IMPROVING THE THROUGHPUT OF AN MT PROCESSOR

R. B. García, J. R. Ardenghi, J. Echaiz

Laboratorio de Investigación de Sistemas Distribuidos LISiDi
Dpto. de Ciencias de la Computación
Universidad Nacional del Sur
E-mail: {rbg, jra, je}@cs.uns.edu.ar

1 Introduction

Multithreading (MT), by simultaneously using both the thread-level parallelism and the instruction-level parallelism of a workload, exploits the available resources more efficiently than single-thread processors allowing a better throughput, i.e there are more instructions per cycle (IPC), over the single thread approach. An MT approach is a chip multiprocessor (CMP) [1], which is a static one that could exploit a moderate amount of the ILP on a fixed number of threads. The other one, simultaneous multithreading (SMT) [2], uses dynamic mechanisms and policies to exploit the available ILP of a varying number of threads. SMT using both TLP and ILP interchangeably will provide larger IPC rates than CMP. However, due to its complex and tightly coupled microarchitecture, SMT increases the pressure on cycle time.

A previous work of Agarwal et al. [4] shows that in the future technologies, as wire delays grow relative to gate delays, the improvement in clock rate and IPC becomes directly antagonistic. With conventional microarchitectures, designers are faced with a difficult choice: increase the clock rate aggressively at the cost of reducing IPC, or mitigate the decline in IPC by slowing the rate of the clock speed growth. Compared to de SMT solution, CMP accomplishes a capacity scaling which can reduce the impact of higher clock frequencies in the IPC (less deeper pipelines).

We assume an MT processor composed of two engines: the MT front-end unit and the MT execution core. The front-end fetches, decodes and renames instructions from several threads and puts them into one or several instruction queues. The execution core executes those instructions, retires non-speculative results, and sends a signal to the front-end each time a branch missprediction is detected. Adopting a producer-consumer relationship between the two engines it is possible to study how the front-end's instruction delivery limits execution performance and also it permits roughly independent designs of these units.

The goal of our design is an MT front-end unit that maximizes, for a fixed number of fetch resources, the overall instruction bandwidth of an execution workload with varying number of threads. In a previous work [3] we have measured the IPC rate for two "pure" approaches: CMP-based and SMT-based, considering their implementation complexities. The requirement for MT front-end designs 2was not to increase the cycle time and not to consume too much additional chip area compared to a single-thread front-end that was used for reference (ALPHA 21264) in order to preserve single thread performance.

As we notice in this research, the inter-thread instruction selection is orthogonal to share or distributed fetch storage. In other words, the instruction select mechanism only requires widening the per-thread fetch width, regardless of the fetch storage organization. The results reveal that per-thread fetch buffers coupled with per-thread instruction selection are essential, while decreasing the per-

thread decode/rename width alleviate cycle time pressure with little impact in single- and multi-thread performance.

The results of simulation, using workloads with different number of threads, identify the lost cycles in an MT processor. In case of a few threads, one or two, block fragmentation and misspredicted branches are fundamentals in the increase of the CPI. With an increased number of threads, block fragmentation is attenuated, but it remains the penalty of the misspredicted branch hazards.

2 Objectives

We are going to investigate suitable solutions to the remaining cycles lost in an MT processor. We know that current branch prediction schemes are based on recognizing branching patterns, and are extremely effective for some branches. However other branches will never fit into this category because they are data-dependent on relatively random data. For those branches prediction is not sufficient despite the use of an elaborate predictor. In case of having only one thread the problem is that we can not hide the block fragmentation, like the case of a workload with several threads. Also, we have plenty of unused resources.

We consider exploring several options to improve the throughput of an MT processor in two different scenarios:

A workload with many threads

- Using some kind of branch predictor confidence [8] it is feasible to select the more confident threads in order to improve the utilization of the fetch resources and reduce the lost CPU cycles.

A workload with a few threads

- Speculatively execute multiple paths of execution exploiting the existing resources, one primary path (the predicted one) and forked alternative paths [5, 6]. If there is significant progress down the correct path when the branch is resolved, we can eliminate the misspredict penalty.

- We are going to explore solutions to the problem of block fragmentation: managing multiple, non-sequential instructions in a single cycle [7]. The idea is to use the idle fetch clusters to increase the effective, per-thread, fetch width.

3 Future work

We are going to evaluate through simulation the possibility of improving the CPI comparing a baseline of a standard selection of the instruction threads (ICOUNT) against the ideal case of a perfect confidence estimator for speculation control. We are going to simulate different conditions of execution analyzing the nature of the lost cycles. Those results allow the evaluation of the potential in performance improvement with our methods of confidence estimation.

We have studied different alternatives of confidence evaluation of the bibliography, they will be simulated and compared against our own confidence branch prediction estimation.

Bibliography

1. L. Hammond, B. A. Nayfeh, K. Olukotun,. “A Single-Chip Multiprocessor”, *IEEE Computer*, 1997.

2. D. M. Tulsen, S. J. Eggers, H. M. Levy. “Simultaneous Multithreading: Maximizing On-Chip Parallelism”, *Proc. of the 22nd Int. Symp. on Microarchitecture 1999.*

3. J. C. Moure, D: J. Rexachs, E. Luque. “Fetch Unit Design for Multithreaded Processors”

4. V. Agarwal, M. S. Hrishikesh, S. W. Keckler, D. Burger. “Clock Rate versus IPC: The End of the Road for Conventional Microarchitectures”, *ISCA ´27 Proc. 2000*.

5. P. L. Ahuja, K. Skadron, M. Martonosi, D. W. Clark. “Multipath Execution: Opportunities and Limits”.

6. S. Wallace, B. Calder, D. M. Tullsen. “Threaded Multiple Path Execution”, *Proc. ISCA 25*, 1998.

7. T. M. Conte, K. N. Menezes, P. M. Mills, B. A. Patel. “Optimization of Instruction Fetch Mechanisms for High Issue Rates”, *ISCA ´95*.

8. E. Jacobsen, E. Rotenberg, J. E. Smith, “Assigning Confidence to Conditional Branch Predictions”, *Proc. of the 29th Int. Symp. on Computer Architecture 1998*.

9. J.C. Moure, D. I. Rexachs, E. Luque, R. B. García, “Scalable Simultaneous Multithreading (ScSMT)”, *PARCO 1999.*