

Prozessorarchitekturen – Ausgewählte Prozessoren

Sommersemester 2001

Heterogeneous Element Processor (HEP)

Martin Grabmüller *

25. Juni 2001



*mgrabmue@cs.tu-berlin.de

Inhaltsverzeichnis

1	Einführung	3
2	Der Denelcor Heterogeneous Element Processor	3
2.1	Entstehungsgeschichte	3
2.2	Architekturüberblick	4
2.3	Aufbau	4
3	Process Execution Modules	4
3.1	Fine-Grained Multithreading	5
3.2	Tasks und Threads	6
4	Das Kommunikationsnetzwerk	8
4.1	Switch Nodes	8
4.2	Routing	8
5	Speicherorganisation	9
6	Ein- und Ausgabe	10
6.1	Ein-/Ausgabe-Cache	10
6.2	Massenspeicher	10
7	Nachteile des HEP-Designs	10
8	Zusammenfassung	11

1 Einführung

Der Heterogeneous Element Processor (kurz HEP) der Firma Denelcor war der erste kommerzielle MIMD-Rechner der Welt. Weiterhin war er auch der erste Prozessor, der Multithreading auf Instruktionsebene unterstützte, und er war der erste pipelined Skalarprozessor.

Das HEP-Design brachte einige Neuerungen und erzielte durchaus Aufmerksamkeit in der Welt der nebenläufigen Programmierung, konnte am Markt aber nur kurz bestehen.

2 Der Denelcor Heterogeneous Element Processor

Die Entwicklung des HEP-Rechners wurde bei der Firma Denelcor durchgeführt und vom U.S. Army Ballistic Research Laboratory mitfinanziert.

2.1 Entstehungsgeschichte

Im Jahre 1974 begann Dr. Burton Smith mit dem Entwurf des HEP. 1979 war dann schließlich der erste Einprozessor-Prototyp lauffähig. Das erste voll funktionsfähige Mehrprozessormodell wurde 1982 ausgeliefert und in den USA installiert. Schon bald nach dem Verkauf der ersten Maschinen kamen jedoch ähnlich leistungsfähige Computer anderer Hersteller auf den Markt, so dass die Herstellerfirma 1985 schließlich aufgeben und schließen musste.

Insgesamt wurden nur fünf Exemplare des HEP verkauft: Eins ging an das Ballistic Research Laboratory, eins an das Los Alamos National Laboratory, eins an das Argonne National Laboratory, eins an die Colorado State University und das letzte an die Firma Messerschmitt in Deutschland. Bei Messerschmitt wurde der HEP tatsächlich für Produktionssoftware eingesetzt, bei den anderen Installationsorten wurden die Maschinen vor allem genutzt, um auf dem Gebiet der Mehrprozessor- und Multithreading-Programmierung zu forschen.

Da das Los Alamos National Laboratory seinen HEP zur Forschung für viele Wissenschaftler freigab, sammelten viele Wissenschaftler auf dieser Maschine ihre ersten Erfahrungen in der Programmierung nebenläufiger Systeme. Dies ist wahrscheinlich der Grund, warum der HEP in Forscherkreisen einen hohen Bekanntheitsgrad erreichte.

Insgesamt wurden acht Exemplare des HEP-1 gebaut. Das Nachfolgemodell HEP-2 wurde zwar geplant, konnte aber nicht mehr gebaut werden. Nachdem Denelcor nicht mehr existierte, gründete Burton Smith mit seinem Kollegen James Rottsolk die Firma Tera Computer Co. (1988), die ähnliche Computer entwickelte wie den HEP. Diese Modelle hießen HORIZON und TERA. Burtons Firma wurde später von Cray aufgekauft. Cray verkauft heute den MTA Supercomputer, ein von Burton mitentwickelter Rechner[1], der einige erstmals in der HEP-Architektur vorhandene Merkmale aufweist.

2.2 Architekturüberblick

Der HEP ist ein Skalarprozessor, der auf Befehlsebene nebenläufige Programmierung unterstützt. Er ist damit ein MIMD (Multiple Instruction, Multiple Data) Rechner, d.h. dass mehrere Programme (Instruktionsströme) gleichzeitig unterschiedliche Daten verarbeiten können.

