intel Paragon MPP: System wurde verdoppelt

Shannon Miller/Dirk Sihling/Alfred Geiger

Die intel Paragon war das erste massiv-parallele System, das an der Universitä t Stuttgart als fü r alle Institute allgemein zugä nglicher Rechner 1992 beschafft wur-de. In der Zwischenzeit wurde eine Vielzahl wichtiger Codes aus verschiedenen Anwendungsgebieten auf dieses System gebracht, wobei sich die Paragon als ide-ale Plattform fü r die Entwicklung portabler Software erwies. Ohne den Einsatz der Paragon als Pathfinder System wä re ein Schritt in die Spitzenklasse heutiger Pa-rallelrechner, wie er jetzt mit der an anderer Stelle in dieser Ausgabe vorgestellten Cray T3E vollzogen wurde, niemals mö glich gewesen.

Nach Installation der T3E stellte sich für das RUS aber natürlich die Frage nach dem Sinn eines Weiterbetriebs der Paragon. Da das Betriebsmodell der T3E aufgrund des Einsatzes im Bundeshöchstleistungsrechenzentrum HLRS so gestaltet wird, daß Produktionsrechnungen mit Knotenzahlen jenseits von hundert optimal bearbeitet werden können, ist eine Nutzung der T3E für die Programmentwicklung nur sehr eingeschränkt und in einem späten Stadium des Entwicklungsprozesses möglich. Dies begründet sich in dem Umstand, daß massiv-parallele Systeme aus Effizienzgründen im Space-Sharing Modus (evtl. später Gang-Scheduling) gefahren werden müssen. Prinzipiell könnte parallele Software zwar auch auf Workstation-Clustern entwickelt werden, in vielen Gesprächen mit Benutzern und in eigenen Analysen hat sich jedoch ergeben, daß auch zur Entwicklung eine Ressource zur Verfügung stehen muß, deren Knotenzahl, Verhältnis von Latenzzeit zu Rechenleistung sowie Tool Environment mit den Produktionssystemen verträglich ist. Diese Bedingungen erfüllt die Paragon in Bezug auf die Cray T3E in geradezu idealer Weise, weshalb wir uns für einen Weiterbetrieb entschlossen ha-ben. Im Rahmen der Wartungsvereinbarungen mit der Firma intel konnte sogar ein Ausbau auf die doppelte Leistung vereinbart werden. Dabei kommen SMP-Knoten zum Einsatz, die es ermöglichen, auch Codes mit gemischtem Message-Passing/Shared-Memory-Programmiermodell zu entwickeln, was insbesondere in Bezug auf künftige Rechnergenerationen äußerst interessant ist. Die neuen Knoten sind zudem mit doppelt soviel (64MB) Memory pro Knoten ausgestattet wie die alten, was sicherlich die Programmentwicklung oft vereinfacht. Das RUS wird alle Entwicklungs-Tools, die für Parallelrechner verfügbar sind, in noch stärkerem Maße als bisher für die Paragon zur Verfügung stellen. Wir hoffen mit diesen Maßnahmen alle notwendigen Schritte zu tun, um den Nutzern der Universität Stuttgart eine ideale Entwicklungsumgebung für die MPPs des HLRS zu bieten.

Hardwareänderungen

Bis vor kurzem standen den Benutzern der intel Paragon am RUS 73 General Purpose (GP)-Knoten in der .compute-Partition für parallele Anwendungen zu Verfügung. Ein GP-Knoten besitzt 32 MB Hauptspeicher, von denen der Benutzer 26 MB für seine Anwendungen benutzen kann, und zwei 50-MHz i860-Prozessoren. Einer der Prozessoren kümmert sich ausschließlich um Message Passing, der andere steht dem Benutzer für Anwendungen zu Verfügung.

Letzten Monat wurde das System um 41 Multi Processor (MP)-Knoten erweitert. Jeder MP-Knoten besitzt zusätzlichen einen dritten i860-Prozessor, der zur Bereitstellung von Rechenleistung zu Verfügung steht, und 64 MB Hauptspeicher, von denen der Benutzer 56 MB verwenden kann. Die beiden compute-Prozessoren greifen dabei auf den selben gemeinsamen Hauptspeicher zu.

