Kenngrößenermittlung für parallele rekursive Programme

Stefan U. Hänßgen*
IPD, Fakultät für Informatik, Universität Karlsruhe
haenssgen@ira.uka.de

Oktober 1995

Zusammenfassung

Viele mathematische und naturwissenschaftliche Probleme lassen sich in natürlicher Weise als rekursive Programme ausdrücken. Diese effizient parallel auszuführen überfordert die bestehenden parallelen Programmiersysteme, weil der Ablauf hochgradig dynamisch und irregulär ist.

Wir zeigen einen Ansatz zur effizienten Lösung solcher Probleme auf. Dabei werden zusätzlich zu statischen Analysen des Übersetzers auch zur Laufzeit Kenngrößen des Programms gemessen, die im nächsten Übersetzungslauf weitergehende Optimierungen zulassen und dem Laufzeitsystem Hinweise zur Lastverteilung geben.

1 Einführung

Für viele Probleme der parallelen Programmierung gibt es inzwischen Systeme, die dem Benutzer die Arbeit wesentlich erleichtern. Maschinenunabhängige explizit datenparallele Programme sind bereits Realität [Phi94], und es gibt erfolgversprechende Ansätze zur automatischen Parallelisierung. Allerdings scheitern all diese Systeme, wenn das Programm ein irreguläres dynamisches Verhalten zeigt; die Behandlung von Rekursion im Parallelen ist ebenfalls nur mit großen Leistungseinbußen möglich.

Das wäre keine große Einschränkung, wenn rekursiv dynamische Probleme lediglich in Spielzeugbeispielen vorkämen. Nicht wenige reale Anwendungen werden jedoch als sich rekursiv verzweigende Programme ausgedrückt, bei denen sich eine Parallelisierung der Äste des Aufrufbaums anbietet. Beispiele solcher Probleme sind Programme mit rekursiv definierten, dynamischen Datenstrukturen (z.B. Bäume), Teile-und-Herrsche-Algorithmen, sowie n-Körper-Probleme, die etwa bei Molekülgeometrie- und Galaxiensimulationen auftreten.

Wir entwickeln zur Zeit ein System, bei dem der Benutzer ein rekursives Programm schreibt und der Übersetzer in Zusammenarbeit mit dem Laufzeitsystem wichtige Kenngrößen des Programms automatisch ermittelt. Diese ermöglichen dann beim nächsten Übersetzungsvorgang und während des Programmablaufs Optimierungen, die rein statischen Ansätzen überlegen sind.

Dieser Beitrag konzentriert sich auf den Aspekt der Messung solcher Kenngrößen. Wir stellen zunächst das Gesamtsystem vor und beschreiben in Abschnitt 3 die Infrastruktur und den Einsatz der Instrumentierungen zur Ermittlung der Kenngrößen. Danach stellen wir in Abschnitt 4 die Problembeispiele vor und zeigen die Ergebnisse der Messungen. Abschnitt 7 erläutert Bezüge zu anderen Arbeiten, gefolgt von der Diskussion der Ergebnisse und dem Ausblick.

2 Systemaufbau

Der Aufbau des Gesamtsystems ist in Abbildung 1 gezeigt:

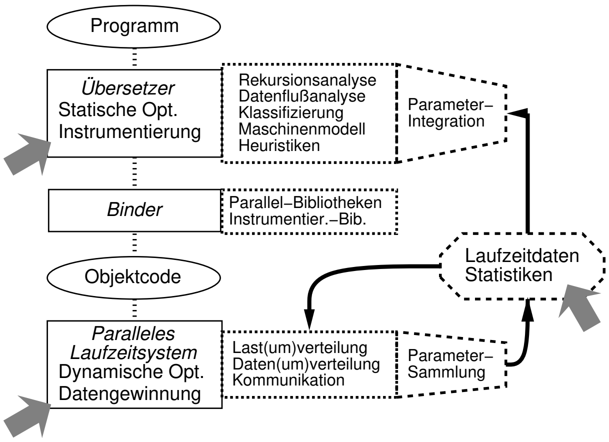

Abbildung 1: Systemübersicht — Zusammenspiel der Komponenten und Schwerpunkte des Papiers.