Jeder Prozessor der Maschine verfügt über einen lokalen Programmspeicher. Alle Prozessoren teilen sich einen gemeinsamen Datenspeicher (shared memory), sowie ein gemeinsames Ein-/Ausgabeundersystem. Die Prozessoren, der Datenspeicher und das Ein-/Ausgabesystem sind über ein internes Netzwerk verbunden.

2.3 Aufbau

Der HEP ist aus mehreren Modulen zusammengesetzt, wobei einige dieser Module je nach Ausstattung und gewünschter Leistungsfähigkeit in unterschiedlicher Anzahl installiert werden können. Der generelle Aufbau ist in Abbildung 1 dargestellt.

Zunächst besteht ein HEP-Rechner aus einem oder mehreren Process Execution Modules (PEM), welche die eigentliche Datenverarbeitung durchführen. Diese können mehrfach vorhanden sein, um die Leistung zu erhöhen. Jedes PEM ist mit einem Program Memory Module verbunden, in dem die Programme gespeichert werden, die auf diesem PEM laufen sollen. Genauer werden die PEMs in Abschnitt 3 beschrieben.

Für die Ein- und Ausgabe steht ein I/O-Subsystem zur Verfügung, das von allen PEMs gemeinsam benutzt wird. Für die Kommunikation zwischen den einzelnen PEMs und dem gemeinsamen Speicher bzw. dem Ein-/Ausgabesystem sind die einzelnen Bestandteile der Maschine über ein Kommunikationsnetzwerk miteinander verbunden. Dieses Netzwerk wird in Abschnitt 4 näher erläutert.

3 Process Execution Modules

Die Process Execution Modules sind das Herzstück des HEP. Abbildung 2 stellt ein PEM im Blockbild dar. Ein PEM besteht aus einem Programmspeichermodul, der Instruktionsverarbeitungseinheit sowie den spezialisierten Funktionseinheiten. Der Programmspeicher enthält die Programme, die auf der PEM abgearbeitet werden sollen. Er hat eine Größe von 128K 64-Bit Worten.

Die Befehlsverarbeitungseinheit beinhaltet zum einen den Register- und Konstantenspeicher, und zum anderen die Kontrolleinheit, die für das Laden und Dekodieren der Instruktionen sowie die Thread- und Taskverwaltung zuständig ist. Jedes PEM besitzt 2048 64-Bit Register sowie im Konstantenspeicher 4096 nur lesbare Register (ebenfalls mit je 64 Bit).

Die Funktionseinheiten schließlich umfassen die synchronen Recheneinheiten (Integer-Einheit, Gleitkomma-Addierer und -Multiplizierer, Lade-/Speichereinheit) und die asynchronen (Gleitkomma-Dividierer), sowie ein paar spezielle Einheiten, die der Kommunikation mit dem internen Netzwerk und dem Erzeugen neuer Threads dienen. Die synchronen Einheiten arbeiten mit einer 8-stufigen Pipeline.

