Die Connection Machine

Seminarvortrag am 26.01.2001
zum Thema
Unkonventionelle Rechnerarchitekturen

Steffen Schröder
sschroed@cs.uni-frankfurt.de

25. Januar 2001

Inhaltsverzeichnis

1 Der Weg zur Connection Machine

Der Computer nach dem von Neumann Prinzip arbeitet mit einem Prozessor, der über einen Bus mit dem Speicher verbunden ist. Im Gehirn, das schneller ist, sind jedoch die untereinander verbundenen Nerven Speicher und Rechenwerke. Gerade das Gebiet der künstlichen Intelligenz war es, welches den Entwicklungsanstoß gab.

Die Connection Machine wurde entwickelt, teilweise um dem Traum von der „denkenden Maschine“ näherzukommen, die Firma ihres Erfinders heißt „Thinking Machines Corp.“. Sie wurde später für viele wissenschaftliche Berechnungen, z.B. der Hochenergiephysik oder auch globaler Klimamodelle eingesetzt (1993). Die Connection Machines wurden 1996 aus der Liste der Top500 Rechner entfernt, da Thinking Machines Corp. die Hardwareentwicklung aufgab und sich nun mit Softwareentwicklung zu retten versuchte. Sie spezialisierten sich auf Data Mining und wurden später von Oracle aufgekauft.

Wie jedoch kam W. Daniel Hillis auf diese Idee der Connection Machine?

In konventionellen Rechnern sind Prozessor und Speicher räumlich getrennt angeordnet, aber beide aus Silizium. Wieso?

Als Von Neumann sein Konzept entwickelte, waren die Prozessoren z.B. aus schnellen, teuren Vakuumröhren, der Speicher jedoch aus langsameren, billigeren Materialien, z.B. Verzögerungleitungen, man denke auch an Ringkernspeicher. Kurz: Sie waren technologisch komplett anders aufgebaut. Dies ist heute aber nicht mehr so, was spricht also dagegen, Speicher und Recheneinheiten zu vermischen und einander zuzuordnen?

1.1 Bildverarbeitung

In der Bildverarbeitung wäre es zum Beispiel praktisch, jedem Pixel einen eigenen kleinen Prozessor zuzuordnen, der jeweils mit den Nachbarprozessoren/Pixeln kommunizieren kann. So kann die häufige Aufgabe, für jeden Pixel den Mittelwert seiner Nachbarn auszurechnen mit vielen kleinen, schwachen Prozessoren viel schneller als mit einem großen erledigt werden. Es reichen also kleine Prozessoren mit einem fixen Kommunikationsnetzwerk aus.

1.2 VLSI-Simulation

Hier könnte jedem Transistor ein Prozessor zugeordnet werden, der dann seine Ergebnisse an die mit seinem Ausgang verbundenen Transistoren schickt. Aufgrund der vielen verschiedenen Schaltkreise ist hier ein flexibles Kommunikationsnetz gefragt, die einzelnen Prozessoren müssen nicht viel können.

1.3 Semantische Netze

Diese werden gerne in der KI benutzt, um Gedankengänge zu modellieren. So werden verschiedene Konzepte verknüpft, z.B. *Apfel* mit *rot*, *grün* und *Obst*. Ist jetzt *Birne* auch mit *Obst* verknüpft, kann ein Programm feststellen, dass Birnen und Äpfel Ähnlichkeiten haben, zumal wahrscheinlich auch beide über *wächst am* mit *Baum* verknüpft sein werden.

Hier kann jedem Konzept ein Prozessor zugeordnet werden, die erforderlichen Verknüpfungen können sich sogar dynamisch ergeben.

2 Forderungen

Eine „denkende Maschine“, muss also flexibel einsetzbar sein, die Hardware darf nicht mit zu starren Kommunikationswegen einschränken. Die gewünschten Eigenschaften der Maschine hängen sehr stark vom Einsatzgebiet ab.

Deshalb wollen wir nun anhand eines Algorithmus Forderungen aufstellen. In einem ungerichteten Graphen ist die Länge eines kürzesten Pfades zwischen den Knoten A und B gesucht:

1. Beschrifte alle Knoten mit ∞.
2. Beschrifte A mit 0.
3. Beschrifte alle Knoten mit 1 plus dem Minimum der Beschriftungen der anderen Knoten und sich selbst.
4. Wiederhole Schritt 3, bis B eine Endliche Beschriftung hat.

Dieser Algorithmus ist nicht der beste, aber auf seriell arbeitenden Rechnern nicht wesentlich effektiver zu lösen. Arbeiten in Schritt 3 alle Zellen gleichzeitig, ist das Problem aber einfach.

Wir benötigen wegen der vielen unterschiedlichen Eingabemöglichkeiten

- **viele Prozessoren**, damit wir jedem Knoten einen zuordnen können,
- ein **anpassbares Kommunikationsnetzwerk**, da die Kanten zufällig liegen können,
- eine **erweiterbare Struktur** falls die Eingabegraphen mit der Zeit wachsen und nicht immer komplett neue Maschinen gekauft werden sollen, oder zum gleichen Zweck
- **virtuelle Zellen**, um auch auf kleinen Maschinen mit großen Problemen fertig zu werden. Eine Zelle kann so die Arbeit von mehreren simulieren.
- **Fehlertoleranz**: Wenn ich viele Prozessoren zur Verfügung habe, sollte nicht schon der Defekt eines einzigen die gesamte Maschine unbrauchbar machen.

