EFFIZIENZ PARALLELER RECHNER

Z. Galil und W. Paul

Dieser Aufsatz gibt einen informalen und gerafften Überblick über einige Ergebnisse aus [PV 79] und [GP 80]. Er beschäftigt sich mit der Frage, wie man einen Computer mit sehr vielen Prozessen verdrahten soll.

Wir beginnen mit einigen historischen Bemerkungen. Mit der Turingmaschine hatte man bereits Anfang der 30er Jahre ein Konzept für Rechenanlagen, das einerseits technisch realisierbar war und es andererseits zumindest prinzipiell gestattet, jede überhaupt algorithmisch lösbare Aufgabe zu lösen. Dafür, daß sich diese Maschinen in der Praxis nicht durchgesetzt haben, scheinen zwei Schwächen des Modells verantwortlich zu sein:

(1) das verallgemeinerte Tonbandgerät-Design ist technisch unattraktiv und schwerfällig

(2) auf den ersten Blick scheint es, als ob man für jede Funktion, die man berechnen will, eine eigene Maschine bauen müßte. Solange man das glaubt, wird man eine Maschine für eine spezielle Aufgabe nur dann bauen, wenn man ein sehr großes Interesse an der Lösung gerade dieser Aufgabe hat. Zumindest muß man vorher wissen, welche Aufgabe man später mit der Maschine lösen will.

Wie wir wissen ist die zweite Schwäche behebbar, aber das Programmieren universeller Turingmaschinen ist natürlich denkbar umständlich.

Der von-Neumann-Maschine haften diese Schwächen nicht mehr an. Ihre hervorragenden Merkmale sind

(3) technisch die 2-Teilung der Maschine in eine kleine aktive Komponente (den Prozessor) und eine große passive Komponente (den Speicher).

(4) logisch die Universalität: die Maschine kann relativ komfortabel so programmiert werden, daß sie jede andere Maschine simuliert.

Man kann sich also einen von-Neumann-Rechner kaufen, ohne im voraus eine genaue Vorstellung davon zu haben, was man später damit ausrechnen will. Besser noch, man weiß sogar, daß man (wenn man von der Beschränkung des Speichers absieht) zur Lösung jeder überhaupt algorithmisch lösbaren Aufgabe gewappnet sein wird.

Universalität ist unabhängig von der verfügbaren Technologie eine höchst wünschenswerte Eigenschaft von Rechenanlagen. Die Teilung der von-Neumann-Maschine in eine kleine aktive und eine große passive Komponente scheint hingegen auf eine Technologie zugeschnitten, wo logische Bausteine (Röhren, Transistoren) teuer sind. Inzwischen

sind logische Bausteine sehr billig geworden. Man kann Taschenrechner und Armbanduhren, die zu den erstaunlichsten Kunststücken fähig sind, für ein Spottgeld kaufen... und ein Rechner mit 1 Million Prozessoren ist wohl bald technisch realisierbar. Was kann man durch den Bau eines solchen Rechners gewinnen, und wie soll man überhaupt so viele Prozessoren verdrahten?

Es ist klar, daß man mit N Prozessoren nicht mehr Aufgaben lösen kann, als mit einem einzigen. Man könnte aber hoffen, gewisse Aufgaben bis zu N mal schneller lösen zu können, was für die praktische Lösbarkeit einer Aufgabe natürlich von entscheidender Bedeutung wäre. Das folgende Beispiel zeigt, daß dieser Effekt in der Tat auftritt.

Die Verteilung der Temperatur T auf dem Rand eines Stücks Blech wird von außen konstant gehalten. Wir interessieren uns für die Verteilung der Temperatur für sämtliche Punkte des Blechs. Die Verteilung der Temperatur auf dem Blech wird durch eine unter Physikern wohlbekannte Differentialgleichung beschrieben. Um die Verteilung der Temperatur auf dem Blech zu finden, muß man eine Randwertaufgabe lösen, was analytisch höchstens dann gelingt, wenn sowohl die Form des Blechs als auch die Verteilung der Temperatur auf dem Rand einfach sind. Numerisch weiß man sich jedoch zu helfen, indem man die Aufgabe diskretisiert:

