Leibniz-Rechenzentrum

der Bayerischen Akademie der Wissenschaften

Technischer Bericht

Leistungsvergleich zwischen dem Vektorsystem Siemens Fujitsu VPP700 und Intel IA32 Pentium 4 Systemen

Dr. R. Bader (LRZ)

Sept 2004 **LRZ-Bericht 2004-03**

Direktorium:

Prof. Dr. H.-G. Hegering (Vorsitzender)
Prof. Dr. A. Bode
Prof. Dr. Chr. Zenger

Leibniz-Rechenzentrum
Barer Straße 21
D-80333 München

UST-ID-Nr. DE811305931

Telefon: (089) 289-28784
Telefax: (089) 2809460
E-Mail: lrzpost@lrz.de
Internet: http://www.lrz.de

Öffentl. Verkehrsmittel:

U2, U8: Königsplatz
U3, U4, U5, U6: Odeonsplatz
Tram 27: Karolinenplatz

1 Einleitung

Die nachfolgenden Benchmarks untersuchen Leistungscharakteristika der *Fujitsu Siemens Computers* VPP700 und von Einprozessor *Intel* IA32 Systemen (Pentium 4 mit 3,06 GHz) unter Einschluss einiger weiterer dem LRZ zugänglichen aktuellen Prozessorarchitekturen. Sie basieren auf einfachen Kernen und auf dem vorrangig von der Chemie genutzten Programm Gaussian 98. Erwartungsgemäß schneidet die Vektorarchitektur immer da gut ab, wo Bandbreite zum Hauptspeicher ein ausschlaggebendes Leistungsmerkmal ist. Hingegen läuft Gaussian auf Cache-basierten Systemen deutlich besser.

Es wird versucht, eine Anwort auf die Frage zu geben, in wie weit ein moderner Off-the-shelf Prozessor in der Lage ist, mit einem klassischen 32-bit Vektorrechner leistungsmäßig mitzuhalten oder ihn sogar zu übertreffen. Darüber hinaus wird auch der Einfluss des in der Pentium-Architektur implementierten Hyper-Threadings untersucht, mit dem eine Verbesserung der internen Instruktions-Parallelität erreicht werden soll.

2 Messungen

2.1 Vektor-Triaden und andere eindimensionale Schleifen: Rinf

Die Rechenleistung für die Vektor-Triade

```
DO I=1,N
      A(I) = B(I)*C(I) + D(I)
```

war für die Leistungsmessung von größter Relevanz; bei der Bewertung der Angebote zu Los 1 trug sie mit einem Gewicht von insgesamt 80% bei. Abbildung 1 zeigt den Leistungsverlauf als Funktion der Vektorlänge N.

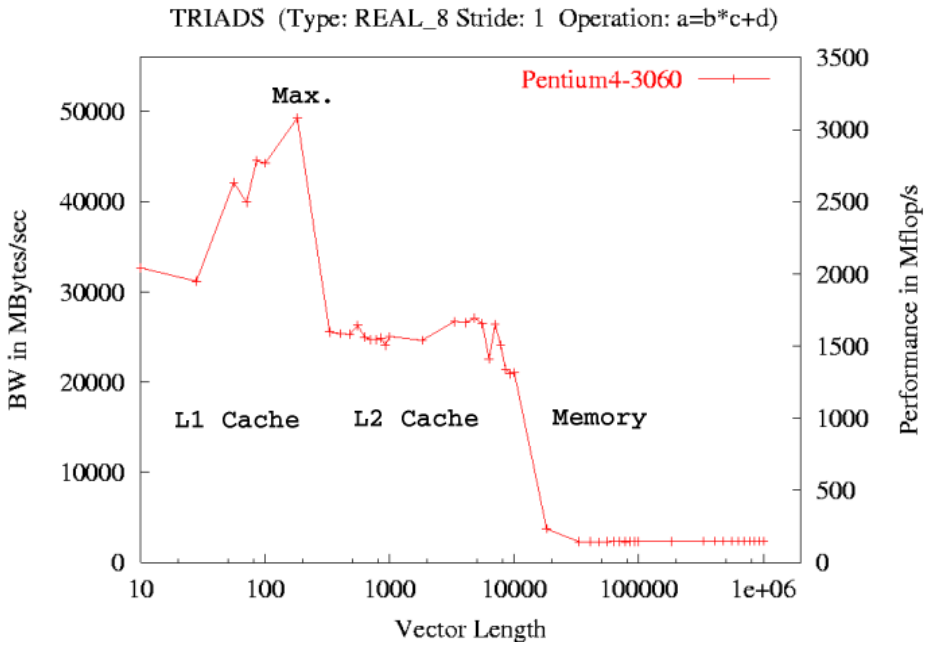

Abbildung 1: Bandbreite und Leistung für Vektor-Triade auf der IA32 Architektur

Das Leistungsmaximum wird bei der Pentium-4 Architektur für die größte noch in den Level-1 Cache passende Vektorlänge N erreicht. Neben diesem Wert war für die Leistungsmessung noch die mit einem speziell optimierten ausführbaren Programm erreichbare Leistung für N=1000000 („Hauptspeicher-Leistung“) relevant. Aufgrund dieser Optimierung liegt letztere um etwa 20% oberhalb der in obiger Abbildung gezeigten „Hauptspeicher“-Leistung.

Es wurden Messreihen mit und ohne Hyperthreading-fähigem Linux-Kernel ausgeführt. Die Streuungen der Maximalperformance nach unten sind bei der Verwendung von Hyperthreading (HT) grösser: die Durchschnittsleistung aus dem Cache nimmt um etwa 2 Prozent ab. Ursache sind wohl eher Kernel-Scheduling-Probleme als die Hardware, sodass für die Abnahme der Fall ohne HT relevant war. Für die entsprechenden Messreihen wurden der beste und schlechteste Wert gestrichen und das Mittel über die vier verbleibenden Werte gebildet.

Messung Nr.	Mit HT		Ohne HT	
	Peak (MFlop/s)	Memory (MFlop/s)	Peak (MFlop/s)	Memory (MFlop/s)
1	2860	188.8	3071	189.4 [2)]
2	2923	189.1	2988 [1)]	188.9
3	3087	189.1	3095	189.1
4	2934	188.9	3097 [2)]	189.0
5	3085	189.1	3038	188.9
6	3084	189.0	3059	188.7 [1)]
2.1.1.1.1Mittelwert	3007	189.0	**3066**	**189.0**
Commitment			3032	171

1) Niedrigster Messwert, gestrichen
2) Höchster Messwert, gestrichen

Zum Vergleich mit den Leistungsdaten anderer am LRZ verfügbarer Architekturen wurden neben der Vektor-Triade auch die Leistungsdaten anderer Schleifentypen betrachtet.

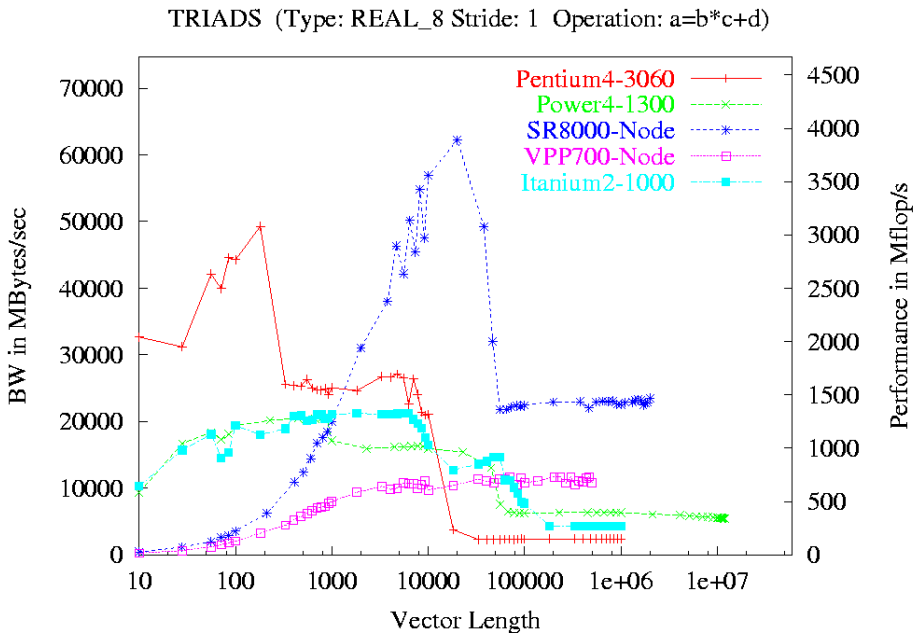

Abbildung 2: Gleitpunkt-Vektortriade für verschiedene Architekturen im Überblick. Zwei Sieger: Hoher Takt IA32 gegen automatische Parallelisierung (Spitzen-Leistung etwa 4 GFlop/s) und Pseudo-Vektorisierung (aus dem Hauptspeicher saturierbare Leistung etwa 1.5 GFlop/s für große Vektor-Längen) des SR8000-Knotens.