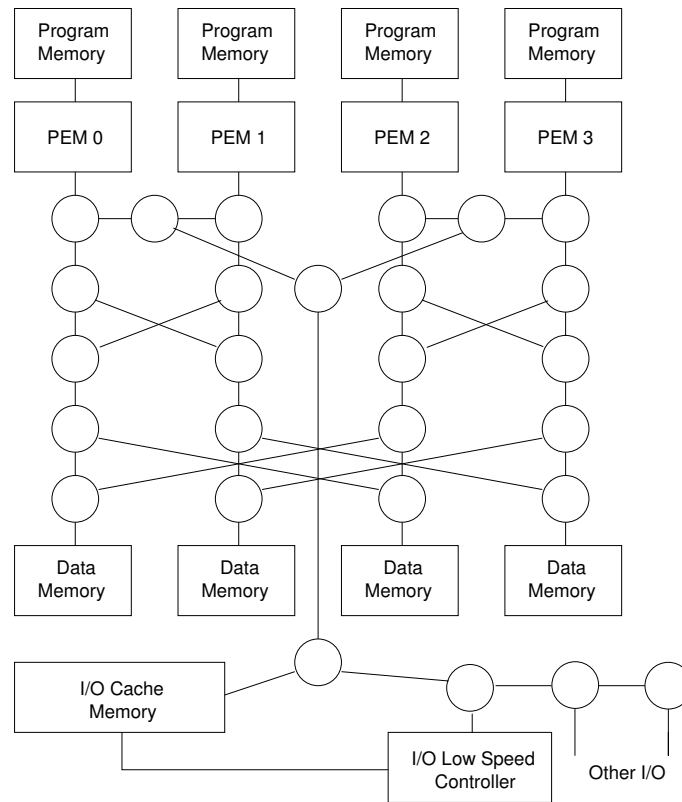


Abbildung 1: Blockdiagramm

Bis zu 16 dieser Module können in einem Rechner verbaut werden. Jede PEM ist mit 10 MHz getaktet, eine PEM erzielt damit bei voller Auslastung der Pipelines der Funktionseinheiten eine Leistung von 10 MIPS.¹ Ein Rechner im Vollausbau ist damit in der Lage, 160 MIPS zu erzielen.

Um langsame Speicherzugriffe zu vermeiden, besitzt jede PEM einerseits 2048 Register, andererseits unterstützt der HEP Fine-Grained Multithreading (siehe Abschnitt 3.1.)

3.1 Fine-Grained Multithreading

Um die Geschwindigkeitendifferenz zwischen den Speichermodulen und der Prozessorgeschwindigkeit zu umgehen, unterstützt der HEP sogenanntes “Fine-Grained Multithreading”. Das bedeutet, dass der Prozessor mehrere Programmfäden quasi gleichzeitig laufen lässt. Immer dann, wenn einer dieser Fäden auf den Speicher zugreift und auf die Beendigung dieses Zugriffs warten muss, wird dieser Faden suspendiert und ein anderer Faden wird abgearbeitet. Auf diese Art und Weise wird die Verzögerung des Speicherzugriffs verschleiert. Man spricht dabei von “Latency Hiding.”

¹Die Funktionseinheiten können nicht gleichzeitig arbeiten, der Prozessor ist also nicht superskalar.

