

Performancetests paralleler Computersysteme am Beispiel eines standardisierten Anwendungsbeispiels

Dirk Sihling

Studienarbeit im Fach Angewandte Informatik

Betreuer: Dipl.-Ing. Michael Resch
Dr.-Ing. Alfred Geiger

Januar 1997

RECHENZENTRUM DER UNIVERSITÄT STUTTGART
ANWENDUNGEN DER INFORMATIK IM MASCHINENWESEN
PROF. DR.-ING. R. RÜHLE
RUS, ALLMANDRING 30, 70550 STUTTGART

Zusammenfassung

Um parallele Computersysteme hinsichtlich ihrer Eignung zur Lösung von CFD-Problemen einheitlich untersuchen zu können, hat Akin Ecer ein Beispielproblem zur Durchführung von Leistungstests vorgeschlagen. Bei diesem Beispiel handelt es sich um eine Vereinfachung der Wärmeleitgleichung, deren Gleichgewichtslösung für das gegebene Randwertproblem berechnet werden soll.

Bei den in der Studienarbeit untersuchten Rechnersystemen handelt es sich um intel Paragon XP/S-5, IBM-SP2 und Cray-T3D. Der Algorithmus zur Lösung der Differentialgleichung wurde in C implementiert, wobei zur Bereitstellung der Routinen für die Nachrichtenübertragung auf MPI zurückgegriffen wurde.

Die Ergebnisse der durchgeführten Zeitmessungen für Rechnung und Kommunikation erlauben Aufschlüsse über Performance Charakteristika der verschiedenen Rechner.

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung

Derzeit existiert eine sehr große Vielfalt an Rechnern, die im naturwissenschaftlichen Bereich für Berechnungsprobleme eingesetzt werden können. Hauptsächliches Unterscheidungsmerkmal ist die Architektur, mit der die jeweiligen Rechner aufgebaut sind. Die für komplexe Strömungsprobleme eingesetzten Rechner sind

- Vektorrechner
- Parallelrechner

Es zeichnet sich ab, daß vor allem Parallelrechner in den nächsten Jahren an Bedeutung gewinnen werden, da sich mit ihnen theoretisch beliebig große Rechenleistungen und Hauptspeichergrößen erreichen lassen. Die unterschiedlichen Hersteller haben verschiedene Strategien zur Kopplung der Einzelrechner umgesetzt, weshalb nicht jeder Parallelrechner für jede Problemart gleich gut geeignet ist.

Für Programme, in denen die Anzahl der einzelnen Nachrichten sehr groß ist, ist ein möglichst schneller Verbindungsaufbau entscheidend. Dieser wird gekennzeichnet durch die sogenannte Latenzzeit, womit der Zeitraum gemeint ist, der zwischen dem Abschicken des ersten Bytes der Nachricht vom Sender und dem Empfang des ersten Bytes der Nachricht beim Empfänger vergeht. Es bietet sich ein Vergleich mit der Schallausbreitung an, bei der auch eine gewisse Zeit vergeht, bis der Empfänger das Signal der Quelle wahrnehmen kann.

Es gibt andere Berechnungsaufgaben, bei denen nicht so oft kommuniziert werden muß, wo aber die Menge der übertragenen Daten sehr groß ist. Hierfür sind Verbindungen mit großer Bandbreite gefordert. Mit der Bandbreite wird

beschrieben, welche Datenmenge (z.B in Bytes) die Verbindung pro Sekunde übertragen kann. Da diese beiden rechnerabhängigen Kennzahlen das Verhalten der Kommunikation eines parallelen Programms beschreiben, versucht man, diese Eigenschaften mit verschiedenen Testfällen zu bestimmen.

Viele Testprogramme können jedoch nicht den vielfältigen Anforderungen gerecht werden, die strömungsmechanische Probleme an die Rechner stellen. Ausserdem sind durch die Ergebnisse einzelner Tests keine allgemeinen Vergleiche möglichst aller aktuellen Rechnerarchitekturen möglich. Aus diesen Gründen hat Akin Ecer eine Sammlung von Testfällen vorgeschlagen, die die Grundlage für die in dieser Studienarbeit vorgenommene Untersuchung bildet. Damit sollen aussagekräftigere Vergleiche zwischen verschiedenen Rechnerarchitekturen ermöglicht werden.

Kapitel 2

Aufgabenstellung

Um die Rechenleistungen und Eigenschaften verschiedener Parallelrechnerarchitekturen im Hinblick auf deren Einsatz für CFD (Computational Fluid Dynamics) Probleme testen und auch vergleichen zu können, hat Akin Ecer einen Testfall vorgeschlagen, der von möglichst vielen Personen auf möglichst vielen Plattformen bearbeitet werden sollte.

Übersetzung des Originals aus dem Englischen:

Wir würden gerne ein einheitliches Testproblem festlegen, das von jedermann auf jeder beliebigen Rechnerarchitektur gelöst werden kann. Die Entwicklung verschiedener Software-Tools und Rechnersysteme könnte dadurch verfolgt werden, daß nur ein genau festgelegter Algorithmus verwendet werden darf. Für einen bestimmten Fragenkatalog könnten die Antworten durch jenen gegebenen Testfall standardisiert werden. Im Parallelrechnen macht die Vielfalt der verfügbaren Basismaschinen einen Vergleich sehr schwierig. Es ist nicht leicht, allen interessierten Personen Tests mit bestimmten Programmen zu ermöglichen, und manche dieser Programme sind zudem kompliziert. Für die Numerik im Bereich Strömungsmechanik sind auch Standards wie LINPACK nicht unbedingt interessant. Wir haben einen einfachen Algorithmus gewählt und die Eingabeparameter festgelegt. Dieses Vorgehen bringt den Vergleich vieler verschiedener Möglichkeiten auf den kleinsten gemeinsamen Nenner. Wir hoffen, daß jeder den unten aufgeführten Fragenkatalog für seine Basismaschine(n) beantworten kann.

Für das gegebene Beispielproblem sind folgende Einzelheiten festgelegt:

- Die partielle Differentialgleichung $\frac{\partial f}{\partial t} = \nabla^2(f)$ soll auf einem Parallelrechner gelöst werden.
- Explizite Zeitintegration, Vorwärtsdifferenz in Zeit, Zentraldifferenz im Ort.
- Als Berechnungsgebiet wird der Einheitswürfel gewählt.
- Anfangsbedingung: $f = 0$ im Innern der Würfels.
- Randbedingung: $f = x$ auf allen Oberflächen.
- Es soll die Gleichgewichtslösung berechnet werden.
- Die x -, y -, z -Koordinaten aller Gitterpunkte werden im Hauptspeicher abgelegt.

Das Testbeispiel soll mit folgenden Parametern untersucht werden:

- Anzahl der Rechengitterpunkte in jeder Koordinatenrichtung: 10, 30, 100, 500.
- Jeder Zeitschritt soll 1, 10, 100 mal berechnet werden.
- Die verschickten Nachrichten sollen mit den Faktoren 1, 10, 100 vergrößert werden.
- Die x -, y -, z -Koordinaten jedes Rechengitterpunktes sollen 1, 10, 100 mal abgespeichert werden.

Vorgaben für die Programmierung sind:

- Programmiersprache: Fortran.
- Message Passing: Die PVM-Library soll verwendet werden.
- Das Verschicken und Empfangen von Nachrichten soll durch die Verwendung von Standard-PVM-Buffern (via PVMPACK) realisiert werden. Normale Datenkodierung (PVMDEFAULT) und normales Routing (no PVM-ROUTEDIRECT) sollen verwendet werden.
- Lastverteilung: Das Rechenpunktegitter soll möglichst gleichmäßig verteilt werden, damit alle Rechenknoten gleich viele Operationen ausführen.

- Alle Tests sollen auf dedizierten Knoten ausgeführt werden. Es ist uns bewußt, daß das auf manchen Maschinen nur schwer zu erreichen sein wird. In diesem Fall sollen die anderen laufenden Prozesse so gut wie möglich beschrieben werden.

Im Bericht sind aufzuführen:

- Rechenzeit auf einem Prozessor: Dies sollte die serielle Version des Testprogramms sein. Wenn die parallele Version auf einem Prozessor läuft, sollte die Zeit auch erwähnt werden.
- Die gesamte Abwicklungszeit für den Rechenlauf
- Ausführungszeit für jeden Prozeß
- Kostenfaktor: Dies ist die Zeit pro Rechengitterpunkt pro Zeitschritt. Geben Sie außerdem den Gesamtpreis der benutzten Parallelrechner an.
- Speedup: Die Rechenzeit auf einem Prozessor geteilt durch die Zeit, die auf N Prozessoren gebraucht wurde.
- Skalierbarkeit: Dies wird durch die oben erwähnten Parameterstudien erreicht.
- Hardware: Beschreibung von CPU, Taktfrequenz; Architektur von Cache, Hauptspeicher und Netzwerk.

2.1 Änderungen des Testfalls

Die Anzahl der Parameterkombinationen, die im Testfall vorgeschlagen werden, ist zu groß, als daß alle innerhalb eines vernünftigen zeitlichen Rahmens untersucht werden könnten. Für jede Architektur wären 108 Kombinationen für alle zu untersuchenden Knotenzahlen nötig. Außerdem sprengen manche Kombinationen den Rahmen heutiger Parallelrechner; für 500 Rechengitterpunkte in jeder Koordinatenrichtung sind bei Verwendung doppelter Genauigkeit (8 byte pro Zahl) ungefähr 5 Gigabyte Hauptspeicher erforderlich, um die Koordinaten und Funktionswerte abzuspeichern. Diese Zahl bildet die Obergrenze der Fälle, die auf den zur Zeit modernsten Rechnern bearbeitet werden können, wie z.B der NEC-SX4 an der Universität Stuttgart. Wenn die Koordinaten auch noch 100mal abgespeichert werden sollen, sind Speichergrößen um 300 Gigabyte erforderlich, also jenseits der Möglichkeiten heutiger Rechner.

Einige Parameterkombinationen sind außerdem weniger interessant, weshalb sie in dieser Arbeit nicht beachtet wurden. Das Hauptinteresse bestand darin, die Message Passing Eigenschaften der verschiedenen Architekturen zu untersuchen. Aus diesem Grund wurde auf die mehrfache Berechnung desselben Zeitschritts zur Simulation eines höheren Rechenaufwands verzichtet. Ebenso machte es unter diesem Gesichtspunkt keinen Sinn, die Koordinaten mehrfach abzuspeichern, weshalb dieser Parameter konstant bleibt. Wie oben erwähnt, ist die Zahl der Rechengitterpunkte pro Koordinatenrichtung ein Parameter, der an die Speichergröße der untersuchten Rechner angepaßt werden sollte. Deshalb wurden hier auch nur zwei geeignete Werte verwendet, die ein kleines und ein großes Problem simulieren sollten. Dieser Parameter hat außerdem sehr starke Auswirkungen auf die erforderliche Rechenzeit, wie später noch gezeigt werden wird, so daß für große Werte sehr lange Laufzeiten erforderlich würden.

Die tatsächlich verwendeten Parameter sind:

- Anzahl der Rechengitterpunkte in jeder Koordinatenrichtung: 30, 76
- Jeder Zeitschritt wird einmal berechnet.
- Die verschickten Nachrichten werden mit den Faktoren 1 und 10 vergrößert.
- Die x-, y-, z-Koordinaten jeden Rechengitterpunktes werden einmal abgespeichert.

Der Fall mit 76 Rechengitterpunkten pro Koordinatenrichtung wurde ausgewählt, weil er die Diskretisierung mit der größten Zahl von Rechengitterpunkten darstellt, die noch in den Hauptspeicher eines Knotens der intel Paragon passen.

Für den Vergleich der Architekturen können nur Zweierpotenzen als Knotenzahlen verwendet werden, da man auf der Cray-T3D keine anderen Knotenzahlen erhalten kann.

Im Testbeispiel werden FORTRAN und PVM (Parallel Virtual Machine) als Programmiervorgaben gefordert. Es zeichnet sich jedoch ab, daß MPI (Message Passing Interface) der künftige Standard sein wird, weshalb dieses in dieser Arbeit verwendet wird. MPI bietet Unterstützung für FORTRAN77 und C. Das Beispiel wurde deshalb in C implementiert, da C-Compiler noch lange verfügbar sein werden, während die Möglichkeit besteht, daß FORTRAN77 durch FORTRAN90 abgelöst werden wird. Für FORTRAN90 gibt es noch keine MPI-Unterstützung, da im FORTRAN90-Standard keine Aussagen darüber gemacht werden, wie Daten im Speicher abgelegt werden. Diese Kenntnis ist aber für MPI von großer

Bedeutung, weil die Kommunikationsroutinen direkt auf den Speicher zugreifen müssen.

Zusätzlich zu den vorgeschlagenen Parametern wurde untersucht, welche Unterschiede sich zwischen globalen und lokalen Kriterien ergeben, die zur Synchronisierung zwischen den Zeitschritten und zur Überprüfung einer Abbruchbedingung verwendet werden können. Im ersten Fall werden alle am Problem beteiligten Einzelrechner in die Kommunikation miteinbezogen, im zweiten kommuniziert jeder Einzelrechner nur mit seinen direkten Nachbarn.

Durch die verwendeten Parameterkombinationen ergeben sich vier Problemfälle, die jeweils auf drei Architekturen untersucht werden. Diese Problemfälle werden wiederum jeweils mit sechs unterschiedlichen Knotenzahlen untersucht. Das ergibt eine Gesamtzahl von 72 Rechenläufen, die zum Vergleich der drei Architekturen erforderlich sind.

Um den Einfluß der Gebietszerlegung zeigen zu können, sind Rechenläufe mit allen Knotenzahlen von eins bis einschließlich 73 nötig. Diese Untersuchung ist allerdings nur auf der intel Paragon möglich.

Zusätzlich sind 12 weitere Rechenläufe erforderlich, um den Unterschied zwischen globaler und lokaler Synchronisierung auf T3D und Paragon zu untersuchen.

Insgesamt sind schon für diese Untersuchungen ca. 160 Rechenläufe notwendig, womit der Aufwand für die Untersuchung weiterer Parameterkombinationen aus dem Testfall deutlich wird.

Kapitel 3

Software - MPI

Viele der für naturwissenschaftliche Anwendungen benutzten Programmiersprachen, wie Fortran, C oder C++, bieten keine Funktionalitäten zur Kommunikation zwischen Einzelrechnern. Um diesem Mangel abzuhelpfen, haben verschiedene Hersteller von Hard- und Software, wie auch Wissenschaftler verschiedener Universitäten, das Message Passing Interface Forum (MPIF) [?] gebildet. Das Ergebnis der Arbeit dieses Forums ist der Standard MPI, der einen Funktionsumfang für die Entwicklung portabler paralleler Anwendungen bereitstellt. Der große Erfolg von MPI hat dazu geführt, daß das MPIF derzeit an einer Erweiterung des Funktionsumfangs arbeitet, die den neuen Standard MPI-2 bilden soll.

3.1 Begriffserklärungen

Es gibt einige Konzepte und Begriffe, die durch MPI erstmals verwendet wurden. Diese werden hier zum besseren Verständnis der in ?? beschriebenen Routinen erklärt. Eine ausführliche Einführung in die Begriffe und Konzepte von MPI findet man in [?].

3.1.1 Kommunikator

Ein Kommunikator besteht aus einem Kontext und aus einer Gruppe von Prozessen, die auf verschiedenen Einzelrechnern ablaufen. Bei der Initialisierung von MPI wird automatisch der Kommunikator MPI_COMM_WORLD erzeugt, der alle Prozesse der Anwendung beinhaltet. Aus programmtechnischen Gründen kann

es vorteilhaft sein, neue Kommunikatoren zu erzeugen, die jeweils bestimmte Prozesse in einen gewissen Bezug zueinander setzen. Dadurch lassen sich Einheiten ausgewählter Prozesse bilden, die aus bestimmten Gründen zusammengefaßt werden sollen. Die Routinen zum Versenden oder Empfangen von Nachrichten müssen immer angeben, in welchem Kommunikator die Kommunikation stattfinden soll. Die Nummer eines Prozesses gilt immer nur bezüglich eines Kommunikators. Ein Prozeß kann deshalb mehrere Nummern besitzen, wenn er in mehreren Kommunikatoren verwendet wird. Im Testbeispiel wird ein neuer Kommunikator erzeugt, um alle beteiligten Prozesse in einer neuen, dem Problem angepaßten, Struktur zu ordnen.

3.1.2 Rank

Mit dem Rang wird die Nummer eines Prozesses innerhalb des jeweiligen Kommunikators bezeichnet. Diese Nummern laufen von 0 bis $n - 1$, wobei n die Zahl der Prozesse im Kommunikator ist.

3.2 Verwendete Routinen

Ein paralleles Programm, das Message Passing verwendet, läßt sich schon mit sechs MPI-Routinen implementieren. MPI stellt aber mehr als hundert Funktionen zur Verfügung, die die Organisation der Kommunikation für den Programmierer vereinfachen und ihm viel eigene Arbeit abnehmen können. Die zur Implementierung des Testbeispiels benutzten Funktionen sind im folgenden beschrieben. Eine detaillierte Beschreibung des Funktionsumfangs findet man ebenfalls in [?].

MPI_Init

*Syntax: int MPI_Init(int *argc, char ***argv)*

Diese Funktion muß aufgerufen werden, bevor irgendeine andere MPI-Routine verwendet werden kann. Sie sorgt für die Initialisierung aller intern von MPI benötigten Variablen. Außerdem erzeugt sie den Kommunikator MPI_COMM_WORLD, der alle Prozesse der gestarteten Anwendung beinhaltet.

MPI_Finalize

Syntax: int MPI_Finalize(void)

MPI_Finalize bildet das Gegenstück zu MPI_Init. Die Routine sorgt dafür, daß alle Prozesse die MPI-Umgebung auf geordnete Weise verlassen. Nach dem Aufruf von MPI_Finalize darf keine andere MPI-Routine mehr aufgerufen werden, auch nicht MPI_Init. Der Benutzer muß dafür sorgen, daß alle Nachrichten für den Prozeß abgeschlossen sind, der MPI_Finalize aufruft. Es empfiehlt sich, MPI_Finalize als letzten aufzurufenden Befehl zu benutzen, da manche MPI-Implementierungen in der Routine den Programmablauf beenden.

MPI_Wtime

Syntax: double MPI_Wtime(void)

Mit dieser Routine stellt MPI eine Möglichkeit zur Verfügung, Zeitmessungen für Programmteile vorzunehmen. MPI_Wtime liefert die Zeit in Sekunden, die seit einem bestimmten Zeitpunkt vergangen ist. Dabei wird sichergestellt, daß sich dieser Zeitpunkt während der Laufzeit des Prozesses nicht verändert. Laufzeiten zwischen zwei Aufrufen von MPI_Wtime erhält man also aus der Differenz der beiden Rückgabewerte. Die zurückgegebenen Zeiten gelten jeweils nur für den eigenen Knoten, da sich andere Knoten auf andere bestimmte Zeitpunkte in der Vergangenheit beziehen können.