3 Design der Connection Machine

Die Connection Machine, in der ja jedem Prozessor sein eigener kleiner Speicherbereich zugeordnet ist, wird vom Hostrechner als ein großer RAM-Bereich gesehen (Distributed Shared Memory) und kann über den Memory-Bus auch so beschrieben und ausgelesen werden, um schnelle Initialisierung zu ermöglichen. Der einzige Unterschied zu normalem RAM ist, dass dieses selbst `aktiv` seinen Inhalt ändern kann, die Zellen durch ein beliebiges Kommunikationsnetz verbunden sind und die Ein-/Ausgabe zur Peripherie viel schneller als durch den Host geht.

3.1 Prozessoren

Wir haben gesagt, wir brauchen viele Prozessoren für die Connection Machine. Als erstes fallen einem da „normale“ Universalprozessoren von der Stange ein.

Für viele Anwendungen ist eine solche Leistung pro Zelle aber gar nicht gefordert, man kommt auch mit wesentlich schwächeren aus. Z.B. reicht für symbolisches Rechnen ein sehr schmaler Datenpfad, man kann auch auf FPUs verzichten. Außerdem kann die schiere Zahl von schwachen Prozessoren die Nachteile wieder wettmachen.

Um die Kosten nicht zu hoch zu treiben muss man sich also zwischen vielen schwachen und relativ wenigen mächtigen Prozessoren entscheiden, je nach geplanter Anwendung. Die Entscheidung ist alleine vom geplanten Einsatzgebiet abhängig.

3.2 Kommunikationsnetz

Das Kommunikationsnetz sollte sehr generell, also flexibel ausfallen. Dazu muss jede Zelle schnell mit jeder anderen kommunizieren können, wenn sie will. Da man aber zur Laufzeit keine neuen Drähte ziehen kann, muss man sich für eine feste Netzwerktopologie entscheiden, die dann die generelle Kommunikation simulieren muss.

Eine `Crossbar` ist aber sehr aufwendig zu implementieren, im `Ring` wäre weder Fehlertoleranz möglich, noch eine effiziente Kommunikation zwischen zwei beliebigen Zellen. Ein `Mesh` oder `Torus` ist zwar Fehlertolerant, aber für die Kommunikation nicht schnell genug. Auch im `Baum` gibt es Engpässe an der Wurzel, wenn man ihn nicht anpasst. Im n-`Würfel` ist der Durchmesser gering und die Zellen haben einen nicht zu hohen Grad. Auch redundante Wege sind hier vorhanden, die die Fehlertoleranz ermöglichen.

Die generelle Kommunikation wird dadurch simuliert, dass in jedem Zyklus Nachrichten in das Netz geschickt und dort über Router an das Ziel geleitet werden, aber mit einem höheren Takt, so dass im nächsten Zyklus alle Nachrichten zugestellt sind. Die Router können sehr schnell arbeiten, da sie die Nachrichten nicht verstehen müssen. Auch können diese die Fehlertoleranz gewährleisten, da sie Nachrichten an oder von defekten Zellen einfach ignorieren, bzw. deren Nachbarn die Leitungen zu diesen einfach als belegt kennzeichnen. Auch Staus können je nach Netzwerktopologie über alternative Wege umgeleitet werden. Günstig wäre es auch, wenn die Router uniform aufgebaut sind, falls die Topologie das zulässt.

Verbindungen zwischen zwei Zellen werden als Pointer im Speicher der jeweiligen Zellen realisiert. Einen beliebigen Fanout bei Bäumen erreicht man durch Staffelung.

3.3 Kontrollfluss

Irgendwie müssen die Prozessoren wissen, was sie ausführen sollen. Wenn ich jedem sein eigenes (potenziell von den Anderen verschiedenes) Programmstück geben will (**MIMD**), ist ein aufwändiges Netz zur Befehlsverteilung notwendig (ein Host muss jeden einzeln ansprechen können). Auch die Synchronisation zwischen den Prozessoren ist recht aufwändig und erfordert viel und schnellen Netzwerkverkehr.

Einfacher ist es, alle Zellen parallel mit dem gleichen Datenstrom zu versorgen, d.H. alle machen zu jedem Zeitpunkt das selbe (**SIMD**), nur durch die Auswahl von Flags zur Konditionalisierung können Teile der Zellen einfach die folgenden Befehle ignorieren, d.H. „ihre Zeit verschwenden“.

Beide Konzepte lassen sich dem jeweils anderen simulieren, wenn auch mit teilweise sehr gravierenden Zeitstrafen. Soll ein volles MIMD Programm auf einer SIMD Maschine simuliert werden, muss jeder Prozessor `nacheinander` arbeiten, die anderen warten jeweils. Will ich ein SIMD-Programm auf einer MIMD Maschine simulieren, muss sehr oft synchronisiert werden, da davon ausgegangen wird, dass alle Prozessoren die Daten des vorherigen Schrittes schon berechnet haben, dies ist aber bei unabhängiger Ausführung nicht immer der Fall.

