Leibniz-Rechenzentrum
der Bayerischen Akademie der Wissenschaften

Technischer Bericht

Die Intel Itanium2 Architektur und ihr Einsatz im Hochleistungsrechnen

Dr. R. Bader (LRZ), H.-U. Schäfer (LRZ)

Sept 2004 **LRZ-Bericht 2004-02**

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

1.1 Vorbemerkungen

Seit August 2003 wird am LRZ ein auf der Itanium-2 Architektur basiertes Linux-Cluster aus 17 vier-Wege Systemen und Myrinet 2000 Interconnect betrieben. Auch am RRZE (regionales Rechenzentrum Erlangen) kommt diese Architektur zum Einsatz, im Gegensatz zum LRZ Cluster wird dort jedoch eine mit 28 CPUs bestückte shared Memory Maschine („Altix 3700“) betrieben; in dieser von der Firma SGI entwickelten Implementierung kommt statt des üblichen Speicher-Interfaces ein skalierbares, cache-kohärentes NUMA (ccNUMA) zum Einsatz.

Dieser Bericht beschreibt die Leistungs-Charakteristika der beiden obigen Systeme für eine Reihe von Benchmark-Programmen, und zieht einen Vergleich zu dem am LRZ noch betriebenen Vektorrechner Fujitsu-Siemens VPP 700, aber auch zu dem am LRZ betriebenen Höchstleistungsrechner Hitachi SR8000-F1, sowie der von IBM entwickelten Power4-basierten Multiprozessor-Architektur p690 „Regatta“.

1.2 Beschreibung der verwendeten Systeme

1.2.1 Itanium-basierte Architekturen

Die Prozessoren sowohl im LRZ Cluster als auch in der Altix 3700 am RRZE sind mit 1.3 GHz getaktet; sie haben einen 3 MByte großen L3 Cache. Aus dem 256 kByte großen, achtfach assoziativen L2 Cache können maximal 2 Loads und 2 Loads/Stores von 8 Byte Gleitpunkt-Worten in den Satz von 128 Gleitpunkt-Registern geladen werden, wodurch im besten Falle die doppelt ausgelegten Multiply-Add Einheiten eine Spitzenrechenleistung vom vierfachen der Taktfrequenz erreichen können. Auf Grund der Bank-Struktur des L2 Caches muss allerdings der Speicher für simultane Zugriffe auf mehrere Cache-Lines (die 128 Bytes groß sind) durch den Compiler geeignet über die Bänke verteilt werden, was nicht immer gelingt und damit zu Leistungs-Einbußen führen kann. Der 16 kByte große L1 Daten-Cache wird nur für Ganzzahl-Daten verwendet; der L1 Instruktionscache ist ebenfalls 16 kByte groß. Ein Teil der CPU-Register ist rotierend verwendbar und damit für ein effizientes Software-Pipelining optimal ausgelegt. Für den Transfer von Daten zwischen L2 und L3 Cache steht im Prinzip dieselbe Bandbreite zur Verfügung wie vom L2 zu den Registern. Allerdings müssen zum Einen immer komplette Cache-Lines transferiert werden – das Zusammensetzen von 32 Byte Chunks zur Cache-Line kostet etwas zusätzliche Zeit –, zum Anderen ist der L2 Cache kein write-through, sondern ein write-back Cache – es müssen also auch Worte, die nur geschrieben werden, zunächst aus dem Speicher geladen werden –, sodass die erzielbare Rechenleistung bei Verlassen des L2 Caches deutlich absinkt. Noch schlimmer wird es, wenn Daten aus dem Hauptspeicher angefordert werden: Der Systembustakt von 200 MHz lässt eine maximale Transferrate von 6.4 GByte pro Sekunde zu; kommt ein handelsüblicher Multiprozessor-Chipsatz (wie in den am LRZ installierten Systemen) zum Einsatz, müssen sich alle CPUs des Rechenknotens darüber hinaus diese Bandbreite teilen.

Als Hochgeschwindigkeits-Netzwerk kommt am LRZ Cluster ein Myrinet 2000 zum Einsatz; pro 4-Wege Multiprozessor sind zwei single-Port Karten mit einer aggregierten MPI Bandbreite von 2 * 622 MByte/s und einer MPI Latenz von 11 µs eingebaut.

Einen anderen Weg ist bei der Implementierung eines Itanium-Multiprozessorsystems die Firma SGI gegangen. Hier teilen sich maximal zwei CPUs die verfügbare Bandbreite zum Systembus, der nicht direkt, sondern über einen sog. SHUB Crossbar sowohl zum eigentlichen Hauptspeicher als auch zu einem benachbarten Knoten bzw. über das sog. NUMAlink zu anderen Knoten des Systems.

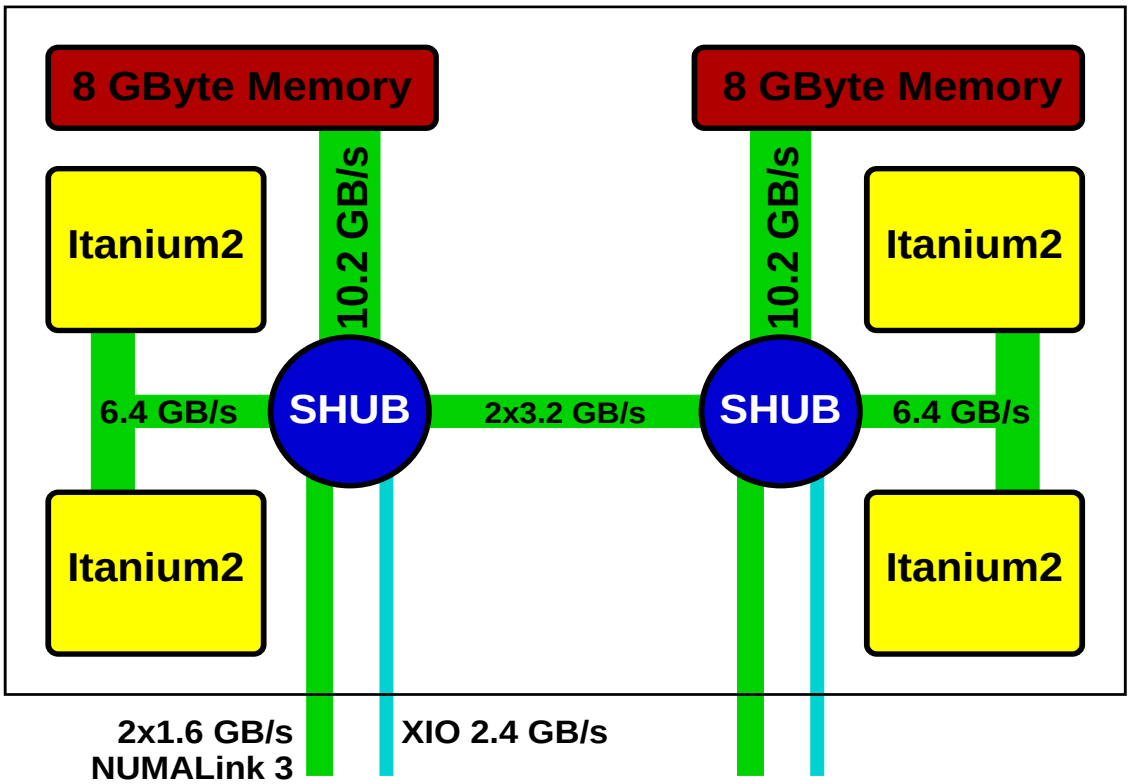
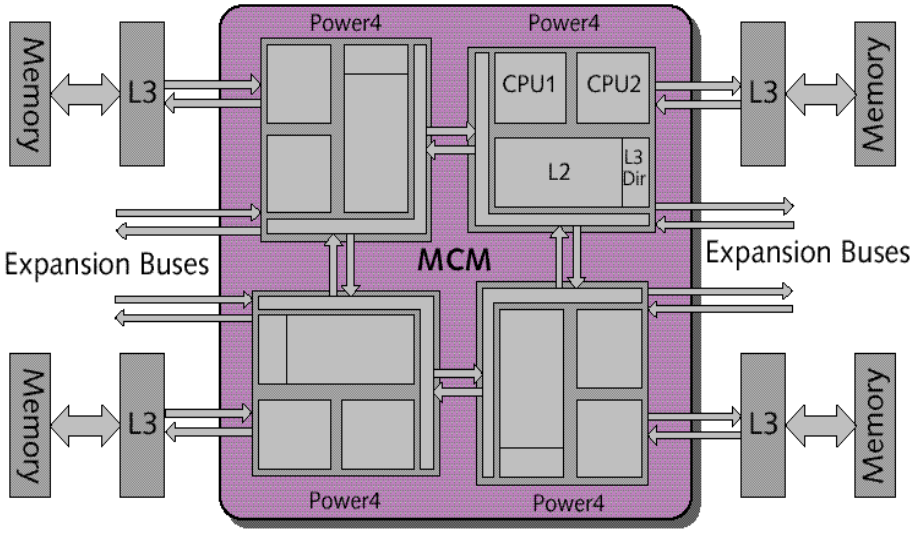


Abbildung 1: Altix Systemarchitektur

Abbildung 1 illustriert diese Systemarchitektur unter Angabe der verfügbaren Bandbreiten. Die NUMA-link Verbindungen zu anderen Knoten sind hierbei als sog. „Fat-Tree“ Topologie ausgelegt; darüber hinaus sind die Verbindungen cache-kohärent. Da der Zugriff eines Knotens auf den Speicher eines anderen Knotens mit kleinerer Bandbreite und größerer Latenz erfolgt, wird diese System-Architektur als cache-kohärentes non-uniform Memory (ccNUMA) bezeichnet. Durch geeignete Modifikation am Linux Betriebssystemkern können ggw. 128 CPUs als single Image System betrieben werden, getestet werden aber auch schon größere Systeme mit bis zu 512 CPUs.

1.2.2 Power4 Architektur

Im Jahr 2001 hat die Firma IBM eine auf dem PowerPC Instruktionssatz basierende, aber ansonsten architekturell vollkommen neuartige Entwicklung auf den Markt gebracht; Grundlage dafür ist ein dual-core Power4 Chip, der zumeist mit 1.3 GHz oder (Power4+) mit 1.7 GHz getaktet ist; für die hier durchgeführten Untersuchungen wurden mit 1.3 GHz getaktete Systeme verwendet. Wie der Itanium Prozessor kann die Power4 CPU maximal vier Gleitpunkt-Operationen pro Takt erzielen; die zwei Prozessoren eines Chips teilen sich 1.44 MByte L2 Cache. Der 36 MByte große L3 Cache, der einem Dual-Core Chip formal zugeordnet werden kann, liegt jedoch außerhalb des Prozessor-Chips, und ist darüber hinaus nur mit 333 MHz getaktet, was zu deutlich erhöhten Zugriffs-Latenzen und im Vergleich zur Itanium2 Architektur deutlich kleinerer Rechenleistung bei Zugriffen auf den L3 Cache führt. Darüber hinaus sind die Cache-Lines mit 512 Byte viermal so lang wie auf dem Itanium2, so dass diskontinuierliche Speicherzugriffe die Rechenleistung deutlich weiter herunterdrücken können als auf der Intel Architektur. Abbildung 2 zeigt, wie ein Multiprozessor MCM („multi-chip module“) aufgebaut ist; im Grunde stellt sich der L3 Cache als ein Memory Subsystem mit etwas erhöhter Effizienz dar.



Four Power4 chips in a single MCM package form an 8 processor SMP. Bandwidth to memory is 6.9 GByte/s per CPU, within the MCM 20.8 GByte/s and between MCMs 10.4 GByte/s. All bandwidths are bi-directional.

Abbildung 2: Power4 Multiprozessor-Architektur

Theoretisch kann man für ein mit 8 CPUs bestücktes MCM pro CPU eine Bandbreite von 6.9 GByte/s pro CPU erwarten: Wenn jede CPU auf ihren eigenen Speicher zugreifen kann, hat man eine skalierbare Architektur. In der Praxis sieht man jedoch deutlich weniger, wofür mehrere Ursachen massgeblich sind:

1. Die Bestückung mit Memory-Karten kann im ungünstigsten Fall die verfügbare Bandbreite halbieren
2. Die Verwendung von Speicherseiten üblicher Größe führt zu einer Leistungseinbuße um einen Faktor von mehr als 2; durch Verwendung von 16 MByte großen Speicherseiten („large pages“) lässt sich dies zwar beheben, jedoch ist diese Funktionalität betriebssystemseitig nicht ausreichend flexibel implementiert.
3. Eine perfekte Zuordnung von Hauptspeicher zu CPU („memory affinity“) ist in der Praxis bislang nicht vernünftig durchführbar; auf einem 32-Wege SMP führt das zu einer Verstopfung der Inter-MCM Verbindungen und damit zum Einbrechen der Skalierung.

Durch Verbesserungen bei Compiler- und Betriebssystem-Software wird sich ein Teil der hier diskutierten Probleme beheben lassen; gegenüber der Vorgänger-Architektur (Power3), deren Speicher-Interface für die Nutzung durch mehr als zwei CPUs nicht ausgelegt war, ist dieses Konzept dennoch ein enormer Fortschritt.

1.2.3 Hitachi SR8000

Der seit Anfang 2000 am LRZ betriebene Höchstleistungsrechner in Bayern, die Hitachi SR8000-F1, ist ein Cluster von 168 Acht-Wege SMP Knoten mit einem drei-dimensionalen Crossbar als Verbindungsnetzwerk. Letzterer besitzt eine MPI Latenz von 14 µs und eine bidirektionale Bandbreite von (2 *) 1 GB/s. Das Dateisystem ist als Single System Image ausgelegt, auf jedem Knoten läuft eine Instanz eines Mikrokern-basierten UNIX Betriebssystems, das jedoch, um laufende Anwendungen nicht zu beeinträchtigen, eine eigene (neunte) CPU auf dem Knoten für seine Aktivitäten (etwa I/O) nutzt. Für

seine proprietäre Prozessor-Entwicklung hat Hitachi den PowerPC Instruktionssatz von IBM lizenziert und um einige für das Hochleistungsrechnen wichtige Instruktionen erweitert. Die CPU ist im F1 Modell mit 375 MHz getaktet, es kann durch Nutzung von zwei FMA Einheiten auch hier maximal das vierfache der Taktfrequenz als Spitzenleistung erzielt werden, also 1.5 GFlop/s pro CPU oder 12 GFlop/s pro Knoten. Jede CPU hat 128 kByte L1 Daten-Cache, pro Knoten also 1 MByte akkumulierten Cache; angesichts der enormen Speicherbandbreite von nominell 32 GByte/s für den Knoten ist der Einsatz eines L2 Caches auf dieser Architektur nicht erforderlich. Die effiziente Nutzung dieser architekturellen Voraussetzungen beruht auf folgenden Mechanismen:

1. Durch compiler-gesteuertes Software **Pre-Fetching** von Daten aus dem Hauptspeicher in den Cache können kontinuierliche Datenströme die Speicherbandbreite voll ausnutzen
2. Für nichtkontinuierliche Speicherzugriffe kann die verringerte Ausnutzung von Cache-Lines dadurch umgangen werden, dass einzelne Speicherworte am Cache vorbei in den großen Registersatz geladen werden können (Pre-**Loading**); auch dies erfolgt durch den Compiler. Die Abbildung des großen Registersatzes auf die 32-Register des Instruktionssatzes erfolgt über eine Sliding-Window-Technik
3. Der Compiler unterstützt automatische Parallelisierung für viele, z. T. auch äußere Schleifenkonstrukte; ggf. kann man dem Compiler durch entsprechende Direktiven einen Hinweis darauf geben, dass eine Schleife zu parallelisieren ist. Um die Thread Startup-Zeiten, die bei dünnem Schleifenkörper die Leistung drücken können, klein zu halten, gibt es eine in Hardware implementierte Synchronisierungs-Instruktion.

Die unter den ersten beiden Punkten diskutierten Merkmale werden auch als „Pseudo-Vektorisierung“ bezeichnet, der dritte Punkt als COMPAS („Cooperative Multiprocessors in Shared Adress Space“); in vieler Hinsicht verhält sich der SR8000 Knoten auch aus Sicht der Applikationen wie eine leistungsfähige Vektor-CPU.

