Parallelisierung einer Molekulardynamik-Aufgabe mit Verwendung eines gemeinsamen Speichers

Rudolf Weber

Januar 1995 TR-I4-95.05

Interner Bericht

Institut für
Mathematische Maschinen
und Datenverarbeitung
der
Friedrich-Alexander-Universität
Erlangen-Nürnberg

Lehrstuhl für Informatik IV
(Betriebssysteme)

This paper was also published as:

Parallelisierung einer Molekulardynamik-Aufgabe mit Verwendung eines gemeinsamen Speichers

Rudolf Weber

IMMD IV

Überblick

Ein Molekulardynamik-Programm wird auf einem MIMD-Rechner mit gemeinsamen Speicher implementiert. Einmal mit UNIX-SYSTERM V Semaphoren und Shared Memory als Mulithread-Version.

1 Einleitung

1.1 Allgemeines zur Computersimulation der Molekulardynamik

Viele physikalische Phänomene lassen sich hinreichend genau durch mathematische Modelle beschreiben, die durch numerische Methoden auf Computern berechnet werden können. In vielen Gebieten der Physik, wie z. B. der Fluidphysik, stellt sich häufig das Problem, makroskopische Größen eines Systems, wie Druck, innere Energie oder Temperatur, aus den mikroskopischen Daten der es aufbauenden Teilchen zu berechnen. Die Güte der Computer-Molekulardynamik-Simulation heutzutage ist so groß, daß sie bereits als Quasi-Experiment anerkannt wird. Dies ermöglicht die theoretische Durchführung von Experimenten, die praktisch wegen extremer Umweltbedingungen, wie zu hohen Drucks, oder extremer Größe des Systems, wie bei Simulationen der Wechselwirkungen zwischen Galaxien, nicht oder nur sehr schwer durchführbar sind.

Trotz der Vielfalt der auf diese Weise auf Rechnern beschreibbaren Probleme läßt sich die Mehrheit anhand der verwendeten Modelle in zwei große Klassen einteilen, die Kontinuummodelle zum einen, die Vielteilchenmodelle zum anderen.
Kontinuummodelle beschreiben reale Systeme durch einen Satz gekoppelter nichtlinearer partieller Differentialgleichungen, für die mit bekannten numerischen Verfahren eine Näherungslösung bestimmt wird.
Vielteilchenmodelle nähern reale Systeme durch vereinfachte Modellsysteme an. Interessiert der Gleichgewichtszustand des Systems, so können Monte-Carlo-Methoden verwendet werden. Hierbei werden mittels statistischer Verfahren Stichproben möglicher Konfigurationen des Systems genommen. Die makroskopischen Größen des Gesamtsystems können durch Mittelung näherungsweise bestimmt werden. Soll allerdings das dynamische Verhalten des Systems simu-

molshm.doc: 19.06.95 19:21

liert werden, muß man sich den rechenaufwendigen Molekulardynamik-Verfahren zuwenden. Das klassische Molekulardynamik-Verfahren hat mindestens eine Komplexität von $O(N^2)$, da es die Wechselwirkung zwischen jedem Paar von Teilchen berechnen muß. Anwenderaufgaben dieses Typs sind jedoch gut parallelisierbar durch Aufteilung der Daten (im Fall der Molekulardynamik der Moleküle) auf die einzelnen Rechnerknoten eines Parallelrechners ([Frit90], [FrVo87]). Bei der Entwicklung neuer Parallelrechner sollte die Art der überwiegend darauf laufenden Anwendungen beim Entwurf der Architektur berücksichtigt werden.

1.2 Beispielprogramm

Das Programm stammt aus [Hess88],[Hess91].

Es gibt schon folgende Parallelisierungen:

- MEIKO-Transputersystem mit Datenpationierung nach Lagrangeschma und Ring-Topologie [Hofm92],[HoKn92]
- MEMSY mit Datenpationierung nach Lagrangeschma und Ring-Topologie [Buch92]
- MEMSY mit Datenpationierung nach Lagrangeschma mit einer MEMSY-spezial-Topologie net [Buch92]
- MEMSY mit Datenpationierung nach dem Eulerschema mit einer MEMSY-spezial-Topologie net [Buch92]

Hier soll nun diese Aufgabe auf einem Rechner mit gemeinsamen Speicher parallelisiert werden. Als Rechner dient der CONVEX-SPP des RRZE. Die Parallelisierung wird mit zwei verschiedenen Modellen durchgeführt:

- Mit UNIX SYSTEM V -Prozessen, -Shared-Memory und -Semaphoren
- Mit Convex-Threads, Shared-Memory und Atomaren Zählern der Convex-AIL (im folgenden Multithreaded-Version genannt).

1.3 Überblick

In Kapitel 2 wird das mathematische Modell der Molekulardynamik-Simulation erläutert. Nach der Vorstellung des sequentiellen Berechnungsalgorithmus in Kapitel 3 wird in Kapitel 4 die Parallelisierung und die dabei auftretenden Probleme und ihre Lösungen dargestellt.

2 Mathematisches Modell der Molekulardynamik

Reale Systeme (Flüssigkeiten, Gase) werden durch vereinfachte Vielteilchenmodelle ersetzt

Folgende vereinfachte Modellannahmen werden gemacht:

(1) Ein Teilchen besitzt keine innere Struktur und keine räumliche Ausdehnung.

(2) Alle Teilchen haben dieselbe Masse m.

2.1 Zustandsbeschreibung

Def 1: ***Zustand eines Teilchens***

Zu einem Zeitpunkt t läßt sich der Zustand eines Teilchens mit folgenden Größen beschreiben:

p sei der Ortsvektor des Teilchens

v sei der Geschwindigkeitsvektor des Teilchens

Def 2: ***Zustand des Systems***

Der Zustand des Systems läßt sich durch folgende Größen beschreiben:

N : Zahl der Teilchen (konstant)

m: Menge der Moleküle/Teilchen (von 0 bis (N-1) durchnumeriert)

2.2 Dynamik

Die neuen Werte von $\vec{p}_i$ und $\vec{v}_i$ zum nächsten betrachteten Zeitpunkt t+Δt mit Schrittweite Δt errechnen sich aus der Newtonschen Bewegungsgleichung bei gegebener Paar-Wechselwirkung zwischen den Teilchen.

Vereinfachtes Modell:

Formel 3: ***Kraft von Teilchen j auf Teilchen i***

Sei $\vec{r}_{ij} = \vec{p}_i - \vec{p}_j$ der Abstand zwischen den Molekülen i und j und $r_{ij} = \|r_{ij}\|$ und $\phi(r)$ diePotentialfunktion.

Die die Kraft $\mathbf{f}_{ij}$, die von Teilchen j auf Teilchen i wirkt:

$$\vec{f}_{ij} = \frac{\vec{r}_{ij}}{r_{ij}} \cdot \frac{d\phi}{dr}$$

Im vorliegenden Algorithmus ist wird als Potentialfunktion das Lennard-Jones-Potential verwendet:

Formel 4: ***Lennard-Jones-Potential***

$$\phi(r) = 4\varepsilon\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right]$$

Der Term $(\sigma/r)^{12}$ modelliert die Abstoßung der Teilchen bei kleinerem und der Term $(\sigma/r)^6$ die van der Waals-Anziehung bei größerem Abstand. [vWav87].

Es gilt $\phi(\sigma) = 0$.

ε ist die Tiefe des Potentialminimums bei $r = \sqrt[6]{2}\sigma$, also $\left|\phi\left(\sqrt[6]{2}\sigma\right)\right| = \varepsilon$.

molshm.doc: 19.06.95 19:21

Werden alle Paarwechselwirkungen berücksichtigt, so müssen bei N Teilchen N(N-1)/2 Kraftberechnungen durchgeführt werden. Um diese Anzahl zu verringern, wird eine Cutoff-Entfernung r_{cut} eingeführt, so daß für $r > r_{cut}$ $\frac{d\phi}{dr}$ vernachlässigbar klein ist. Die Kraftberechnung wird nur noch dann durchgeführt, wenn $|r_{ij}| < r_{cut}$ ist.

Nach dem 3. Newtonschen Gesetz (actio = reactio) gilt (wie man hier auch wegen $\vec{p}_j - \vec{p}_i = -(\vec{p}_i - \vec{p}_j)$ leicht sieht:

Formel 5: ***Beziehung aufgrund 3. Newtonschengesetzes***

$$\vec{f}_{ji} = -\vec{f}_{ij}$$

2.3 Berechnung der Makroskopischen Größen

Die makroskopischen Größen, wie Druck p und Energie E, zu einem Zeitpunkt t berechnen sich aus den mikroskopischen Größen und der Potentialfunktion. Der Druck p besteht aus einem kinetischen und einem potentiellen Anteil. p wird wie folgt berechnet:

$$p = \frac{1}{3V}\left(m\sum_i \mathbf{c}_i^2 + \sum_i \sum_{j>i} \mathbf{r}_{ij}\mathbf{f}_{ij}\right)$$

Hierbei bezeichnet $\mathbf{c}_i = \dot{\mathbf{p}}_i - v(\mathbf{p}_i)$ die Pekuliargeschwindigkeit des Teilchens i relativ zur mittleren Strömungsgeschwindigkeit v = v(p). Die Energie E des Systems setzt sich ebenfalls aus einem kinetischen und einem potentiellen Anteil zusammen.

$$E = \frac{m}{2}\sum_i \mathbf{c}_i^2 + \sum_i \sum_{j>i} \phi(|\mathbf{r}_{ij}|)$$

Um endgültige Werte für Energie und Druck zu erhalten, wird gewöhnlich noch über die Anzahl der Zeitschritte gemittelt.

In der Molekulardynamik-Simulation ist es wie bei realen thermodynamischen Systemen nötig, Zustandsvariablen festzulegen. Meist wird neben dem Volumen V die Temperatur T gewählt, die durch

$$T = \frac{m}{3Nk_B}\sum_i \mathbf{c}_i^2$$

bestimmt ist, wobei k_B die Boltzmannkonstante ist. Um z. B. ein isothermes System mit vorgegebener Temperatur zu simulieren, muß die gesamte kinetische Energie

$$E_{kin} = \frac{m}{2}\sum_i \mathbf{c}_i^2$$

konstant gehalten werden, z. B. durch Reskalierung der Pekuliargeschwindigkeiten nach jedem Zeitschritt.

Meist ist die Anzahl der Teilchen im Modell sehr viel geringer als in realen Systemen, bzw. ein unendliches oder sehr großes System soll auf endlichem Volumen simuliert werden. Um dies zu ermöglichen, werden periodische Randbedingungen eingeführt. In dieser Arbeit wird z. B.

angenommen, daß für ein Teilchen i, das mit einer bestimmten Geschwindigkeit $\mathbf{v}_i$ den betrachteten Raum verläßt, ein Bildteilchen i' den betrachteten Raum mit derselben Geschwindigkeit $\mathbf{v}_i$ an einem dem Austrittsort von i gegenüberliegenden Ort betritt (anschaulich wird auf diese Weise der Raum zum Torus geschlossen). Eine weitergehende Erläuterung der physikalischen Grundlagen der Molekulardynamik-Simulation findet sich in [Hess88].

2.4 Spezielles Problem

N :Anzahl der Teilchen

n : Dichte der Teilchen

Kantenlänge des Würfels: $L = \sqrt[3]{N/n}$

Volumen = L^3

(siehe [Buch92] S.4 unten)

Ermittlungen im Berechnungsprogramm ergaben, daß der Ursprung des Koordinatensystems in der Mitte des Würfels liegt.

Der Anschauungsraum ist also der Würfel
$\{(x,y,z) \in R^3 \mid x_{min} \leq x \leq x_{max} \wedge y_{min} \leq y \leq y_{max} \wedge z_{min} \leq z \leq z_{max}\}$
mit $x_{min}=-x_{max}$, $y_{min}=-y_{max}$, $z_{min}=-z_{max}$ und $x_{max}=y_{max}=z_{max} = L/2$.

