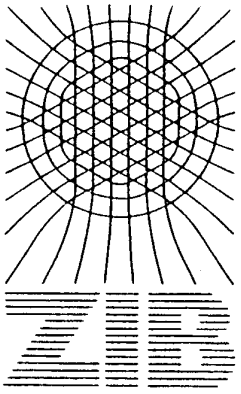
Konrad-Zuse-Zentrum für Informationstechnik Berlin



Gerhard Maierhöfer Georg Skorobohatyj

Implementierung des parallelen TRAPEX auf Transputern

Technical Report TR 89–8 (November 1989)

Inhalt

1 Verwendete Hardware 1

2 Vertikale Parallelisierung 4

3 Meßergebnisse 10

3.1 Vertikale Parallelisierung 10

3.2 Horizontale Parallelisierung 15

Abstract

Der folgende Bericht ist eine Ergänzung des ZIB Technical Report TR 88–5. Entsprechend wird hier nicht auf die grundsätzlichen Fragen der Parallelisierbarkeit des sequentiellen TRAPEX eingegangen. Diese sind im TR 88–5 erörtert, die dort beschriebenen Algorithmen (vertikale und horizontale Parallelisierung) werden auch für die Transputerarchitektur verwendet. Meßergebnisse sind im letzten Teil angefügt.

1. Verwendete Hardware

Im ZIB stehen derzeit zwei VMTM–Boards (Firma Parsytec; im folgenden als Boards bezeichnet) mit je 4 Transputern T800 in einer SUN3–Umgebung zur Verfügung. Ausgehend von einer damit möglichen Topologie des Netzwerkes, ist als erstes die Verteilung der einzelnen Tasks auf die Prozessoren festzulegen. Analog zu unserer Vorgehensweise bei der Entwicklung des parallelen Algorithmus aus dem sequentiellen (TRAPEX–) Algorithmus für SUPRENUM–Rechner und der Simulation mit Hilfe des SUPRENUM–Simulator SUSI, wurde sowohl eine "vertikal" als auch eine "horizontal" parallele Version implementiert (siehe Bericht "Parallel–TRAPEX", Technical Report TR 88–5 des ZIB). Die Tasks sind jeweils dieselben wie bei der SUSI–Version: bei der vertikalen Parallelisierung ein CALLER, ein TRAPEXMASTER und mehrere TRAPEX–Tasks, bei der horizontalen Parallelisierung ein CALLER, eine TRAPEX–Task und mehrere Tasks zur Berechnung von Funktionswerten. Beiden parallelen Versionen liegt eine modifizierte "Farming"–Methode zugrunde. Beim reinen Farming erhält jeder "Slave" Arbeit zugewiesen und gibt, nachdem er diese insgesamt erledigt hat, die errechneten Ergebnisse an den MASTER wieder ab. Bei der hier verwendeten Methode entscheidet ein Slave eigenständig, ob er die ganze zu leistende Arbeit selbst erledigt, oder ob ein Teil der "Last" (die zu bewältigende Rechenarbeit) wieder an den MASTER zurückgegeben wird. Jeder Slave bearbeitet adaptiv, d.h. abhängig von dem durch ihn selbst ermittelten Arbeitsaufwand, seine ihm zugewiesene Aufgabe.

Wir verwenden das von der Firma Parsytec modifizierte Transputer–Entwicklungs–System TDS (Transputer Development System), das auf (dem Host–Rechner) der SUN3/260 gestartet wird und auf einen der 4 Transputer des Boards, dem sogenannten Host–Transputer, läuft. Da das TDS nur für den Host–Transputer eine Terminal–Schnittstelle bereitstellt, muß der CALLER auf diesem Host–Transputer laufen. Welcher der vier Transputer als Host verwendet werden soll, kann beim Aufruf des TDS als Parameter angegeben werden; den Transputer–Sektionen A, B, C und D einer Karte sind dazu die Nummern 0, 1, 2 und 3 zugeordnet. Bei Aufruf des TDS wird außerdem eine Standard–Konfiguration für das Prozessor–Netzwerk eingestellt, d.h. eine Hardware–Verbindung der Prozessoren untereinander und mit dem Host–Rechner über die Transputer–Links. Die Standard–Konfiguration ist im Bild 1 angegeben:

Ein auf mehrere Transputer zu verteilendes Programm–System wird als ganzes nach einem sogenannten Down–Loading vom Host–Transputer aus gestartet; dadurch erhält jeder Prozessor (ausgenommen den Host) den für ihn bestimmten Code, der nach dem Laden selbständig aktiviert wird. Der Code für den Host–Transputer wird anschließend getrennt geladen und gestartet. Inner-