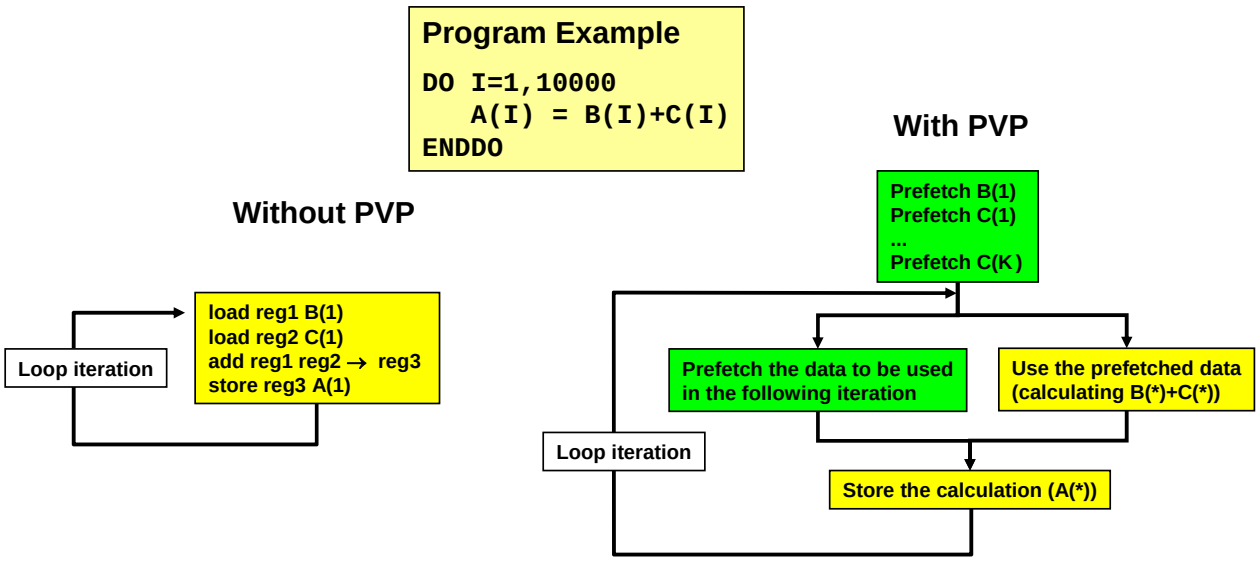


Abbildung 3:Prinzip des Prefetching der Hitachi SR8000-F1

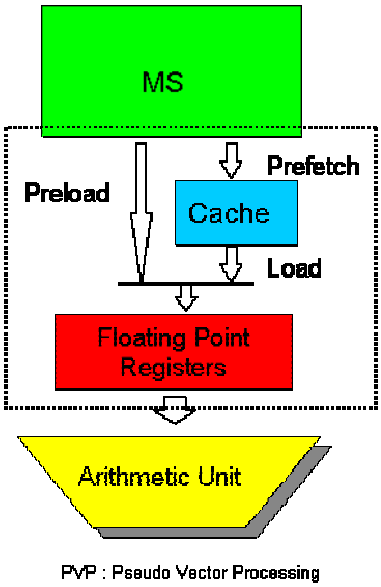


Abbildung 4: Prefetching und Preloading der Hitachi SR8000-F1

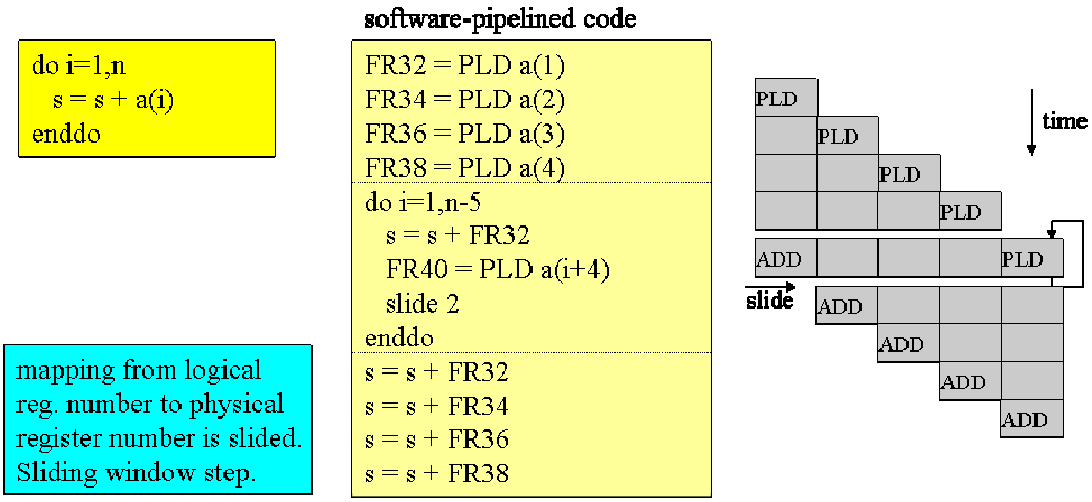


Abbildung 5: Preload und Sliding-Window-Technik

1.2.4 VPP 700 Architektur

Der Vollständigkeit halber soll auch diese, am LRZ seit 1997 betriebene Architektur kurz beschrieben werden. Am LRZ sind 52 Vektor-CPUs installiert, die durch einen Crossbar mit einer bidirektionalen Bandbreite von (2 *) 570 MByte/s verbunden sind. Die mit 143 MHz getakteten Prozessoren können auf maximal 8-fach ausgelegten Vektor-Pipes jeweils maximal 2 Gleitpunkt-Operationen ausführen, woraus sich eine Spitzenleistung von 2.29 GFlop/s errechnet. Die Vektor-Pipes arbeiten auf insgesamt 128 kByte langen, in einem Bereich von Zweierpotenzen frei konfigurierbaren Vektor-Registern. Begrenzt wird die Vektorleistung in der Praxis durch die notwendigen Vektor-Lade und Speicher-Operationen: Es stehen jeweils eine Lese- und eine Schreibeinheit auf den in 512fachem Interleaving ausgestatteten Hauptspeicher zur Verfügung, die eine Speicherbandbreite von 2 * 9 GByte/s ausschöpfen können. Der wesentliche Unterschied zu der Pseudo-Vektorisierung besteht darin, dass das Vektor-Pipelining in viel größerem Ausmaß in Hardware ausgeführt wird. Auf der VPP CPU gibt es für nicht vektorisierbare Operationen auch eine Skalar-Einheit mit einer Maximalleistung von 275 MFlop/s.

2 Synthetische Messungen am Speicher-Interface

2.1 Kontinuierlicher Speicherzugriff

Der mit 200 MHz getaktete Speicherbus der Itanium-Systeme gestattet eine maximale Übertragungsrate von 6.4 GByte/s, damit liegt die Speicherbandbreite erst bei etwas mehr als ein Drittel von dem, was die Fujitsu VPP 700 bereits im Jahre 1997 zu leisten imstande war. Für die Vektor-Triaden ist die Leistung der mit OpenMP parallelisierten Triaden-Schleife

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

auf einem 4-Wege SMP in Abbildung 6 illustriert. Für mehr als einen Thread erhält man bei kurzen Vektorlängen auf Grund der Thread-Startup Zeiten eine vektor-ähnliche Charakteristik, jedoch lässt sich dieser leistungsmindernde Effekt durch Versionierung – etwa durch Verwendung einer „`if (n < 7000)`“ Klausel in der parallelen Region reduzieren; es wird dann für kurze Vektoren nur ein Thread verwendet; eine ähnliche Strategie lässt sich auch auf dem Vektorrechner anwenden, um die für kurze Vektorlängen schlechte Leistung zu verbessern (dies ist hier jedoch nicht illustriert). Damit erhält man bis zur Ausschöpfung der Cache-Hierarchie auf dem Multiprozessorsystem die 3-7 fache Rechenleistung eines VPP Knotens. Für Streaming von Daten aus dem Hauptspeicher liegt jedoch, wie auf Grund der Spezifikation des Speicher-Interfaces zu erwarten, der Itanium Multiprozessor bei etwa einem Drittel der VPP Knotenleistung. Es ist klar, dass das günstige Preis-Leistungsverhältnis für die Itanium-Plattform den technischen Möglichkeiten für eine verbesserte Hauptspeicher-Anbindung Grenzen setzt; aus Sicht des technisch-wissenschaftlichen Hochleistungsrechnens stellen sich also zwei Fragen:

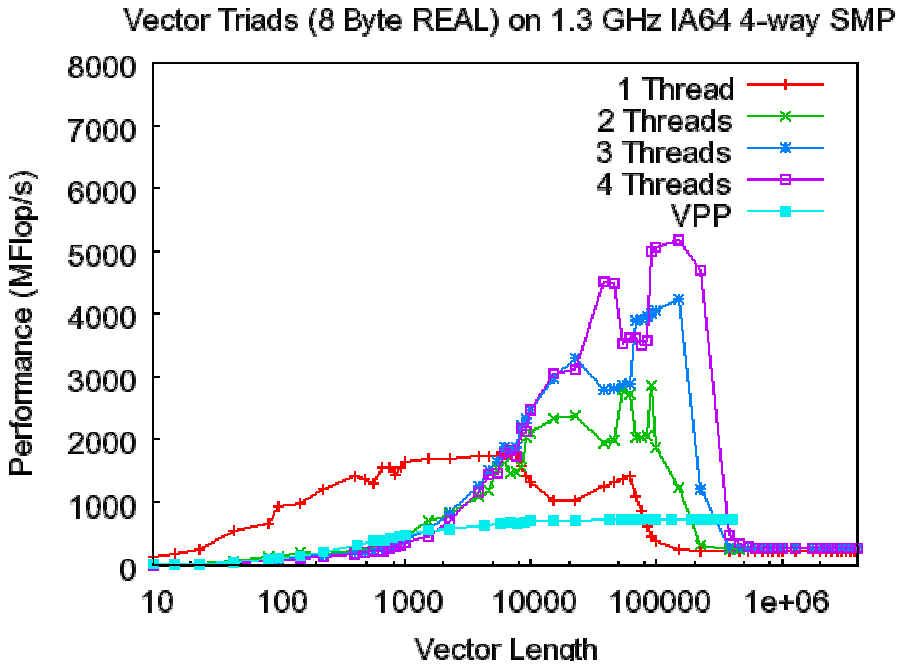


Abbildung 6: OpenMP parallele Vektor-Triade im Vergleich mit der VPP 700

Welche Applikationen können die Architektur dennoch – insbesondere durch Einsatz von Multiprozessor-Knoten – effizient nutzen?

1. Gibt es Möglichkeiten, trotzdem vergleichsweise kostengünstig ein System mit skalierender Hauptspeicher-Anbindung zu betreiben?

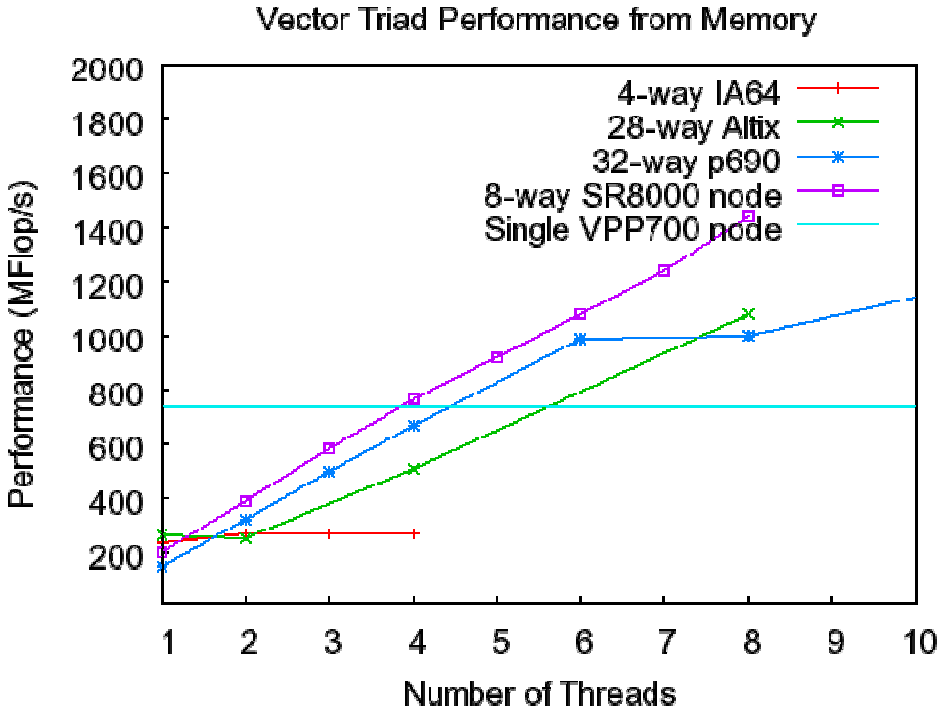


Abbildung 7: Skalierbarkeit der Speicher-Anbindung für Multiprozessoren

Zur Untersuchung der zweiten Frage betrachten wir die Triaden-Leistung aus dem Hauptspeicher für verschiedene Multiprozessor-Systeme (Abbildung 7); die erste Frage wird im Verlauf dieses Berichts noch wiederholt aufgegriffen.

Hier zeigt sich (grüne Kurve) das skalierbare Hauptspeicher-Subsystem des Altix-Systems; allerdings müssen sich jeweils zwei Prozessoren einen Speicherkanal teilen (in der Abbildung sieht man, dass die Werte für ein und zwei Prozessoren i. W. gleich sind), so dass der effektive Anstieg der Rechenleistung nur halb so schnell erfolgt als man im besten Falle erwarten könnte. Es werden dementsprechend unabhängig von ihrer Taktung 6 Itanium CPUs benötigt, um dieselbe Speicherstromleistung zu erzielen wie ein VPP Prozessor; bei der von Intel geplanten Erhöhung der Speicher-Taktung auf 266 MHz bzw. 333 MHz würde sich dieses Verhältnis auf 5 bzw. 4 Itanium CPUs pro VPP CPU verbessern, falls sich die Architektur sonst nicht verändert. Das Verhältnis 4:1 ist auf dem Hitachi SR8000 System schon jetzt realisiert, von den 32 GByte/s nomineller Speicherbandbreite können mit 8 CPUs über 23 GByte/s mit der Vektor-Triade gemessen werden. Etwas schlechter ist es um die Hauptspeicher-Anbindung der IBM p690 bestellt, obwohl dieses System eine Prozessorgeneration jünger als das von Hitachi ist; zwar können durch den Einsatz sog. „large pages“ (für diese Messungen nicht verwendet) Verbesserungen erzielt werden, jedoch ist die Verwendung dieses Betriebsmodus unter den gegenwärtig verfügbaren AIX-Betriebssystemen nicht ausreichend flexibel bzw. benutzerfreundlich implementiert.

2.2 Diskontinuierlicher Speicherzugriff

Betrachten wir zunächst reguläre diskontinuierliche Speicherzugriffe, die über eine Vektor-Triade mit Schrittlängen („strides“) realisiert werden, so tritt auf Hochleistungssystemen mit „interleaved“ Speichertechnologie bei gewissen Schrittlängen auf Grund von wiederholten Zugriffen auf dieselbe Speicherbank eine latenzbedingte Leistungsverminderung („bank busy time“) ein, wie sie für VPP700 und SR8000

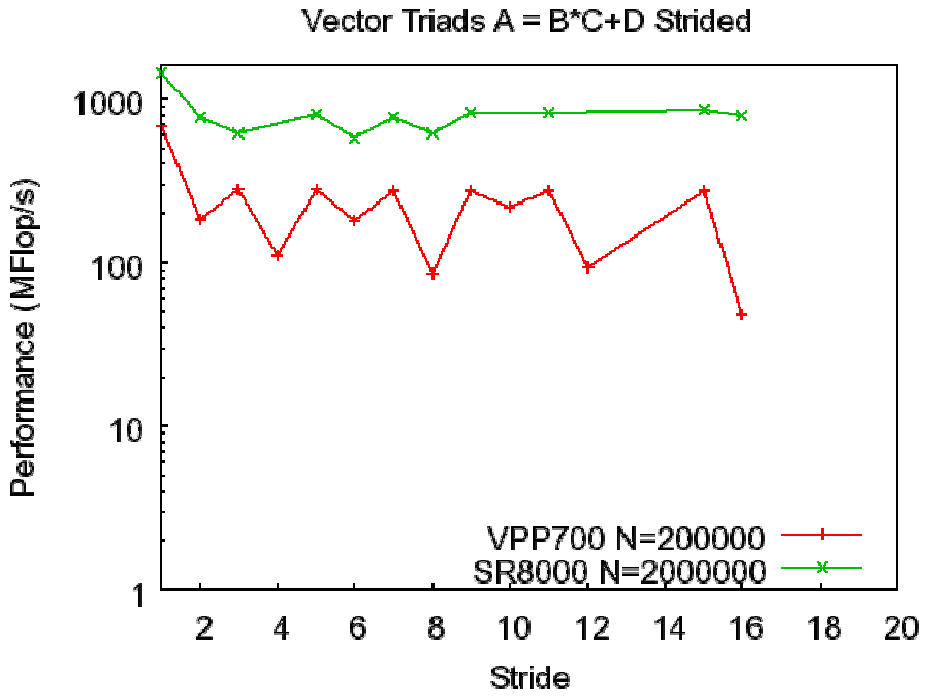


Abbildung 8: Regulärer diskontinuierlicher Speicherzugriff auf VPP und SR8000

in Abbildung 8 illustriert ist. Dieser Effekt ist auf der SR8000 wesentlich weniger ausgeprägt als auf der VPP 700, was auf eine deutlich kürzere „bank busy time“ schließen lässt. Auf der VPP kann man im besten Falle etwa 40% der Leistung von Schrittlänge 1 erreichen, auf der SR8000 sind es etwa 60%. Dieser hohe Prozentsatz ist auf die Pseudo-Vektorisierung durch Pre-Loading zurückzuführen, die ab einer Schrittlänge von 2 nicht mehr ganze Cache-Lines lädt, sondern einzelne Worte am Cache vorbei direkt in den umfangreichen Registersatz. Lädt man nämlich ganze Cache-Lines, so sinkt die Performance mit der Schrittlänge als Faktor ab, da nur ein entsprechender Bruchteil der Cache-Line auch benutzt wird. Für die Itanium Architektur wird das in Abbildung 9 sichtbar. Für das Laden von Daten aus dem Speicher (blaue Kurve) erhält man, wie die Fit-Kurve zeigt, das beschriebene Leistungsverhalten, bis bei einer Schrittlänge von 16 nur noch genau ein Wort pro 128 Byte großer Cache-Line erhalten wird, danach bleibt die Leistung konstant auf dem Niveau, das durch die maximale Zahl von Cache-Lines bestimmt ist, die für die Triade aus dem Hauptspeicher geholt werden (4 pro gerechnetem Element) bzw. geschrieben werden (1 pro gerechnetem Element) kann.

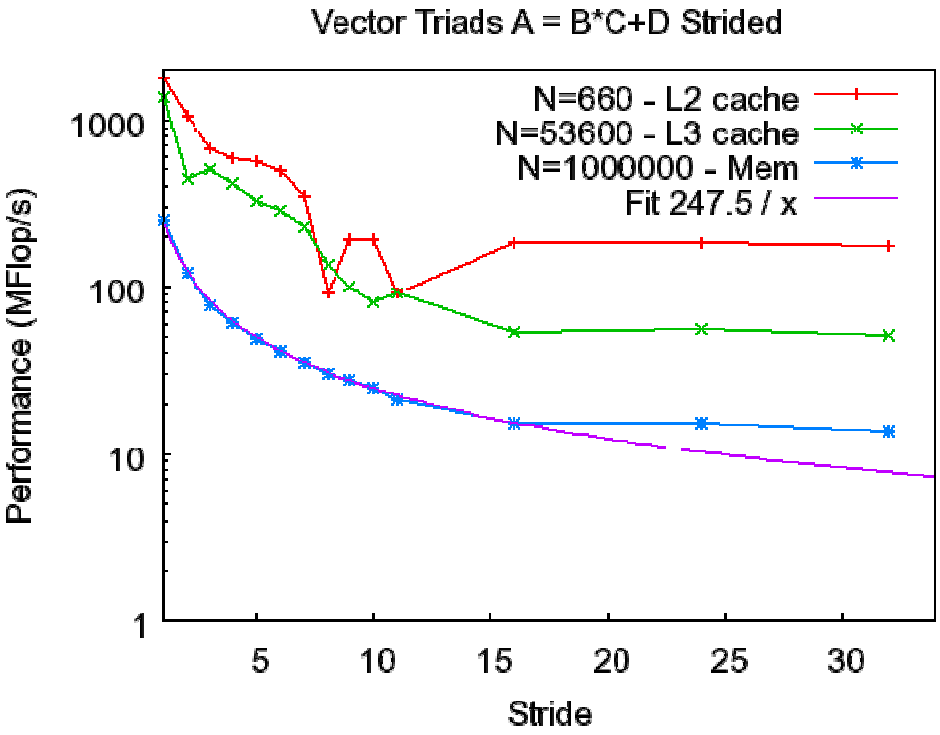


Abbildung 9: Regulärer diskontinuierlicher Zugriff auf Itanium CPUs

Derselbe Effekt tritt für solche Vektorlängen ein, bei dem Daten vom L3 in den L2 Cache transferiert werden müssen; auf Grund der höheren dort verfügbaren Bandbreiten liegt die Gesamtleistung auf höherem Niveau. Es ist nicht ganz klar, warum auch beim Laden von Daten aus dem L2 Cache in die Register ein ähnlicher Effekt auftritt; hier könnte die Handhabung der rotierenden Register und/oder das Banking des L2 Cache eine Rolle spielen: in jedem Fall sinkt auch bei L2 Zugriffen die Leistung auf etwa 10% der bei Schrittweite 1 erzielten Leistung ab. Dennoch werden auf der Itanium Architektur, solange man im L2 oder L3 Cache arbeitet und nicht zu große Schrittweiten auftreten, durchaus Leistungszahlen im Bereich der SR8000 – und damit deutlich oberhalb der VPP 700 – erreicht.

