CS 5341

Oscar Galindo

**Simultaneous Multithreading: Maximizing On-Chip Parallelism**

This work begins by outlining the main idea discussed throughout the paper which is called “Simultaneous Multithreading” (SM). Which in the author’s words allows to “issue to multiple functional units each cycle.” The idea behind this proposed architecture was to “[combine] the multiple-issue-per-instruction features of [processors] with the latency-hiding of multithreaded architectures.” As the authors mention in this work the idea is to prevent vertical and horizontal waste which occur when not all the issue slots that can be used are used, therefore decreasing the utilization of bandwidth of the processor.

An explanation offered by the authors as to why their implementation is an improvement lies on the fact that the previously mentioned “waste” is actually underutilized functional units capacity or missed chances to utilize resources to run programs faster. The authors show that there is not a single reason why such waste of functional units exist, in fact they mention many possible reasons. Trying to solve an issue that causes waste will probably likely not be enough to remediate the situation, hence a different solution is needed, which they propose is Simultaneous Multithreading.

It is important to note that at the time of writing of this article, there was not a simultaneous multithreaded processor implementation, so experiments are performed through simulations. Authors do concede that their simulations for simultaneous multithreading could be optimistic. After mentioning that their analysis could be optimistic the authors, the authors present different implementations for simultaneous multithreading which are then tested in terms of performance. In terms of results, the simultaneous multithreaded implementations outperform single-thread execution or single-issue processors. The results of implementing simultaneous multithreading are as surprising as offering speedups between 3.2 to 4.2 with an issue as high as 6.3 IPC (Instructions Per Cycle).

Among the conclusions offered by the work are that “It is not necessary to have extremely large caches to achieve the speedups in this section…” additionally “… our results show that simultaneous multithreading surpasses limits on the performance attainable through either single-thread execution or fine-grain multithreading on a wide superscalar.” For the case of cache size, the paper offers an analysis in terms of size of private cache (i.e., L1) and the publicly shared cache (i.e., L2 and L3) with respect to performance. The paper concludes that if the system is to have 8 threads, with a four-issue implementation, which are to be constantly issuing requests we ought to implement a 64p\*64s or 64kb of private cache shared among 8 threads or 8 kb of L1 private cache/thread, and a pool of 64kb of cache public among all threads.

Finally, the paper further experiments with different configurations of Simultaneous Multithreading compared to Single-Chip Multiprocessing. These tests vary from setup to setup in the number of Functional Units, Total Issue Bandwidth, and number of register sets. In all cases simultaneous multithreading proves to provide a bigger throughput or a higher number of instructions per cycle. The authors point that “simultaneous multithreading outperforms the single-chip multiprocessing due to dynamic partitioning of functional units.” The authors conclude the article with following ideas:

* Simultaneous Multithreading outperforms single-threaded wide superscalar by up to a factor of 4.
* Fine-grain Multithreading closes the gap with respect to Simultaneous Multithreading but still is outperform by a factor of up to 2.
* A simultaneous multithreaded architecture is superior in per- formance to a multiple-issue multiprocessor, given the same total number of register sets and functional units. Moreover, achieving a specific performance goal requires fewer hardware execution resources with simultaneous multithreading.

In my opinion this article presents important motivations that I did not understand about “waste”. This was fundamental to understand to ‘catch’ the advantages of simultaneous multithreading. In terms of complains, only one, we don’t see what the programs are being run in the tests, but the results seem to be solid anyways. So, overall great, just as always I think transparency on how benchmarks are run can only help.