View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Chalmers Research

Thesis for The Degree of Licentiate of Engineering

An Automated and Controlled Numerical Precision Reduction Framework for GPUs

Alexandra Angerd

Division of Computer Engineering
Department of Computer Science & Engineering
Chalmers University of Technology
Göteborg, Sweden, 2018

An Automated and Controlled Numerical Precision Reduction Framework for GPUs

Alexandra Angerd

© Alexandra Angerd, 2018

Technical Report No 182L
ISSN 1652-876X
Department of Computer Science & Engineering
Division of Computer Engineering
Chalmers University of Technology
SE-412 96 Göteborg
Sweden
Telephone: +46 (0)31-772 1000

Printed by Reproservice
Chalmers Tekniska Högskola
Göteborg, Sweden 2018.

An Automated and Controlled Numerical Precision Reduction Framework for GPUs

Alexandra Angerd
Department of Computer Science and Engineering
Chalmers University of Technology

Thesis for the degree of Licentiate of Engineering, a swedish degree between M.Sc. and Ph.D.

Abstract

Reducing the precision of floating-point values is an effective approach to achieve higher performance as well as higher energy-efficiency. This is especially true for GPUs, since many of its common tasks are inherently insensitive to precision-reduction. A substantially lower bitwidth can open up for many novel microarchitectural optimizations such as resource-efficient register files, functional units, and cache memory subsystems. However, to reduce the precision of floating-point values in a controlled manner, a connection has to be established between the application and the microarchitecture, since it is decided at the application level if deviations from the exact answer are tolerable.

This thesis proposes a GPU framework which establishes such a connection. The first part of the framework consists of a method for automatically selecting an appropriate precision for each floating-point value given the tolerable output deviation. The results show that by allowing a small, but acceptable, degradation of output quality, the number of bits needed to represent the floating-point values can be significantly reduced.

The second part of the framework is a novel GPU register file organization together with a register allocation algorithm capable of leveraging the precision-reduced floats given by the first part of the framework. The register allocation algorithm uses the precision-reduced floats to lower the register footprint of each thread. This is of great importance for GPUs since, unlike traditional CPU architectures, GPUs hide latency by keeping a large number of threads in flight simultaneously. Also, to enable fast context switching, the state of all active threads are readily available in the register file. As the thread register footprint limits the number of active threads, it might impede latency hiding. Our evaluation shows that the increase in active threads is translated into a significant performance improvement when using our proposed GPU register file organization, for a smaller cost than increasing the number of threads by using a larger register file.

Keywords: Floating-Point Precision, GPU, Approximate Computing, Microarchitecture, Register File

Acknowledgment

I would like to thank my advisor Professor Per Stenström for his invaluable guidance. With his great knowledge in research and organized way of solving any kind of problem he has taught me so much, and I am really looking forward to learn even more.

I also would like to thank my co-advisor Dr. Erik Sintorn, for his immense support and patience, both in general and during our sessions of turning problems inside out.

Thanks to Dr. Lars Svensson, for always supporting me and for all interesting discussions concerning everything under the sun.

I am also grateful to all my great colleagues and friends at Chalmers: Nadja, Christoffer, Lena, Dan, Sverker, Fatemeh, Rolf, Jan, Miquel, Pedro, Ulf, Monica, Erik, Boel, Victor, Andreas, and others.

Finally, I want to express my deepest gratitude to my wonderful family, Sven-Åke, Marie, Catharina; my love, Calle, and Ångström, for supporting me in all my life decisions and always being by my side.

This thesis is based upon work supported by a grant from the Swedish Research Council (Vetenskapsrådet) under the Approximate Algorithms and Computing Systems Project.

Alexandra Angerd
Mölndal, July 2018

List of Appended Papers

This thesis is based on the following papers:

I. **Alexandra Angerd**, Erik Sintorn, and Per Stenström. “A Framework for Automated and Controlled Floating-Point Accuracy Reduction in Graphics Applications on GPUs”, *ACM Transactions on Architecture and Code Optimization (TACO), Volume 14 Issue 4, December 2017* .

II. **Alexandra Angerd**, Erik Sintorn, and Per Stenström. “A Register File Organization to Support Variable Floating-Point Precision in GPUs”, Submitted to *The 25th International Symposium on High-Performance Computer Architecture (HPCA) 2019.*

Contents

1 Introduction

In an era when computational performance growth is heavily constrained by a number of technical barriers (end of Dennard scaling and end of transistor scaling according to Moore's law), radical new ways are needed to continue increasing computing efficiency. Reducing the precision of floating-point values by narrowing the bitwidth is an effective approach to both achieve higher performance [3] as well as higher energy-efficiency [4, 12]. This is especially true for GPUs, since many of its common tasks such as image-rendering, and recently also the growing domain of deep learning inference, are inherently insensitive to precision-reduction [9, 10]. A substantially narrower width of floating-point values can open up for many novel optimization approaches such as more resource-efficient register files, data paths, functional units, and cache memory subsystems.

