Two lecturers

- Arnaud Legrand, CNRS, INRIA, UGA, POLARIS team
- Bruno Raffin, INRIA, UGA, DATAMOVE team

Eleven 3-hours lectures + 3-hours for reviewing previous exams

Web page with all practical information, syllabus, etc:

- http://mescal.imag.fr/membres/arnaud.legrand/teaching/2016/M2R_PC.php

To email us:

mailto:arnaud.legrand@imag.fr mailto:bruno.raffin@inria.fr

We will also set up a mailing list

- The basic requirements are Operating Systems, Networking and Algorithms.
- The content of this lecture is very **dense** and is intended to give you a broad overview of this area
- The slides comprise all the material you need. We will give you some extra pointers when needed.

- Many of the comments we do are very general and will be enlightening only if you spend time trying to figure out the whole picture. Ask yourselves what are the main messages of the lectures.
- You cannot reasonably expect to have understood everything at the end of the slides

  1 hour of lecture = at least 1 hour of personal work to re-read and understand the corresponding slides

- This is an interactive class not a projection of slides. Fell free to ask questions, make comments, before, during, after the class.
- You will have to do some parallel programming (MPI) to get some practical experience.

The M2R is not an exam. It is a contest to pursue a PhD.

There are few grants. You work to prepare yourself to a career in research.

- The Performance Evaluation lecture is important.
- The list of internship proposals is here:

  http://im2ag-pcarre.e.ujf-grenoble.fr/
- We will give you a brief presentation of three Grenoble teams wokring on HPC: **POLARIS**, **DATAMOVE** and **CORSE**.

- This class is about Parallel Systems with a specific focus on High Performance Computing, i.e. parallel computing from large scale numerical simulations.
- It is distincts from Cloud Architectures and Big Data Analytics, even if most of what we talk about also applies to these domains.
- We will cover some Big Data Analytics from a parallel system point of view in the last lectures.

Parallel Architectures

Arnaud Legrand, CNRS, UGA Bruno Raffin, INRIA, UGA

LIG lab - CNRS/INRIA , arnaud.legrand@imag.fr,bruno.raffin@inria.fr

September 30, 2016

Outline

Outline

Parallel Computing is difficult

- Your computer is to slow to solve your problem. Use 10, 100, 1000, 1 000 000 to run it faster.
- But them, these computers need to coordinate to do the expected job.
- It's actually not as easy has it may sound at first. You need to be carefull about concurrency issues (concurrent R/W and W/W, deadlocks, livelocks, load balancing).
- Automatic parallelization never reached the necessary level to relieve the programmer from parallelization effort.
- Why ? Often the best parallel algorithm is not just a parallelization of the best sequential algorithm, but a very different algorithm.
- Parallel computing is almost has old as computer science.

Machines are complex

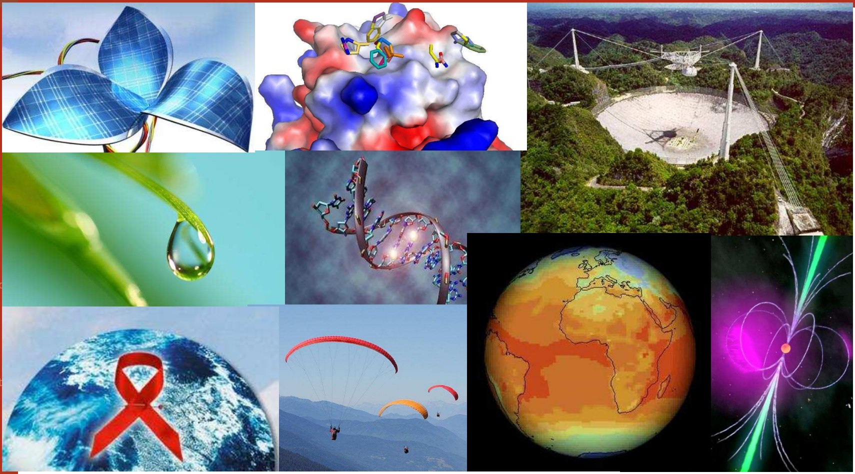
- A parallel computer is made of compute units, a memory hierarchy, an interconnection network and a permanent storage.
- A basic taxonomy of parallel machines:
  - Distributed memory parallel machine (PC cluster): each compute unit has its own address space. Communications are explicit (send/receive)
  - Shared memory parallel machine (a multi-core processor): the compute units share the same adress aspace. Communications are implicit (locks, mutex, barriers)
  - Synchronous compute units (SIMD - Single Instruction Multiple Data) (threads in a GPU)
  - Asynchronous compute units (MIMD - Multiple Instructions Multiple Data)
- First supercomputers where vector machines (Cray): SIMD + shared address space.
- Today supercomputers are a mix of all this and are very complex.
- If you want to know about the fastest (non confidential) machines look at http://www.top500.org

Parallel Programming

- For most of you, your experience with parallel (actually concurrent) programming is limited to playing with a few Java threads
- But direct thread programming does not scale.
- Actually parallel programming is a very difficult issue. Taught in advanced class only. No standard language like C or Java integrate parallel constructs so far.
- During this class we will cover different parallel programming paradigms like message passing (MPI), task programming (Cilk, TBB, OpenMP), GPU programming or some Domain Specific Languages (DSL) like map/reduce.
- Writing an efficient parallel code requires to have a good understanding of the machine architecture. That's why in this class we will talk about parallel computer architectures (cache hierarchies, network topologies,...)

Outline

What do . . . have in common?

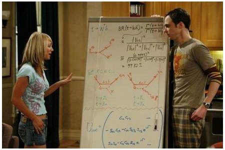


Clean water, solar cells, new drugs against Ebola/AIDS/Cancer, climate evolution, weather forecast for paragliding, searching for Extra-Terrestrial

Computer Technology and other sciences

Pencil and paper alone cannot solve all our problems. Computer can be used be used as a scientific instrument.

Computer technology has brought us a two new scientific paradigms:



The Big Bang Theory

Big Data

- Dig huge amounts of data (sensors, transaction records, genome and protein databanks,. . .)
- Enables to discover phenomena or truths that would otherwise remain unseen

Computational Science

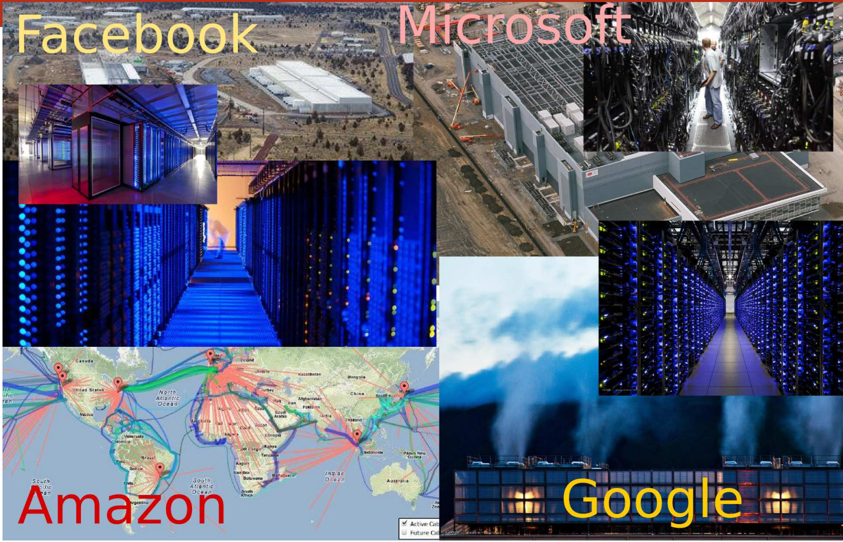
- Performing real experiment is very costly and even sometimes simply impossible
- Allows to explore and investigate designs or phenomena in a few hours instead of years

- Motivated the development of major computational infrastructures

Supercomputers

- 100,000 to 1,000,000 cores with accelerators (GPU, Xeon Phi) and a high throughput/low latency interconnection network
- An international race (Top500)

The Cloud



A Breathtaking Evolution

Hybrid and very large scale parallel architectures to answer computation needs in restricted power envelopes.

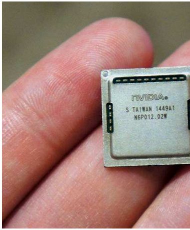
1996

ASCI Red
1 Teraflop
9298 Pentium II
1 000 Flops/W

2009

ATI Radeon
2.4 Teraflop
1600 Stream Processors
1 600 000 Flops/W

2015



Nvidia Tegra
1 Teraflo
8-core ARM
667 000 000 Fl

My smartphone is as powerful as a 20 years old supercomputer

Parallelism for Killer Applications

Our unsatisfied appetite has always been answered by aggregating several (dozens, thousands or millions depending on the context and the decade) processing units with a more or less implicit communication network.

This domain is known under various names:

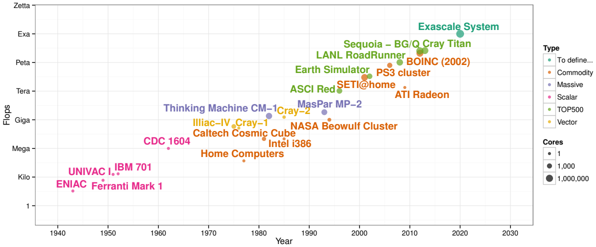
- parallel computing
- *distributed* computing
- High Performance Computing
- supercomputing

and more recently as

- grid computing
- ambiant computing
- cloud computing
- sky computing, . . .

Although parallelism is now everywhere, it has known several up and downs. . .

A Journey Through Time



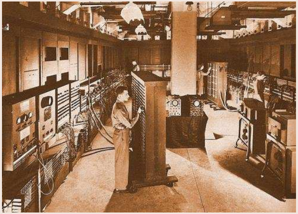
1943: the early days

ENIAC, **35 Flops**!

Designed to compute artillery firing tables

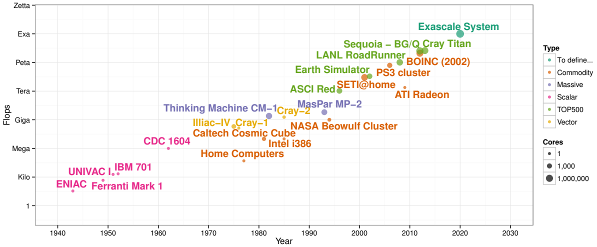
Approx $6,000,000 today

"It was possible to connect several accumulators to run simultaneously, so the peak speed of operation was potentially much higher due to parallel operation."



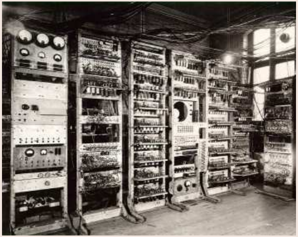
ENIAC

A Journey Through Time

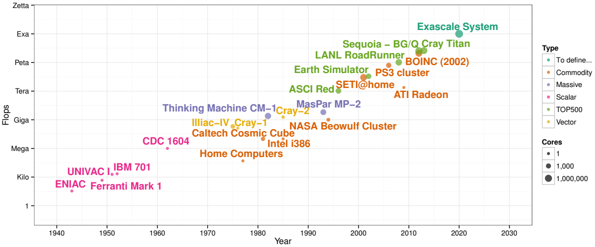


1949: the early days

Manchester Mark 1.
One of the world's first stored-program computers.
Ran Mersene Prime search error-free for 9 hours!


Manchester Mark 1

A Journey Through Time

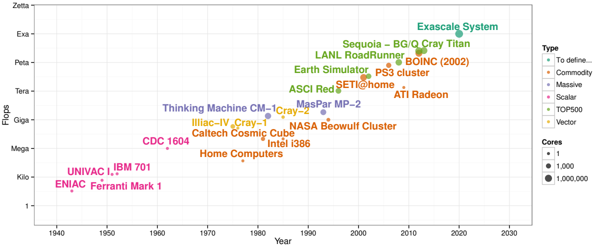


1951: a new market ?

- **Ferranti Mark 1.** world's first commercially available general-purpose electronic computer. **460 Flops**.
- **UNIVAC I** (Universal Automatic Computer) was delivered to the U.S. Census Bureau. The fifth machine (built for the U.S. Atomic Energy Commission) was used by CBS to predict the result of the 1952 presidential election.

  Remington Rand eventually sold 46 machines at more than \$1 million each (\$8.95 million as of 2012). UNIVAC was the first "mass produced" computer. **1,905 Flops**.

A Journey Through Time



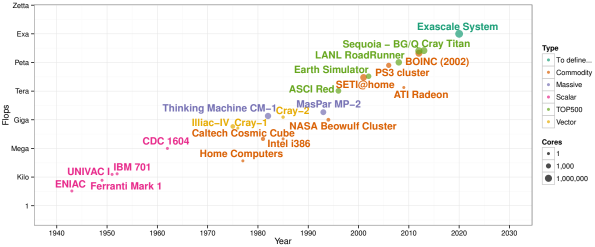
1952: a new market!

IBM 701 (aka Defense Calculator) is IBM first's commercial scientific computer. **2,200 FLOPS**. Rental charge was about $12,000 a month.

"I think there is a world market for maybe five computers"
– Thomas Watson Jr

Watson visited 20 companies that were potential customers:
"as a result of our trip, on which we expected to get orders for five machines, we came home with orders for 18."

A Journey Through Time



1962: Control Data Corporation

CDC delivers first **CDC 1604** to US Navy.

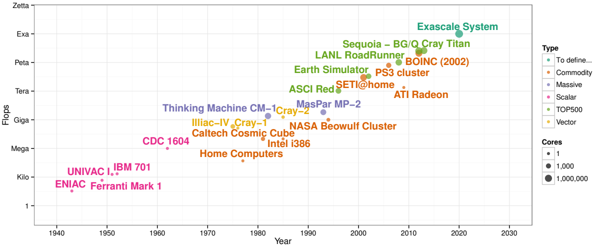
First commercially successful **transistorized computer**.

Designed by **Seymour Cray** and his team.

One processor, 48 bit words and a 6 microsec memory cycle time, **0.1MFLOPS**.