Der Übersetzer führt Analysen des Programms durch, insbesondere bezüglich des rekursiven Aufbaus, plaziert geeignete Instrumentierungen im Code und optimiert das Programm aufgrund der

*Diese Arbeit wurde teilweise unterstützt durch das Graduiertenkolleg „Beherrschbarkeit komplexer Systeme“ (DFG Vo 287/5-2)

statisch ermittelbaren Information. Der entstandene Code wird mit Bibliotheken für die parallele Ausführung und Datensammlung gebunden und dann unter einem Laufzeitsystem parallel abgearbeitet. Dabei werden durch die Instrumentierung Daten zum rekursiven Verhalten des Programms gesammelt, die sowohl dem Übersetzer für weitergehende Optimierungen beim nächsten Übersetzungsvorgang zur Verfügung stehen, als auch dem Laufzeitsystem dynamische Entscheidungen zur Optimierung des Programmablaufs ermöglichen.

Dieser Beitrag behandelt die in der Systemübersicht mit Pfeilen hervorgehobenen Komponenten Instrumentierung, Datengewinnung und Ermittlung von Statistiken.

3 Instrumentierung

Um das Verhalten eines rekursiven Programms zu messen, haben wir eine Bibliothek entwickelt, die die Datensammlung zur Kenngrößenbestimmung unterstützt. Das Programm wird mit Aufrufen an diese Bibliothek sowie einigen zusätzlichen Variablen instrumentiert[1]. Zusätzlich verfügt das System über eine Datenbasis mit maschinenspezifischen Kostenfunktionen wie Prozessorgeschwindigkeit und Kommunikationsleistung, um Optimierungsentscheidungen treffen zu können.

Zunächst führen wir im folgenden die Kenngrößen ein, dann erläutern wir das Vorgehen bei der Instrumentierung und zeigen den Einsatz an einem konkreten Beispiel.

3.1 Kenngrößen

Wir ermitteln bei unseren Messungen dynamische Kenngrößen der Programme, von denen wir hier die drei wichtigsten betrachten wollen. Alle Kenngrößen sind statisch nur sehr schwer oder überhaupt nicht zu ermitteln, da sie auch von den Eingabedaten abhängen. Ergebnis der Messungen ist jeweils eine komplette Verteilung der Parameterwerte für den betrachteten Programmablauf.

- **Rekursionstiefe** – beschreibt, wie tief ein rekursiver Aufruf im Programm geht. Sie wird gemessen, indem jeder rekursiven Prozedur ein zusätzlicher Parameter (die Aufruftiefe) mitgegeben, bei jedem rekursiven Aufruf erhöht und am Rekursionsende abgespeichert wird.

- **Verzweigungsgrad** – gibt an, wie oft das Problem beim rekursiven Aufruf unterteilt wird. Wir messen den Verzweigungsgrad, indem wir die Zahl der Aufrufe pro Rekursionsschritt zählen und zusammen mit der aktuellen Rekursionstiefe ablegen. In Sonderfällen läßt sich dieser Parameter auch statisch ermitteln, z.B. wenn bedingungslos immer zwei oder null Aufrufe erfolgen.

- **Rechenaufwand pro Rekursionsschritt** – wird durch Zähler in jedem Block des Programms gemessen und ermöglicht zusammen mit einer Liste der Kosten jedes Befehls eine Abschätzung der Laufzeit des betreffenden Abschnitts.

Außer diesen gibt es noch weitere Kenngrößen wie z.B. Kommunikationspfade (entlang der Aufrufhierarchie, in einer Ebene oder beliebig) und Speicheraufwand eines Schrittes, auf die wir hier nicht näher eingehen.

Natürlich sind die vorgestellten Kenngrößen zumeist abhängig von den Eingabedaten. Falls diese Abhängigkeit zu komplex ist, können die Optimierungsmethoden nicht viel erreichen, aber unsere Erfahrung zeigt, daß sich auch bei zufälligen Eingabedaten oft eindeutige Verteilungen herausstellen, die der Übersetzer auswerten kann. Meistens besteht eine Abhängigkeit der Verteilung von der Anzahl n der Eingabedaten, die sich z.B. durch Regressionsanalyse berechnen läßt[2].

Jeder Probelauf des Programms erzeugt bei der Instrumentierung einen neuen Datensatz, so daß der Übersetzer signifikante Entwicklungen beobachten kann und das Laufzeitsystem aufgrund der Statistiken genauere Entscheidungen dynamisch fällen kann. Unsere Messungen zeigen, daß oft schon ein einziger Testlauf signifikante Aussagen über das Verhalten nachfolgender Berechnungen zuläßt (siehe Abbildung 5ff).

