278.93	1021.69	65.61	704.08	3.15
6	H	305.84	2015.97	149.87	280.85	1056.40	140.24	821.51	2.70
8	H	324.38	2115.90	172.44	306.68	1150.60	161.93	861.33	2.57

Tab. 13. Benchmarks mit hohem Speicherbedarf und parallelen Programmen mit erhöhter Priorität

Im Gegensatz zu den Test-Benchmarks mit gleicher Priorität aller Programme tragen hier keine parallelen Programme zur Berechnung der *Bench-time* bei. Auch hier ist die *Bench-time* der Test-Benchmarks in Relation zum Referenz-Benchmark verkürzt, d.h. der Systemdurchsatz ist verbessert. Für $NPROC = 6$ ergibt sich eine Verschlechterung des Systemdurchsatzes aufgrund der Prioritätserhöhung, während der *Real-time*-Faktor größer geworden ist.

Durch die Ausführung von parallelen Programmen wurde bei dem vorliegenden Speicherengpaß eine Verbesserung des Systemdurchsatzes von bis zu 32% erzielt. Dabei wurden sowohl die Ausführungszeiten der parallelen Programme als auch die Ausführungszeiten der Programme der Hintergrundlast verkürzt. Bei den betrachteten Benchmarks war die Verbesserung des Systemdurchsatzes und die damit verbundene Verkürzung der Ausführungszeiten bei der Nutzung von 6 Prozessoren am größten. Da die gewonnenen Ergebnisse jedoch von der Arbeitslast sowie von weiteren möglichen Parametern wie z.B. der Anzahl der parallelisierten Programme und den Startzeitpunkten der Programme abhängen, stellen die beschriebenen Benchmarks nur einen Teil der in

³⁷ Da keine Daten über die Zeitpunkte der Vergabe von Prozessoren an die Programme zur Verfügung stehen, ist hier keine genauere Auswertung möglich.

diesem Zusammenhang relevanten Untersuchungen dar, die mit Hilfe von *PAR-Bench* durchgeführt werden können.

6.2.4. Die Portierung von Benchmarks auf eine Cray X-MP

Um einen Vergleich der Rechner Cray X-MP/416 und Cray Y-MP8/832 zu ermöglichen, wurden die in Kapitel 6.2.2 beschriebenen Benchmarks auch auf der X-MP durchgeführt. Dabei wurden die auf der Y-MP erzeugten Programmkernfolgen verwendet. Es wurden der Referenz-Benchmark ohne Speicherengpaß und der Test-Benchmark mit einem Programm hoher Parallelität bei gleicher Priorität aller Programme durchgeführt. Abb. 31 und Abb. 32 stellen die auf der X-MP und auf der Y-MP durchgeführten Benchmarks gegenüber.

Der deutlich kleinere Systemdurchsatz bei der Cray X-MP ist im Vergleich zur Cray Y-MP zum einen auf die höhere Taktzeit (die Taktzeit der X-MP beträgt 8.5 Nanosekunden, die Taktzeit der Y-MP beträgt 6 Nanosekunden) und zum anderen auf die kleinere Anzahl der Prozessoren (die X-MP verfügt über 4 Prozessoren, die Y-MP verfügt über 8 Prozessoren) zurückzuführen. Aufgrund der unterschiedlichen Taktzeiten der beiden Rechner wird für den XY-Quotient der *CPU-time*, d.h. für den Quotient aus *CPU-time* X-MP und *CPU-time* Y-MP, ein Wert von

$$\frac{8.5}{6} = 1.42$$

erwartet. Der tatsächliche Wert liegt jedoch sowohl beim Referenz-Benchmark als auch beim Test-Benchmark bei ca. 1.18 und damit um ca. 17% niedriger. Diese Zahlenwerte gelten in etwa auch für die einzelnen Benchmark-Programme. Da die *Benchmark-Plots* keine Hinweise auf mögliche Ursachen dieses Effekts liefern, wurden zusätzliche Analysen durchgeführt. Mit Hilfe des HPM wurde beobachtet, daß die Wartezeiten aufgrund von Adreß- und Skalarrechnungen bei den Benchmarks auf der Y-MP anteilig höher waren als bei den Benchmarks auf der X-MP. Die erhöhte Wartezeit kann ein Grund für den zu niedrigen XY-Quotienten sein.

Darüber hinaus wurde bereits in [Hof90] bei Vergleichsmessungen zwischen X-MP und Y-MP festgestellt, daß die Ausführung von Triaden bei kleinen Vektorlängen auf der Y-MP mehr Maschinentakte erfordert als auf der X-MP. Da auch in den Programmkernen der Benchmark-Programme Triaden mit kleinen Vektorlängen ausgeführt werden, ist zu vermuten, daß bei den durchgeführten Messungen häufig vergleichbare Situationen aufgetreten sind. Damit sind die kleinen Vektorlängen eine mögliche Ursache für die erzielten Ergebnisse, zur vollständigen Erklärung des beobachteten Effekts sind jedoch weitere Untersuchungen notwendig.

Der Systemdurchsatz und damit auch die *Bench-time* werden sowohl von der Taktzeit als auch von der Anzahl der Prozessoren beeinflusst. Damit ergibt sich der Erwartungswert für den XY-Quotient der *Bench-time*:

$$\frac{8}{4} \times \frac{8.5}{6} = 2.83$$

Der tatsächliche Wert des XY-Quotienten der *Bench-time* liegt ebenfalls um ca. 17% niedriger bei 2.34, d.h. der relative Systemdurchsatz in bezug auf die vorhandenen

