Multiprocessor scheduling, part 1
-Challenges-

Pontus Ekberg

2017-10-03
What is a multiprocessor?

• Simplest answer: A machine with >1 processors!
  → In scheduling theory, we include multicores in this definition

• Multiprocessors allow us to run >1 threads at the same time
  → Great for performance, cost, weight etc.
Multiprocessor scheduling

- Multiprocessor scheduling has been studied for many decades, also for real-time systems
  → The *multicore revolution* has made it highly relevant today!
Multiprocessor scheduling

- Multiprocessor scheduling has been studied for many decades, also for real-time systems
  → The *multicore revolution* has made it highly relevant today!

- Multiprocessor scheduling for real-time systems is *hard*
  → It is difficult to find good scheduling algorithms and schedulability tests
  → There are many huge practical challenges with using multicore processors in real-time systems
  → Lots of open problems (*exciting!*).
What is a multicore processor?

Multicore \( \approx \) Tightly coupled CPU cores that share a memory system
Are $m$ slow cores as good as 1 fast?
Are $m$ slow cores as good as 1 fast?

- Only if your workload can utilize all $m$ cores simultaneously
  → That is, you can *parallelize* a single application...
  → ... or have many different things to run at once
Are $m$ slow cores as good as 1 fast?

- Only if your workload can utilize all $m$ cores simultaneously
  → That is, you can *parallelize* a single application...
  → ... or have many different things to run at once

- Only if your workload is mostly *computation-heavy*
  → It doesn't saturate shared resources, such as caches, buses, I/O, ...
Why do we use multicores?
Caches

- DRAM performance has increased much slower than CPU performance
  → Relatively speaking, memory accesses become slower and slower

- Caches were introduced to *hide* this speed gap
  → By storing (caching) data the processor believes will be used soon in a smaller and faster memory
  → Works well because typical programs exhibit temporal and spatial memory locality
Memory hierarchy

- The further you go in the memory hierarchy, the slower your accesses will be.

→ Caches are hugely important for performance.

“Typical” speed: ~4 cycles ~10 cycles ~40 cycles ~100 cycles

“Typical” size: ~64 KB ~1 MB ~10 MB >1 GB

Usually: private private/shared shared shared
Intel Core i7 die
For real-time systems

• Caches are designed for good average-case performance, *not for predictability*

• Safely predicting cache hits in *private cache* is very difficult (though we have seen remarkable progress)

• Safely predicting cache hits in *shared cache* is practically impossible

• Other micro-architectural features (memory controller, hyper-threading, shared buses, I/O etc.) also make it *very difficult to predict WCET* on multicores
Potential solution: cache coloring

core 1
Private L1 cache

core 2
Private L1 cache

core 3
Private L1 cache

core 4
Private L1 cache

Shared L2 cache
Potential solution: cache coloring

- core 1: Private L1 cache
- core 2: Private L1 cache
- core 3: Private L1 cache
- core 4: Private L1 cache
- Shared L2 cache
Potential solution: cache coloring

- core 1
  - Private L1 cache

- core 2
  - Private L1 cache

- core 3
  - Private L1 cache

- core 4
  - Private L1 cache

Shared L2 cache
Potential solution: cache coloring

- Core 1: Private L1 cache
- Core 2: Private L1 cache
- Core 3: Private L1 cache
- Core 4: Private L1 cache
- Shared L2 cache
Potential solution: cache coloring

Logical pages of task A

Logical pages of task B

Physical pages

L2 cache

Controlled by software (OS)

Indexed by hardware
Potential solution: cache coloring

- core 1: Private L1 cache
- core 2: Private L1 cache
- core 3: Private L1 cache
- core 4: Private L1 cache

Shared L2 cache
Case study

- A simple experiment on a dual-core Linux machine
Other solutions to MC issues?

- Buses can often be made predictable using time-division multiple access (TDMA)
  → But this is inefficient

- A better solution is to use a special-purpose memory controller
  → But then someone has to build it!

- By exploiting performance measurement registers, it is sometimes possible to implement memory accesses budgeting in the operating system
Multiprocessor scheduling: Themes

- **Global scheduling**
  - New task
  - Waiting queue
  - CPU 1: 8
  - CPU 2: 1
  - CPU 3: 6

- **Partitioned scheduling**
  - CPU 1: 2, 5, 1
  - CPU 2: 3, 8, 6
  - CPU 3: 4, 9, 7

- **Semi-partitioned scheduling**
  - CPU 1: 2, 5
  - CPU 2: 3, 7
  - CPU 3: 1, 6, 4