Doctoral Thesis

Seamless Heterogeneous Computing: Combining GPGPU and Task Parallelism

Author(s):
Kolesnichenko, Alexey

Publication Date:
2016

Permanent Link:

https://doi.org/10.3929/ethz-a-010813587 →

Diss. ETH No. 24006

Seamless Heterogeneous Computing: Combining GPGPU and Task Parallelism

A thesis submitted to attain the degree of
DOCTOR OF SCIENCES of ETH ZURICH
(Dr. sc. ETH Zurich)

presented by
Alexey Kolesnichenko
Specialist in Mathemathics and System Programming, Lomonosov Moscow State University, Russia

born on
January 17th, 1989

citizen of
Russia

accepted on the recommendation of

Prof. Dr. Bertrand Meyer, examiner
Dr. Judith Bishop, co-examiner
Prof. Dr. Torsten Hoefler, co-examiner
Dr. Christopher M. Poskitt, co-examiner

2016

Abstract

Concurrent and parallel computing is ubiquitous today. Modern computers are shipped with multicore CPUs and often equipped with powerful GPUs, which can be used for general-purpose computing (GPGPU) as well. CPUs have evolved into highly sophisticated devices, with many intricate performance-enhancing optimizations, that are targeted for task-parallel computations. However, the number of CPU cores is usually quite limited, and using CPU-based computations may not be the best fit for the data-centric parallelism. GPUs, on the other hand, feature hundreds and thousands of simpler cores, providing a viable alternative to CPUs for data-centric computations. In order to write efficient applications, programmers should be able to use these heterogeneous devices together.

The programming model for GPUs is quite different from traditional CPU-based programs. While there are some solutions for using GPGPU in high-level programming languages via various bindings by executing native code blocks, these solutions lack many aspects of high-level languages and are error-prone.

Task parallel computing on CPUs is much more challenging than sequential computing, with new kinds of bugs, both related to correctness and performance. Many of these issues are caused by programming models based on low-level concepts, such as threads, and lack of documentation on important aspects, such as task cancellation and subtle differences in semantics for similar APIs across languages.

In this thesis we explore two complementary approaches, which allow software developers to solve these problems within a flexible and general framework. We investigate how programmers can benefit from the advantages of CPUs and GPUs together, and within a high-level programming language.

For data-centric problems, we propose SafeGPU, a high-level library that abstracts away the technicalities of low-level GPU programming, while retaining the speed advantage. Our approach is modular: the user benefits from combining low-level primitives into high-level code. This modularity al-

lows us to introduce an optimizer, which analyzes code blocks and produces optimized GPU routines. SafeGPU also integrates the design-by-contract methodology, which increases confidence in functional program correctness by embedding executable specifications into the program text. Finally, we show that with SafeGPU contracts can be monitored at runtime without diminishing the performance of the program, even with large amounts of data.

For task parallel problems, we propose an extension of SCOOP, a concurrent object-oriented programming model, in order to leverage its strong safety and reasoning guarantees. Our extension includes a safe mechanism for task cancellation and asynchronous event-based programming. We address the lack of documentation for task cancellation by providing a survey on existing techniques and classify them based on use cases. Finally, we show how our extended version of SCOOP cooperates with SafeGPU to provide a seamless heterogeneous experience for programmers.

Zusammenfassung

Parallelrechner sind heute allgegenwärtig. Moderne Computer sind mit Multicore-CPUs und oft mit leistungsstarken GPUs ausgestattet, welche ebenfalls für Allzweck-Berechnungen auf Grafikprozessoreinheiten (GPGPU) verwendet werden können. Die CPUs sind heute hochentwickelte Instrumente mit unzähligen leistungsverbessernden Optimierungen. Die Anzahl CPU Kerne ist jedoch normalerweise eher limitiert und CPU-basierte Rechnungsprozesse sind möglicherweise nicht die beste Lösung für datenzentrische Parallelverarbeitungen.

GPUs hingegen beinhalten Tausende von einfacheren Verarbeitungskernen und stellen hiermit für datenzentrische Parallelverarbeitungen eine geeignete Alternative zu CPUs dar. Das Programmiermodell für GPUs unterscheidet sich von traditionellen CPU-basierten Programmen. Zwar existieren verschiedene Lösungen für die Anwendung von GPGPU in höheren Programmiersprachen durch das Binding von ausführenden nativen Code-Blöcken, jedoch sind bei diesen viele Aspekte höherer Programmiersprachen vorenthalten. Dazu sind die meisten Lösungen fehleranfällig. Eine andere Quelle für Fehler sowie Performance-Probleme könnte der schlecht organisierte Task-Parallelismus sein.

In dieser Arbeit behandeln wir zwei komplementäre Ansätze, welche es Softwareentwicklern erlauben, diese Probleme zu lösen und ihnen eine flexible Methode zur Verfügung stellt, die in vielen unterschiedlichen Situationen anwendbar sein könnte. Wir untersuchen, wie Programmierer von den Vorteilen von CPUs und GPUs im Zusammenspiel und innerhalb einer höheren Programmiersprache profitieren können. Weiter erweitern wir unser Modell mit einem Task Cancellation-Verfahren, welches eine sichere und transparente Art darstellt, einen Task abzubrechen.

Wir beginnen damit, datenzentrische Probleme aufzugreifen und führen die SafeGPU ein, eine High-Level Library, welche die technischen Einzelheiten des low-level GPU Programmierens wegabstrahiert und gleichzeitig den Geschwindigkeitsvorteil beibehält. Unser Ansatz ist modular aufgebaut: der Benutzer profitiert von der Integration von low-level Primitiven in einen

Hochsprachen-Code. Diese Modularität erlaubt es uns, einen Optimierer einzuführen, welcher Code Blocks analysiert und optimierte GPU-Abläufe produziert.

SafeGPU integriert zudem die Design-By-Contract Methodologie, welche das Vertrauen in die funktionsgemässe Richtigkeit des Programms erhöht, indem ausführbare Spezifikationen im Programmtext eingebettet werden. Zuletzt zeigen wir, dass Runtime Contract-Checking in SafeGPU realisierbar wird, da die Contracts in der GPU ausgeführt werden können.

Als weiteren Aspekt unserer Doktorarbeit diskutieren wir die Task-basierten Probleme und untersuchen den wichtigen Fall der Task Cancellation, welcher für das gleichzeitige Ausführen mehrerer Tasks sehr bedeutsam ist. Wir schlagen eine Erweiterung für SCOOP vor (Simple Concurrent Object Oriented Programming Modell, als Teil der Eiffel Programmiersprache), welche einen sicheren und einfachen Weg für Task Cancellation darstellt und wir zeigen, dass dieser Mechanismus zusammen mit SafeGPU angewendet werden kann, um das Strukturieren rechnerischer Prozesse zu vereinfachen.