Lecture 04—SMP, Processes vs Threads

ECE 459: Programming for Performance

Patrick Lam

University of Waterloo

January 12, 2015

Last Time: Laws, Thread Models

Amdahl's Law: speedup is bounded by serial part.

Gustafson's Law: with more resources,
can compute more in same time.

Thread models: kernel (1:1), user (N:1), hybrid (M:N).

Admin

Log on to ecgit by Wednesday and upload your ssh key.

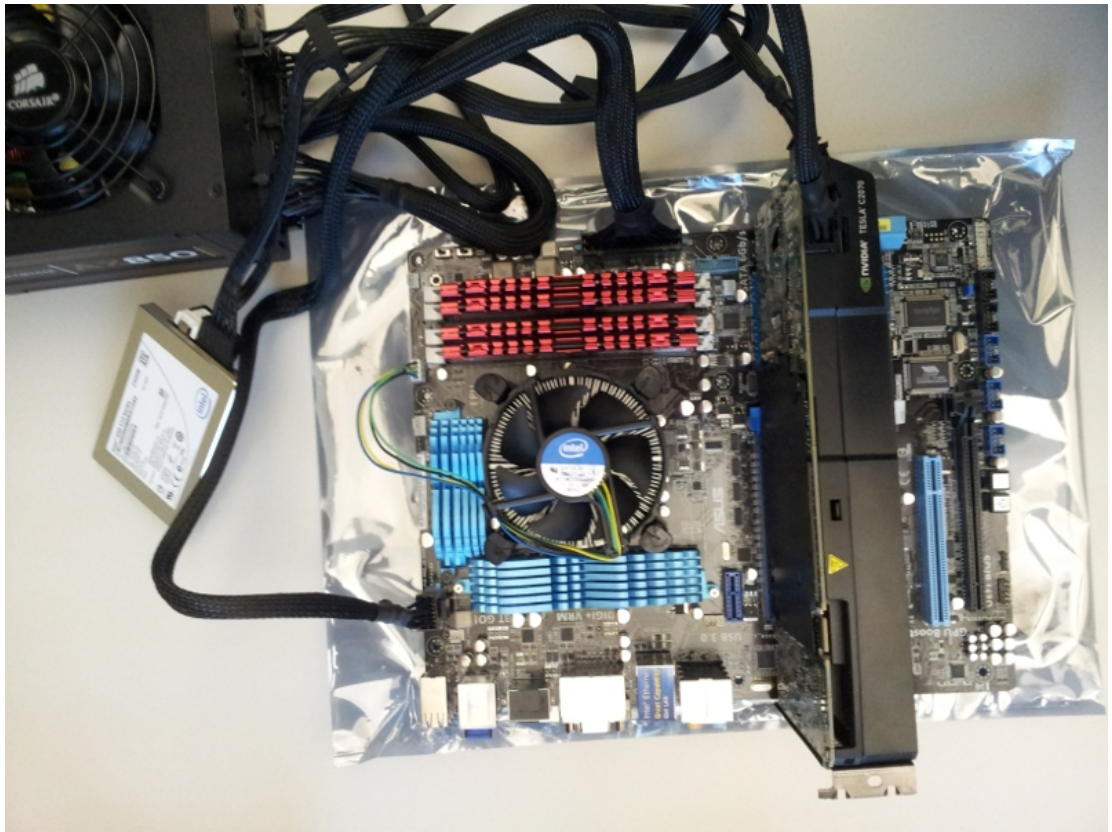
In case you're interested, this course has 155 students enrolled at the moment (a record).

SE: 47, NE: 1, MECTR: 11, exchange: 2
ELE: 14, COMPE: 80.

Part I

Types of Multiprocessing

Example System—Physical View



- Only one physical CPU

Example System—System View

```
jon@ece459-1 ~ % egrep 'processor|model name' /proc/cpuinfo
processor : 0
model name: Intel(R) Core(TM) i7-2600K CPU @ 3.40GHz
processor : 1
model name: Intel(R) Core(TM) i7-2600K CPU @ 3.40GHz
processor : 2
model name: Intel(R) Core(TM) i7-2600K CPU @ 3.40GHz
processor : 3
model name: Intel(R) Core(TM) i7-2600K CPU @ 3.40GHz
processor : 4
model name: Intel(R) Core(TM) i7-2600K CPU @ 3.40GHz
processor : 5
model name: Intel(R) Core(TM) i7-2600K CPU @ 3.40GHz
processor : 6
model name: Intel(R) Core(TM) i7-2600K CPU @ 3.40GHz
processor : 7
model name: Intel(R) Core(TM) i7-2600K CPU @ 3.40GHz
```