However, reducing the precision of floating-point values needs support across many of the computer abstraction layers. At the application level, it must be decided if deviations from the exact answer is tolerable, and if it is, the magnitude of tolerable deviations must be established. Then, at the source-level or the assembly language level, each and every floating-point value has to be annotated with an appropriate precision in such a way that the precision of the final output is guaranteed to be within the deviation bound decided at the application level. At last, architectural support is needed to make use of the annotations to utilize register file, data path, functional unit, or cache resources more efficiently. In short, the possibilities for utilization of the architectural resources is decided at the application level. Therefore, a connection has to be established between the application and the microarchitecture.

This thesis proposes a GPU framework which establishes such a connection, with the goal of reducing floating-point precision in an *automated* and *controlled* fashion. Controlled refers to guaranteeing the error deviation requirement set at the application level. Automated means that the framework automatically, and transparently to the programmer, classifies the precision-reduction possible for each floating-point value at the instruction level. The thesis is a summary of two papers, referred to by the Roman numerals **I** and **II**. The first contribution is a method for automating the process of selecting the precision of each floating-point value given a certain application-specific output quality threshold (**I**). Unlike previous methods [11, 7], our method operates directly on compiled and optimized assembly code, and annotates each separate floating-point destination register with an appropriate precision.

The second contribution is a novel GPU register file organization (**I** and **II**) together with a register allocation algorithm (**I**) which leverages the floating-point annotations for increased capacity and performance. The functionality specification of the register file is first presented in **I**, while the detailed microarchitectural implementation together with an overhead analysis is presented in **II**.

The last contributions are evaluations of the benefits in terms of reduced register file pressure and improved performance. **I** shows that, for a negligible quality loss, up to twice as many threads in a GPU can be active simultaneously. In **II**, we show that this is translated to a significant improvement in performance.

The rest of the thesis is organized as follows. The problem formulations and contributions made in **I** and **II** are summarized in Sections 2 and 3 respectively. Finally, concluding remarks are made in Section 4, which also points out possible future work.

2 Floating-Point Accuracy Reduction for Graphics Applications

In order to leverage narrow floating-point values for optimizations at architecture level, they have to be annotated with suitable precisions at the instruction level. Previous efforts have been made to provide such a framework. Precimonious [11] provides a framework for selecting among IEEE754-compliant data types for each source-level floating-point variable in order to improve performance while generating an output within some guaranteed error bound. However, there are three limitations: First, the precision-levels are constrained to IEEE-754 compliant data types. Second, the optimization targets source-level floating-point values as opposed to instruction-level floating-point variables. Third, the algorithm has limited scalability ($O(pn^2)$, where n is the number of floating-point values and p is the number of precision levels).

Also Chisel [7] supports automatic precision-tuning, but like Precimonious, it does not target instruction-level registers. In addition, Chisel employs a static interval analysis, which yields conservative results.

In **I**, we present a data-driven precision-selection method which, transparently to the programmer, classifies the amount of precision reduction possible for individual values at the *instruction level*, given a quality threshold and a representative set of input data. Also, the algorithm we employ has a worst-case complexity of $O(pn)$ as opposed to $O(pn^2)$. This is important since we target variables in single static assignment form[1], of which there can be hundreds or thousands, even in small kernels.

The goal of the algorithm is to identify *which* floating-point value can be represented by lower precision formats and *how much* the precisions can be reduced while meeting an output quality threshold. To find the *optimal* set of all values and all precision levels, an exhaustive search that evaluates all combinations are needed. Such an approach would have a complexity of $O(p^n)$, which is not practically viable. Instead, our approach is a heuristic algorithm which is likely to find a solution close to the optimal.

To show the benefits of the reduced-precision floats, **I** also specifies a register file capable of storing values of variable width densely. This is achieved by dividing each physical register into a number of slices, thereby making it possible to store multiple values in the same physical register. The slices are then accessed using an indirection table. In addition, to bridge the gap between the value annotations and the register file, **I** proposes a register allocation algorithm which takes the widths of the values into account.