Da die Connection Maschine für sehr große Probleme gedacht ist und es sehr schwer ist, für tausende von Prozessoren jeweils eigene, unterschiedliche Programme zu schreiben (auch für Comiler), wird es meist so sein, dass sowieso ein Großteil der Prozessoren das Gleiche wie die anderen tut, so dass der SIMD-Nachteil nicht so extrem zu Tage tritt, eine MIMD Maschine bräuchte eine sehr schnelle synchronisationsmöglichkeit, um auch SIMD Maschinen effektiv simulieren zu können.

Dann sind aber die Unterschiede nicht mehr sehr groß, die Wahl ist freigestellt, da die Connection Machine typischerweise irgendwo zwischen den beiden Extremen MIMD und SIMD betrieben wird.

4 Implementierungen

Forderungen an die Connection Machine waren also vor allem:

- viele Prozessoren
- generelle Kommunikation
- Erweiterbarkeit

Weiter ist hier nichts festgelegt, weswegen die Implementierungskonzepte grundsätzlich verschieden sein können, weder die Prozessorgröße noch die Anzahl ist festgelegt, ja sie entwickelten sich sogar mit den Zielsetzungen im Laufe der Geschichte der Firma von Ansätzen der Künstlichen Intelligenz zum rein wissenschaftlichen Rechnen.

4.1 CM-1

Die CM-1 wurde 1986 gebaut mit dem Ziel, eine „Denkende Maschine“ zu entwickeln. Da sie nur ein Prototyp sein sollte, der auch ein Jahr später vom nächsten Modell gefolgt wurde, war sie nicht auf Geschwindigkeit, sondern auf Erweiterbarkeit und Eleganz getrimmt.

Bei der Wahl des Prozessors wurde davon ausgegangen, dass man für KI symbolisches Rechnen eher benötigt als numerisches. Diese Fixierung sieht man auch darin, dass nur LISP als Programmiersprache vorgesehen war, die der anderen Wissenschaften wurden vernachlässigt (C, FORTRAN,...).

Um Flexibilität und Geschwindigkeit zu gewährleisten, wurde den Prozessorzellen nur eine 1Bit ALU spendiert, die drei Eingänge hat und deren Wahrheitstabelle, Ziel und Quellregister bei jedem Befehl erneut übertragen werden.

Die Zellen werden alle mit den selben Instruktionen versorgt (SIMD), die sie in Abhängigkeit von Flags ausführen oder nicht.

Das Kommunikationsnetzwerk ist ein 12 dimensionaler Würfel. Es ist packet switched, an jeder Ecke sitzt ein Router, davon ist immer einer für 16 Zellen zuständig. Somit sind also 64K Zellen vorhanden, von denen jeder 4K Speicher zugeordnet sind.

Zusätzlich gibt es noch ein Mesh/Grid über alle Zellen, um 2D-Operationen besser zu unterstützen. Dafür hat jede Zelle noch NEWS-Bits, über die sie mit ihren unmittelbaren Nachbarn kommunizieren kann.

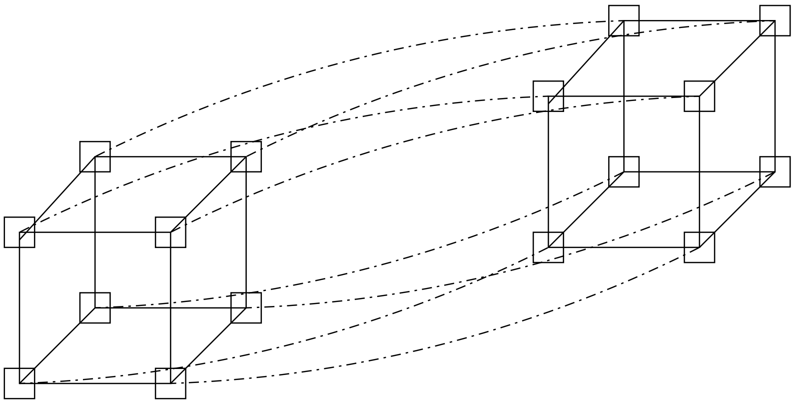

Abbildung 1: Die Topologie des Würfels

Das alles ist gepackt in einen Würfel mit ca. 1,3m Kantenlänge, die Kenndaten sind in Tabelle 1 zusammengestellt.