- Many processors

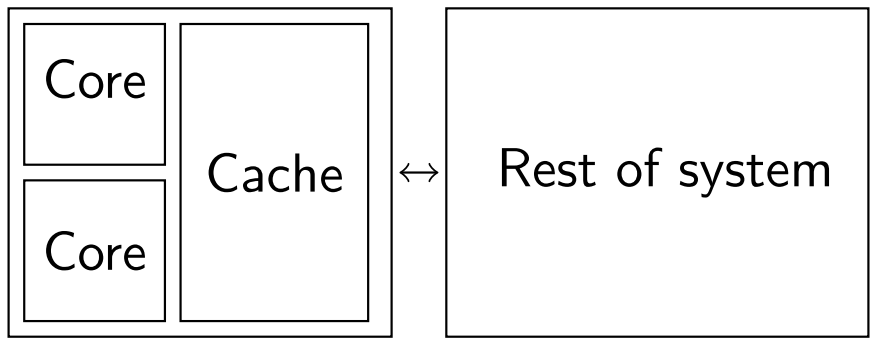
SMP (Symmetric Multiprocessing)

Identical processors or cores, which:

- are interconnected, usually using buses; and
- share main memory.

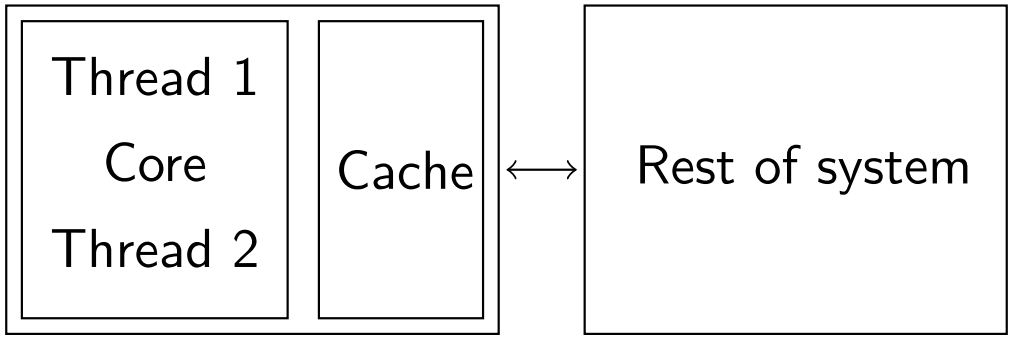
- SMP is most common type of multiprocessing system.

Example of an SMP System



- Each core can execute a different thread
- Shared memory quickly becomes the bottleneck

Executing 2 Threads on a Single Core



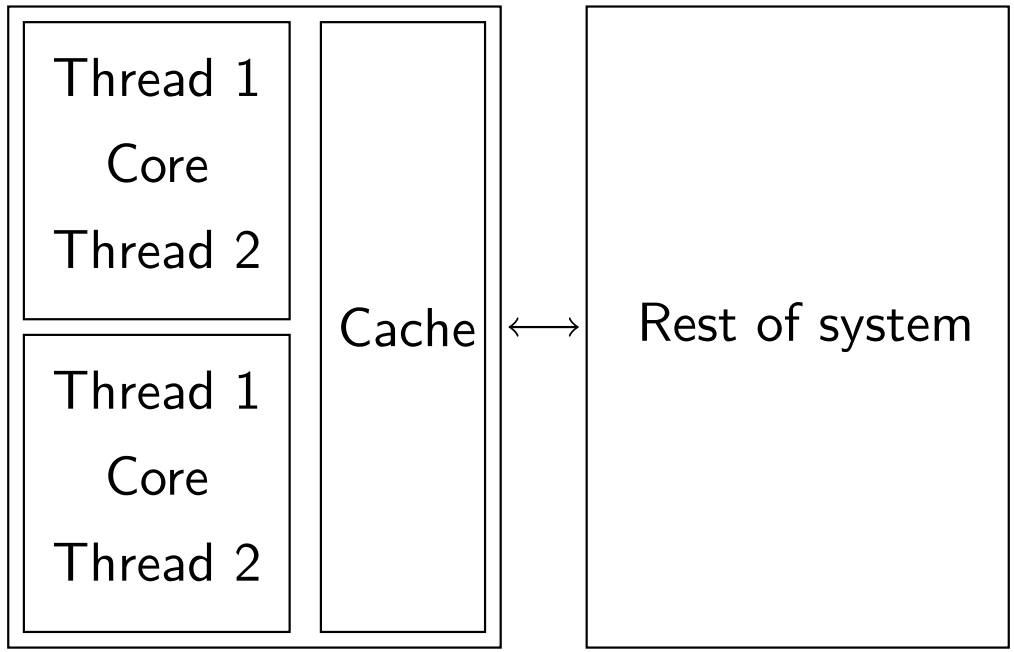
On a single core, must context switch between threads:

- every N cycles; or
- wait until cache miss, or another long event

Resources may be unused during execution.