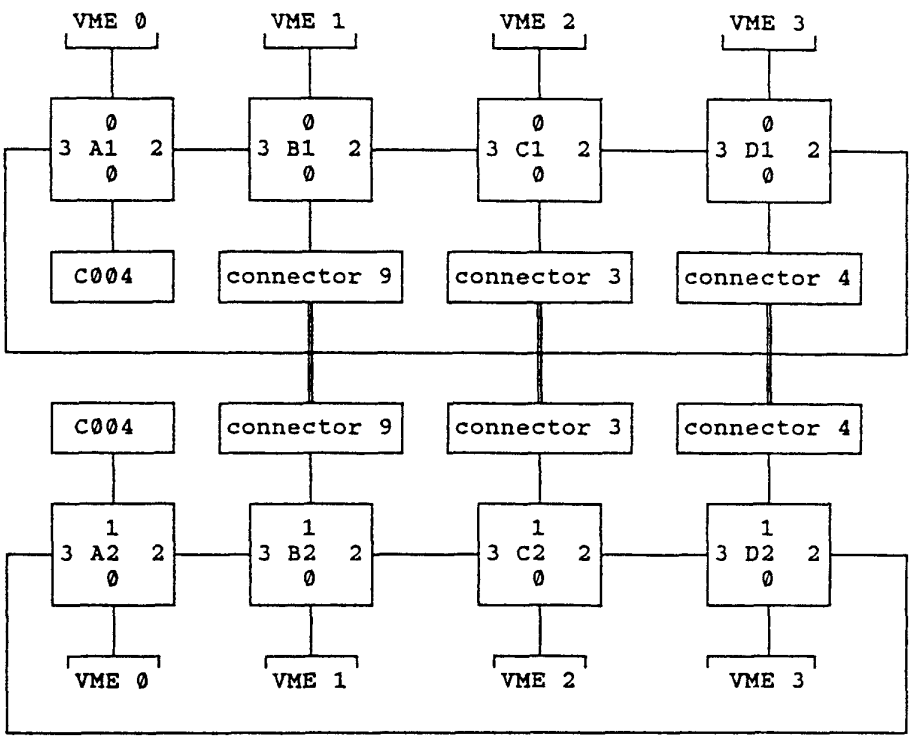


Bild 1 Bei der Standard–Konfiguration ist das Link 0 eines jeden Prozessors mit dem VME–Bus verbunden; dadurch ist es möglich, daß mehrere TDS–Benutzerprozesse parallel laufen mit jeweils eigenen Host–Transputern.

halb des TDS ist ein dynamisches Down–Loading nicht möglich, d.h. zur Ausführungszeit eines parallelen Programms können keine weiteren Prozessoren auf die beschriebene Weise aktiviert werden. Aus diesem Grunde wurde in der Transputer Version des "vertikal" parallellen TRAPEX nur der von uns so genannte Festgrenzenmodus implementiert (siehe oben genannten Bericht).

Im Bild 1 ist auch der auf jedem Board einmal vorhandene Link–Switch–Baustein C004 dargestellt, der dem Crossbar–Switching der Links dient. Jeder C004–Baustein hat eine feste Verbindung mit Link 1 der Transputer–Sektion A des betreffenden Boards, über die er die zur Konfigurierung nötigen Kontrollinformationen erhält. Da ein Benutzerprogramm auf Link 1 des Transputers A wie auf jedes andere Link Zugriff hat, ist eine Rekonfigurierung des Netzwerkes oder eine dynamische Konfigurierung möglich, wenn das TDS mit Transputer 0 als Host gestartet wird. Von dieser Möglichkeit wurde auch Gebrauch gemacht, weil die Standard–Konfiguration den Nachteil hat, daß der TRAPEXMASTER nicht alle vier Links zur Kommunikation mit anderen Prozessoren verwenden kann, wenn er auf einem Host–Transputer läuft. Das Bild 2 zeigt die für die parallelen TRAPEX–Versionen zugrunde gelegte Konfiguration, die nach Aufruf

des TDS einmal mittels der (von uns formulierten) Programme CFGBRD2TP8 und TP8CONFIG einzustellen ist:

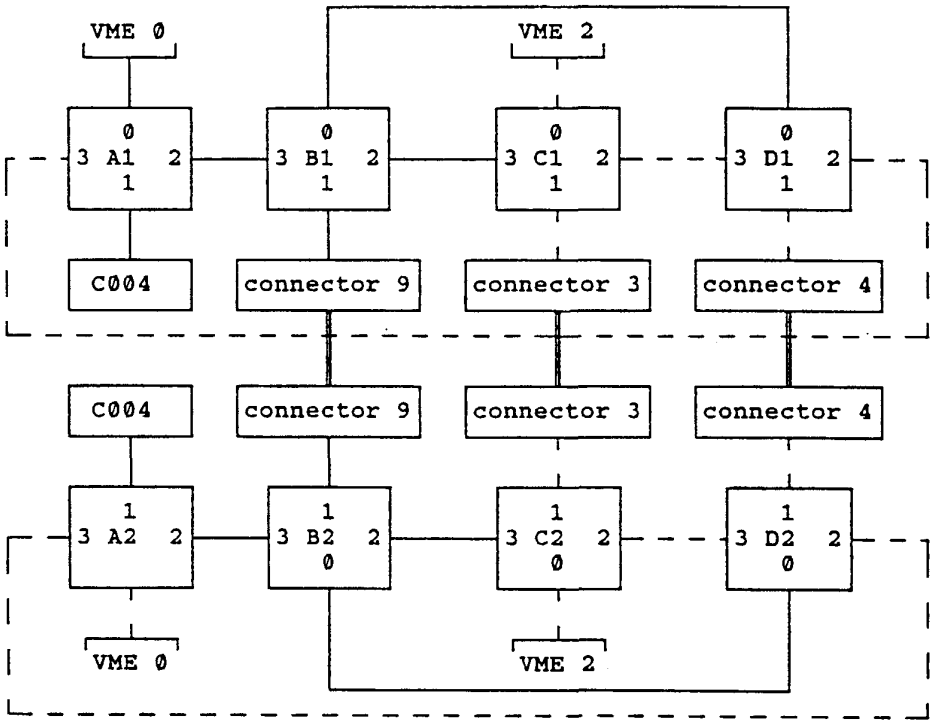


Bild 2 Die *gestrichelten* Linien kennzeichnen von uns *nicht verwendete* Verbindungen.