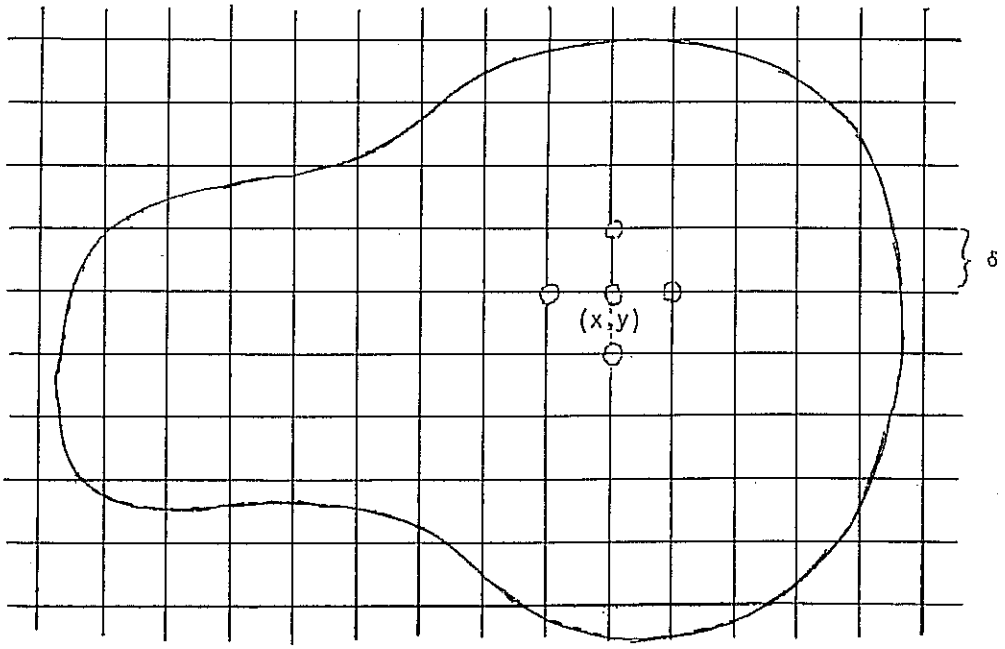


Bild 1

Man legt ein Gitter mit Maschenweite δ über das Blech und löst die Aufgabe iterativ. $t_i(x,y)$ bezeichne die Näherung der Temperatur $T(x,y)$ im Gitterpunkt (x,y) nach dem i-ten Iterationschritt. Man beginnt mit

$$t_0(x,y) = \begin{cases} T(x,y) & \text{für } (x,y) \text{ auf dem Rand} \\ \text{mehr oder weniger beliebig} & \text{sonst} \end{cases}$$

Aus der Differentialgleichung erhält man eine Regel zum iterieren, indem man Differentialquotienten (etwa $\frac{\partial t(x,y)}{\partial x}$) durch Differenzenquotienten ($\frac{t(x+\delta,y) - t(x,y)}{\delta}$) ersetzt und beispielsweise nach einem Vorkommen von $t(x,y)$ auflöst. Man erhält so mit einer passenden Funktion f zum Beispiel

(5) $t(x,y) = f(t(x,y), t(x-\delta,y), t(x+\delta,y), t(x,y-\delta), t(x,y+\delta))$.

Indiziert man das t auf der linken Seite der Gleichung (5) mit i und die t's auf der rechten Seite mit $i-1$, so erhält man eine Iterationsregel, und man kann in vielen Fällen zeigen, daß die Folgen $t_i(x,y)$, $i \in \mathbb{N}$ gegen einen Wert konvergieren, der in Abhängigkeit von der Maschenweite δ mehr oder weniger nahe bei der gesuchten Lösung $T(x,y)$ liegt.

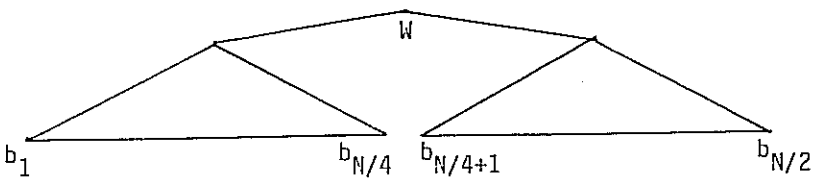
Ist N die Anzahl der Gitterpunkte, so muß ein von-Neumann-Rechner für eine Iteration N mal die Funktion f auswerten. Säße hingegen auf jedem Gitterpunkt (x,y) ein Prozessor $P(x,y)$, der die Aufgabe hat, die Folge $t_i(x,y)$, $i \in \mathbb{N}$ zu berechnen, so könnte für jedes i die i-te Iteration wie folgt durchgeführt werden: jeder Prozessor $P(x,y)$ konsultiert seine Nachbarn um die Daten $t_{i-1}(x-\delta,y)$, $t_{i-1}(x+\delta,y)$, $t_{i-1}(x,y-\delta)$, $t_{i-1}(x,y+\delta)$ zu beziehen und wertet dann die Funktion f aus. Da die Prozessoren $P(x,y)$ gleichzeitig arbeiten, vergeht für eine Iteration jetzt im wesentlichen nur die Zeit, die gebraucht wird um 1 mal die Funktion f auszuwerten.

