RETROSPECTIVE: Exploiting Choice: Instruction Fetch and Issue on an Implementable Simultaneous Multithreading Processor

Dean M. Tullsen[*], Susan J. Eggers[†], Joel S. Emer[§,^], Henry M. Levy[†,‡], Jack L. Lo[‡], and Rebecca Stamm Fox[**]

[*]University of California, San Diego
[†]University of Washington
[§]Massachusetts Institute of Technology
[^]Nvidia
[‡]Google
[**]Retired from Digital/Compaq

I. Technical Backdrop

In the early 1990s, hardware support for instruction-level parallelism was advancing at a rapid rate. For example, Digital introduced the 2-way superscalar Alpha 21064 in 1992 and Intel introduced its 2-way Pentium in 1993. By 1995, the Alpha architecture had already jumped to 4-way superscalar (the 21164). This appeared to be a trend that would continue. While that did not pan out the way many expected, what was abundantly clear even then was that our ability to build hardware for instruction level parallelism was quickly outpacing our ability to find it in software.

Multithreading was a technology with a very long history, going back to the 1950s (and in fact, one of the authors did his PhD thesis on multithreading in 1979). It had appeared in real machines like the CDC 6600 PPU [6], the Denelcor HEP [4], etc. A multithreaded architecture keeps the state of multiple contexts in hardware at once, allowing it to introduce instructions into the pipeline from multiple threads without waiting for a slow software context switch. Several of the earliest machines were fine-grain multithreaded, able to introduce an instruction from a distinct thread each cycle. Others, like MIT's TX-2 and Alewife [1] architectures, utilized a less aggressive model, coarse-grain multithreading, which can be thought of as a hardware accelerator for context switches (able to do them in a handful of cycles, fast enough to profitably switch on long memory latencies). Fine-grain multithreading, however, saw a revival in the 1990s Tera MTA architecture [2].

II. Simultaneous Multithreading (SMT)

We decided to speculate on how multithreading would combine with modern, high performance superscalar processors, resulting in our ISCA 1995 paper introducing the term Simultaneous Multithreading (SMT) [7]. There were two aspects to that combination that were critical.

First, we combined multithreading with superscalar, and defined a new model of multithreaded execution, which aggressively mixed instructions from multiple threads in instruction issue, seeking to maximize utilization of the issue bandwidth and the rest of the pipeline. That's the model we called SMT and demonstrated the benefits in a wide superscalar processor (recall, most of the community thought 8-wide superscalar processors were one announcement away).

But, perhaps more critically, and ultimately of much higher impact, we combined multithreading with an aggressive high-performance pipeline. This represented a huge break in mindset. Multithreading, particularly the more aggressive fine-grain machines, had used multithreading primarily to simplify the pipeline. The CDC 6600 PPU demonstrated that with 10 threads, and instructions fetched and executed in strict thread sequence, a 10-cycle memory latency could be completely hidden without the need for caches. The HEP (and similarly, the Tera MTA that came later) had no support for forwarding, thus even simple arithmetic operations incurred a latency of 8 cycles, which could be completely hidden in the presence of enough threads. This, then, was the tradeoff that many believed was inherent to multithreaded architectures – they were fast in the presence of many threads, but agonizingly slow without sufficient thread-level parallelism. No one had seen a multithreaded architecture that rivaled the fastest single-thread pipelines, even when running only one thread. Thus, the real impact of our first SMT papers was this revolutionary notion that we could build a pipeline that was just as fast as ever with one thread, but faster with more – key to that was (1) retaining the full suite of performance features of a state-of-the-art pipeline, and (2) allowing each thread full access to as many pipeline resources as possible.

If that was indeed the most critical contribution of those papers, that explains why the authors have always believed that the second SMT paper in 1996 (the subject of this retrospective) was the most important and impactful of the two. The first paper was more about the execution model – what you could get if you could build it. This paper was about how to actually build it.

But there were a couple of developments that made "how to build it" far more interesting than a follow-up implementation paper. The first was the rapid switch from in-order processing to out-of-order processors. The first SMT paper was written in the context of in-order processors. But between the start of the first paper and the second, there were numerous announcements of out-of-order machines and the ground had officially shifted.

Second, intrigued by ideas in the 1995 SMT paper, an Alpha architect (Joel) contacted the group, resulting in Dean going to Massachusetts to spend a week brainstorming with Joel and Rebecca (of DEC's VSSAD advanced R&D team) how SMT might be used to enhance a future Alpha processor (the 21464) – which was planned to be both out-of-order and 8-wide superscalar. This began an academic/industry collaboration that resulted in the second SMT paper, which was thus deeply grounded in the issues of a real, industry-leading processor design.