2. Vertikale Parallelisierung

Der TRAPEXMASTER läuft auf Transputer B1 (erstes Board); er wird vom CALLER auf dem Host–Transputer mit den Start–Daten versorgt und liefert an diesen zum Schluß die Ergebnisse. Auf den übrigen Transputern und auf dem Host–Transputer läuft jeweils eine TRAPEX–Task, weil der CALLER nach der initialen Datenübertragung an den TRAPEXMASTER bis zur Entgegennahme der Ergebnisse inaktiv ist. Bemerkt sei hier noch, daß die in einem Anwenderprogramm zu vergebenden Prozessor–Nummern beliebig sind, insbesondere also nicht mit denen des TDS übereinzustimmen brauchen. Für die Verwaltung der TRAPEX–Tasks innerhalb des TRAPEXMASTER wurden die Nummern 1 bis 7 gewählt und 0 für eine weitere optionale Task, die zusätzlich zum TRAPEXMASTER auf demselben Prozessor laufen kann. Bei der neuen Konfiguration übernimmt der Prozessor B2 (zweites Board) eine Verteilungsfunktion bezüglich der Daten, die zwischen dem MASTER und den TRAPEX–Tasks auf dem zweiten Board ausgetauscht werden. Solche Verteilungsprozesse sind stets zur Kommunikation zweier nicht direkt durch Links verbundenen Prozessoren nötig und müssen vom Anwender selbst programmiert werden, und zwar als Prozesse, die zum Anwendungsprozeß auf dem jeweiligen Vermittlungsprozessor parallel laufen. In der Programmiersprache OCCAM ist dazu nur ein PAR–Konstrukt erforderlich, durch das die Parallelität der zugehörigen Komponenten–Prozesse spezifiziert wird; auf der T800– bzw. T414–Transputer Hardware wird die Parallelität mehrerer Prozesse durch eine interne Task–Verwaltung unterstützt, die im Mikro–Code implementiert ist und spezielle Maschineninstruktionen wie START PROCESS und END PROCESS bereit stellt. Die Bearbeitung der einzelnen Tasks geschieht dabei im üblichen Zeitscheiben–Verfahren; Datenübertragungen über die Links sind jedoch unabhängig von der CPU und können somit tatsächlich gleichzeitig zu einer anderen Task erfolgen. Wartezeiten, die durch die prinzipiell synchrone Kommunikation zwischen Prozessen enstehen können, werden stets zur Bearbeitung anderer Prozesse ausgenutzt, sofern welche vorhanden, d.h. durch den Programmmierer vorgesehen sind – in OCCAM im allgemeinen als eine der sequentiellen Komponenten eines PAR–Konstrukts. Für den Entwurf eines Systems paralleler Prozesse unter OCCAM bedeutet dies, daß Kommunikationsteile möglichst als parallele Prozesse formuliert werden sollten. Ausgehend von der parallelen Version TPAR8 für SUSI wurde deshalb als erste folgende Prozeß–Struktur für eine Transputer–Version zugrunde gelegt:

In Bild 3 sind die auf den einzelnen Transputern vorgesehenen Prozesse und die Datenübertragungskanäle zwischen ihnen dargestellt (entsprechend den Strukturelementen der Programmiersprache OCCAM: Prozeß, Kanal und Prozessor). Den Kanälen zwischen Prozessen auf Prozessoren, die direkt verbunden sind,

