Implementierung adaptiver numerischer Verfahren auf komplexen Geometrien mit leichtgewichtigen Prozessen

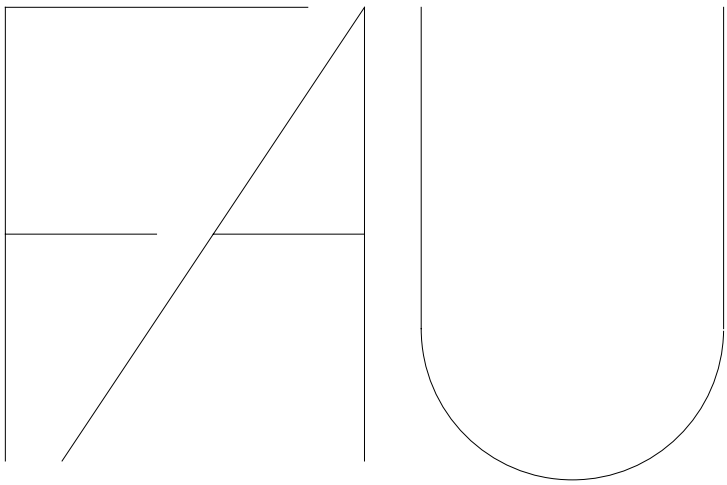
Frank Bellosa

Mai 1994 TR-I4-7-94

Interner Bericht

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

Lehrstuhl für Informatik IV
(Betriebssysteme)



Implementierung adaptiver numerischer Verfahren auf komplexen Geometrien mit leichtgewichtigen Prozessen

Frank Bellosa
email: bellosa@informatik.uni-erlangen.de

Universität Erlangen-Nürnberg
Lehrstuhl für Betriebssysteme

Zusammenfassung

Bei der Entwicklung effizienter paralleler Verfahren zur Berechnung komplexer Geometrien soll erreicht werden, daß die numerische Effizienz der Algorithmen mit der der besten sequentiellen Verfahren vergleichbar ist und dabei eine hohe parallele Effizienz erreicht werden kann.

Beim Einsatz von Rechnern mit verteiltem Speicher (**NO R**emote **M**emory **A**ccess Architekturen wie z.B. IBM SP1, Intel Paragon und Workstation Cluster) haben sich dabei Ansätze bewährt, bei denen das Problemgebiet in Blöcke zerlegt wird und diese Blöcke den verfügbaren Prozessoren zugewiesen werden. Die Kopplung der einzelnen Problemgebiete erfolgt durch einen Datentransfer zwischen den Prozessoren, bei dem die Randbereiche ausgetauscht werden.Kritisch ist bei diesem Ansatz das Partitionierungsproblem. Für das Problemgebiet muß eine Partitionierung in Blöcke gefunden werden, bei der die Rechenlast gleichmäßig verteilt und der Aufwand für die Kopplung der Teilgebiete minimal gehalten wird. Dies ist insbesondere bei sehr komplexen Geometrien, wie sie in realen Anwendungen auftreten, ein schwer lösbares Problem, so daß ein Wissenschaftler aus den Anwendungsgebieten im allgemeinen ohne langjährige Schulung kaum in der Lage ist, eine optimale Partitionierungsstrategie für sein Anwendungsfeld zu entwickeln. Um die Wettbewerbsvorteile durch das numerische Hochleistungrechnen auch kleinen und mittleren Unternehmen ohne große Forschungsabteilung zu eröffnen, muß nach neuen Techniken gesucht werden, damit effiziente numerische Verfahren für komplexe Problemgebiete einfacher auf Hochleistungsrechnern implementiert werden können.

Ein vielversprechender Ansatz liegt in der Verwendung leichtgewichtiger Prozesse, wie sie auf modernen Parallelrechnerarchitekturen mit einem gemeinsamem Speicher möglich werden. Beispiele hierfür sind die **U**niform **M**emory **A**ccess Architekturen wie z.B. SGI Power Challenge oder die **N**on **U**niform **M**emory **A**ccess Architektur der Rechner KSR2 und Convex SPP. Bei dem am Lehrstuhl für Betriebssysteme der Uni Erlangen entwickelten Programmiermodell muß sich der Anwendungswissenschaftler nicht mehr um eine explizite Partitionierung der Daten oder um die Verteilung der Rechenlast kümmern, sondern kann sich ganz auf die Beschreibung des numerischen Verfahrens konzentrieren. Wenn dabei die Beschreibung nach vorgegebenen einfachen Regeln erfolgt, ist ein als Bibliothek zum Programm gebundenes Laufzeitsystem in der Lage, die Aufgabe der Partitionierung und Lastverteilung für beliebige Prozessorzahlen zu übernehmen.Damit wird dem Anwender ein leistungsfähiges Werkzeug in die Hand gegeben, das es ihm ermöglich, eine optimale numerische und parallele Effizienz auf Hochleistungsrechnern mit gemeinsamem Speicher zu erzielen.