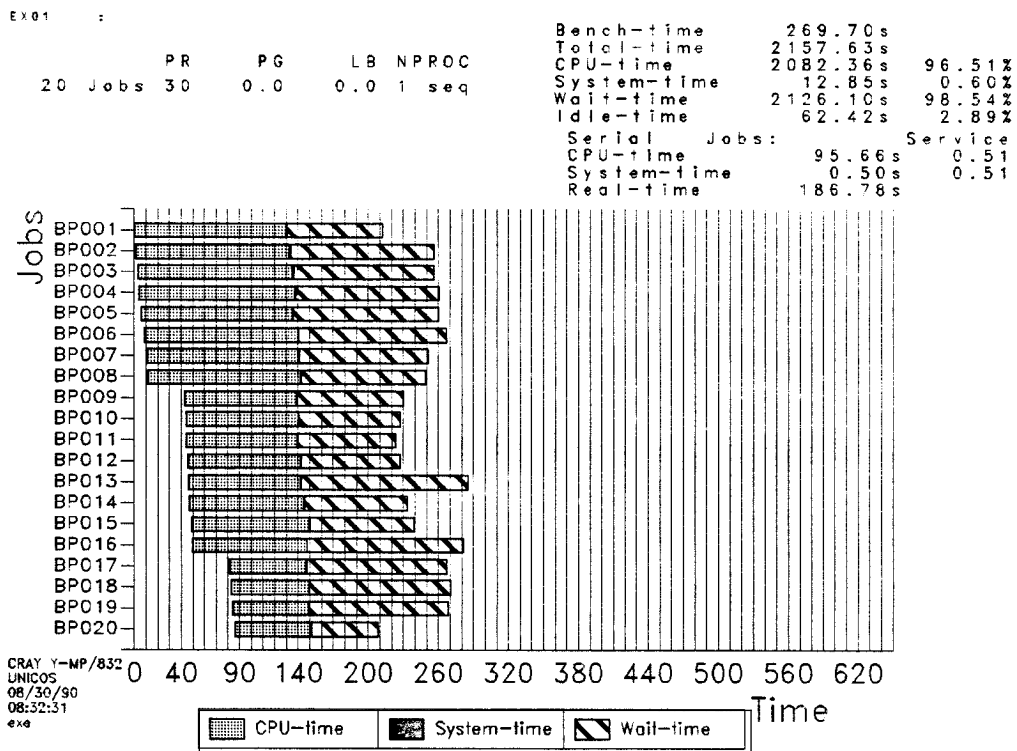
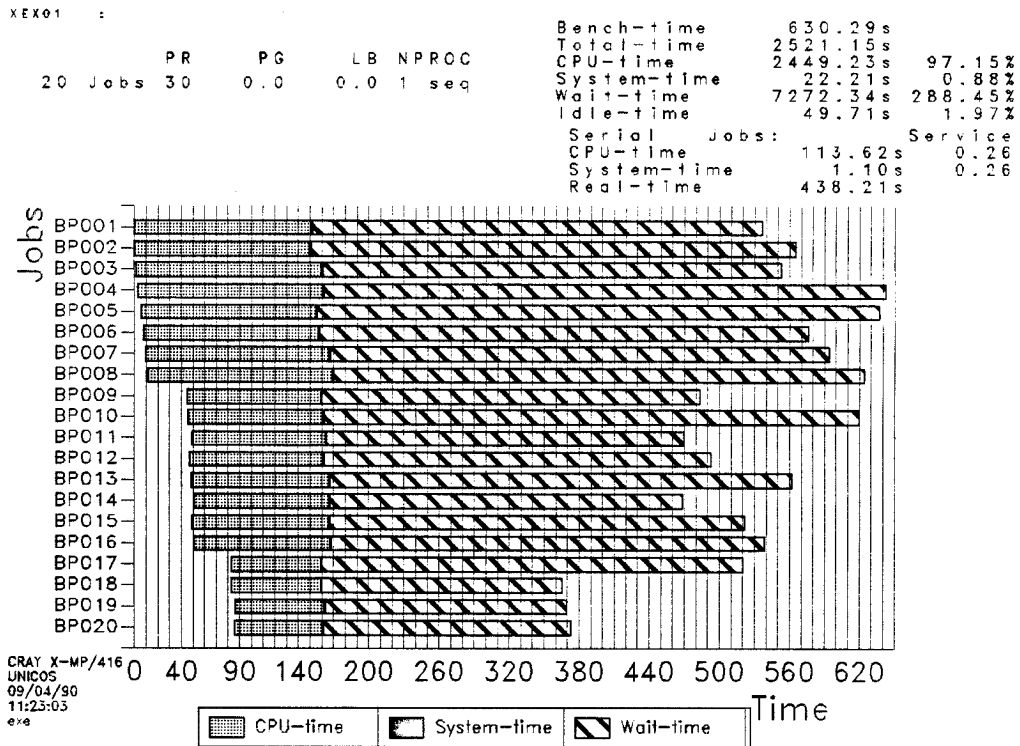
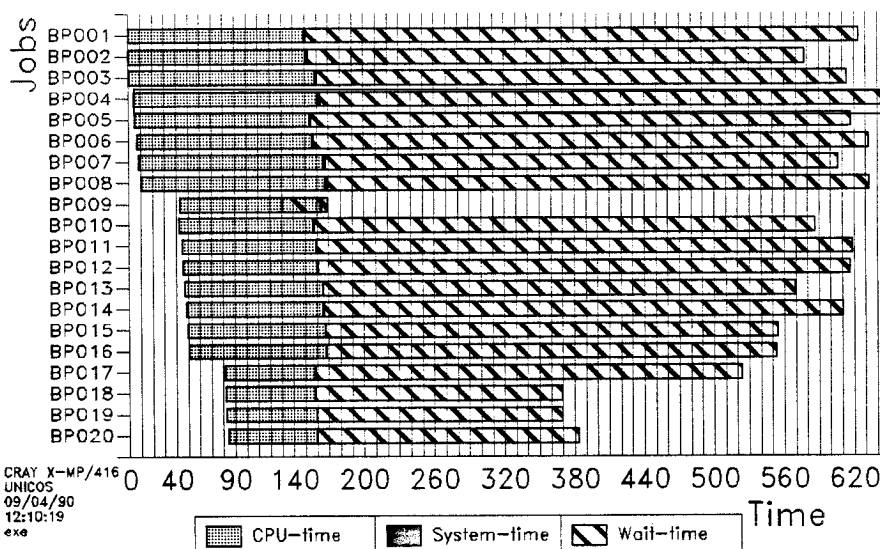


