RETROSPECTIVE:

Simultaneous Multithreading: Maximizing On-Chip Parallelism

Dean M. Tullsen, Susan J. Eggers, and Henry M. Levy

Department of Computer Science and Engineering
University of Washington
{eggers,levy}@cs.washington.edu
tullsen@ucsd.edu

This paper was published in 1995 and so it seems early to do a retrospective; in fact, research in simultaneous multithreading (SMT) is still ongoing. The original project began in early 1994. We were beginning to see commercial microprocessors that could issue many instructions per cycle (wide superscalars), but which rarely did so due to dependencies and long memory latencies. In fact, processor utilization seemed to be declining as fast as instruction issue width was increasing.

We actually began by looking at targeted solutions to the low processor utilization problem, such as improved branch prediction, but quickly realized that no single such mechanism was likely to solve the overall problem we faced. The graph in the paper attributing the many causes of lost cycles was one key to our intuition, and made us realize that we needed a more global, latency-tolerant solution. This led us to the basic idea of using a much finer-grained multithreading than had been previously attempted as a general way to tolerate *all* forms of lost utilization. The idea was surely influenced by previous designs such as the Tera, MIT Alewife, and M-machine projects, and by Radhika Thekkath's UW thesis. Several other projects had also looked at various forms of multithread, superscalar issue. However, as we examined them, each of these studies seemed to be limited in some way by the constraints of a particular hardware architecture in which it was embedded. None of the previous projects, to our mind, had really explored or analyzed the total potential of the fully-general concept we were considering, nor had they described it in the way we were thinking about it. We chose the name "simultaneous multithreading" to give this general concept a new label.

The more we thought about it, though, the more we realized that simultaneous multithreading (SMT) was significantly different from the traditional (context-switching) multithreading designs. In particular, there was something aesthetically pleasing about the concept of sharing *all* processor resources *every* cycle: basically, just throw all the threads in the machine, and let it make the best dynamic decision about what instructions to send to what functional·units at every instant. The result of this was effectively to use thread-level parallelism to make up for a lack of instruction-level parallelism in individual programs. This is a somewhat different goal than that addressed by previous multithreaded designs. If you have one or a few threads with moderate ILP each, that's fine; many threads with a little ILP each, that's fine; multiprogramming, that's fine. The figure that appears in the paper, showing the effect of "horizontal waste" and "vertical waste," was also a useful tool for us in understanding and explaining why SMT was likely to work better than the alternative schemes (superscalar and traditional multithreading, and later single-chip multiprocessors).

There were a few major goals of the work from the start, both arising from our desire to target mainstream processor designs. First, we were very aware that poor single-thread performance would not be acceptable in this market (as opposed to the types of uses the Tera is targeting, for example); therefore, it was crucial that single-thread performance not be harmed by the addition of SMT. Second, we wanted SMT to be easily implementable on state-of-the-art microprocessors. The "state-of-the-art" that eventually facilitated meeting this goal was dynamic instruction issue (i.e., out-of-order processors); in fact, we were a little ahead of this at the time, which caused many people to doubt that SMT was achievable. Following this paper, we were lucky to work with colleagues Joel

Emer and Rebecca Stamm from Digital's Alpha group, who greatly contributed to the microarchitecture design and helped us to show how SMT required only limited changes to an out-of-order processor; we also discovered how SMT performance could be improved significantly by fetching from the "right" threads, i.e., those making best use of the processor. By the time our second paper was published at the following ISCA (1996), many people saw the appearance of out-of-order machines and realized that once you have dynamic instruction issue, you've already provided most of the complexity with respect to the instruction issue mechanism required by SMT. It was quite interesting (and exciting) in retrospect to see a major change in response to the idea of SMT that occurred over the period of less than one year.

Acknowledgments

We would like to thank Digital Equipment Corporation and Equator Technologies Inc. for access to the Multiflow compiler. This work was supported by the National Science Foundation Grant No. MIP-9632977.

About the authors

Susan Eggers is Associate Professor of Computer Science and Engineering at University of Washington. Her research includes computer systems architecture, machine-dependent compiler optimizations, and dynamic compilation techniques.

Henry Levy is Professor of Computer Science and Engineering at the University of Washington. His research focuses on operating system design, computer architecture, and their interaction.

Dean Tullsen finished his PhD at University of Washington on the topic of Simultaneous Multithreading. He is currently Assistant Professor of Computer Science at University of California, San Diego, where he works on architecture and simultaneous multithreading.