Simultaneous Multithreading (SMT) Processors

Jennifer M. Jayme
Department of Electrical and Electronics Engineering
University of the Philippines
Diliman, Quezon City
Philippines

Abstract

Different Simultaneous Multithreading (SMT) architectures have been proposed and implemented in the industry. Though, as powerful as it sounds over a superscalar processor, design issues are not rare. Addressing a certain problem can be done in hardware or in software; depending on what SMT architecture will you apply it to. Two papers are presented here. The first one, implemented in hardware a solution for wasted resources, heterogeneously clustered SMT architecture. The Heterogeneously Distributed SMT (hdSMT) architecture maximizes the hardware budget by taking into account the heterogeneity of applications. The second paper proposed an Implicitly-Multithreaded (IMT) processor utilizing SMT's support for multithreading by executing speculative threads. It relied mostly on the compiler to select suitable thread spawning points and orchestrate inter-thread register communication.

I. INTRODUCTION

One problem for multithreaded microprocessors is that they have a very poor instruction level parallelism. Some issue slots in an execution sequence for a given cycle can be used, but not all. However, they have the advantage of better tolerance for long-latency operations, thereby eliminating a completely unused cycle in an execution sequence. One solution is to implement simultaneous multithreading which has features of a multithreaded processor with the ability to issue multiple instructions per cycle. [1]

A lot of SMT architectures have been proposed and used in the industry. An SMT architecture in which the hardware is heterogeneously clustered in order to reduce the amount of wasted resources is one implementation. It showed better performance over monolithic SMT and homogeneously clustered SMT. Another modified SMT architecture, the Implicitly Multithreaded architecture used the power of a compiler to speculate threads to be executed.

POWER5™ is an example of an SMT processor [3] which is currently in production.

II. RELATED WORK

1. The hdSMT Architecture [2]

The foundations of the hdSMT architecture are comprised of a threefold combination of well known principles and techniques: *SMT*, *clustering*, and *heterogeneity-awareness*. An hdSMT processor proposes a multithreaded alternative that lays on the spectrum that extends in between SMT and CMP processors. There is multiple possible hardware configurations in between SMT and CMP processors, as we vary the amount of resources shared among the execution cores. However, the heterogeneity in applications' behavior makes vary the hardware requirements among different applications. To better profit from the available hardware it should be heterogeneously clustered and the applications appropriately matched with the clusters according to their needs. The hdSMT architecture maximizes the available hardware budget by taking into account the heterogeneity in this way. The hdSMT architecture overview is depicted in Fig. 1. As in a conventional SMT processor, all threads share the caches, register file, and fetch engine. However, the rest of the pipeline stages and resources are arranged in heterogeneous clusters (or pipelines). So, each pipeline comprises all the pipeline stages

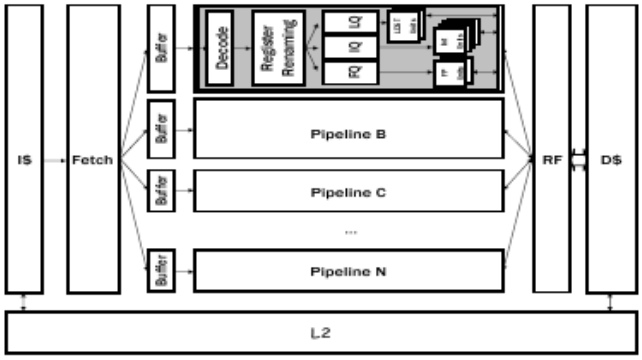


Fig.1. The hdSMT architecture

of the conventional processor but the fetch stage. Each pipeline also has got its own private instruction queues, renaming map tables and functional units. The size and number of these resources may vary from pipeline to pipeline. Additionally, each thread's instructions are stored in a private reorder buffer (ROB), one per thread. In this clustered multithreaded architecture, entire threads are assigned to pipelines according to heterogeneity. This implies that there are no dependencies between instructions in different clusters, since all instructions from a single thread are mapped to the same pipeline. The heterogeneity aware fetch engine strives to match both the needs of each running application and the interaction among each application with the heterogeneously distributed hardware. This software-hardware mapping is performed each time the job scheduler of the operating system selects a new bunch of active threads. The whole subsequent execution of the workload is done according to this mapping. The number of hardware contexts and width of each pipeline may vary from pipeline to pipeline.

So, hdSMT microarchitecture may be comprised of both narrow single-threaded and wide-multithreaded pipelines, as well intermediate pipelines. Depending on the resource needs of each application and the interaction between application behaviors, more than one application may be mapped to a single pipeline. This distribution of the hardware contexts along the chip can be profited to turn off idle pipelines whenever the number of running applications does not reach the number of hardware contexts.

Notice that multipipeline-awareness in hdSMT uncovers new fetch policies not available in conventional SMT processors. The shared fetch engine is limited by the number and width of the instruction cache ports. However, the number of instructions that each pipeline accept per cycle may vary from pipeline to pipeline. In order to decouple the fetch engine from the characteristics of each specific pipeline it feeds, some small buffers are added before each pipeline. Thus, the fetch engine inserts in-order the fetched instructions at its own rate while each pipeline extracts in-order instructions according to its width. The fetch policy takes into account these buffers in order to appropriately balance the instructions fetched among the pipelines. Depending on the pipeline set characteristics, this may result in a wider global decode bandwidth since all pipelines are fed from their private buffer each cycle.

