Section 1: Week 1: Processor Scheduling

Nate Bachmeier

TIM-8120: Distributed Systems

September 22nd, 2019

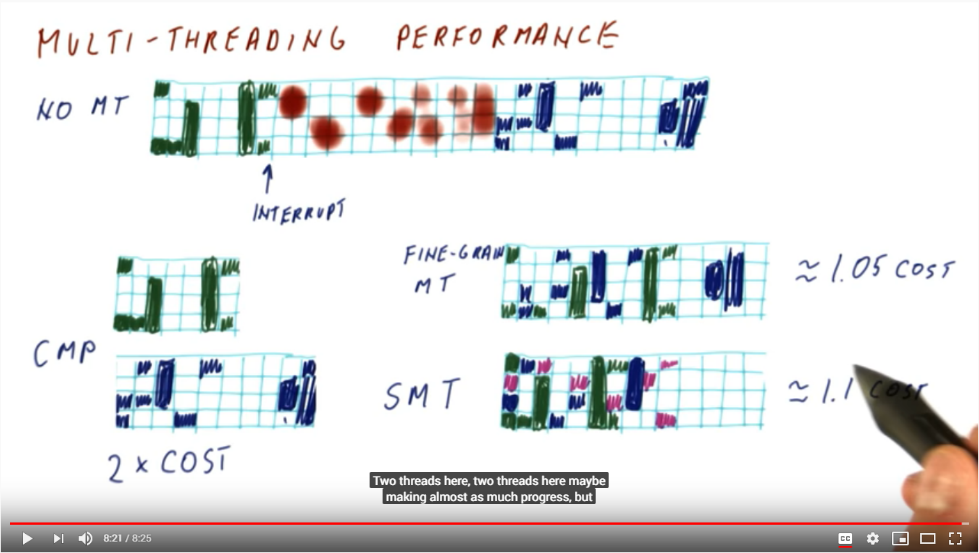
North Central University

# Processor Scheduling

Modern processors can run hundreds of processes in parallel, while only containing a handful of physical cores. They accomplish this goal through efficient scheduling strategies.

## Interweaving

Georgia Tech’s High-Performance Computer Architecture course includes this diagram to explain interweaving threads (Prvulovic, 2015).



At the top is the behavior of a single core and no hardware multi-threading support. For this scenario a thread is scheduled and runs until an interrupt. After the interrupt, context switching takes place, and for some number of cycles, no progress made. Eventually, the second thread, shown in blue, can be scheduled.

To the bottom left is the behavior for a chip multiprocessor, which uses two physical cores to run two concurrent threads. A challenge with this approach is that it is prohibitively expensive due to requiring nearly twice the silicon.

The middle right is a fine-grained multi-thread core, that attempts to schedule something in every clock cycle. These processors do not wait for interrupts and evaluate instructions as soon as they become available.

The bottom right is a simultaneous multithreaded core (SMT) and is capable of running different threads during the same clock cycle. This scenario provides the most parallel execution as work is continuously flowing to all circuits as quickly as possible.

## Pipelining

Prvulovic provides an analogy that the instruction pipeline is similar to an oil pipeline. Let’s say that you pump a gallon of oil through the pipe, and that takes one day, how long does it take to get the second gallon? The answer is near real-time, provided they started at the same point. This behavior is because both gallons took one day, but the offset between them is near zero.

The same behavior exists for instruction pipelines as the processor attempts to feed in many operations from the different threads. For instance, the Hummingbird E203 will perform the services (stage 1) finger, decode, execute, write back; (stage 2) access via LSU; and (stage 3) copy back to a general register. (Tianchuan & Zhenbo, 2019).

## Speculative Execution

Modern processors follow the SMT strategy and use instruction pipelines as described above. Their goal is to minimize the amount of idle space within that pipeline as it is time that could be used to do something productive.

One strategy is to pre-emptively execute sections of the program that are likely to follow (Kocher, et al., 2018) (Lipp, et al., 2018). If the prediction is correct, then the results for that code block are already evaluated. Otherwise, the speculative execution is not committed, and the right instructions evaluated. Since the worst case of speculative execution equals the time of having been idle, there is no harm in doing it.

With the Spectre and Meltdown vulnerabilities, it has shown that speculative execution can be used to bypass memory isolation protections and allow arbitrary memory reads. According to Kocher et al. and Lipp et al., these attacks rely on causing the pre-emptive execution to leak state into the processor cache. Though time measurement strategies, it becomes possible to determine the values leaked.

## Reorder Buffer

The densest and most complex component of the scheduling component is the reorder buffer (Choi, Park, & Jeong, 2013). This component is responsible for reassembling the out of order execution into the program defined order. Increasing the size of this structure can improve performance by increasing the amount of speculative execution; however, it also decreases power efficiency.

Choi et al. propose that one strategy is to move the reorder buffer into a separate component. Then exploit the fact that most basic blocks are around six instructions and by default correctly ordered. By optimizing for the typical case, and treating the exception as an exception, they can drastically reduce the power usage.

# Conclusions

Processor scheduling is an exciting area of computer research that can be important for distributed system engineers to understand. Various software based systems attempt to mimic these patterns as they are very efficient approaches to handling arbitrary work.

# References

Choi, Park, & Jeong. (2013). Revisiting reorder buffer architecture for next generation high performance computing.

Kocher, Horn, Fogh, Genkin, Gruss, Haas, . . . Yarom. (2018). Spectre Attacks: Exploiting Speculative Execution.

Lipp, Schwarz, Gruss, Prescher, Hass, Fogh, . . . Hamburg. (2018). Meltdown: Reading Kernel Memory from User Space.

Prvulovic. (2015). *High-Performance Computer Architecture.* Retrieved from Udacity: https://classroom.udacity.com/courses/ud007/lessons/3650589023/concepts/9999288670923

Tianchuan, & Zhenbo. (2019). An ultra-low power RISC-V processor pipeline structure.