3.2 Unterstützende Funktionen

Zur Instrumentierung wurde eine Bibliothek erstellt, die die folgenden Funktionen zur Sammlung und Verwaltung der Statistiken zur Verfügung stellt:

3.2.1 Infrastruktur und Verwaltung

Hier werden die Statistiken initialisiert und statische Daten wie Prozedurname, Zeilennummer, Konstrukt-Kosten usw. definiert. Die Ausgabe der gesammelten Daten findet ebenfalls hier statt.

[1] Dies geschieht im Prototyp von Hand, soll aber später automatisch durch den Übersetzer vorgenommen werden.

[2] Die Größe der Eingabedaten läßt sich frühzeitig dynamisch im Programm bestimmen oder notfalls durch Benutzerannotationen angeben.

3.2.2 Rekursion

Statistiken zum rekursiven Ablauf werden mit den folgenden Funktionen[3] gesammelt:

`AddDepthStat(`*Tiefe*`)` – wird am Rekursionsende aufgerufen, d.h. wenn in einer rekursiven Prozedur kein weiterer rekursiver Abstieg erfolgt, und zeichnet die erreichte Tiefe auf – der Rekursionsgrad ist natürlich gleich Null.

`AddDegreeStat(`*Tiefe, Grad*`)` – wird nach jedem sich rekursiv verzweigendem Aufruf benutzt, um den Rekursionsgrad und die aktuelle Rekursionstiefe aufzuzeichnen.

Sowohl Tiefe als auch Grad werden in dynamischen Feldern gehalten, die bei Bedarf zur Laufzeit erweitert werden.

3.2.3 Profiling

Die folgende Funktion dient zur Bestimmung der Zahl der Durchläufe durch ein Konstrukt:

`UpdateProfileEntry(`*Zähler*`)` – wird nach dem Ende des entsprechenden Konstrukts aufgerufen und summiert die diesmalige Zahl der Durchläufe auf die interne Gesamtzahl für das Konstrukt auf.

Dadurch, daß die Durchlaufzahl in Hilfsvariablen gehalten und erst möglichst spät nach Verlassen des Konstrukts registriert wird, bleibt der Mehraufwand gerade in inneren Schleifen zur Laufzeit gering.

3.2.4 Hilfsvariablen

Für jede relevante Prozedur werden zusätzliche Variablen eingeführt, um die Zahl der Durchläufe durch alle auftretenden Konstrukte wie `WHILE` oder `IF-THEN-ELSE` zu messen. Außerdem erhalten alle rekursiven Prozeduren und ihre entsprechenden Aufrufe einen zusätzlichen Parameter, nämlich die Tiefe des Aufrufs.

3.3 Instrumentierungsregeln

Für alle Instrumentierungsfunktionen gibt es Regeln, nach denen sie im Programm eingefügt werden, ebenso für die Definition und Plazierung der Hilfsvariablen. Wir zeigen im folgenden exemplarisch die Regeln zur Ermittlung der Profiling-Information als informellen Klartext, da eine komplette Auflistung den Umfang dieses Beitrags sprengen würde:

- Am Anfang des Hauptprogramms für alle zu messenden Konstrukte einen Aufruf einfügen: `AddProfileEntry("`*Prozedurname*`", ` *Zeile*`, ` *KonstruktKosten*`,"`*KonstruktSchachtelungsName*`");` mit

  "*Prozedurname*": Name der Prozedur, in der das Konstrukt textuell steht

  Zeile: Zeile im uninstrumentierten Programm, in der das Konstrukt beginnt

  KostruktKosten: Statische Kostenabschätzung für das Konstrukt

  "*KonstruktSchachtelungsName*": Textuelle Darstellung für die statische Schachtelung des Konstrukts, z.B. `"REPEAT_THEN"` für den `THEN`-Teil eines `IF`s innerhalb einer `REPEAT`-Schleife

- In der entsprechenden Prozedur für jedes Konstrukt eine neue lokale Zähler-Variable '*Prozedurname_KonstruktSchachtelungsName_Zeile*' im Deklarationsteil einführen, Typ CARDINAL, Initialwert 0.
- Ganz am Anfang des Konstrukts diesen Zähler mit `INC(..)` erhöhen, damit alle Durchläufe durch das Konstrukt während des aktuellen Prozeduraufrufs gemessen werden.
- Möglichst spät in der Prozedur die Zählerstände aller Konstrukte zu den bisher gesammelten Ständen dazusummieren per

  `UpdateProfileEntry(`*Zeile*`, ` *Zählervarible*`);` mit

  Zeile: Zeile im uninstrumentierten Programm, in der das Konstrukt beginnt

  Zählervariable: automatisch eingefügter Zähler, s.o.