Die hier betrachtete Molekulardynamik-Simulation (als Original in FORTRAN [Hess91]) ist so stark vereinfacht, daß sie kein reales System abbildet. So sind sämtliche Naturkonstanten (wie die Boltzmannkonstante k_B), ebenso ε und σ aus dem Lennard-Jones-Potential sowie die Masse der Teilchen gleich eins gesetzt. Die mittlere Strömungsgeschwindigkeit ist v = 0, so daß sich für die Pekuliargeschwindigkeit eines Teilchens i $\mathbf{c}_i = \dot{\mathbf{p}}_i$ ergibt. Durch Anpassung der entsprechenden Funktionen läßt sich das Programm jedoch leicht so erweitern, daß reale Systeme modelliert werden können. In der vorliegenden Version ist es ein Experimentierprogramm, um die Möglichkeiten der Parallelisierung von Problemen der Molekulardynamik zu überprüfen.

3 Berechnungsalgorithmus

Initialisierung der Startwerte
Kraft()
Berechne Starttemperatur T
für alle Zeitschritte t_o bis t_{ende} Schrittweitte Δt
Berechne $\forall$ i=1...N : $\vec{p}_i$ und $\vec{v}_i$
Kraft()
Berechne $\forall$ i=1...N : $\vec{v}_i$
Berechne makroskopische Größen

Algorithmus 3.1Gesamt-Algorithmus

Die Anfangsgeschwindigkeiten werden auf T skaliert, so daß die kinetische Energie des Initialzustandes konstant ist, und somit auch bei zufälliger Verteilung der Anfangsgeschwindigkeiten von einem bestimmten Initialzustand ausgegangen werden kann.

Das Kernstück des Algorithmus ist eine Schleife über die Anzahl der Zeitschritte. Aus den Positionen $\mathbf{p}_i^t$ der Teilchen zum Zeitpunkt t wird die Gesamtkraft $\mathbf{F}_i^t$, die auf ein Teilchen i wirkt, berechnet. Unter Berücksichtigung der periodischen Randbedingungen und der Geschwindigkeiten $\mathbf{v}_i^t$, können nun die neuen Positionen $\mathbf{p}_i^{t+\Delta t}$ und Geschwindigkeiten $\mathbf{v}_i^{t+\Delta t}$ zum nächsten betrachteten Zeitpunkt $t + \Delta t$ berechnet werden und die Iteration beginnt von vorne. Über 90% der Rechenzeit nimmt hierbei die Kraftberechnung (Prozedur Kraft) in Anspruch, obwohl $\mathbf{f}_{ij}$ nur berechnet wird wenn für den Abstand r zwischen Teilchen i und Teilchen j (oder eines der Bildteilchen von j, je nachdem welches sich näher bei i befindet) $r < r_{cut}$ gilt. Damit nicht gleichzeitig Wechselwirkungen zwischen i und j sowie i und einem Bildteilchen von j auftreten, muß $r_{cut} \le L/2$ gewählt werden. Gemäß Formel 5 läßt sich die Kraftberechnung vereinfachen. Für die Kraftberechnung ergibt sich Algorithmus 3.2. Anschließend an die Kraftberechnung werden die Summen und Mittelwerte zur Berechnung der makroskopischen Größen Energie und Druck gebildet.

Für die Parallelisierung kann es dennoch vorteilhaft sein, wenn die Vereinfachung nach Formel 5 nicht vorgenommen wird. Dann nennen wir den Algorithmus den *langsamen Kraftberechnungsalgorithmus*, den Algorithmus 3.2 den *schnellen*.

FOR i = 0 TO N-1

  $F_i = 0$

FOR i = 0 TO N-1

  FOR j = i+1 TO N-1

    /* Abstand r = $|\mathbf{r}_{ij}|$ zwischen Teilchen i und j berechnen */

    r = Distance(i, j)

    $r < r_{cut}$?

    yes

      $\mathbf{f}_{ij}$ = ForceBetweenParticle(i, j)

      /* Gesamtkraft, die auf ein Teilchen wirkt, summieren */

      $\mathbf{F}_i = \mathbf{F}_i + \mathbf{f}_{ij}$

      $\mathbf{F}_j = \mathbf{F}_j - \mathbf{f}_{ij}$

    no

Algorithmus 3.2 schnelleKaftberechnung mit sequentiellen Algorithmus Seq

FOR i = 0 TO N-1

  $F_i = 0$

FOR i = 0 TO N-1

  FOR j = 0 TO N-1 und $i \neq j$

    /* Abstand r = $|\mathbf{r}_{ij}|$ zwischen Teilchen i und j berechnen */

    r = Distance(i, j)

    $r < r_{cut}$?

    yes

      $\mathbf{f}_{ij}$ = ForceBetweenParticle(i, j)

      /* Gesamtkraft, die auf ein Teilchen wirkt, summieren */

      $\mathbf{F}_i = \mathbf{F}_i + \mathbf{f}_{ij}$

    no

Algorithmus 3.3 langsame Kaftberechnung mit sequentiellen Algorithmus

4 Parallelisierung mit gemeinsamen Speicher

Hier wird nun die Parallelisierung nach dem Lagrange-Schema vorgenommen (vgl. [Hofm92],[HoKn92],[Buch92]).

Die Grundidee ist, daß die Moleküldaten im gemeinsamen Speicher abgelegt werden und so von allen Prozessoren gelesen werden können.

Wie in [Hofm92],[HoKn92],[Buch92] wird die i-Schleife der Kraftberechnung im Algorithmus 3.2 bzw. im Algorithmus 3.2 aufgeteilt.

4.1 Datenabhängigkeiten und -modifikationen

Die Berechnungen der Kräfte $\vec{f}_{ij}$ für $0 \leq i,j \leq N$ ist, wie auch schon in [Hofm92],[HoKn92] und [Buch92] bemerkt wurde, voneinander unabhängig und erfolgt parallel.

Bei der Verwendung eines gemeinsamen Speichers hat man darauf zu achten, daß die Modifikation der globalen Variablen in diesem Speicher kontrolliert erfolgt.

Kritisch ist in diesem Programm nur die (schnelle oder langsame) Kraftberechnung. Die Prozeduren zur Initialisierung sowie zur Berechnung der Positionen und Geschwindigkeiten bei jedem Zeitschritt modifizierten nur die dem Prozeß zugeordneten Moleküle.

Bei der Kraftberechnung muß zwischen dem langsamen und schnellen Algorithmus unterschieden werden:

- Der langsame Algorithmus modifiziert nur die Kraftdaten f[i=aM..eM] des eigenen Prozesses, wobei aM die Nummer des ersten und eM die Nummer des letzten Moleküls ist.
- Der schnelle Algorithmus modifiziert zusätzlich noch f[j=i..N-1].

Damit muß beim schnellen Algorithmus durch Semaphore ein write-write-Konfikt verhindert werden, was die Parallelität einschränkt. Da die Kraftdaten in diese Phase nur geschrieben werden, werden die aufgrund Formel 5 berechneten Kraftdaten in einem globalen Hilfsfeld `fh` gesammelt, wobei dieses Hilfsfeld in Bereiche unterteilt ist, in dem nur ein Prozessor unter exklusiven Ausschluß diese Kraftdaten aufdatieren darf. Nach dieser Berechnungsphase addiert jeder Prozessor die Hilfsdaten zu die ihm zugeordneten Kraftdaten.

4.2 Lastverteilung

Bei der Verwendung eines gemeinsamen Speichers können ohne Schwierigkeiten in verschiedenen Phasen verschiedene Lastverteilungsschemata benutzt werden, da der Anwendungsprogrammierer sich nicht um das Umkopieren kümmern muß. (Bei UMA-MIMD-Rechnern muß auch das Betriebssystem nichts tun, bei NUMA erledigt das Betriebssystem bzw. die Hardware relativ effizient)

Die generelle Lastverteilung erfolgt genauso wie in [Buch92], bzw. [Hofm92],[HoKn92] und entspricht dem eindimensionalen Lastverteilungschema 5.1 aus [Weber94].[1]

Diese einfache Lastverteilung wurde durch die Funktion `lastverteilung(N,P)` im Modul `last` realisiert und ersetzt nun die äquivalente umständlichere Lastverteilungsfunktion `SetMole()` aus [Hofm92],[Buch92].

4.3 Datenverteilung

4.3.1 bei UNIX_SYS V Version

Bei der UNIX SYSTEM V Version liegen die Moleküldaten in einem geteilten Speichersegment, das mittels `shmget(2)` angefordert wird und mit `shmat(2)` in den Adreßraum des jeweiligen Prozesses eingebunden wird (geschieht in den Funktionen `DataInit` und `DataAttach` im Modul `datainit.c`). Eine genauere Datenverteilung kann und wird nicht vorgenommen.

4.3.2 bei Multithread-Version

Bei der Multithreaded Version ist beim Einsatz von mehreren Nodes die Verteilung bedeutend. Es erscheint vorteilhaft, die einem Thread auf einem Node zugeordneten Moleküle die Speicherklasse `near_shared` zu geben, da der zugeordnete Thread auf diesen Daten am meisten rechnet. Hierfür ist aber eine aufwendigere Molekülverwaltung erforderlich, da die Molekülfelder in dieser Implementierung dann zerteilt werden müßten. [2]

Eine Alternative ist die Wahl der Speicherklasse `block_shared`, die für Daten geeignet sein soll, die von allen Threads zugreifbar sein müssen, aber von bestimmten Threads stärker benötigt werden. Es ist anzunehmen, daß die physikalische Lokalisation vom Betriebssystem dynamisch festgelegt wird.

Auf die Daten im Krafthilfsfeld `fh` wird mit Ausnahme der Initialisierung und beim Aufaddieren gleichmäßig von allen Threads zugegriffen. Hier kommt als Speicherklasse alternativ auch `far_shared` in Frage.

4.4 Notwendige Koordinierungen

Verschiedene Berechnungsphasen müssen voneinander getrennt werden. Die Berechnung der Positionen und Geschwindigkeiten der Moleküle darf erst dann erfolgen, wenn alle Prozessoren die Kraftdaten berechnet haben. Ebenso müssen die Positionen und Geschwindigkeiten berechnet sein, bevor mit der Berechnung von Kraftdaten begonnen werden kann.

1. Hier zeigt sich die Wiederverwendbarkeit von Konzepten!
2. Hier würde sich eine objektorientierte Implementierung anbieten, die beim Zugriff vom Speicherort abstrahiert (z.B. mit sog. Iteratorabstraktion)

molshm.doc: 19.06.95 19:21

Dieses kann mit einem **Barrier-Konstrukt** erreicht werden.

Dann wird noch eine Funktion **MasterBroadcast** zum versenden von Daten und eine Funktion **SumDouble** zum aufsummieren eines Wertes benötigt.

4.4.1 Koordination mit den UNIX-System V - Mechanismen

4.4.1.1 Barrier-Synchronisation

Aufgabenstellung

Gegeben sei eine Menge von Prozessen $\mathbf{P} = \{0,...,P-1\}$ mit $|\mathbf{P}| = P$. Diese sollen sich an einem so synchronisieren, daß sie sich zu einem Zeitpunkt T_B an speziellen Punkt B im Programmcode gleichzeitig sein. Dieses wird dadurch erreicht, daß der erste Prozeß, der diesen Punkt erreicht, auf die anderen P-1 Prozesse wartet.