2.3 Sparse Benchmark

Dieses Benchmark-Programm beinhaltet eine Multiplikation dünn besetzter Matrizen der Form

$$y_{i,j} = \sum_{m=1}^{50} b_{j,m} \cdot x_{i,k(j,m)} ,$$

wobei der Index i zwischen 1 und der Blockgröße i_{Block} variiert; für letztere sind in dieser Untersuchung Werte von maximal 16 möglich (es wird angenommen, dass dies die maximale Anzahl der zu behandelnden Vektoren ist); verwendet wird der für die jeweilige Rechner-Architektur optimale Wert. Der Index j durchläuft Werte zwischen 1 und n, wobei die Vektorlänge n zwischen 10 und 90000 variiert wird. Der indirekte Zugriff $k(j,m)$ auf die Matrix x kann Werte zwischen 1 und 540000 annehmen, d. h. aus der gro-

ßen Matrix *x* werden nur wenige Elemente entnommen. Die Blockung im Index i bewirkt, dass die geladenen Cache-Lines wenigstens einigermaßen genutzt werden können, hierzu muss die Blockgröße natürlich ausreichend groß gewählt werden. Für alle Multiprozessorsysteme wurde darüber hinaus automatische Parallelisierung verwendet. Das Resultat ist für verschiedene Architekturen in Abbildung 10 dargestellt. Für die VPP 700 sind die Blockgrößen zu klein, um eine ausreichende Vektorleistung erzielen zu können; hingegen kann die Hitachi SR8000 bei Problemgrößen n<1000 offenbar zusätzliche Leistung aus den akkumulierten L1 Caches erzielen; insgesamt zeigt sich die SR8000 als die von allen hier betrachteten Architekturen als die am besten balancierte. Für die IA64 Architektur zeigen sich viel deutlichere Thread-Startup Zeiten als für die SR8000, obwohl auf beiden Architekturen die mittlere Schleife (über *j*) parallelisiert wird (die äußere geht über den Index *m*; über den innersten Index *i* wird ggf. entrollt). Hier greift offenbar die auf der SR8000 verfügbare Hardware-Synchronisierung.

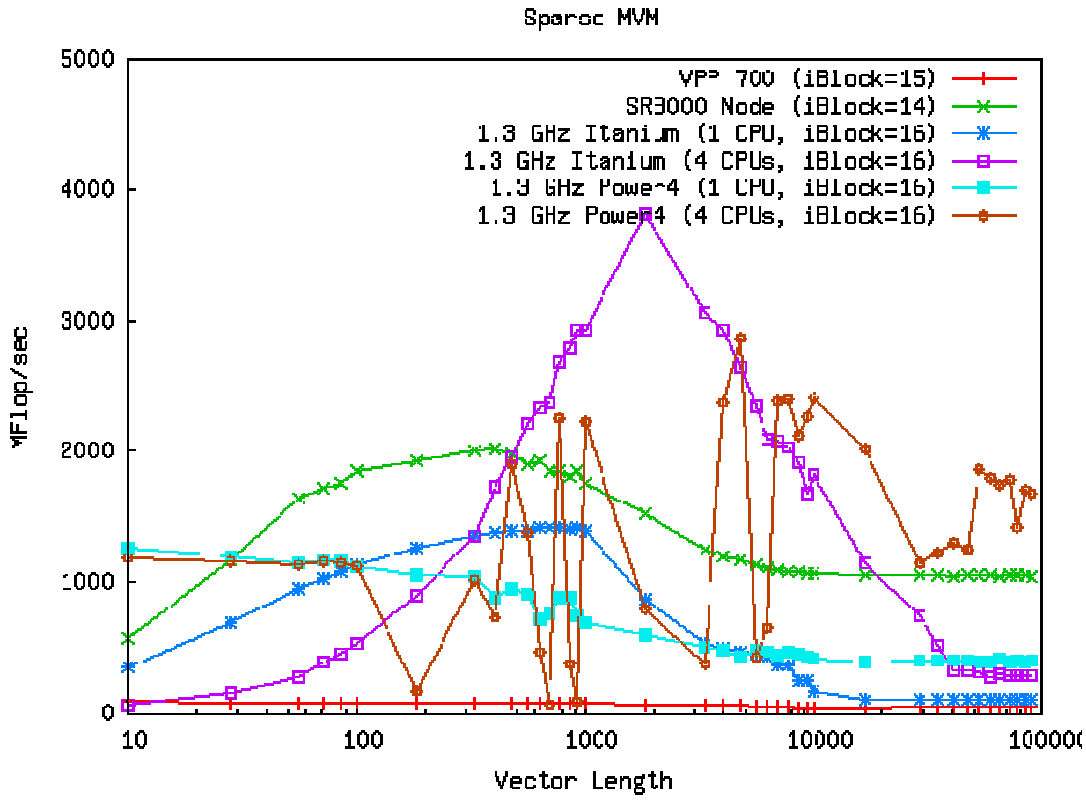


Abbildung 10: Sparse Benchmark auf verschiedenen Architekturen

Ein aus 4 CPUs bestehender Itanium-Knoten kann mit diesem Benchmark maximal etwa 18% der Spitzenrechenleistung ausschöpfen; die SR8000 liegt für den Maximalwert im selben Bereich (rund 17%). Im Gegensatz zum Itanium-Knoten erhält man aber für große Problemstellungen auf dem SR8000 Knoten immer noch 8.8% der Spitzenleistung, während ersterer auf 1.4% abfällt. Für kleinere Probleme ist übrigens auch auf der SR8000 eine Blockgröße von 16 optimal (man erhält dann rund 19% der Spitzenleistung), aber die Rechenleistung fällt für große Probleme dann viel stärker ab. Auf der IBM p690 findet man für eine einzige CPU im Bereich n=100 bis ~5000 eine z. T. deutlich schlechtere Leistung als für Itanium; hier dürften die auf Power4 deutlich größeren Cache Lines von 512 Bytes die Leistung drücken. Im Multiprozessor-Betrieb erhält man für die p690 wilde Sprünge, für die wohl entweder betriebssystembedingte Probleme mit dem Scheduling der Threads oder – für größere Problemstellungen – der doch recht ungleichmäßige Zugriff auf den über die MCMs verteilten L3 Cache verantwortlich sein dürften.

3 Anwendungsnahe Benchmarks

3.1 Gaussian 98

Da die Betriebsanforderungen für das Itanium2-Cluster die Nutzung quantenchemischer Programme mit großem Speicherbedarf vorsehen, wurde bei der Cluster-Beschaffung am LRZ ein typisches Beispiel aus der Quantenchemie in die Leistungsbewertung aufgenommen. Mit am LRZ vorbereiteten ausführbaren Programmen für Gaussian 98 waren an einem Dekan-Molekül ($\mathbf{C_{10}H_{22}}$) eine Dichtefunktional-Strukturoptimierung vorzunehmen und anschließend die Schwingungsfrequenzen zu berechnen. Gaussian 98 kann im shared-memory Paradigma zumindest zeitweise alle vier CPUs eines Itanium2-Knotens nutzen; Messgröße war die Laufzeit des Programms.

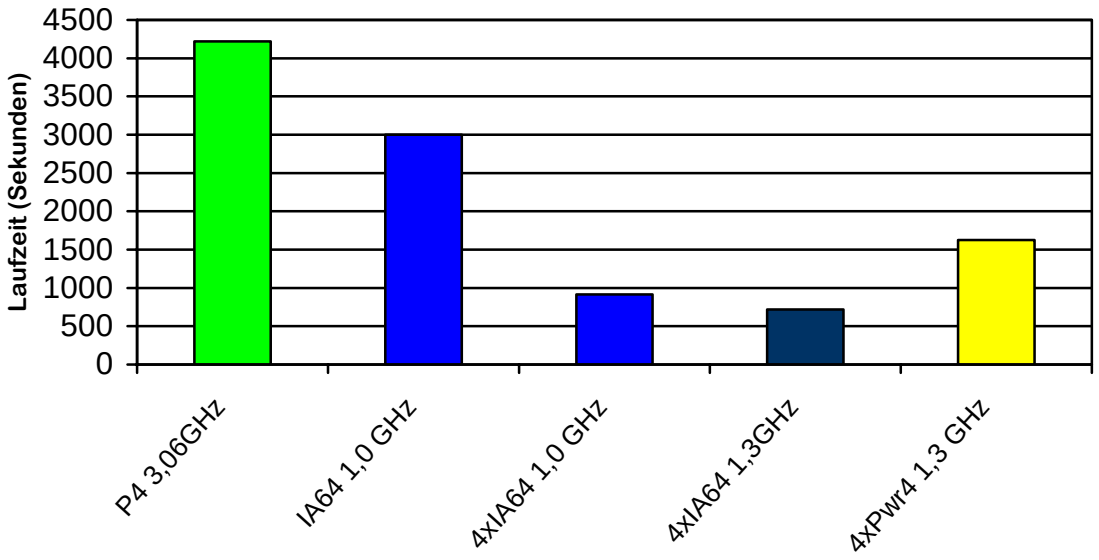


Abbildung 11: Laufzeiten für Strukturberechnung von Dekan mit Gaussian 98

Trotz der deutlich höheren Taktung des Pentium-basierten Systems schneidet eine mit 1.0 GHz getaktete Itanium CPU bei dieser Leistungsmessung rund 30% besser ab; mit einem 4-Wege Knoten reduziert sich die Laufzeit gegenüber der einzelnen Itanium CPU von 3000 auf 916 Sekunden, das entspricht einer parallelen Effizienz von rund 82%. Beim Leistungs-Preisverhältnis liegt trotzdem noch die IA32 Architektur weit vorne; jedoch kann man mit dem Itanium-basierten Knoten nun auch Problemstellungen anpacken, die auf Grund der im Rechenzentrumsbetrieb notwendigen Restriktionen bislang nicht lösbar waren. Des weiteren entnimmt man der Abbildung 11, dass bei einer Erhöhung des Prozessortaktes von 1.0 auf 1.3 GHz, also um 23%, die Rechendauer von 916 auf 720 Sekunden abnimmt; das entspricht einer Erhöhung der effektiven Rechenleistung um 21% und lässt den Schluss zu, dass diese Benchmark-Konfiguration – die durchaus typisch für die zu erwartende Benutzerlast ist – im wesentlichen auf den L2 und L3 Caches operiert. Bestätigt wird dieses Verhalten auch durch im Benutzerbetrieb durchgeführte Statistiken der Hardware-Zähler, die für Gaussian 98 Jobs eine pro-CPU Leistung zwischen 0.8 und 1.0 GFlop/s messen. Das sind bis zu 19% der Spitzenrechenleistung. Ein Vergleich zur VPP ist an dieser Stelle unangebracht, da die Dichtefunktionalmethoden in Gaussian nicht vektorisieren; der Vektorrechner ist für diese Problemklasse nicht geeignet, da sie nicht vektorisierbar implementiert wurde.

3.2 Strömungsmechanik

Für Simulationen aus dem Bereich der Strömungsmechanik werden seit langer Zeit Vektor-Rechner sehr erfolgreich eingesetzt; diese Architektur ist sowohl vom Speicher-Interface als auch vom Programmiermodell her für diese Probleme sehr geeignet. Als erstes Beispiel soll zunächst der am LRZ eingesetzte Benchmark-Code MGLET (Lehrstuhl Prof. Friedrich, TU München) diskutiert werden; dieser Code dient zur direkten numerischen Simulation der Navier-Stokes Gleichungen für inkompressible Strömungen. Für die Propagation in der Zeit kommt hierbei ein leapfrog-Algorithmus zum Einsatz, während die Lösung der Poisson-Gleichung mittels eines „strongly implicit procedure“ (SIP) solvers durchgeführt wird. Gegenüber älteren Versionen dieses Codes, bei dem z. B. Gauss-Seidel Methoden verwendet wurden, zeichnet sich die SIP Implementierung durch eine höhere Genauigkeitsordnung und dadurch auch schnellere Iterationskonvergenz aus. Die Operationsdichte (d. h. die mittlere Zahl der Rechenoperationen pro Speicheroperation) wird durch die Verwendung eines Lösers höherer Ordnung etwas größer, was Cache-basierten Architekturen zugute kommen kann.

Optimierte Varianten des MGLET Codes wurden für die SGI Altix sowie die Hitachi SR8000 vermessen; auf beiden Plattformen wurde der mit 4 Byte Worten programmierte Code durch Angabe von geeigneten Compiler-Schaltern auf 8 Byte (doppelt genaue) Worte umgestellt. Auf der Hitachi SR8000 wurde neben der schwachen Skalierung, bei der die Problemgröße linear mit der Zahl der benutzten CPUs anwächst, auch eine starke Skalierung (d. h. ein Problem konstanter Größe) untersucht.

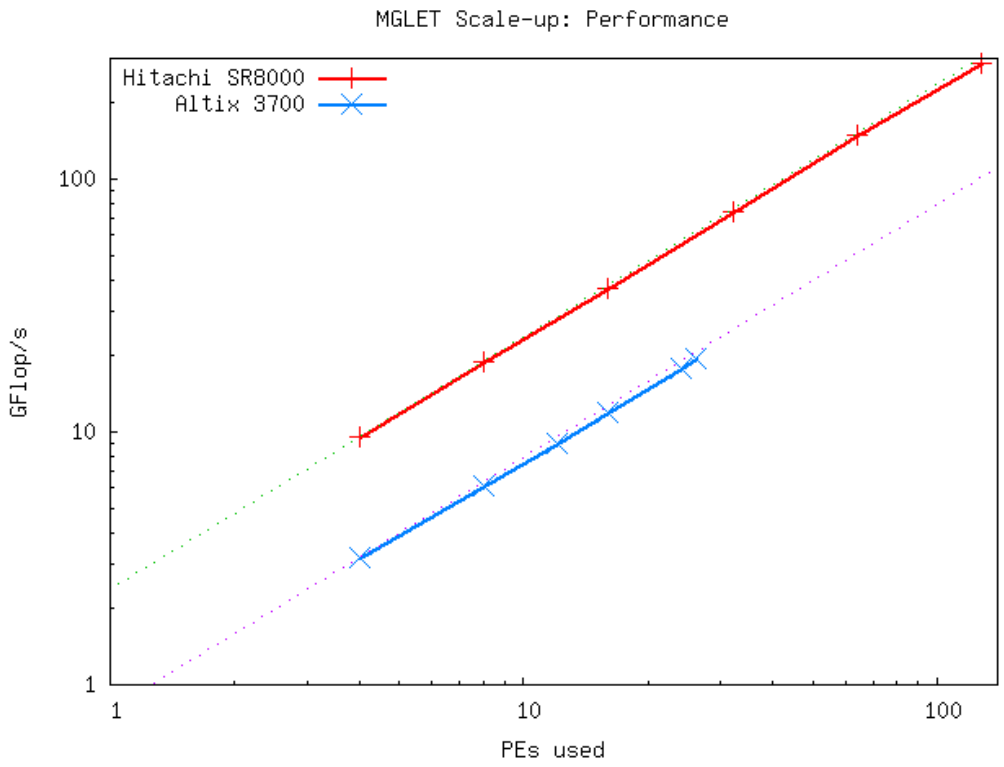


Abbildung 12: Leistungsmessungen für schwache Skalierung MGLET

Die folgende Tabelle zeigt zunächst die schwache Skalierung; hierbei wurde auf 4 Rechenknoten (1 Rechenknoten entspricht hierbei einem 8-Wege Knoten auf der SR8000 bzw. einer Altix CPU) ein Gitter mit 256 * 480 * 400 Punkten gerechnet und die Problemgröße in der ersten Dimension linear mit der Knotenzahl hochskaliert. Auf dem Altix-System ergab sich hierbei ein Speicherbedarf von ca. 2.9 GByte pro CPU, auf der Hitachi von 4.2-4.5 GByte pro Knoten (die genaue Ursache für den höheren Speicherbedarf auf der SR8000 ist nicht bekannt). Für die Leistungsmessung wurden in das Programm eingebaute synthetische Berechnungsformeln verwendet, die – je nach Plattform – bis auf 10% oder besser mit Hardware-Leistungsmessungen konsistent sind. Abbildung 12 zeigt die Skalierungsresultate, mit gepunkteten Linien wird die ideale Skalierung auf der Basis von jeweils 4 PEs angedeutet. Für beide Plattformen

ist man hier noch weit von der Sättigung entfernt: bei der jeweils maximalen Anzahl von MPI Prozessen (26 auf der Altix, 128 auf der SR8000) liegt die parallele Effizienz beider Architekturen bei 93.6%. Die Basisleistung für 4 PEs (ein PE entspricht einer Altix CPU mit einem MPI Prozess, bzw. einem Hitachi 8-Wege Knoten mit einem autoparallel laufenden MPI Prozess) ist für die SR8000 9.53 GFlop/s, für die Altix 3.18 GFlop/s. Auf der Basis eines MPI Prozesses sind also 8 Hitachi SR8000 CPUs leistungsäquivalent zu etwa 3 Itanium CPUs in der Altix ccNUMA Umgebung.

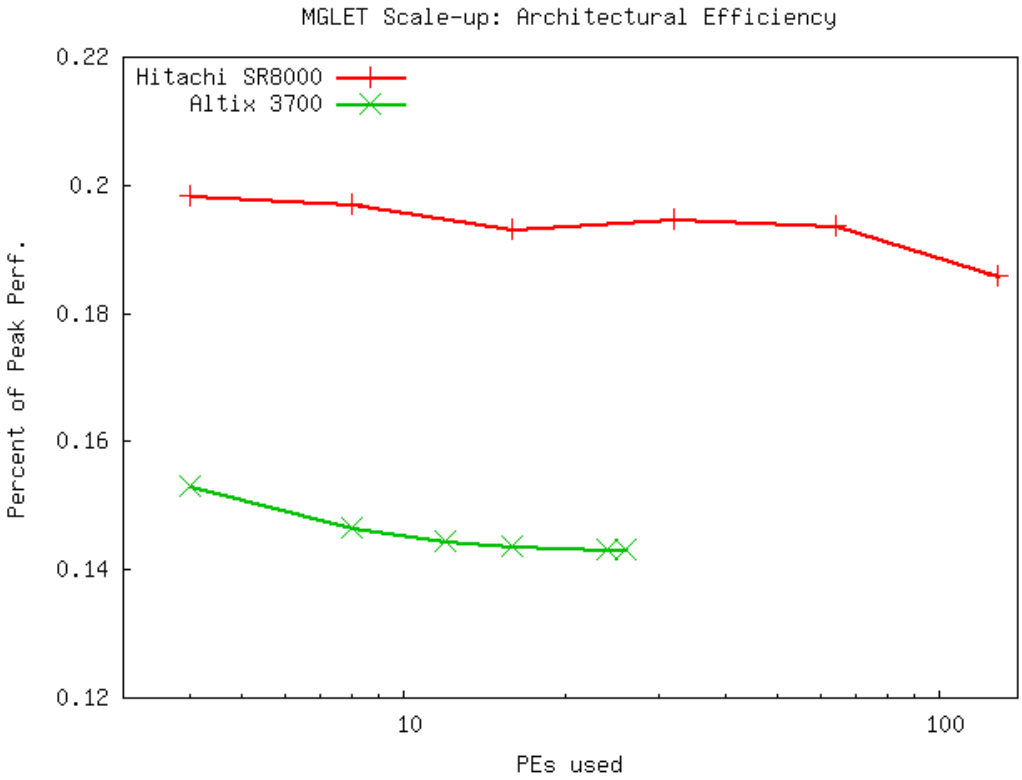


Abbildung 13: Effizienz bezogen auf Spitzenleistung

Als weitere Größe von Interesse soll für MGLET auch die architekturelle Effizienz betrachtet werden, also der Prozentsatz der Spitzenrechenleistung, der bei schwacher Skalierung von dieser Anwendung erzielt werden kann. Abbildung 13 liefert hier den Vergleich zwischen SR8000 und Altix; der Unterschied ist nicht groß, und angesichts des im Vergleich sehr günstigen Preises der Altix (ca. 5,400 US Dollar pro CPU verglichen mit mehr als 80,000 US Dollar für einen SR8000 Knoten zu heutigen Preisen) wird er mehr als wettgemacht. Es ist darüber hinaus anzumerken, dass bei der optimalen Ausnutzung der Itanium Architektur durch den Compiler auch hier noch Spielraum besteht, unterschiedliche Compiler Releases liefern hier durchaus noch deutliche Leistungsunterschiede. Leistungsmessungen am Itanium Myrinet Cluster des LRZ konnten für diese Untersuchung leider nicht mehr durchgeführt werden, es ist jedoch damit zu rechnen, dass hier nur etwa die halbe architekturelle Effizienz erzielt werden kann, da sich vier CPUs statt zwei die voll ausgenutzte Speicherbandbreite von 6.4 GByte/s teilen müssen.