MPI_Comm_size

*Syntax: int MPI_Comm_size(MPI_Comm comm, int *size)*

Mit Hilfe dieser Funktion läßt sich feststellen, wie viele Prozesse zu einem Kommunikator gehören. Vor allem die Größe des Standardkommunikators MPI_COMM_WORLD ist von großem Interesse, damit die Anwendung feststellen kann, wie viele Prozesse insgesamt an der Problembearbeitung beteiligt sind.

MPI_Comm_rank

*Syntax: int MPI_Comm_rank(MPI_Comm comm, int *rank)*

Jeder Prozeß kann mit dieser Routine seine Nummer innerhalb eines bestimmten Kommunikators herausfinden. Diese Nummern dienen der Kennzeichnung von Sender und Empfänger einer Nachricht und haben daher eine sehr große Bedeutung. Sie bilden das einzige Unterscheidungsmerkmal der verschiedenen Prozesse. Die Nummern laufen von 0 bis $n - 1$, wobei n die Anzahl der Prozesse innerhalb des Kommunikators ist.

MPI_Barrier

Syntax: int MPI_Barrier(MPI_Comm comm)

Diese Routine blockiert den aufrufenden Prozeß, bis alle anderen Prozesse des Kommunikators diese Routine ebenfalls aufgerufen haben. Damit lassen sich alle Prozesse innerhalb eines Kommunikators synchronisieren.

MPI_Bcast

*Syntax: int MPI_Bcast(void *buffer, int count, MPI_Datatype datatype, int root, MPI_Comm comm)*

Bcast steht für Broadcast und ermöglicht ein Versenden einer Nachricht von einem Knoten an alle Knoten innerhalb des Kommunikators. MPI_Bcast verschickt eine Nachricht von dem als Sender gekennzeichneten Prozeß an alle Prozesse des Kommunikators, auch den Sender.

MPI_Gather

*Syntax: int MPI_Gather(void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvttype, int root, MPI_Comm comm)*

Jeder Prozeß schickt den Inhalt seines Sendepuffers an den als Empfänger gekennzeichneten Prozeß, auch der Empfänger selbst. Die Nachrichten werden beim

Empfänger nach Prozeßnummern geordnet im Speicher abgelegt. Die Routine wird verwendet, wenn Daten von vielen Prozessen auf einem Prozeß gesammelt werden sollen.

MPI_Allreduce

*Syntax: int MPI_Allreduce(void *sendbuf, void *recvbuf, int count, MPI_Datatype datatype, MPI_Op op, MPI_Comm comm)*

Diese Routine führt eine Operation mit den Daten in den jeweiligen Sendepuffern aller Prozesse durch und legt das Ergebnis in den Empfangspuffern aller Prozesse ab. Die Routine wird verwendet, wenn Daten aller Prozesse miteinander verknüpft werden sollen (z.B. Addition, Maximum, Oder-Verknüpfung, etc.) und das Ergebnis dieser Verknüpfung wieder auf allen Prozessen verfügbar sein soll. Mit Hilfe von MPI_Allreduce läßt sich z.B. eine globale Summe aus den Restfehlern aller Prozesse bilden, die als Abbruchkriterium verwendet werden kann.

MPI_Pack

*Syntax: int MPI_Pack(void *inbuf, int incount, MPI_Datatype datatype, void *outbuf, int outsize, int *position, MPI_Comm comm)*

Mit Hilfe dieser Routine lassen sich die Inhalte nicht zusammenhängender Speicherbereiche zu einer einzigen Nachricht zusammenpacken. Damit kann man ausnutzen, daß die Übertragungszeit für eine einzige lange Nachricht kleiner ist, als für viele kleine Nachrichten mit dem gleichen Gesamtinhalt, weil die zum Verbindungsaufbau notwendige Zeit nur einmal auftritt.

MPI_Unpack

*Syntax: int MPI_Unpack(void *inbuf, int insize, int *position, void *outbuf, int outcount, MPI_Datatype datatype, MPI_Comm comm)*

Mit Hilfe von MPI_Unpack lassen sich die mit MPI_Pack zusammengefaßten Speicherinhalte wieder auspacken und wieder ihren jeweiligen Variablen oder Feldern zuordnen.

MPI_Pack_size

*Syntax: int MPI_Pack_size(int incount, MPI_Datatype, MPI_Comm comm,
int *size)*

Diese Routine ermöglicht dem Programmierer, die erforderliche Größe des Speichers zu ermitteln, in den er verschiedene Daten packen möchte. Sie liefert eine Obergrenze, die höher als die tatsächlich benötigte Speichergrenze liegen kann.

MPI_Isend

*Syntax: int MPI_Isend(void *buf, int count, MPI_Datatype datatype, int dest,
int tag, MPI_Comm comm, MPI_Request *request)*

MPI_Isend ist eine Punkt-zu-Punkt Kommunikationsroutine. Sie führt ein immediate send aus, was bedeutet, daß die Nachrichtenübertragung vom Sender zum Empfänger gestartet, deren Beendigung aber nicht abgewartet wird. Das macht Sinn bei Architekturen, die eine Nachrichtenübertragung parallel zur eigentlichen Berechnung abarbeiten können, ohne diese dadurch zu bremsen. Dann kann die Zeit während der Kommunikation für Berechnungen genutzt werden. Bei Rechnern, die Kommunikation und Anwendung nicht gleichzeitig bearbeiten können, wird der Abschluß des Sendens abgewartet.

MPI_Irecv

*Syntax: int MPI_Irecv(void *buf, int count, MPI_Datatype datatype, int source,
int tag, MPI_Comm comm, MPI_Request *request)*

Diese Routine führt ein immediate receive aus, d.h. ein Knoten bereitet den Empfang einer Nachricht nur vor, wartet aber nicht, bis der Empfang abgeschlossen ist. Sie ist das passende Gegenstück zu MPI_Isend.

MPI_Waitall

*Syntax: int MPI_Waitall(int count, MPI_Request *array_of_requests,
MPI_Status *array_of_statuses)*

Diese Funktion sorgt dafür, daß auf den Abschluß von Sende- oder Empfangsvorgängen wie oben beschrieben gewartet wird. Bei Verwendung der immediate Kommunikationsroutinen erhält man einen sogenannten Request-handle zurück, der als Parameter an die MPI_Waitall-Routine übergeben wird. Der Programm- lauf hält bis zur Beendigung aller mit dem handle verbundenen Kommunikations- vorgänge.

MPI_Cart_create

*Syntax: int MPI_Cart_create(MPI_Comm comm_old, int ndims, int *dims,
int *periods, int reorder, MPI_Comm comm_cart)*

Mit dieser Routine lassen sich die Prozesse eines Kommunikators in einer Topologie anordnen, in der die einzelnen Prozesse zusätzlich zu ihren Nummern durch kartesische Koordinaten gekennzeichnet sind. Man erhält einen neuen Kommunikator zurück, der als Bezeichner für die so angeordneten Prozesse verwendet werden kann. Die an das Problem angepaßte Anordnung der Prozeßstruktur vereinfacht die Organisation der Kommunikation zwischen voneinander abhängigen Prozessen. Die Prozeßtopologie kann an die Gebietszerlegung angepaßt werden.

MPI_Cart_coords

*Syntax: int MPI_Cart_coords(MPI_Comm comm, int rank, int maxdims,
int *coords)*

Wenn ein Prozeß zu einem Kommunikator mit kartesischen Koordinaten gehört, gibt ihm diese Routine die zu einer Prozeßnummer gehörenden kartesischen Koordinaten zurück. Auf diese Weise stellt er auch seine eigenen Koordinaten fest, mit denen er seine Position in der Prozeßtopologie bestimmen kann.

MPI_Cart_shift

*Syntax: int MPI_Cart_shift(MPI_Comm comm, int direction, int disp,
int *rank_source, int *rank_dest)*

In einer mehrdimensionalen Prozeßtopologie können Nachrichten entlang der Koordinatenachsen übertragen werden. Dabei können die direkt benachbarten Prozesse übersprungen und erst der zweite oder dritte Prozeß entlang der Achse angesprochen werden. Für die eigentlichen Sende- und Empfangsroutinen sind jedoch die entsprechenden Prozeßnummern erforderlich. Die Routine MPI_Cart_shift liefert direkt die Nummern der gesuchten Prozesse.

MPI_Type_vector

*Syntax: int MPI_Type_vector(int count, int blocklength, int stride,
MPI_Datatype oldtype, MPI_Datatype *newtype)*

Datenbereiche, die im Speicher konstante Abstände zwischen den jeweils einzelnen Daten haben, lassen sich mit MPI_Type_vector als Datentyp definieren. Dieser selbstdefinierte Datentyp kann direkt von den Kommunikationsroutinen verwendet werden. Das bietet den Vorteil, daß Daten vor dem Verschicken nicht erst aus den verschiedenen Speicherbereichen in einen zusammenhängenden Puffer kopiert werden müssen, sondern durch die Senderoutine direkt aus den richtigen Plätzen ausgelesen und beim Empfänger wieder direkt an den richtigen Stellen abgelegt werden.

MPI_Type_hvector

*Syntax: int MPI_Type_hvector(int count, int blocklength, MPI_Aint stride,
MPI_Datatype oldtype, MPI_Datatype *newtype)*

Diese Routine bietet die gleiche Funktionalität wie MPI_Type_vector. Der Unterschied ist der, daß stride, der Abstand zwischen den einzelnen Datenbereichen, in bytes statt in Elementen des Datentyps angegeben wird. Das bietet sich vor allem bei selbstdefinierten Datentypen an, um neue Datentypen aus selbstdefinierten Datentypen zu erzeugen.

MPI_Type_extent

*Syntax: int MPI_Type_extent(MPI_Datatype datatype, MPI_Aint *extent)*

Mit dieser Routine läßt sich die Größe von selbstdefinierten Datentypen ermitteln. Dies ist nötig, um die Puffergröße richtig einstellen zu können, wenn man mit MPI_Pack verschiedenartige, selbstdefinierte Datentypen zu einer Nachricht zusammenfassen möchte.

MPI_Type_commit

*Syntax: int MPI_Type_commit(MPI_Datatype *datatype)*

Diese Routine meldet selbsterzeugte Datentypen bei MPI an, damit sie von MPI-Routinen verwendet werden können. Die Anordnung der einzelnen Datenteile im Speicher wird an MPI gemeldet.

MPI_Type_free

Syntax: int MPI_Type_free(MPI_Datatype datatype)

Nicht mehr benutzte, selbst definierte, Datentypen werden mit dieser Funktion wieder freigegeben, damit der durch die Verwaltung dieser Datentypen belegte Speicher wieder frei wird. Diese Datentypen können danach nicht mehr benutzt werden.

Kapitel 4

Beschreibung der Basismaschinen

Im Rahmen dieser Studienarbeit werden drei Parallelrechner untersucht, die sich durch ihre Architektur stark unterscheiden. Es sind dies

- intel Paragon XP/S-5
- Cray-T3D
- IBM-SP2

Die Rechner von Intel und Cray sind schon einige Jahre im Betrieb, während die IBM-SP2 mit dem neuen High-Performance-Switch gerade erst in Betrieb genommen wurde.

Das macht sich vor allem bei der Prozessorleistung bemerkbar. Zur Zeit ihrer Markteinführung hatten die Rechner von Intel und Cray sehr leistungsfähige Prozessoren, der DEC 21064, der in der T3D Verwendung fand, war sogar der zu der Zeit stärkste Prozessor. Der neue POWER2-Chipsatz in der SP2 ist jedoch um einiges stärker als der DEC 21064, und um vieles mehr als der i860XP.

Ein weiteres Unterscheidungsmerkmal ist die Verbindung zwischen zwei Rechenknoten. Während bei der IBM-SP2 erst bei Bedarf eine Verbindung zwischen zwei Einzelrechnern geschaltet wird, sind die Knoten der Paragon und T3D fix vernetzt. Diese beiden Netze unterscheiden sich ebenfalls; die Paragon hat ein zweidimensionales Verbindungsnetzwerk, d.h. die Eckknoten haben zwei, die Randknoten drei, und die mittleren Knoten vier Nachbarn im Netz. Bei der T3D, deren Netzwerk als 3D-Torus ausgelegt ist, hat jeder Knoten sechs Nachbarknoten. Für alle Knoten im Netzwerk sieht die Umgebung gleich aus.

4.1 intel Paragon XP/S-5

Die Paragon ist das aus der Experimentalmachine Touchstone-delta (ARPA-Projekt) entstandene Produkt, mit dem intel seinen Weg in Teraflop/s-Regionen beginnen will. Für Altkunden sollte mit dieser Maschine die Kompatibilität mit der bisherigen iPSC-Linie aufrecht erhalten werden, andererseits sollten Konzepte verwendet werden, die für wirkliche später folgende Höchstleistungen tragfähig sind. Die Stuttgarter Maschine wurde 1992 im Rahmen einer Kooperation zwischen dem Institut für Parallele und Verteilte Höchstleistungsrechner (IPVR), dem Institut für Computeranwendungen II (ICA II) und dem Rechenzentrum der Universität Stuttgart (RUS) installiert. Sie ist ein Parallelrechner mit 87 Knoten und einer Leistung von über 5 GFLOPS mit verteiltem Hauptspeicher und wird mittels Message Passing betrieben.

4.1.1 Architektur

Die Paragon ist aus Kabinetts aufgebaut, in die auf jeweils vier Backplanes verteilt 64 Knoten eingesteckt werden können. In Stuttgart stehen zwei Kabinetts mit 87 Knoten. Einige dieser Knoten übernehmen spezielle Aufgaben wie Disk-I/O und Netzwerkverbindungen nach außen. Für Anwender stehen 5 Knoten in einer Service-Partition zu Verfügung, auf denen sie wie auf einer Workstation interaktiv arbeiten können; in einer Compute-Partition stehen Anwendern zur Zeit 73 Knoten für parallele Anwendungen dediziert zur Verfügung. Die Knoten sind über ein zweidimensionales Gitter miteinander verbunden. In Bild ?? wird der schematische Aufbau dargestellt, wie er von dem System Performance Visualizer (SPV) wiedergegeben wird.

4.1.2 Aufbau der Rechenknoten

Alle Knoten der Compute-Partition besitzen 32 MB Hauptspeicher, ein Netzwerkinterface und zwei Prozessoren vom Typ i860XP, von denen einer nur die Anwendung bearbeitet und der andere für die Nachrichtenübertragung zuständig ist. Da auf den Knoten Betriebssystemkerne laufen, können Nachrichten und die Anwendung unabhängig voneinander ablaufen. Der i860XP ist mit 50 MHz getaktet, was eine Peakperformance von 75 MFLOPS und 50 MIPS erlaubt.

Jeder Knoten besitzt außerdem einen Hardware-Monitor, der Angaben über die Auslastungen von Hauptspeicher, Kommunikationsnetzwerk und der Prozes-

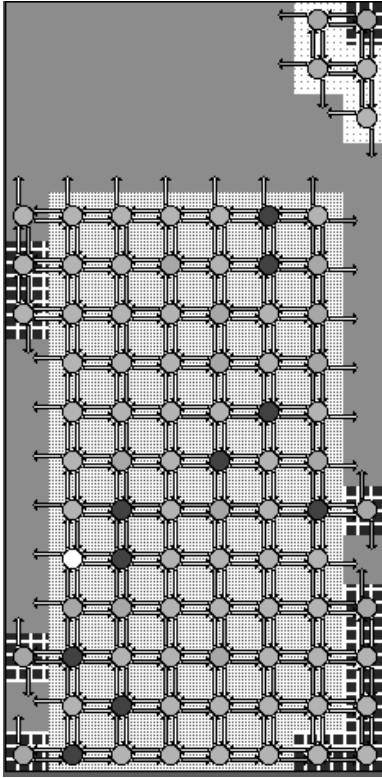


Abbildung 4.1: Aufbau der intel Paragon XP/S

soren machen kann, ohne die Anwendung zu bremsen und damit die Werte zu verfälschen.

4.1.3 Kommunikationsmedium

Das Gitter hat eine Bandbreite von 200 MB/s auf jedem der acht Kanäle eines Knotens. Ein Routing-Chip sorgt dafür, daß Nachrichten weitergeleitet werden, die nicht für den jeweiligen Knoten bestimmt sind. Der beste gemessene Kommunikationsstartup (Latency) beträgt $25 \mu\text{s}$, worin die Zeit für die Abarbeitung der notwendigen Betriebssystemroutinen beinhaltet ist.

4.1.4 Betriebssystem

Das Betriebssystem der Paragon ist OSF/1, ein verteiltes UNIX auf Mach Microkernel-Basis. Die Paragon wird nicht im Front-End/Back-End Betriebsmodus gefahren, anders als die meisten Parallelrechner. Zum Kompilieren wurde der mitgelieferte C-Compiler verwendet, wie auch die von intel speziell für die Paragon entwickelte MPI-Implementierung.

4.2 Cray T3D

4.2.1 Architektur

Die T3D ist der erste Rechner der Firma Cray mit verteiltem Hauptspeicher. Im Rahmen einer Kooperation zwischen dem Institut für Computeranwendungen 3 (ICA 3), der Firma Cray Research und dem RUS wurde im Oktober 1994 eine T3D an der Uni Stuttgart installiert.

Sie ist kein eigenständiges System sondern braucht einen Host für ihren Betrieb, wozu an der Uni Stuttgart eine Cray C-94 verwendet wird. Der Benutzer arbeitet auf dem Hostrechner und benutzt die T3D wie einen Coprozessor. Der verteilte Hauptspeicher kann als Virtual Shared Memory verwendet werden. Bild ?? zeigt den schematischen Aufbau.

4.2.2 Aufbau der Rechenknoten

Das System an der Uni Stuttgart hat 32 sogenannte Processing Elements (PE) mit jeweils 64 MB eigenem Hauptspeicher und einem DEC Alpha 21064 Prozessor. Die Prozessoren sind mit 150 MHz getaktet und erreichen damit eine Peak Performance von 150 MFLOPS, haben also ca. die doppelte Leistung wie der i860XP.

Jeder Rechenknoten besteht aus zwei Processing Elements, einer Netzwerkschnittstelle und einer Block-Transfer-Engine. Die Netzwerkschnittstelle formatiert die Daten, bevor sie auf das Netzwerk geschickt werden. Sie empfängt auch die Nachrichten und leitet sie an die richtige der beiden PEs weiter.