CDC 1604

A Journey Through Time

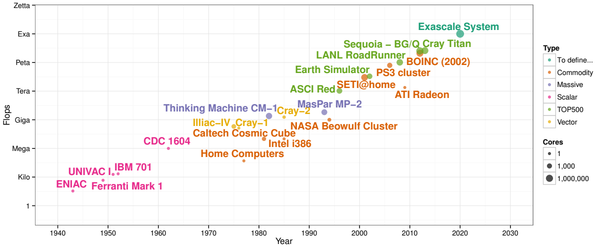


1966–1975: The Illiac-IV

Illiac-IV for NASA.
A linear array of 256 64-bit Processing Elements.
Expected 1 GFlops but reached only **200 MFlops**.
Was somehow the precursor of **vector processing**.

ILLIAC IV

A Journey Through Time



1970–1977: micro-computers

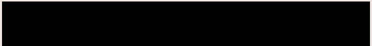
1970 Datapoint 2200

1921 Intel 4004

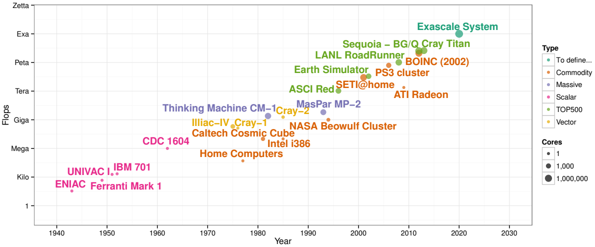
1972 Intel 8008

1972 Micral-N

1977 Second generation: home computers



A Journey Through Time



1976–1995: Massive paralelism

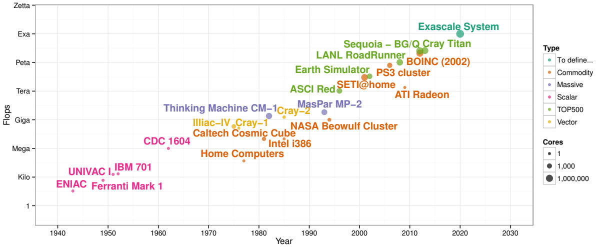
1982 Thinking Machines' **CM-1**, 65,536 1-bit processing elements interconnected as a 12D hypercube. **2,500** MFlops

1995 MasPar **MP-2**. 16,384 proprietary 32 bits processors **6,225** MFlops

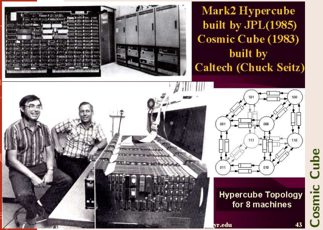
1994-1997 Cray T3D. 128 processors **19,200** MFlops

Connexion Machine-1

A Journey Through Time



1976–1995: commodity hardware. DIY!



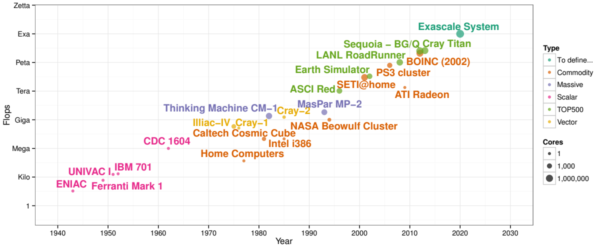
1981 **Caltech's Cosmic Cube**, 64-node hypercube based on Intel 8086 + 8087, **10 MFlops**

1985 Intel **i386**

1994 NASA's **Beowulf Cluster**. 16 Intel PCs + Ethernet

1,000 MFlops for $50,000

A Journey Through Time



1996–...: distributed/volunteer computing

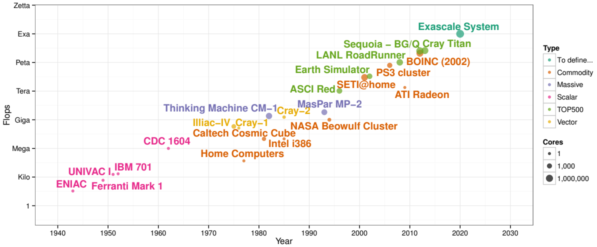
1996 GIMPS

1999 SETI@home: **27.32 TFlops** in 2002 with 300,000 hosts

2000 Folding@home

2002 BOINC: **9.2PFlops** in 2012 with 596,224 active hosts

A Journey Through Time



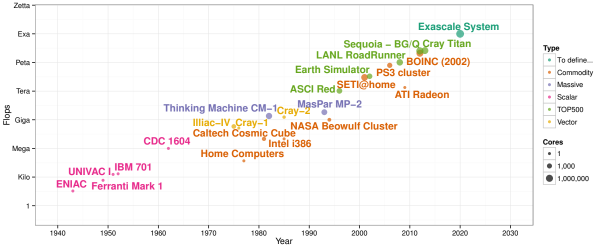
1996–...: Top500 "commodity" hardware

1996-2001 ASCI Red: **1.06TFlops** with 9,298 Pentium Pro

2002 Earth Simulator: **35.9TFlops** with 640 nodes with eight vector processors (5120)

ASCI Red

A Journey Through Time



1996–...: commodity hardware

Clusters Off-the-shelf processors, high-speed networks (SCI, myrinet, Quadrics, ...)

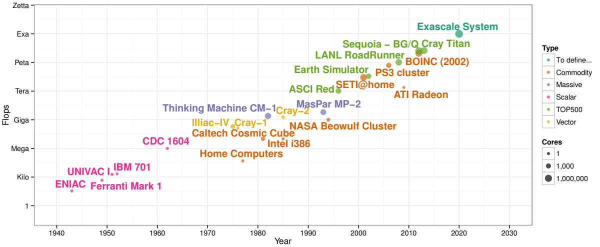
2006 1760 **PS3**. **500 TFLops**

2009 ATI **Radeon**. **2.4 TFlops**

2012 **Xeon-Phi**. x86-compatible **1 TFlops**

ATI Radeon HD 4870X2

A Journey Through Time



2012–2013: Peta-scale systems

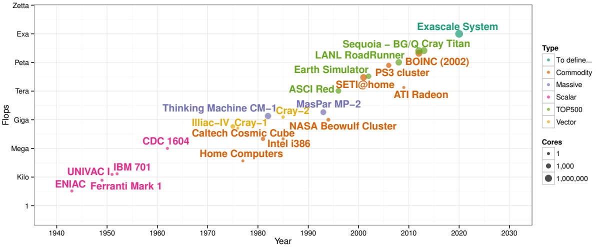
2012 **Sequoia** - BlueGene/Q. 98,304 16-core (1,572,864) Power processors.

16,320,000,000,000,000 FLOPS (**16.32 PFlops**)

Nuclear weapons simulation mainly but also astronomy, energy, human genome, climate change. **7890.0 kW**

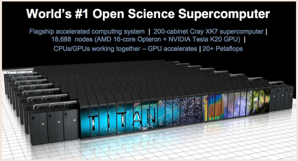
Sequoia

A Journey Through Time



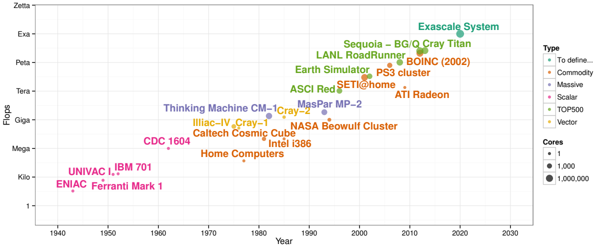
2012–2013: Peta-scale systems

2013 Cray **Titan** (562,960 AMD cores + Nvidia GPUs). (**17.59 PFlops**)



Titan

A Journey Through Time

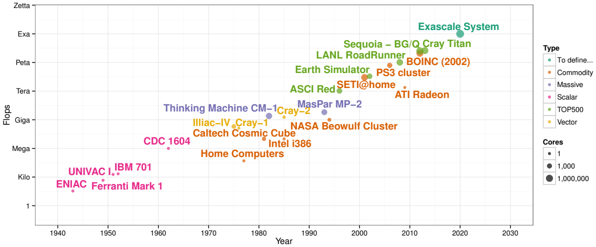


2012–2013: Peta-scale systems

2013 **Tianhe-2** 32,000 Ivy Bridge + 48,000 Xeon Phi, **30.65 PFlops**, "3,120,000 cores" **17,800 kW**

Titan

A Journey Through Time



2020–...: Exa-scale systems

One Exaflops is expected in 2020.

Based on a 20 MW power budget, this requires an efficiency of 50 GFLOPS/Watt. Current leader achieves around 7.0 GFLOPS / Watt

- GPU-based?
- ARM-based (Mont-blanc project)?
- Interconnect ?
- Failure management, speculative execution, communication overlap ?

Recap

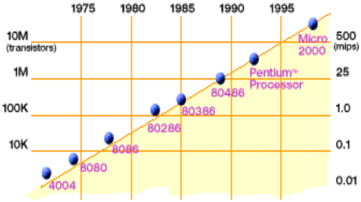
In this area Research, Technology, and Mass production are tightly connected

- Most companies died
- Research ideas make their way to mass production
  - vector processors, accelerators
  - pipelining
  - instruction level parallelism
  - multi-threading
- Some research ideas did not make their way because technology was not ready...
- ...or because there was no market for mass production
- Mass production influences the way research is done

All computers are parallel

Outline

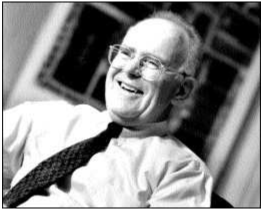
Technology Trends: Microprocessor Capacity



2X transistors/Chip Every 1.5 years

Called "Moore's Law"

Microprocessors have become smaller, denser, and more powerful.



Gordon Moore (co-founder of Intel) predicted in 1965 that the transistor density of semiconductor chips would double roughly every 18 months.

Slide source: Jack Dongarra

Courtesy of Jean-François Méhaut

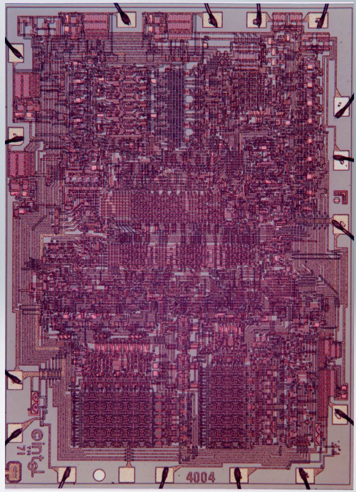
MOORE'S LAW

Sunday, 24 January 2010

Courtesy of Jez Wain (BULL)

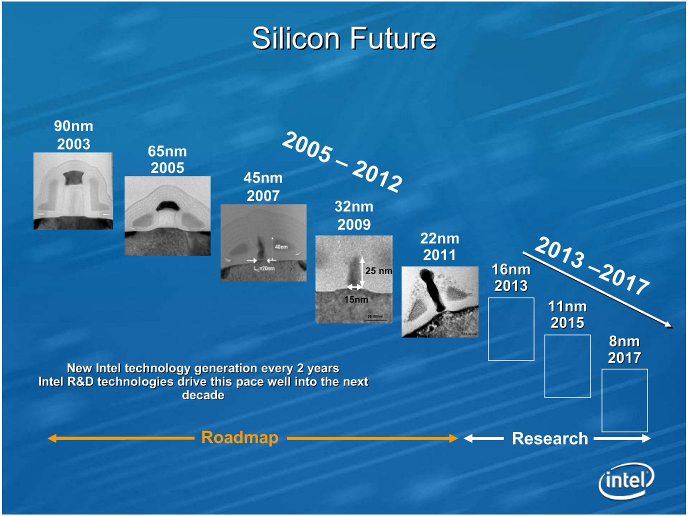
1971: INTEL 4004

With today's technology could place 15 complete processors on each transistor of the original



Courtesy of Jez Wain (BULL)

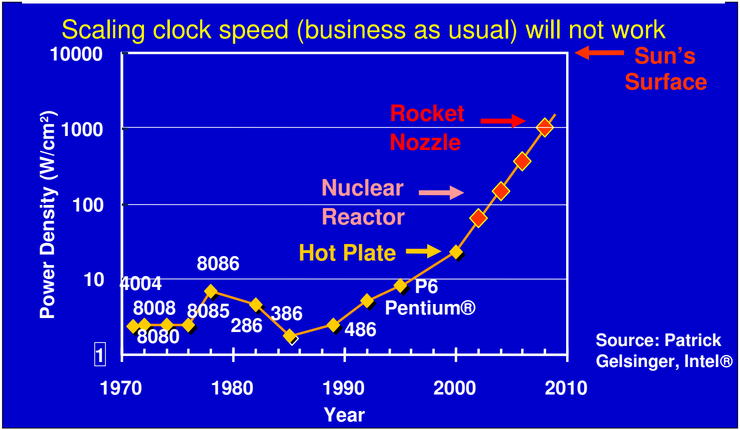
Sunday, 24 January 2010



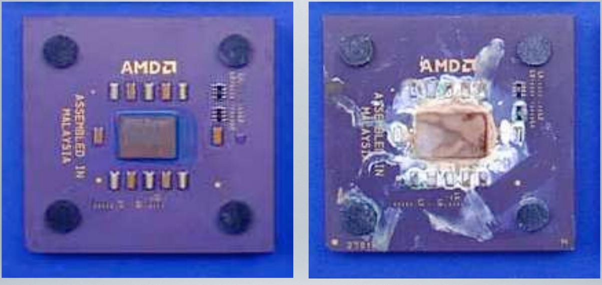
Courtesy of Intel

Limit #1: Power density

Can soon put more transistors on a chip than can afford to turn on.
-- Patterson '07



Courtesy of Jean-François Méhaut



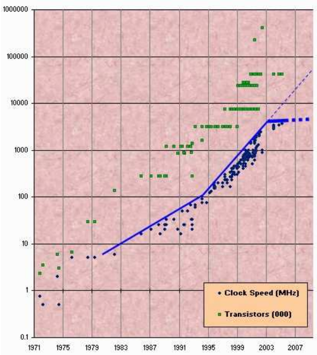
Temperature $\nearrow$ $\Rrightarrow$ Leakage $\nearrow$

Sunday, 24 January 2010

Courtesy of Jez Wain (BULL)

Moore's Law

- Many people interpret Moore's law as "computer gets twice as fast every 18/24 months"
  - which is not true
  - The law is about transistor density
- This wrong interpretation is no longer true
- We should have 20GHz processors right now
- And we don't!