1 Stand der Forschung beim Einsatz komplexer Geometrien

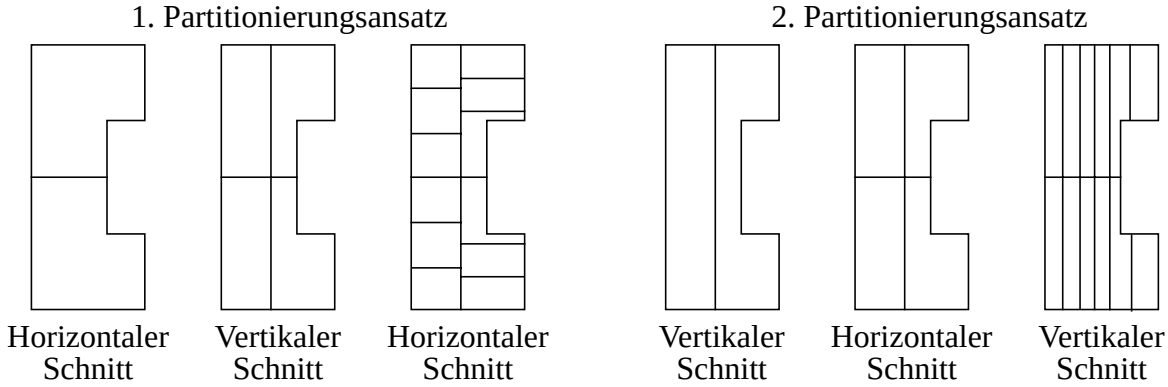
Die numerische Berechnung von Strömungen in einem komplexen Problemgebiet stellt extreme Anforderungen an die eingesetzte Hardware und Software. Leistungsfähige Recheneinheiten sowie eine Verbindungshardware mit hoher Bandbreite und geringer Latenz sind die Grundvoraussetzung für den Einsatz eines Parallelrechners in diesem Bereich der numerischen Simulation. Numerische Verfahren zur Lösung der bei der Berechnung von Strömungen anfallenden Gleichungssysteme, die auf Parallelrechnerarchitekturen mit geringer Bandbreite und hoher Latenz eine durchaus hohe parallele Effizienz aufzeigen(z.B: Red Black Gauß Seidel), weisen eine nur mäßige numerische Effizienz d.h.eine geringe Konvergenzrate auf. Numerisch effizientere adaptive Verfahren hingegen stellen sehr hohe Anforderungen an die zugrunde liegende Hardware und weisen dennoch häufig nur mäßige parallele Effizienzen auf. Es hat sich jedoch bei den Untersuchungen im vergangenen Forschungszeitraum herauskristallisiert, daß die Einsparung der Rechenzeit durch numerisch ausgefeilte Methoden bei weitem größer ist, als beim Einsatz vieler paralleler Prozessoren mit jedoch numerisch ineffizienten Berechnungsverfahren. Ziel der Forschung ist daher die Entwicklung von numerischen Verfahren, die mit einer minimalen Anzahl an arithmetischen Operationen auskommen und sich dennoch zum Einsatz auf modernen Parallelrechnerarchitekturen eignen.