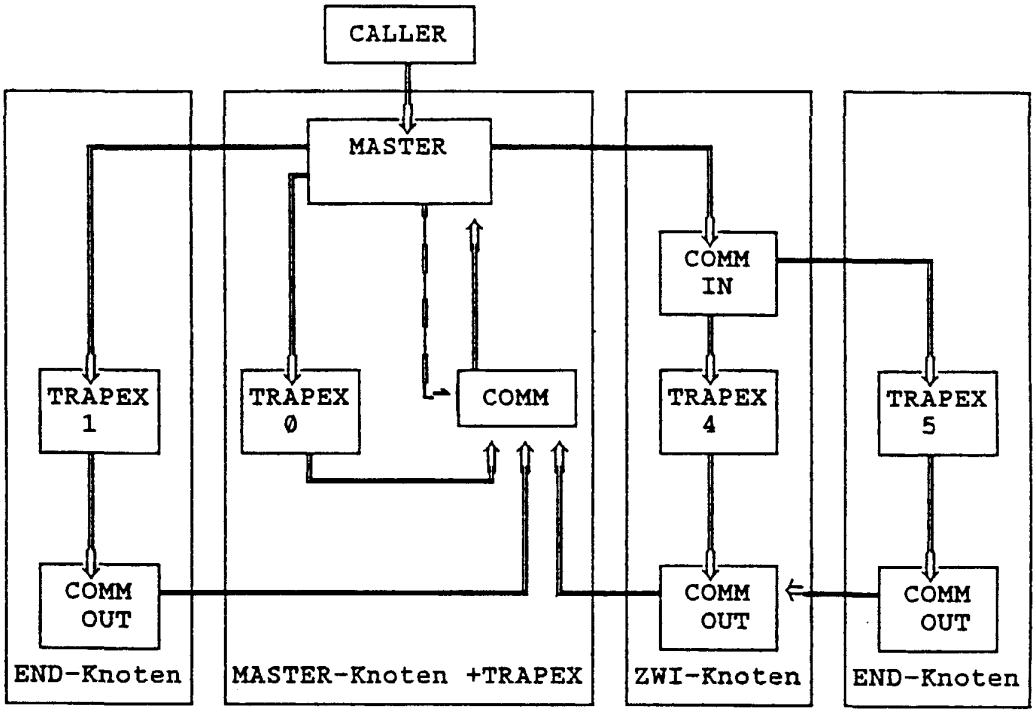


Bild 3

werden entsprechende Links explizit zugeordnet; diese auch als Konfigurierung bezeichnete Zuordnung ist Bestandteil des OCCAM–Anwenderprogrammes und von der Hardware–Verbindung der Links untereinander zu unterscheiden. Innerhalb eines OCCAM–Programmes gibt es für jeden Kanal eine Übertragungsrichtung insofern, als einem Kanal nur je ein Ausgabe– und Eingabeprozeß zugeordnet werden darf, d.h. ein Prozeß darf nicht auf demselben Kanal senden und empfangen, und es dürfen nicht mehrere parallele Prozesse auf demselben Kanal senden oder empfangen. Da die Transputer–Links jedoch bidirektional arbeiten, sind bei der Konfigurierung pro Prozessor und Link je ein Eingabe– und Ausgabekanal anzugeben derart, daß jeder Kanal genau einmal als Eingabe– und Ausgabekanal bei verschiedenen Prozessoren erscheint entsprechend der vorgesehenen Topologie des Prozessor–Netzwerkes. In Bild 3 sind die Kanäle deshalb mit Richtungspfeilen versehen, so daß zwei Kanäle verschiedener Richtung zwischen zwei Prozessor–Knoten einer Verbindung zwischen denselben durch Links entsprechen. Der Übersichtlichkeit halber wurden nicht alle End–Prozessor–Knoten dargestellt.

Auf dem MASTER–Prozessor ist optional ein weiterer TRAPEX–Task initiierbar. Probeläufe mit der SUSI–Version ergaben, daß der MASTER–Prozessor im allgemeinen geringer ausgelastet ist als die TRAPEX–Prozessoren. Auf einem TRAPEX–Prozessor ist zusätzlich ein Kommunikationsprozeß für die Abgabe von Teilintervallen vorgesehen, damit dieses parallel zur Bearbeitung des verbleibenden Teilintervalles geschieht. Dieser Prozeß muß dann auch die Ergebnisse an den MASTER senden, da, wie schon erwähnt, nur ein Prozeß auf dem betreffenden Kanal senden darf. Ein Prozeß darf jedoch auf mehreren Kanälen (parallel) senden oder empfangen, so im Beispiel der TRAPEXMASTER, der auf allen Ausgabekanälen sendet zur Übergabe der initialen Teilintervalle an die TRAPEX–Tasks. Daten, die die TRAPEX–Tasks an den MASTER senden, werden von diesem nicht direkt übernommen, sondern von einem weiteren Kommunikationsprozeß COMM auf dem MASTER–Prozessor, damit auch die Eingabe über die Links parallel zu anderen Aktivitäten des MASTER oder der zusätzlichen TRAPEX–Task abaufen kann. Das Empfangen von Daten über die Links geschieht jedoch nicht wirklich parallel, sondern in der Weise, daß auf allen Links auf Daten gewartet und jeweils der Prozeß aktiviert wird, der zur Entgegennahme der Daten desjenigen Link bestimmt ist, auf dem zuerst Daten eintreffen; dazu gibt es in OCCAM eine spezielle Kontroll–Struktur namens ALT. Treffen auf mehreren Kanälen gleichzeitig Daten ein, wird – implentierungsabhängig – ein Kanal zum Empfang ausgewählt; die Daten auf den übrigen Kanälen gehen jedoch nicht verloren, sondern können mittels weiterer ALT–Input–Aufrufe empfangen werden. Der Kommunikatiosprozeß auf dem MASTER–Prozessor hat daher folgende Struktur:

```
busy := TRUE
WHILE busy
     ALT
       in0 ? inbuf0
           process.input ()
       in1 ? inbuf1
           process.input ()
       in2 ? inbuf2
           process.input ()
       in3 ? inbuf3
           process.input ()
       in4 ? inbuf4
           process.input ()
       inMASTER ? inbufm
           process.MASTER.input ()
```

Dabei sind IN0 bis IN4 die Eingabekanäle, über die die TRAPEX–Tasks Daten senden und INMASTER ein Kanal, über den der TRAPEXMASTER seine