3.4 Beispiel

Das Beispiel in Abbildung 2 zeigt den vereinfachten Kern der Eigenwertberechnung in Pseudocode. Die Aufrufe an die Instrumentierung sind kursiv dargestellt. Man sieht auch exemplarisch vier Durchlaufzähler-Hilfsvariablen, die beim Ablauf erhöht und erst direkt vor Verlassen der Prozedur gesichert werden.

In der Profile-Bibliothek werden die ankommenden Daten in Hashtabellen gespeichert und bei Programmende gesammelt ausgegeben. Pro Prozedur werden nur wenige solche Daten erzeugt, so daß die Gesamtdatenmengen gering bleiben. Abbildung 3 zeigt beispielhaft, welche reale Ausgabe ein instrumentiertes Programm erzeugt.

Es ist die Zahl der Durchläufe durch die entsprechenden Konstrukte einschließlich der Prozedur

[3] Vereinfachte Schreibweise – real sind auch noch Angaben wie Modul und Zeile, die die Zuordnung der Statistiken zu Stellen im Programm eindeutig machen, nötig.

```
PROCEDURE Eigen(lWert,rWert,Anzahl,Tiefe) =
VAR Eigen_206 := 0;
    Eigen_REPEAT_214 := 0;
    Eigen_THEN_227 := 0;
    Eigen_ELSE_230 := 0;
BEGIN
  INC(Eigen_206);
  REPEAT
    INC(Eigen_REPEAT_214);
    Schrumpfe [lWert,rWert]
  UNTIL EW-Anzahl im Intervall geändert;
  IF rWert - lWert < ϵ THEN
    (* Rekursionsende *)
    INC(Eigen_THEN_227);
    AddDegreeStat(Tiefe,0);
    AddDepthStat(Tiefe);
  ELSE
    (* Verzweige rekursiv *)
    INC Eigen_ELSE_230);
    Teile [lWert,rWert] bei mWert;
    mAnzahl := EZ-Anzahl in [lWert,mWert]
    Eigen(mWert,rWert,Anzahl-mAnzahl,Tiefe+1);
    Eigen(lWert,mWert,mAnzahl,Tiefe+1);
    AddDegreeStat(Tiefe,2);
  END;
  AddProfileEntry(Eigen_206);
  AddProfileEntry(Eigen_REPEAT_214);
  AddProfileEntry(Eigen_THEN_227);
  AddProfileEntry(Eigen_ELSE_230);
END Eigen;
```

Abbildung 2: Beispiel für die Instrumentierung

```
Profile for Module 'Eigenwert' :

procedure            line       count
---------            ----       -----
Eigen -            [ 206] :     19998

        recursion depth: sum 64992, # 5000
         [ 0] :  0
         [ 1] :  0
         [ 2] :  0
         [ 3] :  0
         [ 4] :  0
         [ 5] :  2
         [ 6] :  3
         [ 7] :  3
         [ 8] :  19
         [ 9] :  11
         [10] :  45
         [11] :  235
         [12] :  1221
         [13] :  1913
         [14] :  1140
         [15] :  352
         [16] :  56
         [17] :  0

        recursion degree: sum 9998, # 9999
         [ 0] :  5000
         [ 1] :  0
         [ 2] :  4999
         [ 3] :  0

Eigen - REPEAT     [ 214] :    206954
Eigen - THEN       [ 227] :      5000
Eigen - ELSE       [ 230] :      4999
```

Abbildung 3: Beispielsausgabe des instrumentierten Eigenwertprogramms

selbst angegeben sowie für den rekursiven Aufruf die Verteilung der Aufrufe auf die entsprechenden Rekursionstiefen und -grade[4].

4 Die Probleme

Als Testvehikel für unsere Methoden verwenden wir hier die folgenden realen Anwendungen:

4.1 Eigenwertbestimmung

Für reelle symmetrische tridiagonale Matrizen, d.h. Matrizen, bei denen nur die Haupt- und die beiden symmetrischen Nebendiagonalen besetzt sind, läßt sich in Zeit $O(n)$ für ein gegebenes Intervall bestimmen, wieviele ihrer Eigenwerte darin liegen [CRY94]. Mit rekursiver Intervallteilung leitet man nun ein Teile-und-Herrsche Verfahren zur Eigenwertberechnung ab, das ein irreguläres Verhalten zeigt, weil der Aufwand zum Schrumpfen eines Intervalls sehr stark schwanken kann und die Eigenwerte meist ungleichmäßig verteilt sind.