Die Block-Transfer-Engine (BLT) ist ein asynchroner Steuerbaustein, der Daten zwischen den beiden PEs und anderen Rechenknoten umverteilt. Dadurch kann jede PE von anderen Knoten Daten bekommen, ohne diese explizit durch

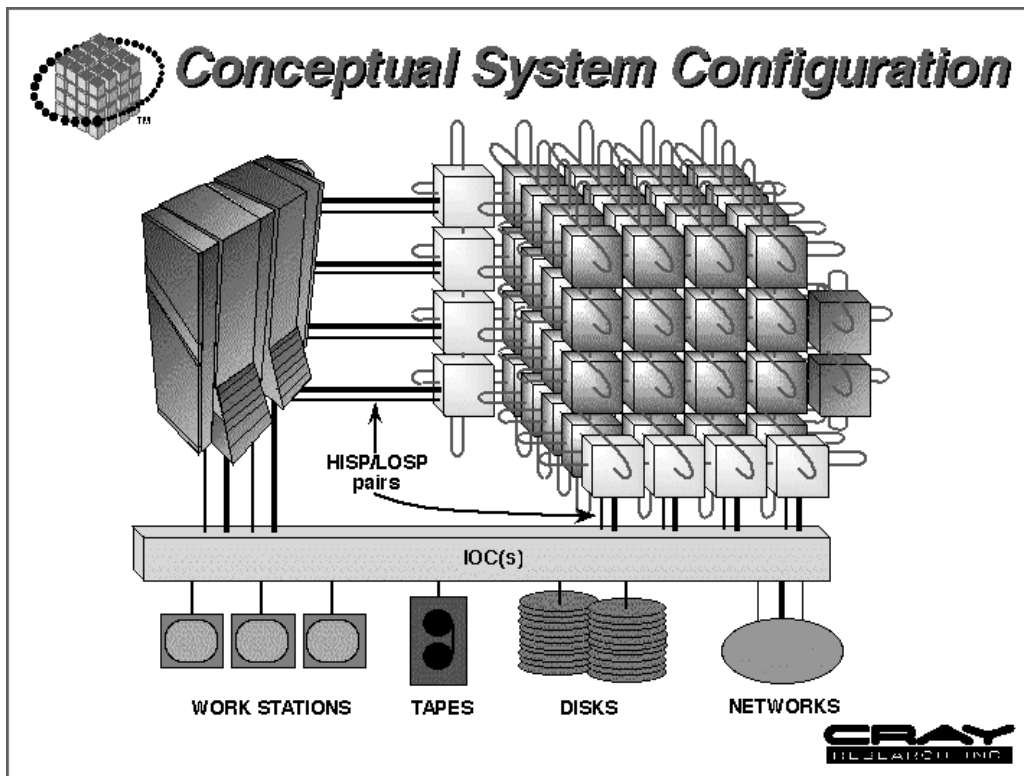


Abbildung 4.2: Aufbau der Cray T3D

Message-Passing anzufordern. Das kann sogar getan werden, bevor die Daten zur Rechnung gebraucht werden, man muß dann also nicht auf sie warten.

4.2.3 Kommunikationsmedium

Das Netzwerk ist als dreidimensionaler Torus aufgebaut und besteht aus Kommunikationsverbindungen und Routern. Die Verwendung der gleichen Switch-Technologie wie für das Prozessor/Memory-Interface der Cray Y-MP ermöglicht es, das Verbindungsnetzwerk mit der gleichen Taktrate wie die Rechenknoten zu betreiben. Dadurch kann auch auf Speicher anderer Knoten schnell zugegriffen werden. Es gibt nur eine globale Uhr in der Maschine und Prozessoren und Netzwerk sind synchronisiert. Über jeden Link des Torus können 300 MB/s übertragen werden.

4.2.4 Software

Auf den Rechenknoten läuft kein Betriebssystem, weshalb auf einem Prozessor nur ein Prozeß laufen kann. Es wurde der C-Compiler auf der C-94 verwendet, der Maschinencode für die T3D erzeugen kann. Die MPI-Implementierung stammt vom Edinburgh Parallel Computing Centre.

4.3 IBM SP2

Das Rechenzentrum der Universität besitzt eine IBM SP2 mit 30 Knoten, von denen aber nur acht für parallele Anwendungen zu Verfügung stehen. Es bestand allerdings die Möglichkeit, die SP2 der Uni Karlsruhe zu benutzen, auf der man 32 Knoten für eine parallele Umgebung erhalten kann.

4.3.1 Architektur

Anders als die meisten Parallelrechner ist die SP2 ein Cluster von eigenständigen Einzelrechnern mit eigenem Hauptspeicher und Festplatte. Außerdem läuft auf jedem Knoten ein vollständiges Betriebssystem. Diese Einzelrechner sind zusätzlich zu einer Ethernetverbindung über den von IBM entwickelten High-Performance-Switch verbunden. Der Karlsruher Rechner hat eine Gesamtleistung von 30 GFLOPS/15 GIPS und verfügt über insgesamt 24 GB Hauptspeicher. Er setzt sich zusammen aus

- 2 Wide nodes mit je 512 MB Hauptspeicher und einer Taktfrequenz von 77 MHz
- 82 Wide nodes mit je 256 MB Hauptspeicher und einer Taktfrequenz von 77 MHz
- 16 Thin2 nodes mit je 128 MB Hauptspeicher und einer Taktfrequenz von 66 MHz

4.3.2 Aufbau der Rechenknoten

Es folgt nur eine Beschreibung der Wide nodes, da diese für das Testbeispiel verwendet wurden. Die Prozessoren der SP2 entstammen der POWER2 Familie,

die die zweite Generation superskalärer RISC-Architekturen von IBM darstellt. Jeder dieser 32-bit Prozessoren ist aus mehreren CMOS Chips mit insgesamt ca. 7 Millionen Transistorfunktionen aufgebaut. Wegen ihrer superskalären Architektur kann eine POWER2 CPU bis zu sieben Instruktionen pro Taktzyklus ausführen, davon vier Fließkommainstruktionen (zweimal $ax + b$). Der große Vorteil der Wide nodes liegt in ihrem 128-bit breiten Datenpfad zum Hauptspeicher, der Übertragungsraten von 2,46 GB/s ermöglicht. Der Prozessor besitzt 32 kB Cache für Instruktionen und 256 kB Cache für Daten.

4.3.3 Kommunikationsmedium

Für seinen High-Performance-Switch hat IBM folgende Leistungsdaten angegeben:

- Bisektionale Bandbreite bis zu 10 GB/s
- Verbindung 160 MB/s bidirektional
- Hardwarelatenzzeit von 500 ns

4.3.4 Software

Die Knoten der SP2 werden unter AIX4.1 betrieben. Es wurde der Compiler mpcc verwendet, der sich auch um die Einbindung der MPI-Funktionalitäten kümmert.

Kapitel 5

Programmbeschreibung

Für die richtige Deutung der Ergebnisse der Zeitmessungen ist es hilfreich, den groben Programmablauf zu kennen. Feinheiten werden erwähnt, wo sie Ergebnisse wesentlich beeinflussen.

5.1 Programmablauf

5.1.1 Initialisierung

Als MPI–Applikation wird das Programm auf allen Knoten gleichzeitig gestartet. Durch MPI wird jedem Knoten, oder Prozeß, eine Nummer zugewiesen, die ihn innerhalb der parallelen Anwendung identifiziert. Der Prozeß mit der Nummer null wird dazu bestimmt, Ein- und Ausgabe zu übernehmen. Er liest die Eingabedatei und schickt die eingelesenen Daten an alle anderen Prozesse.

Von da an laufen auf allen Knoten die gleichen Befehle ab. Jeder Knoten stellt unabhängig von den anderen fest, für welches Teilgebiet er zuständig ist, wie groß dieses ist, und welche Koordinatenwerte es begrenzen. Er stellt fest, in welcher Richtung welche Nachbarknoten liegen und welche Speicherbereiche er ihnen jeweils senden muß. Zudem werden die Speicherbereiche gekennzeichnet, in die jeweils die Nachrichten von Nachbarknoten abgelegt werden müssen.

Bevor mit der eigentlichen Berechnung angefangen werden kann, müssen die richtigen Anfangswerte gesetzt werden. Jeder Knoten stellt fest, ob eine oder mehrere der Oberflächen seines Rechengebietes einen Rand des Gesamtgebietes darstellen; sollte das der Fall sein, werden auf den zugehörigen Rechengitterpunkten

die Werte entsprechend der Randbedingung gesetzt. Alle anderen Rechengitterpunkte werden auf Null gesetzt. Da die Punkte, deren Werte durch Randbedingungen vorgeschrieben sind, während der Rechnung nie überschrieben werden, muß die Randbedingung während der Rechnung auch nicht mehr angewendet werden.

Der folgende Programmblock wird so lange wiederholt, bis der Unterschied zwischen der numerischen und analytischen Lösung innerhalb der gewünschten Genauigkeit liegt.

5.1.2 Schleifenrumpf

Zu Beginn jedes Zeitschritts wird der Empfang der Nachrichten von den Nachbarknoten vorbereitet. Bei Verwendung der unter ?? beschriebenen immediate Routinen bietet das einen Vorteil, weil bei Aufruf der Senderoutinen die Empfänger vorbereitet sind und die Kommunikation sofort starten kann.

Anschließend erfolgen die Funktionsaufrufe, um die Randwerte jedes Knotens an die jeweiligen Nachbarn zu schicken.

Jetzt werden die Funktionswerte des nächsten Zeitschritts für die Gitterpunkte berechnet, die im Innern des Rechengebietes jedes Knotens liegen und nicht von den Randwerten abhängen.

Anschließend warten alle Prozesse, bis der Empfang der Nachrichten von allen direkten Nachbarn abgeschlossen ist. Danach können die neuen Werte auf den Randpunkten des Rechengebietes berechnet werden.

An dieser Stelle im Programm hat der Knoten die Berechnung des Zeitschritts beendet. Er könnte jetzt mit der Berechnung des nächsten Schrittes beginnen. Um zu verhindern, daß Speicherbereiche mit Ergebnissen des neuen Zeitschritts überschrieben werden, bevor die Daten des alten Zeitschrittes verschickt worden sind, wird noch der Abschluß aller Versendeoperationen abgewartet. Damit wird sichergestellt, daß jeder Knoten am selben Zeitschritt wie seine Nachbarn arbeitet.

Dieses Synchronisieren kann auch global erfolgen, wobei abgewartet wird, bis alle Knoten die Berechnung des gleichen Zeitschritts abgeschlossen haben. Dabei wird allerdings immer auf den langsamsten Knoten gewartet, während sich im oben beschriebenen Fall der lokalen Synchronisierung Unterschiede in der Bearbeitungszeit der einzelnen Knoten für einen Zeitschritt ausgleichen können.

Es ist schon an dieser Stelle offensichtlich, daß lokales Synchronisieren einen schnelleren Ablauf zur Folge haben sollte, als globales.

Ist das Versenden von Nachrichten abgeschlossen, beginnt der nächste Zeitschritt.

Da dieser Ablauf für alle Knoten und ihre jeweiligen Nachbarn gleich ist, wird dafür gesorgt, daß einzelne Prozesse unabhängig von den anderen mit neuen Zeitschritten beginnen. Auf diese Weise synchronisieren sie.

5.1.3 Ausgabe

Sobald die Funktionswerte aller Rechengitterpunkte auf einem Knoten im Vergleich zur analytischen Lösung innerhalb der geforderten Genauigkeit liegen, wertet der Prozess seine Zeitmessungen aus und schickt die Ergebnisse an den Knoten, von dem er die Eingabedaten erhalten hatte. Dieser wartet auf die Werte von allen Prozessen und schreibt sie in eine Ausgabedatei.

Danach melden sich alle Prozesse bei MPI ab und terminieren.

5.2 Numerische Realisierung

Das Problem, das im Testfall gelöst werden soll, ist folgende Vereinfachung der Wärmeleitgleichung:

$$\frac{\partial f}{\partial t} = \nabla^2(f) \quad (5.1)$$

oder

$$\frac{\partial f}{\partial t} = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} + \frac{\partial^2 f}{\partial z^2} \quad (5.2)$$

Die Diskretisierung durch Vorwärtsdifferenz für die Zeit ergibt:

$$\frac{\partial f}{\partial t} \Rightarrow \frac{f_{i,j,k}^{n+1} - f_{i,j,k}^n}{\Delta t} \quad (5.3)$$

Eine Diskretisierung der Ableitungen nach den Koordinaten durch Zentralfdifferenzen ergibt:

$$\frac{\partial^2 f}{\partial x^2} \Rightarrow \frac{f_{i+1,j,k}^n + f_{i-1,j,k}^n - 2f_{i,j,k}^n}{\Delta x^2} \quad (5.4)$$

$$\frac{\partial^2 f}{\partial y^2} \Rightarrow \frac{f_{i,j+1,k}^n + f_{i,j-1,k}^n - 2f_{i,j,k}^n}{\Delta y^2} \quad (5.5)$$

$$\frac{\partial^2 f}{\partial z^2} \Rightarrow \frac{f_{i,j,k+1}^n + f_{i,j,k-1}^n - 2f_{i,j,k}^n}{\Delta z^2} \quad (5.6)$$

Dabei kennzeichnen die Indizes i, j, k der Funktionswerte f die Koordinatenrichtungen x, y , und z ; der hochgestellte Index n bezeichnet den aktuellen Zeitschritt.

Aus diesen Termen läßt sich die diskretisierte Gleichung aufstellen:

$$\begin{aligned} f_{i,j,k}^{n+1} = f_{i,j,k}^n &+ \frac{\Delta t}{\Delta x^2} (f_{i+1,j,k}^n + f_{i-1,j,k}^n - 2f_{i,j,k}^n) \\ &+ \frac{\Delta t}{\Delta y^2} (f_{i,j+1,k}^n + f_{i,j-1,k}^n - 2f_{i,j,k}^n) \\ &+ \frac{\Delta t}{\Delta z^2} (f_{i,j,k+1}^n + f_{i,j,k-1}^n - 2f_{i,j,k}^n) \end{aligned} \quad (5.7)$$

Es handelt sich hierbei um eine parabolische Differentialgleichung, sodaß bei expliziter Zeitintegration folgende Einschränkung für die Größe des Zeitschritts Δt gilt:

$$\Delta t \leq \frac{h^2}{2n} \quad (5.8)$$

wobei h der Abstand zwischen zwei benachbarten Gitterpunkten und n die Anzahl der Dimensionen sind, in unserem Fall also $n = 3$.

Laut Aufgabenstellung ist der Einheitswürfel das Rechengebiet, und die Anzahl der Rechengitterpunkte ist in allen Koordinatenrichtungen gleich. Für h gilt bei den gewählten Diskretisierungen

$$h = \frac{1}{30}, \quad \frac{1}{76} \rightarrow \Delta t \leq \frac{1}{5400}, \quad \frac{1}{34656} \quad (5.9)$$

Wegen numerischer Ungenauigkeiten bei der Berechnung des Zeitschritts werden nur 95 % des maximal möglichen Zeitschritts verwendet, damit keine Instabilitäten auftreten.

Als Anfangsbedingung werden die Funktionswerte aller im Inneren liegenden Gitterpunkte auf Null gesetzt. Die Randbedingung wird durch eine Zuweisung der Funktionswerte an die Rechengitterpunkte auf den Außenflächen des Gebiets eingebracht. Im Testbeispiel gilt für die Randpunkte $f(x, y, z) = x$, d.h. daß der Funktionswert dem Wert der x -Koordinate entspricht.

5.3 Einfluß der Diskretisierung

Wenn die Lösung der Zeitintegration einen Zustand erreicht hat, in dem sich die Funktionswerte nicht mehr ändern, wird dieser Zustand als Gleichgewichtszustand bezeichnet und die dazu gehörende Lösung als Gleichgewichtslösung.

Wenn sich die numerische Lösung bis auf die geforderte Genauigkeit an die bekannte analytische Gleichgewichtslösung angenähert hat, wird die Berechnung beendet. Da sich die Oberflächenwerte immer mit gleicher Geschwindigkeit in den Würfel ausbreiten, wird der Gleichgewichtszustand unabhängig von der Diskretisierung auch immer zur selben physikalischen Gesamtzeit erreicht. Diese Überlegung ermöglicht eine Abschätzung der erforderlichen Rechenzeit bezüglich einer Referenz. Die Referenzzeit erhält man, indem man ein kleines Testgebiet berechnen läßt und die benötigte Zeit zum Vergleich mit der gewünschten Gebietsgröße benutzt.

Der Gleichgewichtszustand wird zur konstanten Gesamtzeit T erreicht. Dazu werden $numsteps$ Zeitschritte benötigt. Wegen der Zeitschrittbegrenzung durch Gleichung (??) läßt sich $numsteps$ wie folgt berechnen:

$$numsteps \geq \frac{T}{\Delta t_{max}} \geq \frac{2nT}{h^2} \geq \frac{C}{h^2} \quad (5.10)$$

Dabei ist C nur ein Proportionalfaktor, der alle Konstanten aufnimmt. Da in diesem Beispiel nur auf dem Einheitswürfel gerechnet wird und die Anzahl k der Gitterpunkte laut Aufgabestellung in allen Koordinatenrichtungen gleich ist, läßt sich mit der Wahl von $\Delta x = h$ als Parameter für die Problemgröße eine Vereinfachung vornehmen.

$$h = \Delta x = \frac{1}{k-1}$$

Eingesetzt in Gleichung ?? ergibt sich:

$$numsteps \geq \frac{C}{\Delta x^2} = C(k-1)^2 \quad (5.11)$$

Die Anzahl der zu berechnenden Punkte p ist wegen der gleichen Anzahl k von Gitterpunkten in jeder Koordinatenrichtung:

$$p = (k-2)^3 \quad (5.12)$$

Die Punkte an den Rändern müssen nicht berechnet werden, da sie als Randbedingung schon vorliegen.

Die Gesamtgröße S des Problems ist proportional zur Anzahl der zu berechnenden Punkte p und zur Anzahl der erforderlichen Zeitschritte $numsteps$ bis zur Erreichung des Gleichgewichtszustands. Es gilt

$$S = p * numsteps \quad (5.13)$$

Mit den Gleichungen (??) und (??) ergibt sich aus (??) für die Gesamtgröße S des Problems:

$$S = C(k - 1)^2(k - 2)^3$$

Aus einem Testlauf läßt sich so abschätzen, wie viel Rechenzeit für eine andere Diskretisierung nötig ist. Kennzeichnende Größe dafür ist die Anzahl der Rechengitterpunkte pro Koordinatenrichtung.

$$Faktor = \left(\frac{k - 1}{k_b - 1}\right)^2 \left(\frac{k - 2}{k_b - 2}\right)^3 \sim \left(\frac{k}{k_b}\right)^5$$

Der Wert k_b kennzeichnet die Diskretisierung des Gebietes für die Referenzzeitmessung. Bezogen auf die Anzahl der Gitterpunkte in jeder Koordinatenrichtung steigt die erforderliche Rechenzeit mit der 5. Potenz. Das bedeutet, daß eine Verdoppelung der Anzahl der Rechengitterpunkte pro Koordinatenrichtung ungefähr die zweiunddreißigfache erforderliche Rechenzeit bis zur Erreichung des Gleichgewichtszustandes zur Folge hat.

