Parallel Computing: from KiloFlops to Exascale. Evolution in the Last Decades and Recent Challenges

Arnaud Legrand
CNRS, INRIA, Univ. of Grenoble

M2R MOSIG

September 23, 2013

A Few Words About the Lecture Organization

- **Two lecturers**
  - Arnaud Legrand, CNRS, INRIA MESCAL project. On Monday.
  - Vincent Danjean, UJF, INRIA MOAIS project. On Wednesday.
- **Eleven 3-hours lectures** tentative roadmap: check the web page

  http://mescal.imag.fr/membres/arnaud.legrand/teaching/2013/M2R_PC.php

  - Parallel architectures (A. Legrand) 1
  - How to efficiently program High Performance Architectures? (V. Danjean) 1
  - Parallel algorithms and models (A. Legrand) 4
  - From parallelism-aware algorithms to parallelism-oblivious algorithm (V. Danjean) 3
  - Hype and trends: desktop grids, clouds, exascale (A. Legrand) 2
  - Step-by-step exam [monday afternoon in dec] +1
  - Exam [end jan]

A Few Words About What We Expect From You

- The content of this lecture is very **dense** and is intended to give you a broad overview of this area.
- No real need to read books. The slides comprise all the material you need, which is why there are so many.
- 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.
- 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

- Ask yourselves what are the main messages of the lectures.
- At the beginning of each lecture, you will thus certainly have questions about last lecture. Feel free to ask. . .

Communication

- There is a website with all the slides as well as practical information (room location, roadmap, additionnal readings, homeworks

  http://mescal.imag.fr/membres/arnaud.legrand/teaching/2013/M2R_PC.php
- If you have a question:

  mailto:arnaud.legrand@imag.fr mailto:vincent.danjean@imag.fr
- The basic requirements for following this lecture are: Operating Systems, Networking and Algorithms.
- I'll set up the mailing list this evening. I will send you a very short survey to estimate your current knowledge and background on parallel computing, OS, networking and algorithms.
- You should create an account on Grid5000 ("get an account", fill the form with my name as responsible for you):

  https://www.grid5000.fr/

Preparation to the internship

The M2R is not an exam. It is a contest.

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

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

  http://projets-mastermi.imag.fr/pcarre/

  I will soon give you a brief presentation of the three teams **MESCAL**, **MOAIS** and **NANOSIM**.

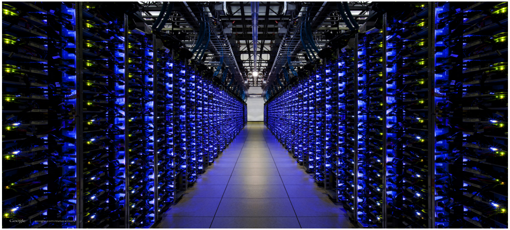
Outline

1. Introduction to the lecture
    - Organization Forewords
    - Computational Science and Digital Revolution
    - Distributed Computing infrastructures: Technology, Engineering and Research
    - A Brief History of Parallel and Distributed Computing
2. Why All Computers Have to be Parallel
    - Moore Law and Computing Limits
    - Multiple Cores Save Power
    - The Memory Wall
3. Parallelism at the CPU level
    - Vector Processing
    - Pipelining
    - Instruction Level Parallelism
    - Multi-Threading
4. When One is Not Enough
    - SMPs, Multi-cores, NUMAs
    - General Purpose GPUs
    - Clusters
    - Grid/Desktop/Internet/Cloud Computing

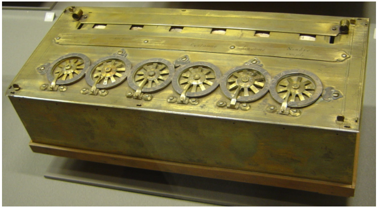
Computing Science: A Very Recent Science

- One could argue its premises start with the Sumerians back in 2700–2300BC.
- "Mechanical" computing has been the subject of research throughout 17th century.
- But computing Science really emerged in the late 20th century.

Abacus



Google Data Center



Pascaline

It has been the basis of a massive worldwide industry and has influenced both society and other fields of science through both **technology** and **science**.

