15 Midterm Review

[Midterm F23 1a] Suppose we have k transistors. We can use those k transistors to build either one big core with high frequency or build several smaller cores with lower frequency. Nowadays, designers opt for the second option. State three reasons for why they do that.

- Smaller cores with lower frequency consume less power than a big core with high frequency.
- Several smaller cores can exploit task-level parallelism.
- Smaller cores are easier to design and test.
- Executing threads in parallel on several cores can hide memory latency.

[Midterm F23 1b] Suppose we have two threads belonging to the same process and we have three choices on where to execute them: on a single superscalar core, on a two-way hyperthreading core, or on two physical superscalar cores.

When will it give better performance if we execute the two threads on a single superscalar core

- There is a dependency between the two threads. That is, one of them cannot execute till the other is done.

When will it give better performance if we execute the two threads on the two-way hyperthreading core?

- The two threads are independent and can execute in parallel.
- The two threads require the similar data so they will help each other filling the caches if any.
- The two threads do not require the same execution units. For example, one thread needs int operations while the other needs floating points. Or, one thread is compute bound while the other is memory bound.

When will it give better performance if we execute the two threads on the two physical superscalar cores?

- Threads are independent and can execute in parallel.
- Threads do not write the same cache block triggering cache coherence.
- Not much communication between the two threads.

[Midterm F23 1c] Suppose we have eight cores on a shared-memory machine: two of the cores are just pipelined, two of them are superscalar, and the remaining four are two-way hyperthreaded. What is the maximum number of processes that can be executed in parallel on that whole system?

To get the maximum number of processes, each process needs to be single-threaded.

Pipelined cores can each execute one thread: $2 \times 1 = 2$

Superscalar cores can each execute on thread: $2 \times 1 = 2$

Each of the four two-way hyperthreading can execute two threads simultaneously: $4 \times 2 = 8$

This makes the total number of threads = $2 + 2 + 8 = 12$.

[Midterm F23 2d] Why we cannot reach perfect load balancing in parallel programs even if we do our best?

- Some threads may get cache misses while the others don't.
- Some threads may get page faults while others don't.
- Some threads may run on weaker cores than others, if we have processor with different cores.
- The code may have if-else, and some threads will go in a different path that contains more computations than others.
- Accessing the memory may lead to different latency depending on where the core is relative to the banks.

[Midterm S22 1a] Suppose we have a core with pipelining but no superscalar or hyperthreading capabilities. Will this core benefit from branch prediction?

Yes, it will. Without branch prediction, whenever a condition branch is encountered, the fetch phase won't know which next instruction to fetch, introducing bubbles (i.e. empty phases) in the pipeline.

[Midterm S22 1c] "By having more variables shared among processes, we increase the chance of false sharing and hence performance will go down due to coherence overhead". Is this statement true or false?

This statement is false. There cannot be any shared variables among processes because processes do not share virtual address space. (only shared among threads)

Shared Memory Systems (Threads)

Distributed Memory Systems (Processes)

[Midterm F21 2b]

(T) We can have a core with superscalar capability but without speculative execution.

(F) In a core with hyperthreading technology, instructions can come from different threads belonging to the same process but not from threads belonging to different processes.

(F) MISD is implemented in real life in GPUs even though multicore can do the same job.

(T) If we have two processes, with one thread each, executing on a shared memory machine, there is no coherence overhead that can result from their execution.

(F) If we have two processes, with multiple threads each, executing on a shared memory machine, there is no coherence overhead that can result from their execution.