Courtesy of Henri Casanova

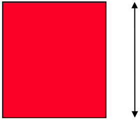
Limit #2: Hidden Parallelism Tapped Out

- **Superscalar (SS) designs were the state of the art; many forms of parallelism not visible to programmer**
  - **multiple instruction issue**
  - **dynamic scheduling: hardware discovers parallelism between instructions**
  - **speculative execution: look past predicted branches**
  - **non-blocking caches: multiple outstanding memory ops**
- **You may have heard of these in 61C, but you haven't needed to know about them to write software**
- **Unfortunately, these sources have been used up**

Courtesy of Jean-François Méhaut

Limit #3: Speed of Light (Fundamental)

1 Tflop/s, 1 Tbyte sequential machine



r = 0.3 mm

- Consider the 1 Tflop/s sequential machine:
  - Data must travel some distance, r, to get from memory to CPU.
  - To get 1 data element per cycle, this means 10^{12} times per second at the speed of light, $c = 3 \times 10^8$ m/s. Thus $r < c/10^{12} = 0.3$ mm.
- Now put 1 Tbyte of storage in a 0.3 mm x 0.3 mm area:
  - Each bit occupies about 1 square Angstrom, or the size of a small atom.
- No choice but parallelism

Courtesy of Jean-François Méhaut

No more Moore?

- We are used to getting faster CPUs all the time
- We are used for them to keep up with more demanding software
- Known as "Andy giveth, and Bill taketh away"
  - Andy Grove
  - Bill Gates
- It's a nice way to force people to buy computers often
- But basically, our computers get better, do more things, and it just happens automatically
- Some people call this the "performance free lunch"
- **Conventional wisdom:** "Not to worry, tomorrow's processors will have even more throughput, and anyway today's applications are increasingly throttled by factors other than CPU throughput and memory speed (e.g.,

Courtesy of Henri Casanova

Commodity improvements

- There are three main ways in which commodity processors keep improving:
  - Higher clock rate
  - More aggressive instruction reordering and concurrent units
  - Bigger/faster caches
- All applications can easily benefit from these improvements
  - at the cost of perhaps a recompilation
- Unfortunately, the first two are hitting their limit
  - Higher clock rate lead to high heat, power consumption
  - No more instruction reordering without compromising correctness

Courtesy of Henri Casanova

Is Moore's laws not true?

- Ironically, Moore's law is still true
  - The density indeed still doubles
- But its wrong interpretation is not
  - Clock rates do not doubled any more
- But we can't let this happen: computers **have** to get more powerful
- Therefore, the industry has thought of new ways to improve them: multi-core
  - Multiple CPUs on a single **chip**
- Multi-core adds another level of concurrency
  - But unlike, say multiple functional units, hard to compile for them
  - Therefore, programmers need to be trained to develop code for multi-core platforms
    - See ICS432

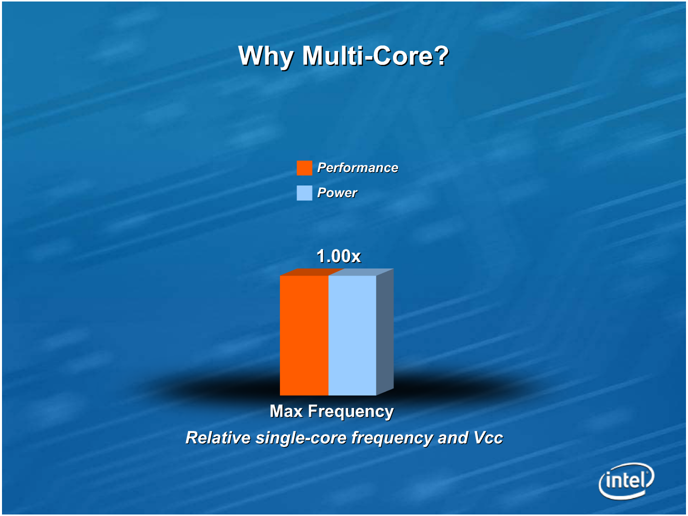
Courtesy of Henri Casanova

Parallelism Saves Power

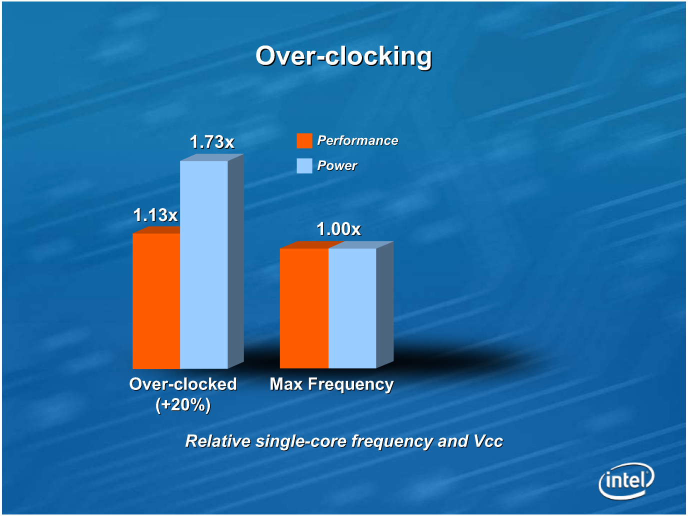
- Exploit explicit parallelism for reducing power
  - Intel Slides

- **Using additional cores**
  - Increase density (= more transistors = more capacitance)
  - Can increase cores (2x) and performance (2x)
  - Or increase cores (2x), but decrease frequency (1/2): same performance at ¼ the power
- **Additional benefits**
  - Small/simple cores → more predictable performance

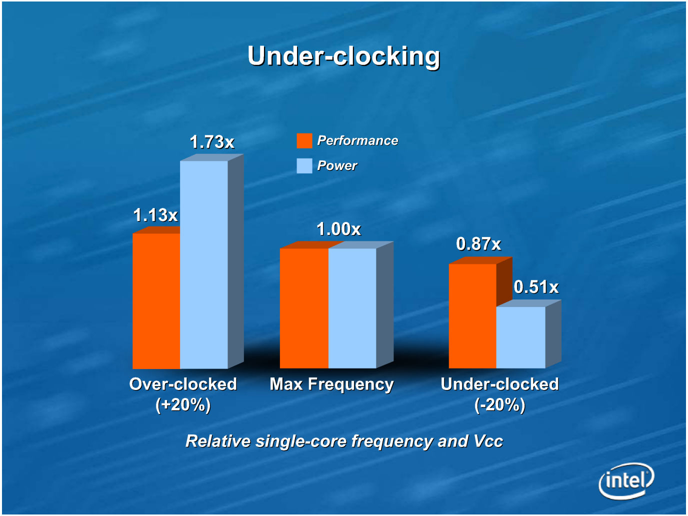
Courtesy of Jean-François Méhaut



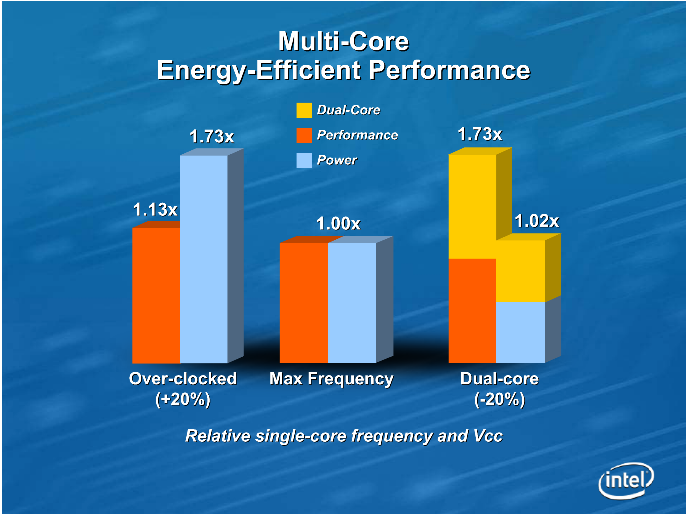
Courtesy of Intel



Courtesy of Intel



Courtesy of Intel



Courtesy of Intel

Parallelism Saves Power

- Exploit explicit parallelism for reducing power
  - Intel Slides

- **Using additional cores**
  - Increase density (= more transistors = more capacitance)
  - Can increase cores (2x) and performance (2x)
  - Or increase cores (2x), but decrease frequency (1/2): same performance at ¼ the power
- **Additional benefits**
  - Small/simple cores → more predictable performance

Courtesy of Jean-François Méhaut

EVOLUTION:TERAFLOP 1996

Sunday, 24 January 2010

Courtesy of Jez Wain (BULL)

1,000 FLOPS PER WATT

Sunday, 24 January 2010

Courtesy of Jez Wain (BULL)

EVOLUTION: 2.4 TERAFLOPS 2009

Sunday, 24 January 2010

Courtesy of Jez Wain (BULL)

1,600,000 FLOPS PER WATT

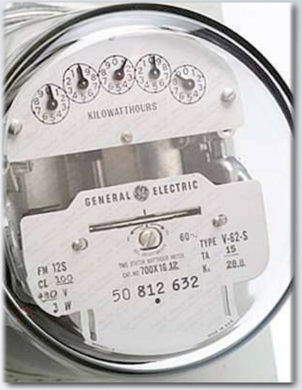
Sunday, 24 January 2010

Courtesy of Jez Wain (BULL)

DATA-CENTRES 2007

200B kWh

$29B in power and cooling



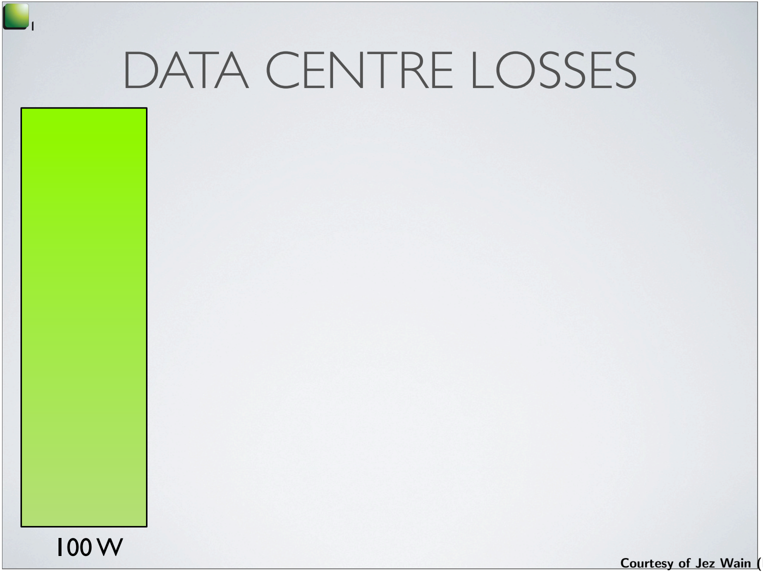
1% of world's electricity goes to cooling IT

Courtesy of Jez Wain (BULL)

IT: 2% OF WORLD CO_2

Sunday, 24 January 2010

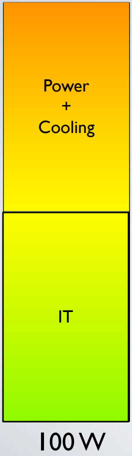
Courtesy of Jez Wain (BULL)



Sunday, 24 January 2010

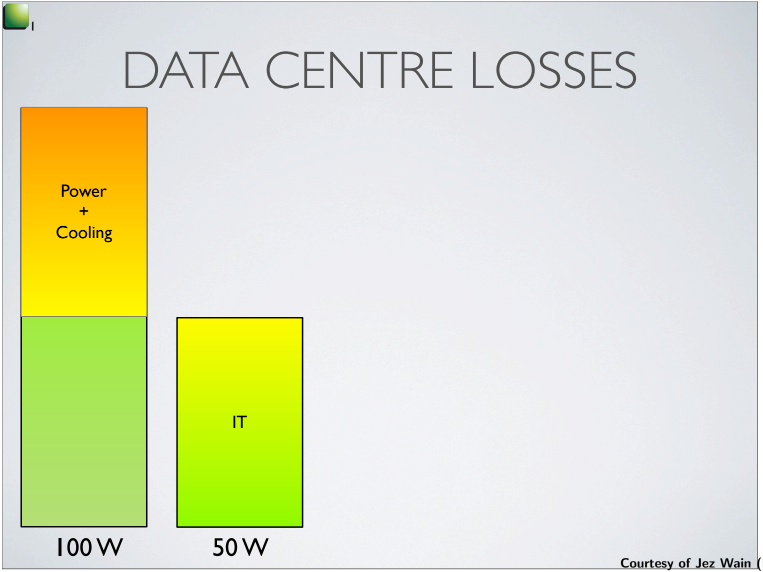
Courtesy of Jez Wain (BULL)

DATA CENTRE LOSSES



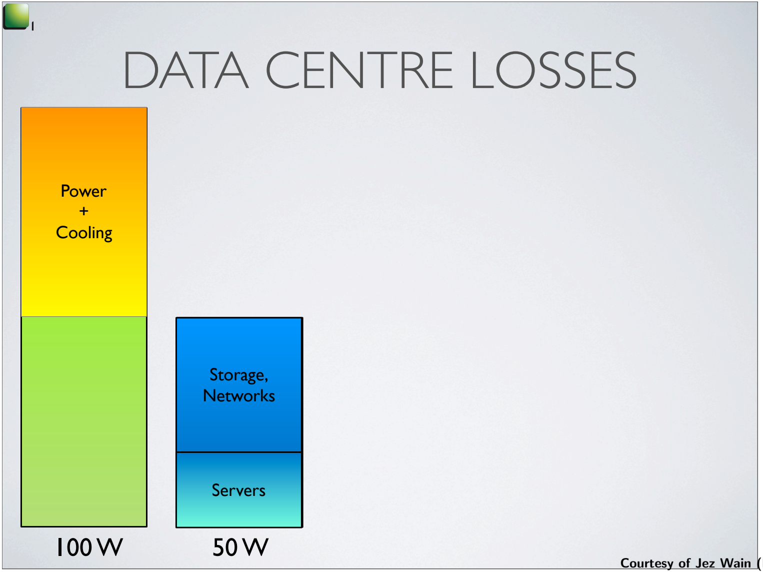
Sunday, 24 January 2010

Courtesy of Jez Wain (BULL)



