Ein universelles Lastverteilungssystem und seine Anwendung bei der Isolierung reeller Nullstellen

Dissertationsschrift

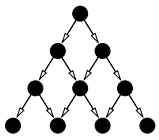
Thomas Decker

Zusammenfassung

Im Mittelpunkt der Arbeit steht das universelle Lastverteilungssystem „*Virtual Data Space*“ (*VDS*). Das System integriert sowohl bekannte Lastverteilungsverfahren als auch neue, die in dieser Arbeit vorgestellt werden. Bevor in Kapitel 6 auf die Konzeption des Systems eingegangen wird, werden in Kapitel 2 Grundlagen für die Beurteilung der Skalierbarkeit eines parallelen Algorithmus eingeführt. In den folgenden Kapiteln 3 und 4 werden zwei statische Planungsstrategien vorgestellt, und in Kapitel 5 wird die statische Lastbalancierung unabhängiger Aufgaben diskutiert. In Kapitel 7 wird die Parallelisierung der Descartes Methode für die Isolierung reeller Nullstellen mit Hilfe von VDS vorgestellt.

In Kapitel 2 wird das von Kumar eingeführte Konzept der *Isoeffizienz* formal definiert, und es werden einige grundlegende Eigenschaften, die sich aus dieser Definition ergeben, hergeleitet. Mit Hilfe dieser Definition kann die Isoeffizienz von Algorithmen angegeben werden, deren Effizienz nicht ausschliesslich von der sequentiellen Rechenzeit abhängt. Es zeigt sich, dass die Klasse der Probleme, für die es einen Algorithmus mit polynomieller Isoeffizienz gibt, eine echte Unterklasse der von Kruskal, Rudolph und Snir eingeführten Klasse EP ist.

In Kapitel 3 werden statische Planungsverfahren für den sogenannten *Pyramidengraphen* diskutiert. Eine Pyramide der Höhe n besteht aus den Knoten $(i,\ j)$ mit $0 \leq i,\ j < n$ und $i + j < n$. Jeder dieser Knoten benötigt für seine Berechnung die Ergebnisse der Vorgängerknoten $(i-1,\ j)$ und $(i,\ j-1)$. Derartige Abhängigkeiten werden unter anderem von Algorithmen erzeugt, die auf dynamischer Programmierung beruhen (z.B. Triangulierung von Polygonen). Weitere Anwendungen mit diesem Abhängigkeitsmuster finden sich in der Computeralgebra (z.B. vollständiges Horner-Schema, Taylor-shift). Die Aufgabe des Planungsalgorithmus besteht darin, die Knoten einer Pyramide der Höhe n auf P Prozessoren abzubilden.



Zunächst werden untere Schranke für die Isoeffizienz von Anwendungen hergeleitet, deren Abhängigkeitsgraph eine Pyramide ist. Unter anderem wird gezeigt, dass die Isoeffizienz in $\Omega(P^2)$ liegt, wenn alle Knoten die gleiche Bearbeitungszeit haben. Anschließend wird ein Planungsverfahren angegeben (*Pie-Mapping*), das, im Gegensatz zu den bislang bekannten Verfahren, den Graphen in P etwa gleich große, zusammenhängende Zonen teilt und so eine Reduzierung des Kommunikationsaufwandes von einem quadratischen Term zu einem linearen Term erreicht. Dieses macht sich in Experimenten bereits bei einer einfachen ebenenweisen Abarbeitung der Zonen durch eine deutlich höhere Effizienz bemerkbar. Eine weitere Verbesserung der Effizienz wird durch eine blockorientierte Abarbeitungsreihenfolge erreicht. Es wird gezeigt,

dass bei dieser Reihenfolge mehr Zeit für die Kommunikation zur Verfügung steht, als bei der ebenenweisen Bearbeitung.