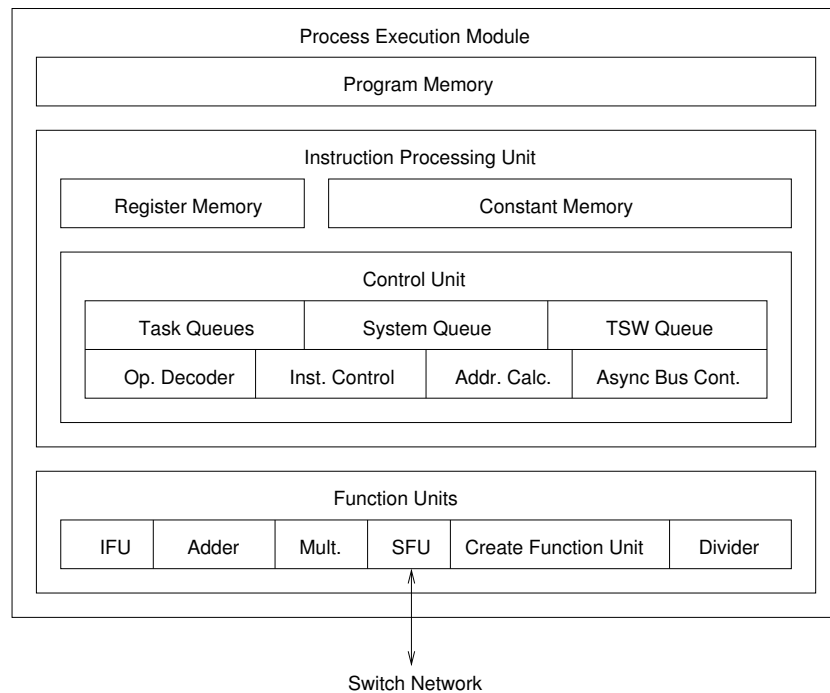


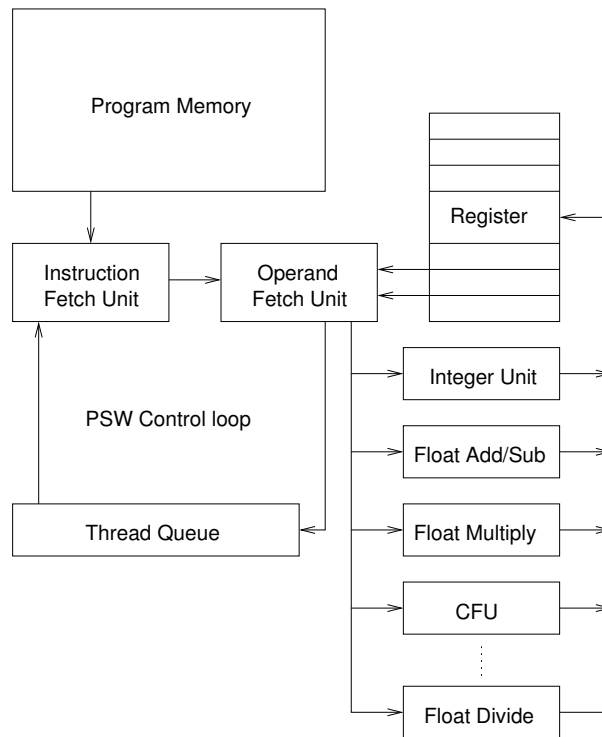
Abbildung 2: Process Execution Module

3.2 Tasks und Threads

Jedes PEM kann 64 Tasks verwalten. Eine Task ist ein eigenständiges Programm, welches wiederum in 64 Threads (Programmfäden) unterteilt werden kann. Ein Task besitzt geschützte Daten- und Programmspeichersegmente, und es gibt User- und Supervisor-tasks. Die Usertasks führen die eigentlichen Anwendungsprogramme aus, während die Supervisortasks Betriebssystemaufgaben wie beispielsweise die Trap-Behandlung übernehmen.

Während die Ausführung einzelner Tasks voreinander geschützt ist, sind die einzelnen Threads nicht geschützt und können auf den selben Speicher zugreifen. Jedem Task wird ein Taskstatuswort (TSW) zugewiesen, das die Basisadressen und Längen der Code-, Daten-, Register- und Konstantenspeicherbereiche enthält. Die Tasks eines PEM werden in einer Task-Queue verwaltet.

Jeder Thread besitzt ein sogenanntes “Process Status Word” (PSW), in dem die Kontextinformationen des Threads enthalten sind, die zum Suspendieren und Starten notwendig sind. Dazu gehört vor allem der Programmzähler. In Abbildung 3 ist dargestellt, wie die PSWs der einzelnen Threads benutzt werden, um mehrere Instruktionsströme quasi-gleichzeitig auszuführen. Zunächst wird ein PSW aus der Thread-Queue genommen. Der darin enthaltene Programmzähler wird benutzt, um die nächste Instruktion aus dem Programmspeicher zu holen. Die Instruktion und das PSW werden dann an die “Operand Fetch Unit” weitergereicht, welche die Operanden aus dem Registerspeicher lädt und den Berechnungseinheiten übergibt. Anschließend wird das PSW wieder in die Thread Queue eingehängt und das nächste PSW wird verarbeitet, d.h.



Process Execution Module

Abbildung 3: PSW Steuer- und Datenschleife

die nächste Instruktion aus dem nächsten Thread wird verarbeitet. Gleichzeitig wird natürlich das Ergebnis der Berechnung in den Registerspeicher zurückgeschrieben. Die Thread Queue enthält ein Verzögerungsglied, so dass die PSW Steuerschleife (links im Bild) und die Datenschleife (rechts) synchron laufen.

Da in jedem Takt eine neue Instruktion aus dem Programmspeicher geholt und den Funktionseinheiten zugeführt wird, befinden sich in den unterschiedlichen Pipeline-Stufen der synchronen Funktionseinheiten Befehle unterschiedlicher Instruktionsströme. Um Schwierigkeiten bei den Abhängigkeiten aufeinanderfolgender Befehle eines Programms zu vermeiden (beispielsweise bei bedingten Sprüngen, die bei anderen Prozessoren zu Pipeline-Stalls führen würden), ist es bei der HEP-Architektur nicht erlaubt, gleichzeitig mehrere Befehle des selben Programms in einer Funktionseinheit zu haben. Das wiederum führt dazu, dass die Pipelines nur dann voll ausgelastet sind, wenn mindestens acht Threads gleichzeitig in einem PEM laufen.

Threads, die auf den Datenspeicher zugreifen, werden gesondert behandelt. Sie werden aus der Thread-Queue entfernt und in einen besonderen Wartebereich verschoben. Sobald das angeforderte Datenwort aus dem Speicher geladen ist, wird der Thread wieder in die Thread-Queue eingefügt.²

Für den Fall, dass nur ein einziges sequentielles Programm läuft, wird somit die

²Dies wird in der Literatur als *split transaction* bezeichnet.