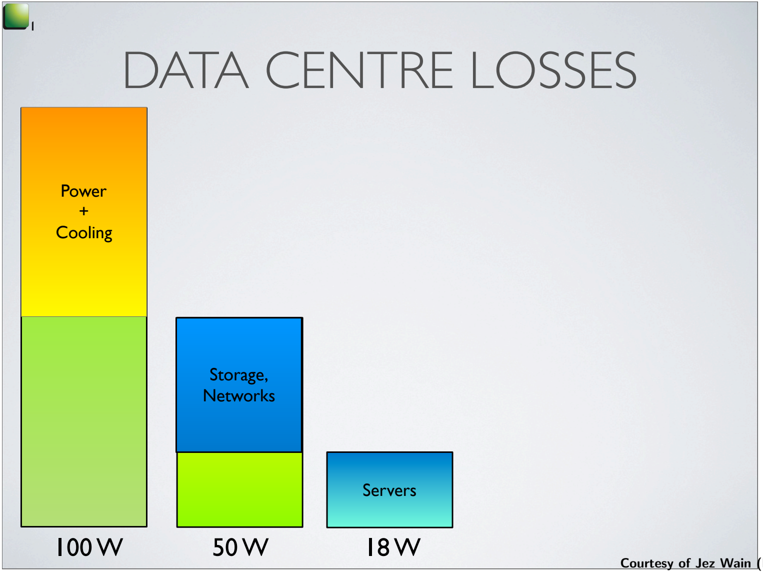
Sunday, 24 January 2010

Courtesy of Jez Wain (BULL)



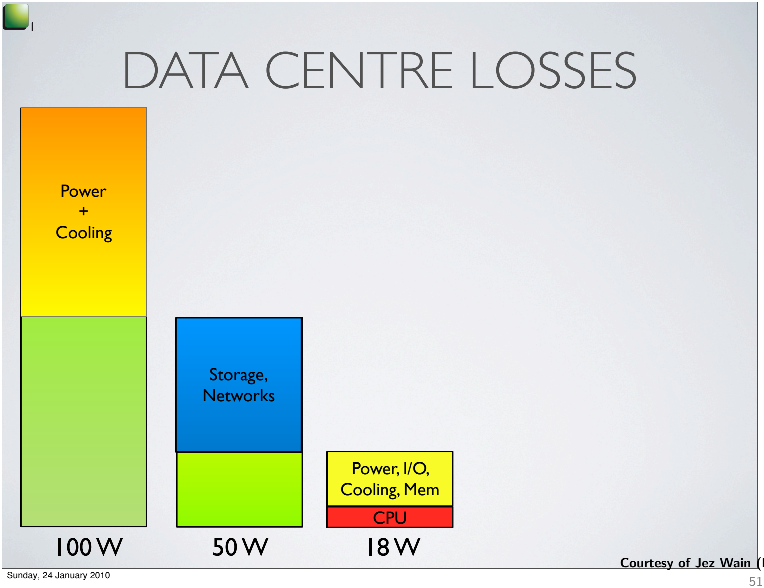
Sunday, 24 January 2010

Courtesy of Jez Wain (BULL)

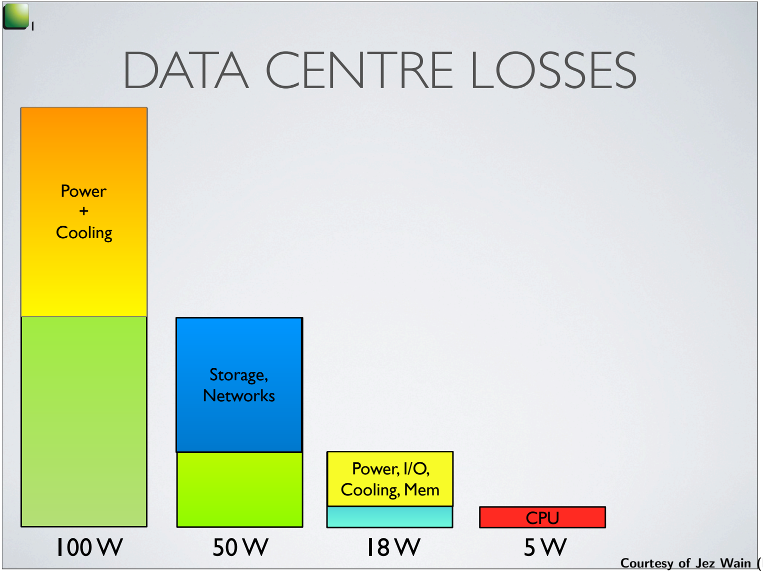


Sunday, 24 January 2010

Courtesy of Jez Wain (BULL)

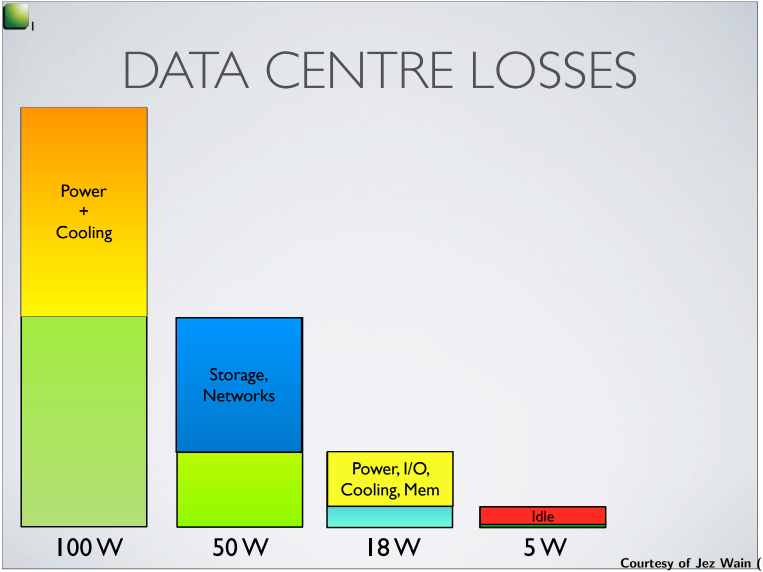


Courtesy of Jez Wain (BULL)



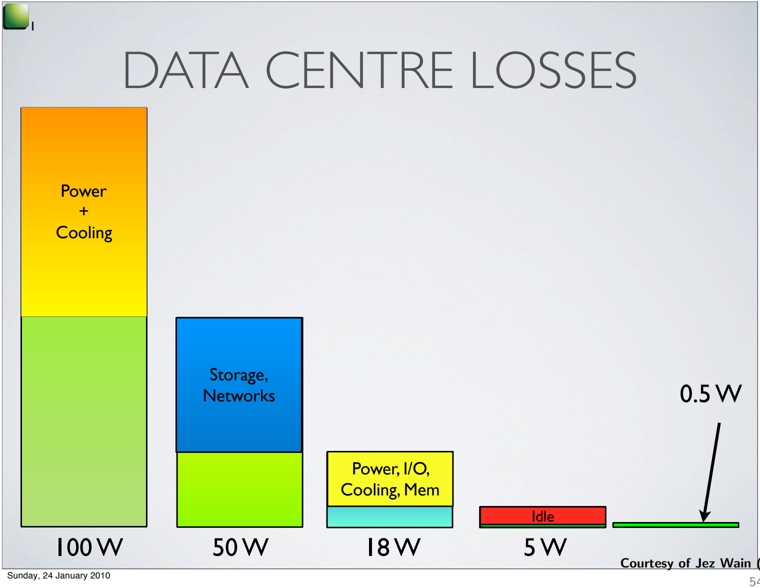
Sunday, 24 January 2010

Courtesy of Jez Wain (BULL)

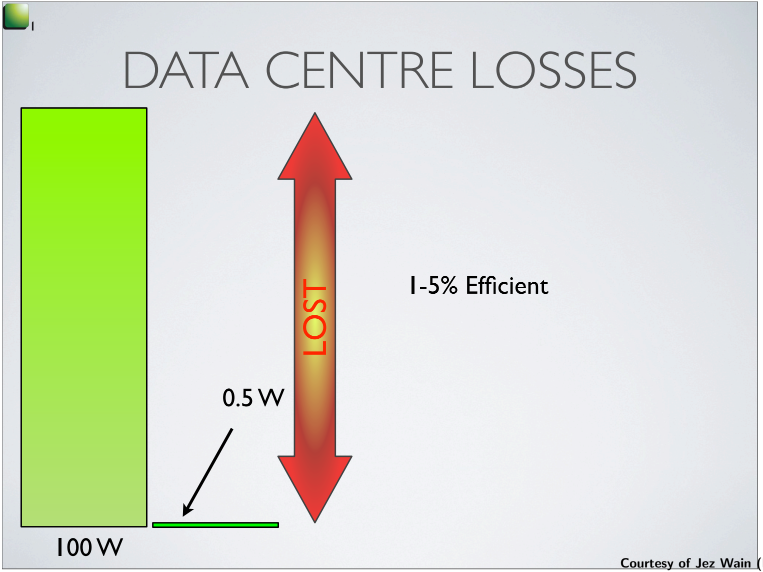


Sunday, 24 January 2010

Courtesy of Jez Wain (BULL)



Courtesy of Jez Wain (BULL)



Sunday, 24 January 2010

Courtesy of Jez Wain (BULL)

DATA CENTRE EFFICIENCY

1-5% Efficient

Sunday, 24 January 2010

Courtesy of Jez Wain (BULL)

DATA CENTRE EFFICIENCY

1-5% Efficient

10-15% Efficient

Sunday, 24 January 2010

Courtesy of Jez Wain (BULL)

The Memory Bottleneck

- The memory is a very common bottleneck that beginning programmers often don't think about
  - When you look at code, you often pay more attention to computation
  - a[i] = b[j] + c[k]
    - The access to the 3 arrays take more time than doing an addition
    - For the code above, the memory is the bottleneck for many machines!

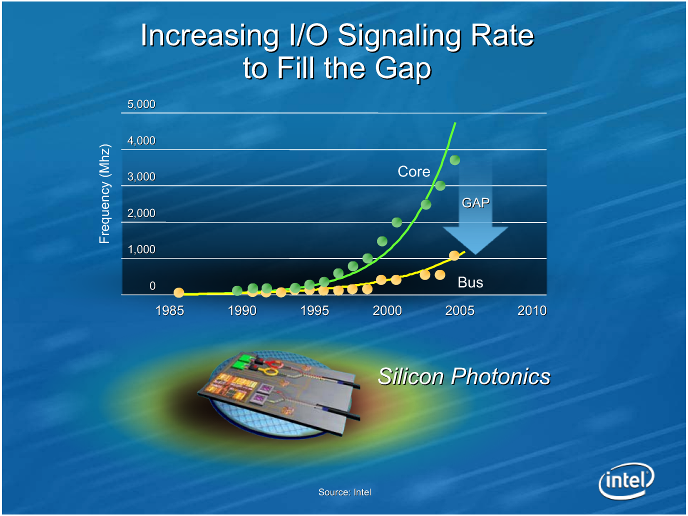
Courtesy of Henri Casanova

Why the Memory Bottleneck?

- In the 70's, everything was balanced
  - The memory kept pace with the CPU
    - n cycles to execute an instruction, n cycles to bring in a word from memory
- No longer true
  - CPUs have gotten 1,000x faster
  - Memory have gotten 10x faster and 1,000,000x larger

➔ Flops are free and bandwidth is expensive and processors are STARVED for data

Courtesy of Henri Casanova



Courtesy of Intel

Current Memory Technology

Memory	Latency	Peak Bandwidth
DDR400 SDRAM	10 ns	6.4 GB/sec
DDR533 SDRAM	9.4 ns	8.5 GB/sec
DDR2-533 SDRAM	11.2 ns	8.5 GB/sec
DDR2-600 SDRAM	13.3 ns	9.6 GB/sec
DDR2-667 SDRAM	???	10.6 GB/sec
DDR2-800 SDRAM	???	12.8 GB/sec

source: http://www.xbitlabs.com/articles/memory/display/ddr2-ddr_2.html

Courtesy of Henri Casanova

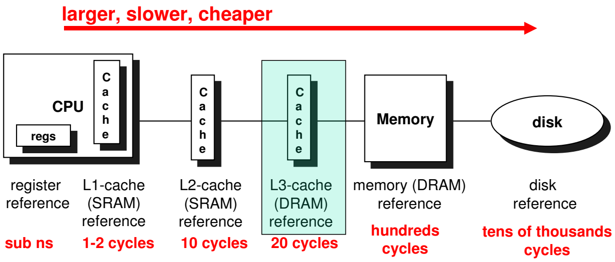
Memory Bottleneck: Example

- Fragment of code: a[i] = b[j] + c[k]
  - Three memory references: 2 reads, 1 write
  - One addition: can be done in one cycle
- If the memory bandwidth is 12.8GB/sec, then the rate at which the processor can access integers (4 bytes) is: 12.8*1024*1024*1024 / 4 = 3.4GHz
- The above code needs to access 3 integers
- Therefore, the rate at which the code gets its data is ~ 1.1GHz
- But the CPU could perform additions at 4GHz!
- Therefore: The memory is the bottleneck
  - And we assumed memory worked at the peak!!!
  - We ignored other possible overheads on the bus
  - In practice the gap can be around a factor 15 or higher

Courtesy of Henri Casanova

Reducing the Memory Bottleneck

- The way in which computer architects have dealt with the memory bottleneck is via the memory hierarchy



Courtesy of Henri Casanova

Locality

- The memory hierarchy is useful because of "locality"
- Temporal locality: a memory location that was referenced in the past is likely to be referenced again
- Spatial locality: a memory location next to one that was referenced in the past is likely to be referenced in the near future
- This is great, but what we write our code for performance we want our code to have the maximum amount of locality
  - The compiler can do some work for us regarding locality
  - But unfortunately not everything

Courtesy of Henri Casanova

Programming for Locality

- Essentially, a programmer should keep a mental picture of the memory layout of the application, and reason about locality
  - When writing concurrent code on a multi-core architecture, one must also thing of which caches are shared/private
- This can be extremely complex, but there are a few well-known techniques
- The typical example is with 2-D arrays