2. Implicitly-Multithreaded Processors [4]

An Implicitly-MultiThreaded (IMT) processor utilizes SMT's support for multithreading by executing speculative threads. Fig. 2 depicts the anatomy of an IMT processor derived from SMT. IMT uses the rename tables for register renaming, the issue queue for out-of-order scheduling, the per-context load/store queue (LSQ) and active list for memory dependences and instruction reordering prior to commit. As in SMT, IMT shares the functional units, physical registers, issue queue, and memory hierarchy among all contexts. IMT exploits *implicit* parallelism, as opposed to programmer-specified, *explicit* parallelism exploited by conventional SMT and multiprocessors. Like Multiscalar, IMT predicts the threads in succession and maps them to execution resources, with the earliest thread as the *nonspeculative* (head) thread, followed by subsequent *speculative* threads. IMT honors the inter-thread control flow and register dependences specified by the compiler. IMT uses the LSQ to enforce inter-thread memory dependences. Upon completion, IMT commits the threads in program order. There are two IMT variations: (1) a *Naive IMT (NIMT)* that performs comparably to an aggressive superscalar, and (2) an *Optimized IMT (O-IMT)* that uses novel microarchitectural techniques to enhance performance.

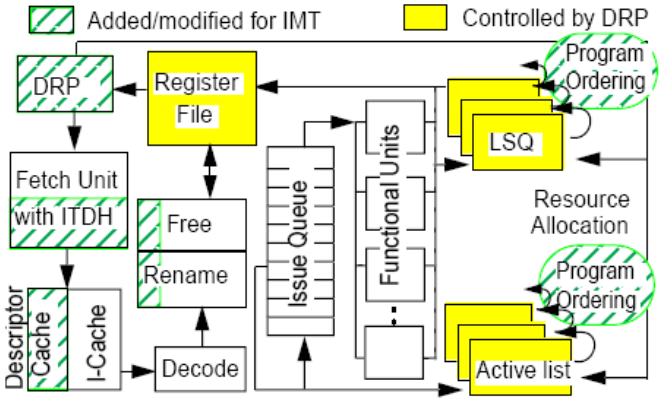
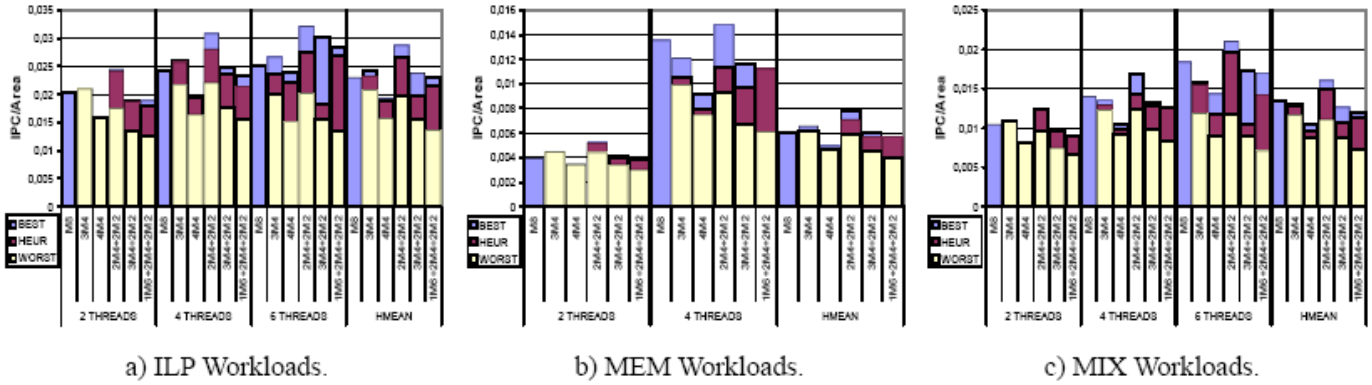


Fig. 2. The Anatomy of an IMT processor



a) ILP Workloads. b) MEM Workloads. c) MIX Workloads.