5.4 Einfluß der Gebietszerlegung

Das Rechenproblem soll möglichst gleichmäßig auf alle verwendeten Prozessoren verteilt werden. Der Algorithmus zur Bestimmung der Prozeßtopologie arbeitet folgendermaßen:

1. Festlegung der Prozeßtopologie.
 - Primfaktorenzerlegung der Prozessoranzahl.
 - Falls höchstens drei Primfaktoren auftreten, sind dies die jeweiligen Prozessorzahlen pro Koordinatenrichtung. Die Anzahl der Dimensionen der Prozeßtopologie entspricht der Anzahl der Primfaktoren.
 - Wenn mehr als drei Primfaktoren auftreten, werden daraus drei möglichst nah beieinander liegende Faktoren erzeugt, die dann die jeweiligen Prozessorzahlen für jede Koordinatenrichtung bilden. Die Prozeßtopologie ist dreidimensional.

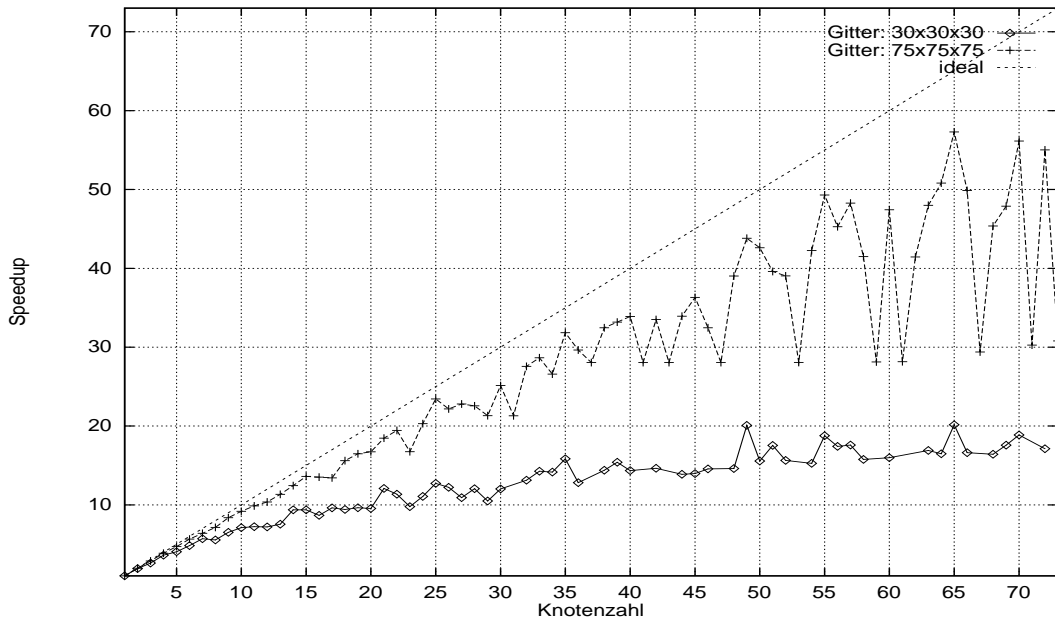
2. Verteilung der Rechengitterpunkte auf die einzelnen Prozessoren.

- Die Anzahl der Rechengitterpunkte pro Koordinatenrichtung werden nun so auf die einzelnen Prozessoren verteilt, daß alle Rechenknoten zunächst einmal gleich viele bekommen und die Randknoten jeweils einen mehr, da Randpunkte nicht berechnet werden müssen.
- Alle noch übrigen Rechengitterpunkte werden danach entlang der Koordinatenachsen einzeln von außen nach innen den Knoten zugeteilt, bis keine mehr übrig sind.

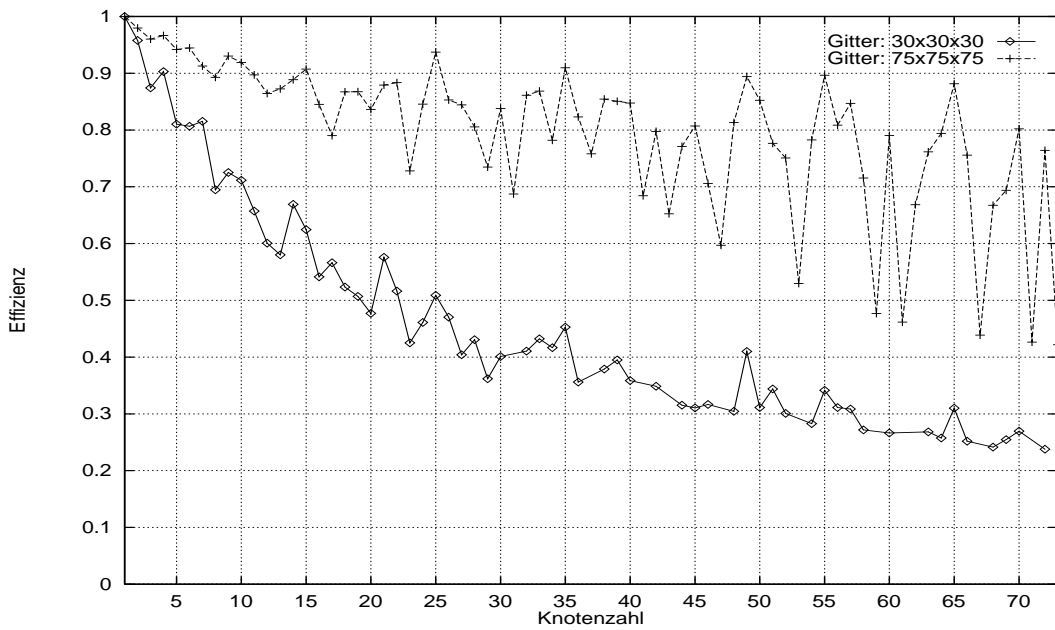
Ein Ungleichgewicht der jeweils erforderlichen Rechenzeiten zweier Knoten ergibt sich dadurch, daß die Anzahl der zu berechnenden Rechengitterpunkte pro Koordinatenrichtung jeweils um eins unterschiedlich sein kann.

Die Lage der jeweiligen Prozesse innerhalb der Prozeßtopologie entscheidet über die Anzahl der Oberflächen, über die kommuniziert werden muß. Deren Anzahl kann Werte von eins bis sechs annehmen. Bei der hier betrachteten Problemgröße ist sie hauptbestimmend für die erforderlichen Laufzeiten. Ein Knoten im Innern der Topologie, der bis zu sechs Nachbarn haben kann, braucht für die Kommunikation deutlich länger als ein Knoten in einer Ecke der Topologie, der höchstens drei Nachbarn haben kann.

In Bild ?? kann man gut erkennen, daß für Primzahlen als Rechenknoten-zahlen die Gebietszerlegung keine gute Effizienz ermöglicht. Für zwei- und dreidimensionale Zerlegungen jedoch läuft die parallele Version des Programms gut. Dabei zeigt sich, daß Kommunikation über viele Oberflächen die Gesamtlaufzeit ebenfalls anheben kann. Für detaillierte Vergleichstests wurden als Knotenzahlen Zweierpotenzen gewählt, die auf allen Architekturen verfügbar sind. Diese Beschränkung wird durch die Cray-T3D auferlegt.



(a) Speedup



(b) Effizienz

Abbildung 5.1: Alle Knotenzahlen auf intel Paragon XP/S

Kapitel 6

Ergebnisse

Im Rahmen dieser Studienarbeit wurden zwei Themenbereiche untersucht. Der erste ist die Untersuchung, welcher Unterschied sich zwischen zwei Algorithmen zeigt, von denen der eine globales, der andere lokales Synchronisieren verwendet. Dieser Unterschied wird anhand der Laufzeiten gezeigt, die beide Algorithmen für die Berechnung der Gleichgewichtslösung auf einem Gebiet von $30 \times 30 \times 30$ brauchen. Die Nachrichten wurden dabei nicht vergrößert. Die zweite Untersuchung ist der eigentliche Vergleich der verschiedenen Architekturen untereinander. Die Auswahl der vier untersuchten Problemfälle wurde bereits in Kapitel ?? beschrieben. Außerdem werden auf den einzelnen Rechnern die verschiedenen Testfälle untereinander verglichen. Daraus wird leichter ersichtlich, welche Änderung in der Anforderung eine deutliche Auswirkung auf die erzielbaren Leistungen mit den einzelnen Rechnern haben.

Aus Gründen der Darstellung wird in manchen Schaubildern auf die Ergebnisse mit 64 Knoten der intel Paragon verzichtet, in den Tabellen im Anhang sind aber alle Ergebnisse aufgelistet.

Als Ergebnis der Untersuchungen werden jeweils die Laufzeiten, der Speedup und die bei der Parallelisierung erreichte Effizienz angegeben. Mit Speedup und Effizienz ist folgendes gemeint:

Der Speedup gibt an, um welchen Faktor ein Programm auf p Prozessoren schneller war, als auf einem einzelnen Prozessor. Der optimale Speedup ist immer so groß, wie die Anzahl der verwendeten Prozessoren. Die Effizienz gibt an, wie stark die einzelnen Prozessoren ausgelastet waren. Der optimale Wert der Effizienz ist 1 (100 %). Eine ausführlichere Beschreibung der beiden Größen und ihrer Bedeutung findet man in [?].

Eine weitere aussagekräftige Untersuchung hätte man so durchführen können, daß man bei Verwendung mehrerer Knoten das Rechengebiet jeweils so anpaßt, daß die Rechenanforderung an alle Knoten für unterschiedliche Knotenzahlen jeweils gleich groß ist. Solche Scale-Up Untersuchungen haben einen größeren Bezug zur Praxis, in der man Parallelrechner in erster Linie zur Lösung so großer Probleme einsetzt, daß man sie nicht mehr auf einem einzigen Knoten lösen könnte. Dabei werden die einzelnen Knoten wesentlich effizienter eingesetzt, als bei Speedup-Untersuchungen.

Aus Zeitgründen war eine solche Untersuchung im Rahmen dieser Arbeit nicht durchführbar. Die Paragon braucht für die Diskretisierung mit $76 \times 76 \times 76$ Gitterpunkte mit einem Knoten ca. fünf Stunden. Bei Verwendung mehrerer Knoten wird die Problemgröße proportional zur Anzahl der Knoten ebenfalls vergrößert, was zur Folge hat, daß die Laufzeiten wieder mindestens fünf Stunden betragen, das aber mit mehreren Knoten gleichzeitig. Für die Untersuchung eines einzigen der Beispielfälle auf einem Rechner wären am Beispiel Paragon also über 600 Rechenstunden nötig gewesen. Ein Vergleich aller ausgesuchten Testfälle auf den drei Rechnern erfordert eine Gesamtrechnzeit, die für diese Untersuchung nicht zu rechtfertigen ist.

6.1 Globale und lokale Synchronisierung

Das Rechengebiet wird zu Beginn der Rechnung in Teilgebiete zerlegt und auf verschiedene Prozessoren verteilt. Jeder Prozessor ist daher zu bestimmten Zeitpunkten im Berechnungsablauf auf Daten seiner Nachbarn angewiesen, die er zur Berechnung der Werte auf den Randpunkten seines Teilgebietes braucht. Damit aber alle Prozessoren immer Daten des gleichen Iterationsschritts zur Berechnung des nächsten Schrittes verwenden und nicht asynchron werden, müssen sie an bestimmten Stellen im Programmablauf synchronisieren.

Diese Synchronisierung ist auch erforderlich, um eine globale Operation auszuführen, wie zum Beispiel die Berechnung einer Fehlergröße über das gesamte, verteilte Rechengebiet als Abbruchbedingung für den Algorithmus.

Am einfachsten läßt sich unter MPI globales Synchronisieren implementieren. An bestimmten Stellen im Berechnungsablauf wird auf alle Prozesse gewartet, der langsamste bestimmt in diesem Fall die Zeitdauer. Für den Datenaustausch würde das bedeuten, daß jeder Prozeß warten muß, bis auch alle anderen Prozesse den Datenaustausch mit ihren jeweils direkten Nachbarn in der Prozeßtopologie

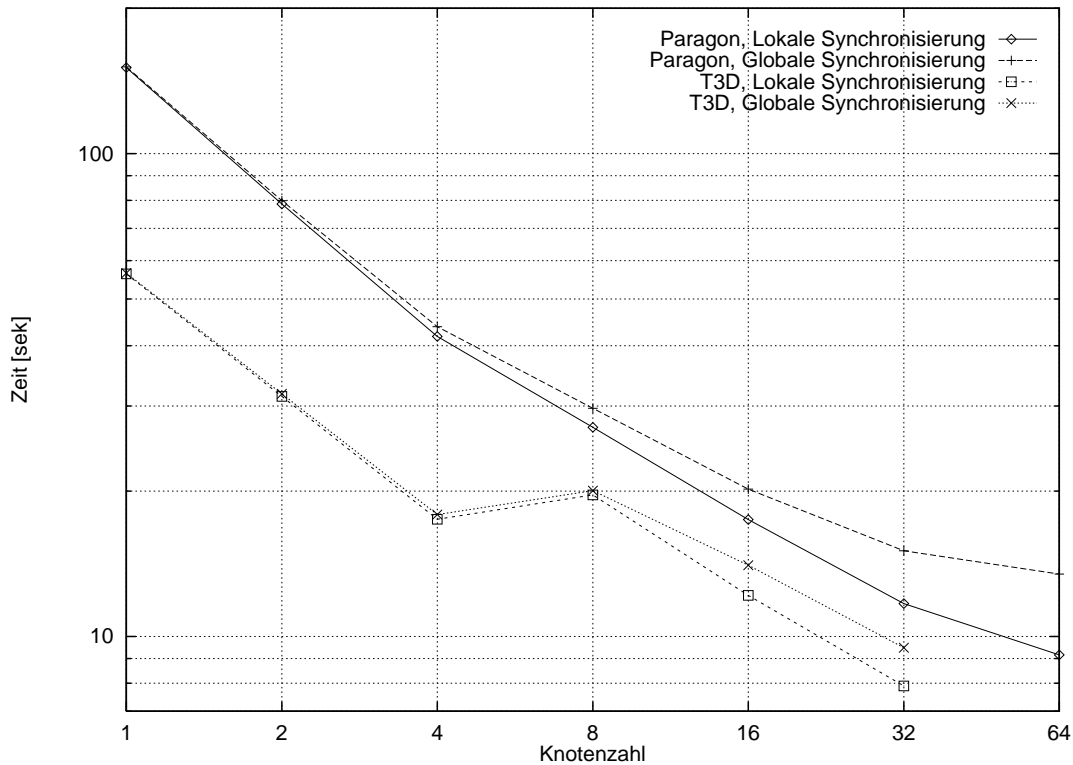


Abbildung 6.1: Laufzeit für globale und lokale Synchronisierung

beendet haben. Deshalb muß ein Prozeß auch auf jene Prozesse warten, mit denen er gar nicht kommuniziert.

Diese Art des Synchronisierens zwingt einige Prozesse zu unnötigen Wartezeiten und verschwendet damit Rechenleistung. In vielen Fällen läßt sich das Synchronisieren auch lokal organisieren, wobei einzelne Prozesse nur mit ihren direkten Nachbarn innerhalb der Prozeßtopologie synchronisieren.

Im Falle des Datenaustausches über Teilgebietsgrenzen hinweg ist in diesem Fall nur erforderlich, daß ein Prozeß auf den Abschluß der Kommunikation mit seinen direkten Nachbarn wartet, und nicht auf alle anderen Prozesse. Dadurch können sich Unregelmäßigkeiten in der Kommunikation ausgleichen, die in den jeweiligen Zeitschritten auftreten.

Damit die Berechnung nicht endlos abläuft, sondern bei Erreichen des gewünschten Ergebnisses beendet wird, ist ein Abbruchkriterium für den Algorithmus notwendig. Die Werte der numerischen Lösung konvergieren theoretisch gegen die analytische Lösung beliebig genau, in der Praxis bis zum Erreichen der Maschinengenauigkeit. Im Beispiel ist es nicht notwendig, den Rechenlauf kon-

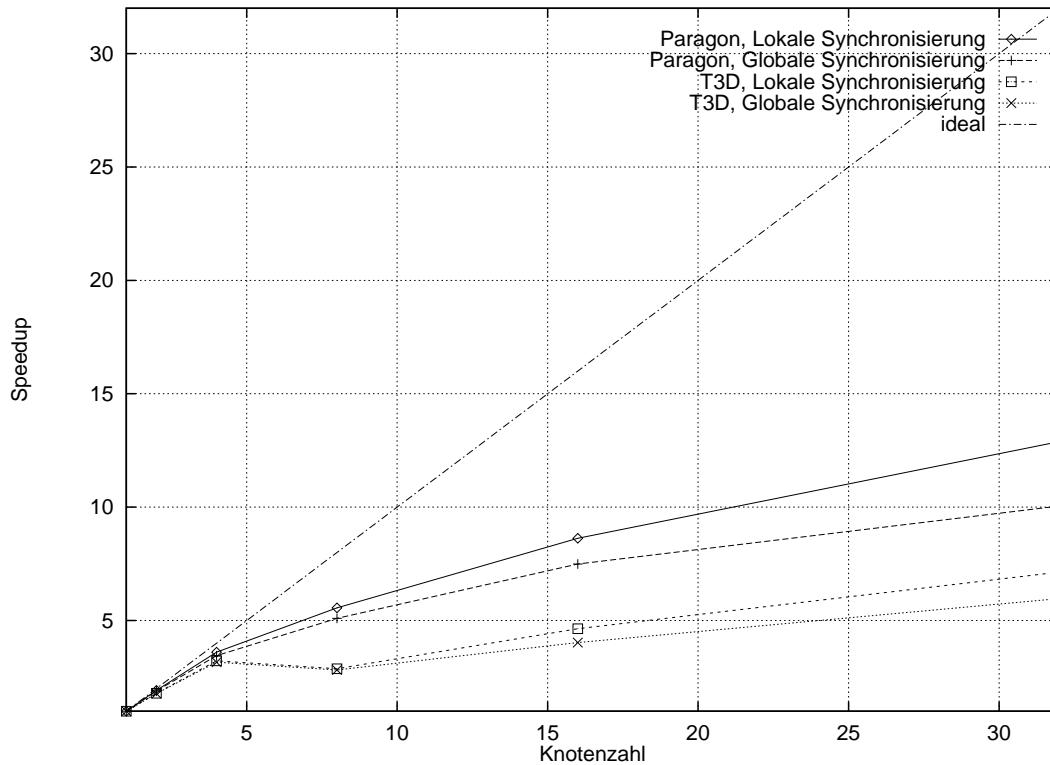


Abbildung 6.2: Speedup für globale und lokale Synchronisierung

vergieren zu lassen, bis sich die numerische Lösung nicht mehr ändert, sondern es wurde festgelegt, daß der Unterschied zwischen der numerischen und der exakten Lösung nicht größer als 10^{-6} sein darf. Diese Begrenzung liefert eine ausreichend genaue Lösung mit vertretbarem Rechenaufwand.