Als Methoden zur Strömungsberechnung haben sich iterative Lösungsmethoden auf der Finite Volumen Basis etabliert. Um diese Lösungsmethoden auch auf Parallelrechnern einsetzen zu können, werden die bei der Diskretisierung des Problemgebietes entstehenden Gitter in einzelne Blöcke zerlegt [Schreck92]. Jeder Block wird einem der verfügbaren Prozessoren zugewiesen und das Teilproblem für jeden Block von den eingesetzten Prozessoren parallel gelöst. Die Kopplung der Blöcke wird durch einen Datentransfer zwischen den beteiligten Prozessoren erreicht. Beim Einsatz von Architekturen mit verteiltem Speicher (Message Passing Architekturen) erfolgt dieser Datentransfer durch explizite Sende- und Empfangsoperationen, bei denen die Daten aus dem Speicher des Senders in den des Empfängers kopiert werden. Im Fall von Architekturen mit gemeinsamem Speicher kann jeder Prozessor auf die Daten des anderen zugreifen. Jedoch muß beim wechselseitigen Zugriff auf die Daten durch Ausschlußmechanismen die Datenkonsistenz gewährleistet werden. Sende- und Empfangsoperationen, wie auch Operationen zum gegenseitigen Ausschluß benötigen ein gewisses Maß an Zeit, das nicht für numerische Berechnungen genutzt werden kann. Um eine maximale parallele Effizienz zu erzielen, muß diese Zeit auf ein Mindestmaß reduziert werden. Dazu muß einerseits das Datenvolumen, das im Laufe der Berechnung ausgetauscht wird, minimal gehalten werden, um bei vorgegebener Übertragungsbandbreite in kurzer Zeit die Daten zu transportieren, andererseits muß die Anzahl der einzelnen Kommunikationsvorgänge minimiert werden, um möglichst wenig Zeit für das Aufsetzen von Botschaften bzw. die Synchronisation zu verbrauchen. Die erzielte parallele Effizienz hängt also einerseits von der Leistungsfähigkeit der Hardware, aber entscheidend auch von der Qualität des eingesetzten Partitionierungsschemas ab, das die Gitter in Blökke zerlegt.

Zwei Techniken zur Partitionierung habe sich bewährt:

- Partitionierung mit alternierenden Schnitten
- Heuristische Ansätze

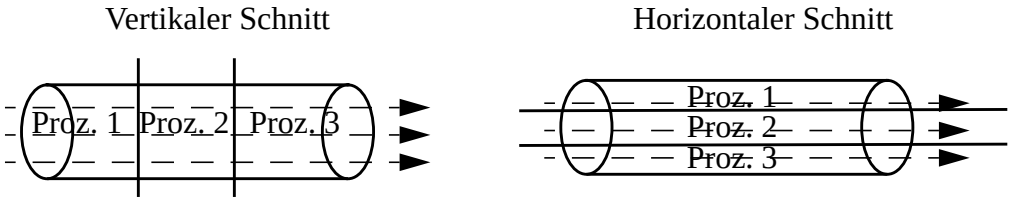
Bei der Partitionierung mit alternierenden Schnitten wird das Problemgebiet durch vertikale und horizontale Schnitte solange geteilt, bis genauso viele Partitionen wie Prozessoren entstanden sind. Dies soll an einem Beispiel mit 12 Prozessoren deutlich werden:



Das Problemgebiet wird durch abwechselnde horizontale und vertikale Schnitte partitioniert. Die Anzahl der Streifen, in die bei jedem Schnitt geteilt wird, kann durch eine Primfaktorzerlegung der Prozessorzahl ermittelt werden. Im Fall von 12 Prozessoren bedeutet das eine Partitionierung in 2x2x3 Streifen. In jedem Partitionierungsschritt kann die Aufteilung so erfolgen, daß alle beteiligten Prozessoren die gleiche Zeit für die Berechnung eines Blocks einschließliche der Datentransfers benötigen. In jedem Partitionierungsschritt wird also ein lokales Minimum gefunden. Die Partitionierung des gesamten Problemgebietes kann mehr oder weniger weit von der optimalen Lösung entfernt sein. Auch unterscheiden sich die Partitionierungen hinsichtlich des zu transportierenden Datenvolumens und der Anzahl der Datentransfers, je nachdem, ob man mit einem vertikalen oder horizontalen Schnitt beginnt (siehe 1. und 2. Partitionierungsansatz).

Weiterhin gibt es eine Reihe von heuristischen Verfahren (siehe [Birken91]), die auch bei komplexen Problemgebieten brauchbare Lösungen liefern und dem Optimum je nach Strategie nahe kommen. Am Lehrstuhl für Strömungsmechanik wurden Partitionierungsstrategien für einige beispielhafte Geometrien untersucht[Schäfer94]. Dabei zeigt sich, daß je nach Geometrie die eine oder andere Strategie ansprechende Ergebnisse liefert. Eine allgemeingültige Strategie konnte jedoch leider nicht gefunden werden, so daß hier immer noch das Expertenwissen eines Strömungsmechanikers gefragt ist. Algorithmen zur Berechnung der optimalen Lösung sind hier leider nicht einsetzbar, da es sich um ein Problem mit exponentieller Komplexität handelt. Im Extremfall könnte die Berechnung der optimalen Lösung des Partitionierungsproblems aufwendiger sein als die eigentliche strömungsmechanische Berechnung, die nur einen linearen bis quadratischen Aufwand mit sich bringt.