Durch paralleles Rechnen kann man also manche Aufgaben sehr viel schneller lösen, als sequentiell. Durch Kombination von Techniken aus [PR 78], [PR 79] und [GP 80] kann man sogar zeigen, daß man t Schritte auf dem von-Neumann-Rechner (formal: RAM[+,-] mit logarithmischem Kostenmaß [AHU 74]) in $O(t \text{ loglog } t/\log t)$ Schritten durch einen vollständigen binären Baum der Tiefe $O(t \text{ loglog } t/\log t)$ von endlichen Automaten simulieren kann (sofern $t \geq n \log n/\text{loglog } n$, wobei n die Länge der Eingabe ist). Das heißt, daß kein Problem so inhärent sequentiell ist, als daß man es nicht durch massiven Einsatz paralleler Rechenkapazität schneller lösen könnte, als auf einer sequentiellen Maschine. Praktisch ist das Ergebnis in dieser Form bedeutungslos, da man für $t = 400$ bereits mehr Automaten braucht, als Elektronen im Universum Platz haben.

Nun mögen immerhin N = ca 1 Million in Form eines vollständigen binären Baums der Tiefe $\log N-1$ verdrahtete Prozessoren als attraktives Design erscheinen. Es gibt jedoch sehr einfache Aufgaben, die sich auf einem Baum von Prozessoren nur sehr mühsam und auf anderen einfachen Netzen sehr effizient lösen lassen:

Nehmen wir an, der Baum ist mit endlichen Automaten besetzt, jeder Automat hat z Zustände, und die Eingabe erfolgt bitweise auf den Blättern $b_1 \ldots b_{N/2}$.

Bild 2



Falls das Rechnernetz entscheidet, ob die Eingabe ein Palindrom ist, d.h. vorwärts wie rückwärts gelesen werden kann, so kann man sich leicht überzeugen, daß der Automat an der Wurzel W des Baums für verschiedene Palindrome der Länge $N/2$ verschiedene Folgen von Zuständen durchlaufen muß. Ist t die längste Zeit, die beim Erkennen von Palindromen der Länge $N/2$ auftritt, so folgt

(6) $(z+1)^t \geq 2^{N/4}$ bzw.

$t \geq N/4 \log(z+1)$.

Erweitert man das Netz für $i \in \{0,\ldots,N/4-1\}$ um die Verbindungen $(b_{N/4}-i, b_{N/4}+i+1)$, so läßt sich die gleiche Aufgabe in $O(\log N)$ Schritten lösen.

Ein ähnlicher Effekt tritt auch auf, wenn wir N Automaten in einem $\sqrt{N} \times \sqrt{N}$ -Gitter verdrahten.

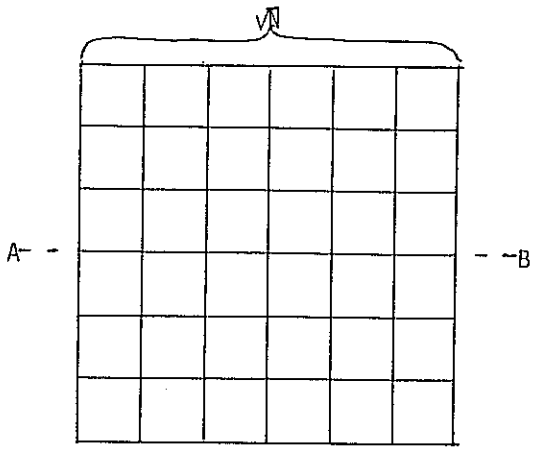


Bild 3

Besteht die Eingabe aus N Bits, von denen die Automaten der ersten Zeile die ersten $\sqrt{N}$ Bits lesen, die Automaten der zweiten Zeile die zweiten $\sqrt{N}$ Bits etc., und es soll entschieden werden, ob die Eingabe ein Palindrom der Länge N ist, so bilden die Automaten auf dem Schnitt $A\ldots B$ in Bild 3 einen Flaschenhals für den Informationsfluß; es folgt

(7) $(z+1)^{t\sqrt{N}} \geq 2^{N/2}$ bzw.