Einer der 73 GP-Knoten mußte allerdings aus der .compute-Partition entfernt werden, um die Verwaltung des zusätzlichen 5 GB RAIDs für Benutzerdaten und Paging übernehmen zu können.

Die .compute-Partition besteht deshalb aus 72 GP- und 41 MP-Knoten. Bild 1 zeigt, wie diese 113 Knoten angeordnet sind.

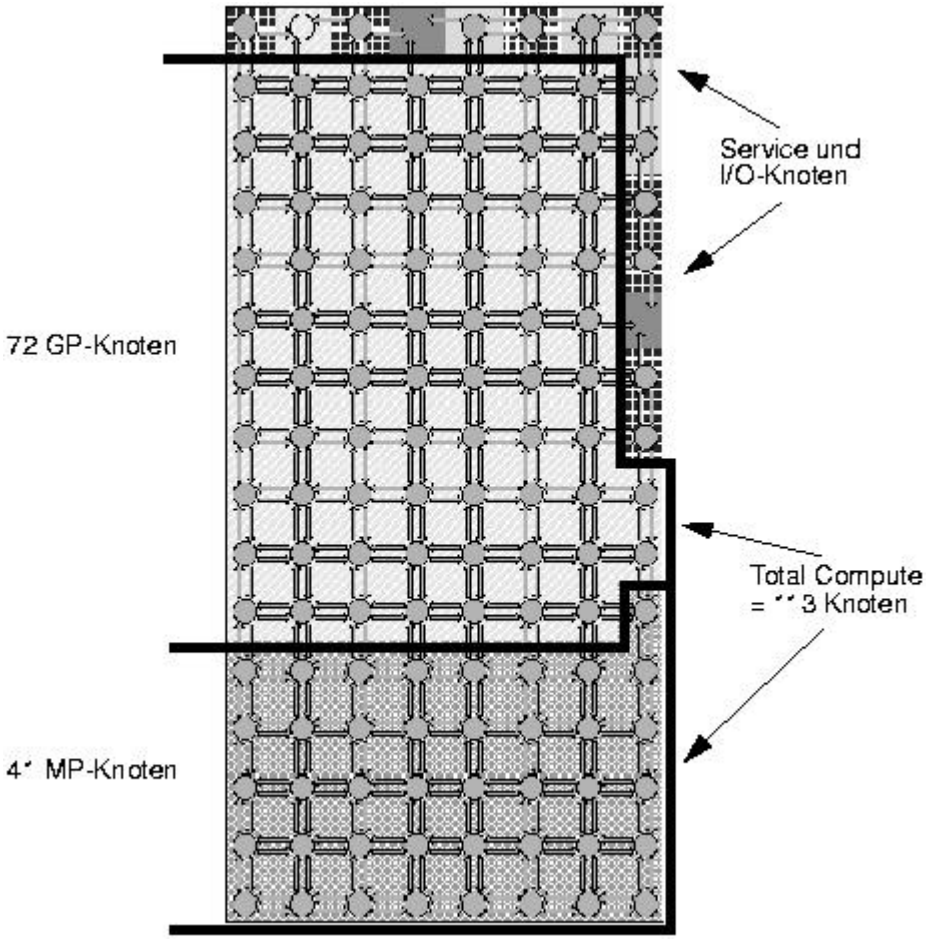

Bild 1: Die neue Paragon-Konfiguration

Theoretisch liegt die Spitzenleistung eines 50-MHz i860-Prozessors bei 75 MFLOPS, wobei der Cache und die Pipeline der Floating Point Unit optimal genutzt werden müssen. Die theoretische Spitzenleistung der Paragon am RUS ist deshalb von 5.475 GFLOPS (73 CPUs x 75 MFLOPS) auf 11.55 GFLOPS (154 CPUs x 75 MFLOPS) angestiegen. Ein realistischer Wert für die tatsächlich meßbare Prozessorleistung liegt so eher bei 50 MFLOPS. Ausgehend von diesem Wert ist auch die tatsächliche Spitzenleistung von 3.65 GFLOPS auf 7.7 GFLOPS angestiegen, wobei ein optimaler paralleler Programmablauf angenommen wird. Wenn eine Anwendung gleichzeitig auf GP- und MP-Knoten läuft, ist es jedoch schwierig, die jeweilige Spitzenleistung der Knoten auszunutzen, es sei denn, daß bei der Lastverteilung die unterschiedlichen Leistungswerte beider Knotenarten berücksichtigt werden. Darauf wird später noch eingegangen.