4.2 Barnes-Hut

Das n-Körper-Problem, d.h. die Simulation eines Systems von n Körpern (z.B. Sternen) mit aufeinander wirkenden Kräften (z.B. Gravitation), wird oft mit dem hierarchischen Barnes-Hut Verfahren [Sin93] gelöst. Dabei wird das Verhalten einer Galaxie simuliert, indem der Raum rekursiv zerteilt wird, bis jedes Teil nur noch einen Körper enthält. Nun kann man für die Berechnung der Kräfte der Körper aufeinander weiter entfernte Ansammlungen von Körpern durch deren Massezentrum approximieren und muß dadurch nicht immer vollständig im Baum der Körper hinabsteigen. Der Aufwand reduziert sich dadurch von $O(n^2)$ auf $O(n \log n)$. Das Prinzip ist für den übersichtlicheren zweidimensionalen Fall in der folgenden Abbildung dargestellt:

[4] Intern wird auch zusätzlich beim Verzweigungsgrad die entsprechende Tiefe aufgezeichnet, aber in der momentanen Version nicht mit ausgegeben.

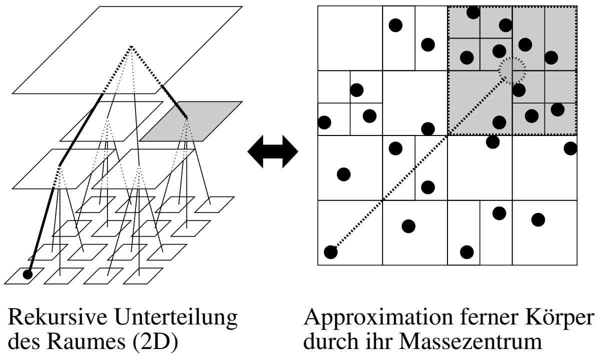

Abbildung 4: n-Körper-Problem, Barnes-Hut Verfahren

Durch die Bewegung der Körper ändert sich die Unterteilung des Raumes ständig, d.h. der entstehende Baum und seine Abarbeitung ist hochdynamisch und irregulär.

4.3 Engstes Paar

Für eine zweidimensionale Menge von Punkten gibt es einen Teile-und-Herrsche Algorithmus, der in $O(n \log n)$ das Punktepaar ermittelt, das zueinander den geringsten Abstand hat [Sed88]. Dieses Verfahren arbeitet unabhängig von den Eingabedaten, zeigt also kein irreguläres Verhalten. Es dient uns als Testfall, um Ergebnisse von irregulären Problemen mit solchen von rekursiven aber regulären Algorithmen zu vergleichen.

4.4 Abgrenzung

Wir betrachten nur Probleme, bei denen alle Berechnungen ins Ergebnis eingehen, d.h. Algorithmen wie Baumsuche, Branch-and-Bound usw., bei denen nach Finden des gesuchten Elements die Berechnung der anderen Teilbäume abgebrochen wird, liegen außerhalb unseres Gebiets. Für diese Probleme gibt es eigene spezielle Verfahren.

5 Ergebnisse

Es wurden die Eigenwertberechnung, der Barnes-Hut Algorithmus und das Engstes-Paar Programm instrumentiert und für verschiedene Eingabedaten und Problemgrößen durchgemessen. Bei der Eigenwertberechnung wurden zufällige Eingabedaten verwendet sowie solche, die in einer Gleichverteilung der Eigenwerte bzw. in einer pathologischen geometrischen Verteilung resultierten. Bei der Galaxiensimulation wurden reale Daten aus der Astrophysik [Ref95] sowie Gauß- und gleichverteilte Zufallsdaten benutzt. Eingaben der Engstes-Paar-Berechnung waren ebenfalls gleichverteilte Zufallsdaten. In keinem Falle widerlegten die Ergebnisse unsere These, daß eine automatische Auswertung der Meßwerte (und eine dementsprechende Optimierung) möglich und aussichtsreich ist.

Abbildungen 5 bis 9 zeigen einige interessante Verteilungen der Rekursionstiefen und Verzweigungsgrade – X-Achse ist die Tiefe bzw. der Grad, die Anzahl der Aufrufe in dieser Tiefe bzw. mit diesem Grad findet sich auf der Y-Achse, und die verschiedenen Kurven in einem Bild entsprechen den verschiedenen Problemgrößen, die man der Legende entnehmen kann.