Bereitschaft zur Entgegennahme von Daten signalisiert sowie ein Terminationssignal an den Kommunikationsprozeß absetzt. Diese Kanäle sind ebenso wie Variable im OCCAM–Programm zu deklarieren, bevor sie verwendet werden können. Den Kanälen IN0 und INMASTER wird kein Link zugeordnet, da diese zur Kommunikation zwischen Tasks auf demselben Prozessor dienen; solche Kanäle heißen auch "Soft–Channnels". Der eigentliche Empfangsprozeß besteht im Füllen eines Eingabepuffers; die ALT–Kontroll–Struktur verlangt für jede Komponente außerdem einen weiteren Prozeß, das ist derjenige, der nach Beendigung der Datenübertragung aktiviert wird. Zu bemerken ist hier, daß ein Prozeß im Sinne von OCCAM als ausführbare Programmeinheit zu verstehen ist, in diesem Sinne sind auch einzelne Anweisungen Prozesse. Im Beispiel wird als Folgeprozeß jeweils eine Prozedur aufgerufen: PROCESS.INPUT leitet die Daten sofort an den TRAPEXMASTER weiter, falls dieser zum betreffenden Zeitpunkt nicht beschäftigt ist, anderenfalls werden die Daten in einen Ringpuffer geschrieben. In PROCESS.MASTER werden im Ringpuffer vorhandene Daten an den MASTER weitergeleitet. Zu Anfang berechnet der MASTER die initiale Intervallaufteilung und sendet den TRAPEX–Tasks Startdaten; es reicht daher aus, im Kommunikationsprozeß eine logische Variable MASTERBUSY vorzusehen, die mit TRUE zu initialisieren, in PROCESS.MASTER auf FALSE und in PROCESS.INPUT nach dem Senden von Daten an den MASTER wieder auf TRUE zu setzen ist. Auf diese Weise wird eine Synchronisation zwischen dem Kommunikationsprozeß und dem TRAPEXMASTER erreicht. Beim SUPRENUM–Rechner und bei SUSI ist unter MIMD–FORTRAN eine solche Synchronisation innerhalb eines Anwendungsprogrammes nicht erforderlich, weil dort Datenübertragungen asynchron ablaufen und im Betriebs– bzw. Laufzeitsystem prozeßspezifische Warteschlangen zur Aufnahme ankommender Nachrichten vorhanden sind. Die für die Transputer–Version des parallelen TRAPEX als erste zugrunde gelegte Prozeß–Struktur soll also die asynchrone Datenübertragung bei der SUSI–Version nachbilden. Bei einer zweiten Transputer–Version wurde darauf verzichtet, indem entbehrliche Kommunikationsprozesse entfernt wurden. Daraus ergibt sich folgende Prozeß–Struktur Bild 4:

Für die meisten Beispiele des Test–Set zeigte sich, daß die Ausführungszeiten bei dieser Version geringer sind als bei der ersten. Genauere Untersuchungen mit dieser Version (ohne eine zusätzliche TRAPEX–Task auf dem MASTER–Knoten), bei denen jeweils für den TRAPEXMASTER und für die TRAPEX–Tasks Datenübertragungs–, Ausführungs– und Wartezeiten getrennt ermittelt wurden, ergaben, daß der TRAPEXMASTER nur zu ca. 5% ausgelastet ist und damit kaum mehr Prozessor–Zeit beanspucht als der Eingabekommunikationsprozeß bei der ersten Version; die Auslastung der TRAPEX–Prozessoren beträgt etwa 80–95%. Dies bedeutet im Vergleich mit SUSI, daß die Art der

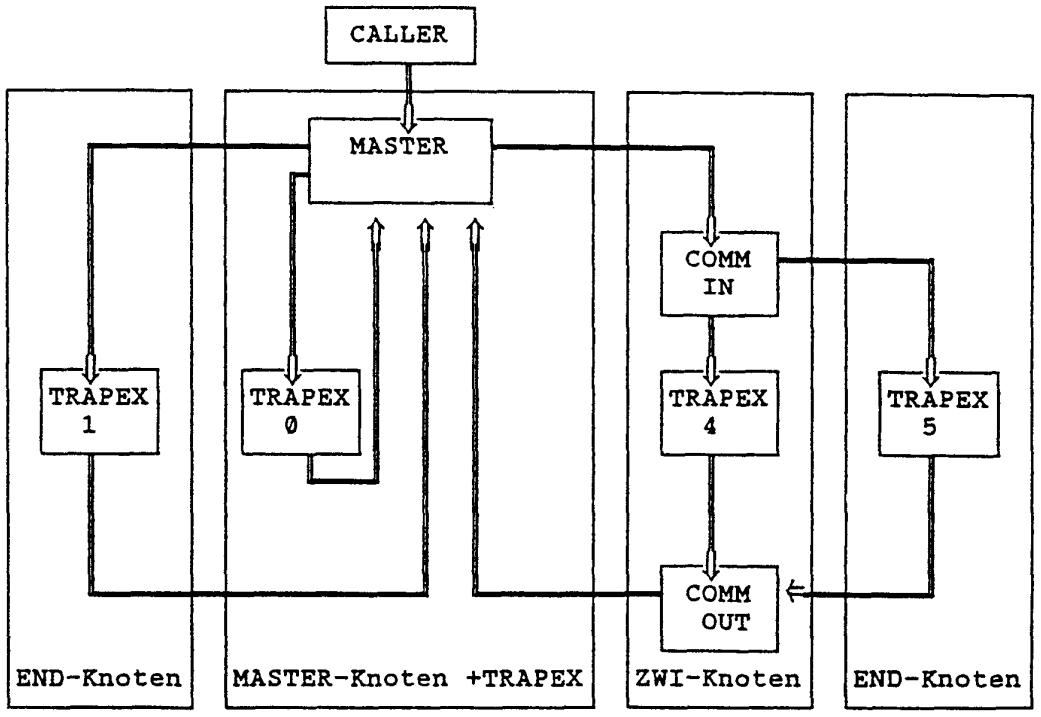


Bild 4

Datenübertragung – synchron oder asynchron – für das parallele TRAPEX keine Rolle spielt. Als nachteilig für die Transputer–Implementierung erweist sich vielmehr die Tatsache, daß ein Prozessor nur mit maximal vier Nachbarn direkt verbunden werden kann, was das Problem einer optimalen Topologie des Prozessornetzwerkes aufwirft. Eine weitere getestete Topologie für das parallele TRAPEX hat disjunkte Wegen zu den Endknoten des zweiten Boards, wie in Bild 5 dargestellt.