Die MP-Knoten werden von bereits bestehenden Programmen so benutzt, als wären sie GP-Knoten. Der zusätzliche Prozessor verrichtet keine Arbeit, wenn der Benutzer nicht die dafür notwendigen Maßnahmen ergreift. Wenn also eine Anwendung auf den MP-Knoten nur jeweils einen der beiden Prozessoren benutzt, liegt die theoretisch erreichbare Spitzenleistung nur bei 8.475 GFLOPS (113 Knoten x 75 MFLOPS).

Benutzung der MP-Knoten

Die beiden Compute CPUs auf einem MP-Knoten teilen sich sowohl den Hauptspeicher als auch einen gemeinsamen Betriebssystemkern. POSIX Threads (PThreads) können auf den beiden Prozessoren parallel ablaufen. Ein Benutzer kann seine eigenen PThreads erzeugen, um beide Prozessoren optimal auszunutzen. Da jedoch alle Threads eines Prozesses auf den selben Prozeßdatenbereich zugreifen und asynchron ablaufen, müssen Vorsichtsmaßnahmen zur Erhaltung der Datenintegrität getroffen werden.

Glücklicherweise können der C- und Fortran-Compiler auf der Paragon Code für parallele Threads erzeugen. Wenn die Compiler mit bestimmten Optionen aufgerufen werden, erzeugen sie Programme, die sowohl auf GP- als auch auf MP-Knoten lauffähig sind; wenn diese Programme allerdings auf einem MP-Knoten laufen, wird automatisch ein weiterer Thread für den zweiten Prozessor erzeugt. Sequentielle Programmteile werden nur auf einem Prozessor abgearbeitet, während der andere nichts tut. Wenn allerdings eine Thread-sichere (reentrant) Schleife durchlaufen werden soll, und der Compiler sich eine Leistungssteigerung davon verspricht, verteilt er die Schleifenwiederholungen auf beide Threads (siehe auch Bild 2). Da der zusätzliche Aufwand für die Benutzung eines zweiten Threads sehr gering ist, werden Schleifen auf MP-Knoten häufig doppelt so schnell abgearbeitet, wie auf GP-Knoten. Eine Verbesserung ist typischerweise zu erwarten, wenn die innerste Schleife mindestens 100 bis 200 mal durchlaufen wird. Dieser standardmäßige Schwellenwert für den Compiler kann allerdings durch Optionen verändert werden.

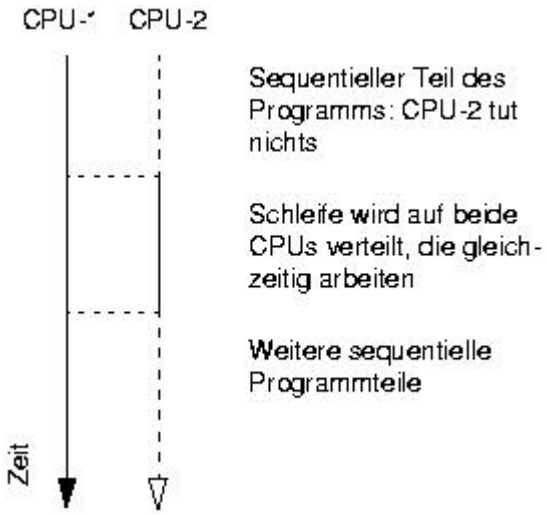

Bild 2: Kontrollfluß auf einem MP-Knoten (`-Mconcur`)

Nicht alle Schleifen können parallelisiert werden. Wenn neue Schleifendurchläufe von vorherigen Durchläufen der selben Schleife abhängig sind, kann der Compiler die Schleife manchmal umformulieren, um sie doch noch parallelisieren zu können. Standardmäßig parallelisiert der Compiler keine Schleifen, die Funktionsaufrufe enthalten. Mit der Option `-Mncall` kann der Benutzer dem Compiler aber signalisieren, daß die enthaltenen Funktionsaufrufe Thread-sicher sind, und die Parallelisierung somit erzwingen.