To evaluate the contributions, five graphics kernels are compiled into a hardware-independent assembly format (LLVM IR [5]). Next, we automatically inject instructions that make it possible to dynamically set the precision of each floating-point value, and use our precision-selection algorithm to annotate

[1]Each variable is assigned exactly once.

each of the values with a precision. These precision-annotations are then fed into the proposed register allocation algorithm. Finally, the output from the register allocation algorithms shows the corresponding register pressure.

The evaluation shows that by allowing a small, but acceptable, degradation in output quality, our method can significantly reduce the number of bits needed to represent the floating-point values in the investigated kernels (between 28% and 60%). This reduces the register pressure per thread by up to 48%, 27% on average, which can enable GPUs to keep up to twice as many threads in flight simultaneously.

3 A Register File Organization to Support Variable Floating-Point Precision

While narrow floating-point values can open up many novel optimization at the hardware level, the register file is of particular interest for GPUs. This is because unlike traditional CPU architectures, which rely heavily on cache hierarchies to hide memory latency, GPUs hide latency by keeping a large number of threads in flight simultaneously, and context-switching to a new thread whenever a running thread stalls on a memory operation. The fast context switching is made possible by a large register file, which keeps the state of all active threads simultaneously. Hence, the register file size together with the thread register footprint limits the number of threads in flight. This is the reason why design trends show that GPUs rely on increasingly larger register files in order to further increase the number of threads in flight [8, 13]. However, register files of contemporary GPUs are already huge and power hungry [6].

Another way of increasing the number of threads in flight is to decrease the thread register footprint by leveraging a more efficient utilization of the register resources. Previously, register organizations for GPUs targeting integer operands have been proposed [13, 2]. However, to the best of our knowledge, none have targeted floating-point operands. Therefore, in **II**, we present a detailed microarchitectural implementation of the register file organization first mentioned in **I**.

The register file organization is centered around a configurable indirection table, which specifies the physical location of each architectural register. The indirection table is configurable because its content is kernel-specific: when a new kernel is uploaded to the GPU, also the content of the indirection table is uploaded. This way, each kernel can have a unique set of architectural registers with respect to size. In addition, since the sizes of the registers are implicitly stated in the indirection table, the instructions themselves do not need to include width information. Hence, no large modification of the ISA is needed.

Although efficient for dense operand packing, the implementation of such a register file brings a number of challenges. First, the indirection table needs to be accessed for each and every register. Since this access is on the critical path, it introduces latency which might have a negative impact on performance. Second, the indirection table has to handle multiple register accesses per cycle to match the amount of register accesses. Third, conversion between different floating-point formats has to be carried out.

To mitigate these challenges, **II** includes a detailed evaluation of timing and overhead incurred by the proposed microarchitecture techniques. This is done by modifying GPGPU-Sim [1] to include the microarchitectural changes. The evaluation shows a performance increase of up to 40%, 12% on average, when allowing for a negligible deterioration of the output quality. Furthermore, we also conduct an overhead analysis, which shows that the implementation incurs an overhead of about 30% of the transistor budget compared to the register file.

4 Concluding Remarks and Future Work

Reducing the precision of floating-point values can open up for many novel optimization approaches at the microarchitecture level. However, to enable such approaches, support is needed across many of the computer abstraction layers. This thesis presents one way of bridging the gap between the software and hardware by presenting an automated precision-selection method together with a configurable register file which can utilize narrow floats in an efficient manner. The results show that it is possible to use the output quality as a knob for controlling the number of threads in-flight in a GPU, by trading bits for efficiency.

One obvious limitation is that the evaluation in **I** only considers graphics applications. Even though the most common task for GPUs is to render images in real-time for e.g. video-games, industrial visualization and image-imaging, GPUs are increasingly being used for other tasks such as general high performance computing kernels and deep learning. The method presented in **I** is applicable to any class of algorithms, but the challenge is to establish a quantitative *quality* metric for all algorithms. This is not the same as an *error* measurement, since that does not tell anything about end users perception of the final output. This also means that the quality metric needs to be application dependent. Hence, to apply the method in **I** on more application domains, quality metrics for these domains have to be established.

Furthermore, this thesis only proposes one microarchitectural optimization, namely for the register file. Yet, with value annotations in place, a multitude of interesting optimization opportunities are possible. For example, the performance of many GPU applications is limited by memory bandwidth. At the same time, floating-point annotations for load and store instructions indicate that more data than necessary might be fetched from or written to memory. On a similar note, many GPU applications are also limited by the sizes of the L1 and L2 caches. Load- and store annotations might also open up for data compression at cache level.