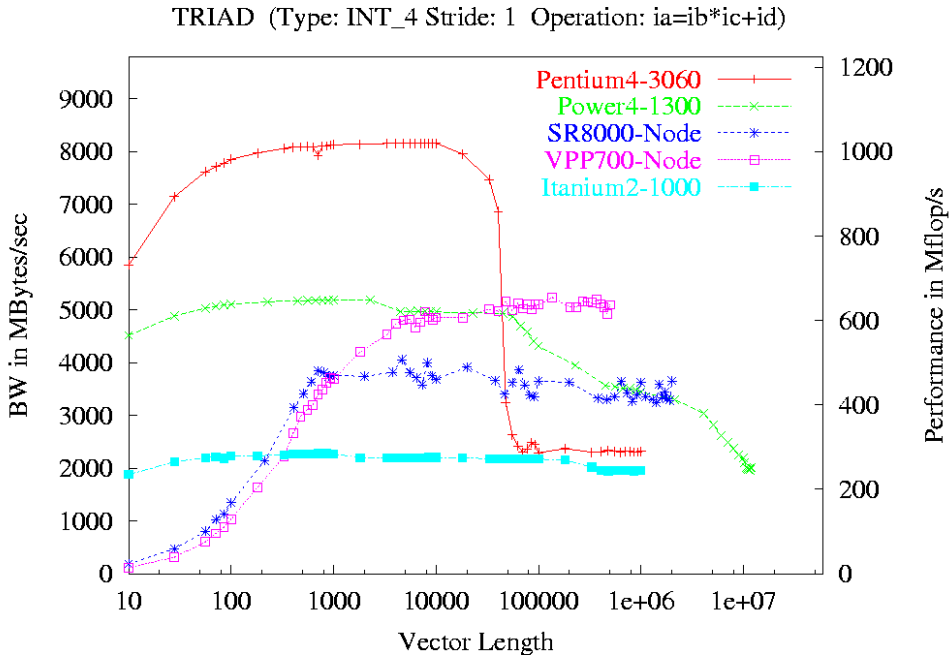

Abbildung 3: Ganzzahl-Triade für verschiedene Architekturen. Hier ist IA32 traditionell stark, IA64 am schwächsten. Trotz ihres Alters hält die VPP700 noch gut mit.

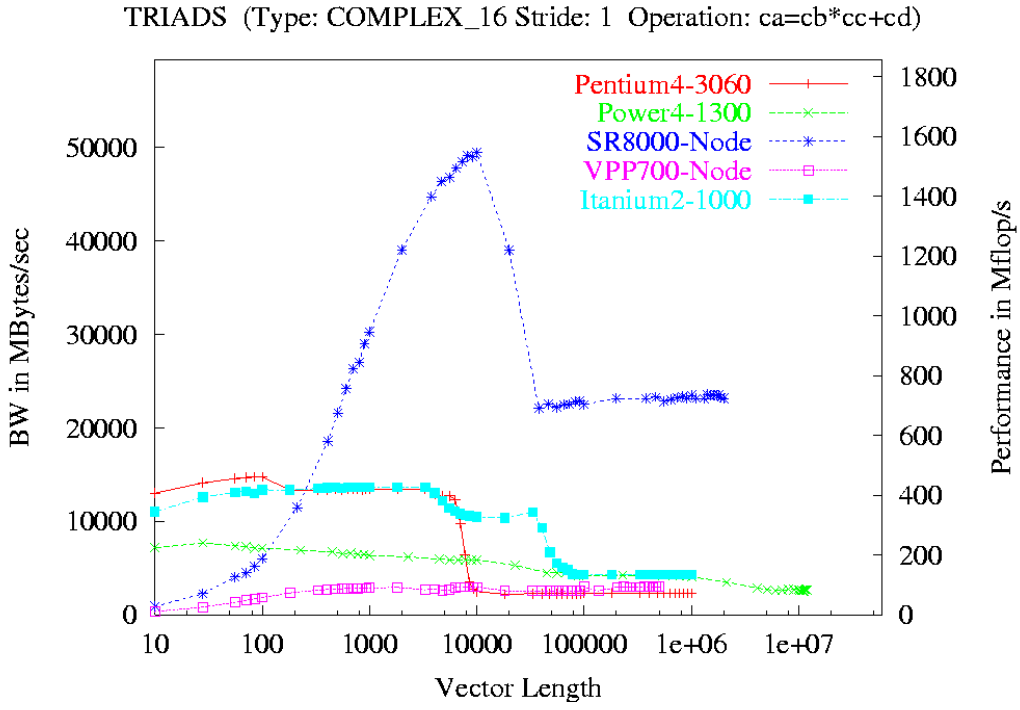

Abbildung 4: Vektortriaden mit komplexen Zahlen. Außer dem SR8000 Knoten brechen fast alle anderen Architekturen deutlich ein. Mit großer Sicherheit ist fehlende Compiler-Optimierung hier ein Problem.

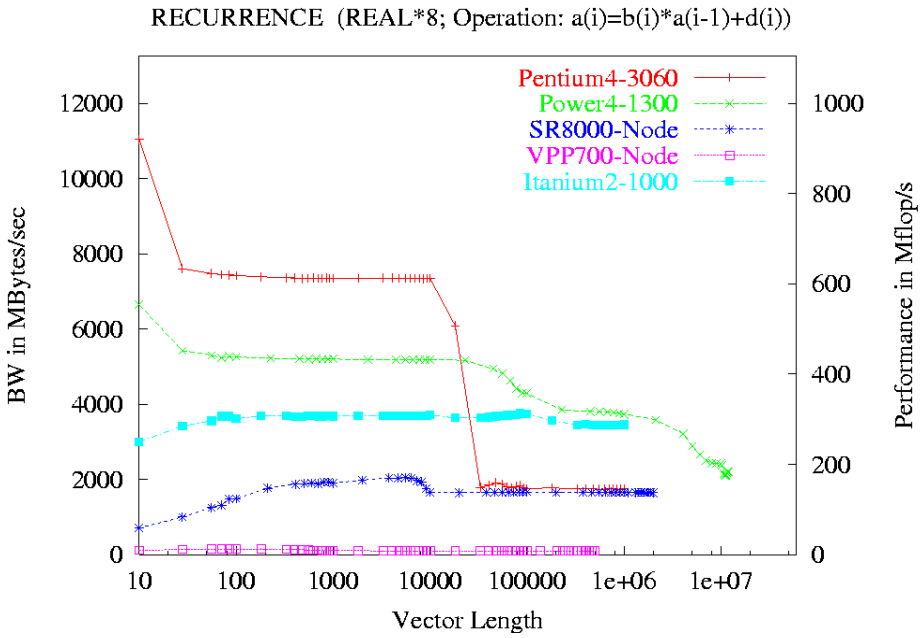

Abbildung 5: Bei der Rekursion haben vektororientierte Architekturen schlechte Karten. IA32 gewinnt aus dem Cache wieder auf Grund der hohen Taktung. Auf der SR8000 kann nur eine von acht 375 MHz-CPUs genutzt werden.

Aus den Caches kann der Pentium 4 auf Grund seiner hohen Taktfrequenz mit allen anderen Architekturen ausgezeichnet konkurrieren, zeigt jedoch im Hauptspeicher recht deutliche Schwächen. Hier muss auch 4 Jahre nach Erscheinungsdatum der Hitachi SR8000 Knoten noch als führend betrachtet werden, solange der Algorithmus die Anwendung von Parallelisierung und Pseudovektorisierung gestattet. Bei der Rekursion ist das nicht der Fall; hier schlägt die IA32 Architektur in der gesamten Speicher-Hierarchie die von Hitachi, da auf derem 8-Wege-Knoten nur eine CPU beschäftigt werden kann, und das nicht mit voller Effizienz bei Zugriff in den Hauptspeicher. Ein Vektorrechner wie die VPP700 bricht in diesem Fall sogar um über eine Größenordnung mit der Performance ein.

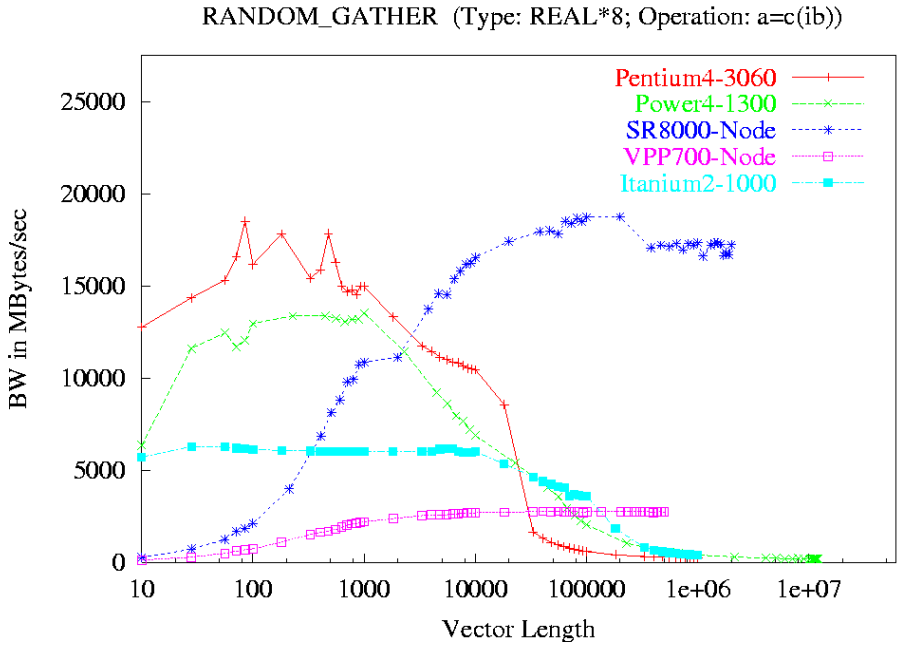

Abbildung 6: Einsammeln von Daten aus dem Speicher. Auf dem SR8000-Knoten sorgt Pre-Loading in die Register für hohe Leistung, im Cache zeigt der Pentium 4 seine Stärken.