Problematisch bei all diesen apriori Partitionierungen ist die mangelnde Flexibilität bei komplexen Geometrien und Strömungen.



Schon bei einer einfachen Rohrsströmung versagen statische Partitionierungsstrategien, da diese nur die geometrische Information nutzen können, nicht aber das Wissen über die sich dynamisch ändernden strömungsmechanischen Gegebenheiten. Im obigen Beispiel wird das durchströmte Rohr in 3 Partitionen auf die vorhanden 3 Prozessoren geteilt. Beim vertikalen Schnitt tragen die Prozessoren 2 und 3 bei den ersten Iterationen nicht zur Problemlösung bei. Prozessor 2 kann das ihm zugeteilte Problemgebiet erst dann berechnen, wenn Prozessor 1 mit seiner Berechnung fertig ist. Danach kann erst Prozessor 3 effektiv zur Lösung beitragen. Die parallele Berechnung wird also nicht wesentlich schneller erfolgen als im sequentiellen Fall.
Beim horizontalen Schnitt sind zwar alle 3 Prozessoren an der Berechnung einer konvergenten Lösung beteiligt, sie führen jedoch bei den ersten Iterationen unnütze arithmetische Operationen im Bereich des Rohrausflusses durch. Auch hier wird kein merklicher Rechenzeitgewinn durch den Einsatz eines Parallelrechners zu verzeichnen sein. Das Problem der Rohrströmung ist wohl der “worst case” für eine statische Partititonierungsstrategie. Es zeigt aber die Grenzen dieses Parallelisierungsansatzes bei allen Arten von konvektiven Problemen auf.

2 Möglichkeiten adaptiver Verfahren

Ein numerisches Lösungsverfahren ist dann am effizientesten, wenn jede eingesetzte arithmetische Operation einen maximalen Schritt auf dem Weg zur Lösung des Problems bedeutet. In diesem Fall kommt man mit einer minimalen Anzahl an Rechenoperationen aus. Voraussetzung dafür ist eine an das Problem angepaßte Lösungsstrategie. Diese Adaptivität läßt sich auf mehreren Ebenen wiederfinden.

- Adaptivität der Diskretisierung:
  Die Feinheit des Gitters wird entsprechend den Anforderungen des Problems dynamisch angepaßt. In Bereichen mit unstetigem Werteverlauf wird die Feinheit der Diskretisierung erhöht, während in Bereichen, an denen sich der Werteverlauf kaum ändert, ein grobes Gitter gewählt werden kann. Da sich die Diskretisierung des Gesamtgebietes nicht mehr an den Bereichen mit der höchsten Auflösung orientieren muß, vermindert sich sowohl der Speicherbedarf als auch der Rechenaufwand entscheidend.
- Adaptivität des Lösungsverfahrens:
  Arithmetische Operationen sollten nur an den Stellen des Problembereiches durchgeführt werden, an denen sie zur Lösung des Problems beitragen. Dies setzt einen gewissen organisatorischen Aufwand voraus, der einiges an Rechenzeit benötigt. Dieser “Zeitverlust” wird aber durch die wesentlich verbesserte numerische Effizienz mehr als nur ausgegli-

chen. Untersuchungen am Institut für Informatik der TU München [Rüde92] haben gezeigt, daß durch eine optimale Problemadaptivität eine minimale Anzahl an arithmetischen Operationen benötigt wird.

– Adaptivität bei der Last- und Datenverteilung:
  Eine gleichmäßige Verteilung der Rechenlast auf die verfügbaren Prozessoren ist eine Grundvoraussetzung um eine hohe parallele Effizienz zu erzielen. Daneben ist die ausreichende Ver- und Entsorgung der Prozessoren mit den benötigten Daten wesentlich. Dieses Problem der Datenverteilung ist jedoch an das Lastverteilungsproblem gekoppelt, da durch die Zuweisung eines Rechenauftrags der Prozessor mit den dazugehörenden Daten versorgt werden muß.