Auch für diese Topologie wurden Vergleichsmessungen durchgeführt; dabei zeigten sich aber keine signifikanten Unterschiede gegenüber der ersten. In den folgenden Tabellen sind die Ausführungszeiten für das sequentielle TRAPEX und für die beiden parallelen Versionen jeweils mit und ohne zusätzlicher TRAPEX–Task auf dem MASTER–Prozessor angegeben sowie die erreichten Beschleunigungen unter Verwendung der zwei angegebenen Konfigurationen des zweiten Board. Die Werte weisen eine Ungenauigkeit bis zu zwei Stellen auf, wie sich bei wiederholter Programmausführung zeigt. Die Beispiele sind dieselben wie bei der SUSI–Version (siehe Bericht TR 88–5 des ZIB).

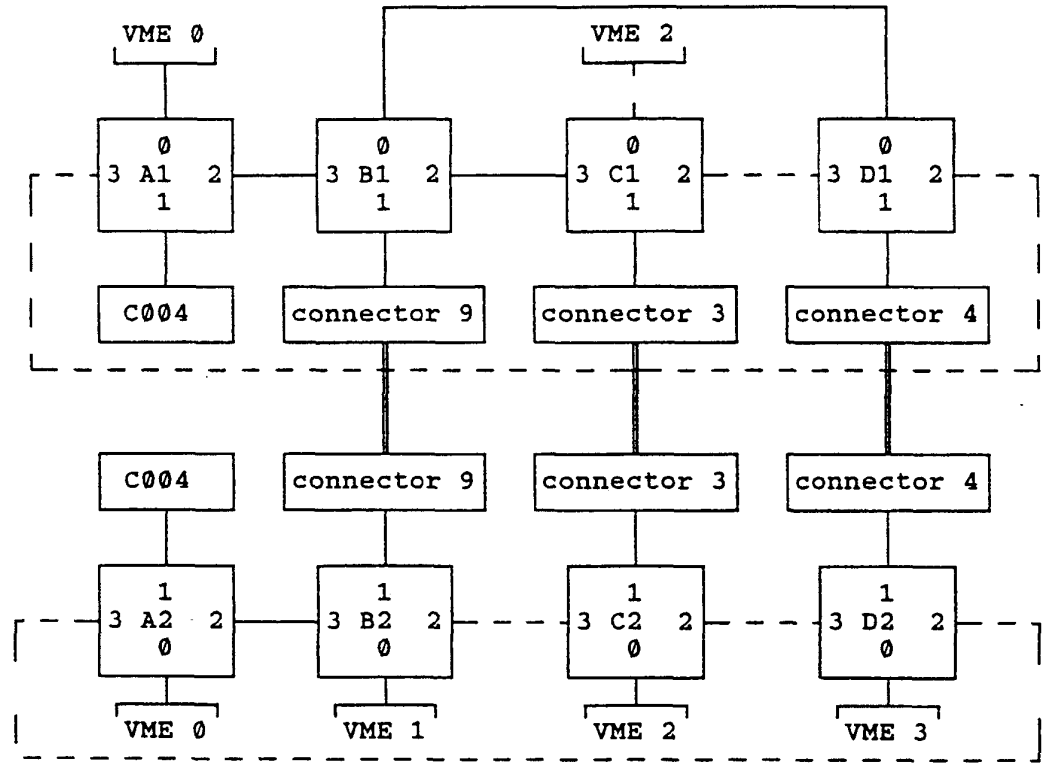

VME 0
VME 2
0
3 A1 2
1
0
3 B1 2
1
0
3 C1 2
1
0
3 D1 2
1
C004
connector 9
connector 3
connector 4
C004
connector 9
connector 3
connector 4
1
3 A2 2
0
1
3 B2 2
0
1
3 C2 2
0
1
3 D2 2
0
VME 0
VME 1
VME 2
VME 3
Bild [5]

Bild 5

3. Meßergebnisse

3.1 Vertikale Parallelisierung

TPARC8: Mit Kommunikationsprozeß und TRAPEXASTER auf Prozessor 0, ein Transfer-Knoten auf zweitem Board

	SEQ	2 * TRAPEX		3 * TRAPEX		4 * TRAPEX	
Bsp	$t[\mu s]$	$t[\mu s]$	ACC	$t[\mu s]$	ACC	$t[\mu s]$	ACC
1	11939	8853	1.35	8792	1.36	8020	1.49
9	353334	143296	2.47	97762	3.61	87221	4.05
13	511875	243305	2.10	155758	3.29	$h = 0$	-
17	472154	264602	1.78	194822	2.55	166770	2.83
20	49766	22274	2.23	19438	2.56	14158	3.52
21	684812	241604	2.83	172252	3.98	147588	4.64
22	238592	90821	2.63	63496	3.76	44912	5.31