Speicher	$2,5 \cdot 10^8$bits
Speicherbandbreite	$2,0 \cdot 10^{11}$bps
Prozessorbandbreite	$3,3 \cdot 10^{11}$bps
Kommunikationsbandbreite	$3,2 \cdot 10^7 - 5,0 \cdot 10^{10}$bps

Tabelle 1: Kenndaten der CM-1

Der Hostrechner kommuniziert mit der Connection Machine wie in Abb(2) zu sehen über einen Memory-Bus, mit dem er die Maschine wie normalen Hauptspeicher benutzt, um sie schnell initialisieren und auslesen zu können, nur ist dieser Speicher aktiv, d.H. sein Inhalt kann sich bis zum erneuten Auslesen verändert haben. Dafür gibt der Host Befehle als Makrobefehle an einen Microcontroller, der sie dann in Nanobefehle an die Zellen übersetzt, das aber viel schneller, als es der langsame Host je könnte. Außerdem sind zwischen Host und Microcontroller zwei FIFOs, um asyncron zum Host arbeiten zu können. Es kann ja ein Makrobefehl teils sehr wenige, dann schneller abgearbeitete, Nanobefehle erzeugen, oder auch eine lange Folge, so dass der Host den nächsten

Befehl zu schnell nachschiebt.

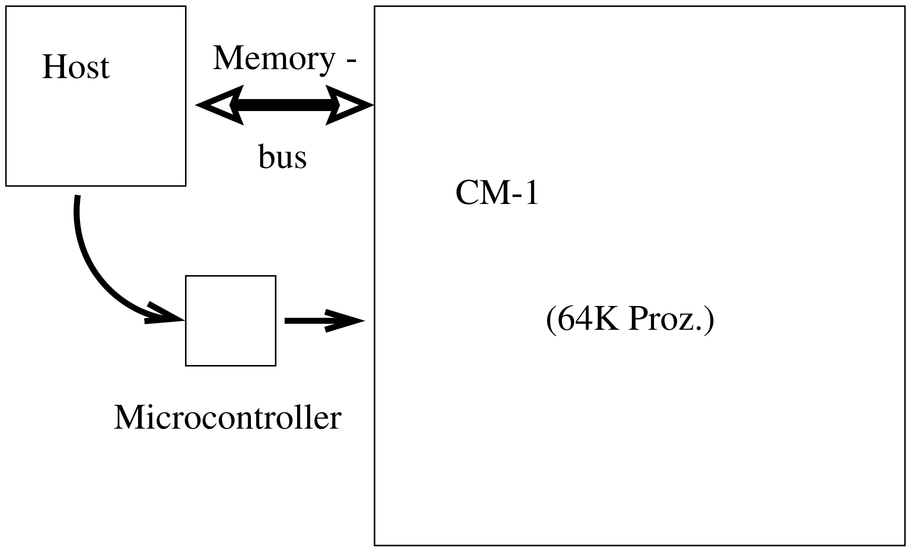

Abbildung 2: Die CM-1 im Host

Die Chips, aus denen die Maschine aufgebaut ist, beherbergen je 16 Zellen mit einem dazugehörigen Router. Ein Chip verbraucht 1W bei 4Mhz! Dazu gehören je Chip noch vier 4Kx4 Memory Chips. Diese Prozessor/Router Chips werden zum Würfel zusammengesteckt. 32 Prozessor/Speicher Einheiten sitzen auf einer Platine, genannt Modul, davon kommen je 16 in eine Backplane, davon zwei in ein Rack, wovon 4 die Würfelform der Maschine ergeben.

4.1.1 Die Prozessorzelle

Diese teibt das RISC-Prinzip auf die Spitze. So gibt es nur einen einzigen Befehl, der aber dafür sehr mächtig ist. Die ALU kann jede mögliche Funktion von zwei Speicherzellen und einem Flag nach einer Speicherzelle und einem Flag berechnen, indem ihr dafür jeweils die Wahrheitstabelle mitgegeben wird. Die Datenpfade sind aber nur ein Bit breit. In jedem Zyklus holt sie 2 Bits und ein Flag aus dem Speicher und schreibt ein Bit und ein Flag zurück. Der Nanocode spezifiziert die Adressen zum Lesen und Schreiben des Speichers und des Flags, die Adresse des Flags, das als Bedingung genutzt wird, ob dieses Flag 1 oder 0 sein soll, damit der Befehl auf dieser Zelle ausgeführ wird, und die Wahrheitstabellen der ALU für Speicher und Flags. Auch die Richtung, in der Daten durch das Grid geschickt werden sollen, ist enthalten.

4.1.2 Konditionalisierung und Flags

Der Prozessor hat 16 Register, davon 8 general purpose und 8 spezielle. Diese beinhalten Daten aus dem Grid, Kommunikation mit dem Router, Parity, Zero und Diagnoseflags, sowie ein Dasy-Chain-Flag, mit dem die 16 Zellen des Chips aufgereiht wie eine 16-Bit-Zelle agieren können.

Eines dieser Bits wird immer zur Konditionalisierung ausgewählt und eines geschrieben.

4.1.3 Der Router

Der Router ist dreigeteilt. Er besteht aus **Injector**, der Pakete ins Netzwerk schickt, **Herz**, für das eigentliche Routing zuständig, und dem **Ejector** für die Zustellung an die Zellen auf dem Chip.