Lösung mit Vektoradditonssystem

Daten: $v \in \mathbf{P}^P$, anfangs: $v = (0,0,..,0)$

Operation für Prozeß $p \in \mathbf{P}$:

B1: warte bis $v[p] = 0$

B2: $e_{p,i,2}[p] = P$ für $i=p$; $e_{p,i,1}[j] = 0$ für $j \in \mathbf{P} \wedge j \neq p$

B3: $e_{p,i,3}[j] = -1$ für $j \in \mathbf{P}$

Behauptung: Diese Operationen hintereinander ausgeführt leisten das Gemeinte.

Bemerkung: Die das Warten darauf, daß $v[p] = 0$ ist, dient zur Vermeidung von Überholvorgängen bei mehrfacher mittelbarer Hintereinanderausführung der `barrier`-Funktion.

Die Implementierung mit UNIX-System V -Semaphoren kann dieser Spezifikation einfach folgen, da diese ein erweitertes Vektoradditionssystem darstellen. Die Erweiterung, daß der Aufruf mit Wert 0 solange den aufrufenden Prozeß blockiert, bis der entsprechende Semaphorkomponente den Wert 0 hat, implementiert die Operation B1.

Das Modul `barrier` implementiert dies.

4.4.1.2 Broadcast / Aufsummieren eines Wertes

Aufgabenstellung

Gegeben sei eine Menge von Prozessen $\mathbf{P} = \{0,...,P-1\}$ mit $|\mathbf{P}| = P$. Der Prozeß 0 sei der Meister genannt, von dem der Broadcast ausgeht.
T sei ein beliebiger Typ.
$W \in T^P$; W_p sei ein Teil der Daten des Prozesses p.

Beim sog. Broadcast müssen alle anderen Prozesse außer dem Master den Wert W_0 empfangen:

$\forall\ p\ \in \mathbf{P}: W_p := {}^{\backprime}W_0$

Beim Aufsummieren soll gelten (Dabei kann auch eine Andere Operation gewählt werden, T muß ein Typ mit einer entsprechenden Operation sein):

$$\forall\ p\ \in \mathbf{P}: W_p = \sum_{i \in P} W_i$$

Die genaue Aufgabenstellung hängt mit der Rechnerarchitektur zusammen.

Lösungen mit gemeinsamen Speicher:

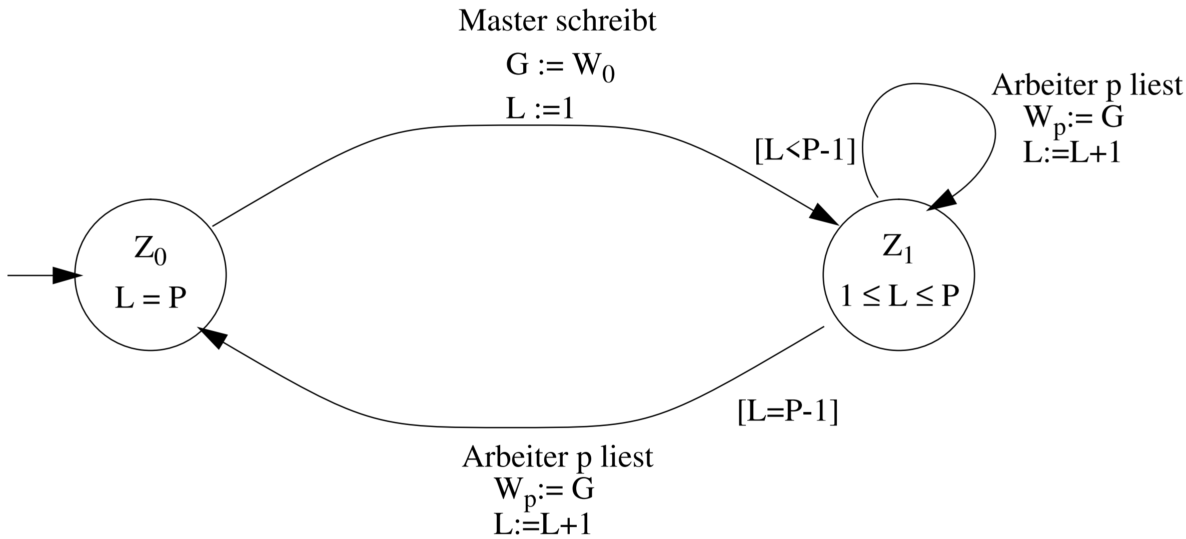
Die Prozesse besitzen einen gemeinsamen Speicher, in dem ein globaler Speicherplatz $G \in T$ befindet. Es wäre schon, wenn beim Broadcastproblem und beim Aufsummierungsproblem möglichst viel gemeinsam verwendet werden könnte.

Bedingungen für Broadcast-Problem:

- Master muß zuerst schreiben, bevor die andern lesen.
- Der Master darf nicht schreiben, bevor die anderen den Wert gelesen haben.

L sei die Zahl der Prozesse, die schon gelesen haben. (der Master hat schon gelesen, L>0)
N sei die Zahl der Prozesse, die noch lesen wollen.
Es gilt die Invariante: L+N = P

Es ergibt sich folgendes Zustandsdiagramm für den Broadcast:



Bedingungen für das Aufsummierungsproblem:

- der Anfangswert muß gesetzt werden. (Dies soll der Master machen).
- Der Master darf den Anfangswert nur setzen, wenn alle Prozesse den alten Wert lesen konnten.

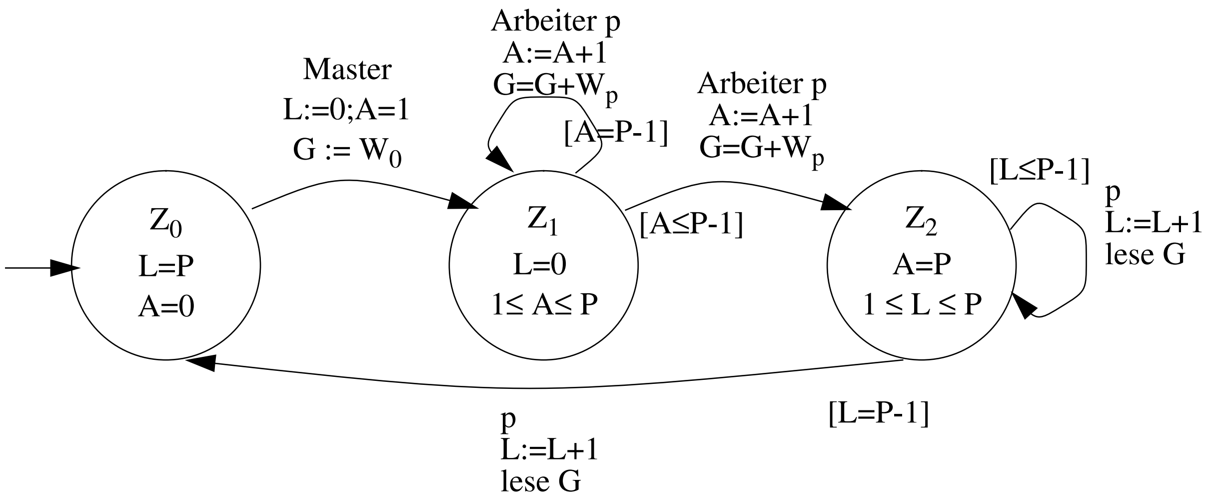
molshm.doc: 19.06.95 19:21

- Bevor die Prozesse den Aufsummierten Wert lesen, müssen alle geschrieben haben.

L sei die Zahl der Prozesse, die gelesen haben.

A sei die Zahl der Prozesse, die aufsummiert haben.

Ebenso ergibt sich folgender Zustandsgraph für das Aufsummierungsproblem:



Implementierung mit Semaphoren

a) Broadcast

Daten:

```
Semaphore L=P;     /* Zahl der Prozesse, die gelesen haben */
Semaphore B=0;     /* Prozesse, die noch nicht gelesen haben.*/
```

Master:

```
P(L:P);
G:= W0;
V(B:P-1);V(L:1);
```

Arbeiter:

```
P(B:1)
lese G
V(L:1);
```

b) Aufsummmierung:

Daten:

```
Semaphore L=P;     /* Zahl der Prozesse, die gelesen haben */
Semaphore A=P;     /* Zahl der Prozesse, die Aufdatiert haben*/
Semaphore NA=0;    /* P-A */
Semaphore M=1;     /* Exklusiver Ausschluß beim Aufdatieren */
```

Master:	Arbeiter:
`P(L:P);P(A:P)`	`P(NA:1):P(M:1)`
`G:=` W_0`;`	`G:=G+`W_p
`V(NA:P-1);V(A:1)`	`V(M:1);V(A:1)`
`P(A:P)`	`P(A:P)`
`lese G`	`lese G`
`V(L:1),V(A:P)`	`V(L:1);V(A:P)`

Bemerkung: Überholvorgänge sind nicht erlaubt, was durch vorschalten der Barrier-Funktion aus Abschnitt 4.3.1 erreicht wird.

Die Implementierung befindet sich im Modul `dataex.c`

4.4.2 Koordination mit der AIL auf CONVEX-SPP

Aus den Koordinationsprimitiven der Architecture-Interface-Library (AIL) `sema(3x)` [ConvON] sollen nun die Koordinationsroutinen `MasterBroadcast` und `SumDouble` sowie `barrier` konstruiert werden.

4.4.2.1 Algemeines

Aus der Manual-Page `sema(3x)` geht hervor, das die Primitiv-Routinen auf den sog. Semaphorevariablen direkt atomar im Hauptspeicher operieren und so die sonst notwendigen Cache-Kohärenzmaßnahmen beim Zugriff von mehreren Prozessoren vermeiden.

Diese “Semaphorevariablen” werden im folgenden als “atomare Zähler” bezeichnet. [3]

Als Anwendungsprogrammierer hat man zu vermeiden, daß die sog. Semaphorevariablen nie in den Cache geladen werden und daß sie auf 16 Byte-Grenzen legen müssen. Da auch zwischen den Nodes Daten ausgetauscht werden, sollte diese Variablen auf 64-byte-Blocke ausgerichtet sein. Da der CONVEX-SPP keine Alignment-Direktiven hat, muß man dies selber tun. Dies wird vorerst mit einem einfachen Prototyp einer blockweisen Allokationsroutine[4] erledigt.
Im Modul `am.h` wird ein Pool von Speicher verwaltet, der mit der Funktion `am_init` initialisiert wird. Die Funktion `am_alloc` liefert dann einen Zeiger auf Speicherbereich, der auf eine 2er Potenz - Adresse beginnt.

Das Modul `spprkoord.h` definiert nun die Typen `AtomarCounter` und `Spinlock`:

```
typedef unsigned int *AtomarCounter;
typedef AtomarCounter Spinlock;
```

3. In der objektorientierten Programmierung bietet sich die Definition einer Klasse “atomare Zahler” an
4. Laut Online-Manual allokiert die malloc-Funktion der C-Runtime-Library einen Speicherblock, der an einer 64-Byte Adresse beginnt. Die hier verwendete Allokationsroutine soll ein erster Prototyp für eine Metaklasse sein, die auf verschiedene Rechner anpassbar sein soll.

Der Typ `AtomarCounter` dient im wesentlichen nur zur klareren Schreibweise in den Programmen. In dem Modul `spprkoord.c` wurde nur die Initialisierungsroutine `ac_init(AtomarCounter ac,unsigned int val)` definiert, die über einen Aufruf von `am_alloc` (Modul `am.c`) richtig ausgerichteten Speicherplatz holt und den Zähler mit dem Wert `val` vorbelegt. Anschließend wird mit einem Aufruf der AIL-Funktion `cache_flush_region` dafür gesorgt, daß sich der Speicherblock nicht mehr im Cache befindet. Sonst können darauf die AIL-"Semaphore"-Funktionen aufgerufen werden.