In Bild 5 zur Eigenwertberechnung sieht man z.B. deutlich den Peak der Rekursionstiefe in logarithmischer Abhängigkeit von n und die gute Übereinstimmung der einzelnen Läufe untereinander. Selbst bei pathologischen Eingabedaten, nämlich einer geometrischen Verteilung der Eigenwerte, die einen einseitigen maximalen rekursiven Abstieg erzwingt, sind eindeutige Maxima in Abhängigkeit von der Problemgröße zu erkennen (Abbildung 6).

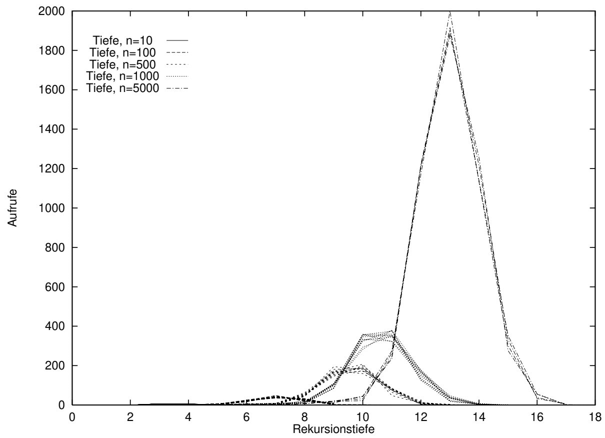

Abbildung 5: Verteilung der Rekursionstiefen bei der Eigenwertberechnung, gleichverteilte Zufallsdaten, je 5 Experimente pro Problemgröße

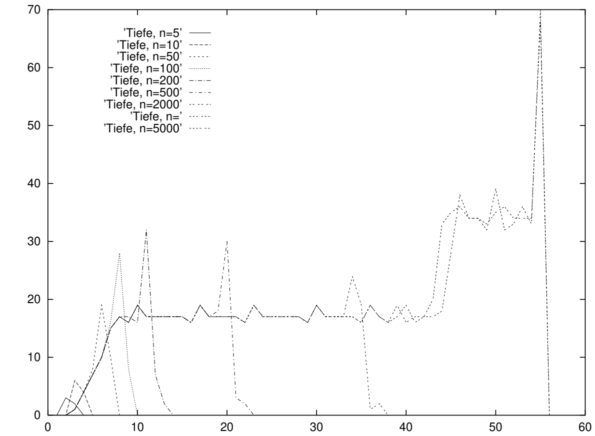

Abbildung 6: Verteilung der Rekursionstiefen bei der Eigenwertberechnung, pathologische Eingabedaten

Abbildungen 7 und 8 zeigen die Verteilung der

Kenngrößen bei der n-Körper-Simulation mit gaußverteilten Eingabedaten. Interessant ist der nicht aus dem Programmcode zu erwartende Peak der Verzweigungsgrade bei 4.

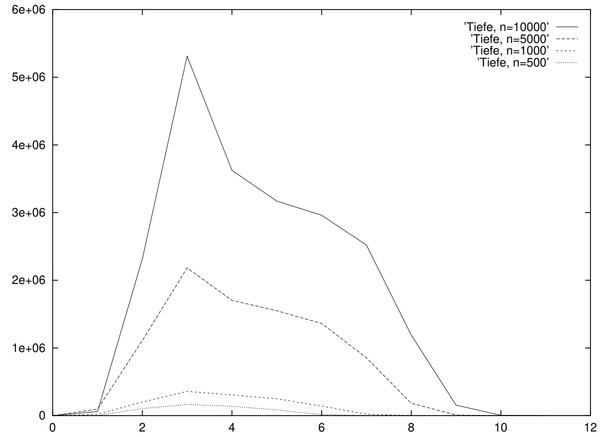

Abbildung 7: Verteilung der Rekursionstiefen beim Barnes-Hut Verfahren für verschiedene Problemgrößen, zufällige gaußverteilte Eingabedaten