Abb. 31. Der Referenz-Benchmark auf Cray X-MP und Cray Y-MP

XED05 :

	PR	PG	LB	NPROC
19 Jobs	34	0.0	0.0	1 seq
1 Job	34	99.8	100.0	4

Bench-time	632.59s	
Total-time	2530.36s	
CPU-time	2456.87s	97.10%
System-time	31.24s	1.25%
Wait-time	7600.48s	300.37%
Idle-time	42.25s	1.67%
Parallel Jobs:		
CPU-time	120.20s	1.37
System-time	6.59s	1.45
Real-time	87.56s	



ED05 :

	PR	PG	LB	NPROC
19 Jobs	34	0.0	0.0	1 seq
1 Job	34	99.8	100.0	4

Bench-time	267.71s	
Total-time	2141.69s	
CPU-time	2068.94s	96.60%
System-time	13.06s	0.61%
Wait-time	2034.54s	95.00%
Idle-time	59.69s	2.79%
Parallel Jobs:		
CPU-time	92.03s	1.85
System-time	2.60s	1.90
Real-time	49.76s	

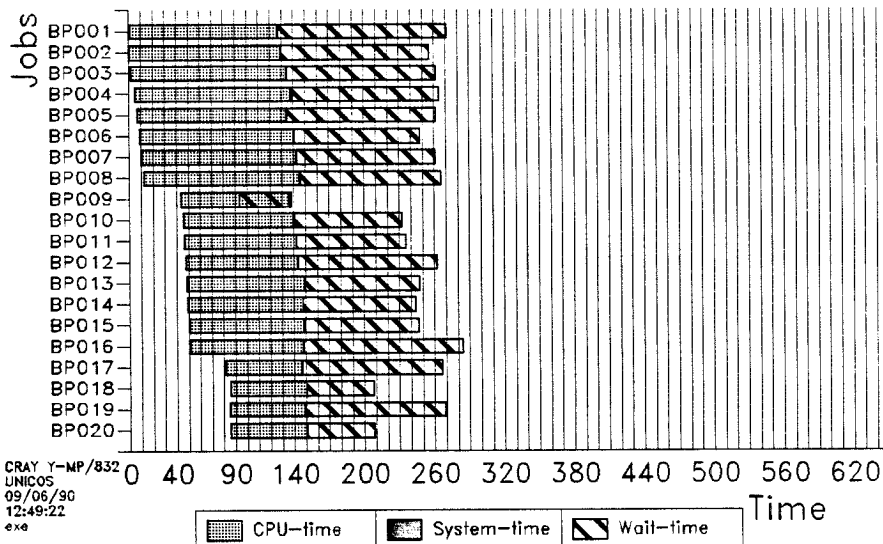


Abb. 32. Der Test-Benchmark mit einem parallelen Programm auf Cray X-MP und Cray Y-MP

Ressourcen ist bei der X-MP höher. Da der XY-Quotient der *CPU-time* als Faktor im XY-Quotient der *Bench-time* enthalten und die relative Abweichung vom Erwartungswert in beiden Fällen gleich ist, wird der beobachtete Effekt allein durch den XY-Quotient der *CPU-time* verursacht.

Aufgrund des deutlich kleineren Systemdurchsatzes bei der X-MP haben sich auch die Ausführungszeiten der Programme wesentlich erhöht. In Verbindung mit der nur leicht gestiegenen *CPU-time* (der XY-Quotient der *CPU-time* beträgt 1.18) und der ebenfalls nur leicht gestiegenen *System-time* führt das zu einer wesentlichen Erhöhung der *Wait-time*.

In einer weiteren Messung wurde der Test-Benchmark mit erhöhter Priorität des parallelen Programms ausgeführt. Tab. 14 und Tab. 15 zeigen die relevanten Daten der drei Benchmarks für X-MP und Y-MP.

		Benchmark				Job			
NPROC	PR	Bench-time	CPU-time	System-time	Idle-time	CPU-time	System-time	Real-time	Real-time-Faktor
1	E	630.29	2448.66	22.21	50.28	113.59	1.10	438.21	1.00
4	E	632.59	2456.87	31.24	42.25	120.20	6.59	87.56	5.00
4	H	634.07	2458.15	37.96	40.16	120.35	13.23	70.82	6.18

Tab. 14. Ergebnisse der Benchmarks, ausgeführt auf der Cray X-MP

		Benchmark				Job			
NPROC	PR	Bench-time	CPU-time	System-time	Idle-time	CPU-time	System-time	Real-time	Real-time-Faktor
1	E	269.70	2082.36	12.85	62.42	95.66	0.50	186.78	1.00
4	E	267.71	2068.94	13.06	59.69	92.03	2.60	49.76	3.75
4	H	268.50	2069.83	15.60	62.53	92.36	4.32	36.59	5.10

Tab. 15. Ergebnisse der Benchmarks, ausgeführt auf der Cray Y-MP

Auch beim Test-Benchmark mit erhöhter Priorität ist die Durchsatzverschlechterung um ca. 17% geringer als erwartet. Die bei den auf der Y-MP durchgeführten Test-Benchmarks beobachtete Verkürzung der *CPU-time* in Relation zum Referenz-Benchmark tritt bei den X-MP Benchmarks nicht auf. Die übrigen Zeitgrößen zeigen auf beiden Rechnern ein qualitativ ähnliches Verhalten. Der *Real-time*-Faktor erreicht bei den X-MP Benchmarks die höheren Werte, d.h. die Beschleunigung des parallelen Programms ist größer.

Erwartungsgemäß ist bei der Cray X-MP der Systemdurchsatz deutlich geringer als bei der Cray Y-MP. Entsprechend sind auch die Ausführungszeiten der Programme bei den X-MP Benchmarks erhöht. Das Ausmaß der Durchsatzverschlechterung ist jedoch um 17% geringer als erwartet. Bei den Benchmarks wurden erhöhte Wartezeiten aufgrund von Adreß- und Skalarrechnungen festgestellt. Eine mögliche Ursache sind die kleinen Vektorlängen in den Programmkerneln der Benchmark-Programme, die verhindern, daß die Vorteile der Cray Y-MP gegenüber der Cray X-MP in vollem Umfang genutzt wer-

den. Zur vollständigen Erklärung dieses Effekts müssen jedoch weitere Analysen durchgeführt werden.