$t \geq \sqrt{N}/2 \log(z+1)$.

Hier drängt sich die Vermutung auf, daß es für jedes Netz G eine Aufgabe A und ein Netz G' gibt, so daß A auf G' erheblich schneller als auf G gelöst werden kann. Modelliert man Netze aus sehr vielen Prozessoren durch unendliche Netze (ebenso wie man große endliche von-Neumann-Rechner durch unendliche Registermaschinen modelliert), so läßt sich das auch durch ein einfaches Diagonalargument zeigen [GP 80]. Im endlichen Fall tritt keines der obigen Probleme auf, wenn wir einfach jedes Paar von Prozessoren direkt verbinden, aber das scheint schon für relativ kleine Prozessorzahlen N technisch nicht machbar zu sein.

All dies liefert Evidenz dafür, daß man mit keinem Netz sämtliche Aufgaben optimal lösen kann. Wird man deshalb parallele Rechner nur für ganz spezielle Aufgaben bauen? Und selbst falls man im voraus weiß, daß man numerisch die Temperaturverteilung auf Blechstücken ausrechnen will: welche Form soll man dem Gitter von Prozessoren geben? Ein quadratisches Gitter ist sicher gut für mehr oder weniger quadratische Blechstücke, aber schon viel weniger gut für sehr lange dünne. Ganz zu schweigen davon, daß man mit der gleichen Anlage vielleicht auch gerne Eigenschaften von Kristallen ausrechnen würde, wozu ein dreidimensionales Gitter von Prozessoren viel angebrachter wäre.

Das Hauptergebnis von [GP 80] besagt, daß es einen Ausweg über universelle "programmierbare" Netze gibt, die jedes andere Netz mit gleich vielen Prozessoren wenigstens einigermaßen effizient simulieren können. Wir skizzieren die entsprechenden Konstruktionen.

Sei ein beliebiges Netz G von P Prozessoren gegeben, in dem jeder Prozessor mit höchstens d anderen Prozessoren direkt verbunden ist; d heißt der Grad von G. In einem Schritt des Netzes G kann jeder Prozessor mit einem seiner Nachbarn Daten austauschen. Diese Kommunikation zwischen Prozessoren in einem beliebigen Netz G muß ohne zu großen Zeitverlust von einem festen Netz G_0 mit ebenfalls P Prozessoren hergestellt werden.

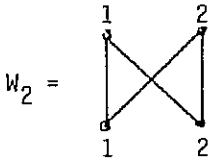
Mit Hilfe des Heiratssatzes kann man zeigen, daß sich die Verbindungen eines beliebigen Netzes vom Grad d in d Klassen $E_1,\ldots,E_d$ einteilen lassen, so daß für jedes i das Netz G_i, in dem nur die Verbindungen aus E_i vorkommen, Grad 1 hat. Ein Schritt im Netz G wird simuliert, indem für $i = 1,\ldots,d$ nacheinander Kommunikation entlang den Verbindungen aus E_i simuliert wird. Es genügt also, Netze F vom Grad 1 zu simulieren. Numerieren wir die Prozessoren eines Netzes von 1 bis P, so definiert ein Netz vom Grad 1 eine Permutation π_F auf $(1,\ldots,P)$ nämlich

$$\pi_F(i) = \begin{cases} i & \text{falls Prozessor } i \text{ mit keinem anderen Prozessor verbunden ist} \\ \text{die Nummer des Prozessors, mit dem Prozessor } i \text{ verbunden ist} & \text{sonst.} \end{cases}$$

Die Waksman-Permutationsnetze W_P sind induktiv durch Bild 4 definiert [W 68]. Man kann zeigen, daß es in W_P zu jeder Permutation π auf $(1,\ldots,P)$ P Wege $w_1(\pi),\ldots,w_P(\pi)$ gibt, wobei jeweils $w_i(\pi)$ am Eingang i beginnt und am Ausgang $\pi(i)$ endet.

Verdrahten wir P Prozessoren, indem wir sie auf die Eingänge von W_P setzen, die inneren Knoten von W_P mit zusätzlichen einfachen Prozessoren bestücken und die Ausgänge von W_P mit den Eingängen identifizieren, so erhalten wir ein Netz, das einen Schritt in einem beliebigen Netz F von Grad 1 in $2 \log P$ Schritten simulieren kann: man stellt für jedes i die Kommunikation zwischen Prozessor i und Prozessor $\pi(i)$ entlang dem Pfad $w_i(\pi_F)$ her; diese Pfade haben ungefähr Länge $2 \log P$.