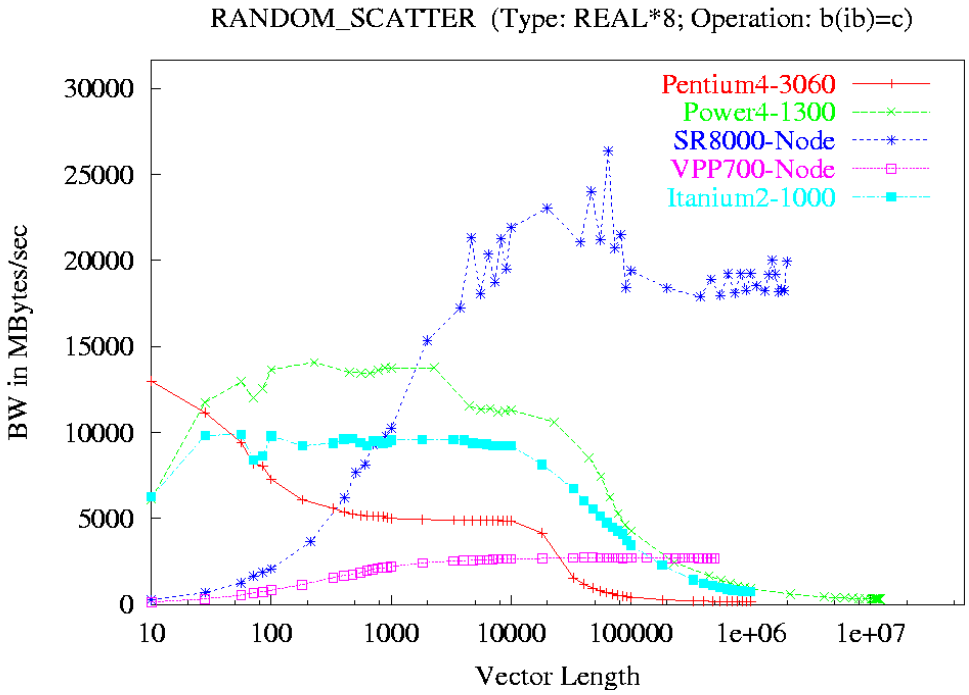

Abbildung 7: Beim Verteilen von Daten in den Speicher tut sich der Pentium deutlich schwerer als beim Einsammeln, während die anderen Architekturen vergleichbare Leistung beibehalten und IA64 sogar noch etwas zulegt.

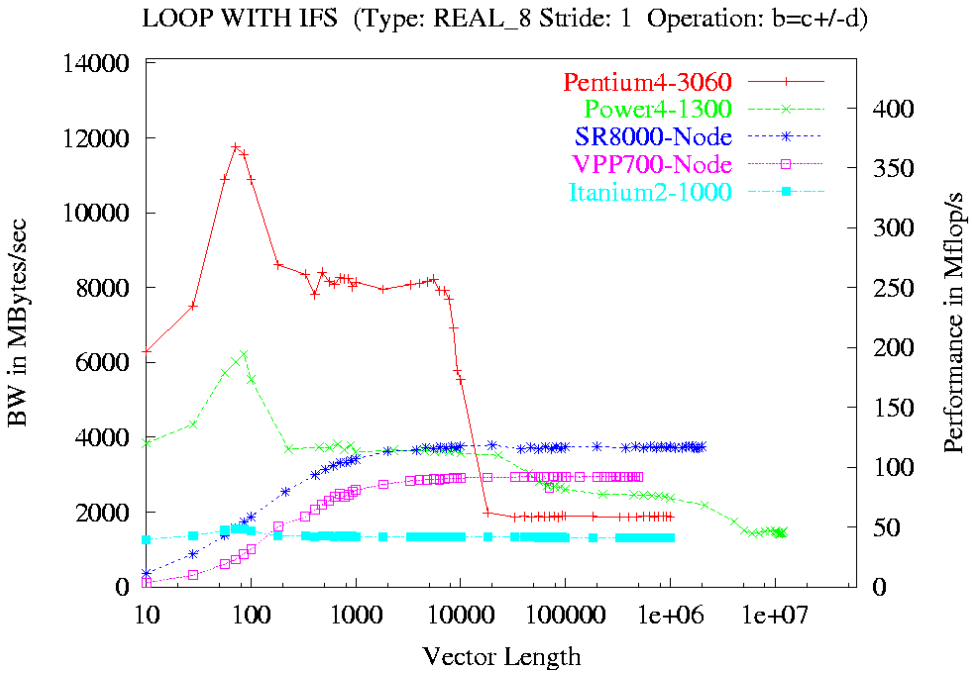

Abbildung 8: Enthält die Schleife eine Bedingung, muss für eine effiziente Vektorisierung meist mehr Rechenarbeit verrichtet werden als eigentlich nötig. Das gilt für IA32 und Power 4 im Cache, für vektorartige Architekturen beim Hauptspeicherzugriff. EPIC bringt hier noch keinen Fortschritt, vorbehaltlich besserer Optimierung durch den Compiler.

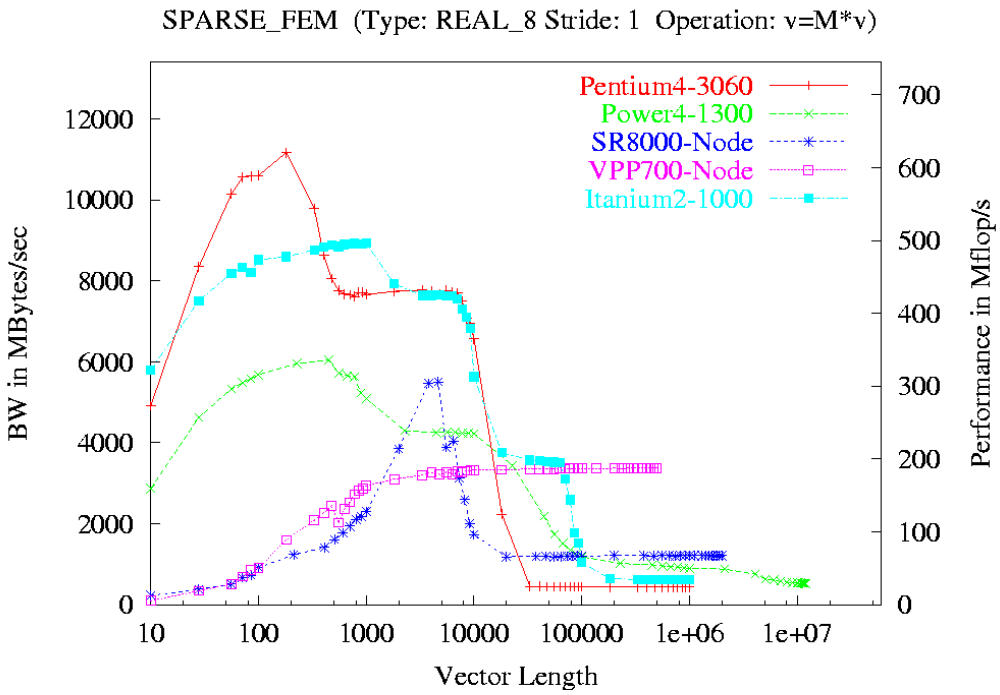

Abbildung 9: In Finite-Elemente Programmen häufig verwendete Multiplikation dünn besetzter Matrizen. Hier schneiden IA32 und IA64 glänzend ab. Erstaunlicherweise fällt der SR8000-Knoten stark zurück, obwohl die Problemstellung, wie aus der Leistung der VPP700 ersichtlich ist, gut vektorisiert.

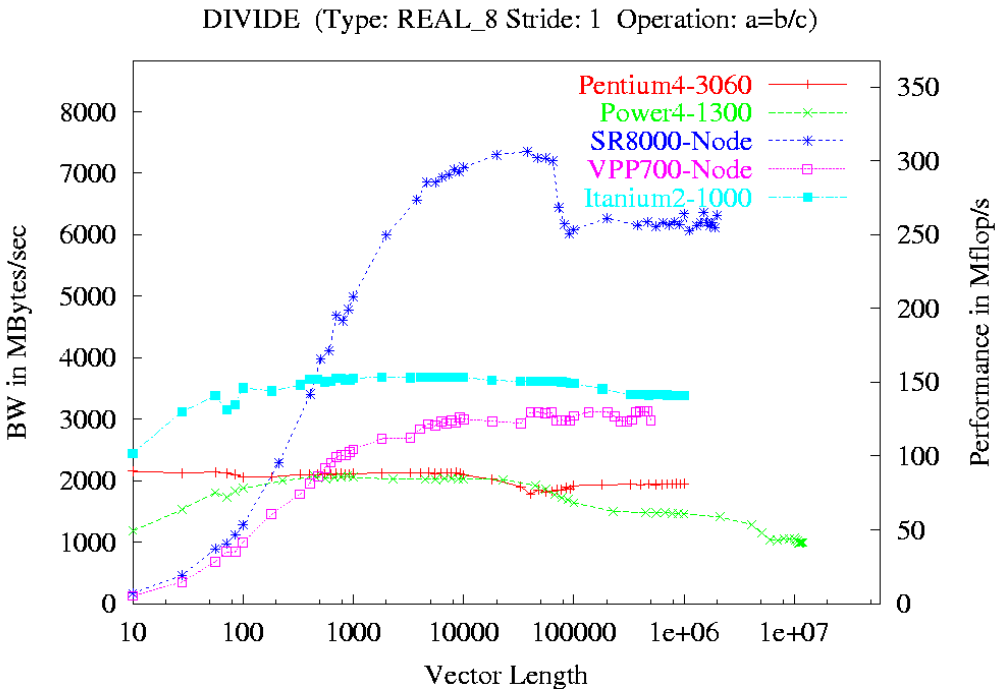

Abbildung 10: Bei Division reeller Zahlen schneidet der Pentium 4 am zweitschlechtesten ab; er wird sogar noch von der VPP700 geschlagen.