Von den meisten Standardbetriebssystemroutinen existieren auch Thread-sichere Versionen. Bei geschachtelten Schleifen wird üblicherweise die äußerste parallelisiert.

Man kann auf einfache Art feststellen, ob ein Programm auf MP-Knoten schneller läuft, indem man es mit folgenden Optionen neu kompiliert und linkt, und dann ausschließlich auf MP-Knoten startet (Beschreibung folgt unten): `-D_REENTRANT -Mconcur`

Hierbei ist `-Mconcur` die wichtige Option. Bei Optimierungsstufen von zwei oder größer veranlaßt diese Option den Compiler, auf MP-Knoten die Parallelisierung von Schleifen zu versuchen und die Thread-sicheren Versionen der Systemroutinen einzubinden. Das Symbol `_REENTRANT` sollte beim kompilieren definiert sein, damit die richtigen Funktionsprototypen und Definitionen aus den header Dateien verwendet werden.

Der Programmierer hat sowohl großen Einfluß darauf, wie Schleifen parallelisiert werden (blockweise, zyklisch), als auch darauf, welche Schleifen parallelisiert werden sollen. Sowohl im System User's Guide [1] als auch in den Handbüchern der jeweiligen Compiler [2 et. al.] werden mehrere Optionen für die globale Kontrolle der Parallelisierung aufgeführt. Zusätzlich werden C- und Fortran-Direktiven zur

Kontrolle der Parallelisierung einzelner Schleifen innerhalb der Anwendung beschrieben.

Mit der Option `-Minfo=loop` erhält man Informationen darüber, welche Schleifen tatsächlich parallelisiert wurden, während die Option `-Mneginfo=concur` den Programmierer darüber informiert, welche Schleifen nicht parallelisiert wurden oder parallelisiert werden konnten, und warum es nicht ging.

Wenn ein Programm ohne die Option `-Mconcur` kompiliert wird, wird es in jedem Fall nur auf einem Prozessor des Knotens laufen, ob der Knoten nun vom Typ MP oder GP ist. Eine andere Möglichkeit für die Benutzung zweier Prozessoren besteht darin, bewußt PThreads in der Anwendung zu verwenden.

Programme, die mit `-Mconcur` kompiliert wurden, laufen sowohl auf GP- als auch auf MP-Knoten. Zur Laufzeit wird automatisch festgelegt, wieviele Threads dem Programm zur Verfügung gestellt werden. Standardmäßig benutzt eine Anwendung auf GP-Knoten einen Thread, auf MP-Knoten zwei Threads. Für den Optimierer im Compiler macht es keinen Sinn, mehr Threads zu erzeugen, als Prozessoren zu Verfügung stehen. Um zu verhindern, daß die zweite CPU auf MP-Knoten benutzt wird, kann man die Umgebungsvariable `DFLT_NCPUS = 1` setzen und exportieren. Um die Standardeigenschaften zurückzuerhalten, muß man `DFLT_NCPUS` zurücksetzen oder der Variable den Wert `0` zuweisen. Wenn `DFLT_NCPUS` größer eins ist, und ein Programm auf GP-Knoten ausgeführt werden soll, wird jeder beteiligte GP-Knoten eine Fehlermeldung ausgeben und den Programmablauf beenden. Wenn mit der Anwendung ebenfalls MP-Knoten allokiert wurden, werden diese Knoten blockieren, sobald sie versuchen mit den GP-Knoten zu kommunizieren oder falls sie globale Operationen wie `gdsum()` oder `gsync()` ausführen möchten.