Fig. 3. Performance per Area Comparison

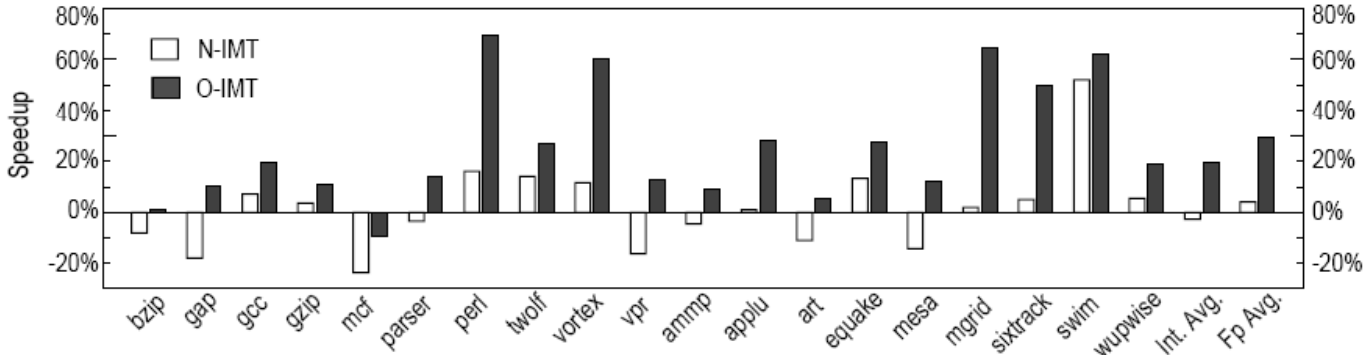


Fig. 4. Performance Comparison of N-IMT and O-IMT Normalized to the Baseline Superscalar

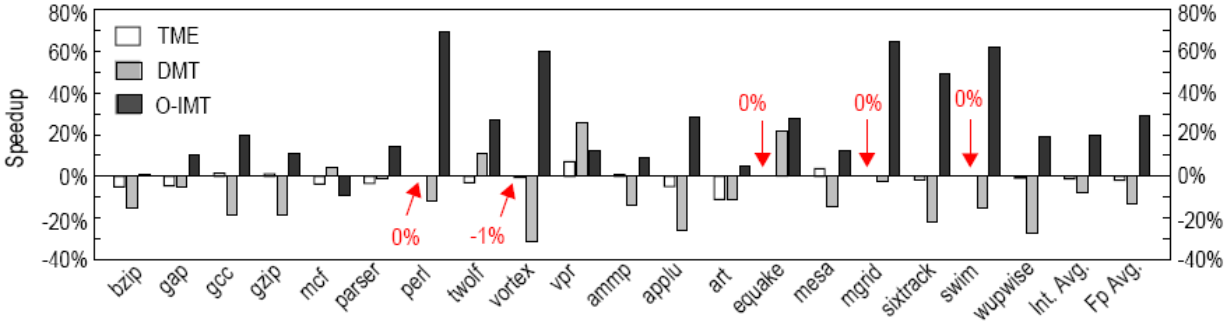


Fig. 5. Performance Comparison of TME, DMT, and IMT Normalized to Baseline Superscalar

III. COMPARISONS

The hdSMT proposed design implementation and optimization of SMT in hardware while the IMT done it in software. hdSMT compared performance using benchmarks with high instruction-level parallelism (ILP), with bad memory behavior (MEM), or a mix of both (MIX). Fig. 3 shows the raw performance results (measured in IPC) for all the microarchitectures evaluated in their study. In each case, the harmonic mean of all workloads of a same type and size is shown. These results point out that, although some hdSMT results are quite similar to SMT baseline ones, the hdSMT results are exceeded by the SMT baseline ones in some cases. As can also be observed from the figure, we can infer that the hdSMT architecture achieves higher performance per area ratios than the monolithic SMT architecture, that is, better relative results than SMT using fewer resources.

Meanwhile, IMT compared a Naïve-IMT with an Optimized-IMT as well as its previous studies of Threaded Multipath Execution (TME) and Dynamically Multi-Threaded (DMT) processors. Figure 4 indicates that N-IMT's performance is actually inferior to superscalar for integer benchmarks. N-IMT reduces performance in integer benchmarks by as much as 24% and on average by 3% as compared to superscalar. Moreover, while the results for floating-point benchmarks vary, on average NIMT only improves performance slightly over superscalar for these benchmarks. The figure also indicates that microarchitectural optimizations substantially benefit compiler-specified threading, enabling O-IMT to improve performance over superscalar by as much as 69% and 65% and on average 20% and 29% for integer and floating point benchmarks respectively. Figure 5 compares speedups for the optimized TME and DMT machines, against O-IMT normalized to the baseline superscalar. Unlike O-IMT, TME and DMT reduce performance on average with respect to a comparable superscalar. TME primarily exploits thread-level parallelism across unpredictable branches. Because unpredictable branches are not common, TME's opportunity for improving performance by exploiting parallelism across multiple paths is limited. TME's eagerness to invoke threads on unpredictable branches also relies on the extent to which a confidence predictor can identify unpredictable branches. A confidence predictor with low accuracy would often spawn threads on both paths, often taking away fetch bandwidth from the correct (and potentially predictable) path. An accurate confidence predictor would result in a TME machine that performs close to, or improves performance slightly over, the baseline superscalar machine.

REFERENCES

[1] S. Eggers, et. al. *Simultaneous Multithreading: A Platform for Next-Generation Processors*. IEEE Micro. September-October, 1997.

[2] C. Acosta, et. al. *A Complexity-Effective Simultaneous Multithreading Architecture*. IEEE paper.

[3] J. Clabes, et. al. *Design and Implementation of the POWER5™ Microprocessor*. 2004 IEEE International Solid-State Circuits Conference. February 16, 2004.

[4] I. Park, B. Falsafi, T. Vijaykumar. *Implicitly-Multithreaded Processors*. IEEE paper.