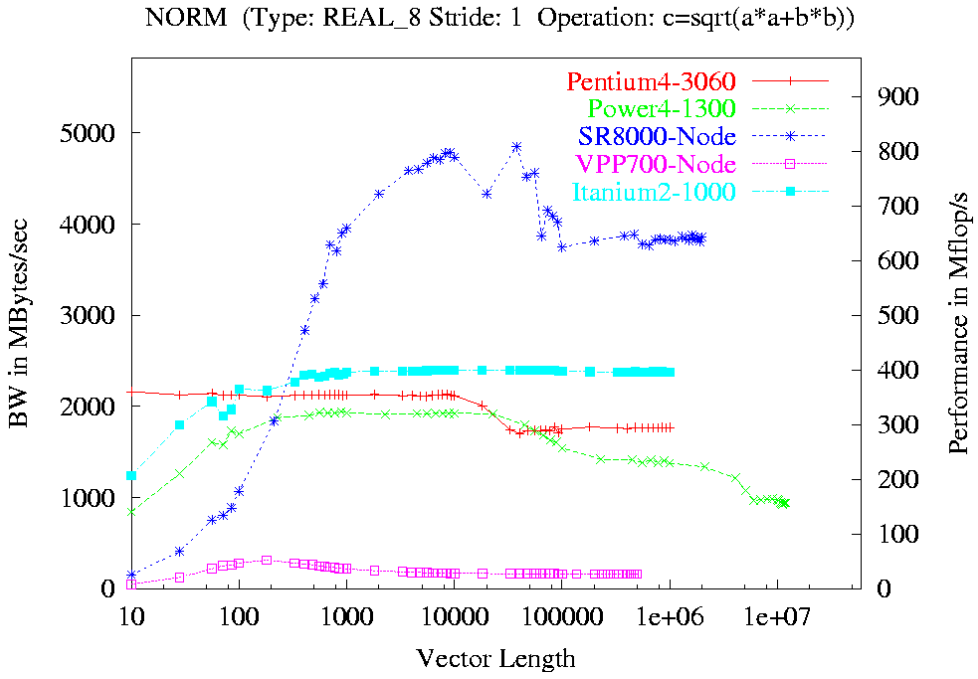

Abbildung 11: Bei der Berechnung der Norm bestimmt die Berechnung der Quadratwurzel die Rechenleistung. Auch hier schneidet die IA32-Architektur nicht überragend ab, sie fällt hinter IA64 zurück, liegt aber noch vor Power 4.

2.2 Applikations-Benchmark: Gaussian 98

Da die Betriebsanforderungen für die IA32-Erweiterung des Linux-Clusters in erster Linie die Durchsatz-Nutzung durch Quantenchemiker vorsehen, wurde ein typisches Beispiel aus der Quantenchemie mit einem Gewicht von 20% in die Leistungsbewertung aufgenommen. Mit am LRZ vorbereiteten ausführbaren Programmen für Gaussian 98 waren an einem Dekan-Molekül ($C_{10}H_{22}$) eine Dichtefunktional-Strukturoptimierung vorzunehmen und anschließend die Schwingungs-frequenzen zu berechnen. Messgröße war die Laufzeit des Programms. Zu Dokumentationszwecken wurden mehrere Messreihen durchgeführt; für die abnahmerelevante Messreihe wurden der beste und schlechteste Wert gestrichen und das Mittel über die vier verbleibenden Werte gebildet.

Messung Nr.	**Zeit (Sekunden)**			
	mit HT, 1 Thread	**mit HT, 2 Threads**	**ohne HT, SMP-Kernel**	**ohne HT, Uni-Kernel**
1	4263.9	3746.7	4270.1	**4222.6** [2)]
2	4268.3	3747.3	4270.4	**4217.2**
3	4311.3	3773.3	4291.7	**4220.1**
4	4263.9	3756.7	4287.0	**4216.5**
5	4264.6	3761.4	4280.7	**4215.2** [1)]
6	4391.8	3745.9	4277.5	**4216.8**
2.2.1.1.1 Mittelwert	4294.0	**3755.2**	4279.6	**4217.7**
Commitment				4241

1) Schlechtester Messwert, gestrichen
2) Bester Messwert, gestrichen

Die Messdaten belegen, dass mit einem SMP-Kernel und/oder eingeschaltetem Hyperthreading (HT) und dem vorgegebenen Gaussian-Konfigurationsfile (das nur einen Programm-Thread zulässt) das von MEGWare abgegebene Commitment noch um 0.9-1.2% verfehlt wird. Die für die Abnahme relevanten Messungen wurden dann mit einem Uniprozessor-Kernel bei abgeschaltetem HT ausgeführt. Durch Tuning der Festplattenparameter (für die von Gaussian erzeugten, recht großen Zwischendateien) konnte das Commitment sogar um mehr als 20 Sekunden übertroffen werden. Dennoch ist zu konstatieren, dass bei tatsächlicher Nutzung von 2 Threads die Effizienzsteigerung durch Multi-Threading (11.5%) die obengenannten Verluste um eine Größenordnung übertrifft; Hyperthreading sorgt also im Normalbetrieb mit zwei Jobs pro CPU durchaus für eine signifikante Erhöhung des Durchsatzes. Applikationsseitig ist dies bei Gaussian 98 wohl auch wesentlich eine Folge der Überlagerung von I/O mit Berechnungen.

Zum Vergleich mit anderen Architekturen wird in Abbildung 12 die Ausführungszeit auf dem Pentium 4 mit denen auf Power4, IA64-2 (Itanium2), SR8000 und VPP verglichen. Hier zeigt sich, dass der Itanium2 trotz seiner relativ langsamen Taktung ein besseres Resultat als der Pentium 4 erzielt. Allerdings zeigen Experimente mit Einsatz Pentium 4-spezifischer Optimierung und Verwendung der BLAS von K. Goto, dass sich die Laufzeit auf IA32 noch einmal um etwa 800 Sekunden reduzieren lassen sollte; der resultierende Code ist leider gegenwärtig für die Produktion nicht geeignet. Das sehr gute Ergebnis auf

dem SR8000 Knoten sollte nicht darüber hinwegtäuschen, dass diese Maschine nicht für Durchsatzbetrieb der oben beschriebenen Art gedacht ist. Auf einem Vektorrechner wie der VPP700 laufen die in Gaussian implementierten Dichtefunktionalmethoden grundsätzlich mit schlechter Leistung (das gilt jedoch nicht notwendigerweise für andere Methoden); das war für das LRZ ja auch mit ein Grund für die vorliegende Beschaffung.

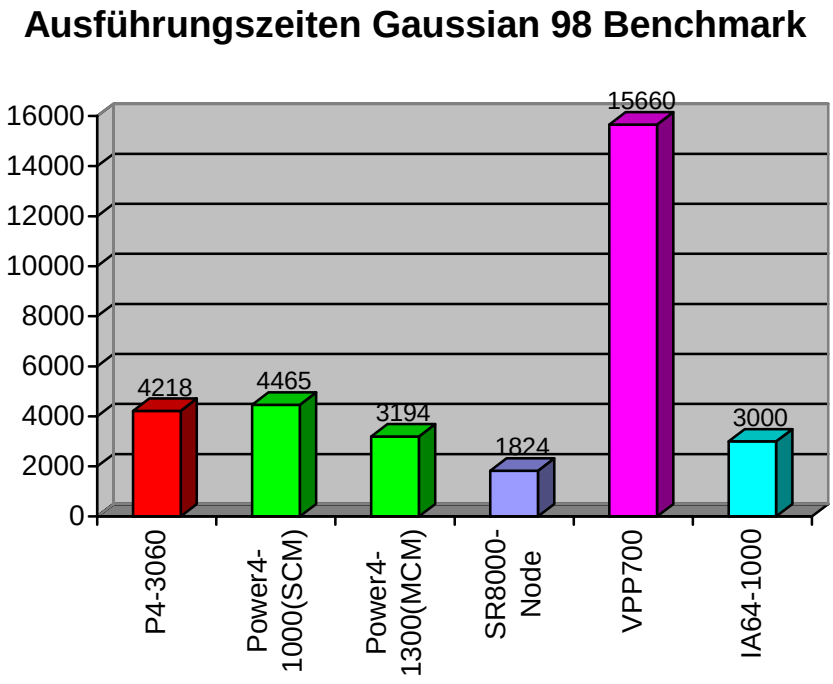

Abbildung 12: Leistungsvergleich Pentium 4 mit anderen Architekturen beim Gaussian 98 Benchmark

2.3 Hyperthreading auf Pentium 4 Prozessoren

Hyperthreading (HT) bedeutet, dass zwar nur eine physische, aber zwei logische Prozessoren auf einem System zur Verfügung stehen. Dadurch wird zwar die Zahl der funktionalen Einheiten nicht erhöht, wohl aber die Zahl der Register und damit der Instruktionsströme. Das soll eine Überlappung von Instruktionen und damit eine Erhöhung des Instruktionsdurchsatzes ermöglichen. Um eine etwas quantitativere Vorstellung von den HT-Fähigkeiten der neueren Pentium 4 Steppings zu gewinnen, wurde mit synthetischen Programmen untersucht, welche zusätzliche Leistung für bestimmte Kombinationen von Codes bestenfalls erzielt werden kann.