Bei den im 1. Abschnitt vorgestellten Parallelisierungsansätzen wird das Problemgebiet unter Zuhilfenahme von Gewichtungsfunktionen für den Rechen- und Kommunikationsaufwand statisch partitioniert. Damit ergibt sich dann auch die statische Verteilung der Rechenlast auf die Prozessoren.
Bei den adaptiven Verfahren wird dynamisch ermittelt, welche Gitterpunkte zur Berechnung anstehen. Die dadurch ermittelte Rechenlast muß dann auf die Prozessoren verteilt werden. Dies impliziert dann die Ver- und Entsorgung der Prozessoren mit den benötigten Daten. Das Vorgehen ist also konträr zu den statischen Partitionierungsverfahren.
Bei Rechnerarchitekturen mit verteiltem Speicher führt diese dynamische Last- und Datenverteilung zu extrem aufwendigen Verteilungsalgorithmen [Bastian93][Birken94], da in den Programmcode für die numerischen Berechnungen verteilte Koordinierungsmechanismen für die Lastverteilung und für das Versenden und Empfangen von Daten eingefügt werden müssen. Der Aufwand kann als dermaßen extrem angesehen werden, daß er außer durch akademisches Interesse nicht zu rechtfertigen ist.

Einen Ausweg aus diesem Dilemma stellen Parallelrechnerarchitekturen mit gemeinsamem Speiher dar, bei denen alle Prozessoren mit der gleichen Zugriffszeit auf den Speicher zugreifen können (UMA Architekturen wie z.B. SGI Power Challenge) oder bei denen der Zugriff auf den gemeinsamen Speicher je nach Lokalität des Speicherbereiches unterschiedlich lang dauern kann (NUMA Architekturen wie z.B. KSR2 oder Convex SPP).
Bei diesen Architekturen entfällt das Problem, geeignete verteilte Mechanismen für das Versenden und Empfangen der Botschaften zu finden, da diese Aufgabe durch die Hardware effizient und ohne Programmieraufwand bewältigt wird. Da hierzu extrem schnelle Hardwarekomponenten benötigt werden, ist diese Rechnerklasse zur Zeit nur an großen Rechenzentren vertreten. Die fortschreitende Entwicklung wird diesen Rechnertyp aber in naher Zukunft auch für kleine und mittlere Unternehmen erschwinglich machen, so daß für dieses zukunftsträchtige Rechnerkonzept schon bald leistungsfähige, portable und beherrschbare Software zur Verfügung stehen sollte.

3 Die Active Threads Strategie für komplexe Geometrien

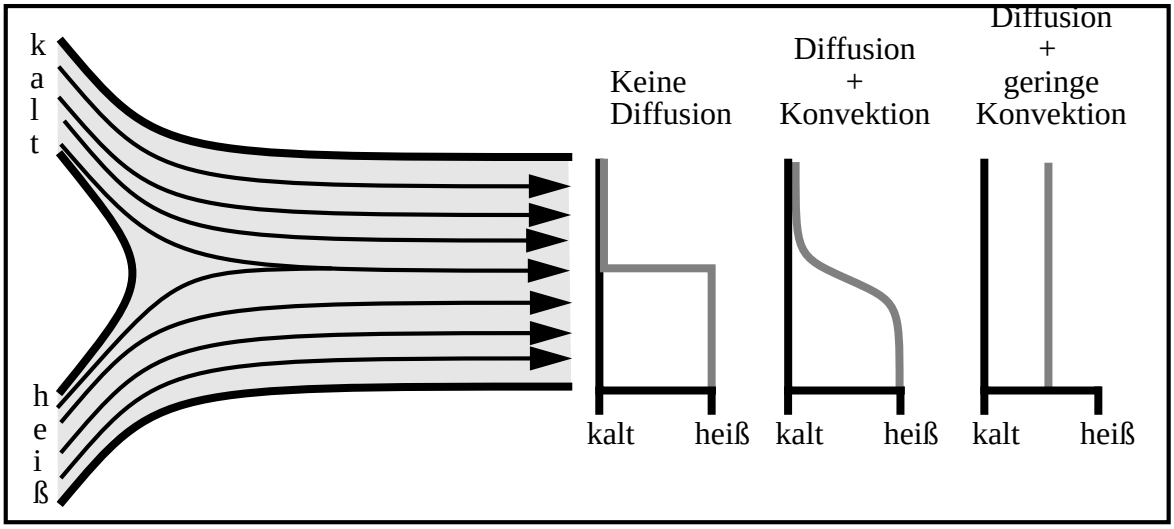
Am Lehrstuhl für Betriebssysteme (Informatik IV) der Uni Erlangen wurde ein neuartiges Konzept zur einfachen Implementierung adaptiver Verfahren auf komplexen Geometrien entwikkelt. Es basiert auf der am Institut für Informatik V der TU München entwickelten **Active Set Strategie** [Rüde92].