Der Typ `Spinlock` dient zum Erreichen von exklusiven Ausschluß.
Die Funktion `sl_init(Spinlock s)` allokiert einen Spinlock und setzt ihn als frei.
Die Funktion `sl_P(Spinlock s)` prüft einen Spinlock, ob er frei ist, und wartet ggf. solange, bis dies der Fall ist. (Sie führt solange ein `fetch_and_clear` aus, bis ein Wert größer 0 zurückgeliefert wird).
Die Funktion `sl_V(Spinlock  s)` gibt einen solchen Spinlock frei (durch `fetch_and_inc`)

4.4.2.2 Barrier

Die AIL stellt eine einfache barrier-Funktion namens `barrier_sync(3)`. Wie man aus dem Pseudocode des Manuals entnehmen kann, droht die Verklemmung bei Überholvorgängen. Falls ein Thread verdrängt ist, so kann er die Freigabe durch die anderen Threads nicht mitbekommen. Die Threads, die laufen konnten, blockieren sich dann in der folgenden Barrier, während die verdrängten Threads sich noch in der vorherigen Barrier befinden. Dadurch kommt die Belegung der sog. Semaphore durcheinander und der Prozeß verklemmt sich.

Dieses Problem wurde nun dadurch umgangen, daß die in `barrier.h` definierte Barrier-Funktion zwei Barrierdaten wechselweise benutzt. Jeder Thread hat lokal einen binären Zähler, der angibt, welche Daten benutzt werden.

4.4.2.3 MasterBroadcast

Die Problemdefinition ist wie in Abschnitt 4.4.1.2. Wegen der notwendigen Barrier kann auf eine Koordination der Leser mit dem Schreiber verzichtet werden:

```
thread_private double w;            /* lokaler Wert */
static far_shared double g;         /* Globale Variable */
static far_shared AtomicCounter L;  /* Zahl der Leser */
```

Schreiber:

```
    barrier();
    g = w;
    for(i=1;i<P;i++)         /* Gebe für P-1 Leser frei */
      fetch_and_inc(L);
```

Leser:

```
    barrier();
```

```
    /* warte auf freigabe */
    while(fetch_and_dec(L) <= 0) { fetch_and_inc(L);}
    w = g;
```

4.4.2.4 SumDouble

Die Problemdefinition ist wie in Abschnitt 4.4.1.2. Ein Zähler S zählt die Summierer. Der erste Summierer setzt den Anfangswert, die restlichen addieren ihren lokalen Wert dazu. Der letzte Summierer setzt eine globale Variable hoch. Die Threads warten, bis eine Variable `rel` hochgesetzt wurde. Das Hochzählen der Summierer und das Aufaddieren muß unter exklusiven Ausschluß geschehen, da sonst nicht gewährleistet ist, daß der erste Prozeß auch wirklich der erste ist (er könnte z.B. inzwischen verdrängt werden). Das Ausummieren muß ohnehin unter exklusiven Ausschluß stattfinden, da sonst die Threads sich gegenseitig das Ergebnis überschreiben. Die Zahl der Summierer und die obige Variable `rel` werden wieder auf 0 gesetzt werden. Da sich die Threads nicht gegenseitig überholen dürfen, muß vorher eine Barrier-Synchronisation durchgeführt werden.

```
thread_private double w;            /* lokaler Wert */
static far_shared double g;         /* Globale Variable */
static far_shared int P;
static far_shared int rel;
static far_shared AtomicCounter S;  /* Zahl der Summierer */
static far_shared Spinlock M;       /* Exklusiver Ausschluß */
int r;
barrier();
rel =0;
P(M);
r=fetch_and_inc(S);
if( r== 0)
{
    g = w;
}
else
{
    g += w;
    if(r==P-1) rel=1;
}
V(M);
while(rel!=1);  /* Warte auf alle */
if(p==0) fetch_and_clear(S); /*Zahler wieder zurücksetzen */
w=g;
```

molshm.doc: 19.06.95 19:21

5 Messungen

5.1 Meßgrößen und Meßmethode

Def 6: ***Speedup und Effizienz***

Sei T_P die Rechenzeit des Programms mit P Prozessoren.
Der Beschleunigungs- oder Speedup-Faktor ist [5]

$$S_P = \frac{T_1}{T_P}$$

und die Effizienz

$$E_P = S_P = \frac{T_1}{P \cdot T_P}$$

Für jede Prozessorzahl wurde das Programm 10 mal ausgeführt und die Zeiten gemittelt.
Der Versuch wurde mit einem `perl(1)` - Script gesteuert. Dieses extrahierte die Zeiten aus der Ausgabe des Programms. Die Größen aus Def 6 wurden mit Hilfe eines `perl(1)` - Scripts `speedup.pl` berechnet, welches zur Mittelung der Zeiten das vom Autor entwickelte Programm `grupstich(1)` aufruft.

5.1.1 Zeitmessung mit UNIX-System V

Als Zeiten wurden die Realzeiten gemessen mit `times(3)` (UNIX-SYSTEM V) genommen, da nur so die Blockierungszeiten mitberücksichtigt werden. Dabei gehen aber leider auch die Zeiten ein, in den alle Prozesse dieser Anwendung suspendiert sind. (Zur der Zeit war aber laut **who(1)** kein anderer Benutzer auf dem Rechner eingelogt).

5.1.2 Zeitmessung im Multi-Thread-Programm

Der CONVEX-SPP hat einen 64-bit-Zähler, der jede Mikrosekunde inkrementiert wird, in der mindestens ein Thread der Anwendung sich im User-Modus befindet. Dieser kann mit der AIL-Funktion `pvt_read(3)` gelesen werden.
Dann gibt es noch auch dem SPP einen 64-bit-Zähler, der jede Mikrosekunde inkrementiert wird und die Mikrosekunden seit 0 Uhr 0 1.1.1970 enthält. Dieser wird für die RealTime-Messungen eingesetzt. Gelesen wird er mit `toc_read(3)`.

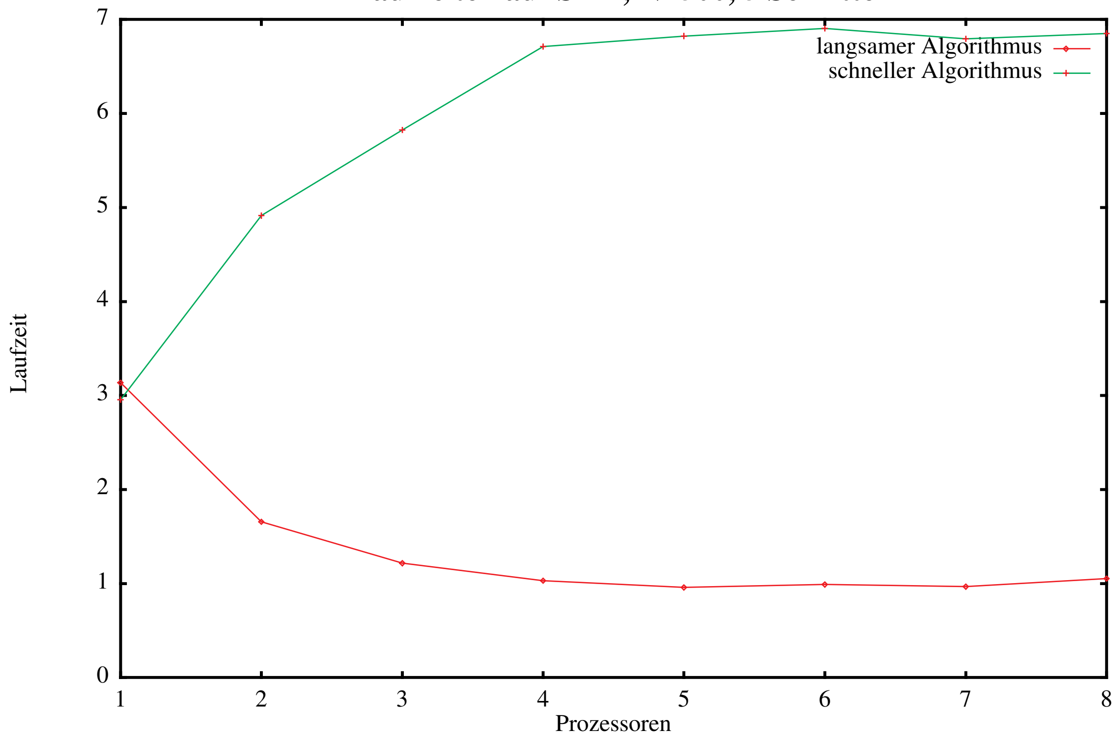
Die Zeitmessung wird im Modul `zeitmess.c` vollzogen (`zm_init` zur Initialisierung, `zm_print` zur Messung und Ausgabe, und `zm_exit` zur Beendigung; es wird jeweils die Zeit zwischen zwei `zm_print`-Aufrufen und die bisherige Gesamtzeit ausgegeben).

5. Manche Autoren setzen in den Formeln statt T_1 die Zeit des (schnellsten) sequentiellen Algorithmus T_S ein. Pragmatiker nehmen aber oft wie hier die Zeit auf einem Prozessor.

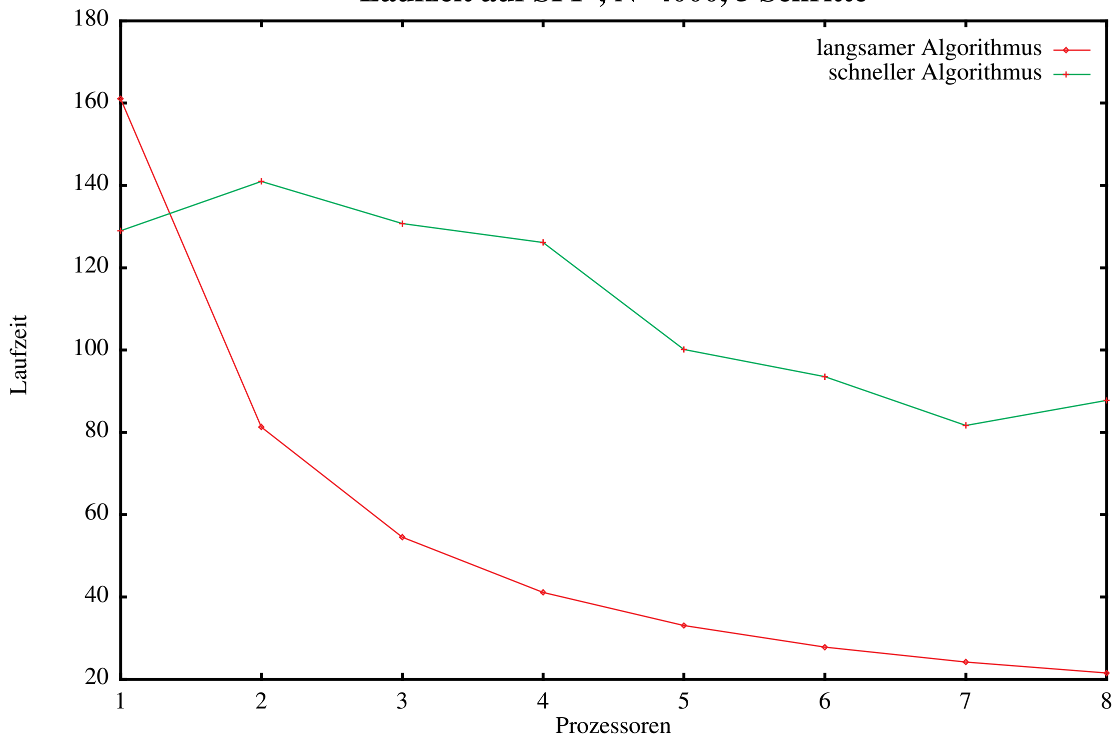
5.2 Meßergebnisse der SYSTEM V-Version in Schaubildern