Leistung des Prozessors auf ein Achtel reduziert. Das entspricht einer Maximalleistung von 1.25 MIPS.

Der FORTRAN-Dialekt, in dem der HEP normalerweise programmiert wurde, enthält ein paar Erweiterungen, die die nebenläufige Programmierung unterstützt. Dazu zählen vor allem die Anweisungen `create` und `resume`, die einen neuen Thread erzeugen bzw. auf die Beendigung eines Threads warten.

4 Das Kommunikationsnetzwerk

Das Kommunikationsnetzwerk verbindet die PEMs mit dem Datenspeicher und dem Ein-/Ausgabesystem. Das Netzwerk ist mit 10 MHz getaktet und besteht aus vielen Knoten ("Switch Nodes"), die über Leitungen verbunden sind.

4.1 Switch Nodes

Jeder dieser Knoten besitzt drei bidirektionale Anschlüsse (Ports) und kann zwischen diesen Anschlüssen Verbindungen durchschalten, so dass zwischen den Endgeräten eine Punkt-zu-Punkt-Verbindung entsteht. Die Daten werden paketweise übermittelt, aber nicht in den Knoten zwischengespeichert.

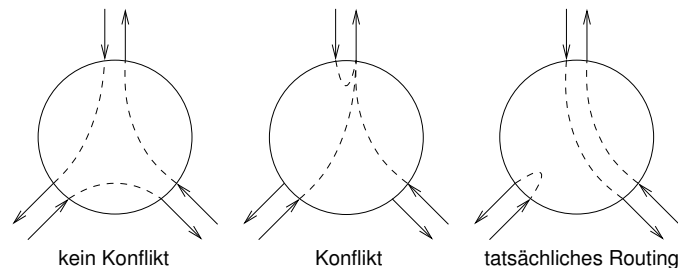


Abbildung 4: Switch Nodes

4.2 Routing

Damit die Pakete ihren Bestimmungsort erreichen, verwaltet jede Switch Node Routinginformationen. Den Paketen wird die Zieladresse entnommen und dann wird die Verbindung durchgeschaltet, die zum Ziel führt.

In Abbildung 4 links ist dargestellt, wie drei Pakete gleichzeitig durch eine Switch Node geleitet werden. Da alle drei Pakete auf unterschiedliche Ausgänge durchgeschaltet werden, tritt kein Konflikt auf. In Abbildung 4 (mitte) dagegen müssten alle drei Pakete auf den selben Ausgang geschaltet werden, um zu ihrem Ziel zu gelangen. Da der Knoten keine Pakete zwischenspeichern und verzögert weiterschicken kann, werden einfach zwei der Pakete fehlgeleitet, in der Hoffnung, dass sie von den angrenzenden Switch Nodes korrekt vermittelt werden können. Für den Fall, dass Pakete in einer Endlosroute im Netzwerk gefangen werden, besitzt jedes Paket ein 4-Bit-Feld, in dem das Alter des

Paketes vermerkt ist. Sollte dieses Alter den Wert 15 erreichen, wird das Paket auf einem vordefinierten Pfad, der alle angeschlossenen Knoten erreicht, durch das Netzwerk geleitet. Damit ist garantiert, dass jedes Paket schließlich sein Ziel erreicht.

5 Speicherorganisation

Die Aufteilung des Speichers in mehrere, unabhängige Einheiten dient der Beschleunigung der Speicherzugriffe. So ist ein PEM in der Lage, gleichzeitig Daten aus dem Datenspeicher zu lesen und weitere Instruktionen aus dem Programmspeicher nachzuladen. Der große Registerspeicher und der Konstantenspeicher für häufig benötigte Konstanten beschleunigen die Operationen zusätzlich, vor allem, weil jedem Thread ein eigener Satz von Registern im Register- und im Konstantenspeicher zugewiesen wird. Einerseits können mehrere Threads sich nicht beim Zugriff auf gemeinsam benötigte Register gegenseitig blockieren, andererseits muss der Registersatz beim Threadwechsel nicht in den Speicher gerettet und später wieder restauriert werden.