3.1 Die Active Set Strategie

Hierbei werden aus der Menge aller Gitterpunkte durch ein Auswahlverfahren einige Punkte bestimmt, aus denen beim Start des Verfahrens die Menge der aktiven Punkte gewonnen wird. Aus dieser Menge wird ein Punkt entfernt. Falls der lokale Fehler für diesen Punkt einen gewissen Schwellwert überschreitet, wird der Wert dieses Punktes an Hand der Werte seiner räumlichen Nachbarn neu berechnet. In diesem Fall werden auch alle Nachbarpunkte in die Menge der aktiven Punkte übernommen. Dieser Vorgang wiederholt sich so lange, bis es keinen aktiven Punkt mehr gibt.

```
func ActiveSetStrategy(grid,startpoints,threshold)
     aktiveset = startpoints;
     while(aktiveset ≠ Ø)
        select(point ∈ aktiveset ∈ grid)
        activeset = activeset \ point;
        if (error(point) > threshold)
            point = calculatepoint(point)
            aktiveset = aktiveset ∪ neighborpoints(point)
        end if
     end while
end func
```

Die Wirkungsweise dieses Verfahrens wird an einem Beispiel aus der Strömungsmechanik deutlich.



Über zwei Anschlüsse fließt eine kalte und eine heiße Flüssigkeit in ein Rohr. Gesucht ist die Wärmeverteilung am Ausfluß des Rohres in Abhängigkeit von der Strömungsgeschwindigkeit und dem Diffusionskoeffizienten. Wählt man als Startpunkte die Punkte im Anschlußbereich, so wird sich die Menge der aktiven Punkte im Laufe der Berechnung entlang der Strömungslinien bis zum Ausfluß verschieben. Je stärker die Diffusion relativ zur Strömung (Konvektion) ist, desto mehr wird sich die Menge der aktiven Punkte aufweiten. Bei geringer Diffusion wird die Menge hingegen kaum mehr Punkte enthalten als die Startmenge.
Wesentlich an diesem Verfahren ist die Tatsache, daß sich die Bearbeitungsreihenfolge dem Problem anpaßt, vorausgesetzt eine geeignete Startmenge wird gefunden. Diese läßt sich bei N Gitterpunkten in O(N) finden, indem man zunächst für alle Gitterpunkte den lokalen Fehler bestimmt und alle Punkte in die Startmenge aufnimmt, die oberhalb des Schwellwertes liegen.

3.2 Repräsentation der aktiven Punkte durch aktive Threads

Der in dieser Arbeit vorgestellte Ansatz beruht auf dem Einsatz extrem leichtgewichtiger Prozesse (Threads) in Verbindung mit einer auf die Gitterpunkte verteilten Aktivitätsinformation. Hierbei wird für jeden Gitterpunkt ein leichtgewichtiger Prozeß (Thread) erzeugt, der zunächst einmal suspendiert wird. Eine Startmenge von Threads wird aktiviert. Jeder Thread führt den folgenden Zyklus aus:

```
func working_thread(myself, threshold)
    while (1)
        if (error(myself) > threshold)
         localpoint = calculatepoint(neighborpoints)
         activate(neighbors)
        end if
        suspend(myself)
    end while
```

Jeder aktive Prozeß berechnet den Wert des ihm zugewiesenen Gitterpunkts aus den Werten seiner Nachbarpunkte und aktiviert anschließend seine Nachbarprozesse, bevor er sich selbst suspendiert. Das Verfahren terminiert, wenn die Menge der aktiven Prozesse leer ist, d.h. alle Prozesse sich selbst suspendiert und keine anderen Prozesse mehr aktiviert haben.
Jeder aktive Prozeß kann auch neue Gitterpunkte und damit neue Threads erzeugen. Er muß jedoch zuvor exklusiven Zugriff auf die Datenstruktur aller Nachbarn der neu generierten Gitterpunkte haben, um dort die neuen Threads in die Liste der zu aktivierenden Threads eintragen zu können.