Programmierern wird die Lektüre der Kapitel 6 in [1] und der Kapitel 3 in [2, 3] (C- und Fortran-Compiler) sehr empfohlen. Am besten optimiert man Programme in kleinen Schritten, wobei die Korrektheit des Programms nach jedem Optimierungsschritt überprüft werden sollte. Durchaus deutliche Geschwindigkeitsvorteile auf den MP-Knoten kann man allein dadurch erreichen, indem man bestehende Programme mit der Option `-Mconcur` neu kompiliert. In den meisten Fällen wird man die beste Leistung aber erst dann erhalten, wenn ein Programm, wie in dieser Literatur beschrieben, sorgfältig optimiert wird.

Interaktiver Betrieb

Auf der RUS-Paragon werden interaktive parallele Anwendungen mit `isub` gestartet. Die Erweiterung der .compute-Partition hat auf die Benutzung von `isub` keine Auswirkung. Parallele Anwendungen können wie bisher gestartet werden, z.B. mit:

```
$ isub -sz 113 myapp
```

Durch diesen Aufruf werden 113 Knoten allokiert, egal ob es sich dabei nur um GP-Knoten, nur um MP-Knoten, oder um Knoten beiderlei Typs handelt. Weil die MP-Knoten im unteren Bereich des Systems installiert sind und angeforderte Knoten ausgehend von der linken oberen Ecke der Partition allokiert werden, erhält der Benutzer im Normalfall GP-Knoten, bevor MP-Knoten allokiert werden.

Mit der Option `-nt` (Node Type) kann man gezielt MP- oder GP-Knoten allokieren, in-dem man als Typ entweder `mp` oder `gp` anfordert. Mit dem folgenden Aufruf erhält man eine rechteckige Partition aus 5x8 MP-Knoten:

```
$ isub -sz 5x8 -nt mp myapp
```

Von nun an ist interaktives Arbeiten auf allen Paragon-Knoten 24 Stunden am Tag mög- lich, wie später noch beschrieben wird. Batch jobs haben nachts keine höhere Priorität mehr.

Wenn man ein einziges Programm in einer Partition mit GP- und MP-Knoten laufen läßt, wird es sich in den meisten Fällen so auswirken, als ob alle Knoten dieser Partition GP-Knoten wären. Das liegt daran, daß eine typische Anwendung viele Schleifen durchläuft, mit einer synchronisierenden Operation zum Schluß. Bei einer gleichmäßigen Lastverteilung können die MP-Knoten ihre Arbeit durchaus vor den GP-Knoten beenden, aber dann werden sie mit der synchronisierenden Operation warten müssen, bis die GP-Knoten aufgeholt haben. Um dieses Problem zu verhindern, muß die Anwendung selber die Last so auf die einzelnen Knoten verteilen, daß die MP-Knoten doppelt so viel zu tun be-kommen, wie die GP-Knoten. Mit Hilfe der Funktion `nx_node_attr()` aus der NX Library kann man die Eigenschaften aller Knoten in der Partition der Anwendung feststellen. Eine solche spezielle Lastverteilung ist allerdings im Normalfall nicht attraktiv, da sie zusätzlichen Programmieraufwand erfordert und das Programm dann nicht mehr portierbar ist.

Batch-Betrieb

Zur Zeit macht NQS noch keinen Unterschied zwischen GP- und MP-Knoten. In Kürze werden aber entsprechende Queues angeboten werden, damit man auch mit `qsub` gezielt GP- oder MP-Knoten anfordern kann. Ähnlich wie bei `isub` werden aber ohne spezielle Angaben auch bei `qsub` in Zukunft Knoten beiderlei Typs gemeinsam allokiert werden.

Änderung: Interaktiver Betrieb hat ab jetzt immer höhere Priorität

Um die Benutzung der Paragon als Entwicklungssystem für die neue Cray T3E zu er-leichtern, wird der Interaktivbetrieb von nun an immer Vorrang gegenüber Batchbetrieb haben. Durch Gang-Scheduling wird diese Änderung sehr leicht möglich.

