Machine-aware memory allocation and synchronization

Author(s):
Kaestle, Stefan

Publication Date:
2016

Permanent Link:

https://doi.org/10.3929/ethz-a-010815953 →

DISS. ETH NO. 23823

Machine-aware memory allocation and synchronization

A thesis submitted to attain the degree of

DOCTOR OF SCIENCES of ETH ZURICH

(Dr. sc. ETH Zurich)

presented by

STEFAN KAESTLE

Diplom-Informatiker, Karlsruher Institut für Technologie (KIT)

born on 25.04.1986

citizen of Germany

accepted on the recommendation of

Prof. Dr. Timothy Roscoe (ETH Zurich), examiner
Prof. Dr. Gustavo Alonso (ETH Zurich), co-examiner
Dr. Timothy Harris (Oracle Research Labs, Cambridge, UK), co-examiner

2016

Abstract

The performance of parallel programs on multicores critically depends on hardware-conscious implementations that considers low-level characteristics of the machine. Among others, this applies to the implementation of memory allocation and synchronization primitives.

However, as a result of hardware being increasingly complex, it is hard to understand the implications of their characteristics on software's performance. Worse, hardware diversity makes it non-trivial to apply optimizations that are promising for one piece of hardware to another. Finally, fast paced changes in hardware-design require programmers to adopt their programs frequently to keep up with trends dictated by hardware making manual tuning an unattractive solution.

One possible solution to the problem is to provide systems that model hardware characteristics and automatically configure themselves accordingly. For *memory allocation*, we investigate this with Shoal, a smart machine-aware memory allocator that applies data distribution, partitioning and replication automatically when allocating memory and further uses advanced hardware features such as DMA engines and large/huge pages where applicable. The choice depends on the application's memory access patterns, which we propose to extract automatically from high-level parallel programs. We implement and evaluate this approach for Green-Marl, a domain specific language in the field of graph analytics.

The second part of this thesis investigates *synchronization* implemented on top of message-passing-based group communication primitives. Optimizing them gains importance even within single machines as a result of an increase in software-parallelism and resulting scalability challenges of shared-memory implementations. We propose the use highly tuned multicast trees as a building block for higher-level applications. If made machine-aware, high level protocols such as barriers or agreement protocols built on top of these trees then directly benefit from optimizations applied on the lower level. As a result, they are simpler to program while being competitive or better than hand-tuned state-of-the art implementations. Further, it relieves programmers from having to understand intricacies of multicore hardware. The challenge with optimizing multicasts is to select the tree topology and the order in which messages are sent from each core. Our approach to solving

this problem is to generate the tree from a small number of intuitive heuristics while simulating it's execution. Simulation requires a detailed model of a machine's message-passing performance. We found that available hardware information is too coarse grained and propose to acquire the model from a small number of carefully crafted micro-benchmarks automatically.

With both our memory allocator and our synchronization framework we show two examples of how to built a system that achieves good performance on a wide range of multicore machines without programmers having to manually tune their implementations and without them having to understand the intricate details of modern multicore hardware.

Zusammenfassung

Die Leistung paralleler Programme kann leiden wenn Speicherallozierung und Synchronisierung nicht optimal konfiguriert sind. Leider machen es zunehmend komplexe und vielfältige Hardware schwierig und aufwändig die Auswirkungen ihrer Charakteristiken auf die Leistung von Programmen zu verstehen. Da eine gute Konfiguration abhängig von der Maschine ist können Optimierungen einer Maschine direkt auf eine andere übertragen werden. Die daraus folgende Notwendigkeit manueller Optimierungen für jede einzelne Maschine sind wenig attraktiv.

In dieser Arbeit schlagen wir automatische Speicherallozierung vor, die intelligent Daten verteilt, partitioniert und repliziert. Darüber hinaus werden erweiterte Beschleuniger wie DMA Hardware und größere Speicherseiten automatisch verwendet. Unser System basiert auf der Verwendung von Informationen über die Struktur von Speicherzugriffen. Diese können von Programmierern angeben werden oder im Idealfall automatisch aus parallel Programmiersprachen höherer Ebenen extrahiert werden. Wir stellen ein System vor das dieses für ein Graphanalyse-Framework durchführt.

Im zweiten Teil dieser Dissertation untersuchen wir Primitive zur Synchronisierung basierend auf Gruppenkommunikation. Wegen der zunehmenden Parallalität von Software und der daraus resultierenden Schwierigkeiten bei der Skalierung von shared-memory Programmierung erlangt dies immer mehr an Bedeutung. Unser Lösungsansatz für dieses Problem sind hochoptimierte Bäume für Gruppenkommunikation. Wenn diese so konstruiert werden, dass sie die Eigenheiten von Rechnern verstehen und berücksichtigen, können syntaktisch höherwertige Programme mit vergleichbarer oder besser Leistung auf diesen aufgebaut werden. Das ist möglich ohne das Programmierer manuell Optimierungen für die Eigenheiten einzelner Maschinen durchführen müssen, da diese automatisch von den Optimierungen auf niedrigeren Ebenen profitieren. Als Beispiele zeigen wir Barrieren und Protokolle zum gegenseitigen Verständnis.

Die Herausforderung bei der Verwendung von Kommunikation auf der Grundlage von Bäumen ist die Auswahl einer geeigneten Baumstruktur und der Reihenfolge der Sendeoperationen in jedem Knote des Baumes. Wir zeigen wie wenige speziell designte Benchmark-Programme verwendet werden können um ein angebrachtes Modell des Rechners automatisch zu generieren.

Darauf aufbauend kann die Baumstruktur und Sendereihenfolge dann offline mit einer Event-basierten Simulation bestimmt werden. Die Simulation sagt den Systemstatus auf Grundlage der Benchmarks voraus und implementiert Heuristiken zur Konfiguration der Topologie.

Unsere Optimierungen von Speicherallozierung und Synchronisierung zeigt wie eine gute Leistung von parallelen Programmen erzielt werden kann ohne dass Programmierer die Eigenheiten von moderner Hardware verstehen und von Hand Anpassungen vornehmen müssen. Wie wir in dieser Dissertation zeigen funktioniert das auf einer breiten Palette von Rechner.