Online Feedback-Directed Optimizations for Parallel Java Code

Diss. ETH No. 21080

Online Feedback-Directed Optimizations for Parallel Java Code

A dissertation submitted to
ETH Zurich

for the degree of
Doctor of Sciences

presented by
Albert Noll
Dipl.-Ing., TU Graz
born April 20, 1981
citizen of Austria

accepted on the recommendation of
Prof. Dr. Thomas R. Gross, examiner
Prof. Dr. Samuel P. Midkiff , co-examiner
Prof. Dr. Walter Binder, co-examiner

2013

Abstract

Multi-core processors are nowadays standard in servers, desktop computers, and mobile devices. To make applications exploit the new hardware resources that are provided by the large number of cores, programmers must write parallel code that runs concurrently on the available cores. State-of-the-art parallel programming techniques require the programmer to define concurrent code regions and the conditions (concurrency condition) under which concurrent code regions are executed in parallel. If the concurrency conditions are set unlucky, the performance of parallel code can drop way below its potential. Moreover, it is even possible that parallel code performs slower than a corresponding sequential version, if the overhead that is associated with every parallel execution exceeds the performance gain of a parallel execution.

This thesis examines an automated, online, and feedback-directed approach to determine the concurrency conditions under which concurrent code regions are effectively executed in parallel. The approach is automated, since the runtime system determines the concurrency conditions based on the knowledge which code regions can be executed concurrently and requires no additional input. The approach is online, since the concurrency conditions are determined at run-time. Traditional parallel programming models require a static specification of the concurrency conditions. Finally, the approach is feedback-directed, since the concurrency conditions are constantly re-evaluated at run-time.

The thesis explores the determination of the concurrency conditions in the context of the Java platform. The extension of the concurrency interface between the Java source language and the Java Virtual Machine (Java VM) enables the Java VM to determine the concurrency conditions automatically at run-time. The concurrency interface defines the available knowledge of the concurrency structure of an application in the Java VM and the just-in-time compiler (JIT compiler). The standard Java concurrency interface is extended by two constructs: concurrent statements and synchronization statements. A concurrent statement *may* be executed in parallel with the subsequent statement. A synchronization statement enables the synchronization of concurrent statements. Concurrent statements and synchronization statements are represented explicitly in the intermediate representation of the JIT compiler. Such an explicit representation enables the JIT compiler to instrument a concurrent statement so that the profiling system can precisely determine

the behavior of the concurrent statement. Based on the measured behavior, the JIT compiler compiles a concurrent statement to (i) sequential code, (ii) parallel code, or (iii) merges a concurrent statement with another concurrent statement and thereby changes the parallel code granularity of the application.

This thesis presents the design, implementation, and evaluation of the extended concurrency interface for the Java platform. More precisely, the implications of such a system for the Java language, the Java VM, the JIT compilers, and the Java Memory Model (JMM) are discussed in detail. We obtain experimental results from a prototype implementation and show that our automated determination of the concurrency conditions results in competitive performance compared to hand-optimized code. More specifically, if the hand-optimized version finds good concurrency conditions, our approach performs equally fast or slightly slower (15%). If, however, the concurrency conditions of the hand-optimized code are set unlucky, our approach is up to 4X faster.

Zusammenfassung

Mehrkernprozessoren findet man in Servern, Desktop-Computern und mobilen Geräten. Die zusätzlichen Prozessorkerne stellen neue Hardware Ressourcen zur Verfügung. Einzelne Applikationen können diese Ressourcen nur dann effizient nutzen, wenn paralleler Code nebenläufig auf den Mehrkernprozessoren ausgeführt wird. Auch bei der Anwendung modernster Programmiertechniken müssen die Kriterien zur parallelen Ausführung des Programms von Hand definiert werden. Diese Kriterien bezeichnen wir als Parallelisierungskriterien. Werden die Parallelisierungskriterien falsch spezifiziert, kann die Ausführungszeit eines parallelen Programms langsamer sein als eine sequentielle Ausführung. Dies kann auftreten, wenn der Mehraufwand, der mit jeder parallelen Ausführung verbunden ist größer ist als der Leistunggewinn, der von einer parallelen Ausführung herrührt.

Das Ziel dieser Dissertation ist eine Methode zu entwickeln, welche die Parallelisierungskriterien automatisch bestimmt, so dass paralleler Code effizient ausgeführt wird. Die Parallelisierungskriterien werden zur Laufzeit bestimmt und passen sich dem Verhalten des Programms an. Diese Dissertation beschreibt eine solche Methode für die Java Umgebung. Dazu wird die Schnittstelle zwischen dem Java Quellcode und der Java virtuellen Maschine (Java VM) um zwei Anweisungen erweitert: potentiell parallele Anweisungen und blockierende Anweisungen. Potentiell parallele Anweisung können, müssen aber nicht parallel ausgeführt werden. Blockierende Anweisungen halten die Programmausführung an bis alle potentiell parallelen Anweisungen ausgeführt sind. Potentiell parallele Anweisungen sowie blockierende Anweisungen werden explizit im Just-In-Time (JIT) Compiler dargestellt. Diese explizite Darstellung ermöglicht das Laufzeitverhalten von potentiell parallelen Anweisungen genau zu analysieren und die Parallelisierungskriterien entsprechend zu optimieren. Der JIT Compiler kann den Mehraufwand reduzieren, indem er zwei (oder mehrere) parallele Anweisungen zu einer parallelen Anweisung zusammenfasst.

Diese Dissertation beinhaltet das Design, die Implementation und die Evaluation der erweiterten Java Schnittstellen. Die Arbeit beschreibt wie sich potentiell parallele Anweisungen und blockierenden Anweisungen auf die Java Programmiersprache, die JIT Compiler, und das Java Speichermodell auswirken. Zusätzlich wurde eine detaillierte Laufzeitanalyse der automatisch bestimmten Parallelisierungskriterien durchgeführt. Die Resultate zeigen, dass manuell optimierte Parallelisierungskriterien zu einer bis zu 15% schnelleren Ausführung von parallelem Code führen kann.

Wenn die Parallelisierungskriterien allerdings nicht optimiert (oder falsch gesetzt) sind, kann die daraus resultierende parallele Ausführung bis zu 4X langsamer sein.