Universidade Estadual de Campinas
Faculdade de Engenharia Mecânica

John Doe

Foreign Language Title

Título em Português

Campinas
2018

UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA MECÂNICA

JOHN DOE

Foreign Language Title

Título em Português

Thesis presented to the School of Mechanical Engineering of the University of Campinas in partial fulfillment of the requirements for the degree of Bachelor of Mechanical Engineering, in the field of Control Theory.

Monografia apresentada à Faculdade de Engenharia Mecânica da Universidade Estadual de Campinas como parte dos requisitos exigidos para a obtenção do título de Bacharel em Engenharia Mecânica, na área de Teoria de Controle.

Supervisor: Supervisor Full Name

ESTE EXEMPLAR CORRESPONDE À VERSÃO FINAL DA MONOGRAFIA/DISSERTAÇÃO/TESE DEFENDIDA PELO(A) ALUNO(A) JOHN DOE, E ORIENTADA PELO(A) PROF(A). DR(A). SUPERVISOR FULL NAME E CO-ORIENTADA PELO(A) PROF(A). DR(A). COSUPERVISOR FULL NAME.

CAMPINAS
2018

I'm Dedications
this is dedications
dedications

Acknowledgements

I'm Acknowledments
hello me.

Abstract

I'm Abstract
hello me.

Resumo

I'm PT Abstract
hello me.

List of Figures

List of Tables

Table of Contents

Chapter 1

Introduction

"Learn from yesterday, live for today, hope for tomorrow. The important thing is not to stop questioning."

— Albert Einstein

Micro-optimization is a technique to reduce the overall operation count of floating point operations. In a standard floating point unit, floating point operations are fairly high level, such as "multiply" and "add"; in a micro floating point unit (μFPU), these have been broken down into their constituent low-level floating point operations on the mantissas and exponents of the floating point numbers.

Chapter two describes the architecture of the μFPU unit, and the motivations for the design decisions made.

Chapter three describes the design of the compiler, as well as how the optimizations discussed in section were implemented.

Chapter four describes the purpose of test code that was compiled, and which statistics were gathered by running it through the simulator. The purpose is to measure what effect the micro-optimizations had, compared to unoptimized code. Possible future expansions to the project are also discussed.

1.1 Motivations for micro-optimization

The idea of micro-optimization is motivated by the recent trends in computer architecture towards low-level parallelism and small, pipelineable instruction sets [misc-full,inbook-full 1, 2]. By getting rid of more complex instructions and concentrating on optimizing frequently used instructions, substantial increases in performance were realized.

Another important motivation was the trend towards placing more of the burden of performance on the compiler. Many of the new architectures depend on an intelligent, optimizing compiler in order to realize anywhere near their peak performance [article-full,mastersthesis-full,unpublished-full 3, 4, 5]. In these cases, the compiler not only is responsible for faithfully generating native code to match the source language, but also must be aware of instruction latencies, delayed branches, pipeline stages, and a multitude of other factors in order to generate fast code [article-full 3].

Taking these ideas one step further, it seems that the floating point operations that are normally single, large instructions can be further broken down into smaller, simpler, faster instructions, with more control in the compiler and less in the hardware, see Figure [ch1:figure1 1.1]. This is the idea behind a micro-optimizing FPU; break the floating point instructions down into their basic components and use a small, fast implementation, with a large part of the burden of hardware allocation and optimization shifted towards compile-time.

Along with the hardware speedups possible by using a μFPU, there are also optimizations that the compiler can perform on the code that is generated. In a normal sequence of floating point operations, there

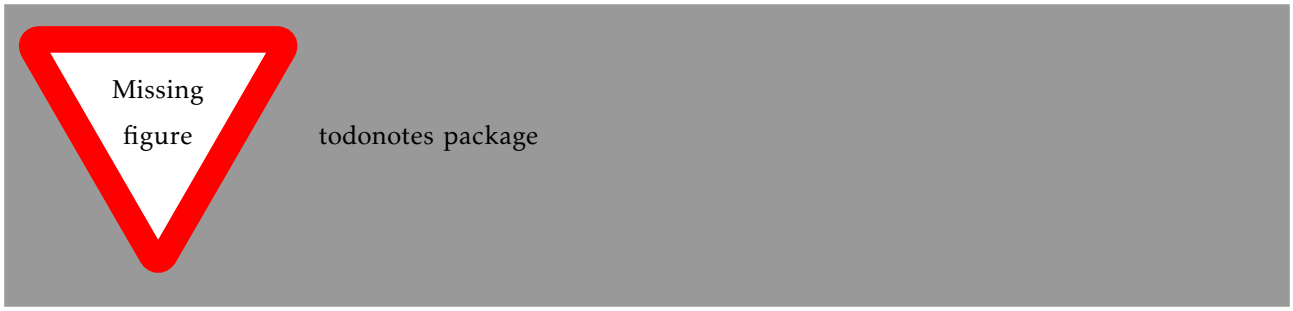

Figure 1.1: I'm a missing figure.

ch1:figur

are many hidden redundancies that can be eliminated by allowing the compiler to control the floating point operations down to their lowest level.