In dieser Konstruktion werden allerdings P Prozessoren von $2 P \log P$ Prozessoren simuliert, von denen jeder nur alle $2 \log P$ Schritte wirklich etwas tut. Preparata



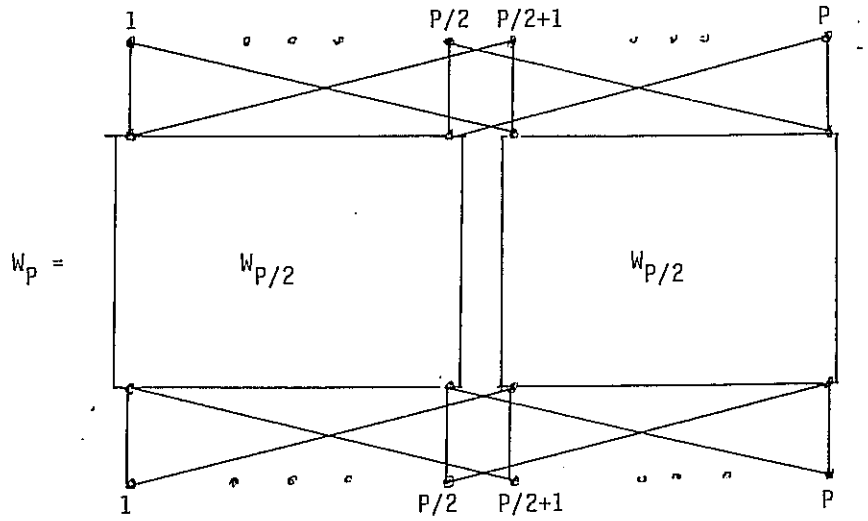


Bild 4

und Vuillemin haben erkannt, daß man in dieser und vielen ähnlichen Situationen bei geringem Zeitverlust bereits mit P Prozessoren auskommt. Betrachten wir als Beispiel die obere Hälfte von W_8 (Bild 5). Identifiziert man für alle i die Prozessoren auf dem senkrechten Weg vom Eingang i zum Ausgang i, so stellt man fest, daß zunächst die Prozessorenpaare

(1,5) (2,6) (3,7) (4,8)

kommunizieren; dann

(1,3) (2,4) (5,7) (6,8)

und schließlich

(1,2) (3,4) (5,6) (7,8) .

Danach finden die gleichen Kommunikationen in umgekehrter Reihenfolge statt. Ordnet man die Prozessoren wie in Bild 6 auf einen Würfel an, so finden die ersten Kommunikationen entlang den Verbindungen in z-Richtung statt, die zweiten in x-Richtung und die dritten in y-Richtung.