Abschließend sollen noch kurz die auf der SR8000 gewonnenen Resultate für die starke Skalierung betrachtet werden: Hierbei wurde auf 8, 16 und 32 PEs ein Problem der Größe 1280 * 240 * 400 behandelt.

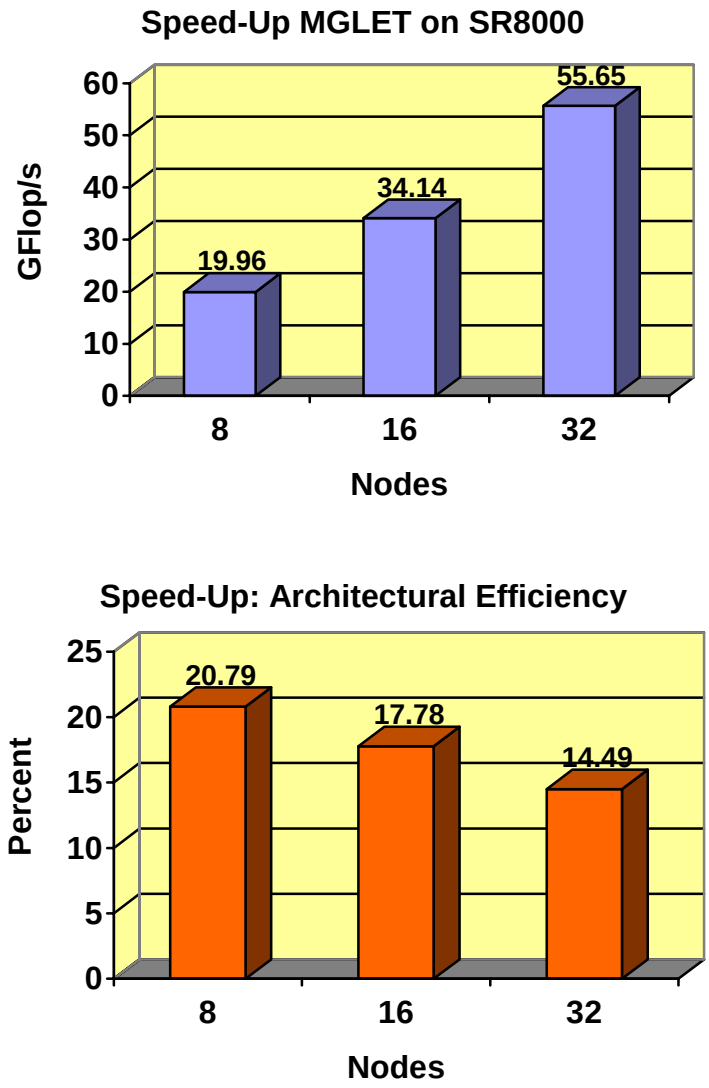


Abbildung 14: Resultate zur starken Skalierung von MGLET auf der SR8000

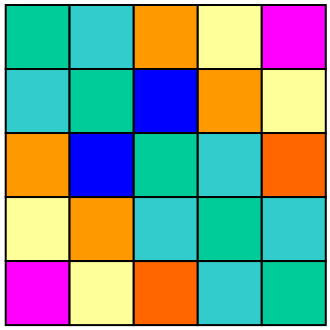
Abbildung 14 zeigt, wie sich die Leistungszahlen sowie auch die architekturelle Effizienz in diesem Fall verhalten; es steht zu vermuten, dass sich für dieses fest gewählte Problem über 128 Knoten hinaus kein wesentlicher Speed-Up auf der SR8000 erzielen lässt.

4 Bibliotheken

Ganz wesentlich mitverantwortlich für einen erfolgreichen Einsatz einer Prozessorarchitektur im wissenschaftlichen Hochleistungsrechnen ist die Verfügbarkeit von hoch optimierten Standardbibliotheken; in erster Linie sind hier natürlich die Bibliotheken zur linearen Algebra (BLAS, LAPACK) zu nennen, aber auch weitere Funktionalität, wie Fourier-Transformationen, Löser für dünn besetzte Gleichungssysteme usw. Da ein ausreichend leistungsfähiger und trotzdem kostengünstiger Ersatz für einen Vektor-Knoten nur in einem Multiprozessor mit mindestens vier, eher acht CPUs bestehen kann, sollten die Bibliotheks-Implementierungen so weitgehend wie möglich shared-memory parallel laufen, wenn dies gewünscht ist; für den Aufruf aus Einzel-Threads von OpenMP parallelen Programmen sollte der serielle Aufruf „thread-safe“ sein, d. h. es darf keine Möglichkeit existieren, durch Seiteneffekte aus einem parallel durchgeführten Bibliotheks-Aufruf falsche Ergebnisse zu erhalten. In diesem Bericht soll auf zwei Funktionsbereiche etwas ausführlicher eingegangen werden: Reell-symmetrische Eigenwertprobleme und komplexe Fourier-Transformationen.

4.1 Eigenwert-Löser

Wir betrachten reelle, symmetrische Matrizen A und B, wie etwa durch Abbildung 15 illustriert. Gesucht sind die reellen Werte λ („Eigenwerte“), für die das lineare Gleichungssystem



$$(A - \lambda B)x = 0$$

Abbildung 15: Symmetrie einer Matrix

eine nicht-triviale Lösung besitzt, sowie auch die zu jedem Eigenwert gehörigen Lösungen („Eigenvektoren“). Probleme dieser Art sind z. B. nach der Diskretisierung von partiellen Differentialgleichungen – etwa durch ein Variationsverfahren – zu lösen. In der LAPACK Bibliothek gibt es ein Standard-Verfahren zur Lösung dieses Problems (DSYGV), das einen zu n^3 proportionalen Zeitaufwand benötigt, wobei n die Dimension der Matrizen bezeichnet; neuere, ebenfalls in LAPACK verfügbare Methoden (DSYGVD) haben auf Grund eines Divide-and-Conquer Algorithmus eine geringere Komplexitäts-Ordnung, sie sollten daher mit vergleichbarem Zeitaufwand erheblich größere Probleme zu lösen gestatten. Abbildung 16 zeigt für verschiedene Architekturen die Laufzeiten als Funktion der Problemgröße. Exemplarisch ist für die Hitachi SR8000 der Fit für die Komplexitätsordnung durchgeführt worden; der erwartete Exponent 3 kommt hier sehr genau heraus. Am besten schneidet im Vergleich nach wie vor die VPP700 ab, für die im Übrigen die DSYGV Routine ein besseres Ergebnis liefert als die DVGSG2 Routine aus der SSL-II Bibliothek.

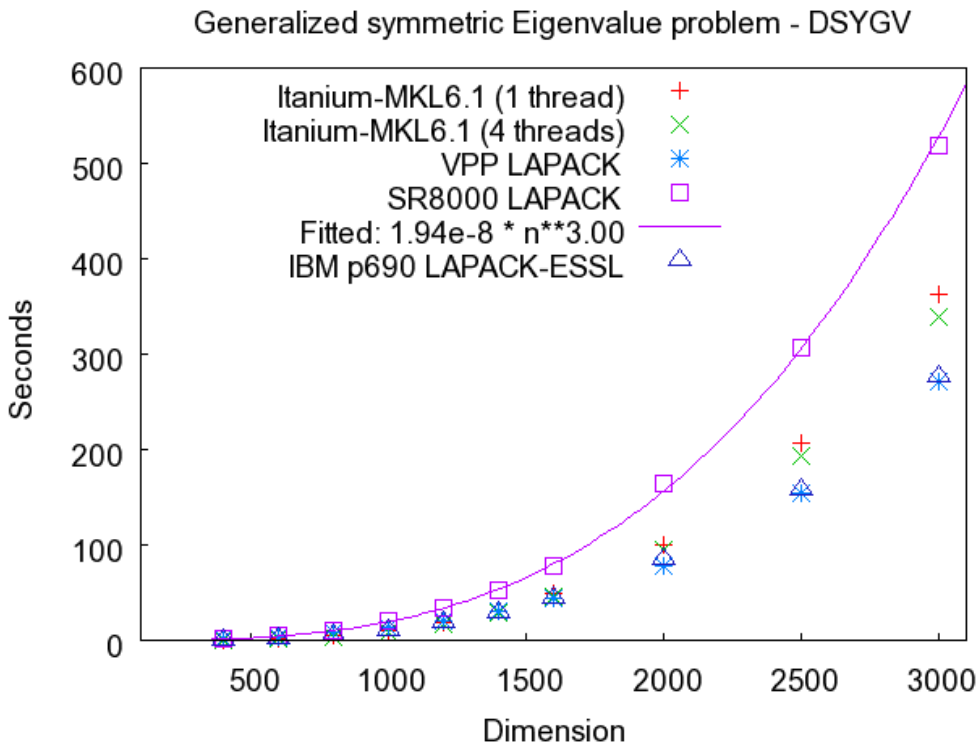


Abbildung 16: Standardverfahren für Eigenwertproblem

Des weiteren zeigt sich, dass auf der Itanium Architektur der Standard-Löser durch Multi-Threading nicht wesentlich an Leistung gewinnt. Dennoch gibt es Problemstellungen, für die nach wie vor der Standard-Löser eingesetzt wird, weil die numerische Stabilität bei der Berechnung der Eigenvektoren mitunter besser als die des Divide-and-Conquer Lösers ist.

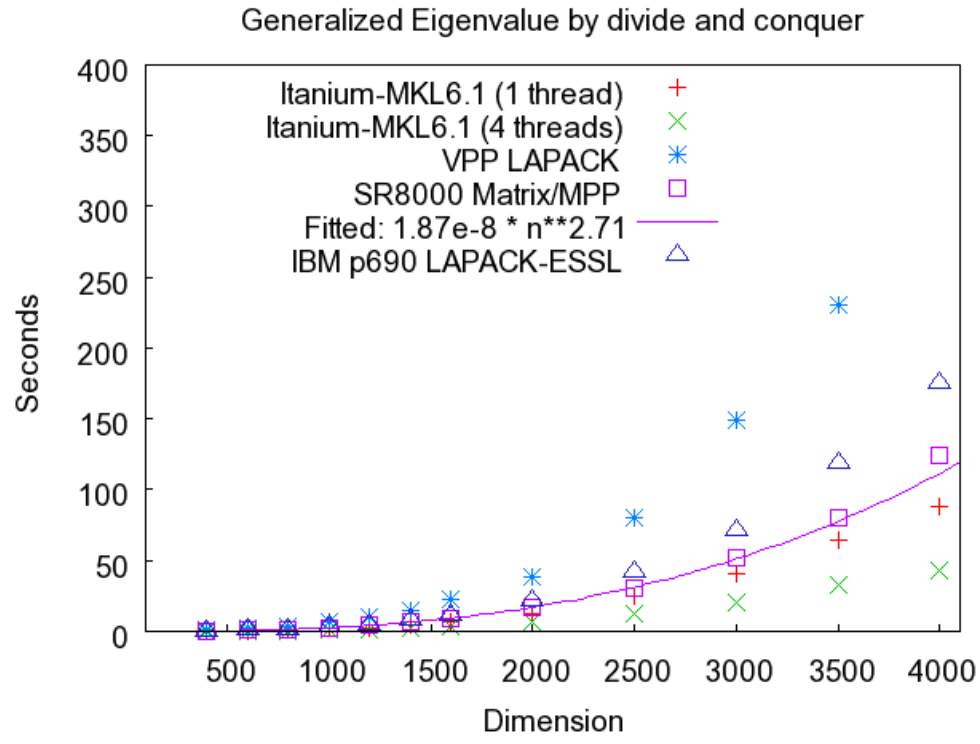


Abbildung 17: Divide-and-Conquer Löser

Insbesondere ist die MKL thread-safe ausgelegt, sodass eine effiziente Implementierung durch parallele Aufrufe des Solvers – für nicht zu große Dimensionierung der Probleme – möglich ist. Betrachten wir nun die Leistungszahlen beim Divide-and-Conquer Löser, die durch Abbildung 17 illustriert sind: Obwohl sich die VPP Zahlen gegenüber dem Standard-Verfahren noch einmal verbessern, liegt der Vektor-Rechner gegenüber den anderen Architekturen deutlich hinten; es ist anzumerken, dass die Version 3 von LAPACK, die hier zum Einsatz kam, nicht mehr gezielt für die VPP optimiert wurde; generell schlägt natürlich auch das Einfrieren der Software-Entwicklung (SSL-II) für die Vektor-Architektur zu Buche. Auf der Hitachi SR8000 schneidet hier die proprietäre Matrix/MPP Routine HDES3M besser ab als die LAPACK; wie durch die Fit-Funktion angezeigt, kommt offenbar auch hier ein Divide-and-Conquer Verfahren zum Einsatz. Dennoch gewinnt schon eine einzige Itanium CPU deutlich gegen den Hitachi 8-Wege Knoten, und Multi-Threading bringt bei diesem Verfahren auf Itanium deutlich mehr parallele Effizienz (etwa 50% auf 4 CPUs) als beim Standardverfahren: damit erweist sich ein Itanium Multiprozessor in diesem Bereich als ein mehr als vollwertiger Ersatz für einen Vektor-Prozessor.

4.2 Fourier-Transformation

Für viele Anwendungen aus der Chemie, Physik und auch der Bildverarbeitung und dem Ingenieurwesen ist die Verfügbarkeit von effizienten Routinen zur Fourier-Transformation (FT) von entscheidender Bedeutung. Obwohl hier mit der Einführung der schnellen FT (FFT) die Komplexität von $O(n^2)$ auf $5 * n * \log_2(n)$ heruntergedrückt wird, gibt es dennoch noch einen weiten Spielraum für Code-Optimierung; der FFT Algorithmus enthält scatter/gather Operationen, die auf die architektur-spezifischen Randbedingungen zugeschnitten werden müssen[1]. Betrachten wir zunächst die eindimensionale FT mit 8 Byte (doppelt genauen) Worten, für die A einen Überblick über die Architekturen zeigt. Gemessen wurden hier nur Vektoren mit Längen in Potenzen von 2; nicht auf allen Architekturen standen mixed-radix Transformationen zur Verfügung (insbesondere nicht auf der VPP mit guter Vektorleistung).

[1] Für cache-orientierte Architekturen gibt es hierfür auch automatisierte Verfahren, wie sie etwa in FFTW (http://www.fftw.org) implementiert sind. Hier werden wir jedoch in erster Linie die proprietären Bibliotheken diskutieren.

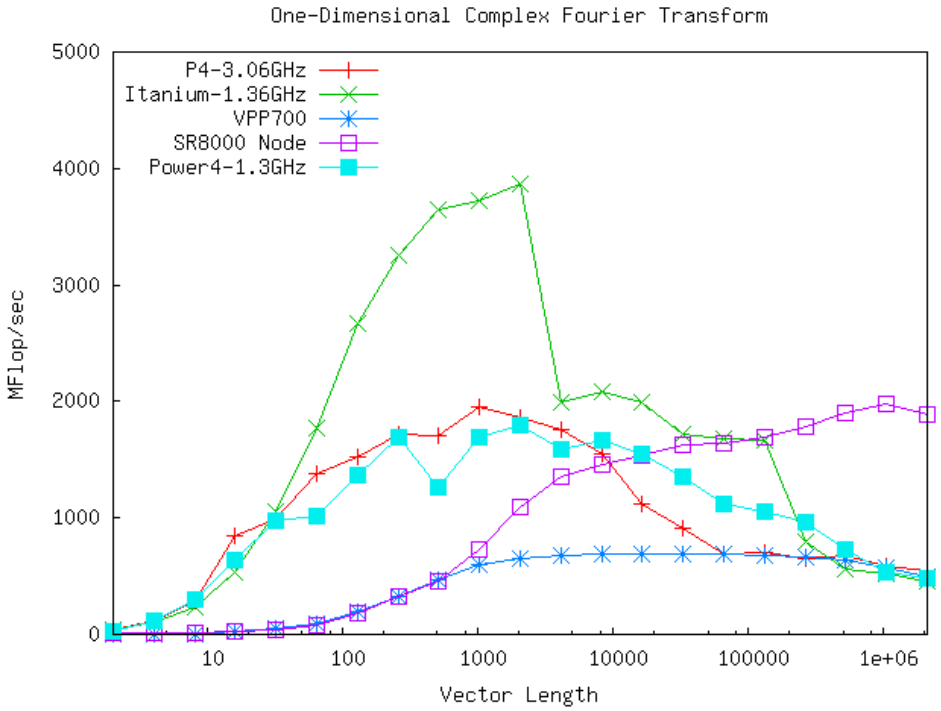


Abbildung 18: Eindimensionale FFT auf verschiedenen Plattformen

Sehen wir einmal von kurzen Vektorlängen ab, für die eine gute Rechenleistung auf allen Plattformen nur durch simultane Transformation mehrerer Vektoren mit einem einzigen Unterprogrammaufruf erzielbar ist, so stellt man fest, dass in weiten Bereichen die Itanium CPU alle anderen Architekturen schlägt: aus dem L2 Cache erzielt sie mit 3.8 GFlop/s über 70% der Spitzenrechenleistung, und im L3 Cache immer noch etwa 2 GFlop/s.