7. Ausblick

Die in Kapitel 6 beschriebenen Benchmarks stellen erste Anwendungen des Benchmark-Systems *PAR-Bench* dar, in bezug auf die Möglichkeiten dieses Systems haben sie jedoch nur Beispielcharakter. Die Vielfalt der sich ergebenden Möglichkeiten wird sich zum Teil erst in der Zukunft beim Auftreten neuer Problemsituationen erweisen. Im folgenden ist eine Auswahl der zukünftigen Anwendungsmöglichkeiten des Systems *PAR-Bench* zusammengestellt:

- Im Zusammenhang mit der Untersuchung der wechselseitigen Einflüsse von parallelen Programmen und Hintergrundlast sind weitere Untersuchungen notwendig. In Erweiterung zu den beschriebenen Untersuchungen können Benchmarks mit anderen Parametervorgaben und anderen Arbeitslasten zu weiteren Erkenntnissen führen. Zum Beispiel können die Prioritäten der einzelnen Benchmark-Programme einen entscheidenden Einfluß auf den Systemdurchsatz haben, insbesondere in Abhängigkeit von der jeweiligen Arbeitslast. Die Untersuchung der in diesem Zusammenhang bestehenden Gesetzmäßigkeiten kann mit Hilfe des Systems *PAR-Bench* vorgenommen werden. Es besteht auch die Möglichkeit, die Untersuchungen mit anwenderspezifischen Arbeitslasten durchzuführen, die mit geringem Aufwand nachgebildet werden können. Dabei ist als weitere Möglichkeit des Systems *PAR-Bench* auch die Auswahl von Programmkernen relevant, die die Ein-/Ausgabemöglichkeiten intensiv austesten.
- Durch die Einbeziehung beliebiger Programmkerne in die Benchmark-Programme können verschiedene Systemkomponenten untersucht werden. Auf diese Weise können z.B. unterschiedliche Multitasking-Implementationen wie etwa Autotasking und Microtasking miteinander verglichen und der Einfluß auf die übrigen Programme untersucht werden. Mit Hilfe spezieller Ein-/Ausgabekerne können gezielt ausgewählte Ein-/Ausgabeeinheiten wie z.B. der Erweiterungsspeicher der Cray (SSD) und damit unterschiedliche Systemkonfigurationen getestet werden.
- Die Einbeziehung von beliebigen Programmen in eine Arbeitslast ermöglicht z.B. die Untersuchung des Verhaltens von Anwendungsprogrammen in Verbindung mit einer bestimmten Hintergrundlast. Auf diese Weise können auch zusätzliche interaktive Benutzerlasten, d.h. Programme mit einem entsprechenden interaktiven Belastungsprofil, in Benchmark-Läufe einbezogen werden.

Aufgrund der Modularität von *PAR-Bench* sind Erweiterungen des Systems leicht möglich, da nur Modifikationen ausgewählter Module erforderlich sind. Analog zu den Multitasking-Parametern kann z.B. ein Lokitätsfaktor eingeführt werden, mit dessen Hilfe der Einfluß der Lokalität eines Programms untersucht werden kann. Dies ist insbesondere bei Untersuchungen auf Rechnern mit hierarchischem Speicherkonzept wie z.B. einem Cache hilfreich, da dann die Leistungsfähigkeit der Speicherhierarchie implizit getestet werden kann.

Die Portierung des Systems *PAR-Bench* auf andere Rechner ist mit geringem Aufwand möglich. In Verbindung mit der Flexibilität des Systems kann ein Vergleich verschiedener Rechner unter weitgehend beliebigen Belastungen durchgeführt werden. In diesem Zusammenhang ist in Zukunft auch die Portierung des Systems *PAR-Bench* auf eine IBM 3090 geplant.

- [AlGo89] G.S. Almasi and A. Gottlieb, Highly parallel computing, *The Benjamin/Cummings Publishing Company, Redwood City, California* (1989).
- [BDW85] J. Beetem, M. Denneau, and D. Weingarten, The GF11 supercomputer, *SIGARCH Newsletter* 13 (3) (1985) 108-115.
- [BEGK88] T. Bönniger, R. Esser, J. Gehm und D. Krekel, Vergleichende Darstellung von Höchstleistungsrechnern auf der Basis eines Kenngrößenkataloges, *Angewandte Informatik* 30 (2) (1988) 51-70.
- [BEK90] T. Bönniger, R. Esser und D. Krekel, Cray Y-MP, NEC SX-3 und Siemens VP S, Vergleichende Darstellung von Höchstleistungsrechnern, *Wirtschaftsinformatik* 32 (3) (1990) 273-286.
- [Bie87] M. Bieterman, The impact of microtasking in a multiprogramming environment, *Proceedings of Cray User Group Meeting (Fall)* (1987) 144-148.
- [Bie88] M. Bieterman, Microtasking general purpose partial differential equation software on Cray X-MP, *The Journal of Supercomputing* 2 (1988) 381-414.
- [Bri88] E.O. Brigham, The Fast Fourier Transform and it's applications, *Prentice Hall, Englewood Cliffs* (1988).
- [Bur89] H. Burg, Implementierung und Untersuchung paralleler Algorithmen zur Schnellen Fourier-Transformation auf Cray-Mehrprozessorrechnern, *Kernforschungsanlage Jülich, Jül-Spez-520* (1989).
- [CDL88] D. Calahan, J.J. Dongarra, and D. Levine, Vectorizing compilers: A test suite and results, *Argonne National Laboratory, Mathematics and Computer Science Division, Technical Memorandum No. 109* (1988).
- [Com89] Supercomputer hardware: An update of the 1983 report's summary and tables, *IEEE Computer* 22 (11) (1989) 63-68.
- [Cra0222] Cray multitasking programmer's manual, *Cray Research, Inc., SR-0222 F* (1989).
- [Cra2011] Cray user commands reference manual, *Cray Research, Inc., SR-2011* (1989).
- [Cra2012] Cray system calls reference manual, *Cray Research, Inc., SR-2012* (1989).
- [Cra2040] Cray performance utilities reference manual, *Cray Research, Inc., SR-2040 B* (1989).
- [Cra2079] Cray UNICOS FORTRAN library reference manual, Vol. 1, *Cray Research, Inc., SR-2079 5.0* (1989).