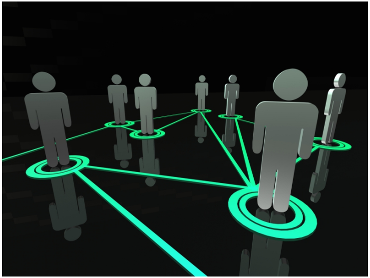
Digital Revolution

Started in the second half of the 20th century:

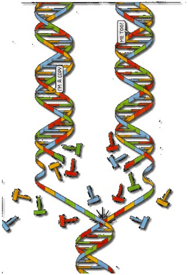
- mass production
- widespread usage of computer related technologies
- worldwide network connection: internet

The impact on **society** has been tremendous and is due both to computer/telecommunication **technologies** and computer **science**:

- RSA algorithms that secure our transactions
- Model checking of complex systems
- Recommendation algorithms and social networking



Computer Science and Other Sciences



Computer science also influenced other sciences like biology

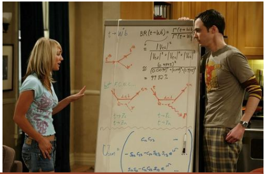
- notions of information and coding are **common tools and concepts**
- DNA sequences ≡ strings of a language
- cells ≡ self-regulatory systems similar like an electronic circuit
- interactions between molecules (proteins and RNA) ≡ process calculus

There is a clear hope data structures and algorithms can help understand the structure and interactions of proteins in ways that **elucidate their function at a global scale.**

Computational thinking is changing the way biologists think because it offers **new ways to conceive phenomena**.

It has also started influencing other disciplines like physics, chemistry, geo-sciences, economy, laws. . .

Computer Technology and other sciences



The Big Bang Theory

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**:

Big Data

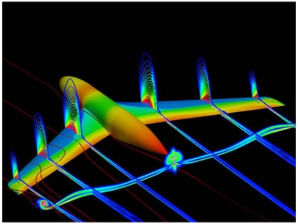
- Dig huge amounts of data (sensors, transaction records, genome and protein databanks,. . .)
- Enables to discover phenomena or truths that would otherwise remain unseen

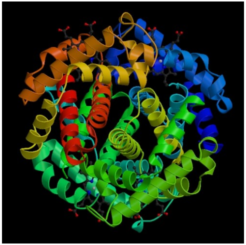
Computational Science

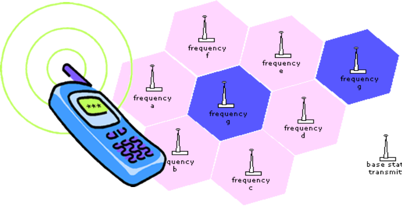
- Performing real experiment is very costly and even sometimes simply impossible
- Allows to explore and investigate designs or phenomena in a few hours instead of years

- Motivated the development of major computational infrastructures
- All fields of science (physics, genomics, astronomy, ecology,. . .) and industry (drug design, avionics, structural engineering, oil companies,. . .)

