ECEn 528
Study Guide - Multithreading

- Read Section 3.12 of H&P
  - Things to focus on
    - Differences between the kinds of multithreading
    - Where performance and energy efficiency come from

  - Clarifications
    - Intel's “hyperthreading” is a form of simultaneous multithreading (SMT)
    - Multiple threads within a process do not require separate page tables, however, the TLB does need to distinguish between the processes different threads may belong to
    - H&P's use of “issue” where we mean “dispatch” occurs in this section too.
    - Figure 3.33 is missing two columns on the right (8-core CPI and effective IPC)
    - Figure 3.35 compares running a benchmark using 2 threads (these are all multi-threaded benchmarks) vs. using 1 thread.

  - Answer the following questions:

    1. What are the main differences between fine-grained and simultaneous multithreading?

    2. Why does SMT not make sense for an in-order pipeline?

    3. Why does fine-grained multithreading not make sense for an OoO pipeline?

    4. The performance improvement due to SMT in current processors such as the i7 is very low. Why do the authors conclude that SMT is “good”?

- Read “Simultaneous Multithreading: Maximizing On-Chip Parallelism” (the paper that originally made the idea stick)
  - Things to focus on
    - Horizontal vs. vertical waste
    - What assumptions are made about the amount of hardware resources?

  - Clarifications
    - Intel's “hyperthreading” is a form of simultaneous multithreading (SMT)
    - Their definition of “issue” matches H&P.
    - Their “limited dynamic execution” is kind of odd: dispatch can only occur up to a data dependence between instructions. This idea allows renaming to occur in parallel, at the cost of delaying instructions from entering the IW and reducing the amount of OoO behavior that occurs. It appears that dispatch can occur from multiple threads, though I'm fuzzy about how many total instructions can be dispatched in a cycle -- it's probably 8. The IW is partitioned by thread, but (our) issue then occurs out of the whole window. Good static scheduling is necessary to ensure that performance isn't limited at the dispatch stage due to dependences.
      - Interesting quote: “We don't need renaming unless we need precise exceptions” -- But why wouldn't you need precise exceptions in a useful processor?

  - Answer the following questions:

    1. While SMT support is logically quite simple to add to an OoO pipeline, what resources must be scaled up in size for SMT to be practical? Which structural hazards are most critical?

    2. Why do SMT processors outperform chip multiprocessors with similar issue bandwidth?

    3. Why haven't SMT processors lived up to the performance potential suggested by this paper?