Operating Systems and Multicore Programming (1DT089)

Background and Motivation

Tuesday January 21
Uppsala University 2014
karl.marklund@it.uu.se
Bang for the buck per time unit

Moore’s Law
The Fifth Paradigm

year 1945
≈ 1 calculation per second per $1000

year 2000
≈ 100 miljon calculations per second per $1000
When trying to develop faster and faster CPUs, the engineers have hit three walls.

**Instruction Level Parallelisms (ILP)** is a measure of how many of the operations in a computer program can be performed simultaneously. The potential overlap among instructions is called instruction level parallelism. Finding ILP is hard and as the clock rate goes up, the performance gains from ILP don’t increase as much as we wish.

**Memory** is much much slower compared the CPU. Even if larger and larger caches are added, this is a serious problem.

**Power consumption** keeps going up as we increase the clock rate.
Power Density Extrapolation

Power Density (W/cm²)

- Sun's Surface
- Rocket Nozzle
- Nuclear Reactor
- Hot Plate
- Pentium® processors

Year:
- '70
- '80
- '90
- '00
- '10
Watt, med symbolen W, är härledd SI-enhet för den fysikaliska storheten effekt, det vill säga energi (eller arbete) per tidsenhet.
1993 - CPU and Cooler

2005 - Cooler alone

1993 - CPU and Cooler
The power $P$ consumed by a CPU, is approximately proportional to current CPU frequency $f$, and to the square of the CPU voltage $V$:

\[ P = C V^2 f \]

where $C$ is capacitance.

Operating at a higher clock rate always requires more power.

- New features generally require more transistors, each of which uses power.
- As a processor model's design matures, smaller transistors, lower-voltage structures, and design experience may reduce energy consumption.
In general, sacrificing 13% in performance cuts the power consumption in half.

In general, a 73% penalty in power consumption for a 13% performance gain.

The trend towards multiple cores is an engineering approach that helps the CPU designers avoid the power consumption problem that came with ever increasing frequency scaling.

While it's certainly possible to build a 6 GHz general purpose x86 CPU, it's not proven economical to do so efficiently. That's why the move to multi-core started.

Multi-core products feature higher performance per watt than single-core ones.
Microprocessor Transistor Counts 1971-2011 & Moore’s Law

The curve shows transistor count doubling every two years.

Date of introduction

Transistor count
Multicore

A 4-core processor from AMD.
the "multicore crisis"
# Flynn’s Taxonomy

<table>
<thead>
<tr>
<th></th>
<th>Single instruction</th>
<th>Multiple instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single data</td>
<td>SISD</td>
<td>MISD</td>
</tr>
<tr>
<td>Multiple data</td>
<td>SIMD</td>
<td>MIMD</td>
</tr>
</tbody>
</table>
The traditional **von Neumann** architecture.

**Pipeline architectures** belong to this type, though a purist might say that the data is different after processing by each stage in the pipeline.

Modern graphics processing units (**GPUs**) are often wide SIMD implementations.

**Multi-core processors** are MIMD. Different cores execute different threads (Multiple Instructions), operating on different parts of memory (Multiple Data).

[Images: http://en.wikipedia.org/wiki/Flynn%27s_taxonomy]
As per-core CPU speed increases have slowed to a halt, processor vendors are embracing parallelism by multiplying the number of cores on CPUs, following what Graphics Processing Unit (GPU) vendors have been doing for years.

The Multi-core revolution promises to provide increases in performance, but it comes with a catch.

- Traditional serial programming methods are not at all suited to programming these processors.
**Basic principle:** large problems can often be divided into smaller ones, which are then solved in *parallel* (at the same time).

**NOTE:** This is not the same as *multithreading*. Multithreading can be done on a single core CPU. In such a case, two threads can never execute at the same time on the CPU.
In parallel systems, two tasks are actually performed simultaneously.

Parallelism is when tasks literally run at the same time, eg. on a multicore processor.

Concurrent systems give the appearance of several tasks executing at once, but these tasks are actually split up into chunks that share the processor with chunks from other tasks by interleaving the execution in a time-slicing way,

Concurrency is when two tasks can start, run, and complete in overlapping time periods. It doesn't necessarily mean they'll ever both be running at the same instant. Eg. multitasking on a single-threaded machine.

Note: In a concurrent system, concurrent tasks may be executed in parallel (if the hardware allows) but can also be executed on a single processor by interleaving the execution steps of each in a time-slicing way,
Two tasks T1 and T2 are **concurrent** if the **order** in which the two tasks are executed in time is **not predetermined**.

- T1 may be executed and finished before T2,
- T2 may be executed and finished before T1,
- T1 and T2 may be executed simultaneously at the same instance of time (parallelism),
- T1 and T2 may be executed alternatively,
- ...

**Concurrency** is often referred to as a **property of a program**, and is a concept **more general than parallelism**.

In this class, we will focus on concurrent programming.