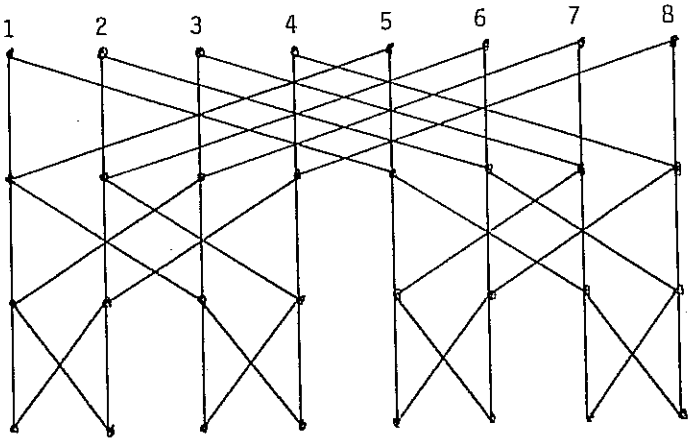


Bild 5

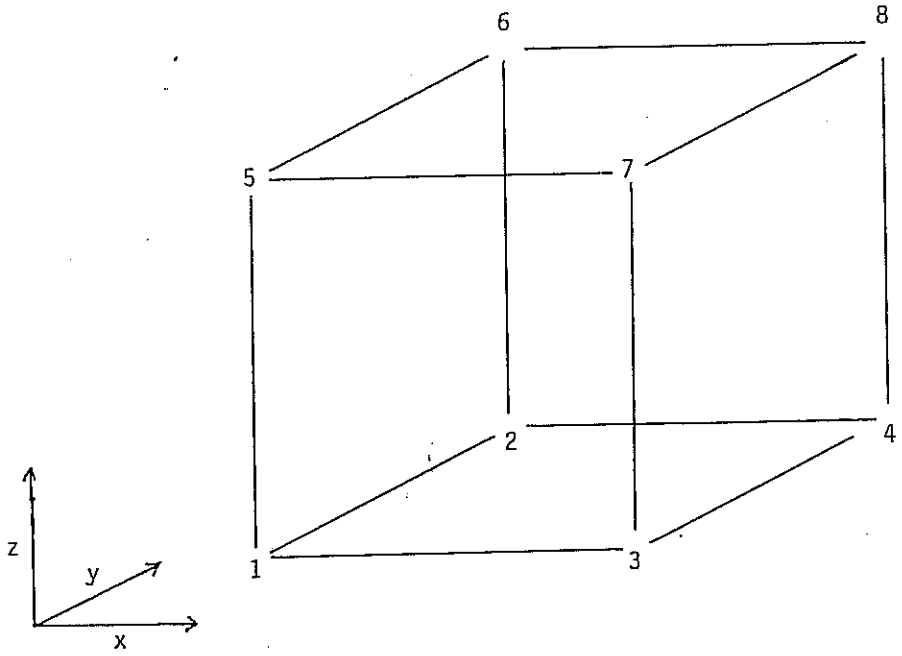


Bild 6

Allgemein können P auf einen $\log P$ dimensionalen Würfel angeordnete Prozessoren das Netz W_p ohne Zeitverlust simulieren.

Unglücklicherweise hat das resultierende Netz Grad $\log P$, was für große Prozessorzahlen P zu groß wird. Dies kann behoben werden, indem man jede Würfelecke durch $\log P$ Knoten ersetzt, die auf einem Kreis angeordnet sind (Bild 7).

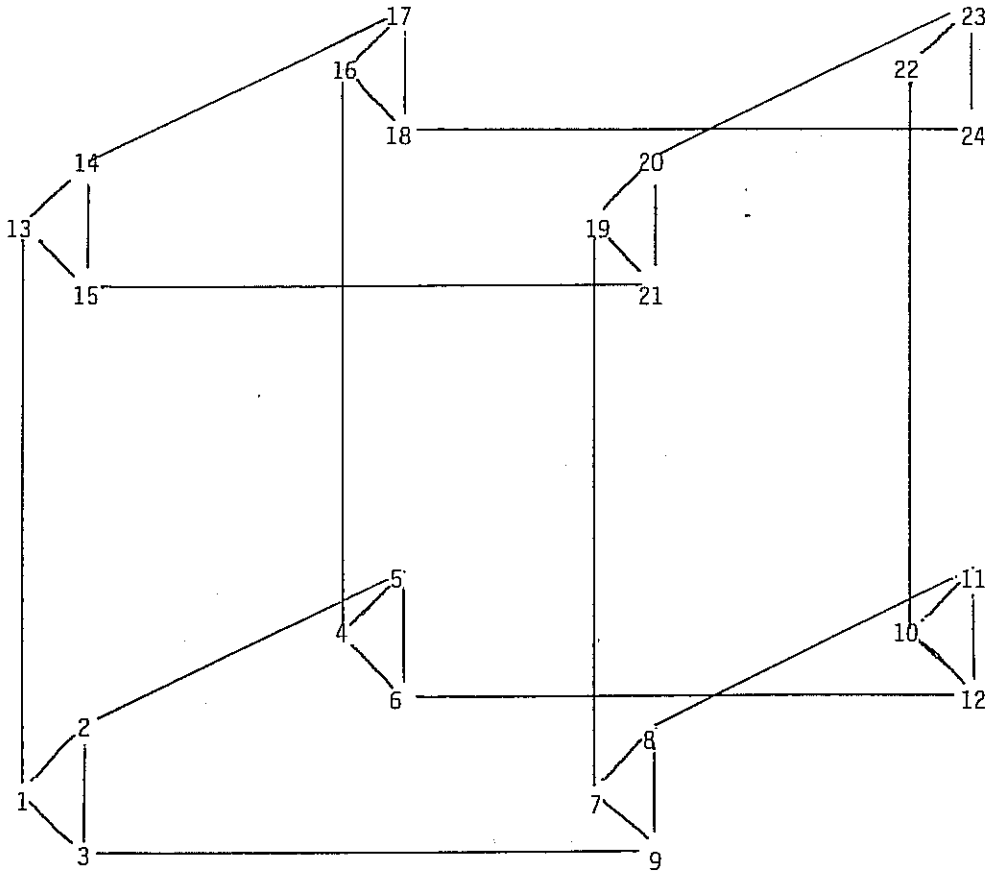


Bild 7

Man erhält so die cube-connected cycles (CCC) aus [PV 79]. Der Graph, den man so erhält, hat wieder $P \log P$ Prozessoren, aber Preparata und Vuillemin beobachten, daß die P-dimensionalen CCC bereits $W_{P \log P}$ simulieren können:

Durch 2-maliges Rotieren der Daten auf den Zyklen können in $4 \log P$ Schritten die Kommunikationen zwischen Prozessoren auf verschiedenen Zyklen hergestellt werden, in weiteren $2 \log P$ Schritten die Kommunikation zwischen Prozessoren auf den gleichen Zyklen. Die obere Hälfte des Permutationsnetzes W_P kann also in $6 \log P$ Schritten von CCC simuliert werden; die gleiche Zeit genügt zur Simulation der unteren Hälfte von W_P. Insgesamt kann damit ein Schritt eines beliebigen Netzes vom Grad d in $12 d \log P$ Schritten simuliert werden. Durch Einfügen von $O(P)$ zusätzlichen Kanten in CCC kann diese Schrittzahl in die Nähe von $2 d \log P$ gebracht werden.

Eine zweite Simulation - ebenfalls mit P Prozessoren, die durch CCC verbunden sind - erlaubt es sogar, ein System von P Prozessoren zu simulieren, die miteinander kom-

munizieren, ohne an feste Verbindungen gebunden zu sein. Dies ist eine Simulation durch Sortieren:

Die Bausteine von Sortiernetzen sind Vergleichsboxen. Sie haben 2 Eingänge und 2 Ausgänge min und max . Legt man an die Eingänge 2 Zahlen a und b an, so erscheint am Ausgang min die kleinere und am Ausgang max die größere der beiden Zahlen.

Ein P-Sortiernetz ist ein Netz von Vergleichsboxen mit P Eingängen und P Ausgängen, so daß gilt: legt man an die Eingänge P beliebige Zahlen an, so erscheinen diese Zahlen an den Ausgängen aufsteigend sortiert.

Batcher [B 68] konstruierte P Sortiernetze mit $O(P(\log P)^2)$ Boxen und Tiefe $O((\log P)^2)$. Dabei werden 2 Typen von Netzen, von denen jedes dem Waksman-Permutationsnetz ähnlich ist, rekursiv geschachtelt. Analog zu der früheren Argumentation kann gezeigt werden, daß CCC mit P Prozessoren Batchers P-Sortiernetz in $O((\log P)^2)$ Schritten simulieren können.

Wir zeigen, wie man durch 2-maliges Sortieren einen Schritt von P Prozessoren $1,\ldots,P$ simuliert, die wie folgt kommunizieren:

In jedem Schritt t kommuniziert Prozessor i mit genau einem Prozessor $q(i,t)$. Kommuniziert Prozessor i mit keinem anderen Prozessor, so sei $q(i,t) = i$. Die Prozessoren i und $q(i,t)$ kommunizieren, indem sie die Inhalte von speziellen Kommunikationsregistern C_i und $C_{q(i,t)}$ austauschen (in [GP 80] wird gezeigt, wie man allgemeinere Situationen handhabt). Wir betrachten im folgenden einen festen Schritt t und lassen den Index t weg. Die Simulation wird durch ein Beispiel mit $P = 6$ illustriert.

i =	1	2	3	4	5	6
q(i) =	5	3	2	6	1	4

Zunächst gibt jeder Prozessor i das Paar [q(i),i] ins Sortierwerk. Man kann sich dieses Paar als eine Postkarte mit Adresse q(i) und Absender i vorstellen. Der Text der Postkarten ist stets der gleiche:

"Lieber Adressat! Ich, der Absender, wüßte gerne den Inhalt Deines Kommunikationsregisters"

und braucht deshalb nicht hingeschrieben zu werden. Im Beispiel werden die Paare

[5,1] [3,2] [2,3] [6,4] [1,5] [4,6]

ins Sortierwerk gegeben. Nun wird nach der 1. Komponente sortiert, und die Adressaten erhalten die Postkarten

[1,5] [2,3] [3,2] [4,6] [5,1] [6,4] .