	5 * TRAPEX		6 * TRAPEX		7 * TRAPEX	
Bsp	$t[\mu s]$	ACC	$t[\mu s]$	ACC	$t[\mu s]$	ACC
1	8115	1.47	8080	1.48	8094	1.48
9	77722	4.55	58248	6.07	53119	6.65
13	172897	2.96	170999	2.99	188392	2.72
17	147115	3.21	106406	4.44	115186	4.10
20	10666	4.66	8984	5.54	8342	5.97
21	144003	4.76	89059	7.67	81535	8.40
22	48840	4.89	36621	6.52	35996	6.63

TPARC8: Mit Kommunikationsprozeß, TRAPEXMASTER und TRAPEX auf Prozessor 0, ein Transfer–Knoten auf zweitem Board.

	2 * TRAPEX		3 * TRAPEX		4 * TRAPEX		5 * TRAPEX	
Bsp	$t[\mu s]$	ACC	$t[\mu s]$	ACC	$t[\mu s]$	ACC	$t[\mu s]$	ACC
1	8919	1.34	8980	1.33	8148	1.46	8215	1.45
9	142506	2.48	103329	3.42	76401	4.62	74532	4.74
13	242327	2.11	210341	2.43	$h = 0$	-	132034	3.88
17	283642	1.66	186194	2.54	160916	2.93	166622	2.83
20	27561	1.81	15322	3.25	14278	3.49	10602	4.69
21	236336	2.90	178029	3.85	137910	4.97	150529	4.55
22	88954	2.68	66721	3.58	46865	5.09	40616	5.87

	6 * TRAPEX		7 * TRAPEX		8 * TRAPEX	
Bsp	$t[\mu s]$	ACC	$t[\mu s]$	ACC	$t[\mu s]$	ACC
1	8167	1.46	8196	1.46	8295	1.44
9	53524	6.60	55084	6.41	44228	7.99
13	237163	2.16	192058	2.67	$h = 0$	-
17	119326	3.96	113556	4.16	97718	4.83
20	9034	5.51	9067	5.49	9085	5.48
21	91812	7.46	84494	8.10	78303	8.75

TPAR8: Ohne Komunikationsprozeß, nur TRAPEXASTER auf Prozessor 0, ein Transfer–Knoten auf zweitem Board

	SEQ	2 * TRAPEX		3 * TRAPEX		4 * TRAPEX	
Bsp	$t[\mu s]$	$t[\mu s]$	ACC	$t[\mu s]$	ACC	$t[\mu s]$	ACC
1	11939	8601	1.39	8590	1.39	7822	1.53
9	353334	141659	2.49	96732	3.65	82703	4.27
13	511875	241426	2.12	154378	3.32	$h = 0$	–
17	472154	261463	1.81	197132	2.40	161356	2.93
20	49766	21898	2.27	19183	2.59	13963	3.56
21	684812	239757	2.86	162302	4.22	147363	4.65
22	238592	89094	2.68	64579	3.69	43000	5.55

	5 * TRAPEX		6 * TRAPEX		7 * TRAPEX	
Bsp	$t[\mu s]$	ACC	$t[\mu s]$	ACC	$t[\mu s]$	ACC
1	7851	1.52	7881	1.51	7903	1.51
9	76594	4.61	51991	6.80	53131	6.65
13	171928	2.98	170296	3.01	188024	2.72
17	143556	3.29	108582	4.35	96853	4.88
20	10544	4.72	8730	5.70	8166	6.09
21	137351	4.99	88353	7.75	79575	8.61
22	47653	5.01	33046	7.22	36452	6.55

TPAR8: Ohne Komunikationsprozeß, TRAPEXMASTER und TRAPEX auf Prozessor 0, ein Transfer–Knoten auf zweitem Board.

	2 * TRAPEX		3 * TRAPEX		4 * TRAPEX		5 * TRAPEX	
Bsp	$t[\mu s]$	ACC	$t[\mu s]$	ACC	$t[\mu s]$	ACC	$t[\mu s]$	ACC
1	8727	1.37	8731	1.37	7957	1.50	7970	1.50
9	147010	2.40	102502	3.45	73282	4.82	76448	4.62
13	228155	2.27	175685	2.91	$h = 0$	-	131401	3.90
17	277595	1.70	182802	2.58	159595	2.96	149295	3.16
20	27202	1.83	14920	3.34	14035	3.55	10501	4.74
21	246232	2.78	182808	3.75	131871	5.19	120826	5.67
22	90230	2.64	62093	3.84	46978	5.08	42883	5.56

	6 * TRAPEX		7 * TRAPEX		8 * TRAPEX	
Bsp	$t[\mu s]$	ACC	$t[\mu s]$	ACC	$t[\mu s]$	ACC
1	7991	1.49	8019	1.49	8051	1.48
9	57559	6.25	60860	5.81	41122	8.59
13	235600	2.17	191536	2.67	$h = 0$	-
17	111734	4.23	102194	4.62	92107	5.12
20	8795	5.66	8819	5.64	8839	5.63
21	109489	6.25	78269	8.75	98561	6.95
22	38739	6.16	34607	6.89	30935	7.70

TPARC8: Mit Komunikationsprozeß und TRAPEXMASTER auf Prozessor 0, disjunkte Wege zu den Endknoten des zweiten Boards.