Parallel programming is the topic of Programming of parallel computers (1TD480).
Concurrency ≠ Parallelism (3)

**Det är lättare att hitta problem som med relativ lätthet låter sig lösas med hjälp av concurrency än det är att hitta probelm som låter sig lösas parallellt.**

Conform to application semantics

High processor utilization

Concurrent

Parallel

Focus in this course!

Relative importance

Functionality

Performance
Operativsystem & Multicoreprogrammering

**Course content**

- Operating systems
- Concurrency
- Concurrent programming

**Period och studietakt:** period 23 (5 Hp, 33%), period 24 (10 Hp, 67%).

**Examination:** Tentamen och uppgifter (9 Hp), projekt (6 Hp).

**Betygsskala:** (U), 3, 4, 5.

**Efter godkänd kurs ska studenten kunna:**

- Redogöra för hur operativsystem och runtimesystem växelverkar med maskin- och programvara.
- Förklara och använda algoritmer och tekniker för schemaläggning och synkronisering i system med en eller flera processorer och processorkärnor.
- Redogöra för hur synkroniseringstekniker kan användas för att hantera samtidighet i datorsystem, och bedöma deras lämplighet i olika situationer.
- Använda högnivåspråk för att utnyttja concurrency hos flerkärniga system.
- Redogöra för principerna för olika programmeringsmodeller av flerkärniga system, till exempel processer, trädar, meddelandeöverföring och ”software transactional memory”.
- Använda processer, trädar och meddelandeöverföring för programmering av flerkärniga system.
- Presentera och diskutera kursens innehåll muntligt och skriftligt med för utbildningsnivån lämplig färdighet.

**Behörighet:** 60 hp inklusive Imperativ och objektorienterad programmeringsmetodik och Datorarkitektur och digitalteknik, eller motsvarande kunskaper.
# Operating systems

<table>
<thead>
<tr>
<th>Structure</th>
<th>The hardware - software interface</th>
<th>Interrupts - a first look at concurrency</th>
<th>System calls</th>
<th>Stack frames</th>
</tr>
</thead>
</table>

## Processes

### The process concept
- Introduction
- PCB and context switch
- Stack, heap, data and text
- State: new, ready, running, waiting, terminated

### Inter process communication (IPC)
- Overview
- Pipes and file descriptors
- Signals
- Shared memory

## Threads

### Threads vs Processes

### Threads-per-process models

### Thread libraries
- User space vs kernel space
- Pthreads
- Java Threads

### Thread pools

## Resource management

### CPU
- Multiprogramming
  - Introduction
  - Long term: First-Come, First-Served (FCFS), Shortest-Job-First (SJF), Preemptive Shortest-Job-First (PSJF), Priority, Round-Robin (RR)
- Scheduling

### Memory
- Logical vs Physical
- Protection
- Fragmentation
- Paging
  - Shared pages
  - Reentrant code
- Virtual

## Files

## Synchronization

### Software based hardware support (atomic instructions)
- Peterson's solution
- Test And Set (TAS)
- Swap, Compare And Swap (CAS), Modify Compre And Swap (MCAS)

### Higher level abstractions
- Condition variables (condition construct)
- Mutex locks
- Semaphores
- Reader-Writer locks
- Barriers (rendezvous)
- Monitors
Concurrent and concurrent programming

<table>
<thead>
<tr>
<th>Need for synchronization</th>
<th>Atomicity, Consistency and data races</th>
<th>Mutual exclusion (mutex) and critical sections</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problems with synchronization</td>
<td>Deadlock</td>
<td>Starvation</td>
</tr>
<tr>
<td></td>
<td>LiveLock</td>
<td>Priority Inversion</td>
</tr>
<tr>
<td>Classic problems</td>
<td>Dining philosophers</td>
<td>Bounded buffer</td>
</tr>
<tr>
<td></td>
<td>Shared buffer</td>
<td>Readers and writers</td>
</tr>
</tbody>
</table>

Concurrent Models

- Message passing
  - Asynchronous
  - Synchronous
  - Introduction
  - Message passing vs Remote Procedure Call (RPC)
  - Communicating Sequential Processes (CSP)
  - Concurrent ML (CML)

- Sequential
  - Erlang
  - Actors
  - Tone
  - Tail recursive functions
  - Anonymous functions (fun)
  - Lists
  - List comprehensions
  - Guards
  - Conditional execution
  - If and Case

- Concurrent
  - spawn()
  - Send and Receive
  - Process supervision
  - spawn_link()
  - Spawn supervision
  - UUnit
  - Virtual machine Guest lecture (Pattrick Nykolen?)

Software transactional memory

- Deadlock-free

- Shared associative memory
  - Tuple space
  - Linda
  - Blackboard

Introduction

- Cooperative
  - Coroutines
  - Future values
  - Lazy evaluation

Multitasking

- Preemptive
  - Shared memory
  - Threads
  - Non-shared memory
  - Processes

Architecture patterns

- Producer - Consumer
- Client - Server
- Pipe and Filter
- MapReduce
- Publisher - Subscriber