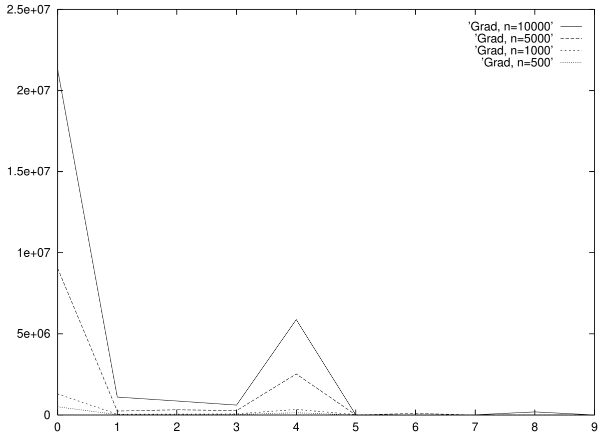

Abbildung 8: Verteilung der Verzweigungsgrade beim Barnes-Hut Verfahren für dieselben Eingabedaten

Bild 9 schließlich stellt die Rekursionstiefen bei der Engstes-Paar-Berechnung dar, die unabhängig von den zufälligen Eingabedaten immer dieselbe Kurve in Abhängigkeit von der Problemgröße liefert.

Die Probeläufe fanden entsprechend der frühen Phase der Realisierung auf sequentiellen Maschinen statt, was aber auf die Ergebnisse keine Auswirkung hat, da die Aufrufe im sequentiellen und parallelen Fall dieselben sind.

Zusätzlich zu den Messungen am Programm stellten wir durch einen Vergleich mit uninstrumentiertem Code auch den Mehraufwand fest, der durch die Instrumentierung verursacht wird. Bei der Eigenwertberechnung liegt er bei ca. 2% der Laufzeit, während er bei Barnes-Hut bis zu 33% der Laufzeit

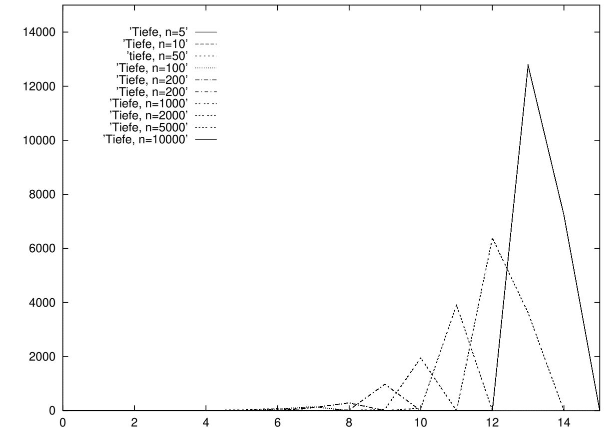

Abbildung 9: Verteilung der Rekursionstiefe beim Engstes-Paar Algorithmus für gleichverteilte Zufallsdaten

beträgt. Die Erklärung liegt in der aufwendigen Berechnung beim Matrixprogramm, während Barnes-Hut sehr feinkörnige Verzweigungen durchführt, bei denen der zusätzliche Code mehr ins Gewicht fällt.

6 Einsatz zur Optimierung

Nachdem wir dargestellt haben, daß eine Ermittlung der Kenngrößen möglich ist, beschreiben wir im folgenden kurz informell die daraus ableitbaren Optimierungen:

- **Rekursionstiefe** – ermöglicht Entscheidungen wie „erwartungsgemäß finden bei dieser Problemgröße jetzt nur noch zwei rekursive Abstiege statt, also lohnt sich die Auslagerung auf einen weiteren Prozessor nicht“, also PE-Zuordnung und Ressourcen-Allokation.

- **Verzweigungsgrad** – erleichtert eine bessere Verteilung der Ressourcen, da z.B. bei bekanntem Verzweigungsgrad auch die Form des Aufrufbaums bekannt ist und spezialisierte Verfahren zur Lastverteilung eingesetzt werden können. Außerdem kann man die Kosten eines Teilbaums bei statistisch bekannter Form besser schätzen, was wiederum die Verteilung erleichtert.

- **Rechenaufwand pro Rekursionsschritt** – ergibt in Verbindung mit einem Maschinenmodell Entscheidungen wie „der Rechenaufwand eines Schrittes ist sehr gering, also sollte man größere Anzahlen dieser Schritte auf einem Prozessor rechnen und nicht alle feinkörnig verteilen“, d.h. Granularitätsabschätzungen und entsprechende Abwägungen zwischen Verteilungs- vs. Berechnungskosten.

Um diese Entscheidungen möglichst trennscharf fällen zu können, werden die Meßkurven analysiert und ihre Merkmale wie Ort des Maximums, Breite des Maximums, Anstiegs- und Abstiegssteilheit usw. klassifiziert. Weiterhin wird auch die Abhängigkeit der Lage der Maxima von der Größe der Eingabedaten durch Regression festgestellt und die Kenngrößen damit angepaßt, so daß auch Vorhersagen für noch nicht gemessene Problemgrößen gemacht werden können.