Das in Kapitel 4 betrachtete Lastverteilungsproblem bezieht sich auf unabhängige Aufgaben, die ihrerseits parallel bearbeitet werden können. Das Interesse gilt vor allem dem Fall, in dem weniger Aufgaben als Prozessoren vorhanden sind. Es wird davon ausgegangen, dass sich die Aufgaben gleich verhalten, und dass die Zeit $t(p)$ für die Bearbeitung einer Aufgabe auf p Prozessoren bekannt ist. Das Problem besteht darin, bei gegebener Prozessoranzahl P und Aufgabenanzahl m jeder Aufgabe eine Prozessormenge und einen Startzeitpunkt so zuzuordnen, dass jeder Prozessor nur an einer Aufgabe gleichzeitig arbeitet, und dass die Gesamtberechnungszeit minimiert wird. Im Fall, dass die Aufgaben nicht dieselbe Laufzeitfunktion haben, ist das Problem selbst bei einer konstanten Prozessoranzahl in strengem Sinne NP-hart. Im Fall, dass die Aufgaben gleichartig sind, wird ein Planungsalgorithmus angegeben, dessen Laufzeit $O\left(n\,(2n+1)^{P^2-P}\,2^P\right)$ ist. Sie kann auf $O\left(m\;(k\,t(1))^{P-1}\;2^P\right)$ reduziert werden, falls die Bearbeitungszeiten $t(p)$ rational sind. Hierbei ist k das kleinste gemeinsame Vielfache der Zeiten $t(1),\ldots,t(P)$. Da der Algorithmus wegen der exponentiellen Abhängigkeit der Laufzeit von der Prozessorzahl in der Praxis nicht anwendbar ist, wird ein Approximationsalgorithmus angegeben, der sogenannte *phasenparallele Pläne* in $\Theta(m^2)$ Schritten berechnet. Die mit Hilfe dieser Pläne bewirkte Gesamtbearbeitungszeit ist höchstens doppelt so lang wie die eines optimalen Plans.

In Kapitel 5 wird die Platzierung unabhängiger Aufgaben für den Fall betrachtet, dass mehr Aufgaben als Prozessoren vorhanden sind. Es wird von gleichartigen Aufgaben ausgegangen, die am Anfang beliebig auf den Prozessoren verteilt vorliegen. Der Lastverteilungsalgorithmus soll die Aufgaben so umverteilen, dass jeder Prozessor anschließend über dieselbe Aufgabenanzahl verfügt. Dieses Problem ist in der Dissertation von Ralf Diekmann für datenparallele Anwendungen behandelt worden. In diesen Anwendungen ergeben sich durch die Nachbarschaftsbeziehungen zwischen den Aufgaben kanonisch auch Nachbarschaften zwischen den Prozessoren. Basierend auf diesen Nachbarschaften werden Diffusionstechniken eingesetzt, um eine Balancierung des Systems zu erreichen. Diese Techniken können jedoch auch bei unabhängigen Aufgaben eingesetzt werden. Durch die Unabhängigkeit der Aufgaben entfällt in diesem Fall die durch die Anwendung vorgegebene Prozessornachbarschaft. Es ergibt sich somit die Möglichkeit, eine künstliche Prozessortopologie so zu definieren, dass der Lastverteilungsalgorithmus möglichst effizient arbeitet. Wir betrachten zwei auf Diffusion basierende Lastverteilungsalgorithmen, die jeweils in zwei Phasen arbeiten. In der ersten Phase (*Flussberechnungsphase*) wird ein Lastfluss berechnet, und in der zweiten Phase (*Migrationsphase*) wird die Last entsprechend dieses Flusses verteilt.

Der Aufwand dieser Phasen wird experimentell untersucht. Es zeigt sich, dass der Aufwand sowohl von Topologieeigenschaften (wie Grad, Durchmesser, Anzahl der positiven Eigenwerte der Laplacematrix) als auch von der Balanciertheit des Flusses abhängt. Da ein balancierter Fluss zu einer kürzeren Migrationsphase führt, dafür aber aufwendiger in der Berechnung ist, ergibt sich ein Tradeoff zwischen dem Aufwand für die Flussberechnung und dem Aufwand der Migrationsphase.

In Kapitel 6 wird im einzelnen auf das Lastverteilungssystem *VDS* eingegangen. Es wird gezeigt, dass die Konzeption des Systems die im vorangegangenen Abschnitt diskutierten Anforderungen an ein universelles Lastverteilungssystem erfüllt. Ferner wird die Programmierung von VDS Anwendungen anhand eines einfachen Beispiels erläutert und die von VDS unterstützten Lastverteilungstechniken beschrieben. Schließlich zeigt ein experimenteller Vergleich mit

anderen Systemen, dass VDS die effizienteste Unterstützung für strikte Baumberechnungen auf Systemen mit verteiltem Speicher bietet.

In Kapitel 7 wird die Anwendung von VDS bei der Parallelisierung der Descartes Methode für reelle Nullstellenisolierung beschrieben. Es zeigt sich, dass diese Anwendung beispielhaft für die Kombination unterschiedlicher Programmierparadigmen (strikte Baumberechnungen und statische Aufgabengraphen) ist, und dass sie den Einsatz eines universellen Lastverteilungssystems erfordert.

Neben der experimentellen Untersuchung der Skalierbarkeit wird die Isoeffizienz der Descartes Methode analysiert. Es wird gezeigt, dass die Isoeffizienz der Methode bei beschränkter Koeffizientenlänge in $O(P^3 \log^2 P)$ liegt.