Courtesy of Henri Casanova

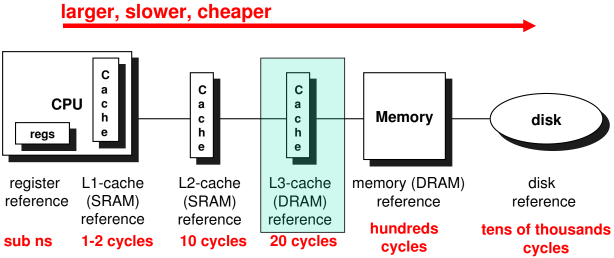
Memory Bottleneck: Example

- Fragment of code: a[i] = b[j] + c[k]
  - Three memory references: 2 reads, 1 write
  - One addition: can be done in one cycle
- If the memory bandwidth is 12.8GB/sec, then the rate at which the processor can access integers (4 bytes) is: 12.8*1024*1024*1024 / 4 = 3.4GHz
- The above code needs to access 3 integers
- Therefore, the rate at which the code gets its data is ~ 1.1GHz
- But the CPU could perform additions at 4GHz!
- Therefore: The memory is the bottleneck
  - And we assumed memory worked at the peak!!!
  - We ignored other possible overheads on the bus
  - In practice the gap can be around a factor 15 or higher

Courtesy of Henri Casanova

Reducing the Memory Bottleneck

- The way in which computer architects have dealt with the memory bottleneck is via the memory hierarchy



Courtesy of Henri Casanova

Locality

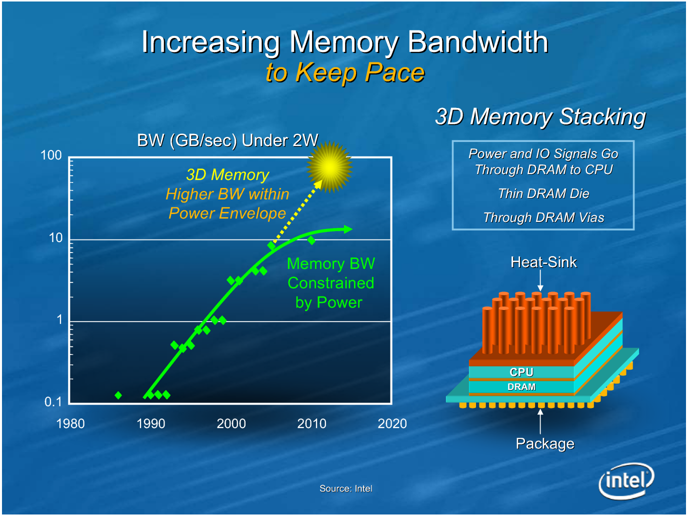
- The memory hierarchy is useful because of "locality"
- Temporal locality: a memory location that was referenced in the past is likely to be referenced again
- Spatial locality: a memory location next to one that was referenced in the past is likely to be referenced in the near future
- This is great, but what we write our code for performance we want our code to have the maximum amount of locality
  - The compiler can do some work for us regarding locality
  - But unfortunately not everything

Courtesy of Henri Casanova

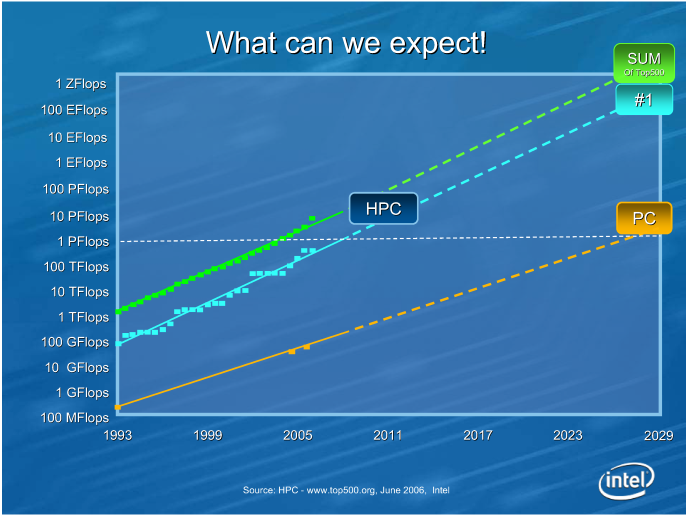
Programming for Locality

- Essentially, a programmer should keep a mental picture of the memory layout of the application, and reason about locality
  - When writing concurrent code on a multi-core architecture, one must also thing of which caches are shared/private
- This can be extremely complex, but there are a few well-known techniques
- The typical example is with 2-D arrays

Courtesy of Henri Casanova

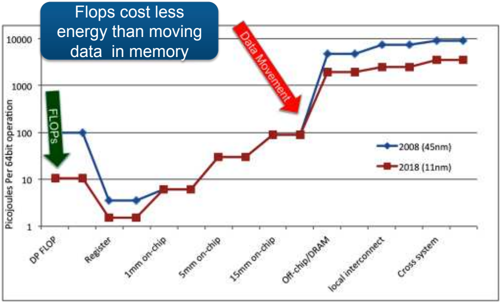


Courtesy of Intel



Courtesy of Intel

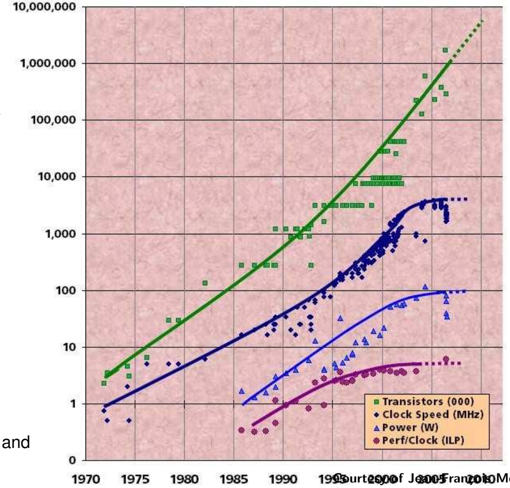
The Data Challenge



K. Yelick, "Software and Algorithms for Exascale: Ten Ways to Waste an Exascale Computer"

Revolution is Happening Now

- Chip density is continuing increase ~2x every 2 years
  - Clock speed is not
  - Number of processor cores may double instead
- There is little or no hidden parallelism (ILP) to be found
- Parallelism must be exposed to and managed by software



Source: Intel, Microsoft (Sutter) and Stanford (Olukotun, Hammond)

Courtesy of Jean-François Méhaut

Multicore in Products

- "We are dedicating all of our future product development to multicore designs. … This is a sea change in computing"

  Paul Otellini, President, Intel (2005)

- All microprocessor companies switch to MP (2X CPUs / 2 yrs)
  ⇒ Procrastination penalized: 2X sequential perf. / 5 yrs

Manufacturer/Year	AMD/'05	Intel/'06	IBM/'04	Sun/'07
Processors/chip	2	2	2	8
Threads/Processor	1	2	2	16
Threads/chip	2	4	4	128

And at the same time,

- The STI Cell processor (PS3) has 8 cores
- The latest NVidia Graphics Processing Unit (GPU) has 128 cores
- Intel has demonstrated the TeraScale processor (80-core), research chip

Courtesy of Jean-François Méhaut

Moore's Law still holds but we are limited by the law of physics.

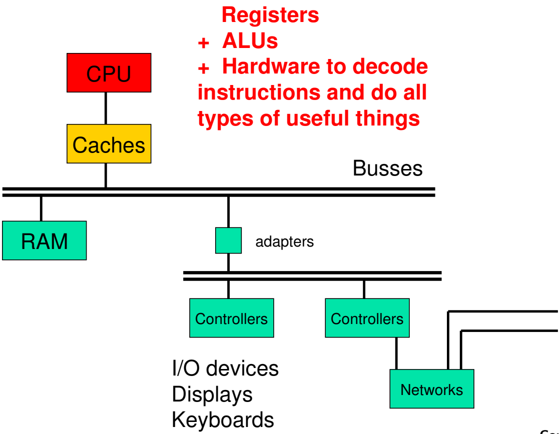
- With a single CPU, the speed of light will keep us away from TeraFlops.
- Increasing clock rate is bad (higher energy consumption, higher temperature $\rightsquigarrow$ need for cooling and thus even higher energy consumption).
- Automatic concurrency inside CPU is already there without you even noticing it. Don't expect too much on this side.

To improve performances:

- We need many different computation units.
  - Yet, INTEL doesn't see the power-of-2 doubling of number of cores every 2 years or so (will work on improving socket architecture, cache, registers, instructions, ...)
  - the biggest challenge is keeping the reasonable balance we have today between memory bandwidth and flops
- Data need to be close to computation units and well managed.
- We need to expose parallelism and program with such architectures in mind.
- We need to keep the architecture in mind when designing algo-

Outline

Concurrency within a CPU



Courtesy of Henri Casanova

Concurrency within a CPU

- Several techniques to allow concurrency within a single CPU
  - Pipelining
    - RISC architectures
    - Pipelined functional units
  - ILP
  - Vector units
  - Hardware support of multi-threading
- Let's look at them briefly

Courtesy of Henri Casanova

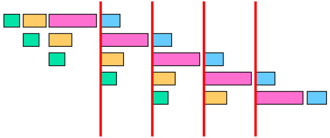
Concurrency within a CPU

- Several techniques to allow concurrency within a single CPU
  - Pipelining
    - RISC architectures
    - Pipelined functional units
  - ILP
  - Vector units
  - Hardware support of multi-threading
- Let's look at them briefly

Courtesy of Henri Casanova

Pipelining

- If one has a sequence of tasks to do
- If each task consists of the same n steps or *stages*
- If different steps can be done simultaneously
- Then one can have a pipelined execution of the tasks
  - e.g., for assembly line
- Goal: higher throughput (i.e., number of tasks per time unit)



Time to do 1 task	= 9
Time to do 2 tasks	= 13
Time to do 3 tasks	= 17
Time to do 4 tasks	= 21
Time to do 10 tasks	= 45
Time to do 100 tasks	= 409

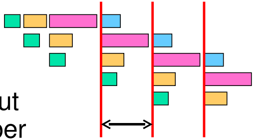
Pays off if many tasks

Courtesy of Henri Casanova

Pipelining

- Each step goes as fast as the slowest stage
- Therefore, the asymptotic throughput (i.e., the throughput when the number of tasks tends to infinity) is equal to:

  1 / (duration of the slowest stage)



- Therefore, in an ideal pipeline, all stages would be identical (balanced pipeline)
- Question: Can we make computer instructions all consist of the same number of stage, where all stages take the same number of clock cycles?

Courtesy of Henri Casanova

RISC

- Having all instructions doable in the same number of stages of the same durations is the RISC idea
- Example:
  - MIPS architecture (See THE architecture book by Patterson and Hennessy)
    - 5 stages
      - Instruction Fetch (IF)
      - Instruction Decode (ID)
      - Instruction Execute (EX)
      - Memory accesses (MEM)
      - Register Write Back (WB)
    - Each stage takes one clock cycle

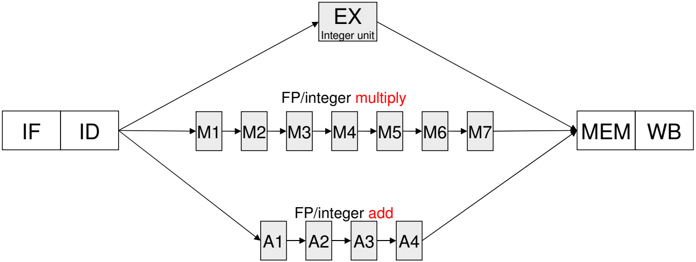
Concurrent execution of two instructions

LD R2, 12(R3)	IF	ID	EX	MEM	WB	
DADD R3, R5, R6		IF	ID	EX	MEM	WB

Courtesy of Henri Casanova

Pipelined Functional Units

- Although the RISC idea is attractive, some operations are just too expensive to be done in on clock cycle (during the EX stage)
- Common example: floating point operations
- Solution: implement them as a sequence of stages, so that they can be pipelined



Courtesy of Henri Casanova

Pipelining Today

- Pipelined functional units are common
- Fallacy: All computers today are RISC
  - RISC was of course one of the most fundamental "new" ideas in computer architectures
  - x86: Most commonly used Instruction Set Architecture today
  - Kept around for backwards compatibility reasons, because it's easy to implement (not to program for)
  - BUT: modern x86 processors decode instructions into "micro-ops", which are then executed in a RISC manner
  - New Itanium architecture uses pipelining
- Bottom line: pipelining is a pervasive (and conveniently hidden) form of concurrency in computers today
  - Take a computer architecture course to know all about it

Courtesy of Henri Casanova

Concurrency within a CPU

- Several techniques to allow concurrency within a single CPU
  - Pipelining
  - ILP
  - Vector units
  - Hardware support of multi-threading

Courtesy of Henri Casanova

Instruction Level Parallelism

- Instruction Level Parallelism is the set of techniques by which performance of a pipelined processor can be pushed even further
- ILP can be done by the hardware
  - Dynamic instruction scheduling
  - Dynamic branch predictions
  - Multi-issue superscalar processors
- ILP can be done by the compiler
  - Static instruction scheduling
  - Multi-issue VLIW processors
    - with multiple functional units
- Broad concept: More than one instruction is issued per clock cycle
  - e.g., 8-way multi-issue processor

Courtesy of Henri Casanova

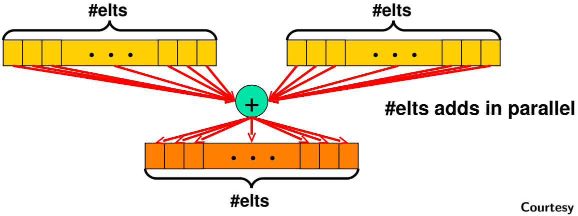
Concurrency within a CPU

- Several techniques to allow concurrency within a single CPU
  - Pipelining
  - ILP
  - Vector units
  - Hardware support of multi-threading

Courtesy of Henri Casanova

Vector Units

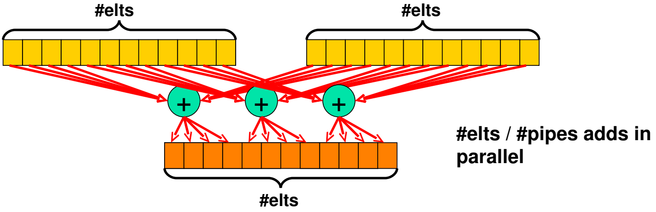
- A functional unit that can do elt-wise operations on entire vectors with a single instruction, called a vector instruction
  - These are specified as operations on vector registers
  - A "vector processor" comes with some number of such registers
    - MMX extension on x86 architectures



Courtesy of Henri Casanova

Vector Units

- Typically, a vector register holds ~ 32-64 elements
- But the number of elements is always larger than the amount of parallel hardware, called vector pipes or lanes, say 2-4



Courtesy of Henri Casanova

MMX Extension

- Many techniques that are initially implemented in the “supercomputer” market, find their way to the mainstream
- Vector units were pioneered in supercomputers
  - Supercomputers are mostly used for scientific computing
  - Scientific computing uses tons of arrays (to represent mathematical vectors and often does regular computation with these arrays
  - Therefore, scientific code is easy to “vectorize”, i.e., to generate assembly that uses the vector registers and the vector instructions
- Intel’s MMX or PowerPC’s AltiVec
  - MMX vector registers
    - eight 8-bit elements
    - four 16-bit elements
    - two 32-bit elements
  - AltiVec: twice the lengths
- Used for “multi-media” applications
  - image processing
  - rendering
  - ...

