Improving Performance of Software Implemented Floating Point Addition

Hindborg, Andreas Erik; Karlsson, Sven

Published in:
Proceedings of the Fourth Swedish Workshop on Multicore Computing

Publication date:
2011

Improving Performance of Software Implemented Floating Point Addition

Andreas Erik Hindborg
DTU Informatics
Technical University of Denmark
s062068@student.dtu.dk

Sven Karlsson
DTU Informatics
Technical University of Denmark
ska@imm.dtu.dk

ABSTRACT

We outline and evaluate hardware extensions to an integer processor pipeline which allow IEEE 754 floating point, *FP*, addition to be efficiently implemented in software. With a very moderate increase in hardware resources, our performance evaluation shows that, for a benchmark that executes 12.5% FP addition instructions, our approach exhibits a relative slowdown of 3.38 to 15.15 as compared to dedicated hardware. This is a significant improvement of pure software emulation which leads to relative slowdowns up to 45.33.

1. APPROACH

We propose twelve special instructions to efficiently implement FP addition in software. The instructions, when executed in sequence, will realize FP addition. The instructions can be implemented by reusing many of the logic blocks found in modern processor cores. Therefore we estimate that the additional logic needed to implement the instructions is relatively low compared to the amount of logic required to implement a dedicated FP addition pipeline.

2. EXPERIMENTAL RESULTS

We evaluate our approach using the cycle accurate SimpleScalar ARM [1] `sim-outorder` simulator.

We establish a baseline for our results by simulating a micro-benchmark that measures the relative slowdown when using soft FP addition over using hard FP addition. The benchmark is compiled using GCC 4.6.0 with the configuration flags `-msoft-float` and `-mhard-float` respectively. The pure software version of the benchmark shows a relative slowdown of 45.33 compared to the hardware FP addition version. This is in line with the results produced by Rodolfo et. al [2]. They show a relative slowdown of 22 on real applications when using a software FP implementation over a hardware FP implementation.

To measure the effect of the proposed method, we execute the SPEC2006 benchmark `469.lbm` on various simulated architectures. Highlights of the results are shown in table 1. Each row shows the relative slowdown of using the proposed method over using a conventional hardware implementation. The row labeled "Ideal memory subsystem" uses one in-order integer pipeline, *IP*, with an ideal memory subsystem. The row labeled "Realistic system" uses one out-of-order , *OoO*, IP with a realistic memory subsystem and a dedicated address calculation unit. The row labeled "Best" uses 4 OoO IPs and a dedicated addressing unit.

Configuration	Slowdown
Ideal memory subsystem	9.22
Realistic system	12.52
Best	3.38

Table 1: Select simulation results.

Our measurements show that going from an in-order pipeline to an OoO pipeline has a large impact on the effect of the proposed method. This is reflected in the difference from "Ideal memory subsystem" to "Realistic system". When using an OoO pipeline, there is more instruction level parallelism, *ILP*, available to the instruction scheduler when using the FPU for additions; the additions take more time when using the integer pipeline and the occupancy of the integer pipeline increases. Instructions that need the integer pipeline will be blocked for a longer time, thus blocking depending instructions and decreasing the available ILP. With a decreased level of available ILP, the instruction scheduler will be less efficient in hiding latencies in the pipeline.

Using an OoO pipeline with four IPs that are all extended to accelerate FP additions show the best results. The reason for the good results in this situation is the fact that more instructions can execute simultaneously and more ILP is available to the instruction scheduler.

3. CONCLUSIONS AND FUTURE WORK

Our proposed system allow for efficient software implementation of floating point addition arithmetic. We have executed `470.lbm` from SPEC2006 using the cycle accurate SimpleScalar ARM simulator. We show that our approach exhibit a relative slowdown of 3.38 to 15.15 compared to a pipelined hardware implementation of floating point arithmetic. This is significantly faster than a pure software approach which causes relative slowdowns of up to 45.33.

4. REFERENCES

[1] D. Burger and T. M. Austin. The simplescalar tool set, version 2.0. *SIGARCH Computer Architecture News*, 25:13–25, June 1997.

[2] T. A. Rodolfo, N. L. V. Calazans, and F. G. Moraes. Floating point hardware for embedded processors in fpgas: Design space exploration for performance and area. In *Proceddings of ReConFig'09 - 2009 International Conference on ReConFigurable Computing and FPGAs*, pages 24–29, 2009.