Lecture 03—Threading I

ECE 459: Programming for Performance

Patrick Lam

University of Waterloo

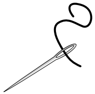
January 15, 2013

Parallelism versus Concurrency

Parallelism
Two or more tasks are parallel if they are running at the same time.
Main goal: run tasks as fast as possible.
Main concern: dependencies.

Concurrency
Two or more tasks are concurrent if the ordering of the two tasks is not predetermined.
Main concern: synchronization.

Threads



- What are they?
- How do operating systems implement them?
- How can we leverage them?

Processes versus Threads

Process

An instance of a computer program that contains program code and its:

- Own address space / virtual memory;
- Own stack / registers;
- Own resources (file handles, etc.).

Thread

"Lightweight processes".

In most cases, a thread is contained within a process.

- Same address space as parent process
  - Shares access to code and variables with parent.
- Own stack / registers
- Own thread-specific data

Software and Hardware Threads

Software Thread:
What you program with.

Corresponds to a stream of instructions executed by the processor.

On a single-core, single-processor machine, someone has to multiplex the CPU to execute multiple threads concurrently; only one thread runs at a time.

Hardware Thread:
Corresponds to virtual (or real) CPUs in a system. Also known as strands.

Operating system must multiplex software threads onto hardware threads, but can execute more than one software thread at once.

Thread Model—1:1 (Kernel-level Threading)

Simplest possible threading implementation.

The kernel schedules threads on different processors;

- NB: Kernel involvement required to take advantage of a multicore system.

Context switching involves system call overhead.

Used by Win32, POSIX threads for Windows and Linux.

Allows concurrency and parallelism.

Thread Model—N:1 (User-level Threading)

All application threads map to a single kernel thread.

Quick context switches, no need for system call.

Cannot use multiple processors, only for concurrency.

- Why would you use user threads?

Used by GNU Portable Threads.

Thread Model—M:N (Hybrid Threading)

Map M application threads to N kernel threads.

A compromise between the previous two models.

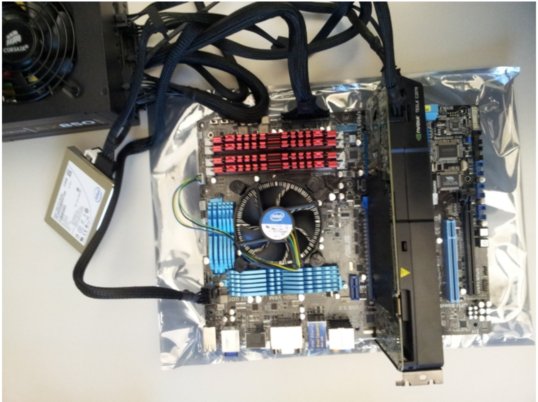
Allows quick context switches and the use of multiple processors.

Requires increased complexity; library provides scheduling.

- May not coordinate well with kernel.
- Increases likelihood of priority inversion (recall from Operating Systems).

Used by modern Windows threads.

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

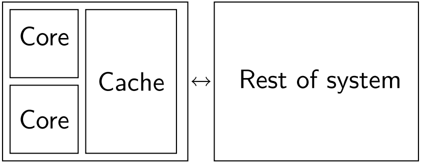
SMP (Symmetric Multiprocessing)

Identical processors or cores, which:

- Are interconnected using buses or another type of communication; and
- Share main memory.

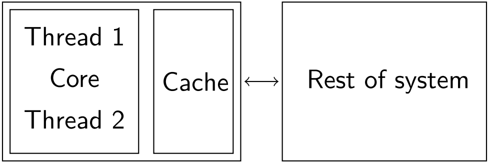
Most common type of multiprocessing system

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



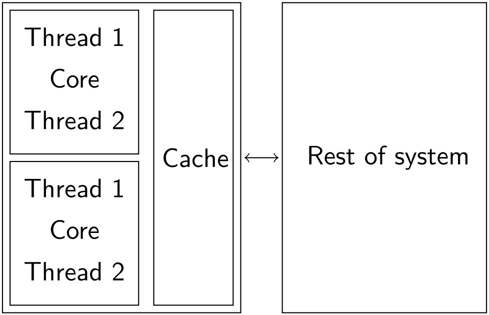
On a single core, must context switch between threads:

- 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 is busy.