Multiprozessor Task Scheduling

Entwicklung und Vergleich von Algorithmen zur optimalen Auslastung eines Parallelrechners

Kai Baumgarten

Zusammenfassung

Diese Arbeit befaßt sich mit der Berechnung eines möglichst optimalen Schedules für eine gegebene Anzahl an formbaren Modulen, wobei jedes Modul von einer beliebigen Anzahl an Prozessoren ausgeführt werden kann. Die Laufzeit eines Moduls hängt dabei von der zur Ausführung des Moduls verwendeten Anzahl an Prozessoren ab. Die Laufzeiten aller Module sind vor dem Start des Schedulingalgorithmus vollständig bekannt. Als weitere Eingabe nimmt ein Großteil der in dieser Arbeit vorgestellten Schedulingalgorithmen einen gerichteten azyklischen Graph entgegen. Dieser repräsentiert die Abhängigkeiten zwischen den einzelnen Modulen. Gehört der gegebene Graph zur Klasse der serien - parallelen Graphen, so wird diese zusätzliche Strukturinformation von einem Algorithmus zur Erstellung des Schedules ausgenutzt.

Als Ergebnis liefern alle Algorithmen einen Schedule, welcher eine möglichst geringe Gesamtlaufzeit zur Ausführung aller Module auf einer gegebenen Anzahl an Prozessoren benötigt. Zum Erreichen einer minimalen Gesamtlaufzeit versuchen alle Algorithmen die Parallelität zwischen verschiedenen Modulen (Taskparallelität) und die Parallelität innerhalb eines Moduls (Datenparallelität) optimal auszunutzen.

Im Gegensatz zu vielen anderen Arbeiten zu diesem oder einem verwandten Thema muß bei den meisten der hier vorgestellten Algorithmen der Aufwand zur Ausführung eines Moduls bei der Hinzunahme eines Prozessors nicht zwangsweise steigen. Dadurch sind die in dieser Arbeit angegebenen Schedulingalgorithmen unabhängig von den Werten der Eingabedaten nutzbar.

Bei der Entwicklung der Algorithmen wurde viel Wert auf ihre praktische Verwendbarkeit gelegt. Dies äußert sich insbesondere in einer geringen Laufzeit der Algorithmen zur Berechnung des Schedules, sowohl bei einer sehr großen Anzahl an gegebenen Modulen, als auch bei sehr vielen zur Verfügung stehenden Prozessoren. Die Laufzeit der meisten in dieser Arbeit vorgestellten Algorithmen wird durch die Sortierung der Module bezüglich ihrer Modullaufzeit dominiert. Es werden aber auch Algorithmen angegeben, deren Laufzeit in der Größenordnung der Mächtigkeit der Eingabedaten liegt. Die in verwandten Arbeiten vorgestellten Schedulingalgorithmen haben sehr häufig eine wesentlich höhere Laufzeit, so daß diese nur bedingt in der Praxis einsetzbar sind.

Selbstverständlich soll trotz der geringen Laufzeiten der Algorithmen ein sehr guter Schedule berechnet werden. Falls keine Abhängigkeiten zwischen den Modulen vorliegen, so wird ein Schedule erreicht, welcher im schlechtesten Fall die dreifache Gesamtlaufzeit gegenüber der Gesamtlaufzeit des optimalen Schedules benötigt. Liegen hingegen Modulabhängigkeiten vor, so richtet sich die garantierte Ergebnisgüte bei allen vorgestellten Schedulingalgorithmen nach den Eigenschaften der Eingabedaten.

Die in der Arbeit angegebenen Schedulingalgorithmen lassen sich nach ihrer Funktionsweise grob in drei Gruppen einteilen: Die erste Art partitioniert alle Module in mehrere Modulmengen, so daß zwischen den in einer Modulmenge enthaltenen Modulen keine Abhängigkeiten mehr bestehen. Für jede dieser Modulmengen wird dann ein Schedule errechnet. Diese Teilschedules werden am Ende zu einem Gesamtschedule zusammengefügt. Die zweite Art versucht, durch die gezielte Parallelisierung von Modulen, die auf dem kritischen Pfad liegen, einen möglichst guten Schedule zu berechnen. Die dritte Art reduziert schrittweise die Modulanzahl. Dies geschieht durch die Zusammenlegung zweier oder mehrerer in bestimmten Abhängigkeiten stehender Module. Diese Art des Schedulings ist im Gegensatz zu den erstgenannten beiden Arten noch in keiner anderen bekannten Arbeit untersucht wurden.