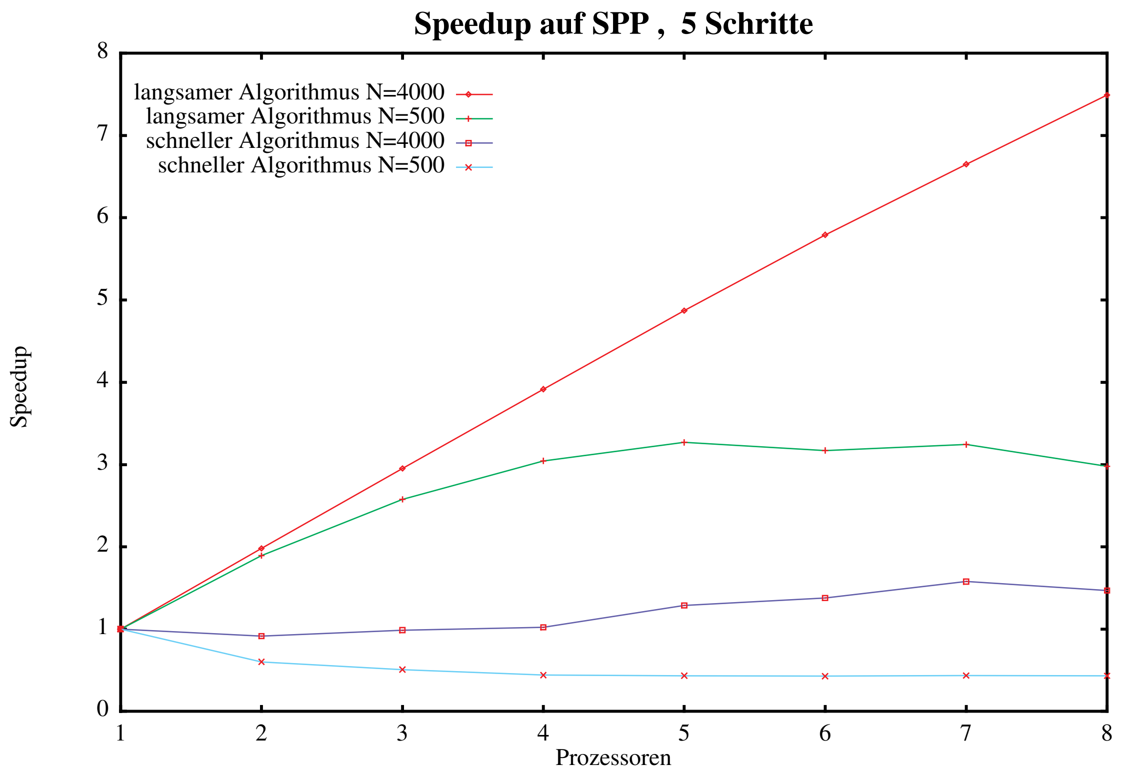
Im folgenden sei N die Zahl der Teilchen. Die Schritte sind die Zeitschritte.