Why not take advantage of this?

Executing M Threads on a N Cores



Here's a Chip Multithreading example.

UltraSPARC T2 has 8 cores, each of which supports 8 threads. All of the cores share a common level 2 cache.

SMT (Simultaneous Multithreading)

Use idle CPU resources (may be calculating or waiting for memory) to execute another task.

Cannot improve performance if shared resources are the bottleneck.

Issue instructions for each thread per cycle.

To the OS, it looks a lot like SMP, but gives only up to 30% performance improvement.

Intel implementation: Hyper-Threading.

Example: Non-SMP system

PlayStation 3 contains a Cell processor:

- PowerPC main core (Power Processing Element, or “PPE”)
- 7 Synergistic Processing Elements (“SPE”s): small vector computers.

NUMA (Non-Uniform Memory Access)

In SMP, all CPUs have uniform (the same) access time for resources.

For NUMA, CPUs can access different resources faster (resources: not just memory).

Schedule tasks on CPUs which access resources faster.

Since memory is commonly the bottleneck, each CPU has its own memory bank.

Processor Affinity

Each task (process/thread) can be associated with a set of processors.

Useful to take advantage of existing caches (either from the last time the task ran or task uses the same data).

Hyper-Threading is an example of complete affinity for both threads on the same core.

Often better to use a different processor if current set busy.

Part II

Processes vs Threads

Background

Recall the difference between processes and threads:

- Threads are basically light-weight processes which piggy-back on processes' address space.

Traditionally (pre-Linux 2.6) you had to use `fork` (for processes) and `clone` (for threads).

History

`clone` is not POSIX compliant.
Developers mostly used `fork` in the past, which creates a new process.

- Drawbacks?
- Benefits?

Benefit: `fork` is Safer and More Secure Than Threads

1. Each process has its own virtual address space:
   - Memory pages are not copied, they are copy-on-write—
   - Therefore, uses less memory than you would expect.
2. Buffer overruns or other security holes do not expose other processes.
3. If a process crashes, the others can continue.

Example: In the Chrome browser, each tab is a separate process.

Drawback of Processes: Threads are Easier and Faster

- Interprocess communication (IPC) is more complicated and slower than interthread communication.
  - Need to use operating system utilities (pipes, semaphores, shared memory, etc) instead of thread library (or just memory reads and writes).
- Processes have much higher startup, shutdown and synchronization cost.
- And, Pthreads/C++11 threads fix issues with `clone` and provide a uniform interface for most systems **(Assignment 1)**.

Appropriate Time to Use Processes

If your application is like this:

- Mostly independent tasks, with little or no communication.
- Task startup and shutdown costs are negligible compared to overall runtime.
- Want to be safer against bugs and security holes.

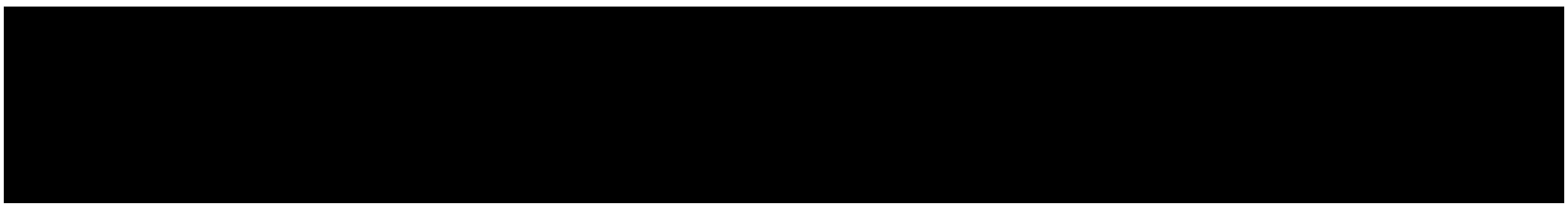
Then processes are the way to go.

For performance reasons, along with ease and consistency, we'll use Pthreads.

Live coding: overheads of threads vs fork

Results: Threads Offer a Speedup of 6.5 over `fork`

Here's a benchmark between `fork` and Pthreads on a laptop, creating and destroying 50,000 threads:



Clearly Pthreads incur much lower overhead than `fork`.