Another potential future direction is to investigate the possibility to extend the framework to also include integer values. While they might not be approximable, previous authors [13, 2] have shown that many integers in GPU workloads are narrow. Such integers in combination with the precision reduction of floats would probably show even greater potential of reducing register pressure. Hence, this is a clear target for future research.

References

[1] A. Bakhoda et al. "Analyzing CUDA workloads using a detailed GPU simulator". In: *2009 IEEE International Symposium on Performance Analysis of Systems and Software.* Apr. 2009, pp. 163–174. DOI: 10.1109/ISPASS.2009.4919648.

[2] S. Z. Gilani, N. S. Kim, and M. J. Schulte. "Power-efficient computing for compute-intensive GPGPU applications". In: *High Performance Computer Architecture (HPCA2013), 2013 IEEE 19th International Symposium on.* Feb. 2013, pp. 330–341. DOI: 10.1109/HPCA.2013.6522330.

[3] A. Jain et al. "Concise loads and stores: The case for an asymmetric compute-memory architecture for approximation". In: *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO).* Oct. 2016, pp. 1–13. DOI: 10.1109/MICRO.2016.7783744.

[4] D. Jeong et al. "An eDRAM-Based Approximate Register File for GPUs". In: *IEEE Design Test* 33.1 (Feb. 2016), pp. 23–31. ISSN: 2168-2356. DOI: 10.1109/MDAT.2015.2500185.

[5] Chris Lattner and Vikram Adve. "LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation". In: *Proceedings of the International Symposium on Code Generation and Optimization: Feedback-directed and Runtime Optimization.* CGO '04. Palo Alto, California: IEEE Computer Society, 2004, pp. 75–. ISBN: 0-7695-2102-9. URL: http://dl.acm.org/citation.cfm?id=977395.977673.

[6] Jingwen Leng et al. "GPUWattch: Enabling Energy Optimizations in GPGPUs". In: *Proceedings of the 40th Annual International Symposium on Computer Architecture.* ISCA '13. Tel-Aviv, Israel: ACM, 2013, pp. 487–498. ISBN: 978-1-4503-2079-5. DOI: 10.1145/2485922.2485964. URL: http://doi.acm.org/10.1145/2485922.2485964.

[7] Sasa Misailovic et al. "Chisel: Reliability- and Accuracy-aware Optimization of Approximate Computational Kernels". In: *Proceedings of the 2014 ACM International Conference on Object Oriented Programming Systems Languages & Applications.* OOPSLA '14. Portland, Oregon, USA: ACM, 2014, pp. 309–328. ISBN: 978-1-4503-2585-1. DOI: 10.1145/2660193.2660231. URL: http://doi.acm.org.proxy.lib.chalmers.se/10.1145/2660193.2660231.

[8] S. Mittal. "A Survey of Techniques for Architecting and Managing GPU Register File". In: *IEEE Transactions on Parallel and Distributed Systems* 28.1 (Jan. 2017), pp. 16–28. ISSN: 1045-9219. DOI: 10.1109/TPDS.2016.2546249.

[9] NVIDIA. *NVIDIA Tesla P100.* White Paper. NVIDIA, 2016.

[10] Jeff Pool, Anselmo Lastra, and Montek Singh. "Precision Selection for Energy-efficient Pixel Shaders". In: *Proceedings of the ACM SIGGRAPH Symposium on High Performance Graphics.* HPG '11. Vancouver, British Columbia, Canada: ACM, 2011, pp. 159–168. ISBN: 978-1-4503-0896-0. DOI: 10.1145/2018323.2018349. URL: http://doi.acm.org/10.1145/2018323.2018349.

[11] Cindy Rubio-González et al. "Precimonious: Tuning Assistant for Floating-point Precision". In: *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. SC '13. Denver, Colorado: ACM, 2013, 27:1–27:12. ISBN: 978-1-4503-2378-9. DOI: `10.1145/2503210.2503296`. URL: `http://doi.acm.org/10.1145/2503210.2503296`.

[12] J. Y. F. Tong, D. Nagle, and R. A. Rutenbar. "Reducing power by optimizing the necessary precision/range of floating-point arithmetic". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 8.3 (June 2000), pp. 273–286. ISSN: 1063-8210. DOI: `10.1109/92.845894`.

[13] X. Wang and W. Zhang. "GPU Register Packing: Dynamically Exploiting Narrow-Width Operands to Improve Performance". In: *2017 IEEE Trustcom/BigDataSE/ICESS*. Aug. 2017, pp. 745–752. DOI: `10.1109/Trustcom/BigDataSE/ICESS.2017.308`.