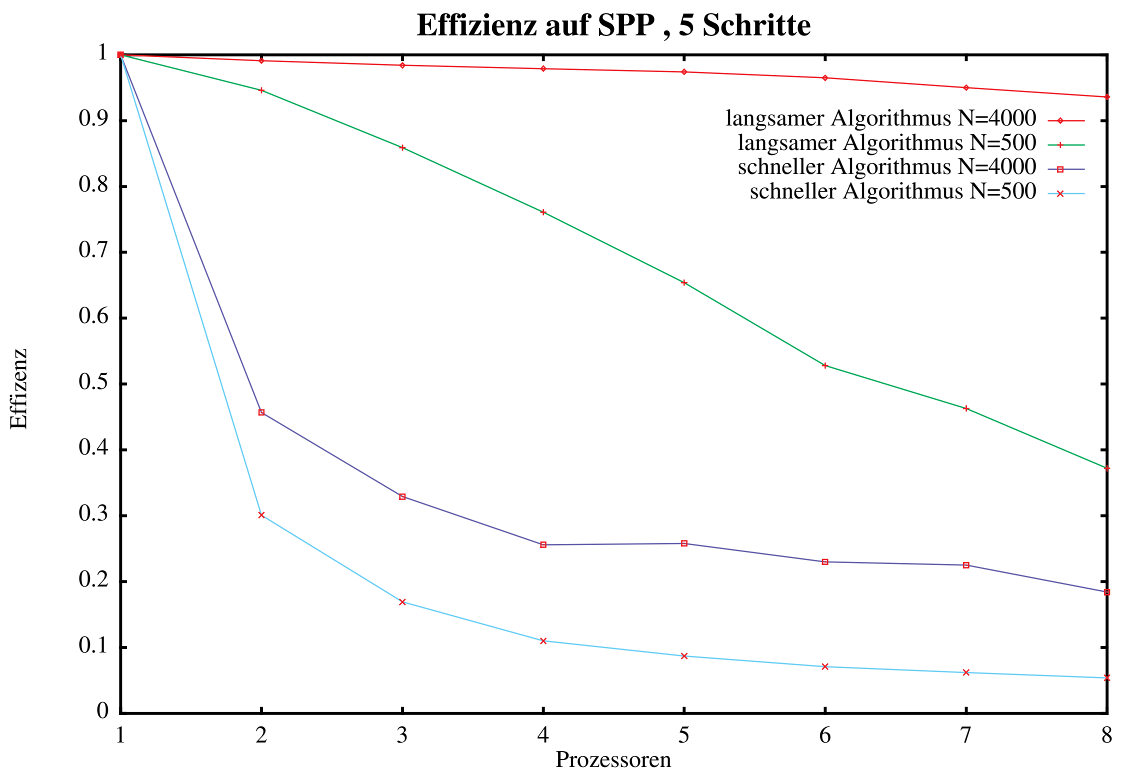
Laufzeiten auf SPP , N=500, 5 Schritte



Laufzeit auf SPP , N=4000, 5 Schritte

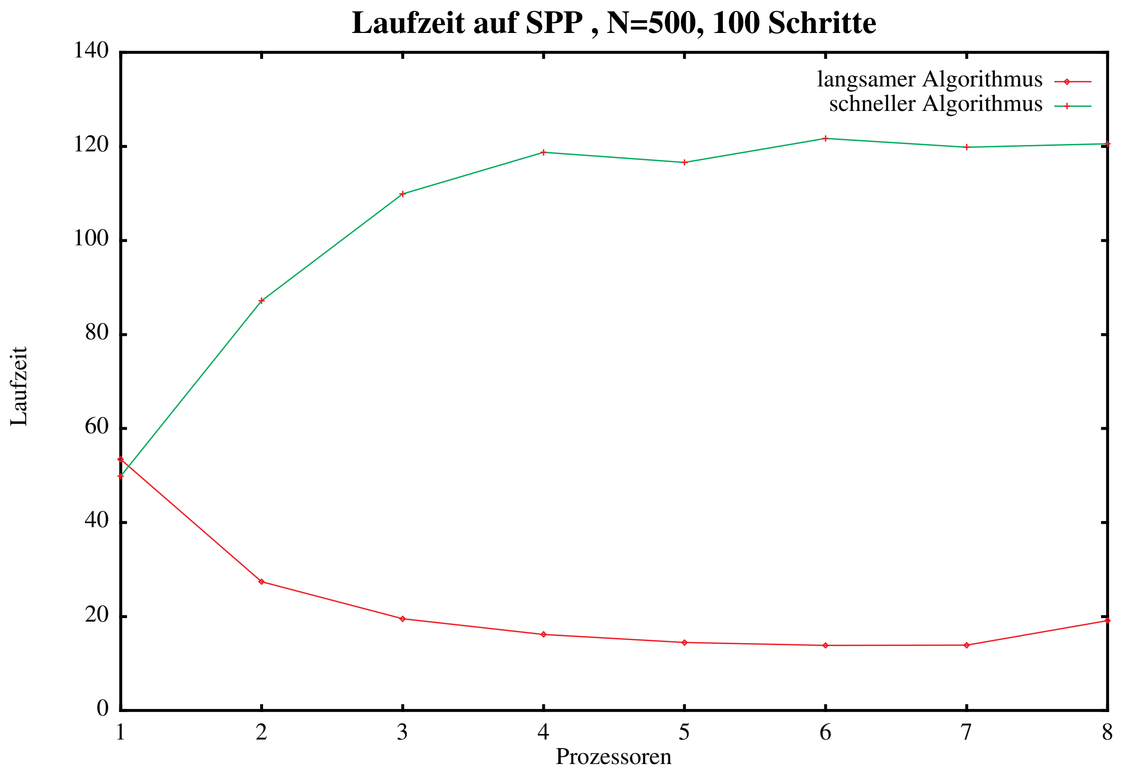


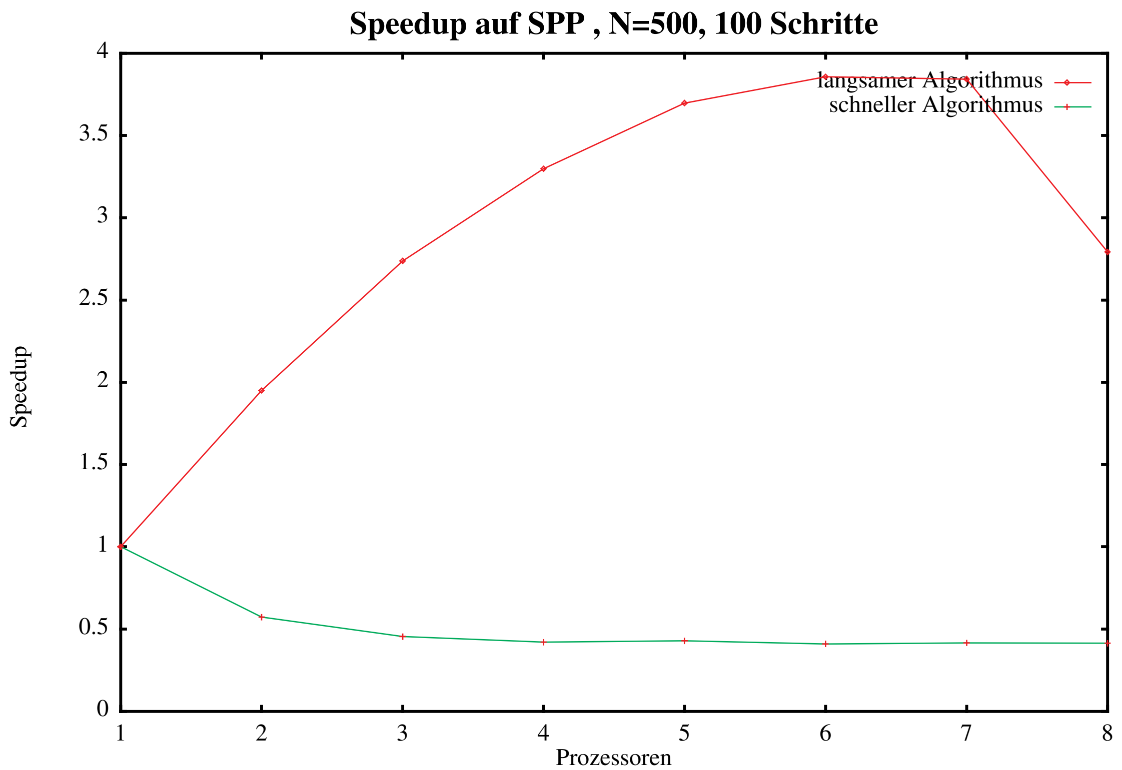




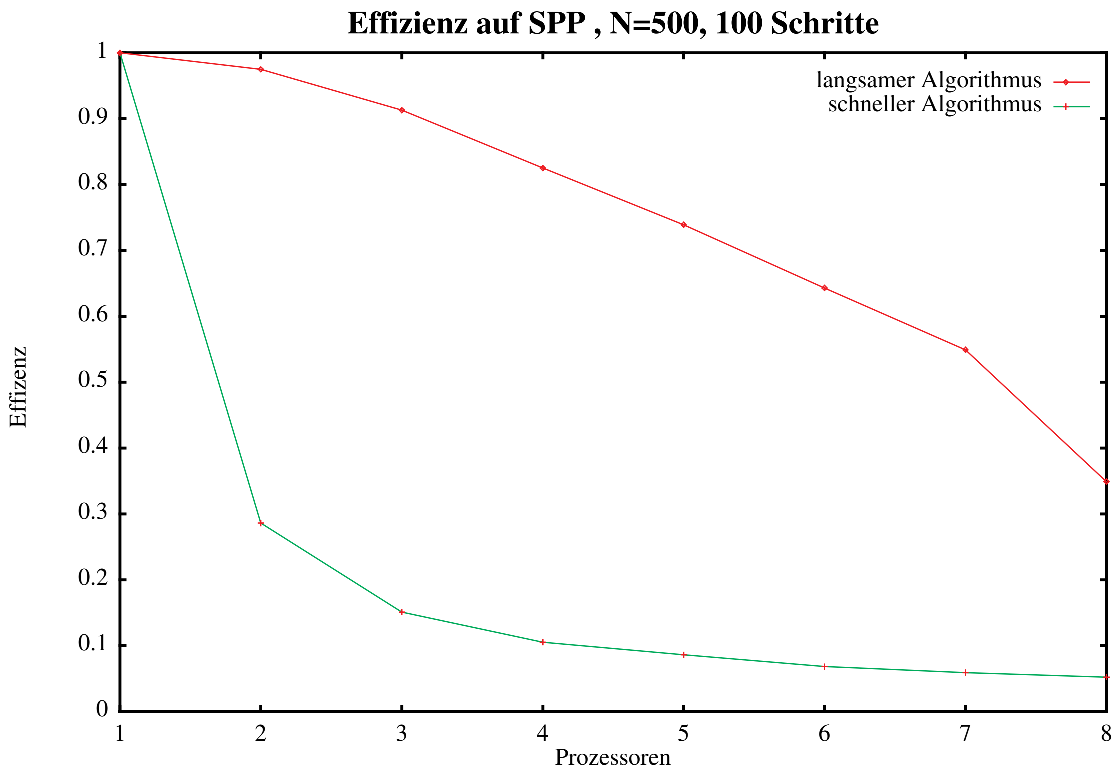
molshm.doc: 19.06.95 19:21

Mit 100 Zeitschritten zeigt sich ein ähnliches Ergebnis, nur verzögert sich der Einbruch der Speedupkurve:





molshm.doc: 19.06.95 19:21



5.2.1 Interpretation

Wie aufgrund der Implementierung zu erwarten war, ist wegen der Notwendigkeit des exklusiven Ausschluß bei den Kraftdaten des “schnellen Algorithmus “der “schnelle Algorithmus” langsamer als der “langsame” Algorithmus.
Erschwerend kommt beim schnellen Algorithmus noch die ungleiche Lastverteilung bei der j-Schleife des Algorithmus 3.2 hinzu.

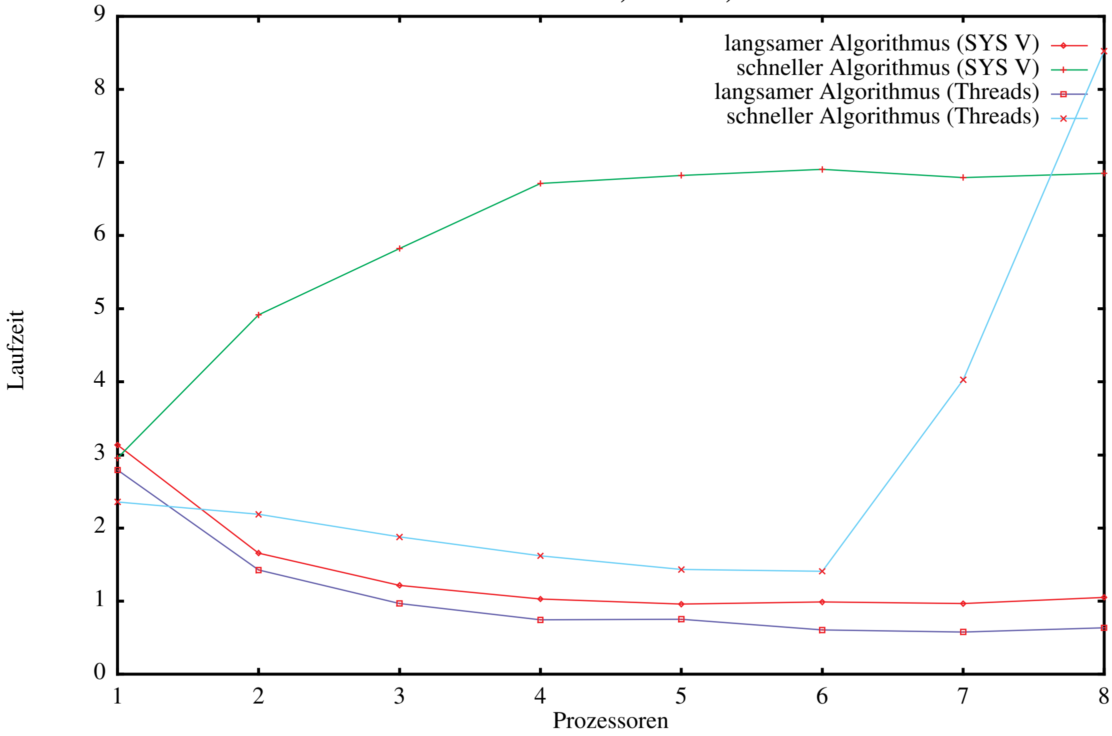
Man kann aus den Graphiken entnehmen, daß der Speedup mit zunehmender Granularität steigt.

Ansonsten zeigen die Kurven das übliche Verhalten auf.

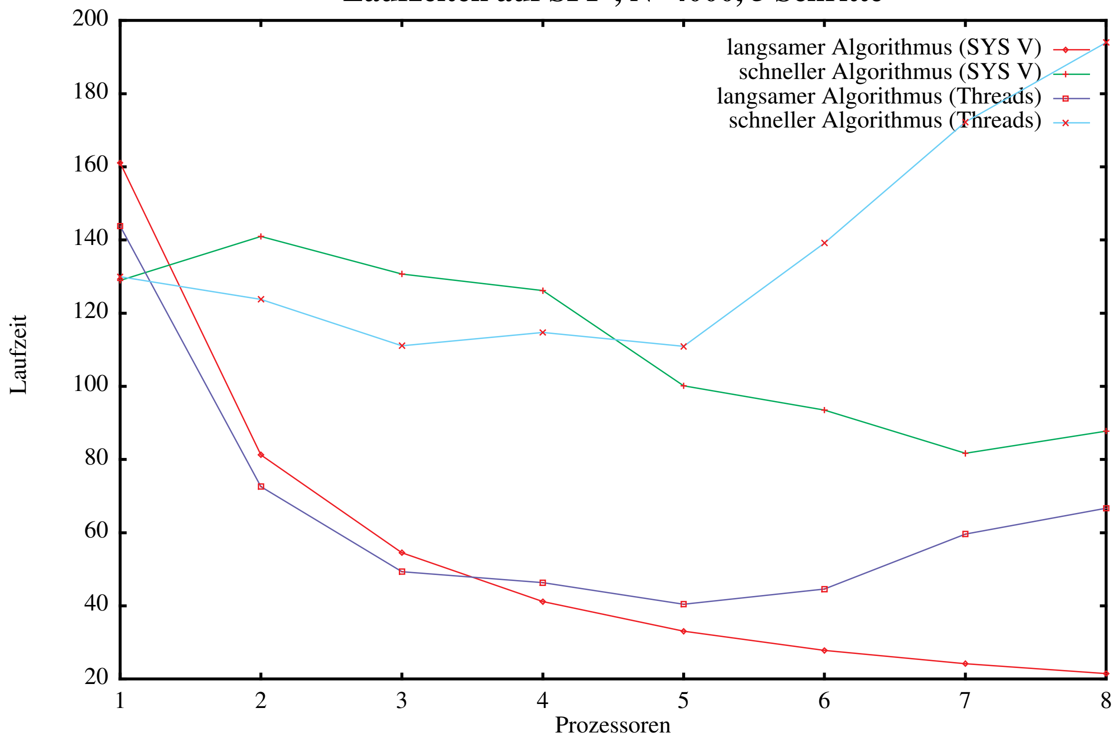
5.3 Meßergebnisse der Multithread-Version in Schaubildern

In den drei folgenden Schaubildern wird die Laufzeit der beiden Algorithmen in beiden Realisierungsarten miteinander verglichen:

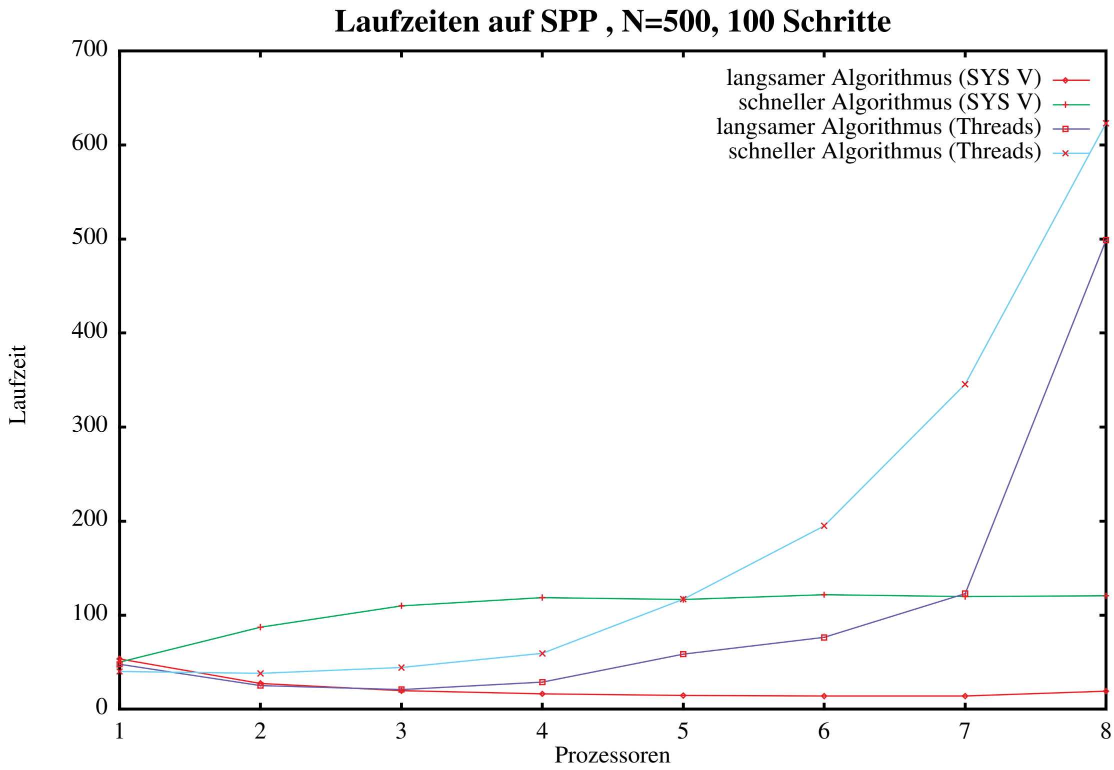
Laufzeiten auf SPP , N=500, 5 Schritte



Laufzeiten auf SPP , N=4000, 5 Schritte



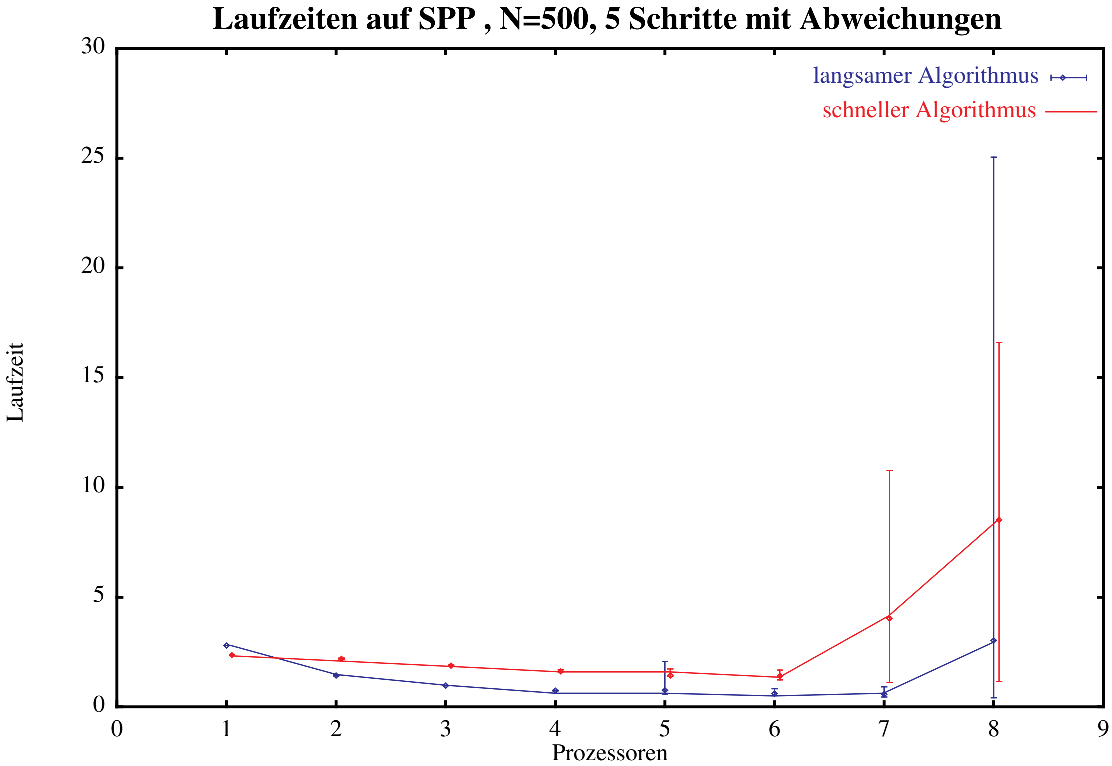
molshm.doc: 19.06.95 19:21



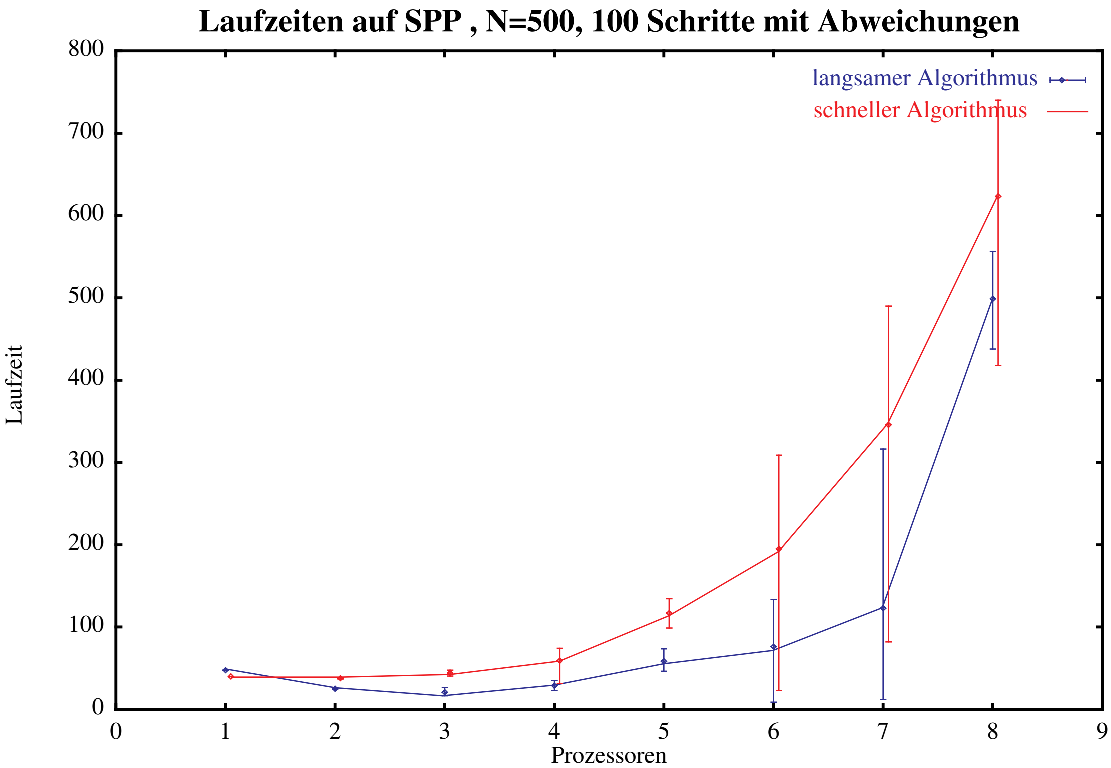
Es läßt sich entnehmen, daß für 2 und 3 Aktivitätsträger die Multithread-Implementierung effizienter ist. Das ungewöhnlich erscheinende schlechtere Verhalten der Multithread-Implementierung ist vermutlich auf die Implementierung der Koordination zurückzuführen.

Eine Untersuchung der Varianzen und Standartabweichungen (mittels `grupstich(1)`) zeigt, daß diese Werte mit der Zahl der Prozessoren zunehmen. Dies ist ein weiterer Hinweis, das die Implementierung der verwendeten Koordinationsmechanismen verbessert werden muß.

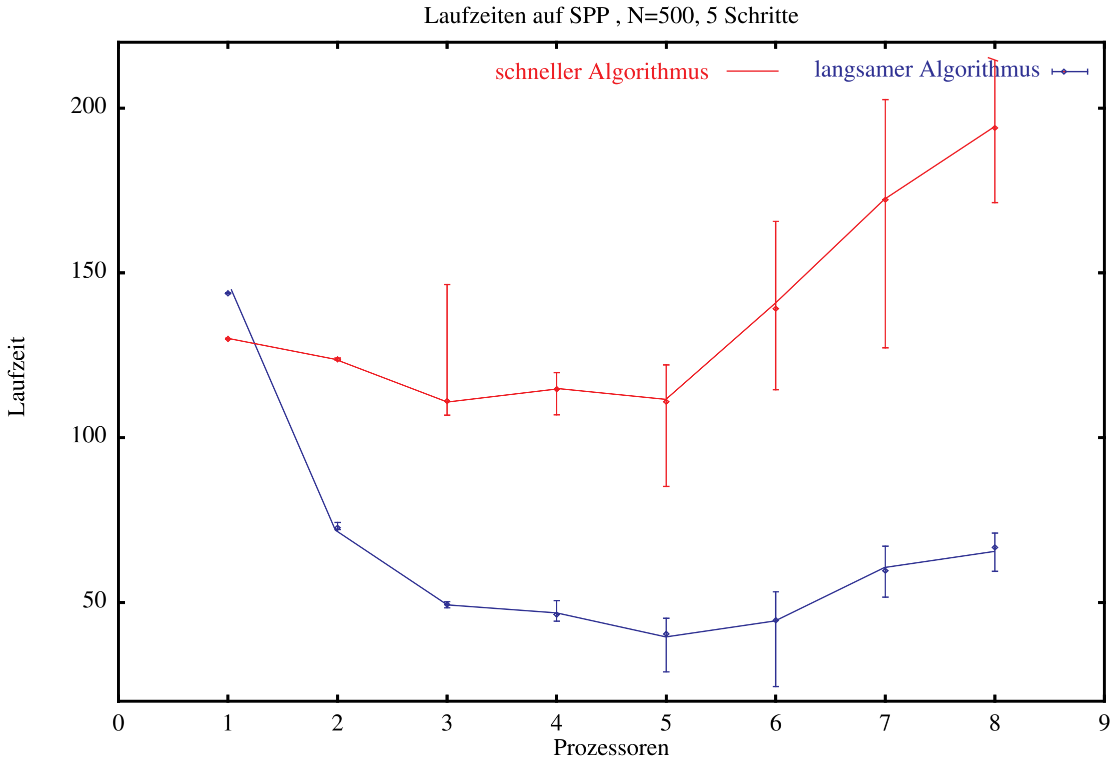
In den folgenden Schaubildern sind die Kurven mit den Multithreadergebnissen mit Minimum- und Maximum aufgetragen:



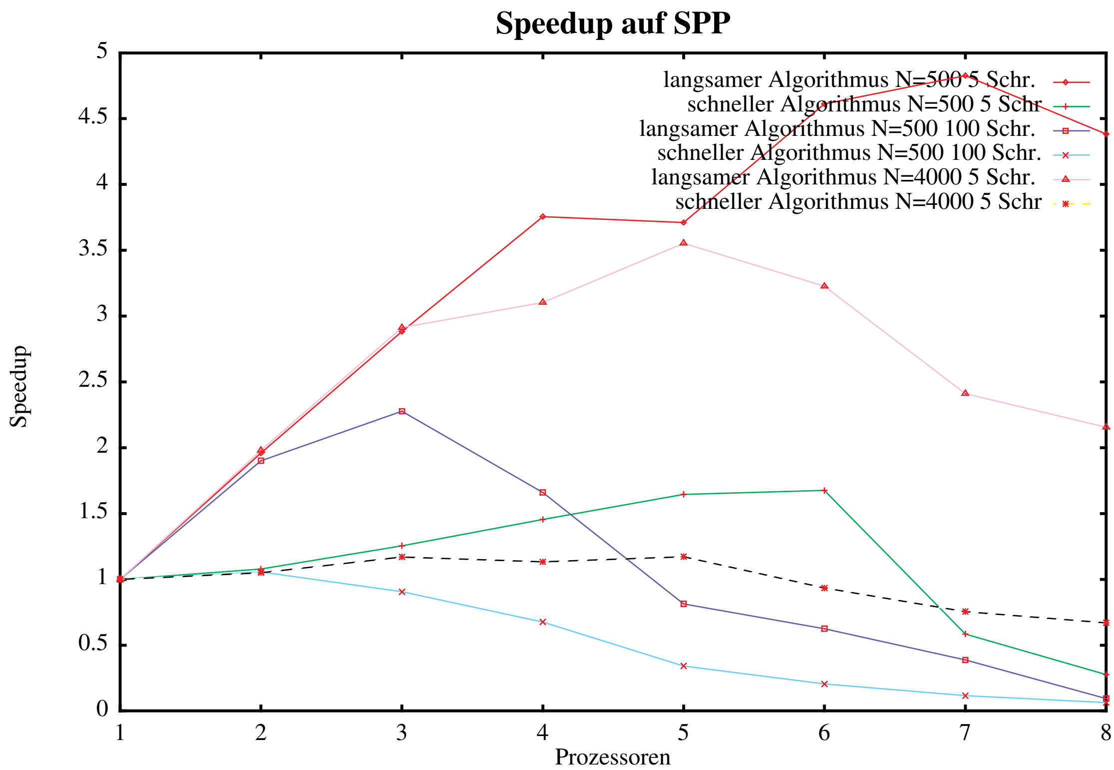
Laufzeiten auf SPP , N=500, 5 Schritte mit Abweichungen
langsamer Algorithmus
schneller Algorithmus
Laufzeit
Prozessoren