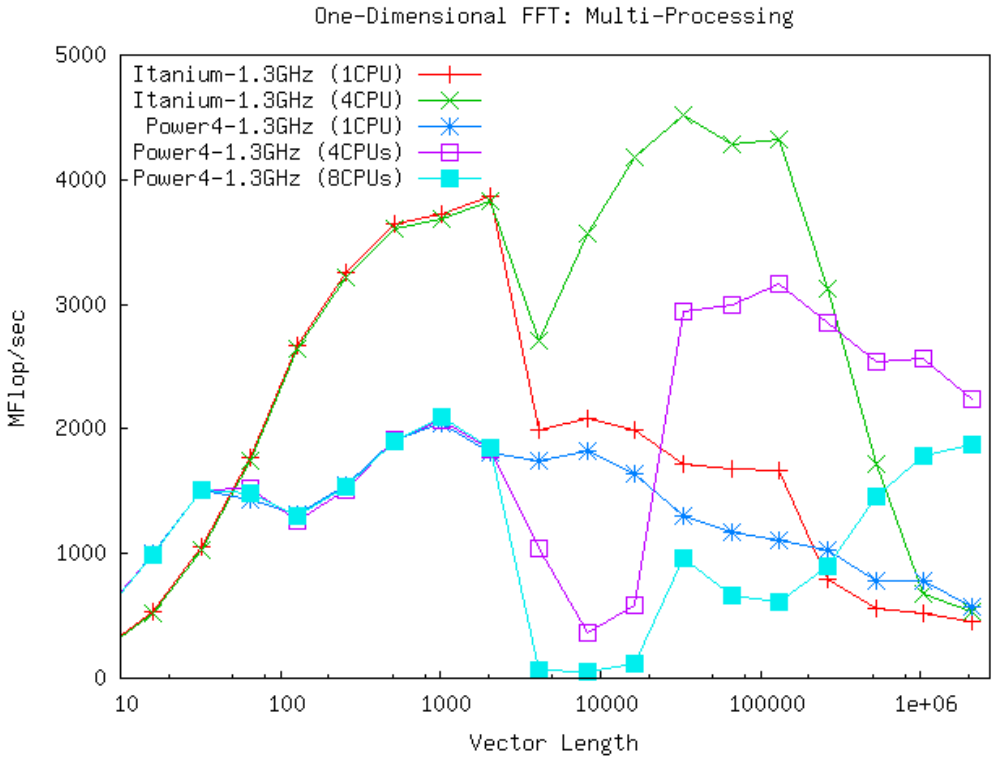


Abbildung 19: Multiprozessor-Leistung für FFT

Wie zu erwarten, siegt bei Vektorlängen oberhalb ~100000 der Hitachi SR8000 Knoten, alle anderen Architekturen – einschließlich der VPP! – erzielen für große Vektorlängen vergleichbare Rechenleistung. Zur Beurteilung der Multiprozessorleistung dient Abbildung 19. Hier zeigen sich sowohl für die Itanium2 als auch die Power4 deutliche Einschränkungen. Die Bibliothek verwendet offenbar Versionierung, um den für kurze Vektoren dominierenden Thread startup zu umgehen, jedoch ist diese nicht optimal implementiert: der Übergang von der seriellen zur parallelen Ausführung ist offenbar zu früh angesetzt oder erfordert zusätzliche, nicht explizit dokumentierte Einstellungen. Experimente mit Veränderung am OpenMP scheduling brachten jedenfalls keine Verbesserungen. Auf der Itanium Plattform erhält man für Vektorlängen zwischen 8192 und 262144 eine parallele Effizienz von etwas mehr als 50% auf 4 CPUs, für größere Probleme verhindert das Memory Interface eine gute Skalierung. Die Power4 Architektur zeigt sich durchwachsen: die Qualität der Versionierung ist noch schlechter als bei Itanium, danach allerdings erzielt man bei Nutzung von 4 CPUs Effizienzen von bis zu 70%, und zwar auch bei großen Problemen. Allerdings werden hier die großen L3 Caches sicher noch nicht ausgeschöpft, und die Skalierbarkeit hört bei 4 CPUs auch schon auf; für 8 CPUs sinkt die Rechenleistung wieder ab, stellenweise sogar unter die seriellen Werte. Auf beiden Architekturen gibt es hier sicher noch Spielraum für Verbesserungen, sowohl auf der Ebene der Bibliotheks-Implementierung, als auch bei der Behandlung der Speicher- und CPU Zuordnung („memory affinity“) durch die Betriebssysteme.

Wenden wir uns abschließend der dreidimensionalen Fourier-Transformation zu. Hier wurden Probleme mit einheitlicher Gesamtgröße betrachten, und für diese eine Reihe von Zerlegungen in Zweierpotenzen mit einer Länge von mindestens 8 betrachtet. So erhält man z. B. für eine Gesamtgröße von 2048 die möglichen Zerlegungen 8 * 8 * 32 bzw. 8 * 16 * 16. Entsprechend sind zu jeder Problemgröße mehrere Leistungszahlen angegeben.

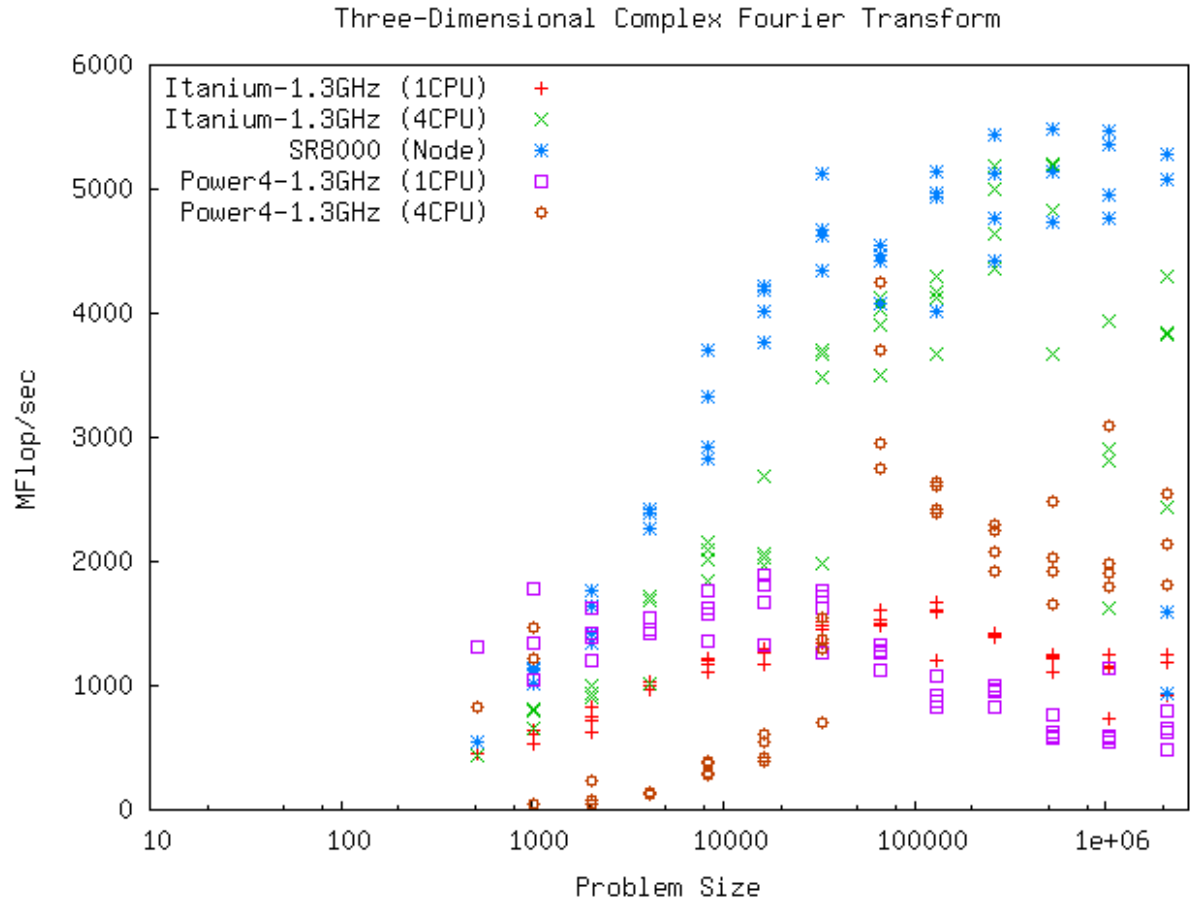


Abbildung 20: Dreidimensionale Fourier-Transformation auf Multiprozessoren

Am besten schneidet für diese Problemklasse der Hitachi Knoten ab; er erzielt bis zu 45% der Spitzenrechenleistung. Für alle Plattformen beobachtet man für bestimmte Problemgrößen deutliche Leistungsstreuungen, so z. B. bei der SR8000 für die größten Probleme (blaue Sterne in Abbildung 20); hier spielt wohl auch das ungünstige Ansprechen der Memory-Bänke eine Rolle, das aber durch geeignetes Überallozieren von Speicherplatz vermieden werden kann. Für die Itanium Architektur (aus der MKL wurde das für hochdimensionale FT's verfügbare DFTI verwendet) zeigt sich ein gegenüber dem eindimensionalen Fall deutlich besseres Skalierungsverhalten für ausreichend große Probleme. Die Absolutleistung eines 4-Wege Itanium reicht schon durchaus an die des Hitachi Knotens heran; man erhält etwa 25% der Spitzenleistung. Für die Power4 Architektur erhält man erst für sehr große Probleme halbwegs vernünftige parallele Skalierung; im hier betrachteten Bereich kommt man über 10-15% der Spitzenleistung nur sehr selten hinaus. Rechnungen mit 8 und 16 Threads – hier nicht dargestellt – zeigen, dass sich die mininale Problemgröße, ab der die shared memory Parallelisierung greift, sehr rasch zu immer größeren Werten verschiebt.

5 Zusammenfassende Bewertung

Nach der aus Sicht des wissenschaftlichen Hochleistungsrechnens unbrauchbaren ersten Inkarnation der Itanium Architektur lässt sich konstatieren: Itanium2 ist nunmehr eine für HPC geeignete Architektur. Die wenigen noch ausstehenden Einwände wären:

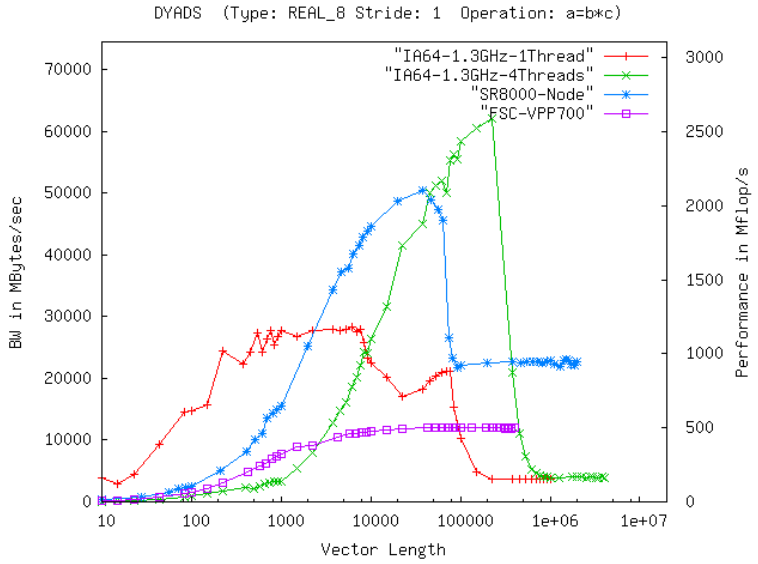
1. Im Bereich der Compiler und Bibliotheks-Software dürfte es noch Spielraum für weitere Optimierung geben; auch die parallele Skalierbarkeit der gegenwärtig eingesetzten Linux Betriebssystemkerne lässt an manchen Stellen noch Probleme offen, die jedoch mit der bald anstehenden Einführung des 2.6 Kernels in den Produktionsbetrieb weitgehend gelöst sein dürften und damit den Betrieb auch sehr großer shared-memory Systeme mit 512 und mehr CPUs ermöglichen.
2. Für den Einsatz im „Grand Challenge Computing“ hat sich aus Sicht des LRZ die Verwendung von mäßig großen shared Memory Multiprozessoren als Äquivalent zu leistungsfähigen Vektor-CPUs bewährt; hier ist jedoch auf ein knotenintern skalierbares Speicher-Interface zu achten. Von allen bekannten Itanium Multiprozessor Implementierungen erfüllt die SGI Altix diese Forderung am besten; die Standard-Chipsätze von Intel erfüllen die Forderung nur ungenügend.
3. Bei der zukünftigen Hochtaktung der Systemfrequenz und Integration auf Multi-Core CPUs wird leider die Frequenz der Speicher-Anbindung nach wie vor nicht gut mithalten. Während in den nächsten zwei Jahren der CPU Takt von 1.3 auf vermutlich 2.5 GHz etwa verdoppelt wird und sich die Zahl der CPUs verdoppelt, beschränkt sich die Verbesserung der Speicherbandbreite auf eine Erhöhung um 65%.
4. Es ist noch nicht ganz klar, in wie weit die von AMD eingeführte x86_64 Architektur (für die Intel inzwischen ebenfalls eine Implementierung besitzt) vergleichbare Leistung zu evtl. noch günstigerem Leistungs-Preisverhältnis erbringen kann. Eine Untersuchung über dieses Thema wird Gegenstand eines weiteren Berichts sein.

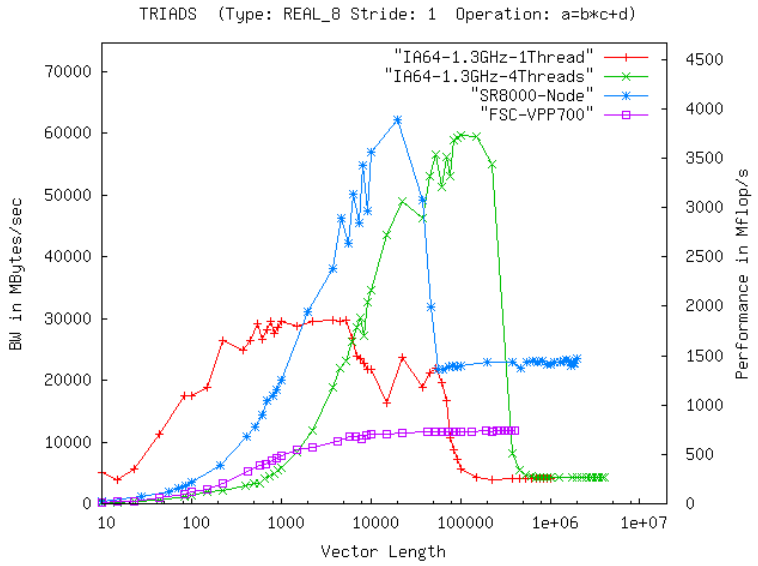
Dennoch überwiegen aus Sicht des LRZ die Vorteile der Architektur deren Nachteile erheblich

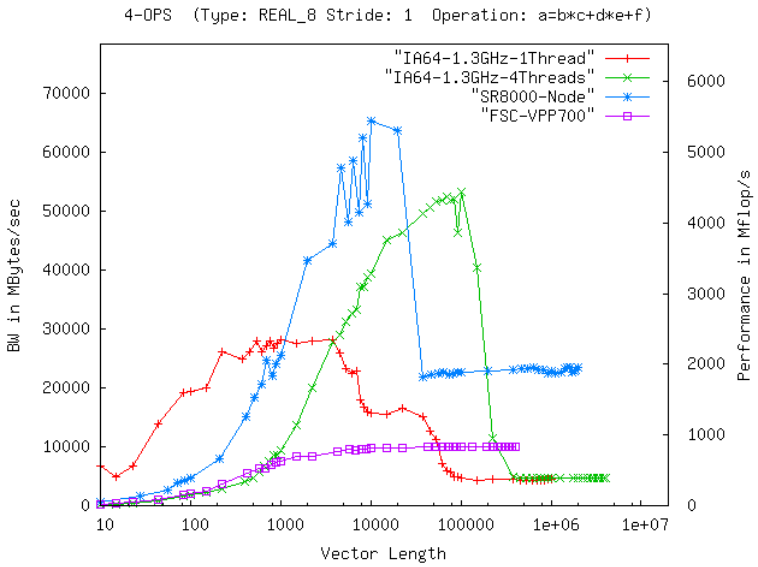
1. Die großen (und demnächst noch anwachsenden) Caches des Itanium gestatten die sehr effiziente Bearbeitung vieler Problemklassen, die nicht extrem speicherintensiv sind. Für den sogenannten Durchsatz-Betrieb, der sowohl quantitativ als auch qualitativ bislang nicht durchführbare Parameter-Studien gestattet, ist das von unschätzbarem Wert; durch die großen Caches wird ein Durchsatz erzielt, der von der x86_64 Architektur bislang und vermutlich noch auf längere Sicht nicht erreicht werden kann.
2. Für eine ganze Reihe von Standardproblemen schlägt die MKL von Intel auf der Itanium Architektur fast alle Konkurrenten, oder ist zumindest vergleichbar gut.
3. Eine skalierbares Speicher-Interface vorausgesetzt, laufen auch vektor-orientierte Applikation skalierbar und mit ausreichender architektureller Effizienz
4. Sieht man von der IA32 Architektur und ihren Derivaten ab, ist das Leistungs-Preis Verhältnis für diese Architektur ausgezeichnet. Für einen 16-Wege Mid-Range Server, der im Sinne des oben Gesagten als „drop-in“ für eine klassische Vektor-CPU dienen kann, dürfte es schätzungsweise (zur Zeit) etwa 30-40% besser sein als das des nächsten Konkurrenten.
5. Es ist zu hoffen, dass die im Vergleich zur IA32 Architektur deutlich niedrigere Taktung sich längerfristig auch in einem wesentlich günstigeren Energieverbrauch niederschlägt. Für große Installationen macht dies einen beträchtlichen Teil der Gesamtkosten aus.

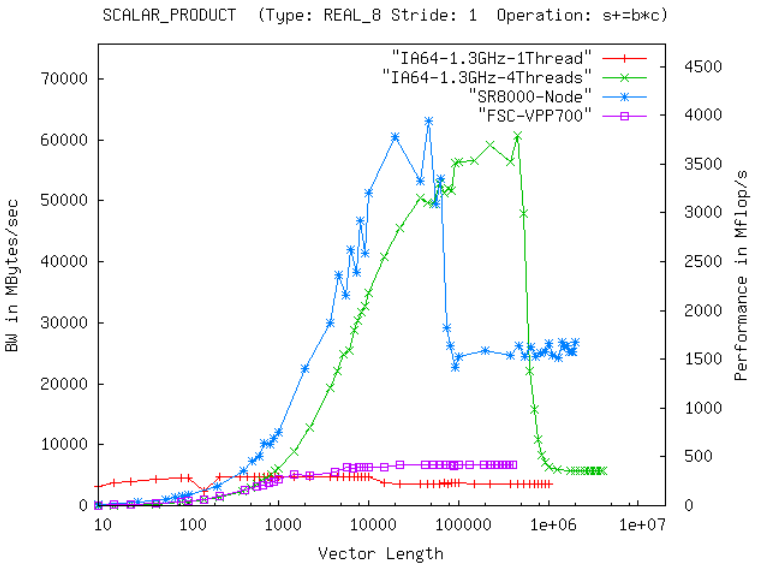
Die VPP700 am LRZ ist im Übrigen nach wie vor – trotz ihrer für Ende 2004 geplanten Abschaltung – ein vollständig ausgelastetes System; obwohl mehr als sieben Jahre in Betrieb, ist die Vektor-CPU durchaus nicht um den Faktor von mehr als 20 hinter die aktuelle Technologie zurückgefallen, den man auf Grund des Moore’schen Gesetzes naiv erwarten würde.