- [Cra2088] Cray autotasking user's guide, *Cray Research, Inc., SN-2088* (1989).
- [CuWi76] H.J. Curnow and B.A. Wichman, A synthetic benchmark, *Computing Journal* **19** (1) (1976)
- [DeNe88] D. Dent and M. O'Neill, Microtasking as a complement to macrotasking, *Parallel Computing* **8** (1988) 149-154.
- [Det87] U. Detert, Programmiertechniken für die Vektorisierung, *Praxis der Informationsverarbeitung und Kommunikation* **10** (3) (1987) 189-195.
- [Det89] U. Detert, Memory performance of Cray X-MP and Cray Y-MP, *Proceedings of Cray User Group Meeting (Fall)* (1989) 50-54.
- [DiIy89] R.T. Dimpsey and R.K. Iyer, Multiprogramming performance degradation: Case study on a shared memory multiprocessor, in *Proceedings of 1989 International Conference on Parallel Processing Vol. II, St. Charles, IL* (1989) 205-208.
- [DiIy90] R.T. Dimpsey and R.K. Iyer, Performance degradation due to multiprogramming and system overheads in real workloads: Case study on a shared memory multiprocessor, in *Proceedings of 1990 International Conference on Supercomputing* (1990) 227-238.
- [Don89] J.J. Dongarra, Performance of various computers using standard linear equations software in a FORTRAN environment, *Argonne National Laboratory, Mathematics and Computer Science Division: Technical Memorandum No. 23* (1989).
- [Esc89] Y. Escaig, Developpement d'outils d'analyse de programmes parallélisés utilisant les techniques de microtasking ou autotasking pour Cray Y-MP et X-MP, *Université de technologie de Compiègne, memoire de D.E.A.* (1989).
- [Gel89] E. Gelenbe, Multiprocessor performance, *John Wiley & Sons, Chichester, England* (1989).
- [Gil88] W. Giloi, SUPRENUM: A trendsetter in modern supercomputer development, *Parallel Computing* **7** (1988) 283-296.
- [GuNo89] N.J. Gunther and M.T. Noga, A benchmark for shared-memory architectures, *Computer Architecture News* **17** (4) (1989) 54-61.
- [HaLu88] U. Harms and H. Luttermann, Experiences in benchmarking the three supercomputers Cray-1M, Cray X-MP, Fujitsu VP-200 compared with the Cyber 76, *Parallel Computing* **6** (1988) 373-382.
- [HKN89] F. Hoßfeld, R. Knecht, and W.E. Nagel, Multitasking: Experiences with applications on a Cray X-MP, *Parallel Computing* **12** (1989) 259-283.

- [HeKa89] J. Helin and K. Kaski, Performance analysis of high-speed computers II, *Tampere University of Technology: Electronics Laboratory Report 3-89* (1989).
- [Hof90] G. Hofemann, Vergleich der Speicherarchitekturen der Mehrprozessor-Vektorrechner Cray X-MP und Cray Y-MP anhand vektorisierender Algorithmen, *Forschungszentrum Jülich, Jül-Spez-555* (1990).
- [HoJe88] R.W. Hockney and C.R. Jesshope, Parallel computers 2, *Adam Hilger, Bristol* (1988).
- [HwBr86] K. Hwang and F.A. Briggs, Computer architecture and parallel processing, *McGraw-Hill Book Company, New York* (1986).
- [IBM20] IBM document composition facility: GML starter set user's guide, *IBM, SH20-9186*.
- [IBM24] IBM VM/SP system product interpreter reference, *IBM, SC24-5239-1*.
- [IBM33] IBM GDDM PGF interactive chart utility, *IBM, SC33-0328-0*.
- [KLS86] J.M. van Kats, R. Llurba, and A.J. van der Steen, A close look at the first generation of Japanese supercomputers, *Academic Computing Centre Utrecht: Technical Report TR-22* (1986).
- [KaSt87] J.M. van Kats and A.J. van der Steen, Benchmarktests on an Alliant FX/8, a Convex C-1, an FPS 64/60 and an SCS-40, *Academic Computing Centre Utrecht: Technical Report TR-24* (1987).
- [Laz86] C. Lazou, Supercomputers and their use, *Clarendon Press, Oxford* (1986).
- [Lie88] A. Liegmann, Die Strategie des Microtasking als Mittel zur Beschleunigung von Programmen auf dem Vektorrechner Cray X-MP, *Kernforschungsanlage Jülich, Jül-Spez-435* (1988).
- [Mar87] J.L. Martin, Performance evaluation: Applications and architectures, *Proceedings of the Second International Conference on Supercomputers 3* (1987) 369-373.
- [Mar88] J.L. Martin, Performance evaluation of supercomputers, *North-Holland, Amsterdam* (1988).
- [McM86] F.H. McMahon, The Livermore FORTRAN kernels: A computer test of the numerical performance range, *Lawrence Livermore National Laboratory, UCRL-53745* (1986).
- [Nag90a] W.E. Nagel, Exploiting autotasking on a Cray Y-MP: An improved software interface to multitasking, *Parallel Computing 13* (1990) 225-233.
- [Nag90b] W.E. Nagel, Parallelism on Cray multiprocessor systems: Concepts of multitasking, erscheint in *Proceedings of Ninth International Conference on Computing Methods in Applied Sciences and Engineering* (1990).