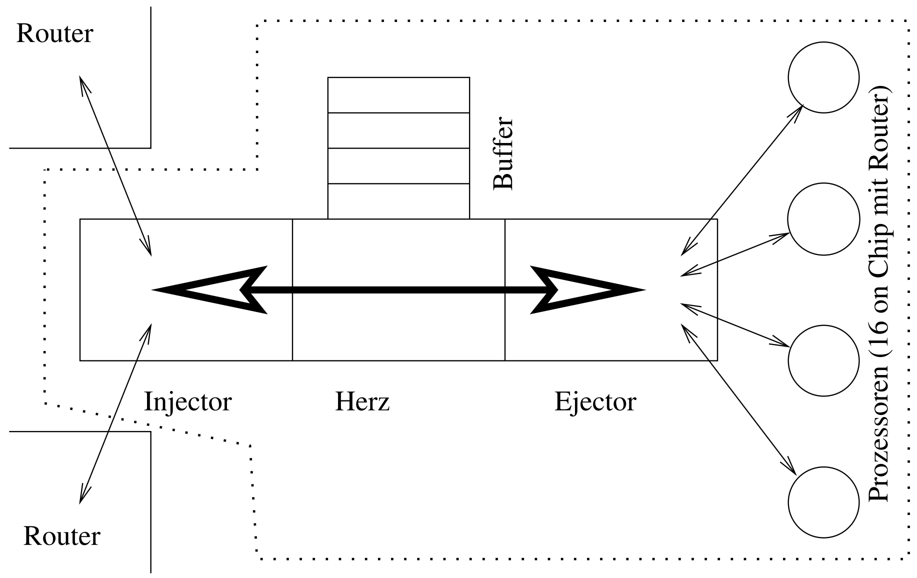

Abbildung 3: Aufbau des Routers in der CM-1

Die Prozessoren schicken ihre zu versendenden Pakete über einen Pin an den Router, der sie zwischenpuffert. Falls er keinen Platz mehr hat, wird das mitgeteilt und der Prozessor muss es später noch einmal versuchen. Dabei besteht jedes Paket aus der Zieladresse, gefolgt von einer 1, den Daten und einem Parity-Bit. Die Adressbits geben an, über welche Dimensionen des Würfels das Paket verschickt werden muss, es handelt sich also um eine relative Adressierung. Der Puffer wird auch von Paketen gefüllt, die über den Injector aus dem Netz kommen.

Dabei werden Pakete, die für Zellen auf dem selben Chip bestimmt sind, lokal zugestellt, alle anderen werden durch das Netz verschickt. Dies geschieht in teilweise mehreren Runden, den **petit cycles**. Diese sind in zwölf Dimensionsphasen aufgeteilt, bei denen in der i-ten Phase jeder Router das erste Paket aus seinem Buffer auswählt, das über die i-te Dimension verschickt werden muss, und tut dies. Dabei wird das i-te Adreßbit invertiert, damit das absolute Ziel gleichbleibt.

In jeder Phase bekommt ein Router also potentiell neue Pakete, die er wieder in seinen Puffer stecken soll. Ist dieser voll, muss er ein Paket nehmen und es über die gerade aktuelle Dimension verschicken, ob es will oder nicht. Dies ist die einzige Situation, in der ein Paket weiter von seinem Ziel wegrückt, als es schon war. Dies passiert aber im Schnitt so selten, dass es in der Laufzeitbetrachtung nicht stört. Diese geht von einer linearen Laufzeit aus, gerade weil die aufsummierte Entfernung aller Pakete von ihrem Ziel in jeder Phase abnimmt, bzw zumindest nicht zunimmt.

Die Fehlertoleranz des Routings wird dadurch erreicht, dass die direkten Nachbarn eines als defekt erkannten Routers Pakete von oder an diesen einfach ignorieren und alle anderen Pakete, deren Weg durch denselben führen würde über andere Dimensionen umleiten.

4.2 CM-2

Die CM-2 erschien genau 1 Jahr nach der CM-1, jedoch hatte man gemerkt, dass es in der KI größere Probleme gab, als gedacht, auch gabe es zu wenig Nachfrage auf dem Markt. So baute man also FPU's ein (2.048 an der Zahl), die sich die Zellen, die immer noch mit 1-bittiger Logik arbeiten teilen. Auch die nun unterstützten C und FORTRAN-Dialekte zeigen die Wende zu numerischem statt symbolischem Rechnen, trotzdem hat sich am Konzept der SIMD-Maschine nichts geändert.

Die CM-2 und der Nachfolger CM-200 waren bis 1991 in der Top500-Liste zu finden, bis sie von der CM-5 abgelöst wurden.

4.3 CM-5

Die CM-5 ist auch eine Connection-Machine, was man an den vielen RISC-Prozessoren mit dem generellen Kommunikationsnetz sieht. Ansonsten ist sie innerhalb der Spezifikation aber grundverschieden.

Sie hat 32, 1.024 oder 16k Prozessoren, also weniger als die CM-1 und CM-2, die beide 64k hatten. Die voll ausgebaute Maschine benötigt ca. 30x30 Meter Platz.

Sie wird nun als CHARM (coordinated homogenous array of RISC microprocessors) Architektur bezeichnet. Das heißt:

- schnelles, breitbandiges Kommunikationsnetz, über das Prozessoren auf die Daten anderer zugreifen können
- schnelles globales Synchronisationsnetzwerk, das es erlaubt, die gesamte Maschine in einen definierten Zustand zu bringen.