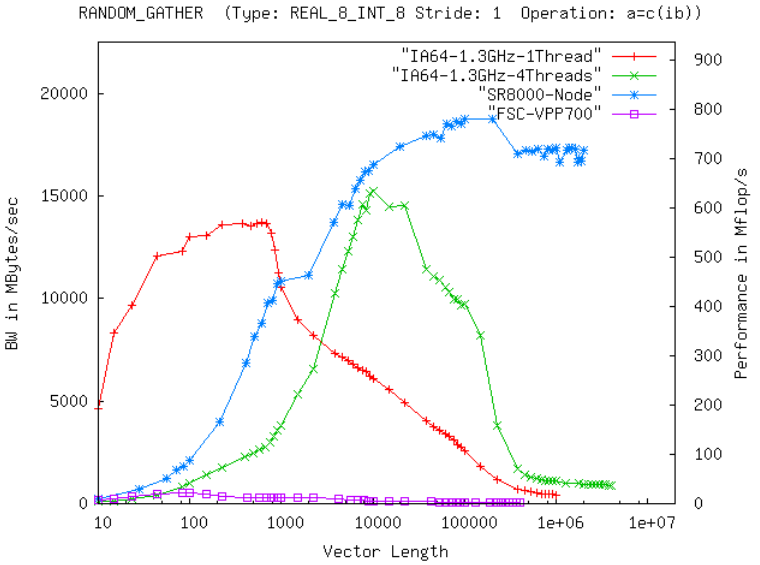
Anhang 1 Synthetische Test-Schleifen (Rinf)

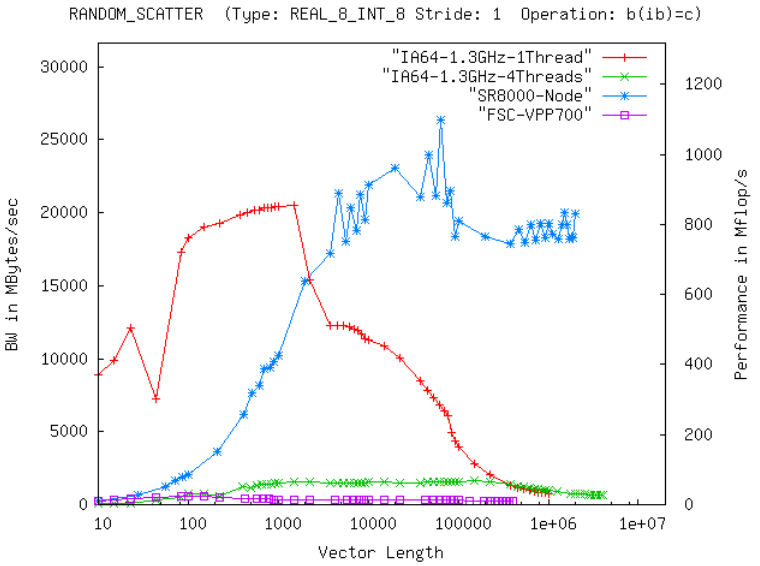


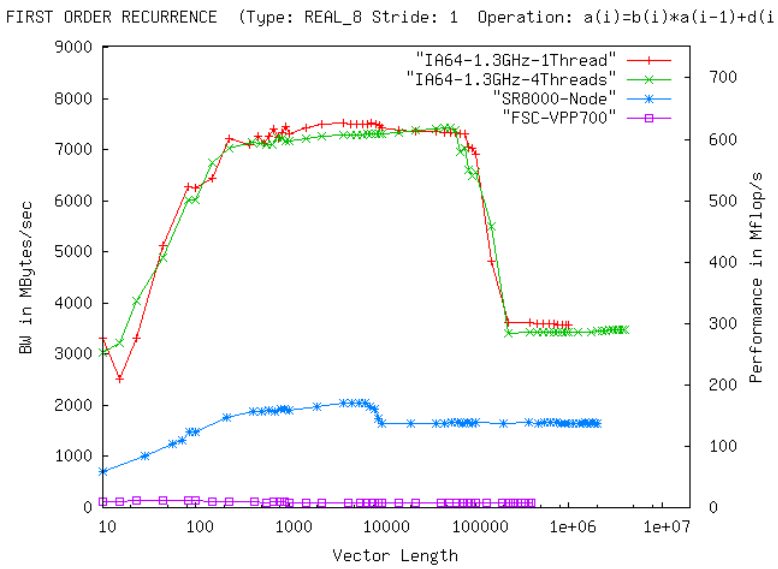


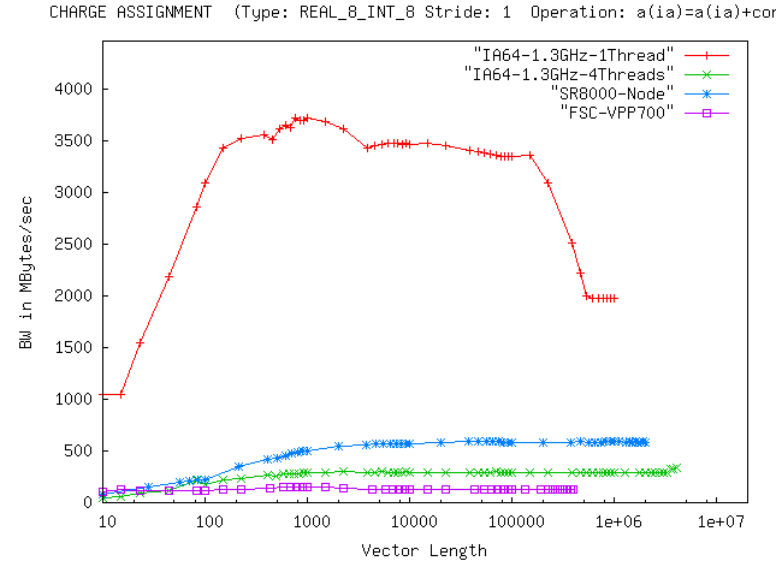


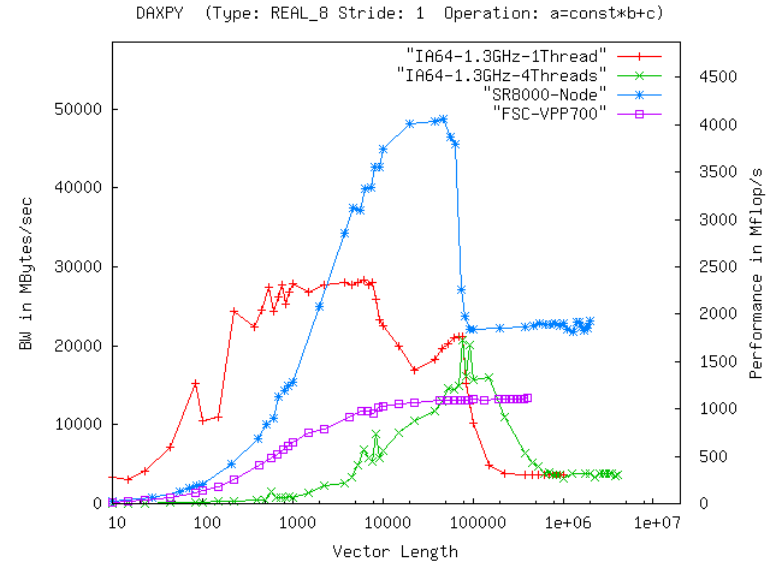

SCALAR_PRODUCT (Type: REAL_8 Stride: 1 Operation: s+=b*c)
"IA64-1.3GHz-1Thread"
"IA64-1.3GHz-4Threads"
"SR8000-Node"
"FSC-VPP700"
BW in MBytes/sec
Performance in Mflop/s
Vector Length
0
10000
20000
30000
40000
50000
60000
70000
0
500
1000
1500
2000
2500
3000
3500
4000
4500
10
100
1000
10000
100000
1e+06
1e+07

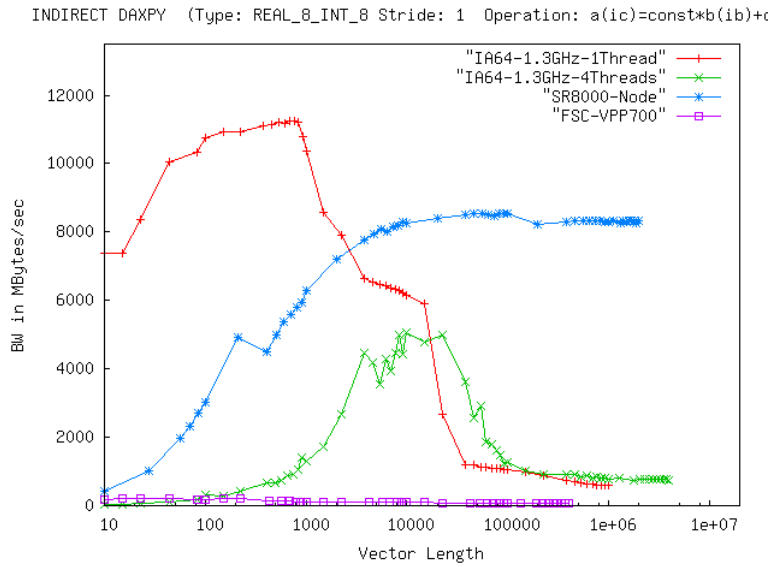

RANDOM_GATHER (Type: REAL_8_INT_8 Stride: 1 Operation: a=c(ib))
"IA64-1.3GHz-1Thread"
"IA64-1.3GHz-4Threads"
"SR8000-Node"
"FSC-VPP700"
BW in MBytes/sec
Performance in Mflop/s
Vector Length
0
5000
10000
15000
20000
0
100
200
300
400
500
600
700
800
900
10
100
1000
10000
100000
1e+06
1e+07

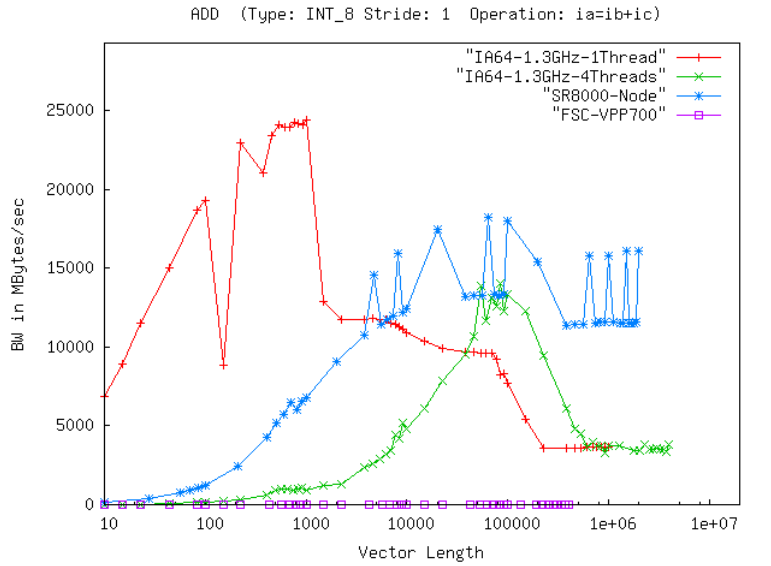

RANDOM_SCATTER (Type: REAL_8_INT_8 Stride: 1 Operation: b(ib)=c)
"IA64-1.3GHz-1Thread"
"IA64-1.3GHz-4Threads"
"SR8000-Node"
"FSC-VPP700"
BW in MBytes/sec
Performance in Mflop/s
Vector Length
0
5000
10000
15000
20000
25000
30000
0
200
400
600
800
1000
1200
10
100
1000
10000
100000
1e+06
1e+07

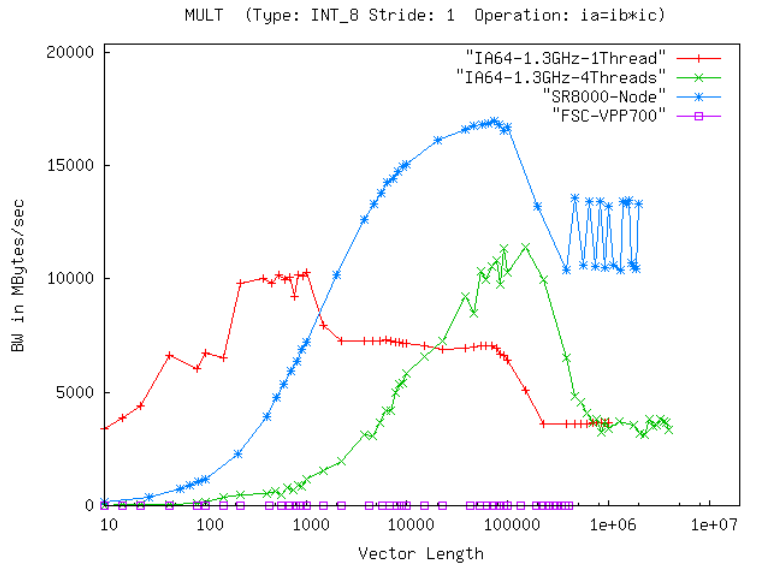

FIRST ORDER RECURRENCE (Type: REAL_8 Stride: 1 Operation: a(i)=b(i)*a(i-1)+d(i
"IA64-1.3GHz-1Thread"
"IA64-1.3GHz-4Threads"
"SR8000-Node"
"FSC-VPP700"
BW in MBytes/sec
Performance in Mflop/s
Vector Length

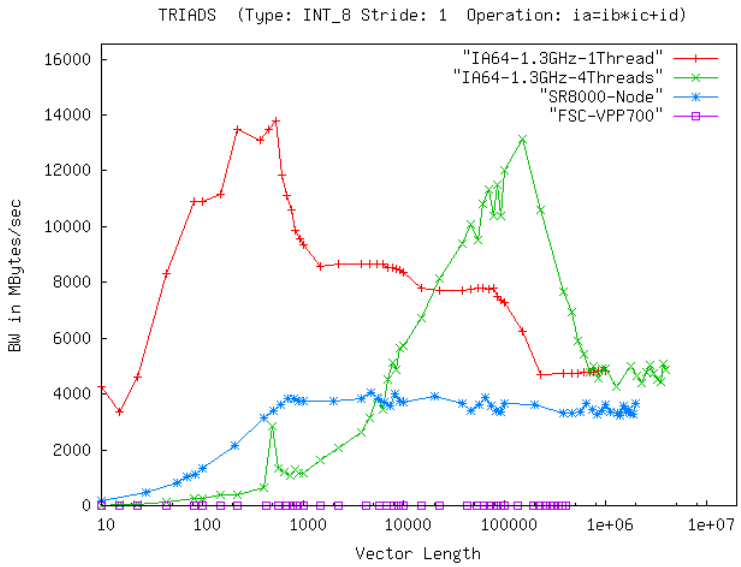

CHARGE ASSIGNMENT (Type: REAL_8_INT_8 Stride: 1 Operation: a(ia)=a(ia)+cor
"IA64-1.3GHz-1Thread"
"IA64-1.3GHz-4Threads"
"SR8000-Node"
"FSC-VPP700"
BW in MBytes/sec
Vector Length

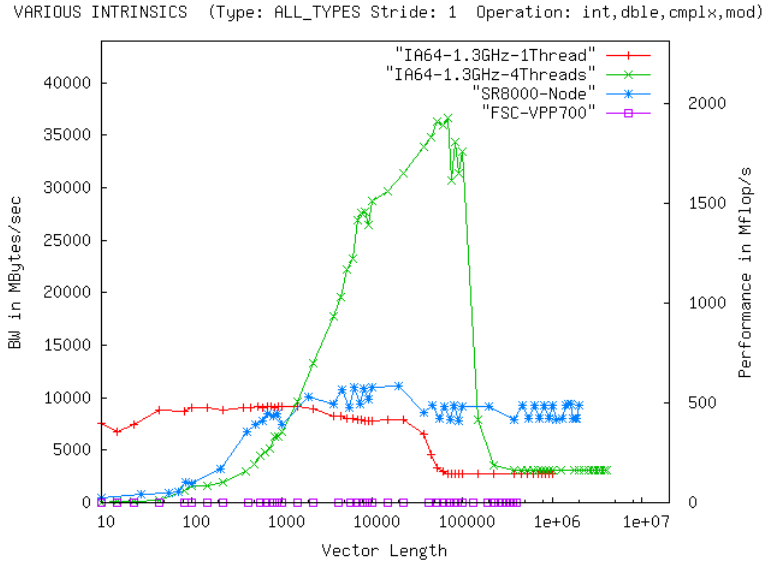

DAXPY (Type: REAL_8 Stride: 1 Operation: a=const*b+c)
"IA64-1.3GHz-1Thread"
"IA64-1.3GHz-4Threads"
"SR8000-Node"
"FSC-VPP700"
BW in MBytes/sec
Performance in Mflop/s
Vector Length

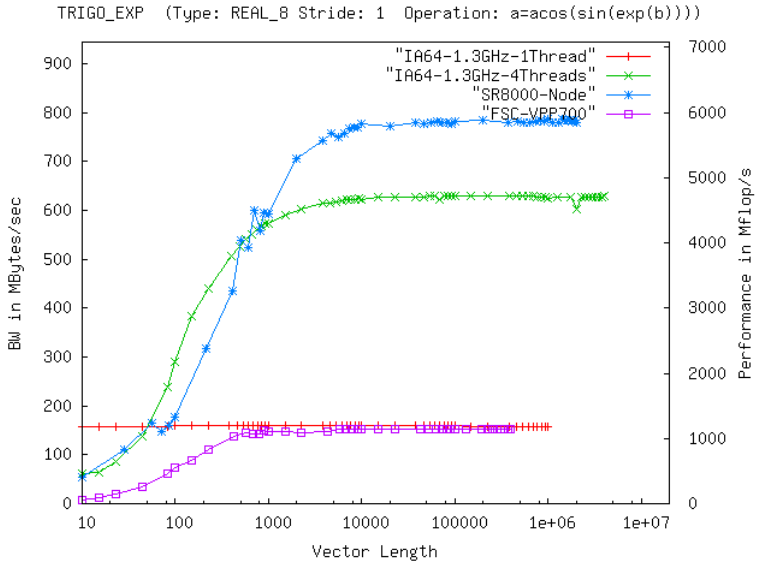

INDIRECT DAXPY (Type: REAL_8_INT_8 Stride: 1 Operation: a(ic)=const*b(ib)+c
"IA64-1.3GHz-1Thread"
"IA64-1.3GHz-4Threads"
"SR8000-Node"
"FSC-VPP700"
BW in MBytes/sec
Vector Length
0
2000
4000
6000
8000
10000
12000
10
100
1000
10000
100000
1e+06
1e+07

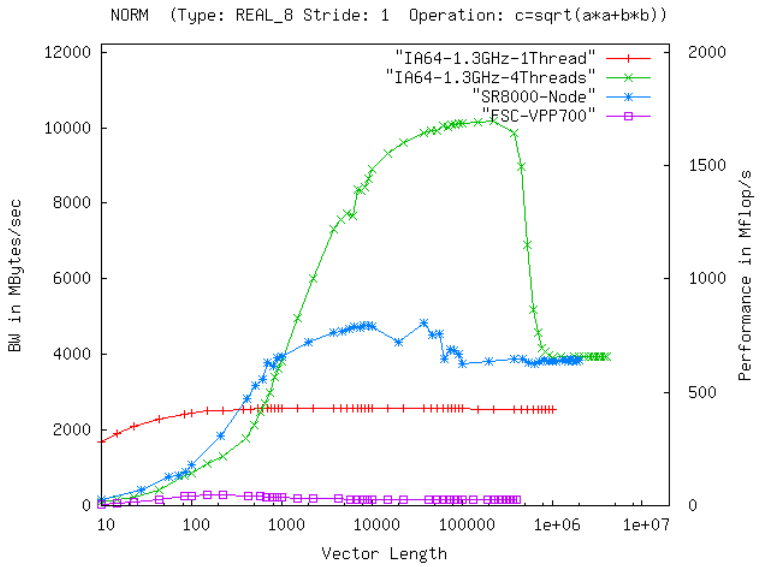

ADD (Type: INT_8 Stride: 1 Operation: ia=ib+ic)
"IA64-1.3GHz-1Thread"
"IA64-1.3GHz-4Threads"
"SR8000-Node"
"FSC-VPP700"
BW in MBytes/sec
Vector Length
0
5000
10000
15000
20000
25000
10
100
1000
10000
100000
1e+06
1e+07