Für den Datenspeicher können bis zu 128 Data Memory Modules (DMM) installiert werden, die den von allen PEM gemeinsam genutzten Speicher enthalten. Jedes DMM hat eine Speicherkapazität von 128K 64-Bit Worten. Der Speicher ist wortweise organisiert, es lassen sich aber auch Halb-, Viertelwörter und Bytes adressieren.

Zu jeder Speicherzelle im Datenspeicher wurde ein weiteres Bit vorgesehen, das den Zustand dieser Speicherzelle anzeigte. Eine Speicherzelle konnte damit als *leer* oder *voll* markiert werden ("full/empty bit"). In der FORTRAN-Programmiersprache, die zum Programmieren des HEP benutzt wurde, gibt es eine spezielle Syntax bei Variablenzugriffen, mit denen Speicherzugriffe synchronisiert werden können. Bei Lesezugriffen wartet der Prozessor dann darauf, dass der Zustand einer Zelle auf *voll* wechselt, bevor der Wert ausgelesen und der Zustand wieder auf *leer* gesetzt wird. Bei Schreibzugriffen wird analog darauf gewartet, dass der betreffende Speicherbereich leer wird, und er wird nach dem Schreiben des Datums als voll markiert.

Die Benutzung dieser Speicherzustände wurde im verwendeten FORTRAN-Dialekt durch sogenannte *asynchrone Variable* unterstützt. Im Programmtext wurden diese Variablen durch ein vorangestelltes Dollar-Zeichen markiert, und der Compiler setzte Zugriffe auf solche Variablen so um, dass die full-/empty-Bits benutzt wurden. Abbildung 5 illustriert die Verwendung asynchroner Variablen: Die obere Programmzeile führt dazu, dass das Programm darauf wartet, dass der Variablen $\$S$ ein Wert zugewiesen wird. Sobald das geschehen ist, wird dieser Wert in die Variable R ausgelesen und gleichzeitig der Zustand der Variablen wieder auf *leer* gesetzt. Im zweiten Fall wird gewartet, bis der Zustand der Variablen $\$S$ auf *leer* wechselt. Dann wird der Wert aus R in die Variable geschrieben und der Zustand gleichzeitig auf *voll* gesetzt. Dadurch, dass ein Thread suspendiert wird, bis eine Speicherzelle den gewünschten Zustand erhält, lassen sich mehrere Threads durch asynchrone Variablen synchronisieren. Des Weiteren existiert in HEP-FORTRAN noch eine Anweisung zum expliziten Setzen eines Speicherzustandes auf *leer* (*purge*).

Der Registerspeicher sieht sogar noch einen weiteren Zustand für jedes Register vor: *reserved*. Ein Register wird als reserviert gekennzeichnet, wenn es das Ziel einer Operati-

on ist. Sollte ein anderer Thread einen Wert aus einem als *reserved* markierten Register lesen wollen, wird er suspendiert.

$R = \$S$ — wait for full, read, set empty
 $\$S = R$ — wait for empty, write, set full

Abbildung 5: Asynchrone Variablen in HEP-FORTRAN

6 Ein- und Ausgabe

Für die Ein- und Ausgabe des HEP wird ebenfalls das Kommunikationsnetzwerk verwendet. Sowohl der angeschlossene Massenspeicher als auch langsamere Ein-/Ausgabegeräte (Terminals, Drucker) werden unterstützt.

Für die Anwendung des Rechners beinhaltet der HEP eine komplette PDP 11/44, über die unter anderem die Benutzereingabe und -ausgabe durchgeführt wird.

6.1 Ein-/Ausgabe-Cache

Neben dem Multithreading zur Vermeidung der Latenzzeiten, der sowohl bei Speicher- als auch bei Ein-/Ausgabeoperationen zum Einsatz kommt, wird für die angeschlossenen Massenmedien noch ein Ein-/Ausgabe-Cache eingesetzt.

Dieser hat zum anderen auch die Aufgabe, die Transfer-Geschwindigkeit der langsamen Ein-/Ausgabegeräte an die Geschwindigkeit des Kommunikationsnetzwerks anzupassen. Dieser Datenpuffer ist 80 MB groß und ist aus 400ns MOS Bausteinen aufgebaut, so dass der Cache in der Lage ist, Daten mit 32 MB/s auszuliefern.