Es gibt mehrere Möglichkeiten, ein Abbruchkriterium zu definieren. Eine gebräuchliche Methode ist die Verwendung einer Fehlergröße, die nach jedem Zeitschritt berechnet und überprüft werden muß. Wenn für die Berechnung dieses Wertes Daten von allen Prozessen nötig sind, spricht man von einer globalen Fehlergröße, wenn jeder Prozeß einen Wert nur für sich selbst berechnet, von einem lokalen Fehler. Bei der Berechnung von globalen Fehlern ist es erforderlich, daß alle Prozesse ihren Anteil am Gesamtfehler in die Berechnung einbringen und danach auf das Ergebnis warten. Anhand des Ergebnisses wird dann die Entscheidung gefällt, ob die gewünschte Genauigkeit erreicht ist, oder nicht.

Eine andere Möglichkeit für ein Abbruchkriterium ist die Verwendung von lokalen Fehlern, die nur eine Aussage über die Güte der Lösung auf dem Teilgebiet machen, das ein bestimmter Prozeß bearbeitet. Wenn ein Prozeß erkannt hat,

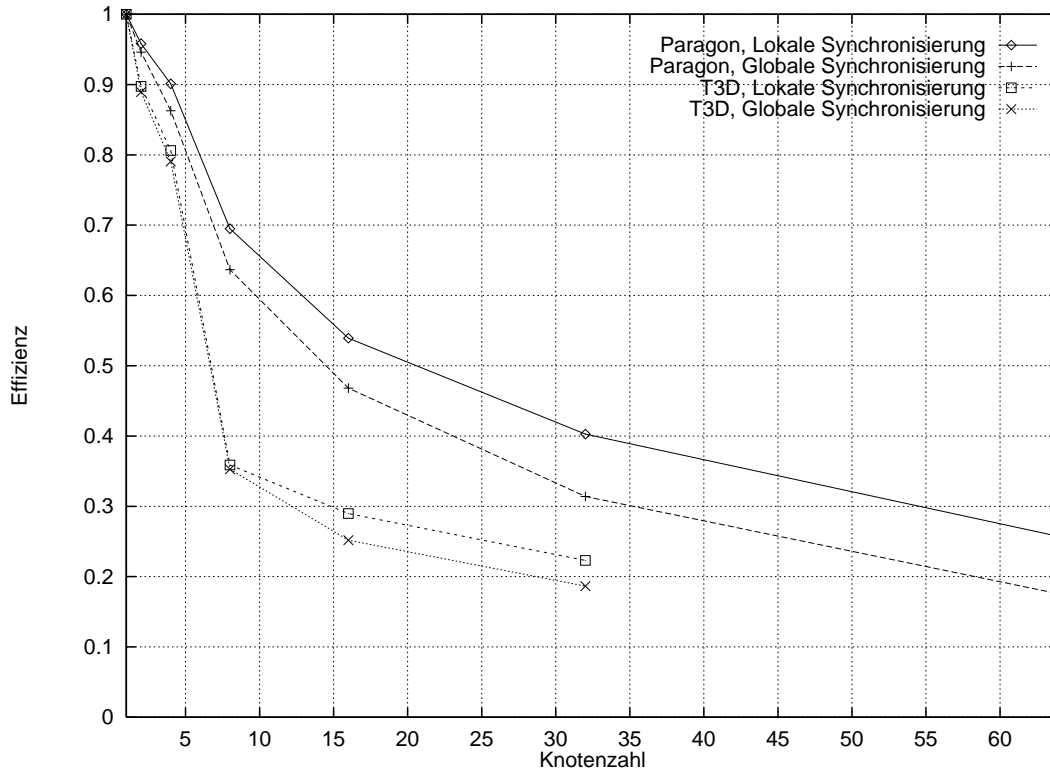


Abbildung 6.3: Effizienz für globale und lokale Synchronisierung

daß sein lokaler Fehler innerhalb der geforderten Genauigkeit liegt, besteht für ihn kein Grund mehr, die Rechnung weiterzuführen und damit Ressourcen zu belegen, die von anderen Prozessen sinnvoller genutzt werden könnten. Er kann seinen Nachbarn in der Prozeßtopologie signalisieren, daß er mit der Rechnung fertig ist und sich von nun an aus der Kommunikation heraushält. Dadurch wird das Kommunikationsmedium weniger belastet, was den anderen Prozessen zugute kommt.

Gerade bei Randwertproblemen, bei denen die Lösung vom Rand des Gebietes nach innen fließt, erreichen die Prozesse mit Teilgebieten am Rand die erforderliche Genauigkeit wesentlich eher als die Prozesse mit im Innern liegenden Teilbereichen. Sie können die Zeitintegration deshalb eher beenden.

In den Bildern ?? sind die Auswirkungen der beiden unterschiedlichen Synchronisierungsarten für die Cray-T3D und die intel-Paragon zu sehen. Die Diskretisierung bestand aus 30 Gitterpunkten pro Koordinatenrichtung und die Nachrichten wurden nicht künstlich vergrößert.

Man kann erkennen, daß die Rechenläufe mit lokalem Synchronisieren auf

beiden verwendeten Rechnern schneller fertig waren, als die Läufe mit globalem Synchronisieren. Es ist bemerkenswert, daß sich dieses Verhalten zeigt, obwohl beide Rechner gute Kommunikationssysteme haben. Bei Rechnern mit schlechten Kommunikationssystemen und speziell bei Workstation-Clustern ist zu erwarten, daß der Zeitvorteil durch lokales Synchronisieren noch deutlicher ausfallen würde.

Es zeigt sich außerdem, daß der Zeitgewinn mit steigenden Knotenzahlen größer wird. Das ist leicht zu verstehen, denn die Zahl der direkten Nachbarn für jeden Prozeß bleibt ab einer bestimmten Knotenzahl konstant, während die Gesamtzahl der Prozesse immer noch zunehmen kann. Beim lokalen Synchronisieren ist die Zahl der direkten Nachbarn bestimmend für die Kommunikationszeit, beim globalen Synchronisieren die Gesamtzahl aller Prozesse.

6.2 Vergleich der Architekturen

In diesem Abschnitt wird gezeigt, welche Leistungen die unterschiedlichen Architekturen für zwei jeweils feststehende Problemgrößen bieten. Das kleine Problem ist ein Gitter mit $30 \times 30 \times 30 = 27.000$ Rechenpunkten, was eine vorgeschlagene Parametergröße aus dem Beispielfall ist. Das Große Problem mit $76 \times 76 \times 76 = 438.976$ Gitterpunkten paßt gerade noch auf einen einzelnen Knoten der intel Paragon, der Maschine mit dem kleinsten Speicher pro Rechenknoten (32 MB). Die reine Rechenanforderung durch die feine Diskretisierung beträgt das 120-fache der Anforderung durch die grobe Diskretisierung. Darin ist die Kommunikation jedoch nicht enthalten. Nachfolgend ist mit Kommunikationszeit immer der Zeitanteil der Kommunikation gemeint, der nicht durch Rechnungen überlagert werden kann und daher spürbar auftritt.

6.2.1 Grobe Diskretisierung (30 x 30 x 30 Gitterpunkte)

Diese Rechenanforderung ist für Parallelrechner eigentlich zu klein. Ein einzelner Knoten der jeweiligen Rechner braucht für die Bearbeitung nur zwischen 30 und 150 Sekunden. Eine parallele Bearbeitung ist deshalb nicht nötig. Man kann aber anhand dieses Beispiels auch mit kurzen Laufzeiten schon einige Aussagen über die jeweiligen Rechner machen. Die Verläufe von Laufzeit, Speedup und Effizienz sind in den Bildern ?? bis ?? zu sehen.

Einfache Nachrichtenlänge

In Bild ?? ist deutlich zu erkennen, daß die SP2 mit den neuen POWER2-Prozessoren die schnellsten und die Paragon die langsamsten Einzelprozessoren hat. Verglichen mit der SP2 braucht die T3D die doppelte, die Paragon die fünffache Laufzeit auf einem Knoten. Bei der Kommunikationszeit erreicht die SP2 die besten Werte für alle Knotenzahlen, allerdings nimmt die Kommunikationszeit ab 8 Knoten mit steigenden Knotenzahlen ebenfalls zu. Die Kommunikationszeiten der T3D liegen für die meisten Knotenzahlen zwischen denen von SP2 und Paragon, bei Verwendung von 8 und 16 Knoten braucht sie aber am längsten. Der deutliche Anstieg der Kommunikationszeit beim Übergang von 4 auf 8 Knoten läßt sich zum Teil dadurch erklären, daß die Gebietszerlegung für 8 Knoten dreidimensional, für 4 Knoten nur zweidimensional erfolgt. Die Zahl der Nachrichten ist bei Verwendung von 8 Knoten deshalb deutlich größer. Die Paragon braucht in den meisten Fällen am längsten für die Kommunikation, bei Verwendung von 8 oder 16 Knoten ist sie jedoch schneller als die T3D. Für Knotenzahlen größer als 16 nimmt die Kommunikationszeit ab. Dies läßt sich dadurch erklären, daß die Menge der zu übertragenden Information mit kleiner werdenden Teilgebieten ebenfalls abnimmt.

Bei Verwendung von 32 Knoten der SP2 ist die Zeit für Kommunikation fünfmal so groß wie für Rechnungen. Dieser Kommunikationsoverhead ist verantwortlich dafür, daß die Gesamtzeit für 32 Knoten größer ist, als für 16 oder gar 8 Knoten. Trotzdem ist die SP2 für alle Knotenzahlen schneller, als die anderen beiden Rechner. Bei der T3D dauert der Rechenlauf mit 8 Knoten länger als der mit 4 Knoten. Ansonsten bedeutet die Verwendung von mehr Prozessoren der T3D auch einen Laufzeitgewinn. Die Paragon liefert im Vergleich der drei Rechner für alle Knotenzahlen die längsten Laufzeiten, allerdings bringt die Verwendung von mehr Knoten immer einen Zeitgewinn, was als gutes Ergebnis für das Netzwerk gewertet werden kann.

Bei Verwendung von 32 Knoten sinkt das Laufzeitverhältnis Paragon:SP2 von 5:1 auf 2:1, das Verhältnis von T3D:SP2 bleibt in etwa gleich. Die Kommunikationszeit wird bei der SP2 ab 16, bei der T3D ab 8, und bei der Paragon ab 32 Knoten größer als die reine Rechenzeit. Im Vergleich zu den beiden anderen Rechnern macht eine parallele Bearbeitung dieses Beispiels auf der Paragon am meisten Sinn. Sie wird durch ihr Kommunikationsmedium nicht behindert, was die relativ guten Werte für Speedup und Effizienz auch zeigen.

Zehnfache Nachrichtenlänge

Bei diesem Beispiel hat die Kommunikation eine noch größere Bedeutung, die Ergebniswerte zeigen deshalb im wesentlichen die Kommunikationsleistung.

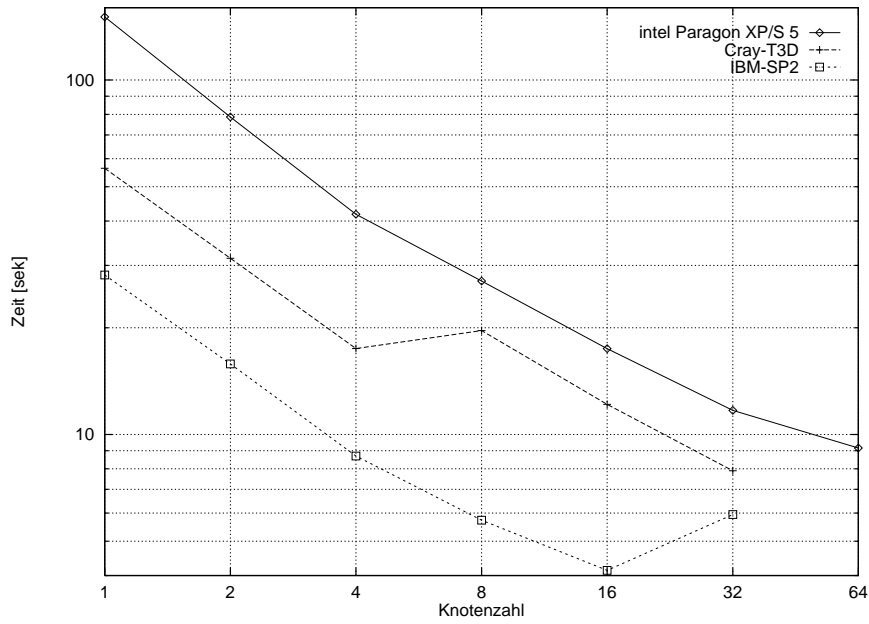
Die reine Rechenleistung wird durch die längeren Nachrichten nicht beeinflusst, deshalb gibt es dafür auch keine neuen Ergebnisse.

Die SP2 hat wieder die kürzesten Zeiten für Kommunikation. Während die Gebietszerlegung an Dimensionen zunimmt, also bis zur Verwendung von 8 Knoten, nehmen die Kommunikationszeiten ebenfalls zu. Sobald drei Dimensionen erreicht sind, nimmt mit zunehmenden Knotenzahlen die Zeit für Kommunikation wieder ab. Die Kommunikationszeit steigt für die zehnfache Nachrichtenlänge um den Faktor 4. Im Vergleich zum vorigen Fall, mit der gleichen Diskretisierung aber kürzeren Nachrichten, scheint die SP2 mit langen Nachrichten besser zurechtzukommen als mit vielen Nachrichten. Bis zu 4 Knoten ist die T3D mit ihrer Kommunikation schneller als die Paragon, ab 8 Knoten langsamer. Sie zeigt, wie die anderen Rechner auch, einen Anstieg der Kommunikationszeit bis 8 Knoten, aber deutlich größer, als die beiden anderen Rechner. Die zehnfache Nachrichtenlänge führt auch wirklich zur zehnfachen Kommunikationszeit. Das deutet darauf hin, daß nicht nur viele, sondern auch große Nachrichten für die T3D ein Problem darstellen. Die Paragon braucht für die Übertragung der zehnfache Nachrichten nur etwa fünfmal so lang, wie für normale Nachrichten. Die Vergleiche gelten für die Verwendung von 32 Knoten auf jedem Rechner.

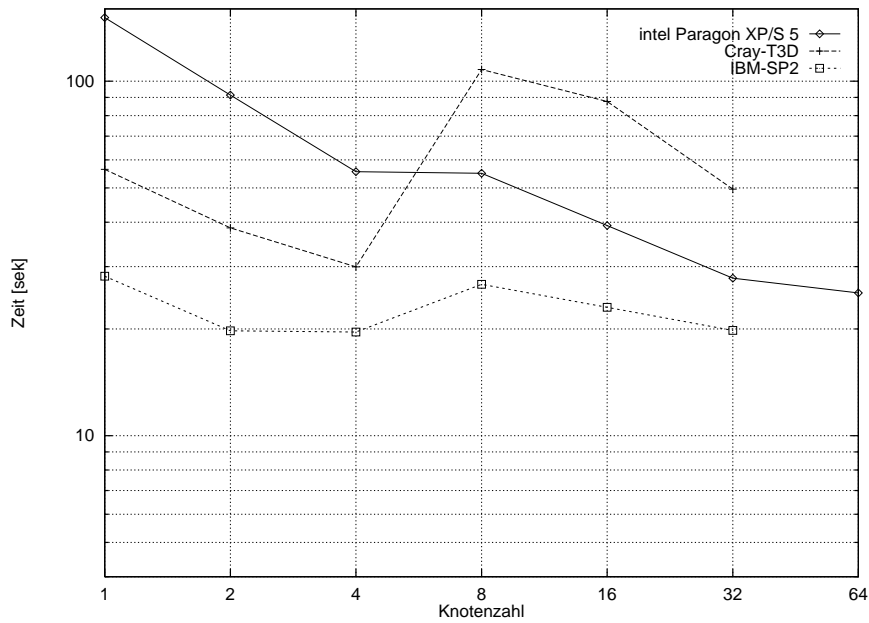
Die SP2 hat für alle Knotenzahlen die kürzesten Laufzeiten im Vergleich. Da schon bei vier Knoten ihre Kommunikationszeit größer als die eigentliche Rechenzeit ist, dauert der Lauf mit 8 Knoten fast so lange, wie der sequentielle Programmlauf. Bei Verwendung von mehr als 8 Knoten ist kein wesentlicher Zeitgewinn mehr erreichbar. Mit bis zu vier Knoten ist die T3D schneller als die Paragon, ab 8 Knoten ist sie wieder langsamer. Die T3D verwendet ebenfalls schon bei vier Knoten so viel Zeit auf Kommunikation wie zum Rechnen. Bei der Paragon ist das erst ab 8 Knoten der Fall. Für 32 Knoten dauern die Rechenläufe im Vergleich zur SP2 auf der T3D 2,5 mal, auf der Paragon nur noch 1,4 mal so lang.