2.3.1 Benchmark-Programm und Methodologie

Eine spürbare Durchsatzverbesserung für ein paralleles Programm ist nur zu erwarten, wenn die Threads unterschiedliche funktionale Einheiten nutzen. Daher wurde unter Verwendung von OpenMP *parallel sections* der in folgender Abbildung illustrierte Algorithmus implementiert, der parallel Vektortriaden *unterschiedlichen* Datentyps (4-Byte Integer und 8 Byte Real) ausführt.

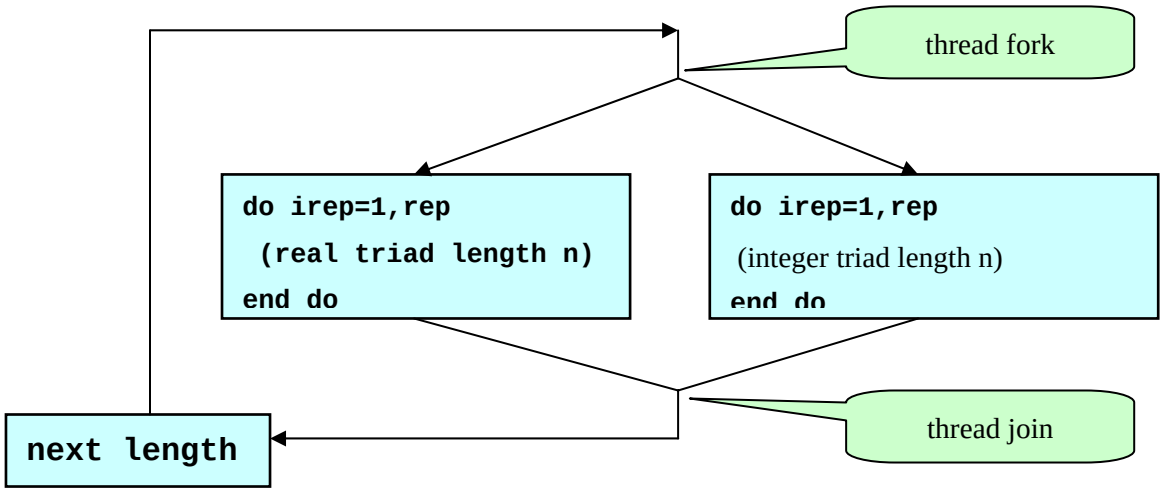

Abbildung 13: Synthetischer Benchmark zur Bewertung des Hyperthreading auf IA32

Das Benchmark-Programm wird dann auf einem single-CPU Rechner ausgeführt. Da in der Regel ein Thread mit seiner Teilaufgabe deutlich früher fertig wird als der andere, wurde zur Zeitmessung folgende Methode verwendet:

1. Beide Teilaufgaben werden im single-thread Modus vermessen, sodass die *seriellen* Rechenleistungen S_{int} und S_{float} bekannt sind.
2. Für den parallen Lauf erhält man zwei Intervalle T_{int} und T_{float}. Ist etwa $T_{int} < T_{float}$, so erhält man für die Vektorlänge N_{int} die parallele Integer-Leistung aus

$$P_{\text{int}} = \frac{2N_{\text{int}}}{T_{\text{int}}},$$

während die Real-Schleife teilweise seriell lief, also bei Berechung der parallen Leistung für diesen seriellen Zeitabschnitt eine Korrektur einzufügen ist:

$$P_{float} = \frac{2N_{float} - S_{float} \cdot (T_{float} - T_{\text{int}})}{T_{float}}$$

Falls umgekehrt $T_{int} > T_{float}$ gilt, sind in beiden obigen Formeln die Indices „int“ und „float“ zu vertauschen. Der Faktor zwei in beiden Formeln rührt daher, dass pro Schleifendurchlauf eine Addition und eine Multiplikation ausgeführt werden.

Führt man das Programm auf Pentium 4 Prozessoren mit bzw. ohne Hyperthreading aus, erwartet man im letzteren Falle, dass die Rechenleistung für Integer und Real auf jeweils die Hälfte absinken, während man beim Einsatz von HT darauf hoffen kann, dass der Performance-Einbruch nicht so groß ist. Das motiviert die folgende Definition des Durchsatz-Gewinns (in %, für Integer):

$$\Delta Eff_{\text{int}} = 100 \cdot \left[\frac{2 \cdot P_{\text{int}}}{S_{\text{int}}} - 1 \right],$$

die entsprechende Größe für „float“ ist analog definiert. Dieses Maß ist im Übrigen auch unabhängig von der Taktung des verwendeten Prozessors, nicht allerdings vom Memory-Subsystem (dieses kann ja auch innerhalb der Pentium 4 Familie recht unterschiedlich ausgelegt sein). Als Vergleichssystem, das hardware-seitig nur single-threaded läuft („ST“) wurde dementsprechend eines der auch am LRZ verfügbaren Pentium 4-1500 Systeme verwendet. Ausserdem wurden ausführbare Programme auf drei Arten erstellt:

1. Mit dem Release 7.0 des Intel Compilers, abgekürzt *Intel*
2. Mit dem Release 4.0 des PGI Compilers, abgekürzt *PGI*
3. Mit Unterstützung von „nontemporal writes“ durch den PGI Compiler, abgekürzt *nPGI*; das liefert verbesserte Leistung bei Zugriff auf den Hauptspeicher, allerdings auf Kosten der Leistung aus den Caches.

2.3.2 Messergebnisse und Interpretation

Zur Überprüfung der Plausibilität der Definition von ΔEff betrachten wir zunächst die Resultate für den Prozessor ohne HT.

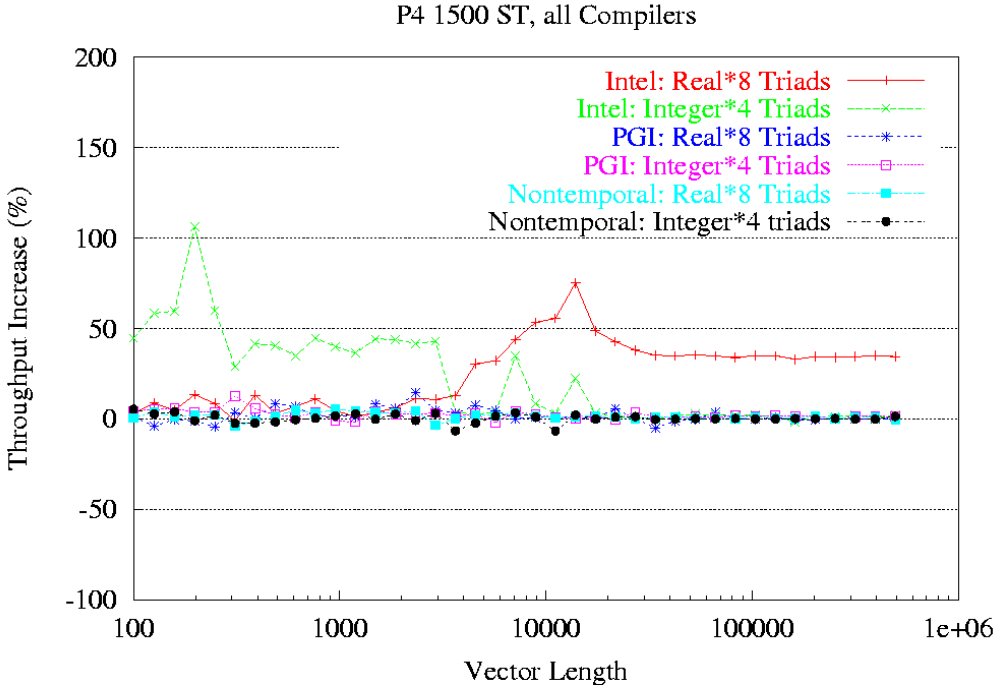

Abbildung 14: Referenzmessungen auf Prozessor ohne Hyperthreading

Hier sollte sich grundsätzlich überall eine Null-Linie zeigen; für die mit dem Intel-Binary durchgeführten Messungen trifft das jedoch nicht zu, und zwar für Integer im Cache, und für Real im Hauptspeicher. Hier scheint man Gewinne zu machen, obwohl kein HT verfügbar ist. Betrachten wir zunächst den ersten Fall, nämlich die Integer Performance etwa für N=1192, anhand der auf Taktfrequenz normierten (Daten-) Bandbreiten. Die Gesamtbandbreite für das mit 2 Threads laufende Programm ergibt sich dann zu

$$B_{parallel} = 16 \cdot P_{real} + 8 \cdot P_{\text{int}}$$

Diese Gesamtbandbreite wird im Balkendiagramm (Abbildung 15) durch die *ganz rechts* liegende Vierergruppe repräsentiert. HT steht dort generell für Hyperthreading (den Pentium 4-3060 Prozessor), ST für Single-Threading (also den Pentium 4-1500 Prozessor).