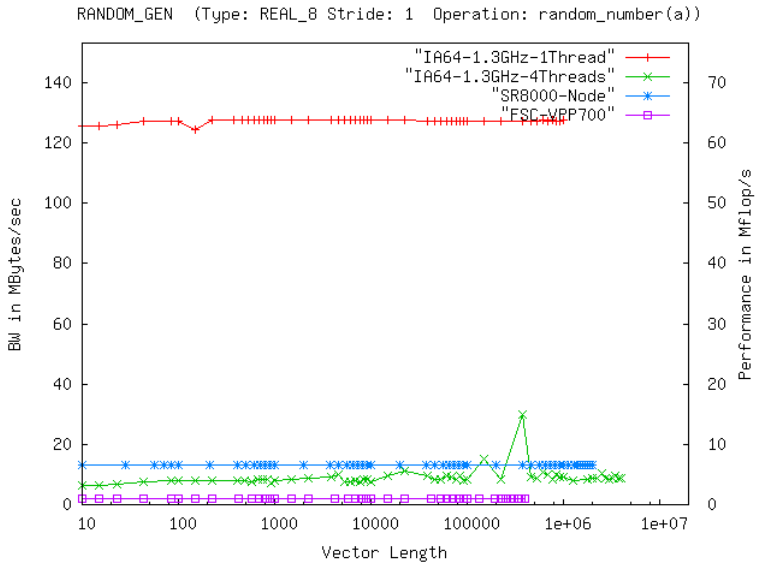

MULT (Type: INT_8 Stride: 1 Operation: ia=ib*ic)
"IA64-1.3GHz-1Thread"
"IA64-1.3GHz-4Threads"
"SR8000-Node"
"FSC-VPP700"
BW in MBytes/sec
Vector Length
0
5000
10000
15000
20000
10
100
1000
10000
100000
1e+06
1e+07

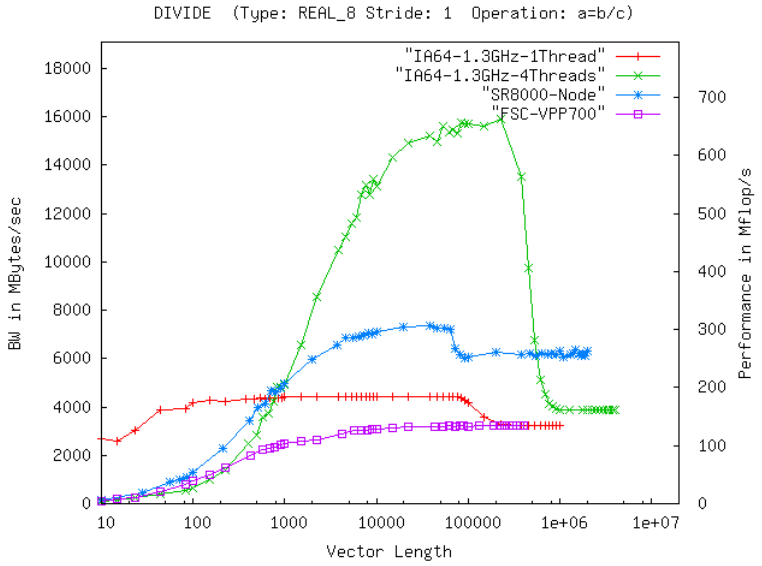

TRIADS (Type: INT_8 Stride: 1 Operation: ia=ib*ic+id)
"IA64-1.3GHz-1Thread"
"IA64-1.3GHz-4Threads"
"SR8000-Node"
"FSC-VPP700"
BW in MBytes/sec
Vector Length
0
2000
4000
6000
8000
10000
12000
14000
16000
10
100
1000
10000
100000
1e+06
1e+07

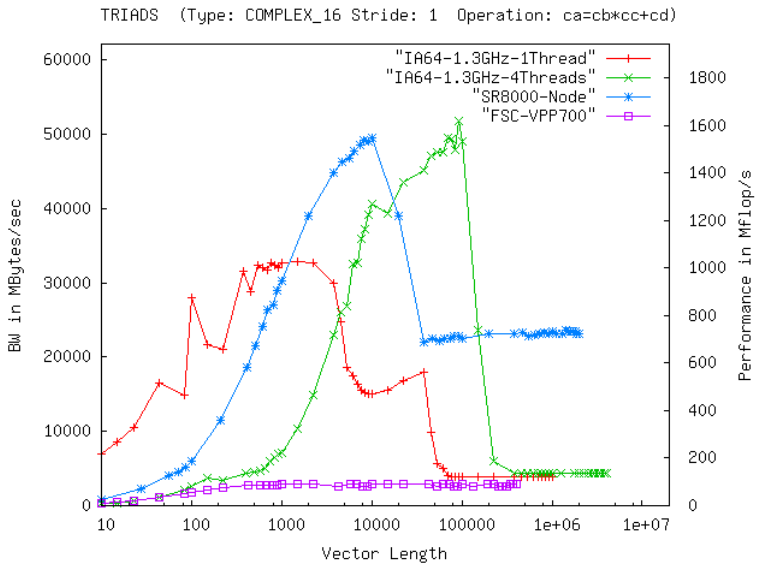

VARIOUS INTRINSICS (Type: ALL_TYPES Stride: 1 Operation: int,dble,cmplx,mod)
"IA64-1.3GHz-1Thread"
"IA64-1.3GHz-4Threads"
"SR8000-Node"
"FSC-VPP700"
BW in MBytes/sec
Performance in Mflop/s
Vector Length
0
5000
10000
15000
20000
25000
30000
35000
40000
0
500
1000
1500
2000
10
100
1000
10000
100000
1e+06
1e+07

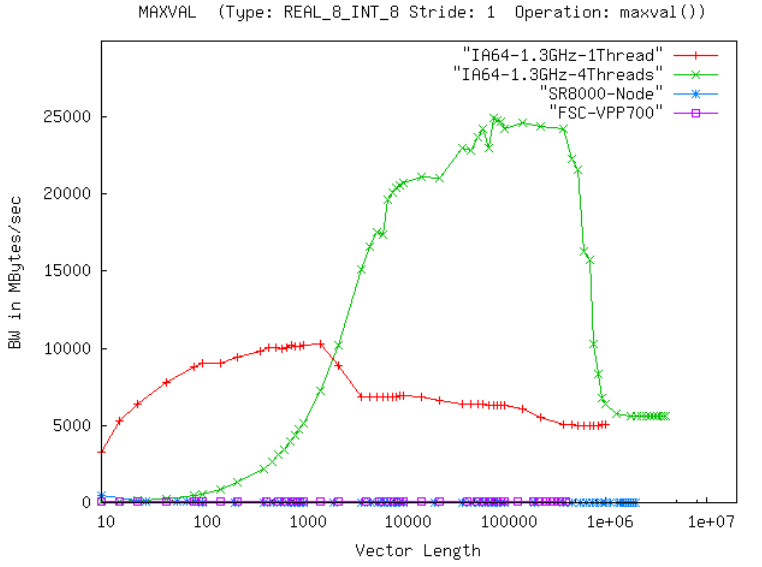

TRIGO_EXP (Type: REAL_8 Stride: 1 Operation: a=acos(sin(exp(b))))
"IA64-1.3GHz-1Thread"
"IA64-1.3GHz-4Threads"
"SR8000-Node"
"FSC-VPP700"
BW in MBytes/sec
Performance in Mflop/s
Vector Length
0
100
200
300
400
500
600
700
800
900
0
1000
2000
3000
4000
5000
6000
7000
10
100
1000
10000
100000
1e+06
1e+07

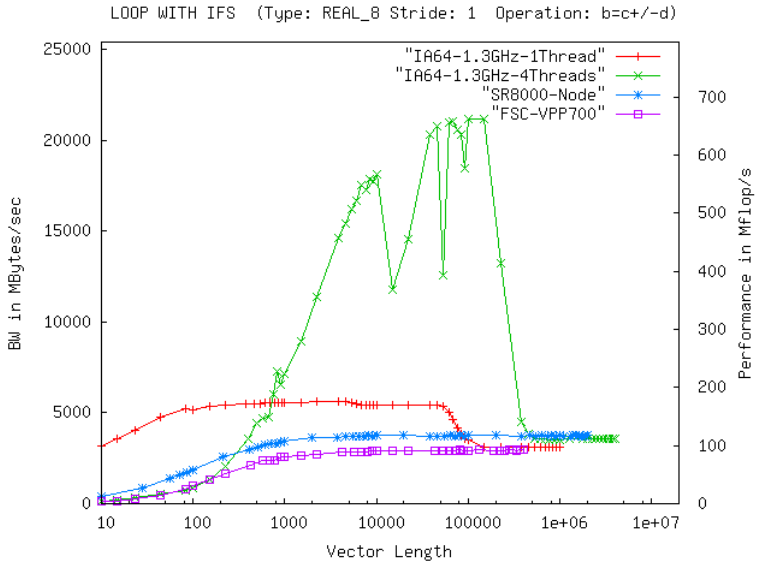

NORM (Type: REAL_8 Stride: 1 Operation: c=sqrt(a*a+b*b))
"IA64-1.3GHz-1Thread"
"IA64-1.3GHz-4Threads"
"SR8000-Node"
"FSC-VPP700"
BW in MBytes/sec
Performance in Mflop/s
Vector Length

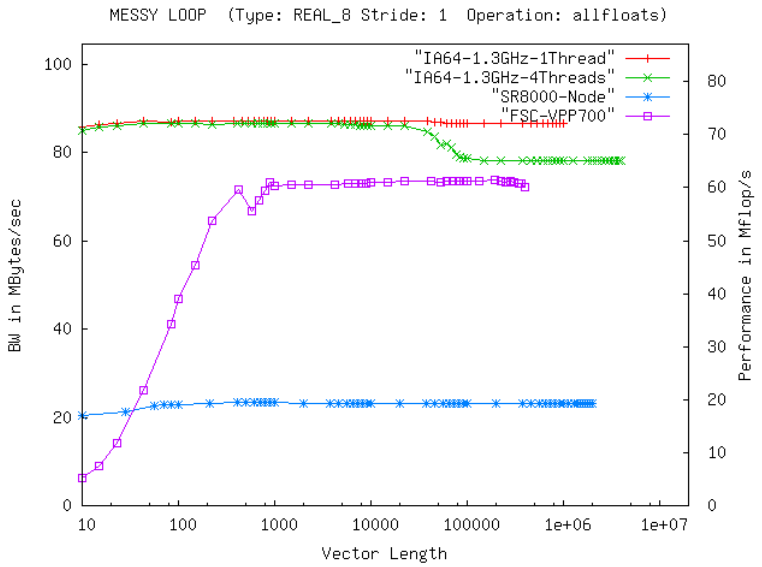

RANDOM_GEN (Type: REAL_8 Stride: 1 Operation: random_number(a))
"IA64-1.3GHz-1Thread"
"IA64-1.3GHz-4Threads"
"SR8000-Node"
"FSC-VPP700"
BW in MBytes/sec
Performance in Mflop/s
Vector Length

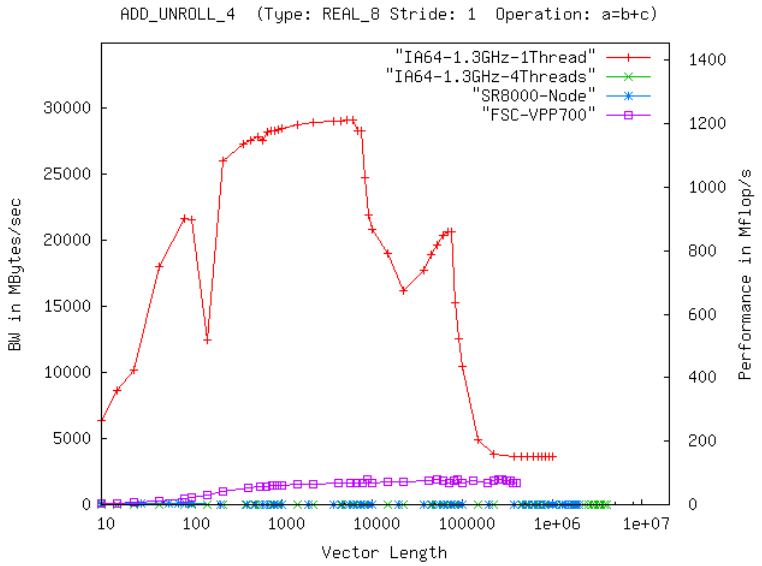

DIVIDE (Type: REAL_8 Stride: 1 Operation: a=b/c)
"IA64-1.3GHz-1Thread"
"IA64-1.3GHz-4Threads"
"SR8000-Node"
"FSC-VPP700"
BW in MBytes/sec
Performance in Mflop/s
Vector Length

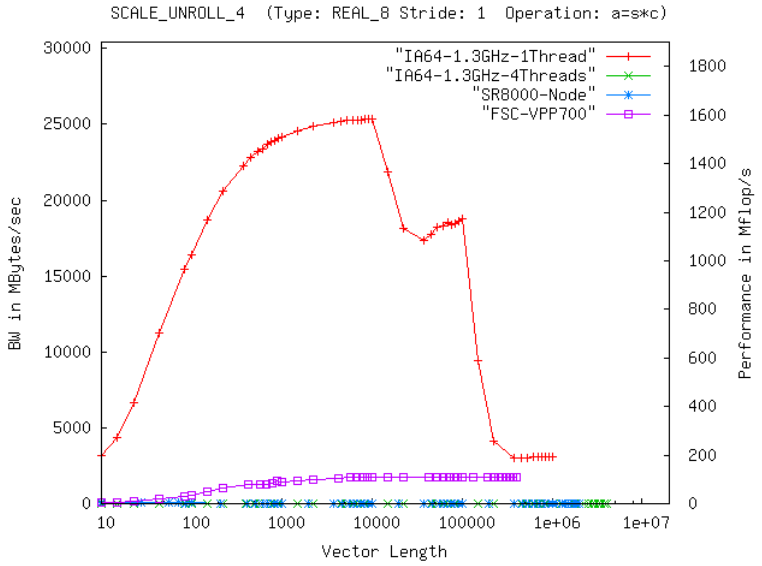

TRIADS (Type: COMPLEX_16 Stride: 1 Operation: ca=cb*cc+cd)
"IA64-1.3GHz-1Thread"
"IA64-1.3GHz-4Threads"
"SR8000-Node"
"FSC-VPP700"
BW in MBytes/sec
Performance in Mflop/s
Vector Length

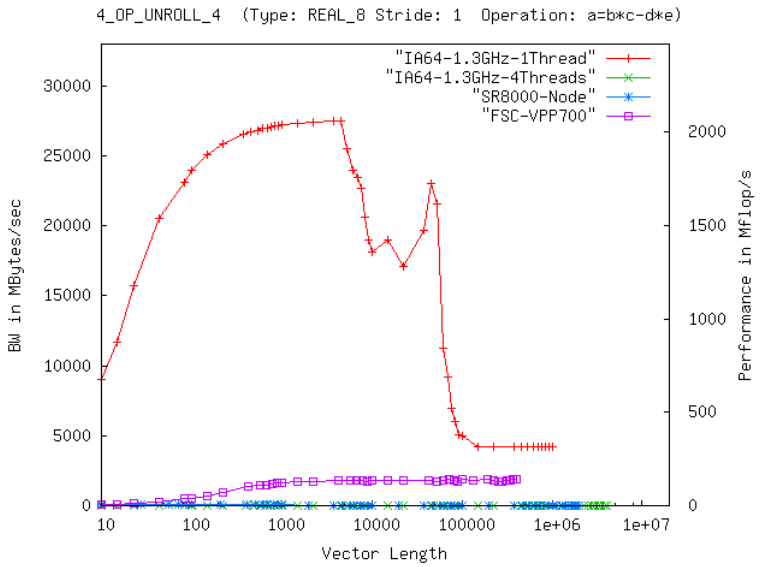

MAXVAL (Type: REAL_8_INT_8 Stride: 1 Operation: maxval())
"IA64-1.3GHz-1Thread"
"IA64-1.3GHz-4Threads"
"SR8000-Node"
"FSC-VPP700"
BW in MBytes/sec
Vector Length

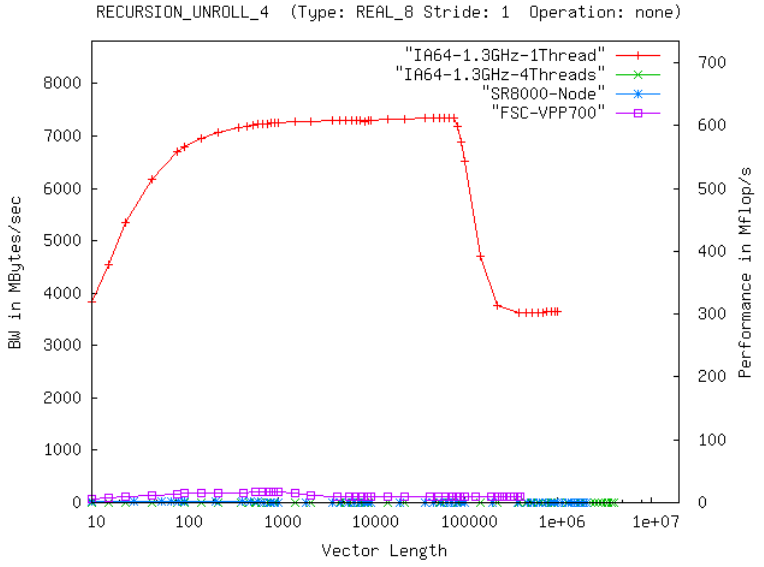

LOOP WITH IFS (Type: REAL_8 Stride: 1 Operation: b=c+/-d)
"IA64-1.3GHz-1Thread"
"IA64-1.3GHz-4Threads"
"SR8000-Node"
"FSC-VPP700"
BW in MBytes/sec
Performance in Mflop/s
Vector Length

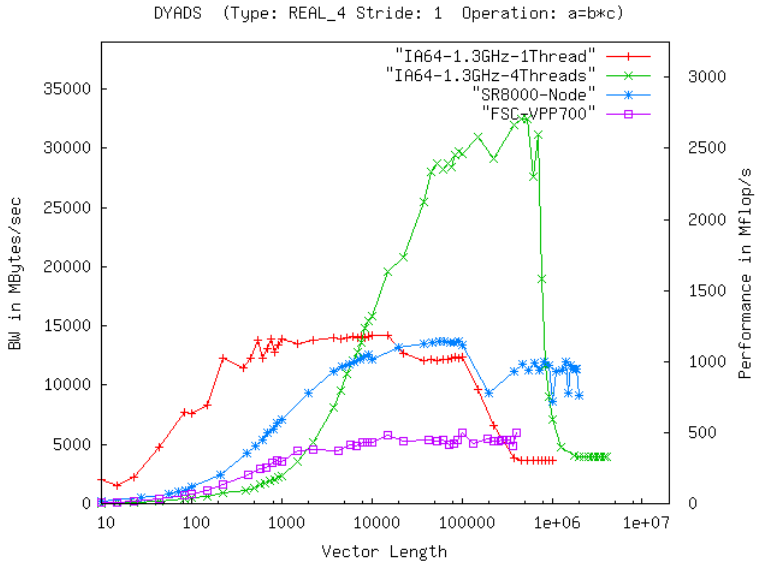

MESSY LOOP (Type: REAL_8 Stride: 1 Operation: allfloats)
"IA64-1.3GHz-1Thread"
"IA64-1.3GHz-4Threads"
"SR8000-Node"
"FSC-VPP700"
BW in MBytes/sec
Performance in Mflop/s
Vector Length
0
20
40
60
80
100
0
10
20
30
40
50
60
70
80
10
100
1000
10000
100000
1e+06
1e+07