3.3 Möglichkeiten des neuen Verfahrens

Das Programmiermodell der aktiven leichtgewichtigen Prozesse bringt gegenüber den bisherigen Parallelisierungsansätzen eine Reihe von wesentlichen Vorteilen mit sich:

- Das Programmiermodell eignet sich zur einfachen Beschreibung adaptiver numerischer Verfahren

- Der Wissenschaftler muß sich nur um die Rechenroutinen und Aktivierungsregeln kümmern.
- Die Lastverteilung wird in ein Laufzeit-/Betriebssystem verlagert, das optimal an die zugrunde liegende Rechnerarchitektur angepaßt ist.
- Der Programmcode wird portabel. Er enthält keine rechnerabhängigen Parallelisierungskonzepte.

Das in diesem Abschnitt vorgestellte Konzept zur Implementierung adaptiver numerischer Verfahren mit leichtgewichtigen Prozessen stellt jedoch hohe Anforderungen an das verwendete Laufzeit-/Betriebssystem, das für eine effiziente Verwaltung der Threads sorgen muß.

4 Ziele

Will man hocheffiziente numerische Verfahren auf komplexen Geometrien auf modernen Parallelrechnerarchitekturen implementieren, müssen neue Ansätze zur Beschreibung des Berechnungsablaufes gefunden werden. Eine vielversprechende Möglichkeit stellt die in Zusammenarbeit zwischen dem Institut für Informatik der TU München (Prof. Zenger, Dr. Rüde) und dem Lehrstuhl für Betriebssysteme der Uni Erlangen (Prof. Hofmann, F. Bellosa) entwickelte Active Threads Strategie dar. Dies setzt effiziente Mechanismen zur Verwaltung von leichtgewichtigen Prozessen voraus. Die Abstraktionsmechanismen von Betriebssystemen reichen jedoch nicht immer aus, um allen Ansprüchen dieses Programmiermodells zu genügen[Ande 89][Ande 92]. Die von den Herstellern von Parallelrechnern angebotenen Betriebssystemmechanismen zielen auf den Markt der Datenbanksysteme und auf den Kundenkreis der langjährigen Vektorrechnerbenutzer, da hier kurzfristig die Gewinnaussichten am größten sind. Neue und noch nicht normierte Mechanismen zur einfachen, effizienten Implementierung adaptiver Verfahren sind daher im kommerziellen Bereich in naher Zukunft nicht zu erwarten. Um diesen zukunftsträchtigen Bereich nicht völlig zu vernachlässigen, wird in Kooperation mit der Firma Convex am Lehrstuhl für Betriebssysteme ein Laufzeitsystem mit Betriebssystemunterstützung entwickelt, das an die speziellen Erfordernisse angepaßt ist. Wesentliche Argumente für den Einsatz eines Laufzeitsystems stellen der Wunsch nach einer effizienten Prozeßverwaltung auf Benutzerebene und nach Synchronisations- und Kommunikationskonstrukten dar, welche an das Programmiermodell angepaßt sind[McCa 93]. Besonders bei NUMA Architekturen ist ein Interface zwischen der Threadverwaltung und dem eigentlichen Anwendungsprogramm wesentlich, um Berechnung und Speichertransfer überlappen zu lassen[LeBl 89]. All dies wird durch das Konzept eines auf adaptive Verfahren zugeschnittenen Programmiermodells mit einer maßgeschneiderten Prozeß- und Speicherverwaltung auf Benutzerebene ermöglicht. Diese Verwaltungsmechanismen müssen jedoch in das Konzept des Betriebssystems eingebunden werden, um einen Einbruch der Parallelität zu verhindern, der durch Threads bedingt wird, die sich im Kern blokkieren (z.B. durch Ein/Ausgabe oder durch Auslagerung von Speicherbereichen). Zudem sollte sich die Prozeßverwaltung des Betriebssystems mit der Verwaltung auf Benutzerebene im Hinblick auf einen optimalen Ablauf der Anwendung koordinieren, um negative Interferenzen zwischen diesen beiden Verwaltungsmechanismen zu vermeiden. Um dieses Probleme zu lösen, sind Erweiterungen des Betriebssystems erforderlich [Ghos 93].