Courtesy of Henri Casanova

Vectorization Example

- Conversion from RGB to YUV

```
Y = (9798*R + 19235*G + 3736*B) / 32768;
U = (-4784*R - 9437*G + 4221*B) / 32768 + 128;
V = (20218*R - 16941*G - 3277*B) / 32768 + 128;
```

- This kind of code is perfectly parallel as all pixels can be computed independently
- Can be done easily with MMX vector capabilities
  - Load 8 R values into an MMX vector register
  - Load 8 G values into an MMX vector register
  - Load 8 B values into an MMX vector register
  - Do the *, +, and / in parallel
  - Repeat

Courtesy of Henri Casanova

Concurrency within a CPU

- Several techniques to allow concurrency within a single CPU
  - Pipelining
  - ILP
  - Vector units
  - Hardware support of multi-threading

Courtesy of Henri Casanova

Thread versus Process - Processors versus Core

Someone could remind us what is the difference between:

- a thread and a process ?
- a processor (socket) and a core ?

Multi-threading

- Multi-threading has been arounds for years, so what's new about this???
- Here we're talking about Hardware Support for threads
  - Simultaneous Multi Threading (SMT)
  - SuperThreading
  - HyperThreading
- Let's try to understand what all of these mean before looking at multi-threaded Supercomputers

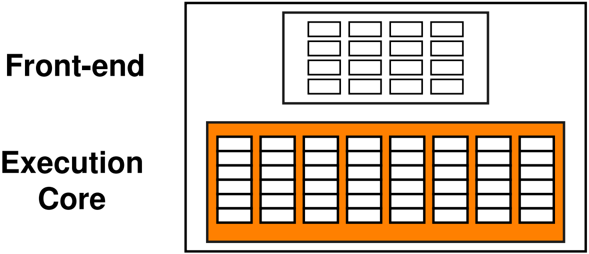
Courtesy of Henri Casanova

Single-threaded Processor

- The processor provides the illusion of concurrent execution
  - Front-end: fetching/decoding/reordering
  - Execution core: actual execution
- Multiple programs in memory
- Only one executes at a time
  - 4-issue CPU with bubbles
  - 7-unit CPU with pipeline bubbles
- Time-slicing via context switching

Courtesy of Henri Casanova

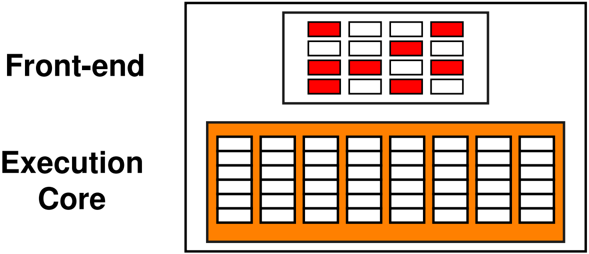
Simplified Example CPU



- The front-end can issue four instructions to the execution core simultaneously
  - 4-stage pipeline
- The execution core has 8 functional units
  - each a 6-stage pipeline

Courtesy of Henri Casanova

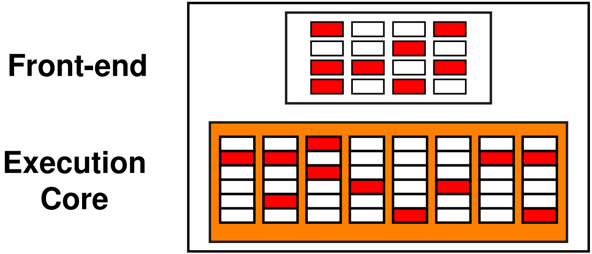
Simplified Example CPU



- The front-end is about to issue 2 instructions
- The cycle after it will issue 3
- The cycle after it will issue only 1
- The cycle after it will issue 2
- There is complex hardware that decides what can be issued

Courtesy of Henri Casanova

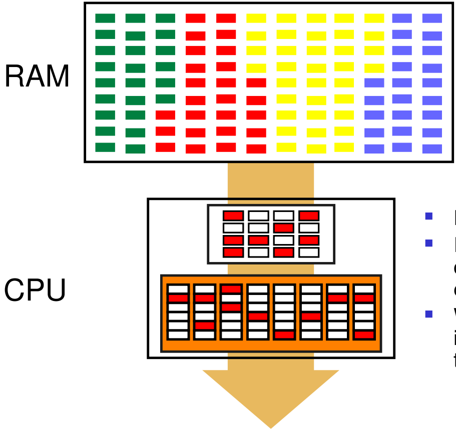
Simplified Example CPU



- At the current cycle, two functional units are used
- Next cycle one will be used
- And so on
- The while slots are “pipeline bubbles”: lost opportunity for doing useful work
  - Due to low instruction-level parallelism in the program

Courtesy of Henri Casanova

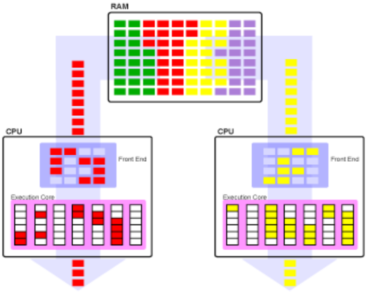
Multiple Threads in Memory



- Four threads in memory
- In a “traditional” architecture, only the “red” thread is executing
- When the O/S context switches it out, then another thread gets to run

Courtesy of Henri Casanova

Single-threaded SMP?

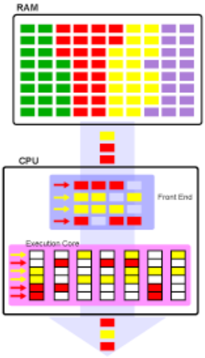


- Two threads execute at once, so threads spend less time waiting
- The number of “bubbles” is also doubled
- ➔ Twice as much speed and twice as much waste

Courtesy of Henri Casanova

Super-threading

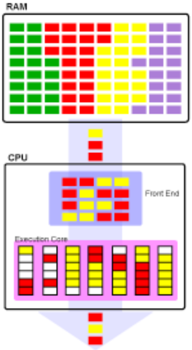
- Principle: the processor can execute more than one thread at a time
- Also called time-slice multithreading
- The processor is then called a *multithreaded processor*
- Requires more hardware cleverness
  - logic switches at each cycle
- Leads to less Waste
  - A thread can run during a cycle while another thread is waiting for the memory
  - Just a finer grain of interleaving
- But there is a restriction
  - Each stage of the front end or the execution core only runs instructions from ONE thread!
- Does not help with poor instruction parallelism within one thread
  - Does not reduce bubbles within a row



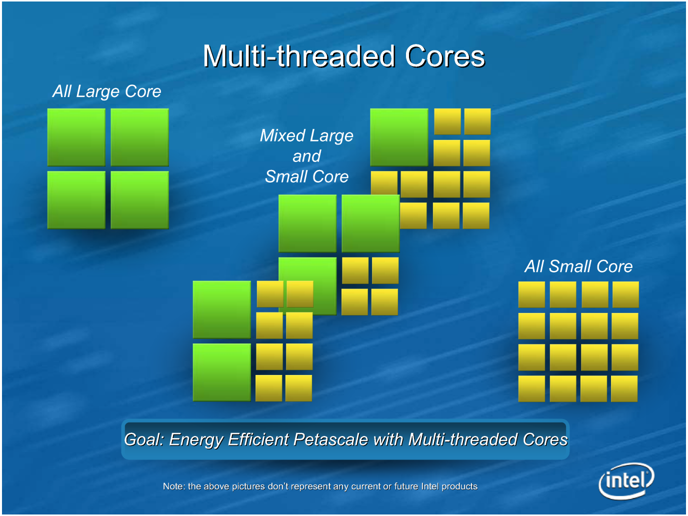
Courtesy of Henri Casanova

Hyper-threading

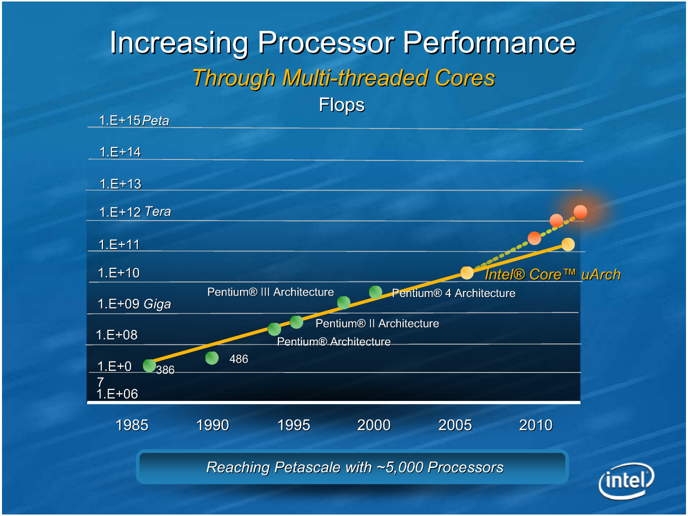
- Principle: the processor can execute more than one thread at a time, even within a single clock cycle!!
- Requires even more hardware cleverness
  - logic switches within each cycle
- On the diagram: Only two threads execute simultaneously.
  - Inter's hyper-threading only adds 5% to the die area
  - Some people argue that "two" is not "hyper" ☺
- Finest level of interleaving
- From the OS perspective, there are two "logical" processors



Courtesy of Henri Casanova



Courtesy of Intel



Courtesy of Intel

Outline

Concurrency within a “Box”

- Two main techniques
  - SMP
  - Multi-core
- Let’s look at both of them

Courtesy of Henri Casanova

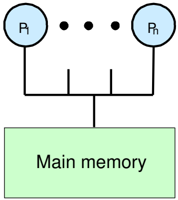
Multiple CPUs

- We have seen that there are many ways in which a single-threaded program can in fact achieve some amount of true concurrency in a modern processor
  - ILP, vector instructions
- On a hyper-threaded processors, a single-threaded program can also achieve some amount of true concurrency
- But there are limits to these techniques, and many systems provide increased true concurrency by using multiple CPUs

Courtesy of Henri Casanova

SMPs

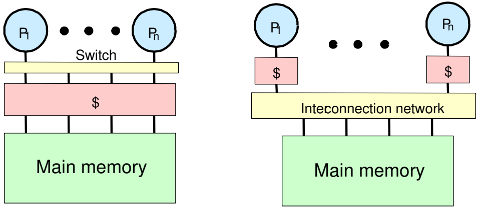
- Symmetric Multi-Processors
  - often mislabeled as "Shared-Memory Processors", which has now become tolerated
- Processors are all connected to a single memory
- Symmetric: each memory cell is equally close to all processors
- Many dual-proc and quad-proc systems
  - e.g., for servers



Courtesy of Henri Casanova

Shared Memory and Caches?

- When building a shared memory system with multiple processors / cores, one key question is: where does one put the cache?
- Two options



Shared Cache **Private Caches**

Courtesy of Henri Casanova

Shared Caches

- Advantages
  - Cache placement identical to single cache
    - Only one copy of any cached block
    - Can't have different values for the same memory location
  - Good interference
    - One processor may prefetch data for another
    - Two processors can each access data within the same cache block, enabling fine-grain sharing
- Disadvantages
  - Bandwidth limitation
    - Difficult to scale to a large number of processors
    - Keeping all processors working in cache requires a lot of bandwidth
  - Size limitation
    - Building a fast large cache is expensive
  - Bad interference
    - One processor may flush another processor's data

Courtesy of Henri Casanova

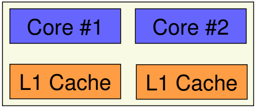
Shared Caches

- Shared caches have known a strange evolution
- Early 1980s
  - Alliant FX-8
    - 8 processors with crossbar to interleaved 512KB cache
  - Encore & Sequent
    - first 32-bit microprocessors
    - two procs per board with a shared cache
- Then disappeared
- Only to reappear in recent MPPs
  - Cray X1: shared L3 cache
  - IBM Power 4 and Power 5: shared L2 cache
- Typical multi-proc systems do not use shared caches
- But they are common in multi-core systems

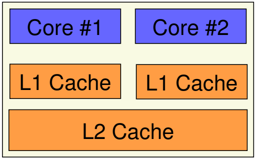
Courtesy of Henri Casanova

Caches and multi-core

- Typical multi-core architectures use distributed L1 caches

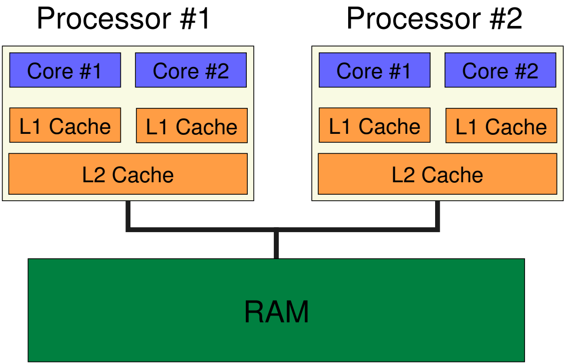


- But lower levels of caches are shared



Courtesy of Henri Casanova

Multi-proc & multi-core systems



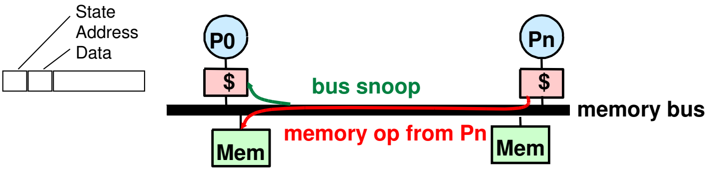
Courtesy of Henri Casanova

Private caches

- The main problem with private caches is that of memory consistency
- Memory consistency is jeopardized by having multiple caches
  - P1 and P2 both have a cached copy of a data item
  - P1 write to it, possibly write-through to memory
  - At this point P2 owns a stale copy
- When designing a multi-processor system, one must ensure that this cannot happen
  - By defining protocols for cache coherence