6.2.2 Feine Diskretisierung (76 x 76 x 76 Gitterpunkte)

Die hier gegebene Problemgröße entspricht eher einer realen Anwendung, die man auf Parallelrechnern bearbeiten möchte. Da sie jedoch so ausgelegt ist, daß sie auf

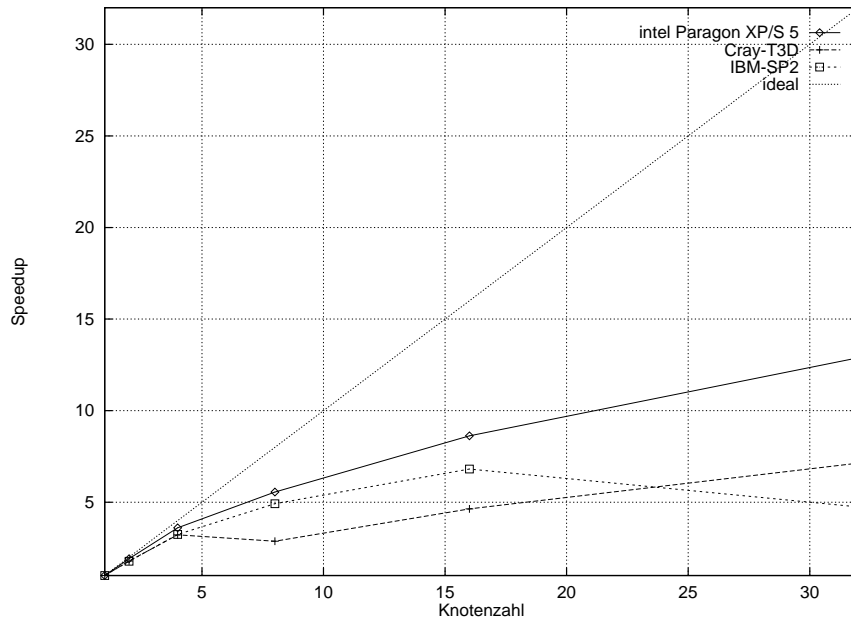


(a) Einfache Nachrichtenlänge

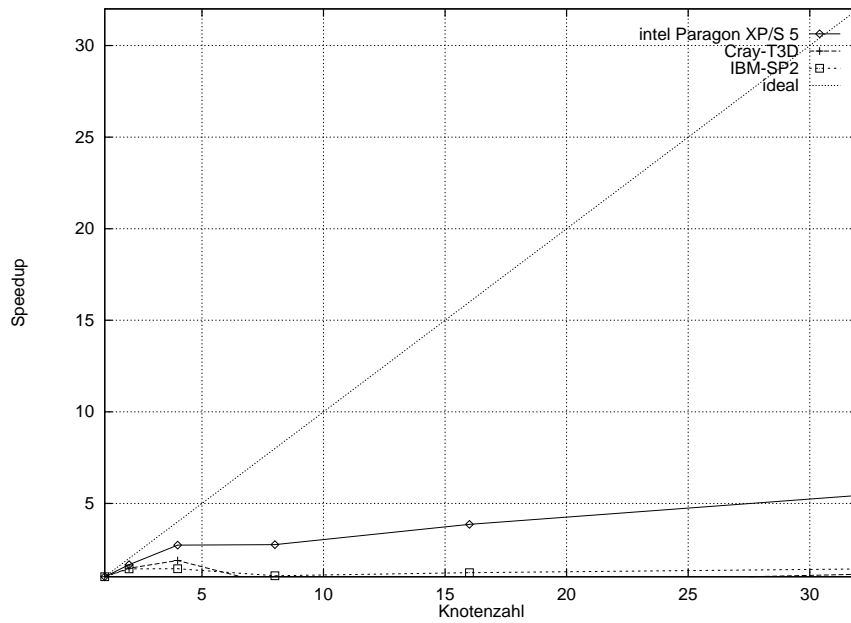


(b) Zehnfache Nachrichtenlänge

Abbildung 6.4: Laufzeit für 30 x 30 x 30 Gitterpunkte

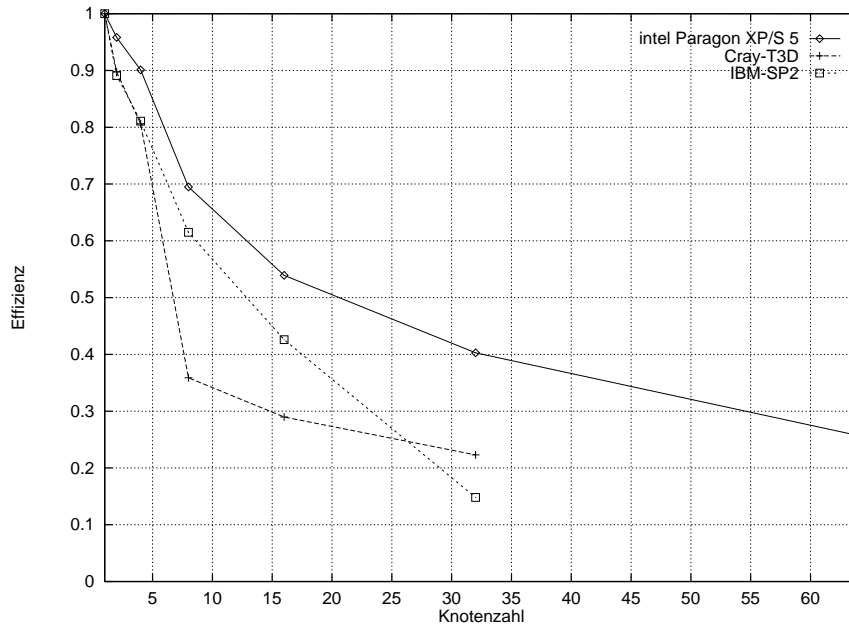


(a) Einfache Nachrichtenlänge

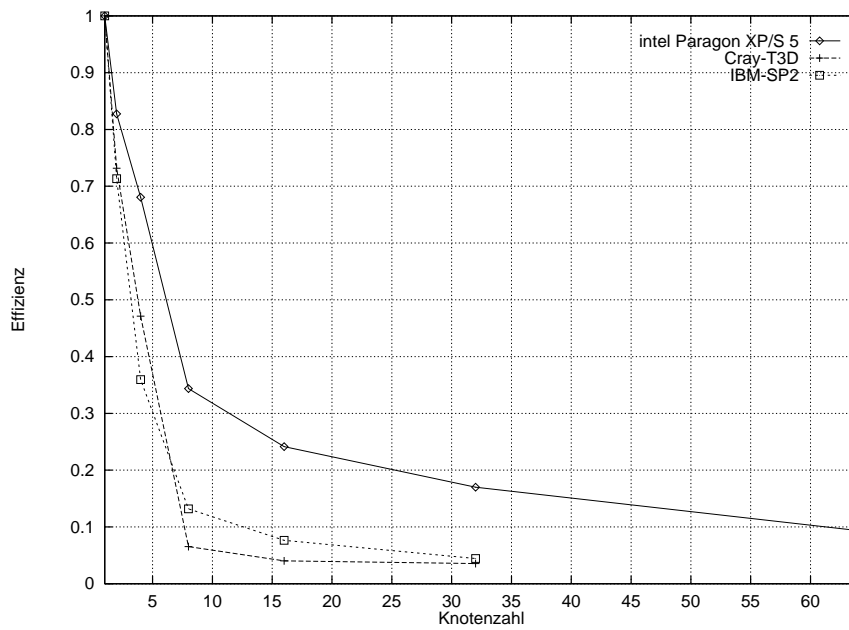


(b) Zehnfache Nachrichtenlänge

Abbildung 6.5: Speedup für 30 x 30 x 30 Gitterpunkte



(a) Einfache Nachrichtenlänge



(b) Zehnfache Nachrichtenlänge

Abbildung 6.6: Effizienz für 30 x 30 x 30 Gitterpunkte

einen Knoten der Paragon paßt, ist sie immer noch klein im Vergleich zu Anwendungen, die die Maschinen auslasten könnten. Die Ergebnisse der Rechenläufe sind in den Bildern ?? bis ?? dargestellt.

Einfache Nachrichtenlänge

Die Diskretisierung dieses Beispiels liefert eine Gitterpunktzahl, die der tatsächlicher Anwendungen entspricht. Das gleiche gilt für die notwendigen Rechenlaufzeiten. Die Läufe mit jeweils einem Prozessor dauern auf der SP2 55 Minuten, auf der T3D eine Stunde 52 Minuten, und auf der Paragon fünf Stunden. Bei so großen Laufzeiten verspricht man sich durch parallele Bearbeitung einen deutlichen Zeitgewinn.

Für die reinen Rechenzeiten gilt nach wie vor das bisher gesagte, die Zeitverhältnisse liegen für Paragon:T3D:SP2 bei 5:2:1. Bis zu vier Knoten hat die SP2 die längste, nicht durch Rechenzeit überdeckbare, Kommunikationszeit, was auch bedeuten kann, daß die Prozessoren mit der Rechnung fertig sind und auf den Empfang der Nachrichten warten müssen. Bei Verwendung von 8 oder mehr Knoten ist die Kommunikation auf der SP2 allerdings schneller, als auf der T3D. Die guten Werte der Paragon erreicht sie jedoch nie. Für vier Knoten ist die Kommunikation auf der T3D am schnellsten, bei Verwendung von zwei Knoten ist die Kommunikationszeit der T3D nur wenig länger als die der Paragon. Bei Läufen mit mehr als 8 Knoten nimmt die Kommunikationszeit wieder ab, die Werte liegen aber über denen von SP2 und Paragon. Die Paragon hat die kürzesten Kommunikationszeiten, nur bei vier Knoten ist die T3D etwas schneller. Auch bei der Paragon nimmt die Kommunikationszeit bei Verwendung größerer Knotenzahlen ab, sobald eine dreidimensionale Gebietszerlegung einmal erreicht ist.

Für die Laufzeiten erreicht die SP2 wieder die kleinsten Werte der drei Rechner. Die Verwendung zusätzlicher Rechenknoten bringt in jedem Fall einen Zeitgewinn. Das gilt für alle drei Rechner, die Größe des Zeitgewinns ist jedoch unterschiedlich. Für 32 Knoten liegen die Laufzeitverhältnisse für Paragon:T3D:SP2 bei 3:1,5:1, was bedeutet, daß sich die parallele Bearbeitung wiederum auf der Paragon am meisten gelohnt hat. Bei ihr ist das Verhältnis von Prozessorleistung zur Leistung des Kommunikationsmediums am ausgeglichensten. Bei keinem der Rechner wird die Kommunikationszeit größer als die Rechenzeit, bei der SP2 wird sie aber fast gleich groß.

Rechner mit schnellen Prozessoren haben einen deutlichen Zeitvorteil bei dem hier gegebenen Beispiel. Auch wenn ihre Kommunikationsleistung nicht ganz so gut ist wie die der Paragon, so ist doch bei Verwendung von 32 Knoten nur ein

Drittel der Laufzeit nötig. Ein anderer Vergleich ist der, daß auf der T3D doppelt so viele und auf der Paragon viermal so viele Knoten notwendig sind, um das Beispiel in der selben Zeit zu bearbeiten, die die SP2 braucht.

Zehnfache Nachrichtenlänge

Dieses Beispiel wird wieder durch die Kommunikationsleistung bestimmt, da die Kommunikationszeiten für größere Knotenzahlen ein Mehrfaches der reinen Rechenzeit betragen kann. Das gilt vor allem für die SP2.

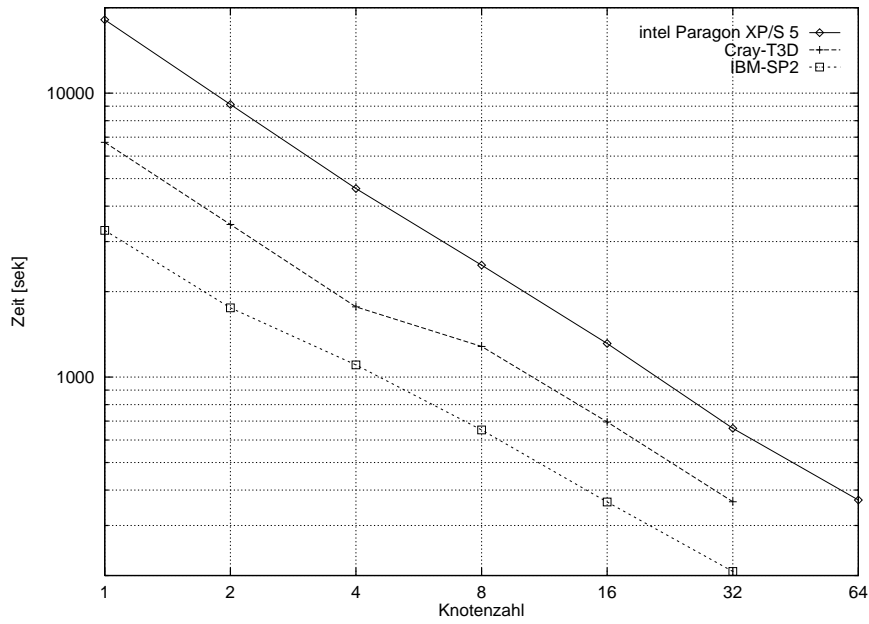
Die reine Rechenleistung wird durch die längeren Nachrichten nicht beeinflußt, deshalb bringt dieses Beispiel keine neue Erkenntnisse hierfür.

Bei zwei Knoten hat die SP2 die längste Kommunikationszeit, bei Verwendung von vier Knoten unterschreitet sie die Zeit der Paragon, und ab 8 Knoten ist sie in der Kommunikation der schnellste der drei Rechner. Da dies bei langen Nachrichten der Fall ist, deutet dieses Ergebnis darauf hin, daß die Datenübertragungsrate bei der SP2 sehr hoch ist. Sobald die Verbindung steht, findet der Transfer sehr schnell statt. Bis zu vier Knoten hat die T3D die beste Kommunikationszeit, ab 8 Knoten mit Abstand die schlechteste. Bei der Paragon ist bemerkenswert, daß ihre Kommunikationszeiten denen der SP2 sehr ähnlich sind. Sie ist dabei allerdings immer ein bißchen langsamer.

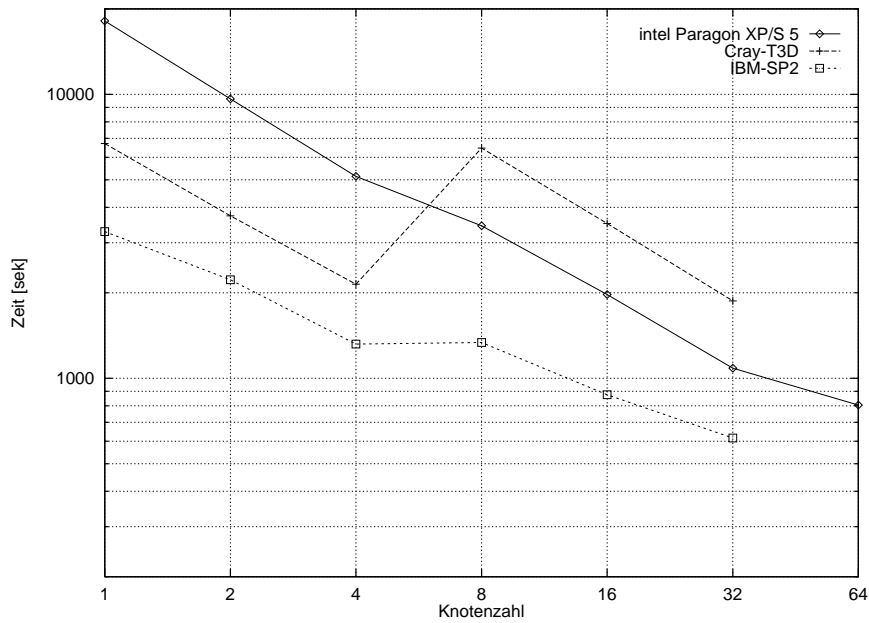
Für die gesamte Laufzeit ergibt sich das gleiche Bild wie bisher. Die SP2 ist für alle untersuchten Knotenzahlen der schnellste Rechner. Der Zeitgewinn durch parallele Bearbeitung ist bei ihr aber nicht so groß, wie bei der Paragon. Für die T3D lassen sich keine einheitlichen Aussagen machen. Mit bis zu vier Knoten liegen ihre Laufzeiten zwischen denen von SP2 und Paragon, ab 8 Knoten ist sie der langsamste Rechner. Ein Lauf mit 8 Knoten auf der T3D dauert sogar länger, als mit vier oder sogar zwei Knoten. Bei der Paragon bedeuten größere Knotenzahlen auch hier immer einen Zeitgewinn. Sie kann ihre schwache Prozessorleistung damit ausgleichen, daß sie mehr Knoten zu Verfügung stellt.

6.3 Vergleich der Testfälle

In diesem Abschnitt werden für jeweils einen Rechner die unterschiedlichen Testfälle betrachtet. Man kann dadurch erkennen, wie sich Änderungen in der Anforderung auf die Rechner auswirken. Plattformtypische Erscheinungen lassen sich so ebenfalls deutlicher zeigen.