7 Verwandte Arbeiten

J.P. Singh in Stanford befaßt sich mit hierarchischen n-Körper-Problemen auf Parallelrechnern [Sin93]. Der Schwerpunkt liegt auf effizienten Partitionierungstechniken für ausgewählte Probleme dieser Klasse, deren expliziter Parallelisierung sowie auf Überlegungen zur Skalierbarkeit passender Hardware-Architekturen.

Divacon ist eine parallele funktionale Sprache von Z.G. Mou an der Brandeis University in Waltham [Mou90]. Sie basiert auf einem formalen algebraischen Modell für Teile-und-Herrsche und wird anhand einiger kleinerer Beispielsprogramme demonstriert. Für irreguläre Probleme ist dieser Ansatz allerdings nicht sehr geeignet.

Die Gruppe um A. Rogers untersucht an der Princeton University dynamische parallele Datenstrukturen für die SPMD-Ausführung [RRH92]. Ihr Ansatz basiert auf „Futures" für Datenzugriffe und einem Übersetzer für annotierten C-Code. Dabei wird die Programmausführung jeweils dorthin verlagert, wo die benötigten Daten stehen.

Der Neuheitswert der in diesem Beitrag vorgestellten Arbeit liegt bei der der direkten Unterstützung von rekursiven Formulierungen, die allgemeiner sind als Teile-und-Herrsche, kombiniert mit der Ermittlung von auf diese Probleme zugeschnittenen Kenngrößen und ihrer Verwendung zu statischen und dynamischen Optimierungen.

8 Ausblick

Die vorgestellten Ergebnisse sehen vielversprechend aus für eine Verwendung in einem System zur Parallelisierung rekursiver Programme.

Die nächsten Schritte sind die Automatisierung der Instrumentierungen und die Auswertung der Meßergebnisse im Übersetzer selbst, sowie daraus resultierende Hinweise an das Laufzeitsystem. Zu diesem Zwecke wird der frei erhältliche Modula-3 Übersetzer [Har92] modifiziert.

Außerdem sollen zusätzliche Beispiele die Brauchbarkeit des Verfahrens weiter belegen, z.B. Voronoi-Diagramme und bitonisches Sortieren.

Der Mehraufwand der Instrumentierung soll durch Optimierung der Rekursions-Endaufrufe vermindert werden, damit er insbesondere bei späteren parallelen Ausführungen die Ergebnisse nur minimal verzerrt.

Schließlich ist auch eine automatische Auswertung der Meßkurven und die Schätzung der Kenngrößen, wie sie im letzten Abschnitt beschrieben wurde, zu implementieren.

Wir sind zuversichtlich, daß der in diesem Beitrag beschriebene Ansatz eine gute Grundlage für die effiziente parallele Ausführung rekursiver Programme bietet, die mit bisherigen Systemen nicht zufriedenstellend bearbeitet werden können.

Literatur

[CRY94] Soumen Chakrabarti, Abhiram Ranade, and Katherine Yelick. Randomized load balancing for tree-structured computation. In *Scalable High Performance Computing Conference*, pages 666–673, Knoxville, 1994. IEEE.

[Har92] Samuel P. Harbison. *Modula-3*. Prentice Hall, 1992.

[Mou90] Zhijing George Mou. *A Formal Model for Divide-and-Conquer and Its Parallel Realization*. PhD thesis, Yale University, May 1990.

[Phi94] Michael Philippsen. *Optimierungstechniken zur Übersetzung paralleler Programmiersprachen*. Fortschrittberichte 10/292. VDI Verlag, Düsseldorf, 1994.

[Ref95] Heiko Reffert. Beispieldaten zur Galaxiensimulation. Persönliche Mitteilung, Astronomisches Recheninstitut, Heidelberg, July 1995.

[RRH92] Anne Rogers, John H. Reppy, and Laurie J. Hendren. Supporting SPMD execution for dynamic data structures. Technical Report TR-374-92, Princeton University, June 1992.

[Sed88] Robert Sedgewick. *Algorithms*. Addison-Wesley, 2nd edition, 1988.

[Sin93] Jaswinder Pal Singh. *Parallel Hierarchical N-Body methods and their implications for parallel processors*. PhD thesis, Stanford University, February 1993.