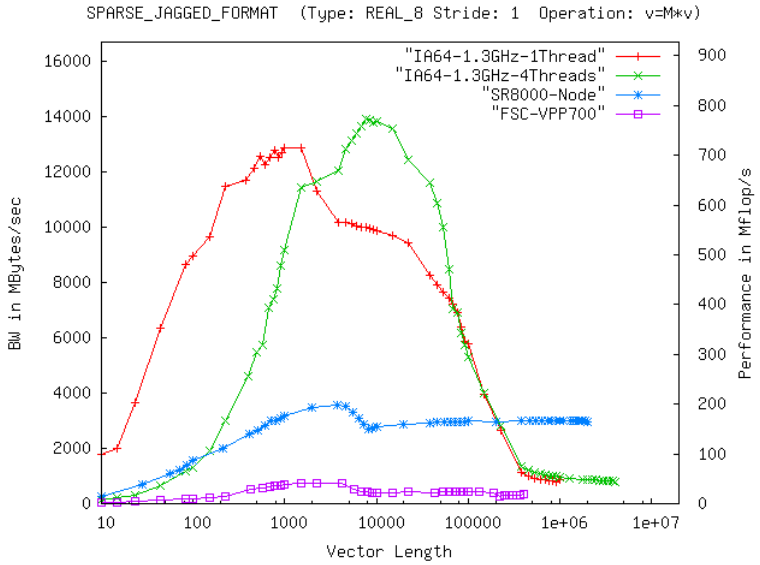

ADD_UNROLL_4 (Type: REAL_8 Stride: 1 Operation: a=b+c)
"IA64-1.3GHz-1Thread"
"IA64-1.3GHz-4Threads"
"SR8000-Node"
"FSC-VPP700"
BW in MBytes/sec
Performance in Mflop/s
Vector Length
0
5000
10000
15000
20000
25000
30000
0
200
400
600
800
1000
1200
1400
10
100
1000
10000
100000
1e+06
1e+07

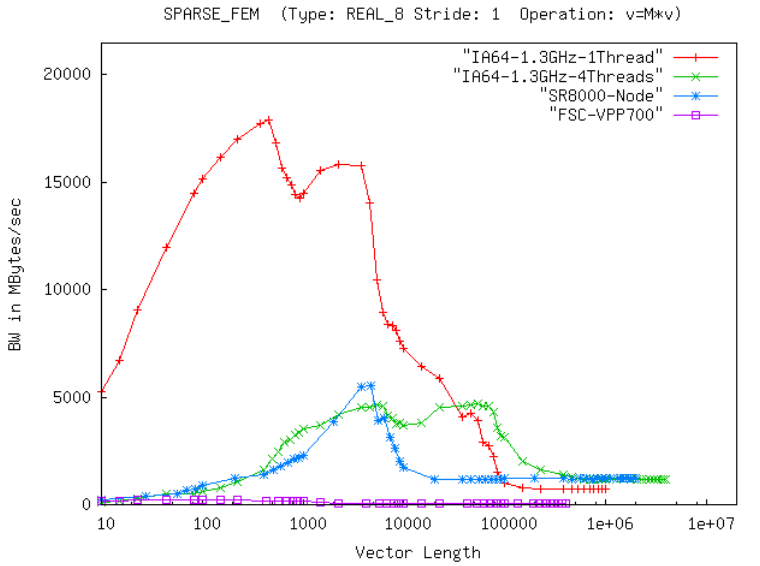

SCALE_UNROLL_4 (Type: REAL_8 Stride: 1 Operation: a=s*c)
"IA64-1.3GHz-1Thread"
"IA64-1.3GHz-4Threads"
"SR8000-Node"
"FSC-VPP700"
BW in MBytes/sec
Performance in Mflop/s
Vector Length
0
5000
10000
15000
20000
25000
30000
0
200
400
600
800
1000
1200
1400
1600
1800
10
100
1000
10000
100000
1e+06
1e+07

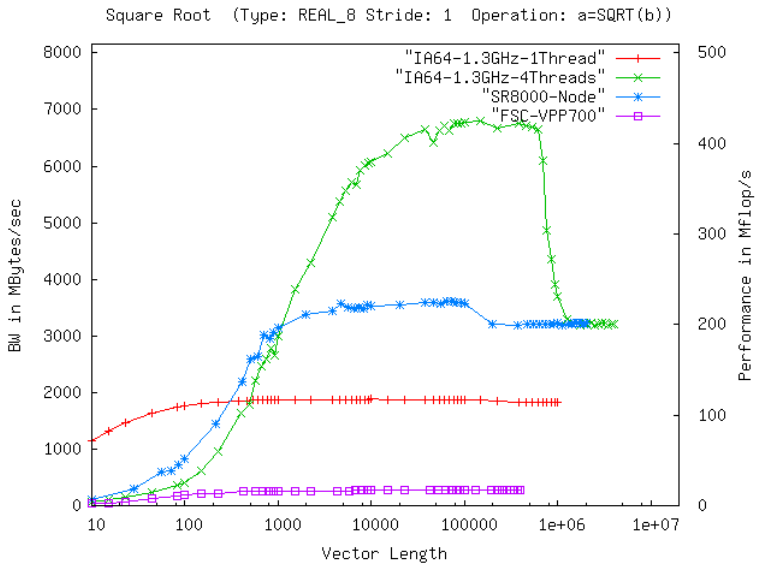

4_OP_UNROLL_4 (Type: REAL_8 Stride: 1 Operation: a=b*c-d*e)
"IA64-1.3GHz-1Thread"
"IA64-1.3GHz-4Threads"
"SR8000-Node"
"FSC-VPP700"
BW in MBytes/sec
Performance in Mflop/s
Vector Length
0
5000
10000
15000
20000
25000
30000
500
1000
1500
2000
10
100
1000
10000
100000
1e+06
1e+07

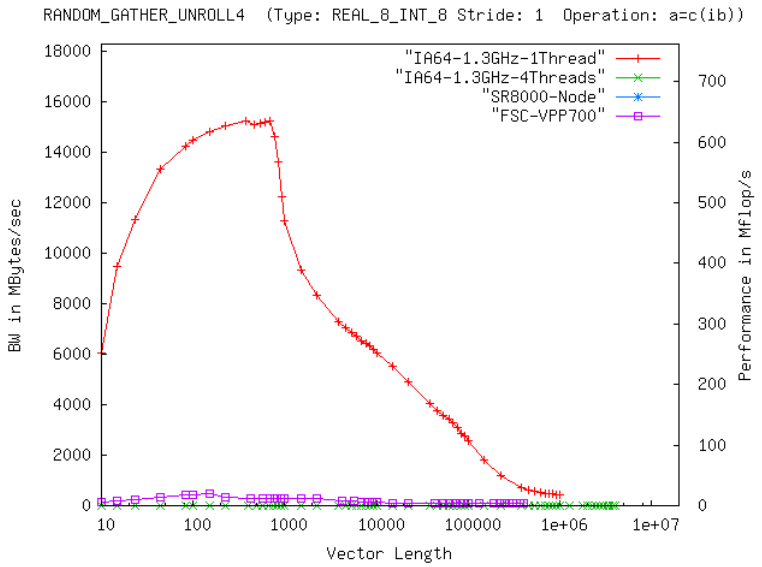

RECURSION_UNROLL_4 (Type: REAL_8 Stride: 1 Operation: none)
"IA64-1.3GHz-1Thread"
"IA64-1.3GHz-4Threads"
"SR8000-Node"
"FSC-VPP700"
BW in MBytes/sec
Performance in Mflop/s
Vector Length
0
1000
2000
3000
4000
5000
6000
7000
8000
100
200
300
400
500
600
700
10
100
1000
10000
100000
1e+06
1e+07

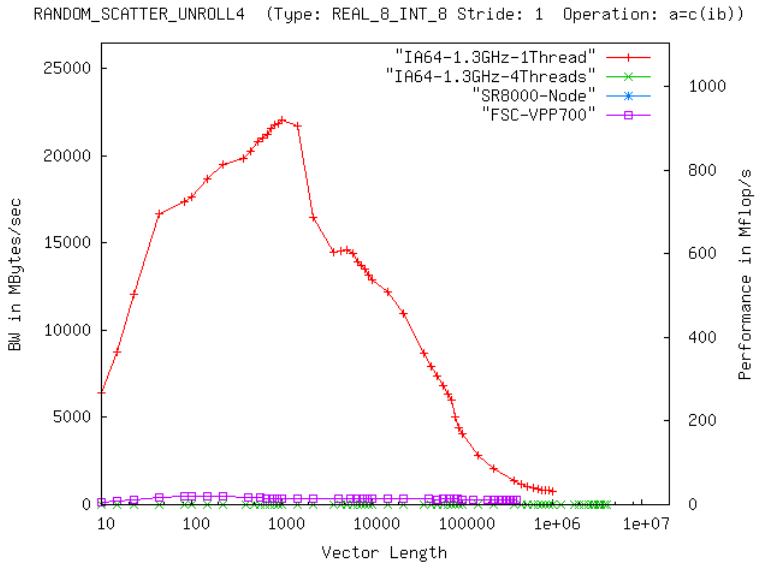

DYADS (Type: REAL_4 Stride: 1 Operation: a=b*c)
"IA64-1.3GHz-1Thread"
"IA64-1.3GHz-4Threads"
"SR8000-Node"
"FSC-VPP700"
BW in MBytes/sec
Performance in Mflop/s
Vector Length
0
5000
10000
15000
20000
25000
30000
35000
500
1000
1500
2000
2500
3000
10
100
1000
10000
100000
1e+06
1e+07

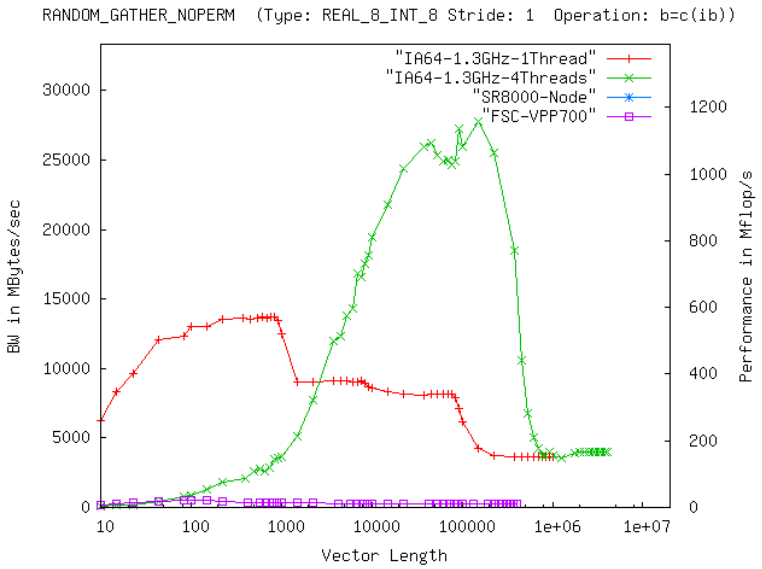

SPARSE_JAGGED_FORMAT (Type: REAL_8 Stride: 1 Operation: v=M*v)
"IA64-1.3GHz-1Thread"
"IA64-1.3GHz-4Threads"
"SR8000-Node"
"FSC-VPP700"
BW in MBytes/sec
Performance in Mflop/s
Vector Length
0
2000
4000
6000
8000
10000
12000
14000
16000
0
100
200
300
400
500
600
700
800
900
10
100
1000
10000
100000
1e+06
1e+07

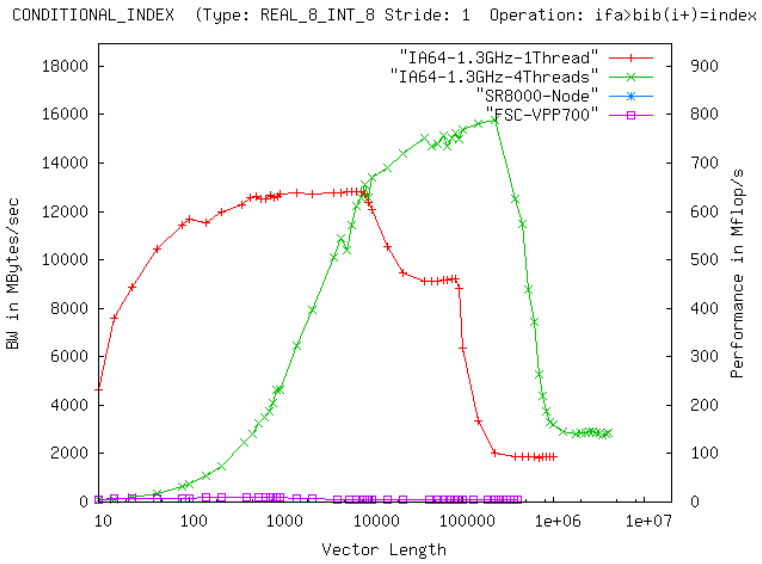

SPARSE_FEM (Type: REAL_8 Stride: 1 Operation: v=M*v)
"IA64-1.3GHz-1Thread"
"IA64-1.3GHz-4Threads"
"SR8000-Node"
"FSC-VPP700"
BW in MBytes/sec
Vector Length
0
5000
10000
15000
20000
10
100
1000
10000
100000
1e+06
1e+07

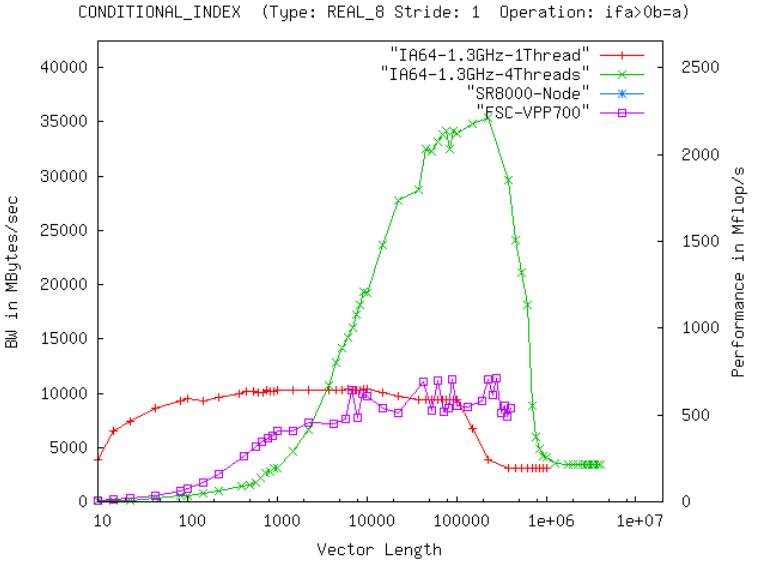

Square Root (Type: REAL_8 Stride: 1 Operation: a=SQRT(b))
"IA64-1.3GHz-1Thread"
"IA64-1.3GHz-4Threads"
"SR8000-Node"
"FSC-VPP700"
BW in MBytes/sec
Performance in Mflop/s
Vector Length
0
1000
2000
3000
4000
5000
6000
7000
8000
0
100
200
300
400
500
10
100
1000
10000
100000
1e+06
1e+07

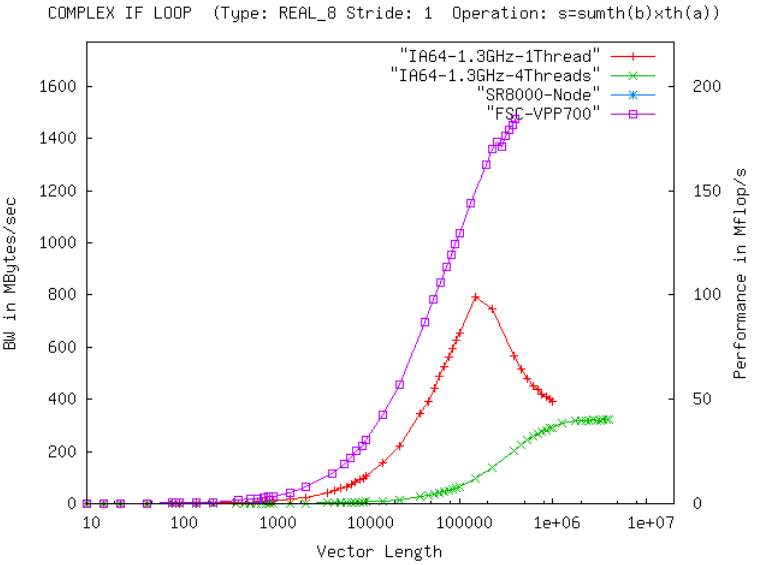

RANDOM_GATHER_UNROLL4 (Type: REAL_8_INT_8 Stride: 1 Operation: a=c(ib))
"IA64-1.3GHz-1Thread"
"IA64-1.3GHz-4Threads"
"SR8000-Node"
"FSC-VPP700"
BW in MBytes/sec
Performance in Mflop/s
Vector Length


RANDOM_SCATTER_UNROLL4 (Type: REAL_8_INT_8 Stride: 1 Operation: a=c(ib))
"IA64-1.3GHz-1Thread"
"IA64-1.3GHz-4Threads"
"SR8000-Node"
"FSC-VPP700"
BW in MBytes/sec
Performance in Mflop/s
Vector Length


RANDOM_GATHER_NOPERM (Type: REAL_8_INT_8 Stride: 1 Operation: b=c(ib))
"IA64-1.3GHz-1Thread"
"IA64-1.3GHz-4Threads"
"SR8000-Node"
"FSC-VPP700"
BW in MBytes/sec
Performance in Mflop/s
Vector Length


CONDITIONAL_INDEX (Type: REAL_8_INT_8 Stride: 1 Operation: ifa>bib(i+)=index
"IA64-1.3GHz-1Thread"
"IA64-1.3GHz-4Threads"
"SR8000-Node"
"FSC-VPP700"
BW in MBytes/sec
Performance in Mflop/s
Vector Length


CONDITIONAL_INDEX (Type: REAL_8 Stride: 1 Operation: ifa>0b=a)
"IA64-1.3GHz-1Thread"
"IA64-1.3GHz-4Threads"
"SR8000-Node"
"FSC-VPP700"
BW in MBytes/sec
Performance in Mflop/s
Vector Length


COMPLEX IF LOOP (Type: REAL_8 Stride: 1 Operation: s=sumth(b)×th(a))
"IA64-1.3GHz-1Thread"
"IA64-1.3GHz-4Threads"
"SR8000-Node"
"FSC-VPP700"
BW in MBytes/sec
Performance in Mflop/s
Vector Length