Killer Applications

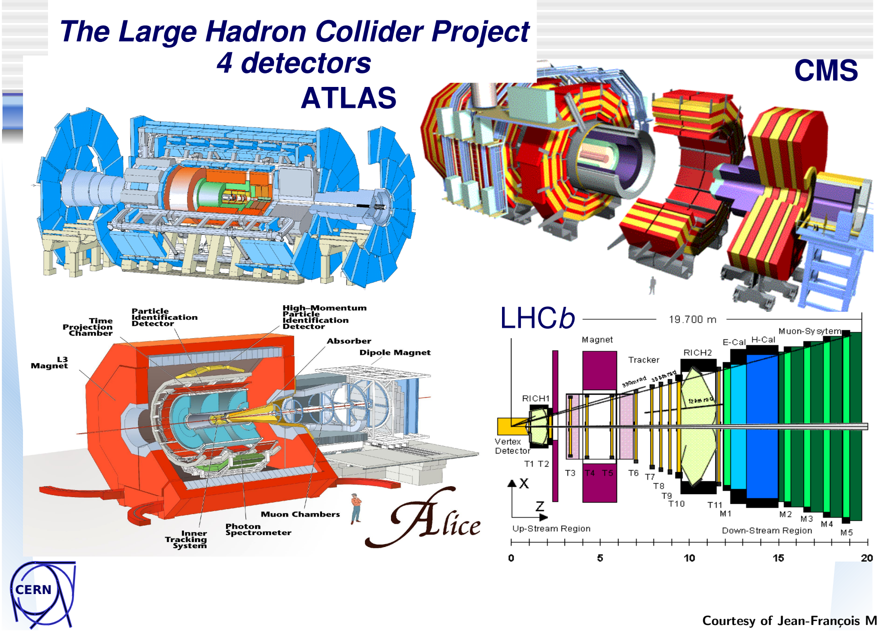






Courtesy of Jean-François Méhaut

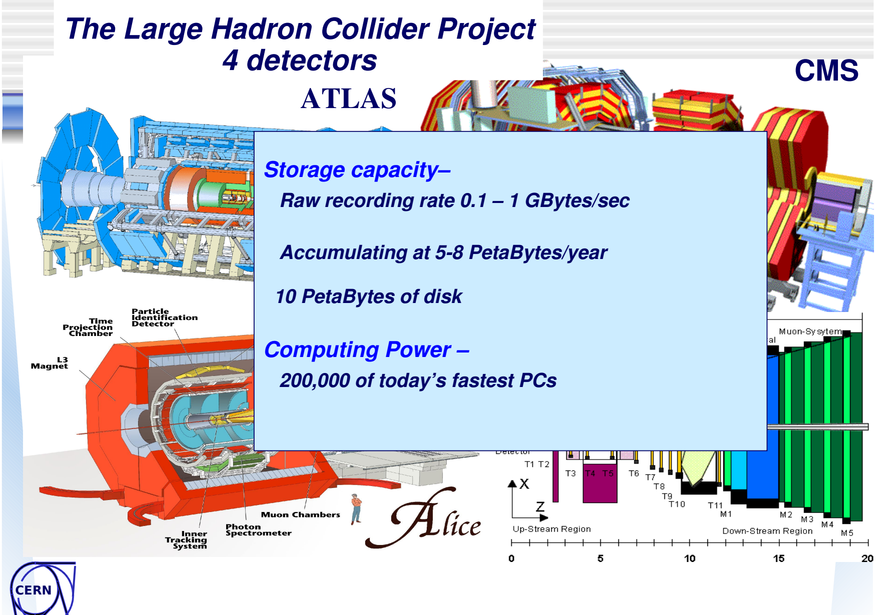
Killer Applications



The Large Hadron Collider Project
4 detectors

Courtesy of Jean-François Méhaut

Killer Applications



The Large Hadron Collider Project
4 detectors

ATLAS

CMS

Storage capacity–

Raw recording rate 0.1 – 1 GBytes/sec

Accumulating at 5-8 PetaBytes/year

10 PetaBytes of disk

Computing Power –

200,000 of today's fastest PCs

Courtesy of Jean-François Méhaut

Killer Applications

Earthquake Hazard Assessment

2001 Gujarati (M 7.7) Earthquake, India

Use parallel computing to simulate earthquakes

Learn about structure of the Earth based upon seismic waves (tomography)

Produce seismic hazard maps (local/regional scale) e.g. Los Angeles, Tokyo, Mexico City, Seattle

Demo

20,000 people killed
167,000 injured
≈ 339,000 buildings destroyed
783,000 buildings damaged

Courtesy of Jean-François Méhaut

Parallelism for Killer Applications

This 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

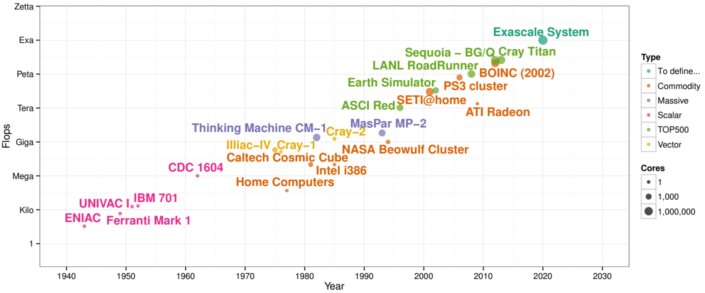
- grid computing
- ambiant computing
- cloud computing
- sky computing, ...

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

Knowing about this history may help to:

- understand the connection between **research** and **technology**
- understand what **research** in this area is about
- discriminate **hype** from **real trends**

A Journey Through Time



1943: the early days

ENIAC, **35 Flops**!

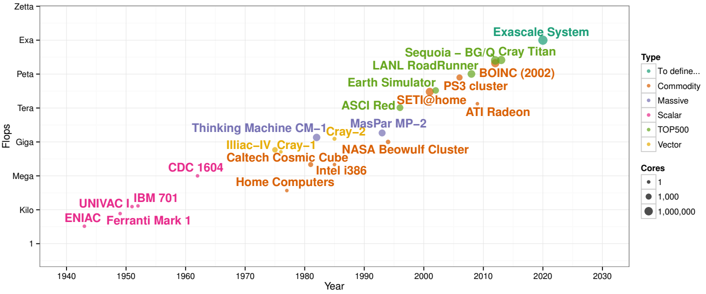
Designed to compute artillery firing tables

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."