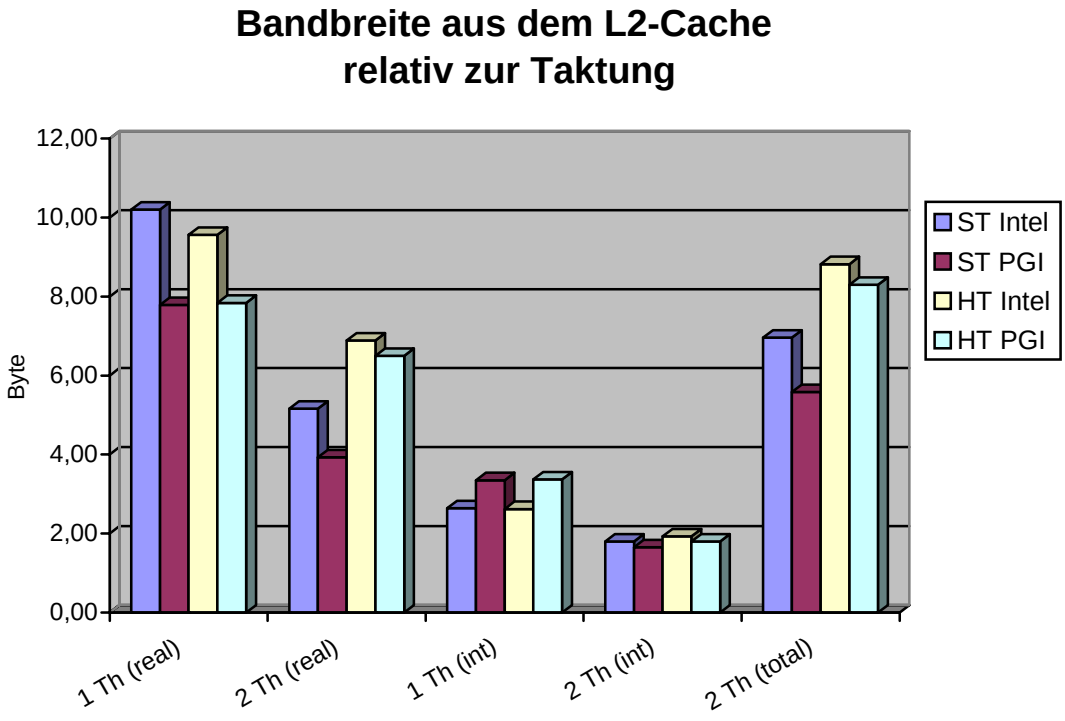

Abbildung 15: Vergleich von Single- (ST) und Hyper-Threaded (HT) Prozessoren bezüglich der in-Cache Bandbreite

Die Bandbreite wird jeweils noch durch die Taktfrequenz dividiert, um die unterschiedlich getakteten Systeme HT und ST direkt vergleichen zu können. Man sieht zunächst, dass die durch die Abkürzung „1 Th“ gekennzeichneten Einzelthread-Bandbreiten sich bei Übergang ST → HT praktisch nicht ändern, wie zu erwarten ist. Bei Gleitpunktzahlen liefert der Intel-Compiler die besseren Werte, bei Ganzzahlen der PGI Compiler. Betrachtet man jetzt die 2-Thread-Zahlen, so steigt beim HT-Prozessor die Real-Bandbreite sehr deutlich an, nicht jedoch die Integer-Bandbreite. Der Netto-Effekt ist, dass mit dem HT-System praktisch die L2-Bandbreite auch mit zwei Threads voll ausgeschöpft werden kann, während auf dem ST-System die sich gegenseitig störenden Instruktionsströme letzten Endes auch den Datenfluss bremsen. Warum also scheint für die Ganzzahl-Performance des mit dem Intel-Compiler erstellten Programms auch das ST-System einen Effizienzschub zu bringen? Es ist zu vermuten, dass der Programmcode vom Compiler so schlecht optimiert wurde, dass durch passendes Thread-Scheduling schon auf ST eine Verbesserung im 2-Thread Code auftritt. In jedem Falle bedeutet dies, dass man den schon auf ST erzielten „scheinbaren“ Effizienz-Gewinn nicht dem Hyperthreading zurechnen kann. Der entsprechende Effekt tritt allerdings für Gleitpunkt-Zahlen bei PGI-Compiler nicht auf, der prozentuale Unterschied ist dort etwa genau so groß. Die folgende Abbildung 16 zeigt für alle in die Caches passenden Vektorlängen den Effizienzgewinn für HT.

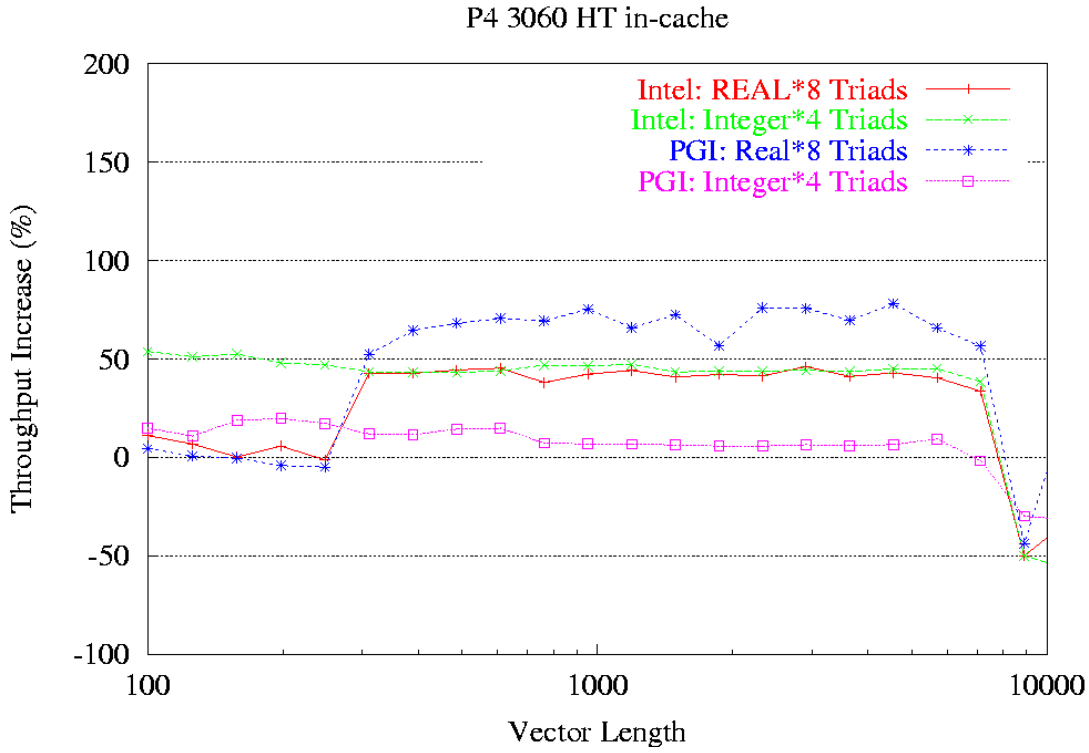

Abbildung 16: In-Cache Effizienzgewinn für Prozessoren mit Hyperthreading

Man sieht also, dass man mit dem Intel-Compiler für Real im L2 Cache etwa 50% Durchsatzverbesserung erhält, beim PGI-Compiler ist die Situation etwas anders: Die Verbesserung bei Integer ist geringfügig, bei Real ist sie dafür im Bereich von 70-75%. Zum Vergleich mit dem Gewinn an Gesamtbandbreite beim Übergang ST → HT: Für den Intel Compiler beträgt er 27%, für den PGI Compiler 48%. Verantwortlich dafür ist die deutlich geringere Basis-Gleitpunktleistung des vom PGI Compiler erzeugten Codes. Effizienz-*Verringerung* ergibt sich für Vektorlängen ab etwa 8000 deswegen, weil dem nur einfach vorhandene L2 Cache bei Nutzung von zwei Threads schon eher der Platz ausgeht als bei Nutzung durch einen einzigen. Unklar ist, warum im L1 Cache für Real *kein* Leistungszuwachs gemessen wird.

Betrachten wir nun die entsprechende Situation beim Zugriff auf den Hauptspeicher, also etwa bei Vektorlänge N≅100000. Hier skalieren wir die erzielten Bandbreiten nicht mit der Taktung, sondern der nominellen Speicherbandbreite, um ebenfalls wieder HT und ST direkt vergleichen zu können. Für das ST-System beträgt sie 3,2 GByte/s, für das HT-System 4,2 GByte/s. Außerdem verwenden wir hier als Vergleichszahl die für Hauptspeicherzugriffe optimierte Variante *nPGI*. Das Balkendiagramm (Abbildung 17) gibt einen Überblick; die Effizienzgewinne sind in Abbildung 18 illustriert.