Die Adressaten überschreiben die Adresse mit dem Inhalt ihres Kommunikationsregisters

$[C_1,5]$ $[C_2,3]$ $[C_3,2]$ $[C_4,6]$ $[C_5,1]$ $[C_6,4]$

und schicken die Karten zurück an die Absender (sortieren nach der 2. Komponente)

$[C_5,1]$ $[C_3,2]$ $[C_2,3]$ $[C_6,4]$ $[C_1,5]$ $[C_4,6]$.

Damit kann auf CCC mit P Prozessoren ein Schritt einer Menge von P frei kommunizierenden Prozessoren in $O((\log P)^2)$ Schritten simuliert werden.

Ähnliche Ergebnisse gelten natürlich auch für jedes andere Netz von P Prozessoren, auf dem man in $O(\log P)$ Schritten Permutationen realisieren und in $O((\log P)^2)$ Schritten sortieren kann, z.B. das Shuffle-Exchange-Netz [K 73] mit P Prozessoren $1,\ldots,P$ mit den folgenden Verbindungen

$(i, 2i-1)$ für $i \leq P/2$
$(i, 2(i-P/2))$ für $i \geq P$
$(i, i+1)$ für $i < P$
$(P,1)$

Bild 8 zeigt das Shuffle-Exchange-Netz mit 8 Prozessoren.

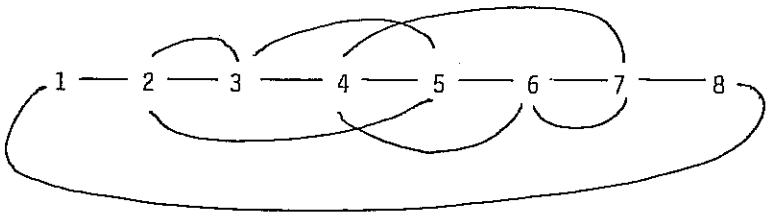


Bild 8

Die obigen Oberlegungen lassen es denkbar erscheinen, daß der Benutzer eines künftigen general purpose parallel computers U neben Daten und Programm auch noch sein eigenes Netz G spezifiziert, das dann von U simuliert wird. Dazu braucht man Sprachen, um sehr große Netze zu beschreiben und für die Simulation durch Permutationsnetze auch "Compilationstechniken", die einem aus der Beschreibung eines Netzes G die Dekomposition in Netze G_i vom Grad 1 und deren Einbettung auf das Permutationsnetz liefert. Die Dekomposition eines Netzes vom Grad d in $\binom{d}{2}$ Netze vom Grad 1 ist einfach. Die besten bekannten Algorithmen zur Dekomposition in d Netze suchen perfekte matchings in gewissen bipartiten Graphen, wofür auch auf parallelen Rechnern keine besonders schnellen Verfahren bekannt sind. Andererseits sieht man für viele Netze wie binäre Bäume und Gitter die Dekomposition mit bloßem Auge. Für kompliziertere aber häufig benutzte Netze kann man sie ein für allemal ausrechnen und abspeichern.

Die Einbettung eines Netzes vom Grad 1 auf das Permutationsnetz W_P erfordert im wesentlichen $\log P$ mal das Finden von Zusammenhangskomponenten von Graphen, die aus

disjunkten Kreisen bestehen und P Knoten haben. Diese Zusammenhangskomponenten kann man z.B. durch O(log P)-maliges Sortieren finden, so daß die Einbettung mit Hilfe von CCC insgesamt in $O((\log P)^4)$ Schritten gefunden werden kann [LPV, GP 80].

Literatur.

[AHU 74] Aho, Hopcraft and Ullman: the design and analysis of computer algorithms, Addison-Wesley 1974

[B 68] Batcher: sorting networks and their applications, proc. AFIPS spring joint computer conference,vol. 32, 307-314, 1968

[GP 80] Galil and Paul: a theory of complexity of parallel computation, preprint Tel Aviv University und Universität Bielefeld, 1980

[K 73] Knuth: sorting and searching, Addison-Wesley 1973

[PR 78] Paul and Reischuk: a graph theoretic approach to determinism versas non-determinism, erscheint in ACTA INFORMATICA

[PR 79] Paul and Reischuk: on time versas space II, proc. 20th IEEE-FOCS, 298-306, 1979

[PV 79] Preparata and Vuillemin: the cube-connected-cycles: a versatile network for parallel computation, proc. 20th IEEE-FOCS, 140-147, 1979

[W 68] Waksman: a permutation network, J. ACM 15, 159-163, 1968

[LPV] Lev, Pippenger and Valiant: a fast parallel algorithm for routing in permutation networks, erscheint in IEEE-TC.