Am Lehrstuhl für Betriebssysteme wird daher im Rahmen des bayerischen Forschungsverbundes für technisch wissenschaftliches Hochleistungsrechnen ein Laufzeitsystem auf Benutzerebene entwickelt, das ein anwendungsspezifisches Interface bietet. Die numerische Anwendung soll dabei in der Lage sein, Informationen über Scheduling-Entscheidungen des Laufzeitsystems zu erlangen, um Speicherbereiche für künftige Berechnungen bereits im Hintergrund in den lokalen Speicher zu transferieren noch bevor diese wirklich benötigt werden. Dadurch läßt sich die Prozessorauslastung beträchtlich steigern, da die Recheneinheit nicht mehr auf benötigte Daten warten muß, sondern stetig mit neuen Daten aus einem schnellen Zwischenspeicher (Cache) versorgt werden kann. Die notwendige Kooperation zwischen dem Laufzeitsystem auf Benutzerebene und dem Betriebssystem wird durch Erweiterungen eines bestehenden Betriebsystemkerns in Zusammenarbeit mit der Firma Convex auf Basis der Architektur des Convex SPP möglich. Erste erfolgversprechende Ansätze und Forschunsergebnisse sind erzielt [Bellosa94][Koppe 94][Reder 94].

Damit wird es dem Anwender ermöglicht, einfach und effizient adaptive numerische Verfahren auf Hochleistungsrechnern der NUMA Klasse zu implementieren.

Literaturverzeichnis

[Ande 89] T. Anderson, E. Lazowska, H. Levy, “The Performance Implication of Thread Management Alternatives for Shared-Memory Multiprocessors”, ACM Trans. on Comp. Vol. 38 No. 12, Dec. 1989

[Ande 92] T. Anderson, B. Bershad, E. Lazowska, H. Levy, “Scheduler Activations: Effective Kernel Support for the User-Level Management of Parallelism”, ACM Trans. on Comp. Sys. Vol. 10, Feb. 1992

[Bastian93] P. Bastion, “Parallelisierung adaptiver Mehrgitterverfahren”, Dissertation, Uni Heidelberg, 1993

[Bellosa94] F. Bellosa, “Implementierung adaptiver numerischer Verfahren mit leichtgewichtigen Prozessen”, Interner Bericht am IMMD IV 2/94, Uni Erlangen, 1994

[Birken91] K. Birken, “Entwicklung eines Algorithmus zur Lastverteilung und Kommunikationsminimierung bei gitterorientierten Berechnungen auf Multiprozessoren”, Studienarbeitarbeit IMMD III, Erlangen, 1991

[Birken94] K. Birken, “An Efficient Programming Model for Parallel Adaptive CFD-Algorithms”, Proc. of Parallel Comp. Fluid Dynamics, Kyoto, 5/1994

[Ghos 93] K. Ghosh, “Experimentation with Configurable, Lightweight Threads on a KSR Multiprocessor”, Georgia Institue of Technology: Technical report GIT-CC-93/37

[Koppe 94] C. Koppe, “Sleeping Threads: Ein Kernmechanismus zur Unterstützung effizienter User-Level-Threads”, Interner Bericht IMMD IV 14-94

[LeBl 89] T. LeBlanc, “Memory management for large-scale numa multiprocessors”, Department of Computer Science: Technical report*311

[McCa 93] C. McCann, R. Vaswani, J. Zahorjan, “A Dynamic Processor Allocation Policy for Multiprogrammed Shared-Memory Multiprocessors”, ACM Trans on Comp. Sys. Vol. 11 No. 2, May 1993

[Reder 94] U. Reder, “Implementierung eines effizienten Prozessumschalters auf Benutzerebene”, Studienarbeit am IMMD IV, Uni Erlangen, 1994

[Rüde92] Ulrich Rüde, “On the multilevel adaptive iterativ Method”, SIAM journal on scientific and statistical computing, Vol. 15, 1994

[Schreck92] E. Schreck, “Numerical Simulation of Complex Fluid Flows on MIMD Computers”, Technical Report LSTM, Universität Erlangen-Nürnberg, 1992

[Schäfer94] M. Schäfer, “An eficient parallel solution technique for the incompressible Navier-Stokes equations”, Technical Report LSTM, Universität Erlangen-Nürnberg, 1994