Man ist nun vom SIMD-Konzept abgekommen, da in den meisten Berechnungen, auch für das datenparallele Programmieren nur definierte Synchronisationspunkte notwendig sind, und die Synchronisation der unabhängig arbeitenden Prozessoren über das zweite Netzwerk sehr schnell geschehen kann. Der Speicher ist weiterhin den Prozessoren zugeordnet, wenn auch nicht mehr so eng auf einem Chip wie bei den Vorgängermodellen, aber globale Adressierung ist aufgrund des schnellen Kommunikationsnetzwerks auch möglich. Insgesamt ist die Maschine sehr flexibel einsetzbar und wurde sogar zur Simulation von anderen Hardwarearchitekturen und Netzwerkverhalten eingesetzt.

Die Bandbreite des Kommunikationsnetzes steigt linear mit der Prozessorzahl (im Gegensatz zu Bussystemen) und die I/O-Kapazität ist auch ausbaubar, da dafür bestimmte Knoten „am Rand“ zuständig sind, die ihre Bandbreite addieren, so dass nur ein schnelles Speichermedium sichtbar ist.

Durch das Kommunikationsnetz, welches auch Bound-Checking-Hardware beinhaltet, kann das Unix-Betriebssystem des Hosts die Maschine nicht nur multitasking, sondern auch partitioniert mehreren Benutzern/Prozessen zur Verfügung stellen, die dann kleinere, vollkommen unabhängige Maschinen sehen. Dafür werden in die Partitionen Control Nodes eingebaut, die annähernd den Host-Rechnern der früheren Maschinen entsprechen.

Auch virtuelle Zellen und Auslagerung von Zellen werden über die I/O-Knoten unterstützt.

4.3.1 Der Prozessor

Hier werden SPARC Prozessoren benutzt, die mit 32MHz (NetzwerkRouter: 40MHz) getaktet sind, und je 32MB Speicher sowie zur Bandbreitenverbesserung noch Gleitkommachips zur Seite gestellt bekamen. Dazu kommt noch eine 64-Bit Vektoreinheit. Weiterer Fortschritt in der Prozessortechnik war vorgesehen, so dass diese sich über die Zeit hätten ändern können.

Leider verlassen sich diese konventionellen Prozessoren zur Leistungssteigerung auf einen Cache, der bei massiv parallelen Anwendungen mit vielen Datentransfers eher stört, da er selten aktuell ist, nur die ungecachete Bandbreite interessiert. Daher wurde die Bandbreite mit vier zusätzlichen Gleitkommapfaden vergrößert.

Die CM-5 hat bei wachsender Problemgröße (mit entsprechender Prozessorzahl) näherungsweise linearen Speedup (Linpack) von ca 45MFLOPS/node.

4.3.2 Kontrollfluss

Durch ein zweites Kontrollnetz als schnellen Synchronisationsmechanismus ist das datenparallele Programmiermodell wie bei SIMD-Maschinen auch auf dieser eigentlich MIMD-Maschine möglich. Meist arbeitet sie irgendwo zwischen MIMD und SIMD, der Compiler kann alles erzeugen, indem er mehr oder weniger Synchronisationspunkte setzt. Aber auch in datenparallelen Programmen sind asynchrone Stücke möglich und nicht von Nachteil.

Message-Passing Programme sind genauso möglich, wenn man auf die Synchronisation weitgehend verzichtet, je nach Compiler und Vorlieben des Benutzers.

Das Kommunikations- und das Kontrollnetz werden über Memory-Mapped-FIFOs angesprochen, damit man den Wechsel in die Systemebene spart. Schutzmaßnamen trifft die MMU.

4.3.3 Das Kommunikationsnetz

Hier werden wieder die Datenpakete mit Adresse, gefolgt von Daten übertragen. Man ist von der Würfelstruktur abgekommen und benutzt ein anderes logarithmisches Netzwerk mit geringem Durchmesser, den Baum. Da dieser an der Wurzel leicht Verstopfungen hervorruft, benutzt man den **Fat Tree**, bei dem Knoten nahe der Wurzel mehrfach ausgelegt sind, genauer hat jeder Knoten genau 4 Nachfolger und 2 oder 4 Eltern, je nach Tiefe im Netzwerk. Die Prozessoren, Control Nodes und I/O-Knoten an den Blättern haben je 2 mal 8 bidirektionale Leitungen zu zwei Eltern (4 hoch, 4 runter), die Router der ersten beiden Hierarchieebenen, die noch auf der Backplane sitzen, haben zusätzlich je

4 Leitungen nach unten, da ein 4-ärer-Fat-Tree benutzt wird. Weiter oben hat jeder Router 4 Eltern. Pakete, die nach oben wandern wählen pseudozufällig einen freien Vorgänger aus, damit keine nachteiligen Kommunikationspatterns auftreten können. Jede Verbindung hat eine Bandbreite von 20MB/sec.