Those discussions, and ones that ensued later, drove many directions pursued in this paper. Several discussions were particularly memorable. First, it became clear quickly that while out-of-order plus SMT was likely more complex than in-order plus SMT, the *marginal* complexity of adding SMT to an out-of-order machine was far less – because many of the capabilities and shareable resources we wanted were already there. Register renaming, which hides the logical names of registers, also hides the thread IDs, and as well it guarantees complete disambiguation between registers belonging to different threads. Therefore, the scheduler need not change at all, and instructions can be mixed from multiple threads each cycle for free. The register file's size is dominated by the demand for renaming registers, not the logical registers, so the incremental cost of adding a thread (32 logical registers) to the register file is lower.

Second, we discussed how to fetch for an SMT processor. The obvious choice of round robin fetching (CDC 6600 PPU style) was quickly found to be a bad idea – it was too easy for a stalled thread to flood the pipeline with bad instructions. It was thus clear that we needed to find a better way, and this insight was the most enduring one; namely, that the fetch unit was the key to everything in an SMT processor.

An out-of-order processor hums when parallelism is maximized in the instruction window – it ensures that ALUs are heavily utilized, throughput is high, and critical structures do not fill up due to stalled instructions, which can cause the whole pipeline to stall. In a single-threaded pipeline, you have few options to increase parallelism, except to pursue more and more speculation. In a multithreaded pipeline, you have this marvelous ability to impact the quality (and parallelism) of the instructions you fetch, because you get to decide which thread you fetch from every cycle.

We examined fetch policies that accounted for level of speculation (based on branches) and likelihood of a memory stall (based on loads). But the best solution was conceptually simple and more holistic – try to keep the number of in-flight instructions balanced between threads (the ICOUNT policy). If a thread starts to fetch stalled instructions, it soon becomes out of favor. A thread that is recovering from a branch mispredict and flush is brought back up to speed quickly. Threads with long chains of dependent instructions will tend to fetch at the same rate as they execute.

III. Impact

There is no question the impact of this work is significant, both on the research community, and then eventually (it took a few years) on industry. Many papers looked at aspects of the SMT design – among them, a whole slew of papers tweaked and modified the ICOUNT mechanism to get even better performance. A number of works attempted to exploit the hardware support for multithreading for other interesting purposes – helper threads, thread-level speculation, etc.

The first machine to be announced as supporting SMT, naturally, was the aforementioned Alpha 21464, announced in 1999 with an expected ship date of 2003 [3]. To the disappointment of all of these authors, unfortunate timing of company breakups and acquisitions ensured that that processor never saw the light of day.

We were forced to wait, then, until Intel introduced an SMT Xeon processor in 2002 (choosing to trademark the term hyperthreading to put their own stamp on the terminology). It can be argued that the introduction of SMT, along with chip multiprocessing which came to Intel a little later, is one of the two most visible changes to processor architecture in the last quarter century, if not longer.

Intel has continued to feature SMT in every major high-performance processor they have introduced. AMD came to the game much later, finally introducing SMT in their Zen architecture in 2017. While both are still stuck at 2 threads, there have been more aggressive SMT implementations, including the IBM Power 8 with eight threads.

Of late, SMT has fallen out of favor in some circles, due to security concerns. Certainly, sharing virtually all core resources maximizes the potential leakage between co-located threads. However, recent work [5] has demonstrated the design of a secure (or at least no less secure than multicore) SMT processor that sacrifices little of the performance and efficiency advantages of SMT.

References

[1] A. Agarwal, B.-H. Lim, D. Kranz, and J. Kubiatowicz, "April: A processor architecture for multiprocessing," in *Proceedings of the 17th Annual International Symposium on Computer Architecture*, 1990.

[2] R. Alverson, D. Callahan, D. Cummings, B. Koblenz, A. Porterfield, and B. Smith, "The Tera computer system," in *Proceedings of the 4th International Conference on Supercomputing*, 1990.

[3] J. Emer, "Simultaneous multithreading: Multiplying alpha performance," in *Proceedings of Microprocessor Forum*, 1999.

[4] B. J. Smith, "Architecture and applications of the HEP multiprocessor computer system," in *SPIE Real Time Signal Processing*, 1981.

[5] M. Taram, X. Ren, A. Venkat, and D. Tullsen, "SecSMT: Securing SMT Processors against Contention-Based Covert Channels," in *USENIX Security Symposium (USENIX Security)*, 2022.

[6] J. E. Thornton, *Design of a Computer—The Control Data 6600.* Scott Foresman Co, 1970.

[7] D. M. Tullsen, S. J. Eggers, and H. M. Levy, "Simultaneous multithreading: Maximizing on-chip parallelism," in *Proceedings of the 22nd Annual International Symposium on Computer Architecture*, 1995.