- Batch jobs können mit NQS `qsub` jederzeit abgeschickt werden
- Batch jobs werden während der Prime-Time (07:30-0:30) nicht gestartet, sondern warten in den Queues, bis die Prime-Time endet
- Batch jobs, die bei Beginn der Prime-Time immer noch laufen, werden nicht mehr gekillt, sie laufen weiter, wenn die gleichen Knoten nicht durch interaktive Jobs belegt werden
- Interaktive Jobs können jederzeit Tag und Nacht gestartet werden. Sollten zu der Zeit auf den angeforderten Knoten Batch jobs laufen, so werden diese Gang-Scheduled (suspended), bis keine Interaktiven Jobs die Knoten mehr brauchen
- Da NQS-Jobs jetzt durch gang scheduling jederzeit unterbrochen werden können, werden ihre Timings keine korrekten Ergebnisse liefern. Die globale Uhr in der Paragon, die von `dclock()` und anderen Zeitmessungsroutinen verwendet wird, läuft für alle Jobs weiter, auch für die, die angehalten und ausgelagert wurden
- Timings sollten aus diesem Grund mit `isub` gemacht werden, nicht mit `qsub`

Debugging und Performance Analyse auf MP-Knoten

Anwendungen mit mehreren User Threads werden nur sehr schwach von den Paragon Tools unterstützt. Diese Threads werden entweder vom Programmierer gewollt gestartet, oder entstehen durch Systemroutinen wie `hrecv()` und Programme, die mit `-Mconcur` kompiliert und auf MP-Knoten gestartet wurden. Der Interaktive Parallele Debugger (IPD) gibt diesbezüglich eine Warnung aus, sobald er eine solche Anwendung erkennt. Die Debug Session kann allerdings fortgesetzt werden.

Wenn man innerhalb IPD einen Prozess anhält, muß man sich mit `frame` informieren, welcher Thread gerade von IPD angezeigt wird. Falls nicht der gewünschte Thread an-gezeigt wird, muß man das Programm immer wieder bis zu der Stelle laufen lassen und dann stoppen, bis der gewünschte Thread angezeigt wird.

Die Methoden zur post mortem-Analyse, `coreinfo` und die Untersuchung von Core Files mit IPD, unterstützen Anwendungen mit mehreren Threads. Sie zeigen den Thread an, der einen Fehler verursacht hat.

Mit dem Performance Visualisierungstool Paragraph lassen sich keine Trace Files an-zeigen, die IPD aus Anwendungen mit mehreren Threads erzeugt hat.

Die Programme `prof` und `gprof` funktionieren zwar mit mehreren Threads, aber die angezeigten Werte für die Zahl der Funktionsaufrufe und Zeitmessungen enthalten alle Threads. Das System Performance Visualization tool (SPV) erkennt MP-Knoten und stellt sie richtig dar.

Literatur und weitere Information

[1] Paragon System User's Guide, April 1996, Order Number 312489-005, Chapter 6. (PostScript: `paragon:/usr/share/ps.docs/psug.ps`)

[2] Paragon Fortran Compiler User's Guide, May 1995, Order Number 312491-003, Chapter 3. (PostScript: `paragon:/usr/share/ps.docs/pfug.ps`)

[3] Paragon C Compiler User's Guide, May 1995, Order Number 312490-003, Chapter 3. (PostScript: `paragon:/usr/share/ps.docs/pcug.ps`)

[4] Paragon High Performance Fortran Compiler User's Guide, January 1996, Order Number 313140-002. (PostScript: `paragon:/usr/share/ps.docs/phpfug.ps`)

[5] Paragon C++ Compiler User's Guide, December 1995, Order Number 313151-003, Chapter 3. (PostScript: `paragon:/usr/share/ps.docs/pcppug.ps`)

[6] `more /usr/news/Betrieb` auf der Paragon

[7] Online man pages (auf der Paragon) für isub, qsub, cc, CC, f77, and hpf

Shannon Miller, NA-5744
E-Mail: `smiller@rus.uni-stuttgart.de`

Dirk Sihling, NA-5744
E-Mail: `sihling@rus.uni-stuttgart.de`

Dr. Alfred Geiger, NA-5719
E-Mail: `geiger@rus.uni-stuttgart.de`