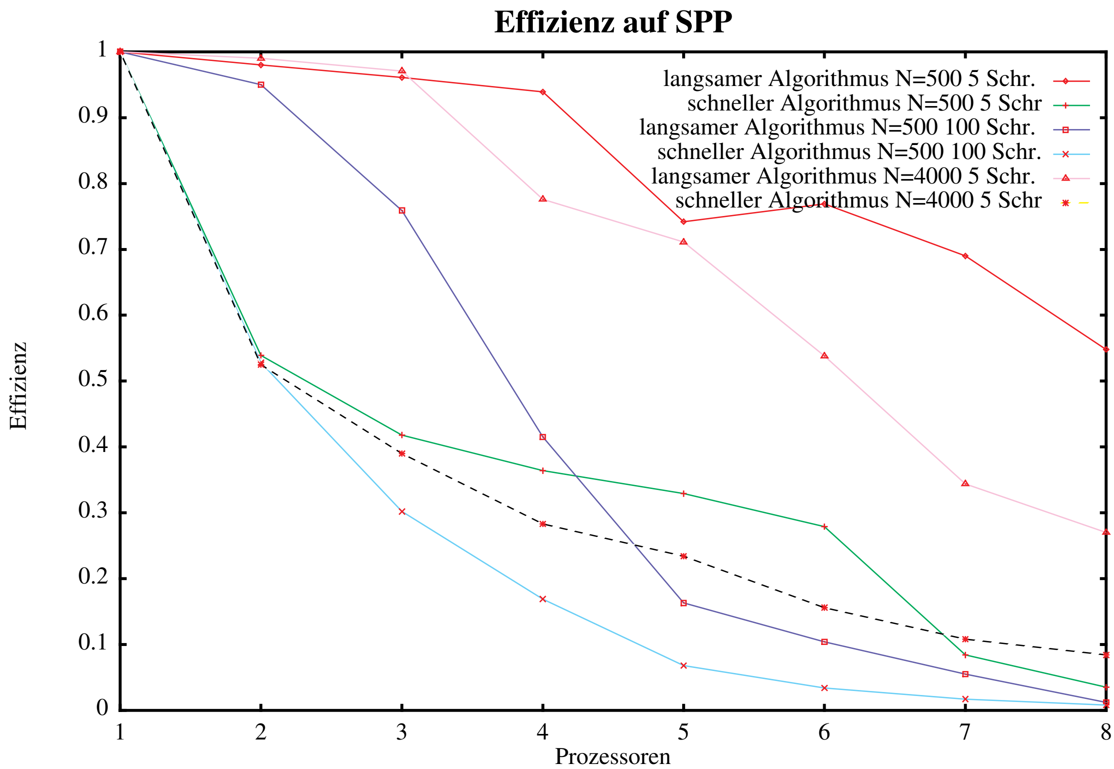


Laufzeiten auf SPP , N=500, 100 Schritte mit Abweichungen
langsamer Algorithmus
schneller Algorithmus
Laufzeit
Prozessoren



molshm.doc: 19.06.95 19:21





molshm.doc: 19.06.95 19:21

6 Vergleich mit den anderen Implementierungen

- Wo bei den anderen ein Broadcast der Molekül- oder Kraftdaten stattfindet, wird nun eine Barrier gesetzt.
- Im Gegensatz zu den bisherigen Parallelisierungen gibt es den Sonderfall durch 4 teilbare Prozessorzahl nicht mehr.
- Der schnelle Kraftberechnung gleicht nun dem sequentiellen Fall. Bei den anderen Implementierungen hat die j-Schleife des Algorithmus 3.2 andere Grenzen; der Algorithmus liefert aber bei dieser Parallelisierung andere Ergebnisse.
  Ein Nachteil der hier verwendeten Parallelisierung ist die ungleiche Lastverteilung: Ein Prozeß, der Moleküle mit niedrigeren Molekülnummern mehr zu Arbeiten.
- Die Kraftberechnungsalgorithmen aus MEMSY unterteilen sich in mehrere Phasen, wo erst die Wechselwirkungen mit den Molekülen vor den eigenen, die nach den eigenen, und schließlich die der eigenen untereinander berechnet werden. Diese verschiedenen Phasen brauchen hier nicht unterschieden zu werden.
- Für die Berechnung der Wechselwirkung mit den eignen Molekülen wurde immer der schnelle Algorithmus ausgeführt. Dies wurde in dieser Implementierung beseitigt.

7 Fazit

Die Parallelisierung mit gemeinsamen Speicher ist einfacher als die Parallelisierung auf einem NORMA-System, hat aber auch seine Tücken. Das Programm ist wesentlich kürzer und einfacher verständlich.

8 Ausblick und Ideensammlung

(1) Objektorientierte Realisierung
Beim Betrachten von Kapitel 2 erscheint es sinnvoll, die Klasse Molekül zu definieren. Das Gesamtsystem ist eine weitere Klasse. Die Molekülverwaltung ist ein weiterer Vorschlag.

(2) Das Eulerschema nach [Hofm92], [Buch92] könnte mit gemeinsamen Speicher realisiert werden.

(3) Auf MEMSY könnte man so die Knoten nochmals parallelisieren, um alle 80 Prozessoren gleichzeitig nutzen zu können.

9 Bedienung des Programms

Mögliche Optionen sind:

`-o` filename	Angabe einer Ausgabedatei
`-f`	Wahl des schnelleren Algorithmus (Ausnutzung von actio = reactio)
`-p` number	Angabe der gewünschten Anzahl Prozessen/Threads/Tasks/Knoten
-a <rechner:port>	Anschluß des graphischen Animationsprogramms

In der Eingabedatei kann der Standardeingabeparametersatz durch den Benutzer verändert werden.
Die Standardeinstellung für die Eingabedatei ist die Dateien `inf` die auch als Beispieldatei dient.

Bei Angabe einer nichtvorhandenen oder fehlerhaften Eingabedatei wird das Programm abgebrochen. Standardausgabedatei ist `stdout`.

Folgende Liste gibt einen Überblick über die möglichen Eingabeparameter. Der Typ des Parameters ist in Klammern angegeben, ebenso der sinnvolle bzw. erlaubte Wertebereich. Kann man nur zwischen einer vorgegebenen Anzahl von Einstellungen wählen, so sind diese angegeben.

- `INITVELO` Wahl der Anfangsgeschwindigkeiten
  `fixed:` `m_v[i].u = (double)i * .9` $u \in \{x, y, z\}$
  `filed:` `m_v[i].x = (double)i * .5`
  `m_v[i].y = (double)i * .4`
  `m_v[i].z = (double)i * .3`
  `random:` zufällige Anfangsgeschwindigkeiten

- `TEMPSCALE` Temperaturskalierung (`yes`, `no`)

- `PSUBQUADER` Anzahl der Moleküle pro Subquader ([1; 4])

- `NQUADER` Anzahl der Subquader pro Raumrichtung ($\mathbf{N}^+$)
  Subquader sind auf Länge 1 normiert, Kantenlänge des betrachteten Raumes ergibt sich aus Anzahl der Subquader pro Raumrichtung

- `NMODULES` Anzahl der Module pro Raumrichtung (nur bei `euler`, $\mathbf{N}^+$)

- `STEPS` Anzahl der Zeitschritte ($\mathbf{N}^+$)

- `AVERAGE` Anzahl der Zeitschritte, nach der Mittelwerte berechnet werden sollen ($\mathbf{N}^+$)

- `DELTA` Länge eines Zeitschrittes ($\mathbf{R}^+$)

- `TEMPERATURE` Anfangstemperatur ($\mathbf{R}^+$)

molshm.doc: 19.06.95 19:21

- `DENSITY` Anfangsdichte ($\mathbf{R}^+$)
  Die Kantenlänge L des betrachteten Raums ergibt sich aus $L = \sqrt[3]{N/n}$ (Abschnitt 2.2)

- `RCUT` Cutoff-Entfernung ($\mathbf{R}^+$)

r_{cut} wird automatisch auf sein erlaubtes Maximum gesetzt, falls es zu groß gewählt wird. Bei Wahl der Eingabeparameter sollte bedacht werden, daß die kinetische Energie durch die Summe der Quadrate der Geschwindigkeitsvektoren aller Moleküle berechnet wird und daher bei großer Anzahl von Molekülen relativ schnell das Darstellungsvermögen des Rechners überschreitet. Um dies zu vermeiden, kann Temperaturskalierung `yes` gewählt werden.
Alle Ergebnisse werden vom designierten Masterknoten ausgegeben. Ausgegeben werden folgende berechnete Daten:

- Anzahl der Moleküle
- Kantenlänge des betrachteten Raumkubus
- potentielle Energie des Initialzustandes
- potentieller Druck des Initialzustandes
- Temperatur des Initialzustandes
- aktuelle Temperatur nach jeweils `AVERAGE` Zeitschritten
- Energie am Ende der Berechnung, Fehlergröße
- Druck am Ende der Berechnung, Fehlergröße
- Temperatur am Ende der Berechnung, Fehlergröße
- User-, System- und Gesamtlaufzeit der Hauptschleife

Die Fehlergröße ist eine Abschätzung für den Fehler, der gemacht wird, da man die Wechselwirkung zwischen je zwei Molekülen nur bis zu einem Abstand von r_{cut} berechnet.

10 Literatur

Buch92 Kerstin Buchacker: “Parallelisierung einer Molekulardynamikaufgabe auf dem MEMSY-Multiprozesor”;Studienarbeit IMMD4 1992

ConvON Convex-SPP -Online Manual

Frit90 G. Fritsch: Numerische Simulation unter Verwendung von Teilchenmodellen. PARAWAN Bericht, IMMD, Universität Erlangen-Nürnberg, 2. Hj. 1990

FrVo87 G. Fritsch, J. Volkert: Parallel Computing of Scientific Problems on Distributed Shared Memory Multiprocessors. Interner Bericht, IMMD III, Universität Erlangen-Nürnberg, 1987

Hess88 S. Hess: Computer-Simulationen von Transportprozessen und Analyse der Struktur von einfachen Fluiden. *Physikalische Blätter 44*, Nr. 8, pp. 325-330, Weinheim, 1988

Hess91 S. Hess: ohne Titel. Sequentielles Fortran Programm zur Molekulardynamik-Simulation, 1991

Hofm92 K. Hofmann: Parallelisierung einer dreidimensionalen Molekulardynamikauf-gabe auf Multiprozessoren mit verteiltem Speicher. Studienarbeit, IMMD III, Universität Erlangen-Nürnberg, 1992

HoKn92 K. Hofmann, R. Knirsch: An Improved Parallel Algorithm for the Solution of Molecular Dynamics Problems on MIMD Multiprocessors. Interner Bericht, IMMD III, Universität Erlangen-Nürnberg, 1992

SFB92 Sonderforschungsbereich 182: Multiprozessor- und Netzwerkkonfigurationen. Arbeits- und Ergebnisbericht für den Förderungszeitraum Jan. 1990 - Dez. 1992, Universität Erlangen-Nürnberg, 1992

vWav87 G.M. van Waveren: Application of sparse vector techniques on a Molecular Dynamics program. In H.J.J. te Riele, Th.J. Dekker, H.A. van der Vorst (Hrsg.): *Algorithms and Applications on Vector and Parallel Computers*. Elsevier Science Publishers B.V. (North Holland), 1987

Weber94 Rudolf Weber: “Objektorentierte Konzepte zur Parallelisierung strömungsmechanischer Anwendungen” Technischer Bericht TR-I4-94.17 IMMD IV