6.2 Massenspeicher

Verschiedene Arten von Massenspeichern können an den HEP angeschlossen werden. Zum einen werden bis zu vier Festplatten mit je 128 MB Speicherkapazität unterstützt, des weiteren benutzt der HEP als Peripheriebus den PDP UNIBUS der PDP 11, so dass beliebige Peripherie, die für die PDP 11 angeboten wurde, theoretisch angeschlossen werden könnte. Für größere Datenmengen lassen sich Bandlaufwerke anschließen.

7 Nachteile des HEP-Designs

Einerseits war das Design des HEP zum Zeitpunkt des Entwurfs und der Markteinführung sehr fortschrittlich, andererseits wurden im praktischen Einsatz einige Nachteile sichtbar, die auf grundsätzliche Entwurfsentscheidungen zurückzuführen sind.

Zum einen ist die relativ geringe Leistung bei sequentiellen Programmen zu nennen, da die Pipelines der Funktionseinheiten nicht ausgenutzt werden können. Des weiteren

hat sich herausgestellt, dass 64 Threads pro PEM für viele reale nebenläufige Problemstellungen nicht ausreichen. Da es nicht möglich ist, diese Einschränkung durch das Hinzufügen weiterer PEM zu umgehen und seine Threads auf mehrere Prozessoren zu verteilen, muss man als Programmierer seine Modelle an diese festgelegte Anzahl anpassen. Eine Emulation virtueller Threads, die dann auf echte Threads verteilt werden, existiert nicht. Aber nicht nur für menschliche Programmierer, auch für parallelisierende Compiler besteht dieses Problem.

Die `resume`-Operation in HEP-FORTRAN wurde über asynchrone Variablen implementiert. Das bedeutet, dass die PSW von wartenden Threads weiterhin in der Thread-Queue zirkulieren und damit eine Zeiteinheit verbrauchen, die von anderen Threads genutzt werden könnte.

Einige dieser Nachteile wurden beim Nachfolger des HEP, der TERA-Maschine beseitigt. Dieser Rechner kann beispielsweise 128 Threads verwalten, außerdem wurde die Unterstützung zum Erzeugen und Zusammenführen paralleler Threads verbessert.

8 Zusammenfassung

Bei dem Denelcor Heterogeneous Element Processor handelt es sich um einen Computer, der bei seinem Erscheinen einige Neuerungen brachte: Zum einen einen pipelined Skalarprozessor, zum zweiten Multithreading auf Instruktionsebene und zum dritten das Konzept der MIMD-Programmierung. Der Einsatz eines Paketvermittlungsnetzwerkes war zwar nicht neu, aber die Kombination mit Fine-Grained-Multithreading zur Kompensation der Kommunikationskosten war eine innovative Eigenschaft.

Kurz nach seiner Fertigstellung kamen allerdings neuere Rechnerarchitekturen auf den Markt, die aufgrund der technologischen Fortschritte schnell die Leistungsfähigkeit des HEP übertrafen. Deshalb wurden auch nur fünf dieser Maschinen verkauft und das Nachfolgemodell gar nicht mehr gebaut.

Besonders das Multithreading, das Kommunikationsnetzwerk und die Synchronisierung der Prozesse über Zustandsbits von Speicherzellen waren aber sehr interessante Ansätze.

Die HEP-Architektur und ähnliche Ansätze haben die Prozessor-Entwicklung trotz der kurzen Lebensdauer der HEP-Rechner nachhaltig beeinflusst. Das Alpha-Design der Firma Compaq beispielsweise wird in einer zukünftigen Version (EV8) Multithreading auf Instruktionsebene unterstützen.[2]

Literatur

- [1] Cray Inc. *Cray MTA*.
URL: <http://www.tera.com/products/systems/craymta/>, geprüft am 6. Juni 2001.
- [2] Andreas Stiller. *Architektur für 'echte' Programmierer*. In: c't 13/2001, S. 153.
- [3] Wolfgang K. Giloi. *Rechnerarchitektur*. Springer, 1981.
- [4] Nava Zvaig. *Denelcor HEP Supercomputer*.
URL: <http://wwwee.eng.hawaii.edu/~nava/HEP/HEP-cover.html>, geprüft am 25. Mai 2001.
- [5] Geoffrey C. Fox, Roy D. Williams, Paul C. Messina. *Parallel Computing Works*. Morgan Kaufmann Publishers, 1994.
URL: <http://www.npac.syr.edu/copywrite/pcw/BOOK.html>, geprüft am 25. Mai 2001.