Courtesy of Henri Casanova

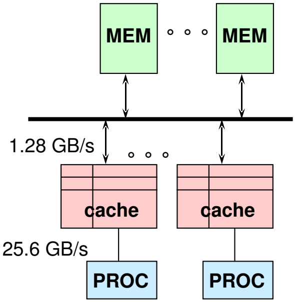
Snoopy Cache-Coherence



- The memory bus is a broadcast medium
- Caches contain information on which addresses they store
- Cache Controller "snoops" all transactions on the bus
  - A transaction is a relevant transaction if it involves a cache block currently contained in this cache
  - Take action to ensure coherence
    - invalidate, update, or supply value

Courtesy of Henri Casanova

Limits of Snoopy Coherence



Assume:

4 GHz processor

=> 16 GB/s inst BW per processor (32-bit)

=> 9.6 GB/s data BW at 30% load-store of 8-byte elements

Suppose 98% inst hit rate and 90% data hit rate

=> 320 MB/s inst BW per processor

=> 960 MB/s data BW per processor

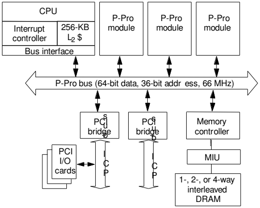
=> 1.28 GB/s combined BW

Assuming 10 GB/s bus bandwidth

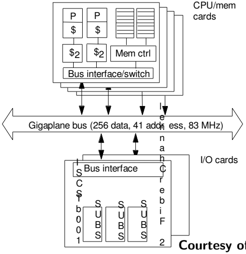
8 processors will saturate the bus

Sample Machines

- Intel Pentium Pro Quad
  - Coherent
  - 4 processors



- Sun Enterprise server
  - Coherent
  - Up to 16 processor and/or memory-I/O cards



Courtesy of Henri Casanova

Directory-based Coherence

- Idea: Implement a “directory” that keeps track of where each copy of a data item is stored
- The directory acts as a filter
  - processors must ask permission for loading data from memory to cache
  - when an entry is changed the directory either update or invalidate cached copies
- Eliminate the overhead of broadcasting/snooping, a thus bandwidth consumption
- But is slower in terms of latency
- Used to scale up to numbers of processors that would saturate the memory bus

Courtesy of Henri Casanova

Example machine

- SGI Altix 3000
- A node contains up to 4 Itanium 2 processors and 32GB of memory
- Uses a mixture of snoopy and directory-based coherence
- Up to 512 processors that are cache coherent (global address space is possible for larger machines)

Courtesy of Henri Casanova

Sequential Consistency?

- A lot of hardware and technology to ensure cache coherence
- But the sequential consistency model may be broken anyway
  - The compiler reorders/removes code
  - Prefetch instructions cause reordering
  - The network may reorder two write messages
- Basically, a bunch of things can happen
- Virtually all commercial systems give up on the idea of maintaining strong sequential consistency

Courtesy of Henri Casanova

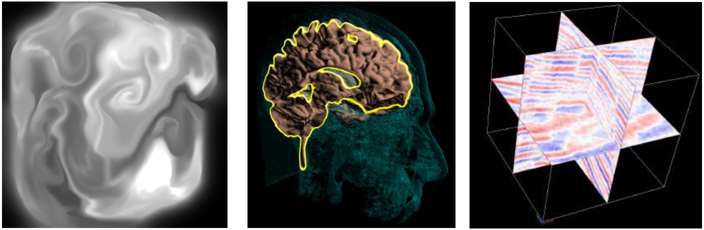
Weaker models

- The programmer must program with weaker memory models than Sequential Consistency
- Done with some rules
  - Avoid race conditions
  - Use system-provided synchronization primitives
- We will see how to program shared-memory machines
  - ICS432 is "all" about this
  - We'll just do a brief "review" in 632

Courtesy of Henri Casanova

GPGPU

- General Purpose computation on the GPU (Graphics Processing Unit)
  - Started in computer graphics community
  - Mapping computation problems to graphics rendering pipeline

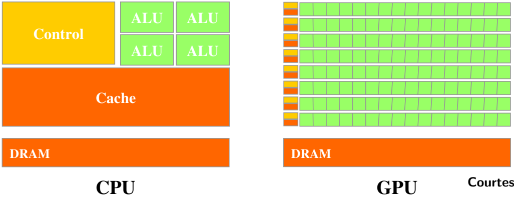


Courtesy Jens Krueger and Aaron Lefohn

Courtesy of Jean-François Méhaut

GPU is for Parallel Computing

- CPU
  - Large cache and sophisticated flow control minimize latency for arbitrary memory access for serial process
- GPU
  - Simple flow control and limited cache, more transistors for computing in parallel
  - High arithmetic intensity hides memory latency



Courtesy NVIDIA

Courtesy of Jean-François Méhaut

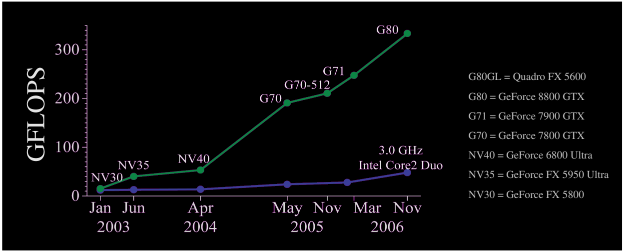
Why GPU for Computing?

- GPU is fast
  - Massively parallel
    - CPU : ~4 @ 3.0 Ghz (Intel Quad Core)
    - GPU : ~128 @ 1.35 Ghz (Nvidia GeForce 8800 GTX)
  - High memory bandwidth
    - CPU : 21 GB/s
    - GPU : 86 GB/s
  - Simple architecture optimized for compute intensive task
- Programmable
  - Shaders, NVIDIA CUDA, ATI CTM
- High precision floating point support
  - 32bit floating point IEEE 754
  - 64bit floating point will be available in early 2008

Courtesy of Jean-François Méhaut

Why GPU for computing?

- Inexpensive supercomputer
  - Two NVIDIA Tesla D870 : 1 TFLOPS
- GPU hardware performance increases faster than CPU
  - Trend : simple, scalable architecture, interaction of clock speed, cache, memory (bandwidth)



Courtesy NVIDIA

Courtesy of Jean-François Méhaut

GPU-friendly Problems

- High arithmetic intensity
  - Computation must offset memory latency
- Coherent data access (e.g. structured grids)
  - Maximize memory bandwidth
- Data-parallel processing
  - Same computation over large datasets (SIMD)
    - E.g. convolution using a fixed kernel, PDEs
    - Jacobi updates (isolate data stream read and write)

Courtesy of Jean-François Méhaut

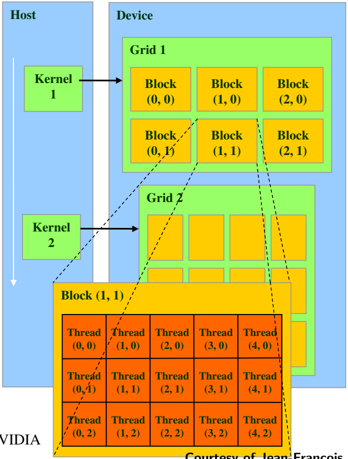
GPU : Highly Parallel Coprocessor

- GPU as a coprocessor that
  - Has its own DRAM memory
  - Communicate with host (CPU) through bus (PCIx)
  - Runs many threads in *parallel*
- GPU threads
  - GPU threads are extremely lightweight (almost no cost for creation/context switch)
  - GPU needs at least several thousands threads for full efficiency

Courtesy of Jean-François Méhaut

Programming Model: SPMD + SIMD

- Hierarchy
  - Device = Grids
  - Grid = Blocks
  - Block = Warps
  - Warp = Threads
- Single kernel runs on multiple blocks (SPMD)
- Single instruction executed on multiple threads (SIMD)
  - Warp size determines SIMD granularity (G80 : 32 threads)
- Synchronization within a block using shared memory

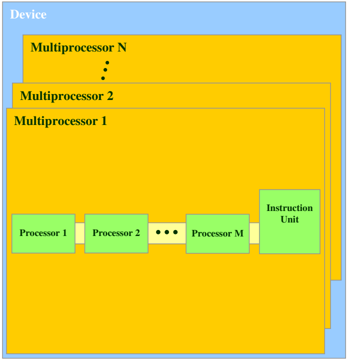


Courtesy NVIDIA

Courtesy of Jean-François Méhaut

Hardware Implementation : a set of SIMD Processors

- Device
  - a set of multiprocessors
- Multiprocessor
  - a set of 32-bit SIMD processors



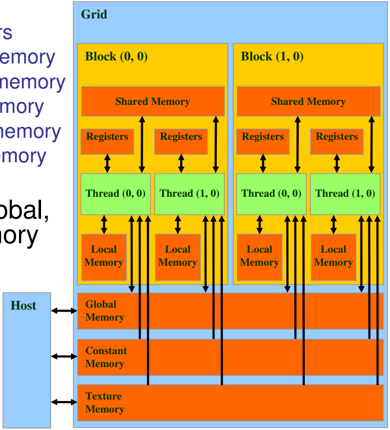
Courtesy NVIDIA

Courtesy of Jean-François Méhaut

Memory Model

- Each thread can:
  - Read/write per-thread registers
  - Read/write per-thread local memory
  - Read/write per-block shared memory
  - Read/write per-grid global memory
  - Read only per-grid constant memory
  - Read only per-grid texture memory

- The host can read/write global, constant, and texture memory

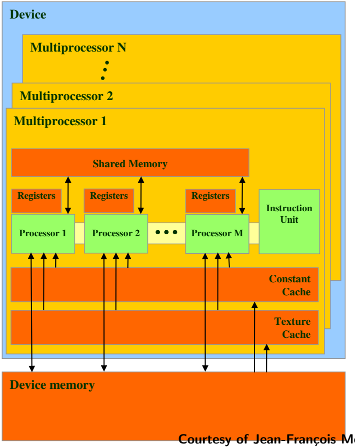


Courtesy NVIDIA

Courtesy of Jean-François Méhaut

Hardware Implementation : Memory Architecture

- Device memory (DRAM)
  - Slow (2~300 cycles)
  - Local, global, constant, and texture memory
- On-chip memory
  - Fast (1 cycle)
  - Registers, shared memory, constant/texture cache



Courtesy NVIDIA

Courtesy of Jean-François Méhaut

Memory Access Strategy

```
Copy data from global to shared memory

Synchronization

Computation (iteration)

Synchronization

Copy data from shared to global memory
```

Courtesy of Jean-François Méhaut

Execution Model

- Each thread block is executed by a single multiprocessor
  - Synchronized using shared memory
- Many thread blocks are assigned to a single multiprocessor
  - Executed concurrently in a time-sharing fashion
  - Keep GPU as busy as possible
- Running many threads in parallel can hide DRAM memory latency
  - Global memory access : 2~300 cycles

Courtesy of Jean-François Méhaut

CUDA

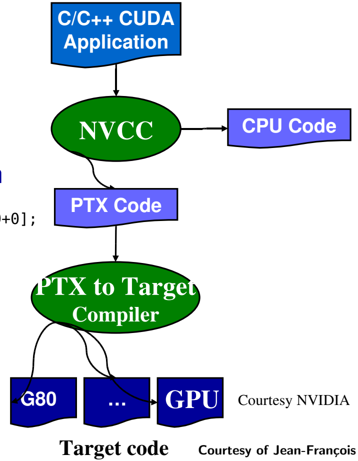
- C-extension programming language
  - No graphics API
    - Flattens learning curve
    - Better performance
  - Support debugging tools
- Extensions / API
  - Function type : __global__, __device__, __host__
  - Variable type : __shared__, __constant__
  - cudaMalloc(), cudaFree(), cudaMemcpy(),…
  - __syncthread(), atomicAdd(),…
- Program types
  - *Device* program (kernel) : run on the GPU
  - *Host* program : run on the CPU to call device programs

Courtesy of Jean-François Méhaut

Compiling CUDA

- nvcc
  - Compiler driver
  - Invoke cudacc, g++, cl
- PTX
  - Parallel Thread eXecution

```
ld.global.v4.f32  {$f1,$f3,$f5,$f7}, [$r9+0];
mad.f32           $f1, $f5, $f3, $f1;
```



Courtesy NVIDIA

Courtesy of Jean-François Méhaut

Outline

Motivation

Parallel Machines

- Parallel machines are expensive.
- The development tools for workstations are more mature than the contrasting proprietary solutions for parallel computers - mainly due to the non-standard nature of many parallel systems.

Workstation evolution

- Surveys show utilization of CPU cycles of desktop workstations is typically $< 10\%$.
- Performance of workstations and PCs is rapidly improving
- The communications bandwidth between workstations is increasing as new networking technologies and protocols are implemented in LANs and WANs.
- As performance grows, percent utilization will decrease even further! Organizations are reluctant to buy large supercomputers, due to the large expense and short useful life span.

Towards clusters of workstations

- Workstation clusters are easier to integrate into existing networks than special parallel computers.
- Workstation clusters are a cheap and readily available alternative to specialized High Performance Computing (HPC) platforms.
- Use of clusters of workstations as a distributed compute resource is very cost effective - incremental growth of system!!!

Definition.

A cluster is a type of parallel or distributed processing system (MIMD), which consists of a collection of interconnected stand-alone/complete computers cooperatively working together as a single, integrated computing resource.

Definition

A typical cluster

- A cluster is mainly homogeneous and is made of high performance and generally rather low cost components (PCs, Workstations, SMPs).
- Composed of a few to hundreds of machines.
- Network: Faster, closer connection than a typical LAN network; often a high speed low latency network (e.g. Myrinet, InfiniBand, Quadrix, etc.); low latency communication protocols; looser connection than SMP.