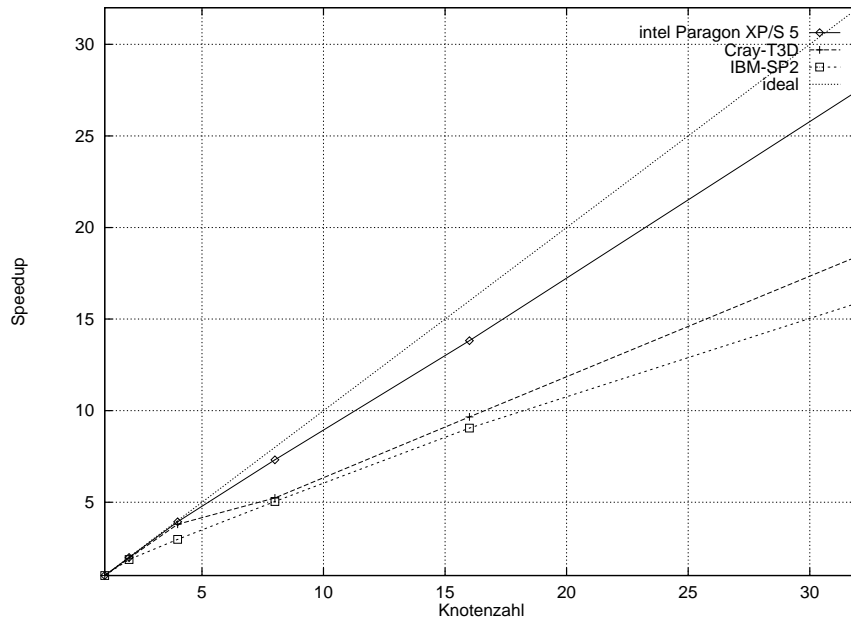


(a) Einfache Nachrichtenlänge

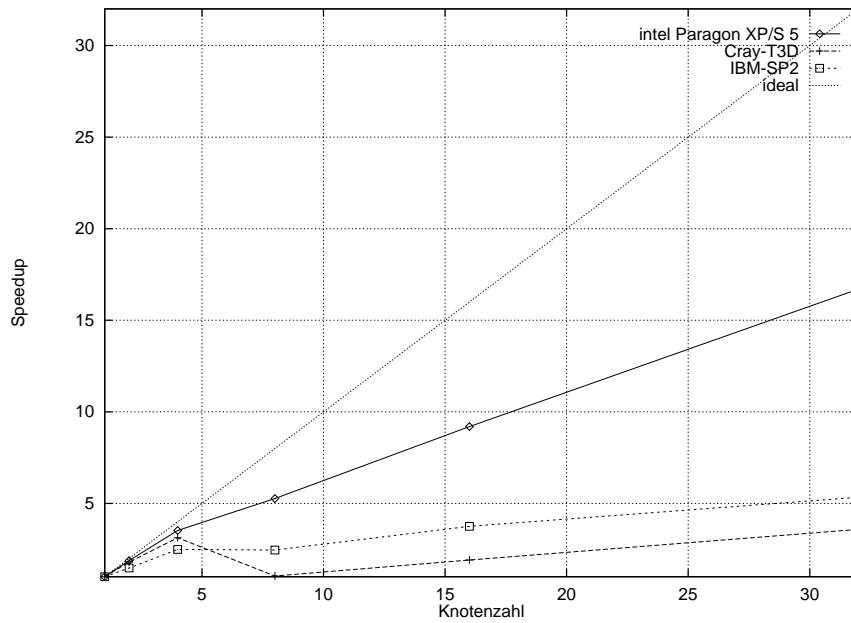


(b) Zehnfache Nachrichtenlänge

Abbildung 6.7: Laufzeit für 76 x 76 x 76 Gitterpunkte

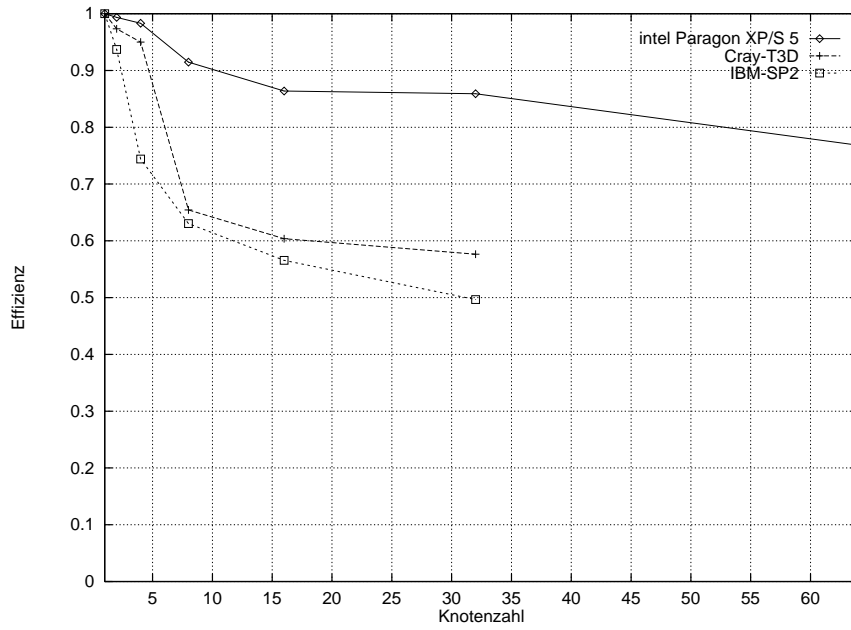


(a) Einfache Nachrichtenlänge

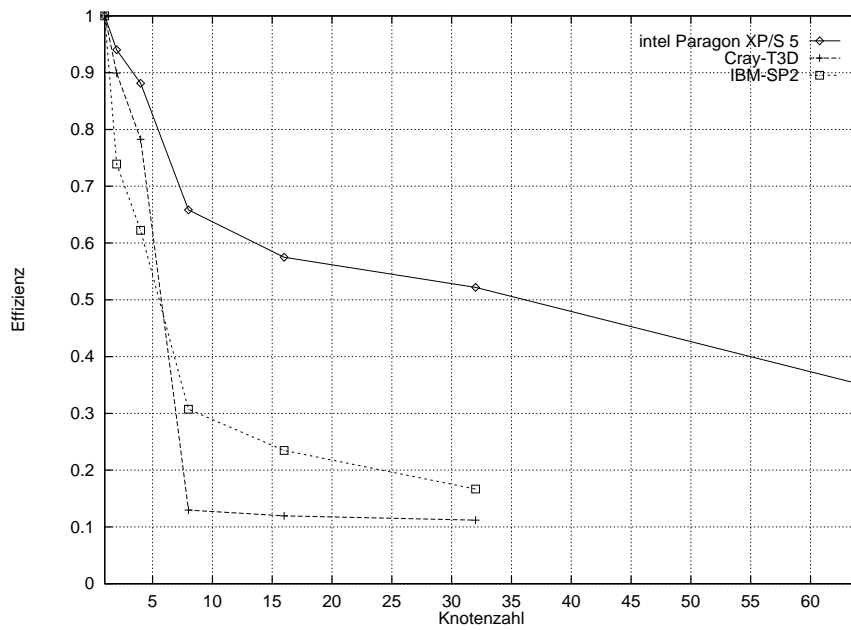


(b) Zehnfache Nachrichtenlänge

Abbildung 6.8: Speedup für 76 x 76 x 76 Gitterpunkte



(a) Einfache Nachrichtenlänge



(b) Zehnfache Nachrichtenlänge

Abbildung 6.9: Effizienz für 76 x 76 x 76 Gitterpunkte

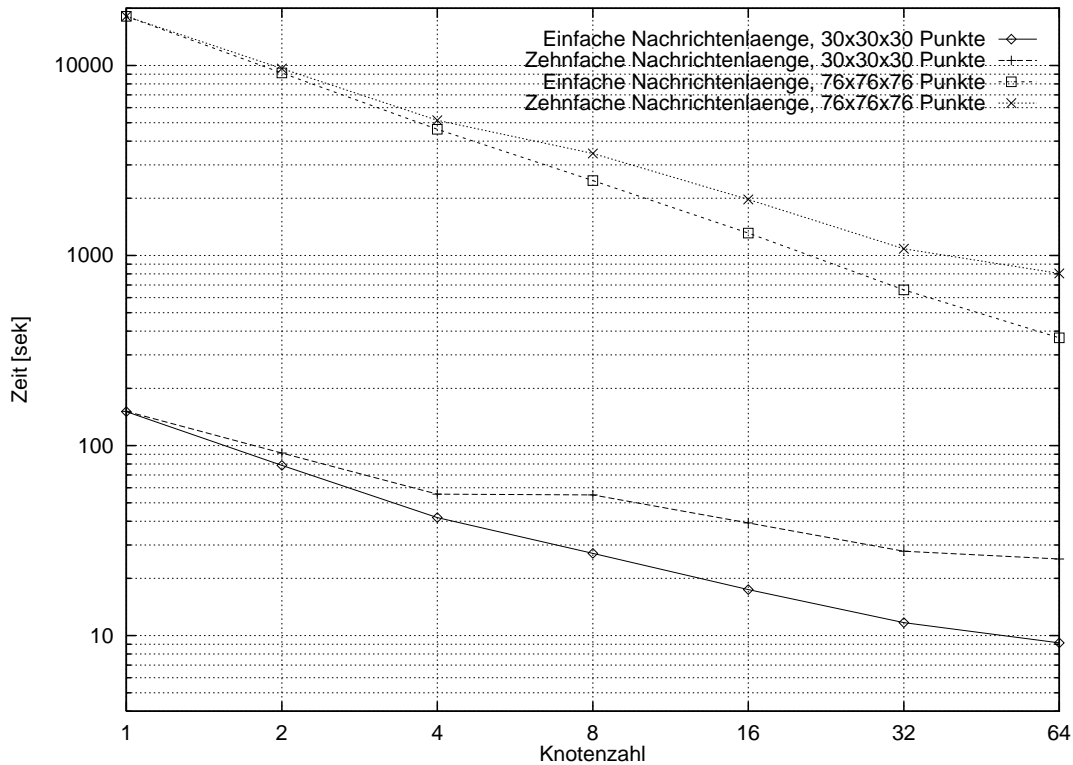


Abbildung 6.10: Laufzeit für intel Paragon

Bei der Betrachtung der Laufzeiten für die verschiedenen Testfälle auf allen Rechnern ist feststellbar, daß die Anforderung an die Rechenleistung durch die feine Diskretisierung mit $76 \times 76 \times 76$ Gitterpunkten wie erwartet die 120-fache Laufzeit im Vergleich zur groben Diskretisierung mit $30 \times 30 \times 30$ Gitterpunkten zur Folge hat.

Bei Rechenläufen auf der Paragon ist feststellbar, daß der Übergang von einer zweidimensionalen auf eine dreidimensionale Gebietszerlegung die Zahl der zu verarbeitenden Nachrichten deutlich erhöht. Dadurch nimmt die Kommunikationszeit merklich zu. Dieser Effekt tritt umso stärker auf, je länger die einzelnen Nachrichten sind. Insgesamt läßt sich für die Paragon aber sagen, das für die untersuchten Beispiele keine Engpässe in der Kommunikation aufgetreten sind, und daß sehr wahrscheinlich auch mit mehr als 64 Knoten noch eine Verkürzung der Laufzeiten zu erreichen ist.

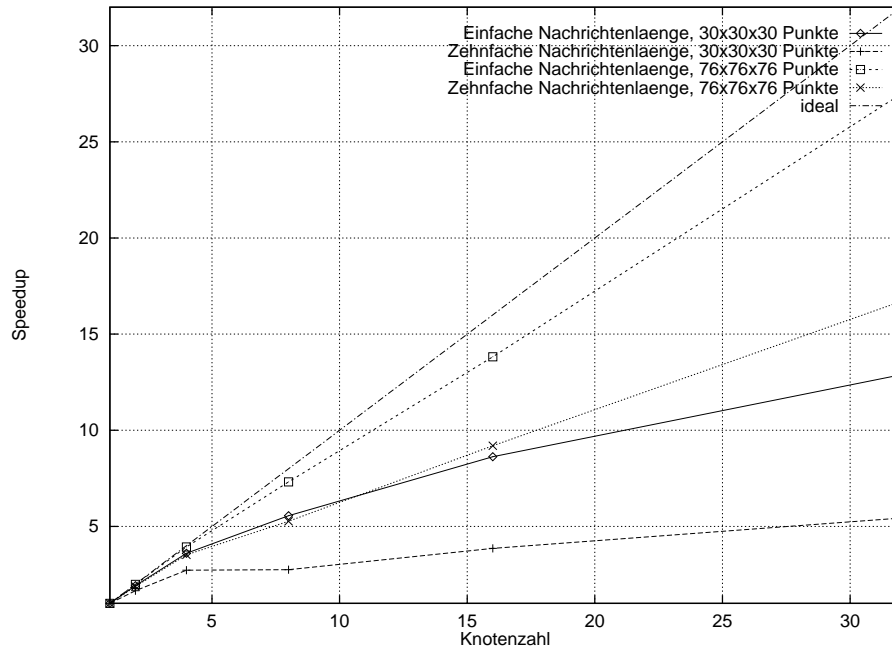


Abbildung 6.11: Speedup für intel Paragon

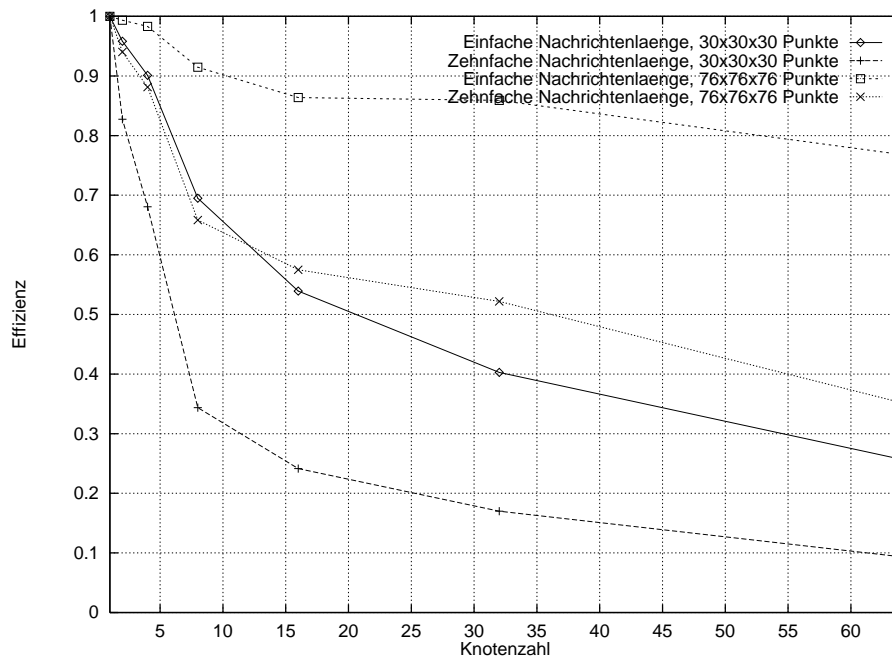


Abbildung 6.12: Effizienz für intel Paragon

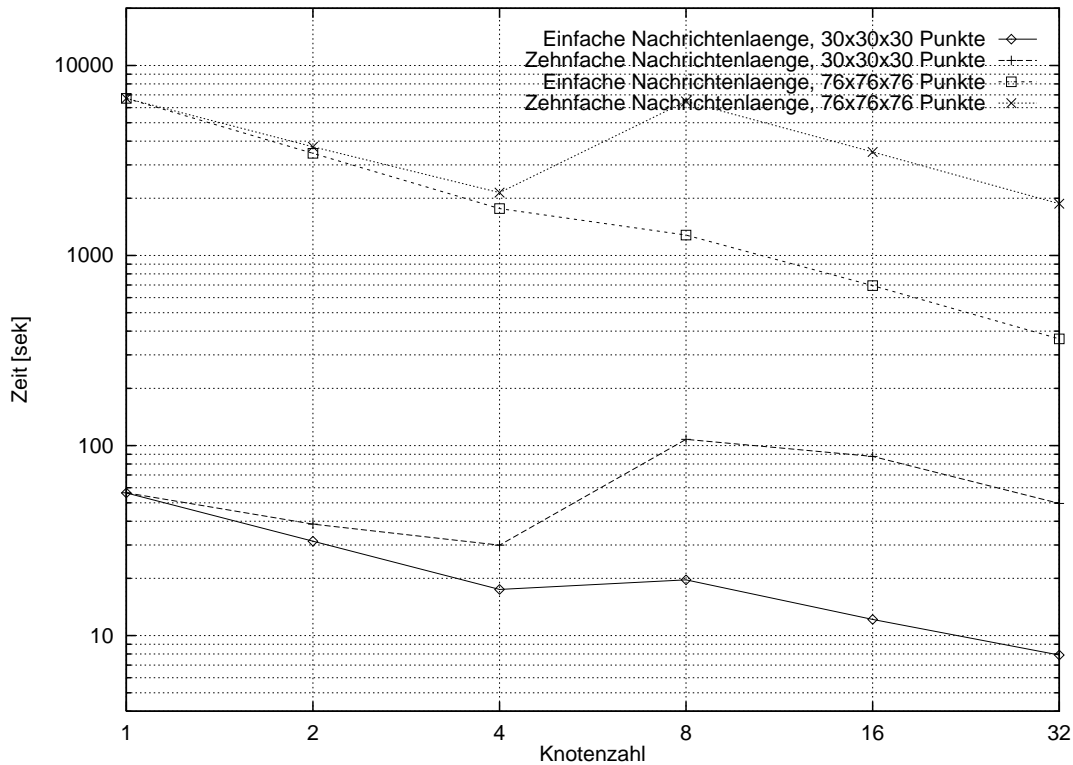


Abbildung 6.13: Laufzeit für Cray-T3D

Die T3D verhält sich bei Läufen auf bis zu vier Knoten ähnlich unspektakulär wie die Paragon. Der Anstieg der Kommunikationszeit beim Übergang von vier auf 8 Knoten ist aber so groß, daß die Laufzeiten für 8 Knoten in drei der vier Fälle über denen für vier Knoten liegen, in einem Fall sogar über der sequentiellen Laufzeit liegt. Scheinbar können auf der T3D trotz des guten Verbindungsnetzwerks, das als 3-D Torus angelegt ist, die Nachrichten nicht gleichzeitig übertragen werden. Eine mögliche Erklärung dafür liegt im Auftreten von Konflikten beim Zugriff auf den Hauptspeicher. Außerdem kann sich das Fehlen eines Betriebssystems auf den Knoten der T3D so bemerkbar machen, wenn eine Organisation der Prozesse für die eigentliche Rechnung und die Kommunikation nicht nebeneinander möglich ist. Am deutlichsten ist der Einbruch der Leistung für mehr als vier Knoten in Bild ?? zu sehen.

Die SP2 fällt vor allem dadurch auf, daß sie unterschiedliche Verhaltensweisen für grobe und feine Diskretisierung zeigt. Wenn eine große Rechenleistung nötig ist, kann sie die Kommunikation sehr gut mit der Berechnung überdecken und diese macht sich nicht störend bemerkbar. Das ist sogar für lange Nachrichten noch der Fall. Anders verhält es sich, wenn hauptsächlich kommuniziert werden soll.