Insgesamt gilt: bei Zugriffen in den Hauptspeicher bringt Hyperthreading offenbar keine Vorteile. Das ist klar, denn angesichts des im Vergleich zu den Caches von vorn herein um über eine Größenordnung kleineren Durchsatzes nützt Optimierung des Instruktionsflusses im Prozessor selbst nichts.

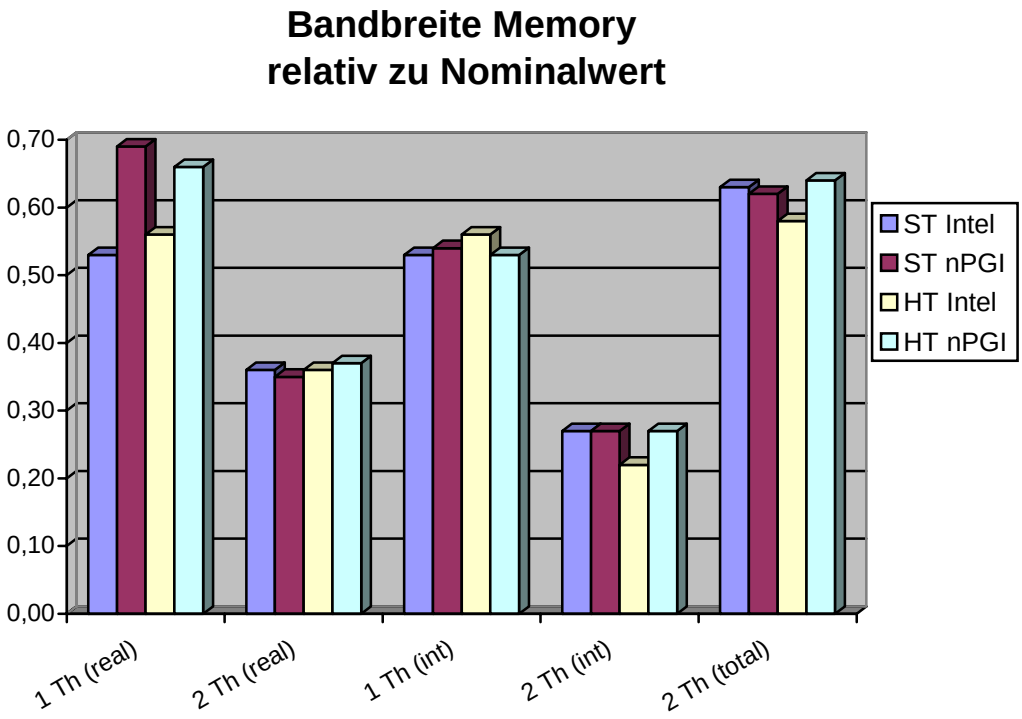

Abbildung 17: Vergleich von Single- (ST) und Hyper-Threaded (HT) Prozessoren bezüglich der Memory-Bandbreite

Für den scheinbaren Durchsatzgewinn beim Intel-generierten Real-Code kann man auch hier mangelhafte Optimierung verantwortlich machen, die von vornherein noch Bandbreite übrig läßt; Letztere kann bei Multi-Threading durch geeignetes Scheduling genutzt werden. Dafür spricht insbesondere, dass bei Einsatz von 2 Threads der Real-Teil von PGI und Intel mehr oder wenig gleiche Leistungsdaten aufweisen. Unklar bleibt, warum auf dem HT Prozessor die Ganzzahl-Leistung des Intel-Compilers bei 2 Threads einbricht.

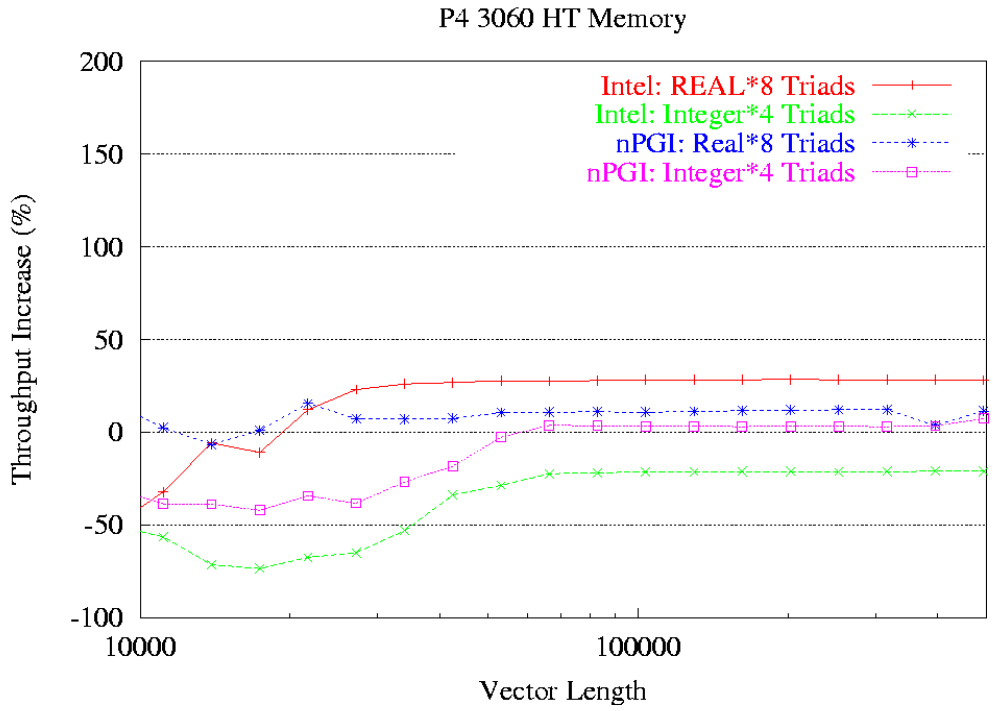

Abbildung 18: Effizienzgewinn des HT Prozessors bei Zugriff in den Hauptspeicher

Der Effizienzgewinn für die Intel/Reals ist dementsprechend wieder nicht anzurechnen, der Verlust für Intel/Integers ist in Abbildung 18 natürlich ebenfalls zu beobachten. Mit dem PGI-Compiler sieht man beim Übergang ST → HT etwa 3% Bandbreitenzuwachs; das dürfte noch innerhalb derjenigen Unsicherheit liegen, die durch BIOS-Einstellungen, Verwendung unterschiedlicher Speicherriegeltypen und Chipsätze bedingt ist und nicht allein durch den Unterschied der Busfrequenzen beschrieben werden kann.

2.3.3 Fazit

Hyperthreading ist im Wesentlichen nur bei Cache-basiertem Rechnen nützlich; bei konstanten Problemgrößen sollte überdies auf einer HT-CPU doppelt soviel Cache verfügbar sein. Das oben eingeführte Maß für die Effizienzsteigerung ist nur dann sinnvoll, wenn der Compiler schon für das seriell laufende Programm optimalen Code erzeugt, ansonsten muss man die bereits auf einer ST Plattform erzielten scheinbaren Effizienzsteigerungen bei der Bewertung berücksichtigen. Dasselbe gilt ggf. für den Einfluss des Betriebssystems auf den Programmablauf. Bei paralleler Verarbeitung von Ganzzahl- und Gleitzahlcode erreicht man für Ersteren Effizienz-Steigerungen um 10%, für Letzteren um 50%. Bezogen auf den Cache-internen Datenfluss für das *Gesamtprogramm* erhält man eine Performance-Verbesserung um 27%, wobei der vom Intel Compiler erzeugte seriell optimal (?) laufende Code zugrunde gelegt wird.

2.4 LINPACK Benchmark

Nach der Installation der Cluster-Erweiterung stehen im Cluster insgesamt 110 Pentium 4-3060 Knoten zur Verfügung; aufgrund der Vernetzungsstruktur mit 30+30+30+20 Knoten auf je einem GE-Switch erwies sich eine Konfiguration mit 105 Knoten in 15-er Gruppen als optimal. Es wurde die folgende Konfiguration des auf http://www.netlib.org/benchmark/hpl/ erhältlichen Standard-benchmarks verwendet:

- Blockgröße: 96
- Problemgröße: 112320
- Prozess-Grid: 7 x 15 (=105)
- Threshold: 16
- PFACT: left (0), NBMIN: 2, NDIV: 2, RFACT: crout (1), BCAST: 1rg (0), DEPTH: 0, SWAP: mix (2), threshold=64, L1 und U: transposed (0), Equilibration: ja (1), Memory alignment: 8

Der Benchmark benötigt zur Durchführung der Matrix-Matrix-Operationen eine BLAS; aufgrund der optimal erzielbaren Rechenleistung (s. u.) wurde die aus dem Internet über die URL http://www.cs.utexas.edu/users/kgoto/libraries/ erhältliche, für Pentium 4 (mit SSE2) optimierte BLAS-Bibliothek von Kazushige Goto verwendet. Die erzielte Rechenleistung war **350,0 GFlop/s**. Die Problemgröße $N_{½}$, bei der mit ansonsten identischen Parametern die halbe Rechenleistung erzielt wird, wurde näherungsweise zu

$$N_{½} = 40320$$

bestimmt; für diese Problemgröße wurden 172,0 GFlop/s gemessen.