- [Nag90c] W.E. Nagel, Performance evaluation of multitasking in a multiprogramming environment, *Forschungszentrum Jülich, KFA-ZAM-IB-9004* (1990).
- [PER89] PERFECT Club, The PERFECT Club benchmarks: Effective performance evaluation of supercomputers, *The International Journal of Supercomputer Applications* 3 (3) (1989) 5-40.
- [Reg89] H. Reger, Ein Vergleich der Multitasking-Implementierungen auf Cray X-MP und IBM 3090, *Kernforschungsanlage Jülich, Jül-Spez-542* (1989).
- [Ste88] A.J. van der Steen, Is it really possible to benchmark a supercomputer? A graded approach to performance measurement, in *Evaluating Supercomputers: Strategies for Exploiting, Evaluating and Benchmarking Computers with Advanced Architecture, UNICOM Seminars Ltd., London* (1988) 114-125.
- [StPa88] A.J. van der Steen and R.J. van der Pas, The broadening spectrum of minisupercomputers: Benchmarktests on a Gould NP1 and a Prime MXCL 5, *Academic Computing Centre Utrecht: Technical Report TR-27* (1988).
- [StPa89] A.J. van der Steen and R.J. van der Pas, A family portrait: Benchmarktests on a Cray Y-MP and a Cray-2S, *Joint publication of the WGS and ACCU (Academic Computing Centre Utrecht): Technical Report TR-30* (1989).

Anhang A. Die Daten aller Programmkerne

#	MMREFS	MFLOPS	MIOS	KTIME
1	0.3042749E+03	0.2026991E+03	0.0000000E+00	0.7426182E+00
2	0.3016307E+03	0.1987915E+03	0.1590330E+01	0.7572162E+00
3	0.3060782E+03	0.2038969E+03	0.1213671E-02	0.7382560E+00
4	0.3049481E+03	0.2024189E+03	0.5397835E+00	0.7436463E+00
5	0.2880145E+03	0.1913155E+03	0.3416171E-02	0.7868458E+00
6	0.2233510E+03	0.1481289E+03	0.0000000E+00	0.1451709E+00
7	0.2173632E+03	0.1340732E+03	0.7508021E+01	0.1603917E+00
8	0.2262922E+03	0.1500619E+03	0.6252532E-02	0.1433020E+00
9	0.2221540E+03	0.1437296E+03	0.2682945E+01	0.1496147E+00
10	0.1727531E+03	0.1123171E+03	0.1403454E-01	0.1915274E+00
11	0.1575439E+03	0.1039462E+03	0.0000000E+00	0.1034382E+00
12	0.1570095E+03	0.9010829E+02	0.1009192E+02	0.1193255E+00
13	0.1560846E+03	0.1029575E+03	0.8579636E-02	0.1044333E+00
14	0.1569285E+03	0.9861825E+02	0.3681716E+01	0.1090274E+00
15	0.1117605E+03	0.7089779E+02	0.1771160E-01	0.1517650E+00
16	0.3670922E+02	0.2294288E+02	0.0000000E+00	0.7322576E-01
17	0.5783942E+02	0.1895437E+02	0.1358462E+02	0.8864615E-01
18	0.3650385E+02	0.2277753E+02	0.1214650E-01	0.7376612E-01
19	0.4453565E+02	0.2136909E+02	0.5105508E+01	0.7862254E-01
20	0.2759438E+02	0.1388885E+02	0.2211943E-01	0.1215221E+00
21	0.1715490E+03	0.2663914E+03	0.0000000E+00	0.9888606E+00
22	0.1712926E+03	0.2622230E+03	0.1198733E+01	0.1004581E+01
23	0.1713882E+03	0.2661345E+03	0.9052187E-03	0.9898160E+00
24	0.1713438E+03	0.2648030E+03	0.4035093E+00	0.9947925E+00
25	0.1638303E+03	0.2534190E+03	0.2585832E-02	0.1039511E+01
26	0.1559293E+03	0.2396437E+03	0.0000000E+00	0.1570332E+00
27	0.1555870E+03	0.2174570E+03	0.6958582E+01	0.1730560E+00
28	0.1553016E+03	0.2386343E+03	0.5681740E-02	0.1576982E+00
29	0.1556026E+03	0.2314529E+03	0.2468826E+01	0.1625906E+00
30	0.1228715E+03	0.1838984E+03	0.1313287E-01	0.2046772E+00
31	0.1152439E+03	0.1750136E+03	0.0000000E+00	0.1075117E+00
32	0.1208599E+03	0.1533546E+03	0.9814581E+01	0.1226974E+00
33	0.1147234E+03	0.1741589E+03	0.8293192E-02	0.1080404E+00
34	0.1173031E+03	0.1671665E+03	0.3566201E+01	0.1125590E+00
35	0.8388067E+02	0.1209646E+03	0.1727351E-01	0.1556140E+00
36	0.2846408E+02	0.3831599E+02	0.0000000E+00	0.7673063E-01
37	0.5013809E+02	0.3188611E+02	0.1305952E+02	0.9221045E-01
38	0.2827184E+02	0.3797785E+02	0.1157343E-01	0.7741873E-01
39	0.3654602E+02	0.3584580E+02	0.4893994E+01	0.8202054E-01
40	0.2283409E+02	0.2364253E+02	0.2155882E-01	0.1246822E+00
41	0.2531509E+03	0.1251224E-03	0.0000000E+00	0.1040581E+01
42	0.2522715E+03	0.3341944E-03	0.1142660E+01	0.1053878E+01
43	0.2532913E+03	0.3174930E-03	0.8615194E-03	0.1040023E+01
44	0.2533512E+03	0.2034575E-03	0.3848703E+00	0.1042970E+01
45	0.2433102E+03	0.7092587E-02	0.2476537E-02	0.1085387E+01
46	0.2307974E+03	0.7985156E-03	0.0000000E+00	0.1630525E+00
47	0.2246981E+03	0.1974947E-02	0.6752635E+01	0.1783339E+00
48	0.2301081E+03	0.2018813E-02	0.5478062E-02	0.1635615E+00
49	0.2284058E+03	0.1260684E-02	0.2384772E+01	0.1683213E+00
50	0.1816588E+03	0.3651905E-01	0.1275145E-01	0.2107996E+00
51	0.1656667E+03	0.1146348E-02	0.0000000E+00	0.1135781E+00
52	0.1653567E+03	0.2739958E-02	0.9368321E+01	0.1285421E+00
53	0.1661650E+03	0.2915266E-02	0.7910594E-02	0.1132658E+00
54	0.1658849E+03	0.1793253E-02	0.3392206E+01	0.1183324E+00
55	0.1213477E+03	0.4796066E-01	0.1674654E-01	0.1605107E+00
56	0.3744226E+02	0.1658096E-02	0.0000000E+00	0.7852381E-01
57	0.5732959E+02	0.3753588E-02	0.1283408E+02	0.9383021E-01
58	0.3708697E+02	0.4158535E-02	0.1128421E-01	0.7940296E-01
59	0.4493155E+02	0.2540110E-02	0.4804998E+01	0.8353968E-01
60	0.2846599E+02	0.6084417E-01	0.2124511E-01	0.1265232E+00
61	0.7909440E+01	0.1542338E+03	0.0000000E+00	0.4757866E+01