Typical usage

- Dedicated computation (rack, no screen and mouse).
- Non dedicated computation: Classical usage during the day (word, latex, mail, gcc) / HPC applications usage during the night and week-end.

Biggest clusters can be split in several parts:

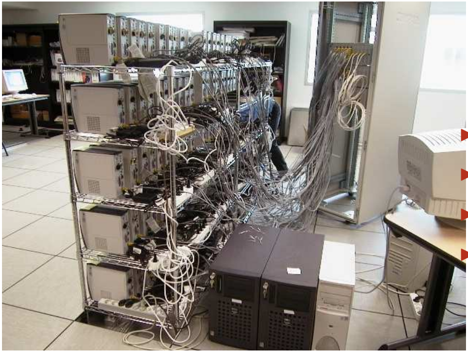
- computing nodes;
- I/O nodes;
- front (interactive) node.

A few examples

Berkeley NOW (1997)

- 100 SUN UltraSPARCs.
- Myrinet 160MB/s.
- Fast Ethernet.

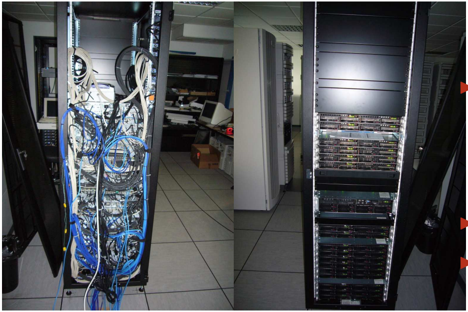
A few examples



Icluster (2000)

- 225 HP iVectra PIII 733 Mhz.
- Fast Ethernet.
- 81.6 Gflops (216 nodes).
- top 500 (385) June 2001.

A few examples

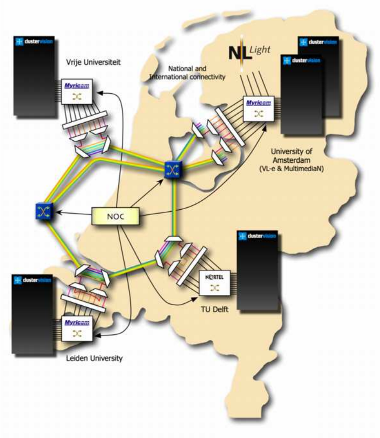


Digitalis (2008)

- 34 nodes (2 xeon quad cores $\rightsquigarrow$ 272 cores) with $2 \times 8\,Gb$ of RAM and $2 \times 160\,Gb$ of HD each.
- Infiniband.
- Giga Ethernet.

Clusters of clusters (HyperClusters)

DAS3: ASCI (Advanced School for Computing and Imaging),Netherlands.



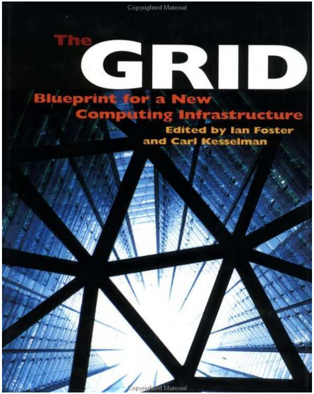
- Five Linux supercomputer clusters with 550 AMD Opteron processors.
- 1TB of memory and 100TB of storage.
- Myricom Myri-10G network inside clusters.
- Clusters are interconnected by a SURFnet's multi-color optical backbone.

The concept of Grid...

The Grid: Blueprint for a New Computing Infrastructure (1998); Ian Foster, Carl Kesselman, Jack Dongarra, Fran Berman, ...

Analogy with the electric supply:

- You don't know where the energy comes from when you turn on your coffee machine.
- You don't need to know where your computations are done.



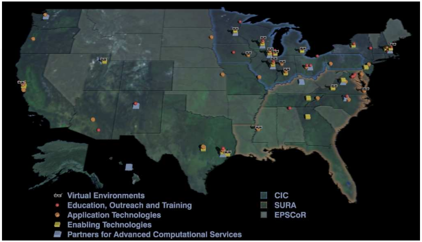
The concept of Grid (cont'd)

A grid is an infrastructure that couples:

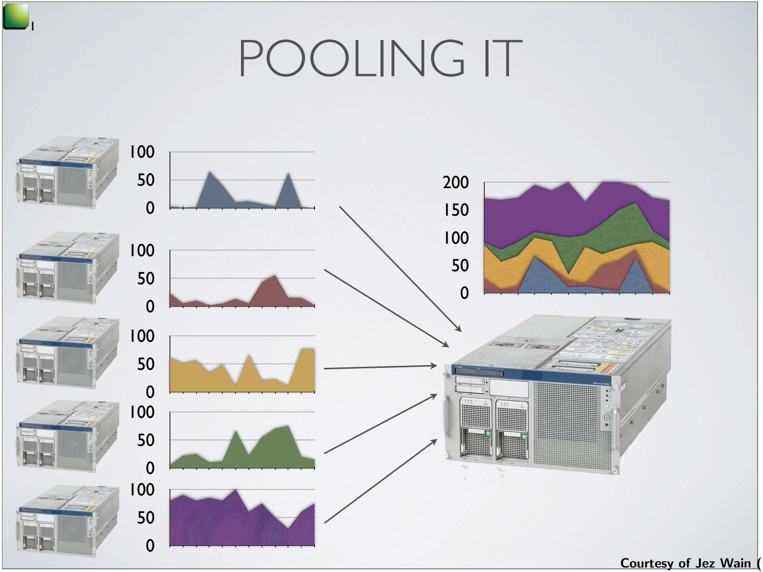
- Computers (PCs, workstations, clusters, traditional supercomputers, and even laptops, notebooks, mobile computers, PDA, and so on);
- Software Databases (e.g., transparent access to human genome database);
- Special Instruments (e.g., radio telescope–SETI@Home Searching for Life in galaxy, Astrophysics@Swinburne for pulsars, a cave);
- People (maybe even animals who knows ?;-)

across the local/wide-area networks (enterprise, organizations, or Internet) and presents them as an unified integrated (single) resource.

What does a Grid look like?



It is very big and very heterogeneous!



Sunday, 24 January 2010

Courtesy of Jez Wain (BULL)

Various versions of "*Grid*"

You have probably heard of many *buzzwords*.

- Super-computing;
- Global Computing;
- Internet Computing;
- Grid Computing;
- Meta-computing;
- Cloud Computing;
- Web Services;
- Cloud Computing;
- Ambient computing;
- Peer-to-peer;
- Web;

Large Scale Distributed Systems

"A distributed system is a collection of independent computers that appear to the users of the system as a single computer"
Distributed Operating System. A. Tannenbaum, Prentice Hall, 1994

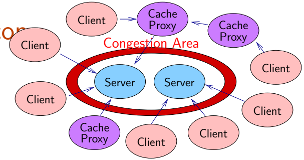
Tentative taxonomy

Purpose

- Information: share knowledge.
- Data: large-scale data storage.
- Computation: aggregate computing power.

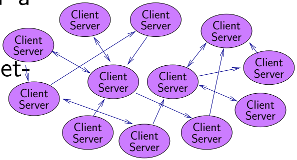
Deployment model

Client/server



- Not necessarily fully centralized.
- Use of caches and proxys to reduce congestion.
- Hierarchical structure is often used.
- Centralized information

Peer-to-peer



- Each peer acts both as a client and a server.
- The load is distributed over the whole network.
- Distributed information.

Example: Web sites
Client/server; information grid

Context

- Probably the first "grid".
- Information is accessed through a URL or more often through a search engine.
- Information access is fully transparent: you generally don't know where the informations comes from (mirrors, RSS feeds,...).

Challenges

Going peer-to-peer ? Web 2.0: users also contribute.

- Social networks (Facebook).
- Recommendations (google and amazon.com).
- Crowdsourcing (wikipedia, marmiton).
- Video and photo sharing (youtube).
- Media improvement (e.g., linking picassa and google maps).
- Ease of finding relevant information and ability to tag data.

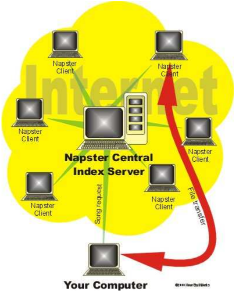
Example: Napster
Client/server; data grid

Context

- The first massively popular "peer-to-peer" file (MP3 only) sharing system (1999).
- Central servers maintain indexes of connected peers and the files they provide.
- Actual transactions are conducted directly between peers.

Drawbacks

- More client/server than truly peer-to-peer.
- Hence, servers have been attacked (by courts and by others to track peers offering copyrighted materials).

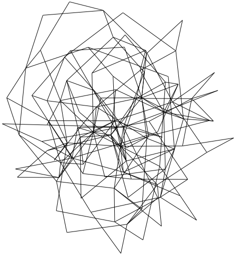


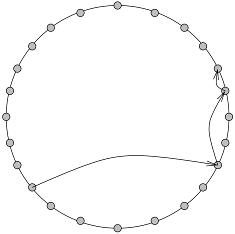
Example: Gnutella, Kazaa, Freenet, Chord

P2P; data grid

Context

- Removal of servers: searching can be done by flooding in unstructured overlays.
- Use of supernodes/ultrapeers (nodes with a good CPU and high bandwidth) for searching.
- Structured (hypercubes, torus, ...) overlay networks.
- Downloading from multiple sources using hash blocks and redundancy.





Example: Gnutella, Kazaa, Freenet, Chord

P2P; data grid

Context

- Removal of servers: searching can be done by flooding in unstructured overlays.
- Use of supernodes/ultrapeers (nodes with a good CPU and high bandwidth) for searching.
- Structured (hypercubes, torus, . . .) overlay networks.
- Downloading from multiple sources using hash blocks and redundancy.

Challenges

- Ensuring anonymity.
- Ensuring good throughput and efficient multi-cast (network coding, redundancy).
- Avoiding polluted data.
- Publish-subscribe overlays for fuzzy or complex queries.
- Free-riders.

Example: Internet Computing (SETI@home)

Client/server; computation grid

Context

- Search for possible evidence of radio transmissions from extraterrestrial intelligence using data from a telescope.
- The client is generally embedded into a screensaver.
- The server distributes the work-units to volunteer clients.
- Attracting volunteers with hall of fame and teams.
- Need to cross-check the results to detect false positives.
- 5.2 million participants worldwide, over two million years of aggregate computing time since its launch in 1999. 528 TeraFLOPS (Blue Gene peaks at just over 596 TFLOPS with sustained rate of 478 TFLOPS).
- Evolved into BOINC: Berkeley Open Infrastructure for Network Computing (climate prediction, protein folding, prime number factorizing, fight cancer, Africa@home, . . .).

Example: Internet Computing (SETI@home)

Client/server; computation grid

Challenges

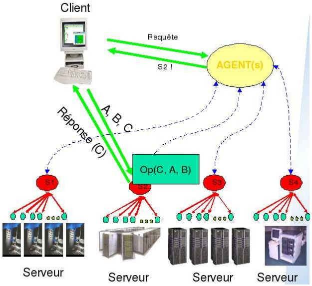
- Attract more volunteers: credits, ribbons and medals, connect with facebook.
- Volunteer thinking: use people's brains (intelligence, knowledge, cognition) to locate' solar dust, fossils, fold proteins.
- Works well for computation intensive embarrassingly parallel applications.
  - Really parallel applications.
  - Data intensive applications.
  - Soft real-time applications.
- Security.
  - Would you let anyone execute anything on your PC?
  - Use sandboxing and virtual machines.
- Need to go peer-to-peer (OurGrid).

Example: Meta-computing

Client/server; computation grid

Context

- Principle: buy computing services (pre-installed applications + computers) on the Internet.
- Examples: Netsolve (UTK), NINF (Tsukuba), DIET and Scilab // (ENS Lyon/INRIA),



Challenges

- Data storage and distribution: avoid multiple transfers between clients and servers when executing a sequence of operations.
- Efficient data redistribution.
- Security for file transfers
- Peer-to-peer deployment.

Example: grid computing

Client/server; computation grid; cloud computing

Context

- Principle: use a *virtual* supercomputer and execute applications on remote resources.
  "I need 200 64 bits machines with 1Tb of storage from 10:20 am to 10:40 pm."
- Need to match and locate resources, schedule applications, handle reservations, authentication, ...
- Examples: Globus, Legion, Unicore, Condor, ...

Challenges

- Obtaining good performances while deploying parallel codes on multiple domains.
- Communication and computation overlap. High-performance communications on heterogeneous networks.
- Need for new parallel algorithms that handle heterogeneity, hierarchy, dynamic resources,
- Complex applications $\rightsquigarrow$ code coupling (message passing $\rightsquigarrow$ distributed objects, components).

Summary

Usage \ Deployment	Client/Server	Peer-to-peer
Data	Napster	Gnutella, Kazaa, Chord, Freenet...
Information	Web 1.0 and 1.5 Search Engines	Web 2.0
Computing	Internet Computing; Meta-computing; Grid Computing	OurGrid

A few other challenges

- Security, Authentication, Trust, Error management.
- Middleware vs. Operating System.
- Algorithms for Grid Computing.
- Software engineering.
- Social aspects (fairness, selfishness, cooperation).
- Energy saving!
- Failures...

Conclusion

- No real new theme but rather a combination of already existing technologies for parallel and distributed computing.
- Such combinations and ambitious goals are very hard to achieve.
- This clearly requires a pluri-disciplinary approach with a good understanding of all aspects (OS, network, middleware, security, storage, algorithms, applications, . . .).
- It would be a mistake to restrict only to computing. Research on all these aspects should be encouraged.
- It is very important to identify and discriminate new concepts from technology and fad.
- A crucial question is:

“Should we hide the complexity or expose it?”

Tunnel Vision by Experts

- "On several recent occasions, I have been asked whether parallel computing will soon be relegated to the trash heap reserved for promising technologies that never quite make it."
  - Ken Kennedy, CRPC Directory, 1994
- "640K [of memory] ought to be enough for anybody."
  - Bill Gates, chairman of Microsoft,1981.
- "There is no reason for any individual to have a computer in their home"
  - Ken Olson, president and founder of Digital Equipment Corporation, 1977.
- "I think there is a world market for maybe five computers."
  - Thomas Watson, chairman of IBM, 1943.

Slide source: Warfield et al.

Courtesy of Jean-François Méhaut