Converting Thread-Level Parallelism to Instruction-Level Parallelism via Simultaneous Multithreading

JACK L. LO and SUSAN J. EGGERS
University of Washington
JOEL S. EMER
Digital Equipment Corporation
HENRY M. LEVY
University of Washington
REBECCA L. STAMM
Digital Equipment Corporation
and
DEAN M. TULLSEN
University of California, San Diego

To achieve high performance, contemporary computer systems rely on two forms of parallelism: instruction-level parallelism (ILP) and thread-level parallelism (TLP). Wide-issue superscalar processors exploit ILP by executing multiple instructions from a single program in a single cycle. Multiprocessors (MP) exploit TLP by executing different threads in parallel on different processors. Unfortunately, both parallel processing styles statically partition processor resources, thus preventing them from adapting to dynamically changing levels of ILP and TLP in a program. With insufficient TLP, processors in an MP will be idle; with insufficient ILP, multiple-issue hardware on a superscalar is wasted. This article explores parallel processing on an alternative architecture, simultaneous multithreading (SMT), which allows multiple threads to compete for and share all of the processor's resources every cycle. The most compelling reason for running parallel applications on an SMT processor is its ability to use thread-level parallelism and instruction-level parallelism interchangeably. By permitting

This research was supported by Digital Equipment Corporation, the Washington Technology Center, NSF PYI Award MIP-9058439, NSF grants MIP-9632977, CCR-9200832, and CCR-9632769, DARPA grant F30602-97-2-0226, ONR grants N00014-92-J-1395 and N00014-94-1-1136, and fellowships from Intel and the Computer Measurement Group.
Authors' addresses: J. L. Lo, S. J. Eggers, and H. M. Levy, Department of Computer Science and Engineering, University of Washington, Box 352350, Seattle, WA 98195-2350; email: {jlo; eggers; levy}@cs.washington.edu; J. S. Emer and R. L. Stamm, Digital Equipment Corporation, HL02-3/J3, 77 Reed Road, Hudson, MA 07149; email: {emer; stamm}@vssad.enet.dec.com; D. M. Tullsen, Department of Computer Science and Engineering, University of California, San Diego, 9500 Gilman Drive, La Jolla, CA 92093-0114; email: tullsen@cs. ucsd.edu.
Permission to make digital/hard copy of part or all of this work for personal or classroom use is granted without fee provided that the copies are not made or distributed for profit or commercial advantage, the copyright notice, the title of the publication, and its date appear, and notice is given that copying is by permission of the ACM, Inc. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or a fee.
© 1997 ACM 0734-2071/97/0800–0322 $03.50

multiple threads to share the processor's functional units simultaneously, the processor can use both ILP and TLP to accommodate variations in parallelism. When a program has only a single thread, all of the SMT processor's resources can be dedicated to that thread; when more TLP exists, this parallelism can compensate for a lack of per-thread ILP. We examine two alternative on-chip parallel architectures for the next generation of processors. We compare SMT and small-scale, on-chip multiprocessors in their ability to exploit both ILP and TLP. First, we identify the hardware bottlenecks that prevent multiprocessors from effectively exploiting ILP. Then, we show that because of its dynamic resource sharing, SMT avoids these inefficiencies and benefits from being able to run more threads on a single processor. The use of TLP is especially advantageous when per-thread ILP is limited. The ease of adding additional thread contexts on an SMT (relative to adding additional processors on an MP) allows simultaneous multithreading to expose more parallelism, further increasing functional unit utilization and attaining a 52% average speedup (versus a four-processor, single-chip multiprocessor with comparable execution resources). This study also addresses an often-cited concern regarding the use of thread-level parallelism or multithreading: interference in the memory system and branch prediction hardware. We find that multiple threads cause interthread interference in the caches and place greater demands on the memory system, thus increasing average memory latencies. By exploiting thread-level parallelism, however, SMT hides these additional latencies, so that they only have a small impact on total program performance. We also find that for parallel applications, the additional threads have minimal effects on branch prediction.

Categories and Subject Descriptors: C.1.2 [**Processor Architectures**]: Multiple Data Stream Architectures

General Terms: Measurement, Performance

Additional Key Words and Phrases: Cache interference, instruction-level parallelism, multiprocessors, multithreading, simultaneous multithreading, thread-level parallelism

1. INTRODUCTION

To achieve high performance, contemporary computer systems rely on two forms of parallelism: instruction-level parallelism (ILP) and thread-level parallelism (TLP). Although they correspond to different granularities of parallelism, ILP and TLP are fundamentally identical: they both identify independent instructions that can execute in parallel and therefore can utilize parallel hardware. Wide-issue superscalar processors exploit ILP by executing multiple instructions from a single program in a single cycle. Multiprocessors exploit TLP by executing different threads in parallel on different processors. Unfortunately, neither parallel processing style is capable of adapting to dynamically changing levels of ILP and TLP, because the hardware enforces the distinction between the two types of parallelism. A multiprocessor must statically partition its resources among the multiple CPUs (see Figure 1); if insufficient TLP is available, some of the processors will be idle. A superscalar executes only a single thread; if insufficient ILP exists, much of that processor's multiple-issue hardware will be wasted.

Simultaneous multithreading (SMT) [Tullsen et al. 1995; 1996; Gulati et al. 1996; Hirata et al. 1992] allows multiple threads to compete for and share available processor resources every cycle. One of its key advantages