2.4.1 Bemerkung über die BLAS-Leistung auf Pentium 4

Es wurde verglichen, was die BLAS-Implementierungen von Intel (MKL) und K. Goto (s. o.) auf dem Pentium 4 leisten; die Tests wurden mit einer Problemgröße von 8000 auf einem 2 x 2 Prozessor-Grid ausgeführt. Ausserdem wurde mit ein und zwei Threads gestartet, um den Einfluss des Hyperthreading bewerten zu können. Die gewählte Blockgröße war 192 (was nicht optimal ist).

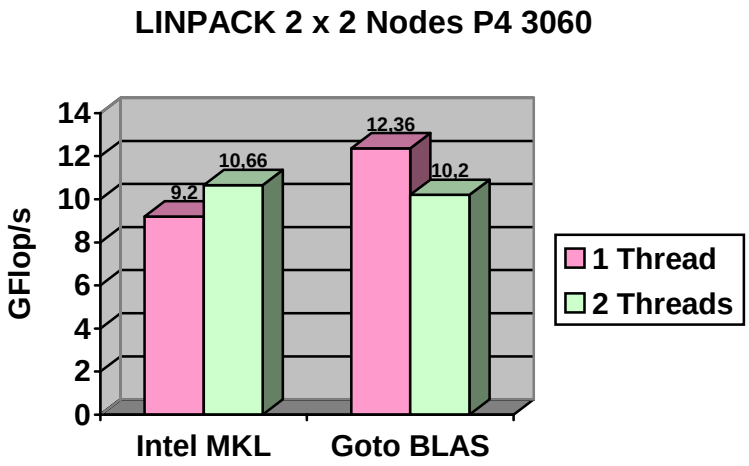

Abbildung 19: Performance-Vergleich MKL und Goto BLAS

Das in der MKL nicht perfekt optimierte Instruktions-Scheduling sieht man auch daran, dass bei Nutzung von 2 Threads noch ein Leistungsgewinn erzielt wird. Selbst dann gewinnt man mit der single-threaded Goto BLAS noch 16% mehr Leistung.

2.4.2 Parallele Effizienz beim LINPACK Benchmark

Für sehr viele Knoten hängt diese in zunehmendem Maße von der Latenz des Interconnects ab. Ein über TCP/IP gefahrenes Gigabit Ethernet (GE) ist mit Latenzen von ~100 µs sicher nicht optimal (es gibt theoretisch die Möglichkeit, einen nur für MPI-Verkehr ausgelegten Treiber direkt auf das GE aufzusetzen und damit die Latenzen um etwa eine Größenordnung zu drücken). Darüber hinaus sind in der LRZ Installation vier GE switches mit je 30 Anschlüssen über Port Trunking mit einer Bandbreite von 4 Gbit/s gekoppelt, sodass beim Versenden von Nachrichten über Switch-Grenzen hinweg sich die Latenzen um bis zu 2 „hops“ (Hüpfer) vergrößern können. Dementsprechend werden Messungen mit bis zu 30 Knoten Switch-intern ausgeführt, Messungen mit bis zu 60 Knoten beinhalten einen „hop“, darüber hinaus benötigt man 2 „hops“. Durch geeignete Anordnung des Process Grid im Benchmark kann man versuchen, die Kommunikationsmuster möglichst lange Switch-intern zu halten. Dies wurde jedoch aus Zeitgründen nicht sehr systematisch durchgeführt. Abbildung 20 zeigt Leistung und parallele Effizienz (bezogen auf die Basisleistung von 4.849 GFlop/s für eine CPU) für verschiedene Knotenzahlen. Wie empfindlich die Effizienz bei großen Knotenzahlen von der Wahl des Prozess-Grids abhängt, zeigen die sehr unterschiedlichen Zahlen für N=104 (das 8 x 13 Grid liefert 13% mehr Leistung als das 13 x 8), N=105 (das 7 x 15 Grid liefert 18% mehr Leistung als das 15 x 7) und N=110 (das 10 x 11 Grid liefert 16% mehr Leistung als das 11 x 10).

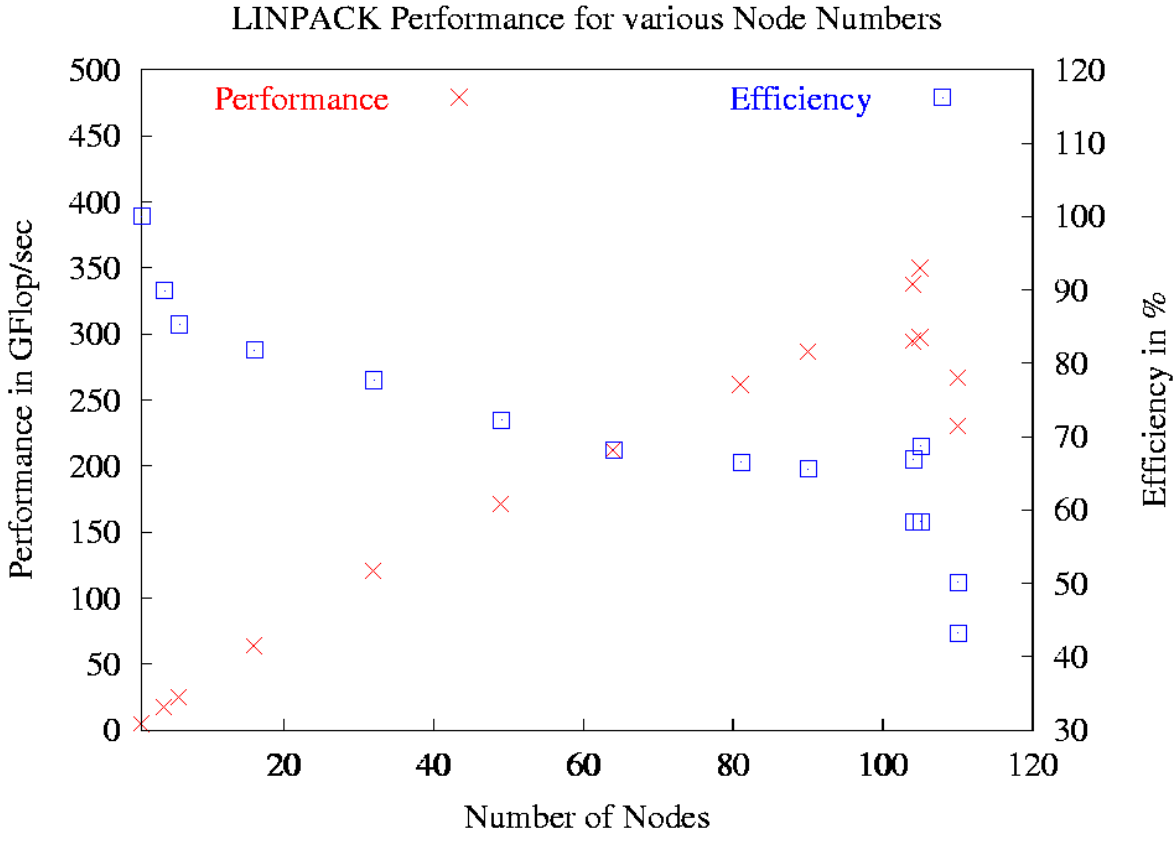

Abbildung 20: Leistung und Effizienz beim LINPACK Benchmark

3 Zusammenfassung

Intel hat seine IA32 Prozessoren im Hinblick auf die hohe Verbreitung für Desktop und transaktionsorientierte Server-Aufgaben im kommerziellen Bereich optimiert. Für Problemstellungen aus dem wissenschaftlichen Hochleistungsrechnen, deren serieller Speicherbedarf unterhalb der durch die Architektur erzwungenen 2 GByte liegt, und deren Datenzugriffsmuster stark Cache-orientiert sind, erhält man auf Grund des enorm hohen System-Taktes bessere Leistungsdaten als mit jeder anderen derzeit auf dem Markt befindlichen Standard-Architektur (die Firma AMD mit ihren IA32 Klonen vielleicht ausgenommen). Für solche Aufgabenstellung ist der klassische Vektorrechner a priori nicht geeignet; zum Teil aber auch aus software-technischen Gründen, wenn nämlich die zentralen Rechenkerne nicht vektorisierbar programmiert wurden. Für kommerzielle Standard-Applikationen ist das zunehmend der Fall.

Anders sieht es aus, wenn eine Simulation rein speicherorientiert arbeitet (also eine geringe Operationsdichte aufweist) oder die serielle 2 GByte Schranke sprengt. Zwar besteht grundsätzlich die Möglichkeit, durch Einsatz eines Hochgeschwindigkeits-Netzwerkes (z. B. Myrinet) die Aufgabenstellung auf viele Einzel-CPUs zu verteilen. Dennoch wird die Skalierbarkeit der Rechenleistung auf Grund der im Vergleich zum Vektorrechner viel geringeren Bandbreite pro CPU auf dem Billig-Rechner deutlich zurückbleiben. Da eine Skalierungsverbesserung häufig durch Vergrößerung des Teilproblems erzielt werden kann, bleibt als Alternative nur die Migration auf eine 64 bit Architektur; die von der Firma Intel entwickelte Itanium CPU oder auch der Opteron Prozessor der Firma AMD wären hier als kostengünstige Alternativen zu nennen. Untersuchungen über diese Architekturen werden Gegenstand eigener Berichte sein.