These optimizations are described in detail in section 1.2.

ch1:opts

1.2 Description of micro-optimization

ch1:opts

In order to perform a sequence of floating point operations, a normal FPU performs many redundant internal shifts and normalizations in the process of performing a sequence of operations. However, if a compiler can decompose the floating point operations it needs down to the lowest level, it then can optimize away many of these redundant operations [6].

inproceedings-full

If there is some additional hardware support specifically for micro-optimization, there are additional optimizations that can be performed. This hardware support entails extra "guard bits" on the standard floating point formats, to allow several unnormalized operations to be performed in a row without the loss information[1]. A discussion of the mathematics behind unnormalized arithmetic is in appendix.

The optimizations that the compiler can perform fall into several categories:

This is a to-do list...

1.2.1 Post Multiply Normalization

When more than two multiplications are performed in a row, the intermediate normalization of the results between multiplications can be eliminated. This is because with each multiplication, the mantissa can become denormalized by at most one bit. If there are guard bits on the mantissas to prevent bits from "falling off" the end during multiplications, the normalization can be postponed until after a sequence of several multiplies. Using unnormalized numbers for math is not a new idea; a good example of it is the Control Data CDC 6600, designed by Seymour Cray. [7, 1] The CDC 6600 had all of its instructions performing unnormalized arithmetic, with a separate `NORMALIZE` instruction.

techreport-full,misc-full

As you can see, the intermediate results can be multiplied together, with no need for intermediate normalizations due to the guard bit. It is only at the end of the operation that the normalization must be performed, in order to get it into a format suitable for storing in memory [8].

proceedings-full

[1] A description of the floating point format used is shown in figures and.

table1

Table head	Table head
Some values	Some values
Some values	Some values
Some values	Some values
Some values	Some values
Some values	Some values
Some values	Some values

Table 1.1: I'm a table.

1.2.1.1 Block Exponent

In a unoptimized sequence of additions, the sequence of operations is as follows for each pair of numbers (m_1,e_1) and (m_2,e_2).

1. Compare e_1 and e_2.
2. Shift the mantissa associated with the smaller exponent $|e_1 - e_2|$ places to the right.
3. Add m_1 and m_2.
4. Find the first one in the resulting mantissa.
5. Shift the resulting mantissa so that normalized
6. Adjust the exponent accordingly.

Out of 6 steps, only one is the actual addition, and the rest are involved in aligning the mantissas prior to the add, and then normalizing the result afterward. In the block exponent optimization, the largest mantissa is found to start with, and all the mantissa's shifted before any additions take place. Once the mantissas have been shifted, the additions can take place one after another. This requires that for n consecutive additions, there are $\log_2 n$ high guard bits to prevent overflow. In the μFPU, there are 3 guard bits, making up to 8 consecutive additions possible. An example of the Block Exponent optimization on the expression X = A + B + C is given in figure.

References

misc-full [1] J.-B. Missilany, "Handing out random pamphlets in airports." Handed out at O'Hare, Oct. 1984. This is a full MISC entry.

book-full [2] D. E. Knuth, *Fundamental Algorithms*, vol. 1 of *The Art of Computer Programming*, section 1.2, pp. 10–119. Reading, Massachusetts: Addison-Wesley, second ed., 10 Jan. 1973. This is a full INBOOK entry.

icle-full [3] L. A. Aamport, "The gnats and gnus document preparation system," *G-Animal's Journal*, vol. 41, pp. 73+, July 1986. This is a full ARTICLE entry.

esis-full [4] É. Masterly, "Mastering thesis writing," master's project, Stanford University, English Department, June-Aug. 1988. This is a full MASTERSTHESIS entry.

shed-full [5] U. Ünderwood, N. Ñet, and P. P̄ot, "Lower bounds for wishful research results." Talk at Fanstord University (this is a full UNPUBLISHED entry), Nov., Dec. 1988.

ings-full [6] A. V. Oaho, J. D. Ullman, and M. Yannakakis, "On notions of information transfer in VLSI circuits," in *Proc. Fifteenth Annual ACM Symposium on the Theory of Computing* (W. V. Oz and M. Yannakakis, eds.), no. 17 in All ACM Conferences, (Boston), pp. 133–139, The OX Association for Computing Machinery, Academic Press, Mar. 1983. This is a full INPROCEDINGS entry.

port-full [7] T. Térrific, "An $O(n \log n / \log \log n)$ sorting algorithm," Wishful Research Result 7, Fanstord University, Computer Science Department, Fanstord, California, Oct. 1988. This is a full TECHREPORT entry.

ings-full [8] W. V. Oz and M. Yannakakis, eds., *Proc. Fifteenth Annual Symposium on the Theory of Computing*, no. 17 in All ACM Conferences, (Boston), The OX Association for Computing Machinery, Academic Press, Mar. 1983. This is a full PROCEEDINGS entry.