Alle vorgestellten Schedulingalgorithmen wurden in C implementiert und anhand von zufällig erstellten Eingabedaten umfangreich getestet. Dabei lag die benötigte Gesamtlaufzeit des erzielen Schedule fast immer unter dem 1,3-fachen der grob nach unten abgeschätzten Gesamtlaufzeit des optimalen Schedules. In anderen Arbeiten wurden Schedulingalgorithmen mit pseudopolynomieller Laufzeit entwickelt, welche für den ermittelten Schedule gegenüber dem optimalen Schedule eine maximal um den Faktor 4,7 höhere Gesamtlaufzeit garantieren. Die praktisch ermittelten Ergebnisse der in dieser Arbeit vorgestellten Algorithmen liegen also in nahezu allen Fällen in dem von laufzeitintensiven Algorithmen garantierten Bereich, was wiederum für den Einsatz der schnellen Algorithmen in der Praxis spricht.

Weiterhin wurden die Schedulingalgorithmen an Beispielen, denen die LR - Zerlegung nach Gauß, die Matrixmultiplikation bzw. die Fast Fourier Transformation zugrunde liegt, getestet. Anhand dieser Beispiele wurden die Schedulingergebnisse der vorgestellten Algorithmen direkt miteinander verglichen.

Zusätzlich wird in dieser Arbeit für sehr viele Schedulingalgorithmen auch eine Version für nichtformbare Module vorgestellt. Dadurch können auch Module verarbeitet werden, die aufgrund ihrer internen Implementierung eine bestimmte Anzahl an Prozessoren voraussetzen.

Multiprocessor Task Scheduling

Design and Comparison of Algorithms for an Optimal Utilisation of Parallel Computers

Kai Baumgarten

Abstract

This thesis deals with the calculation of a schedule for an arbitrary, but fixed number of moldable parallel tasks, where each task can be executed using an arbitrary number of processors. Depending on the number of processors, different execution times may result for different tasks. The execution times of the tasks are assumed to be known before the calculation of the schedule starts. Moreover, a directed acyclic graph might be given. This graph represents the dependencies between the tasks. One presented scheduling algorithm is especially designed for graphs that belong to the class of serial - parallel graphs. This algorithm uses the additional information provided by the graph as a basis for the calculation of the schedule.

The goal of all algorithms presented in this thesis is to minimise the resulting makespan, which means that the overall completion time of the tasks should be a minimum. To get a good schedule, all algorithms try to use the parallelism between different tasks (task parallelism) and the parallelism within a single task (data parallelism) in an efficient way.

In contrast to other publications on the same or similar topics, this thesis does not generally assume that the work to execute a task increases if the number of processors used is increased. Most of the algorithms presented work without this limitation. That's why these algorithms can be used on arbitrary input data.

All algorithms presented are developed with respect to their practicality. This implies that all algorithms should have a minimal runtime, especially if the number of given tasks and/or available processors is high. The runtime of nearly all algorithms is dominated by the time used to sort the tasks. Additionally this thesis presents algorithms with a runtime that is bound linearly by the size of the input data. Algorithms in other publications usually have a much larger runtime. Therefore those algorithms can practically only be used for a quite small number of tasks.

Nevertheless, the short runtime of the presented algorithms does not imply that the calculated schedule is not close to the optimal one. If no dependencies between the tasks are given, it can be proven that the worst-case running time of the tasks using the calculated schedule is never more than three times the running time using the optimal schedule. In case of dependencies between the tasks, the quality of the schedule depends on the properties of the input data.

It is possible to categorise the algorithms presented in this thesis into three groups: The algorithms of the first group separate all tasks into different sets of tasks. There are no dependencies allowed between each pair of tasks within the same set. Afterwards the algorithms calculate a schedule for each set of tasks. These subschedules are finally put together for the overall schedule. The algorithms of the second group try to parallelize the tasks in the critical path to get a good schedule. The algorithms of the third group reduce, step by step, the number of tasks by unifying two or more tasks that are specially dependent on each other to a new task. The last group of algorithms is, in contrast to the first two groups, not part of any other known publication.

All presented algorithms have been implemented and checked extensively by the usage of randomly created input data. In nearly all cases the created schedule has a completion time which is less than 1.3-times of the lower bound of the completion time of the optimal schedule. There are algorithms presented in other publications that guarantee a schedule which needs at most 4.7-times of the completion time of the optimal schedule. Unfortunately the runtime needed to create the schedule is pseudo-polynomial. Since the results computed with the fast algorithms presented in this thesis are almost always within the guaranteed range of the more time consuming algorithms, the fast algorithms are usually the better choice in practice.

Moreover all algorithms are additionally tested with non-random examples based on Gaussian elimination, matrix multiplication, and fast Fourier transformation. These examples provide a data basis to compare all algorithms presented in this thesis.

This thesis also presents a version for non-moldable tasks for many of the scheduling algorithms. Therefore it is also possible to schedule tasks that require a predefined minimum number of processors for their execution.