62	0.8392964E+01	0.1537002E+03	0.2522260E+00	0.4774385E+01
63	0.7908241E+01	0.1541912E+03	0.1882677E-03	0.4759181E+01
64	0.8066426E+01	0.1539668E+03	0.8422124E-01	0.4766114E+01
65	0.7967641E+01	0.1526864E+03	0.5592854E-03	0.4806133E+01
66	0.7894838E+01	0.1539475E+03	0.0000000E+00	0.6809589E+00
67	0.1122188E+02	0.1505283E+03	0.1729142E+01	0.6964287E+00
68	0.7897061E+01	0.1538570E+03	0.1315014E-02	0.6813615E+00
69	0.9018311E+01	0.1527437E+03	0.5848645E+00	0.6863265E+00
70	0.8284732E+01	0.1438609E+03	0.3688479E-02	0.7287557E+00
71	0.7339791E+01	0.1431227E+03	0.0000000E+00	0.3662312E+00
72	0.1343305E+02	0.1373271E+03	0.3154990E+01	0.3816887E+00
73	0.7337370E+01	0.1428266E+03	0.2441472E-02	0.3669917E+00
74	0.9429940E+01	0.1411621E+03	0.1081036E+01	0.3713179E+00
75	0.8088279E+01	0.1265893E+03	0.6490819E-02	0.4141234E+00
76	0.3593577E+01	0.7006483E+02	0.0000000E+00	0.1168919E+00
77	0.2160994E+02	0.6190120E+02	0.9101440E+01	0.1323114E+00
78	0.3617668E+01	0.6975672E+02	0.7631321E-02	0.1174109E+00
79	0.1010889E+02	0.6711974E+02	0.3289638E+01	0.1220219E+00
80	0.6578067E+01	0.4985848E+02	0.1634868E-01	0.1644169E+00
81	0.3174847E+02	0.2111850E+02	0.0000000E+00	0.7127779E+01
82	0.3204097E+02	0.2108582E+02	0.1686863E+00	0.7138836E+01
83	0.3174502E+02	0.2111580E+02	0.1256891E-03	0.7128701E+01
84	0.3173869E+02	0.2103639E+02	0.5609701E-01	0.7155604E+01
85	0.3155053E+02	0.2092671E+02	0.3736727E-03	0.7193462E+01
86	0.2635198E+02	0.1729769E+02	0.0000000E+00	0.1243173E+01
87	0.2795116E+02	0.1707617E+02	0.9562551E+00	0.1259313E+01
88	0.2629335E+02	0.1725689E+02	0.7190300E-03	0.1246123E+01
89	0.2683611E+02	0.1718874E+02	0.3208551E+00	0.1251057E+01
90	0.2584362E+02	0.1663409E+02	0.2078529E-02	0.1293222E+01
91	0.2207260E+02	0.1426912E+02	0.0000000E+00	0.7535161E+00
92	0.2478618E+02	0.1397427E+02	0.1565085E+01	0.7694306E+00
93	0.2203932E+02	0.1424383E+02	0.1186964E-02	0.7548671E+00
94	0.2299288E+02	0.1417107E+02	0.5290489E+00	0.7587352E+00
95	0.2162108E+02	0.1345205E+02	0.3360644E-02	0.7998467E+00
96	0.1167111E+02	0.6483858E+01	0.0000000E+00	0.2591066E+00
97	0.1986505E+02	0.6109540E+01	0.4378713E+01	0.2750178E+00
98	0.1165792E+02	0.6467257E+01	0.3448772E-02	0.2598026E+00
99	0.1451681E+02	0.6355583E+01	0.1518477E+01	0.2643490E+00
100	0.1202264E+02	0.5505057E+01	0.8768538E-02	0.3065505E+00
101	0.1582069E+03	0.1454142E-01	0.7810492E+02	0.1541803E-01
102	0.8082288E+01	0.2485730E+00	0.8951104E-01	0.1000994E-01
103	0.1570411E+03	0.1625423E-01	0.7748907E+02	0.5180189E-02
104	0.1394199E+02	0.1595463E+00	0.5665113E-01	0.4744830E-01
MIN	0.3593577E+01	0.1251224E-03	0.0000000E+00	
MAX	0.3060782E+03	0.2663914E+03	0.7810492E+02	