	5 * TRAPEX		6 * TRAPEX		7 * TRAPEX	
Bsp	$t[\mu s]$	ACC	$t[\mu s]$	ACC	$t[\mu s]$	ACC
1	8120	1.47	8148	1.46	8172	1.46
9	77731	4.55	59139	5.97	56522	6.25
13	129720	3.94	152832	2.35	181401	2.82
17	157748	2.99	110040	4.29	101664	4.64
20	10737	4.64	8944	5.56	8423	5.91
21	144045	4.75	82126	8.34	80242	8.53
22	41132	5.80	34513	6.91	36094	6.61

TPARC8: Mit Komunikationsprozeß, TRAPEXMASTER und TRAPEX auf Prozessor 0, disjunkte Wege zu den Endknoten des zweiten Boards.

	6 * TRAPEX		7 * TRAPEX		8 * TRAPEX	
Bsp	$t[\mu s]$	ACC	$t[\mu s]$	ACC	$t[\mu s]$	ACC
1	8222	1.45	8184	1.46	8188	1.46
9	53332	6.63	55539	6.36	42693	8.28
13	133759	3.83	184833	2.77	$h = 0$	–
17	113560	4.16	104738	4.51	106224	4.44
20	8964	5.55	8990	5.54	8983	5.54
21	100983	6.78	86436	7.92	77632	8.82
22	35579	6.71	33863	7.05	33870	7.04

TPAR8: Ohne Komunikationsprozeß, nur TRAPEXMASTER auf Prozessor 0, disjunkte Wege zu den Endknoten des zweiten Boards.

	5 * TRAPEX		6 * TRAPEX		7 * TRAPEX	
Bsp	$t[\mu s]$	ACC	$t[\mu s]$	ACC	$t[\mu s]$	ACC
1	7864	1.52	7888	1.51	7909	1.51
9	76641	4.61	58409	6.05	52249	6.76
13	165993	3.08	152138	3.36	180921	2.83
17	151909	3.11	110242	4.28	98446	4.80
20	10547	4.72	8736	5.70	8164	6.10
21	143045	4.79	85887	7.97	79699	8.59
22	42079	5.67	38183	6.25	40868	5.84

TPAR8: Ohne Komunikationsprozeß, TRAPEXMASTER und TRAPEX auf Prozessor 0, disjunkte Wege zu den Endknoten des zweiten Boards.

	6 * TRAPEX		7 * TRAPEX		8 * TRAPEX	
Bsp	$t[\mu s]$	ACC	$t[\mu s]$	ACC	$t[\mu s]$	ACC
1	7878	1.52	8006	1.49	8013	1.49
9	51114	6.91	57028	6.20	42424	8.33
13	132108	3.87	184268	2.78	$h = 0$	-
17	111388	4.24	98757	4.78	93751	5.04
20	8788	5.66	8808	5.65	8793	5.66
21	104485	6.55	80589	8.50	75670	9.05
22	42143	5.66	38385	6.22	34917	6.83

3.2 Horizontale Parallelisierung

Für die horizontale Parallelisierung wurde dieselbe Konfiguration zu Grunde gelegt wie bei der vertikalen Parallelisierung. Auf der Transputer–Sektion A1 (erstes Board) läuft ein CALLER und eine Task zur Funktionswertberechnung, auf der Transputer Sektion B1 die TRAPEX–Task und auf den direkt durch Links mit Sektion B1 verbundenen Prozessoren jeweils eine Task zur Funktionswertberechnung. Da auf einer Extrapolationsstufe – siehe Beschreibung des TRAPEX–Algorithmus in unserem oben genannten Bericht – höchsten vier Funktionswerte zusätzlich bereitgestellt werden müssen, reichen vier Prozessoren für die Funktionswertberechnungen aus. In einer ersten Variante werden alle für eine Extrapolationsordnung nötigen Funktionsauswertungen parallel vorgenommen und eine davon auf dem TRAPEX–Prozessor selbst, sodaß insgesamt vier Prozessoren verwendet werden. In einer zweiten Variante werden parallel zur Extrapolation bezüglich einer Ordnung die für die nächste Ordnung benötigten Funktionsauswertungen vorgenommen; bei dieser Version werden fünf Prozessoren verwendet. Die Meßergebnisse zeigen im Gegensatz zur SUSI–Version zwar eine gewisse Verbesserung der Ausführungszeiten; diese ist jedoch wesentlich geringer als bei der vertikalen Parallelisierung und bei der ersten Variante schon dadurch begrenzt, daß allein für den parallelisierten Anteil des Algorithmus ein Gewinn um einen Faktor möglich ist, der kleiner als 2.2 ist. In der folgenden Tabelle sind die Meßergebnisse dargestellt.

	SEQ	PAR 1		PAR 2	
Bsp	$t[\mu s]$	$t[\mu s]$	ACC	$t[\mu s]$	ACC
1	11939	9532	1.25	9722	1.23
9	353334	278388	1.27	269952	1.31
13	511875	431914	1.19	375160	1.48
17	472154	406189	1.16	386355	1.22
20	49766	36456	1.37	392343	1.27
21	684812	483056	1.42	414980	1.65
22	238592	194363	1.23	269952	1.31

PAR 1: nur Funktionsauswertungen parallel

PAR 2: Funktionsauswertungen parallel zur Extrapolation