Diese Struktur gewährleistet eine hohe **Fehlertoleranz**, wie z.B. auch der Würfel, Zusätzlich ist dieser Fat Tree leichter erweiterbar, das Hinzufügen neuer Knoten ist leicht und die Bandbreite vergrößert sich linear mit der Anzahl der Knoten.

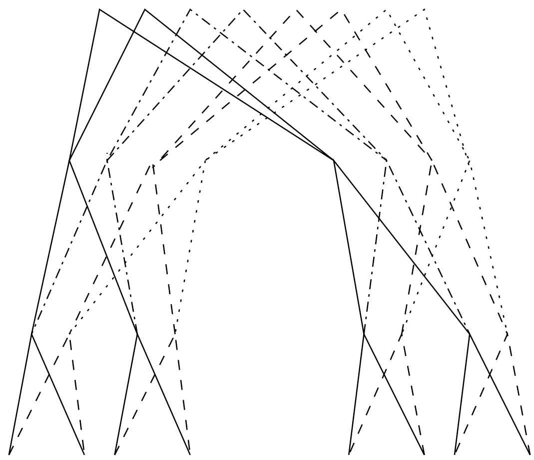

Abbildung 4: Ein Binärer Fat Tree

Die hierarchische Struktur des Fat-Tree ermöglicht auch die **Partitionierung** der Maschine in vollkommen unabhängige Teile, Nachrichten an Prozessoren ausserhalb werden einfach nicht in die höhere Ebene weitergeleitet. Ein normaler Benutzer kann sowieso die Knoten nur relativ zur Partition und nicht absolut adressieren.

Fehlererkennung wird im Netz über Prüfsummen vorgenommen, da den Prozessoren garantiert wird, dass jede Nachricht ausgeliefert wird, solange sie jede an sie selbst adressierte auch wirklich abholen.

Das **Timesharing** einer parallelen Maschine ist normalerweise kompliziert. Was soll mit Paketen vom vorigen Benutzer geschehen, die noch im Netz unterwegs sind, damit sie den neuen nicht behindern? Hierfür werden die Router (nacheinander) in einen all-fall-down Modus versetzt, in dem sie jede Nachricht vom alten Benutzer nach unten fallen lassen. Jede solche von oben empfangene Nachricht wird (egal in welchem Modus der Router ist) sofort nach unten weitergeleitet. So werden auch Nachrichten, die alle an einen einzigen Prozessor adressiert waren, auf viele andere zum Zwischenpuffern aufgeteilt, es kommt zu keinen Verstopfungen. Ist der Benutzer wieder an der Reihe, werden die gepufferten Nachrichten erneut auf ihren Weg geschickt.

Deadlocks können im Netz bei Benutzung der vorgesehenen Compiler nicht auftreten, da immer nur die rechte Seite des Netzes benutzt wird, um Anforderungen rauszuschicken, die linke, um solche zu bearbeiten. Somit ist das Netz schon fast wieder in zwei einander überlappande Teile geteilt.

4.3.4 Das Kontrollnetz

Dieses macht den Broadcast von Informationen, z.B. Codestücken zur Ausführung, die die Prozessorknoten zwischenspeichern, aber auch für die Synchronisation ist es zuständig. Zur Unterstützung typischer paralleler Programme sind globale Verknüpfungen vorgesehen. Das Datenformat ist hier strenger als beim Datennetz auf 65 Bytes festgelegt.

Die Struktur des Netzwerks ist ein einfacher Baum, da hier keine so großen Bandbreiten gebraucht werden, Broadcasts können ja auch am Router vervielfacht werden o.ä. Immer nur eine Zelle kann einen **Broadcast** auslösen, dessen Paket in der Partition nach oben wandert, sich dort umkehrt und nach unten auffächert. Treffen sich mehrere Broadcasts, wird ein Fehler ausgelöst. Allerdings sind die Broadcasts einer Zelle gepipelined, so dass sie viele schnell hintereinander senden kann.

Das Kontrollnetz unterstützt auch **Reduktionen** (globales OR, XOR, MIN, ADD) über alle teilnehmenden Zellen, **Präfix** und **Postfix** dieser Operationen über die Zellen der Partition (das kann man z.B. benutzen, um Felder einer Datenbank aufzusummieren, oder für den Test, ob alle Zellen ein bestimmtes Datum enthalten, ...). Auch teilt es den Prozessoren mit, wann alle Datennetzwerkoperationen abgeschlossen sind, was wichtig für das datenparallele Programmieren ist.

Jeder Prozessor kann aber auch bei seinem Router mit einem speziellen Paket beantragen, nicht von solchen globalen Operationen belästigt zu werden. Dieser sendet dann immer das entsprechende Identitätsdatum.

4.3.5 Das Diagnosenetz

Dies ist auch eine Baumstruktur, ein nicht notwendigerweise vollständiger Binärbaum über die ganze Maschine, wobei an der Wurzel Diagnoseprozessoren sitzen und an den Blättern Pods (Chips auf topologischen Einheiten wie z.B. Platinen), die die eingebauten Diagnosemöglichkeiten nach JTAG der einzelnen Chips auslesen. Somit kann jeder Knoten unabhängig vom Zustand inspiziert werden.