Anhang B. Die Ausgabedatei eines Benchmark-Programms

```

*****
* JOB      = BP009                                     *
* MACHINE  = CRAY Y-MP8/832                           *
* SYSTEM   = UNICOS                                     *
*****

                                EINGABE DATEN
                                =====
STEUERGROESSEN:  VERS IDENTIFIER.....READFIL          BEN448
-----          BETRIEBSART.....MODE                  EXE
KONSTANTEN      :  DIMENSION N.....N                  448
-----          NORMIERUNG MINUS.....EPMX             0.1000000E-05
                  NORMIERUNG PLUS.....EPX              0.1000000E+07
                  EPSILON.....EPS                      0.1000000E-04
AUSWAHL         :  MAX ANZAHL.....MAXVERS             120
-----          ANZ KENNZ VERS.....VERSDIM             5
                  ANZ KENNZ IVERS.....IVERSDIM          4
                  BESTE VERSION .....IBEST              36
BEWERTUNG       :  MEMORY-BEWERTUNG.....MEMFACT        1.00000000
-----          MFLOP -BEWERTUNG.....FLOPFACT          1.00000000
                  IO-OPS-BEWERTUNG.....IOFACT          100000000.00000000
ADAPTION        :  MEMORY-ADAPTION .....MEMADAP         0.39721271
-----          MFLOP -ADAPTION .....FLOPADAP           0.37363103
                  IO-OPS-ADAPTION .....LOADAP           1.27166365

                                ZIELVORGABE
                                =====
ZIELVORGABE:    GEWUENSCHTE LAUFZEIT..TOTTIME           90.00000
-----          MAXIMALE ZEIT.....ENDTIME             389.62706
                  MEMORY-GROESSE.....TMEMSIZ           2000000
                  MEMORY-REFERENZEN/S...TMEMREFS        100.00000
                  MFLOP/S.....TMFLOPS                  150.00000
                  IO-ELEMENTE/S.....TIOOPS              0.00000
                  PARALLELITAETSGRAD.....PG             1.00000000
                  LOAD BALANCE.....LB                   1.00000000
                  ANZAHL CPUS.....NPROC                 1

                                AUSGABE DATEN
                                =====
ZEITEN:         DATUM.....DATUM                       08/30/90
-----          STARTZEIT.....CLOCKST                 08:33:14.386638
                  ENDZEIT.....CLOCKEND                 08:36:21.167174
                  VERBRAUCHTE CPUZEIT...ACTTIME          95.62917
                  VERBRAUCHTE REALZEIT....ZEIT           186.78053

ERGEBNIS:       ITERATIONEN..... ITERAT               588
-----          MEMORY-GROESSE.....TMEMSIZ           2000000
                  MEMORY-REFERENZEN/S...ACTSMEM         0.00000
                  MFLOP/S.....ACTSFLOP                 0.00000
                  IO-ELEMENTE/S.....ACTSIO              0.00000
                  ABWEICHUNG.....ABWEICH               180.27756
                  PARALLELITAETSGRAD.....PGEFF          0.00000000
                  LOAD BALANCE.....LBEFF                0.00000000
                  ANZAHL CPUS.....NPROCEFF              1

...

ANZAHL:....    5 VERSION:...    1 VLAENGE:...    32 IOVERSION: 0
ANZAHL:....    1 VERSION:...    1 VLAENGE:...    5 IOVERSION: 0
ANZAHL:....   460 VERSION:...    2 VLAENGE:...   32 IOVERSION: 0
ANZAHL:....   122 VERSION:...    2 VLAENGE:...    5 IOVERSION: 0

```

```

*****
stderr (standard error file & shell log file)
*****

08:32:24 + sleep 50
08:33:14 + cd /usr9/zdv/zdv202/Linn/nexe
08:33:14 + cat
08:33:14 + /usr9/zdv/zdv202/Linn/nbin/allfseq.out
STOP
CP: 95.655s, Wallclock: 186.794s, 6.4% of 8-CPU Machine

```

```

*****
Hardware performance monitor of all user commands
*****

```

Group 1: CPU seconds : 95.63187 CP executing: 15938645193

Hold issue condition	% of all CPs	actual # of CPs
Waiting on semaphores	: 0.00	213
Waiting on shared registers	: 0.00	0
Waiting on A-registers/funct. units:	9.95	1585862590
Waiting on S-registers/funct. units:	4.98	794392265
Waiting on V-registers	: 0.01	1344045
Waiting on vector functional units :	42.54	6779658214
Waiting on scalar memory references:	0.77	122497388
Waiting on block memory references :	7.32	1166671381

```

*****
Job Accounting Report
*****

```

Job Accounting - Command Report

=====

Command Name	Started At	Elapsed Seconds	User CPU Seconds	...	Ni FI	SBU's
ja	08:32:24	0.0535	0.0039	...	30	0.00
sleep	08:32:24	49.8016	0.0017	...	30	0.00
cat	08:33:14	0.1139	0.0019	...	30	0.00
allfseq.	08:33:14	187.0450	95.6553	...	30	6.18
sh	08:32:24	237.0495	0.0089	...	30	0.00

...

Job Accounting - Summary Report

=====

```

Job Accounting File Name : /usr9/zdv/zdv202/.JOBACCT.19271
Operating System        : sn1011 sn1011 5.1.10 hlrz.18 CRAY Y-MP
User Name (ID)          : zdv202 (19202)
Group Name (ID)         : zdv (19)
Account Name (ID)       : zdv (19)
Job Name (ID)           : bp009 (36)
Report Starts           : 08/30/90 08:32:23
Report Ends             : 08/30/90 08:36:21
Elapsed Time            :          238 Seconds
User CPU Time           :          95.7446 Seconds
System CPU Time         :           0.4998 Seconds
I/O Wait Time (Locked)  :           0.3064 Seconds

```

...

```

*****
End of Job: Thu Aug 30 08:36:21 MEZ 1990 (19271.sn1011)
*****

```

Diese Arbeit wurde als Diplomarbeit am Lehrstuhl für Technische Informatik und Computerwissenschaften der Rheinisch-Westfälischen Technischen Hochschule Aachen erstellt.

Herrn Prof. Dr. F. Hoßfeld danke ich für die Möglichkeit, die Arbeit am Zentralinstitut für Angewandte Mathematik des Forschungszentrums (KFA) Jülich anzufertigen; technische und personelle Unterstützung waren eine wesentliche Grundlage für die Durchführung der Arbeit. Mein besonderer Dank gilt Herrn W.E. Nagel für die intensive Betreuung. Für hilfreiche Unterstützung und angeregte Diskussionen danke ich allen Institutsangehörigen, insbesondere den Herren Dr. N. Attig, Dr. R. Esser, V. Sander, L. Wollschläger sowie Herrn Dr. P. Weidner.

