CMP SCI 635: Modern Computer Architecture

Efficiently Scaling Out-of-Order Cores for Simultaneous Multithreading

Name: Kunjal Panchal

Student ID: 32126469

Date: 10th Oct, 2019

Paper: 11 – Pipelining [Sleiman16]

Strengths:

1. The flexibility of the SMT thread fetch policy is synergistic with simple instruction steering. When instructions are fetched from a slow-moving thread, they are steered to the shelf, avoiding IQ congestion. Conversely, when an instruction is mis-steered to the shelf, stalling execution, other threads benefit from the available IQ capacity and fill the bubbles with useful execution. This synergy facilitates a steering design without large and power-hungry meta-data structures, which would undermine the energy-efficiency objective of our microarchitecture.
2. The shelf-augmented microarchitectures improve performance over the baseline by 8.6% and 11.5% on average and up to 15.1% and 19.2% for the conservative and optimistic microarchitecture assumptions, respectively. This approach captures almost half of the throughput improvement of the larger OOO core with substantially less hardware.
3. The ability of one SMT thread to make progress while another is stalled hides the brief stalls created by incorrect steering decisions, allowing even simple mechanisms that assumes all memory accesses are L1 cache hits to nonetheless make effective use of the shelf.

Weaknesses:

1. The shelf loses performance when less than half of all in-flight instructions are in-sequence, when window requirements are imbalanced across threads (the shelf is statically partitioned), the steering heuristic mis-steers instructions, or when reordered instructions require more LQ or SQ resources.
2. In-sequence instructions arise in single-threaded execution, the interleaving of multiple threads in a SMT core spreads the issue of dependent instructions apart, substantially increasing the fraction of in-sequence instructions. Hence, a shelf won't improve performance in single-threaded execution.

Questions/Assertions:

1. Why not only send those instructions to the shelf which are definitely going to commit and aren't just speculations or branch predictions? Wouldn't that benefit us from doing away with saving register states and re-order buffer entries for shelf instructions.
2. How the maximum remaining resolution cycles are computed? Is that because if the instruction is at the head of shelf, it is obvious that all its dependencies are taken care of and thus, we know how many cycles it will take to perform all fetch-load-store operations? Is there anything that can prove the resolution cycle count wrong?
3. Can we view Branch Delay Slots as an out of order execution?
4. A student from University of Toronto has written his own Out of Order soft-core, verified it, made it support enough x86 to boot an OS, and obtained a score within 50% of haswell at coremark. [http://hdl.handle.net/1807/80713]. Motivation behind this was: "*Although FPGAs continue to grow in capacity, FPGA-based soft processors have grown little because of the difficulty of achieving higher performance in exchange for area. Superscalar out-of-order processor microarchitectures have been used successfully for hard processors for many years, but have so far been avoided for FPGAs due to the area increase and the expectation that a loss in clock frequency would more than offset the instructions-per-cycle (IPC) gains.*"

   "*With careful microarchitectural choices and circuit design, I show that it is possible to build a complex microarchitecture on an FPGA, getting about 2.7x performance per clock and 0.8x clock frequency of Altera's Nios II/f single-issue in-order processor.*"

   Does this sound promising for future? What significance soft-cores hold?

5. The Cray compiler was supposedly an "automatically vectorizing" compiler. How well or often a compiler is able to auto-vectorize loops. The Cray compiler was a Fortran compiler, where it may be a bit easier to do auto vectorization than in some other popular languages, such as C/C++.