Der JTAG Standard (IEEE 1149.1) Spezifiziert ein 4 Pin Interface für Chips, über den eine Scan Chain angesprochen erden kann, die dann die einzelnen Funktonseinheiten des Chips prüft, auch mehrere Chips können aneinander gereiht werden. Über ein Boundary Scan Register können (im Betrieb) die in den Chip ein oder ausgehenden Daten gelesen werden, auch Teststimuli können generiert werden.

Um nicht alle Chips in Reihe schalten zu müssen, wurde das oben erwähnte Netz eingeführt, weil die serielle Analyse über zehntausende von Chips zu lange dauern würde. Deshalb hat man auch ganze Platinen o.ä. als ganzes, Zellen und Chips von der Wurzel aus direkt adressierbar gemacht.

Leider unterstützen weder der SPARC Prozessor noch das RAM diesen Standard in dem ersten Modell der CM-5.

Dieses Netz ist für den Benutzer nie sichtbar. Adressiert wird hier von der Wurzel aus z.B. als 01B0... das heißt für das Paket: Gehe erst nach links, dann nach rechts, dann nach links und rechts, dann wieder nach links, ... Auch mehrere Pfade können überlagert werden, indem sie nacheinander spezifiziert und nicht gelöscht werden.

Ein Fehler wird durch ein globales AND oder OR bestimmt, je nach erwartetem Ergebnis, wobei durch Folgenachrichten die genaue Lokalisation eingegrenzt werden kann.

Getestet wird hier die Funktion der einzelnen Chips (Prozessoren und RAM nicht direkt), sogar die Router der einzelnen Netzwerke, also auch das circuit-switched Diagnosenetzwerk selbst, können so überprüft und bei Bedarf für den Benutzer transparent ausgeblendet/ersetzt werden.

Dieses Netz war trotz der Kosten notwendig, da man sich bei 16k Prozessorknoten lieber nicht auf die Fehlerfreiheit jedes einzelnen verlassen sollte (Fertigungstoleranzen,...)

5 W. Daniel Hillis

Danny Hillis wurde 1956 in Baltimore, Maryland geboren, verbrachte aber einen Großteil seiner Jugend in Afrika und Indien, wo sein Vater als Epidemiologe arbeitete, und beschäftigte sich dort viel mit der Biologie. Später interessierte er sich für „Computer“, die er teilweise sogar selbst bastelte. In einer Bibliothek fand er das für ihn damals faszinierende Buch *An Investigation of the Laws of Thought* von Boole, woraus er die Konzepte UND, ODER und NOT behielt.

Am MIT wollte er Neurophysiologie studieren, da er das Gehirn genauer kennenlernen wollte, dort wurde ihm von Marvin Minsky erzählt, der in der KI forschte. Das interessierte ihn, und er tat vieles, um ihn kennen zu lernen. Nachdem er dessen Computerprojekt verbesserte, zog er bei ihm im Keller ein und sie lernten sich besser kennen.

So entwickelte sich der Wunsch nach der Denkenden Maschine. Auf seiner Connection Machine simulierte Hillis auch später noch „Leben“, wobei er die Evolution an Sortieralgorithmen austestete, die sich entwickeln sollten. Hier setzte er auch Parasiten, die stören sollten, ein, wodurch sich die besseren Algorithmen schneller durchsetzten

Insgesamt ist sein Schaffen sehr von der Biologie, der Künstlichen Intelligenz und später auch ein Wenig von der Eleganz der Mathematik beeinflusst.

6 Zusammenfassung

Das Konzept der Connection Machine ist sehr flexibel, nicht einengend:

- viele Prozessoren — generelle Kommunikation

Die Implementierungen können sehr unterschiedlich sein, gehörten aber einige Zeit zu den stärksten Rechnern der Welt.

Durch die vielen Prozessoren (die auch virtuell sein können) und die generelle Kommunikation sind sogar Simulationen anderer Rechnerarchitekturen im laufenden Betrieb möglich.

Dem Designer sind für seine Maschine je nach Verwendungszweck fast keine Grenzen gesetzt.

Trotzdem ist diese Architektur nach einiger Zeit der Blüte verschwunden, aber immerhin war sie einer der ersten Schritte vom Ein-Prozessor-Großrechner zum massiv-parallelen Rechnen.

Auch zeigte sie, dass man viel mehr Anwendungen als gedacht parallelisieren konnte, wenn nur die Datenmengen groß genug sind.

Literatur

[1] *TMC Report.* http://www.toc.lcs.mit.edu/classes/6.972/TMC%20Corp.html.

[2] C. Lazere D. Shasha. *Out of Their Minds.* Copernicus / Springer-Verlag, 1995.

[3] Charles E. Leiserson et al. *The Network Architecture of the Connection Machine CM-5.* ftp.theory.lcs.mit.edu/pub/bradley/jpdc95.ps.Z, 1996.

[4] W. Daniel Hillis. *The Connection Machine.* MIT Press, Cambridge, 1982.

[5] W. D. Hillis und L. W. Tucker. *The CM-5 Connection Machine: a scalable supercomputer.* Communications of the ACM, 1993.