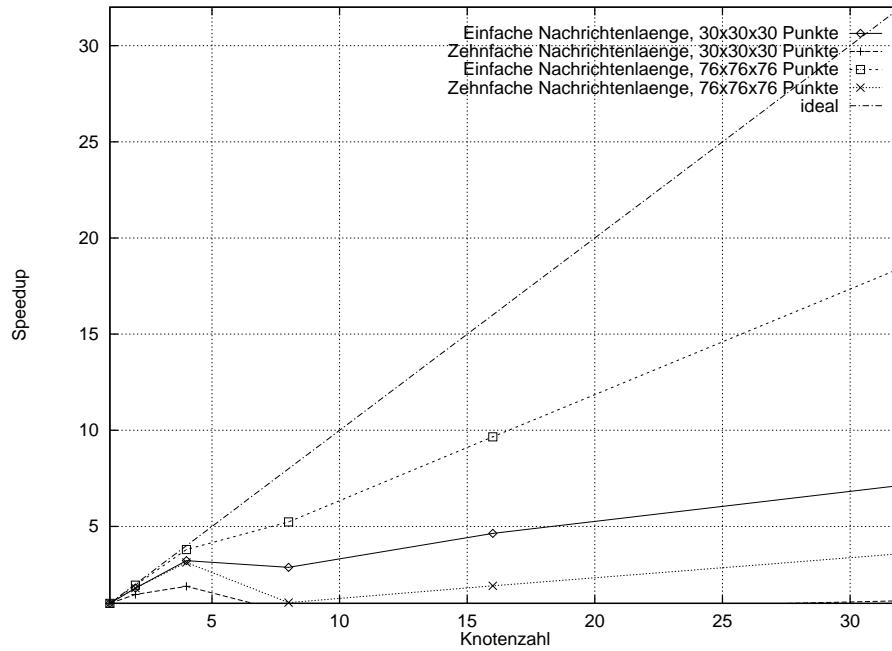


Abbildung 6.14: Speedup für Cray-T3D

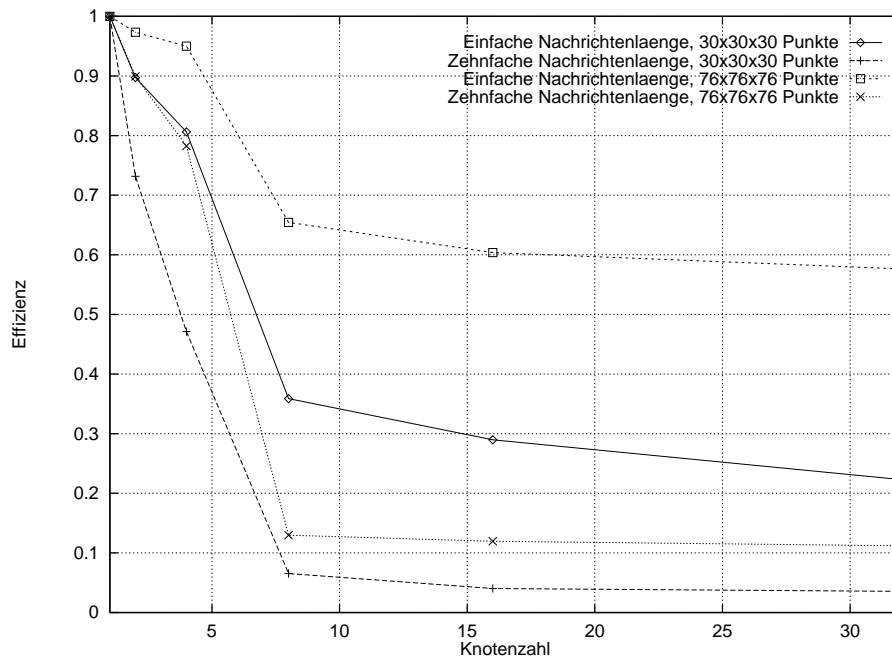


Abbildung 6.15: Effizienz für Cray-T3D

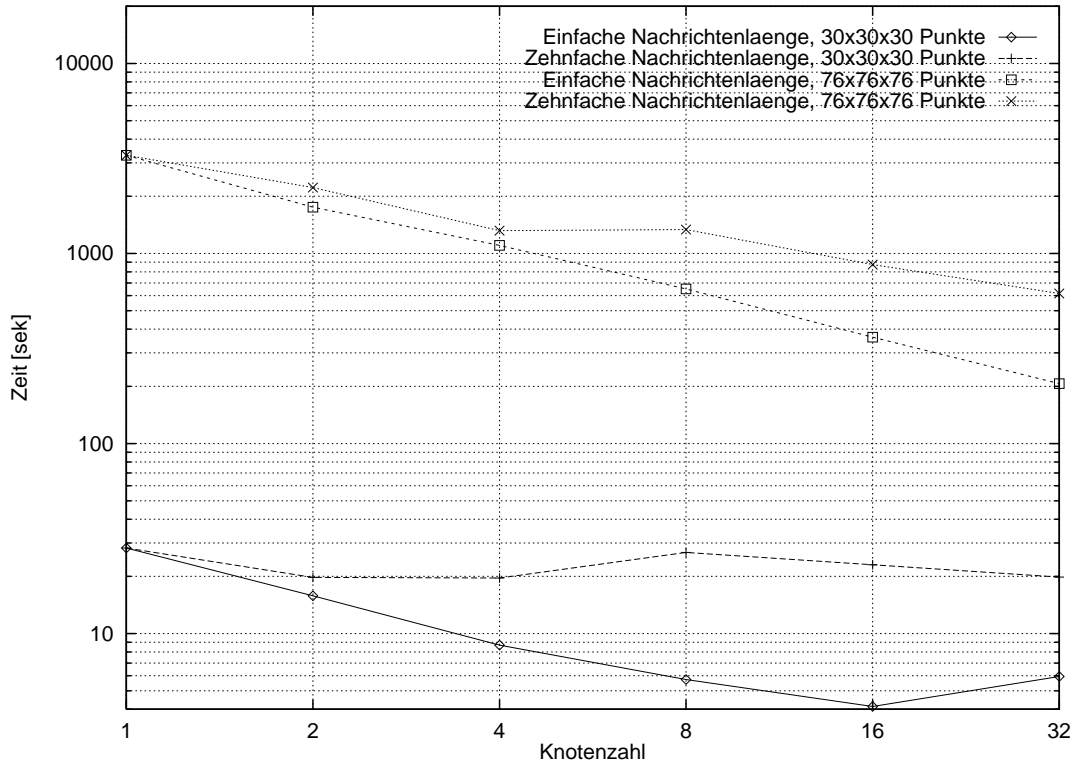


Abbildung 6.16: Laufzeit für IBM-SP2

Die Kommunikation macht sich vor allem dann störend bemerkbar, wenn viele einzelne Nachrichten übermittelt werden müssen. Die Verwendung einer geschalteten Verbindung ist sehr wahrscheinlich der Grund für dieses Verhalten. Die eigentliche Nachrichtenübertragung erfolgt sogar recht schnell, aber eben erst, wenn die Verbindung steht.

6.4 Fazit

Über die drei untersuchten Rechner lassen sich zusammenfassend folgende Aussagen machen. Die SP2 bietet mit Abstand die beste Rechenleistung. Für rechenintensive Anwendungen mit einem geringen Kommunikationsanteil ist sie sehr gut geeignet. Sollte jedoch häufig ein Datenaustausch mit vielen kleinen Nachrichten notwendig sein, so werden die schnellen Prozessoren durch die Nachrichtenübertragung behindert. Für den Anwender heißt das, daß er nur bis zu einer bestimmten, nicht allzu großen Knotenzahl noch sinnvolle Laufzeitverkürzungen erreichen kann.

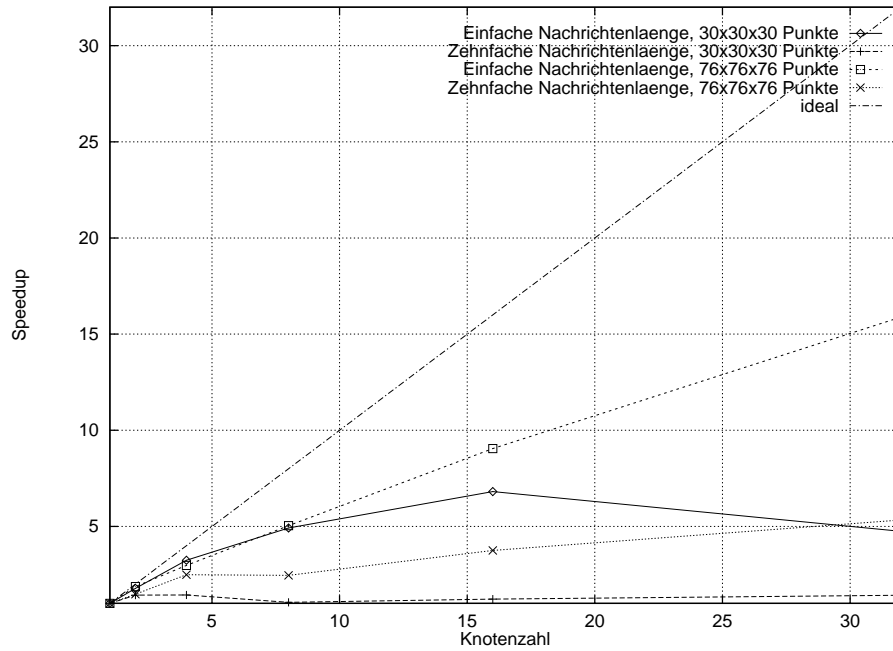


Abbildung 6.17: Speedup für IBM-SP2

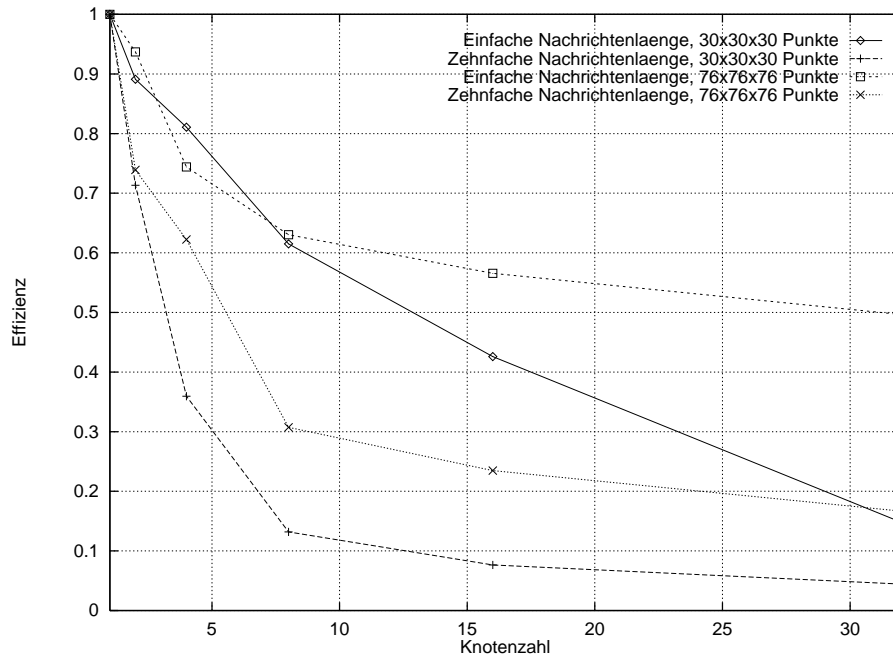


Abbildung 6.18: Effizienz für IBM-SP2

Bei der T3D hat sich gezeigt, daß ihr Kommunikationssystem nur bis zu vier Knoten vernünftig unterstützen kann. Wenn mehr Knoten benötigt werden, sind die Nachteile durch den größeren Kommunikationsaufwand viel größer als der Gewinn an Rechenleistung. Mit vier Knoten ist aber die Auswahl an Problemen, die mit der T3D bearbeitet werden können, stark eingeschränkt, da in dem Fall nur 256 MB Hauptspeicher und 600 MFLOPS Spitzenleistung zu Verfügung stehen. Für Probleme, die man mit Parallelrechnern angehen will, ist das wahrscheinlich nicht genug.

Die Paragon hat bei der Untersuchung gezeigt, daß ihre Prozessoren für Vergleiche mit neuen Rechnern schon zu alt und deshalb zu schwach sind. Um die gleiche Rechenleistung wie auf modernen Systemen zu erhalten, muß man deutlich mehr Knoten einsetzen, mit allen damit verbundenen Nachteilen. Das Kommunikationssystem ist jedoch so gut an die Prozessorleistung angepaßt, daß es keine Behinderung darstellt. Aus diesem Grund ist die Paragon gut geeignet für Probleme, bei denen häufig mit vielen Prozessen kommuniziert werden muß. Das und die große Zahl verfügbarer Rechenknoten machen sie durchaus noch für einige Aufgabenbereiche interessant.

6.5 Ausblick

Für eine Weiterführung dieser Untersuchung bieten sich folgende Bereiche an. Als nächstes könnte man die beiden neuen Großrechner am Rechenzentrum der Uni Stuttgart, eine NEC SX4/32 und eine Cray T3E mit 512 Knoten, mit den hier verwendeten Beispielen untersuchen, und einen Vergleich mit den drei bereits untersuchten Rechner anstellen. Außerdem wäre es interessant, die Verwendung von mehr als 64 Knoten zu untersuchen. Bei der Paragon am RUS wird das nach einer Aufrüstung mit weiteren Knoten möglich sein. Ein Vergleich mit den Leistungen eines Workstation-Clusters könnte ebenfalls interessant sein.

Kapitel 7

Tabellen

Die Ergebnisdaten der Zeitmessungen bieten mehrere Möglichkeiten zu ihrer Auswertung. In der hier vorgestellten Studienarbeit wurden nur Betrachtungen hinsichtlich Laufzeit, Speedup und Effizienz unternommen. Außerdem war die Zahl der durchgeführten Rechenläufe eingeschränkt und kann für weitere Untersuchungen erweitert werden.

Damit man für Auswertungen unter anderen Gesichtspunkten die Daten als Zahlenwerte zu Verfügung hat, werden die Ergebnisse an dieser Stelle in Tabellen angegeben. In den Schaubildern sind die Zahlenwerte vor allem für lange Rechenläufe nicht mehr gut abzulesen, wofür man ebenfalls auf die Tabellen zurückgreifen kann.

Knotenzahl	Paragon	T3D	SP2
1	150.29	56.27	28.19
2	76.00	29.98	14.08
4	37.44	15.01	7.14
8	19.52	8.19	3.81
16	9.74	4.19	1.89
32	4.74	2.17	0.97
64	2.44		

Tabelle 7.1: Rechenzeit in Sekunden für 30 x 30 x 30 Gitterpunkte

Knoten- zahl	Nachrichtenlänge					
	einfach			zehnfach		
	Paragon	T3D	SP2	Paragon	T3D	SP2
1	0.38	0.09	0.00	0.88	0.08	0.00
2	2.62	1.43	1.74	13.94	8.41	5.68
4	4.36	2.47	1.56	16.22	14.97	12.30
8	7.59	11.45	1.92	35.11	99.71	22.96
16	7.72	7.96	2.24	29.29	83.51	21.12
32	6.95	5.73	4.97	22.98	47.45	18.83
64	6.72			22.72		

Tabelle 7.2: Kommunikationszeit in Sekunden für 30 x 30 x 30 Gitterpunkte

Knoten- zahl	Nachrichtenlänge					
	einfach			zehnfach		
	Paragon	T3D	SP2	Paragon	T3D	SP2
1	150.67	56.36	28.19	151.22	56.42	28.19
2	78.62	31.41	15.82	91.38	38.56	19.76
4	41.80	17.48	8.69	55.54	29.94	19.60
8	27.10	19.64	5.73	55.01	107.89	26.73
16	17.46	12.16	4.14	39.15	87.70	23.03
32	11.69	7.90	5.94	27.81	49.60	19.81
64	9.16			25.28		

Tabelle 7.3: Laufzeit in Sekunden für 30 x 30 x 30 Gitterpunkte

Knotenzahl	Paragon	T3D	SP2
1	18134.23	6712.59	3285.04
2	9090.72	3410.50	1638.72
4	4546.56	1718.64	822.21
8	2337.52	887.68	426.76
16	1174.75	462.90	217.49
32	578.78	233.04	108.11
64	290.66		

Tabelle 7.4: Rechenzeit in Sekunden für 76 x 76 x 76 Gitterpunkte

Knoten- zahl	Nachrichtenlänge					
	einfach			zehnfach		
	Paragon	T3D	SP2	Paragon	T3D	SP2
1	2.55	0.50	0.00	2.52	0.50	0.00
2	36.59	37.72	114.38	471.97	316.89	584.59
4	65.24	47.96	281.55	561.89	416.49	497.90
8	141.06	394.49	224.69	1066.28	5578.93	910.11
16	137.64	232.15	145.50	775.72	3045.00	657.51
32	81.03	130.95	98.66	502.02	1640.24	506.27
64	78.16			508.98		

Tabelle 7.5: Kommunikationszeit in Sekunden für 76 x 76 x 76 Gitterpunkte

Knoten- zahl	Nachrichtenlänge					
	einfach			zehnfach		
	Paragon	T3D	SP2	Paragon	T3D	SP2
1	18136.78	6713.09	3285.04	18134.02	6713.09	3285.04
2	9127.31	3448.22	1753.10	9642.69	3732.73	2222.43
4	4611.80	1766.60	1103.76	5143.29	2144.67	1319.61
8	2478.58	1282.17	651.45	3442.92	6470.66	1336.29
16	1312.39	695.05	362.99	1971.51	3509.31	875.04
32	659.81	363.99	206.77	1085.68	1873.43	615.74
64	368.82			804.86		

Tabelle 7.6: Laufzeit in Sekunden für 76 x 76 x 76 Gitterpunkte

Knoten- zahl	Nachrichtenlänge					
	einfach			zehnfach		
	Paragon	T3D	SP2	Paragon	T3D	SP2
1	1.00	1.00	1.00	1.00	1.00	1.00
2	1.92	1.79	1.78	1.65	1.46	1.43
4	3.60	3.22	3.24	2.72	1.88	1.44
8	5.56	2.87	4.92	2.75	0.52	1.05
16	8.63	4.64	6.81	3.86	0.64	1.22
32	12.89	7.14	4.74	5.44	1.14	1.42
64	16.45			5.98		

Tabelle 7.7: Speedup für 30 x 30 x 30 Gitterpunkte

Knoten- zahl	Nachrichtenlänge					
	einfach			zehnfach		
	Paragon	T3D	SP2	Paragon	T3D	SP2
1	100.0 %	100.0 %	100.0 %	100.0 %	100.0 %	100.0 %
2	95.8 %	89.7 %	89.1 %	82.7 %	73.2 %	71.3 %
4	90.1 %	80.6 %	81.1 %	68.1 %	47.1 %	36.0 %
8	69.5 %	35.9 %	61.5 %	34.4 %	6.5 %	13.2 %
16	53.9 %	29.0 %	42.6 %	24.1 %	4.0 %	7.7 %
32	40.3 %	22.3 %	14.8 %	17.0 %	3.6 %	4.4 %
64	25.7 %			9.3 %		

Tabelle 7.8: Effizienz für 30 x 30 x 30 Gitterpunkte

Knoten- zahl	Nachrichtenlänge					
	einfach			zehnfach		
	Paragon	T3D	SP2	Paragon	T3D	SP2
1	1.00	1.00	1.00	1.00	1.00	1.00
2	1.99	1.95	1.87	1.88	1.80	1.48
4	3.93	3.80	2.98	3.53	3.13	2.49
8	7.32	5.24	5.04	5.27	1.04	2.46
16	13.82	9.66	9.05	9.20	1.91	3.75
32	27.49	18.44	15.89	16.70	3.58	5.34
64	49.17			22.53		

Tabelle 7.9: Speedup für 76 x 76 x 76 Gitterpunkte

Knoten- zahl	Nachrichtenlänge					
	einfach			zehnfach		
	Paragon	T3D	SP2	Paragon	T3D	SP2
1	100.0 %	100.0 %	100.0 %	100.0 %	100.0 %	100.0 %
2	99.4 %	97.3 %	93.7 %	94.0 %	89.9 %	73.9 %
4	98.3 %	95.0 %	74.4 %	88.1 %	78.3 %	62.2 %
8	91.5 %	65.4 %	63.0 %	65.8 %	13.0 %	30.7 %
16	86.4 %	60.4 %	56.6 %	57.5 %	12.0 %	23.5 %
32	85.9 %	57.6 %	49.6 %	52.2 %	11.2 %	16.7 %
64	76.8 %			35.2 %		

Tabelle 7.10: Effizienz für 76 x 76 x 76 Gitterpunkte

Literaturverzeichnis

- [1] Thomas Bönisch. *Implementierung eines 3-D Strömungscodes auf Parallelrechnern*. Diplomarbeit, Universität Stuttgart, April 1996.
- [2] Message Passing Interface Forum. *MPI: A Message-Passing Interface Standard*, Juni 1995.
- [3] Alfred Geiger. *Parallelrechner – Architektur und Anwendung*. Forschungs- und Entwicklungsberichte, Rechenzentrum Universität Stuttgart, Februar 1994.
- [4] William Gropp, Ewing Lusk, and Anthony Skjellum. *USING MPI – Portable Parallel Programming with the Message-Passing Interface*. MIT Press, 1995.