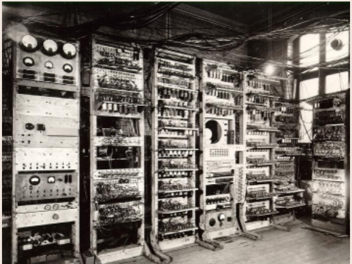
ENIAC

A Journey Through Time

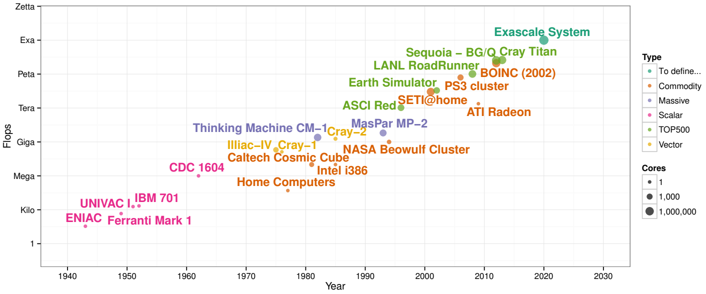


1949: the early days

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


anchester Mark 1

A Journey Through Time

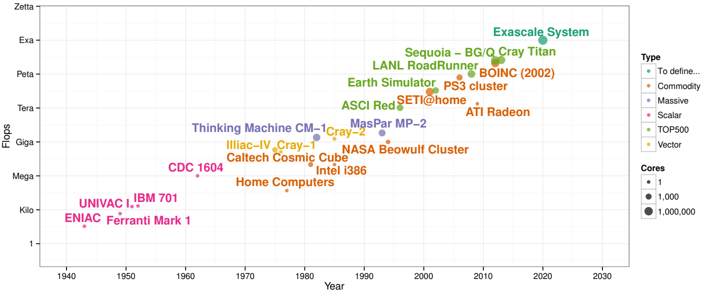


1951: a new market ?

- **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.

A Journey Through Time



1952: a new market!

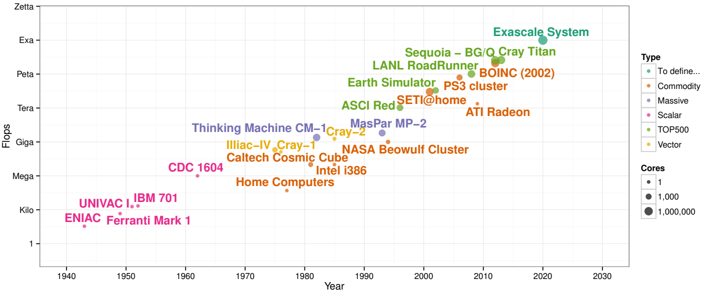
IBM 701 (aka Defense Calculator) is IBM first's commercial scientific computer. **2,200 FLOPS**. Rental charge was about $12,000 a month.

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

Watson visited 20 companies that were potential customers. This is what he said at the stockholders meeting:

“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

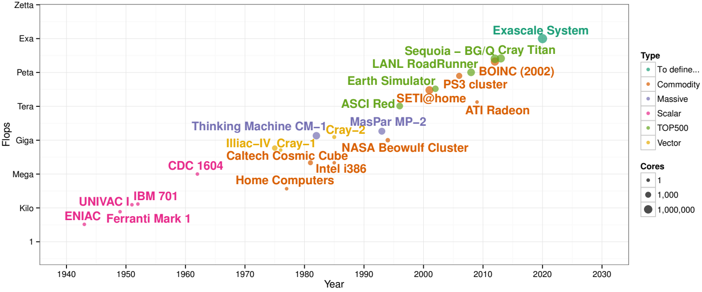


1962: Control Data Corporation

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



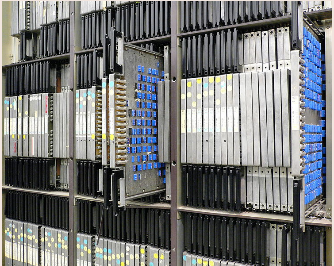
1966–1975: The Illiac-IV

Illiac-IV for NASA.

A linear array of 256 64-bit Processing Elements.

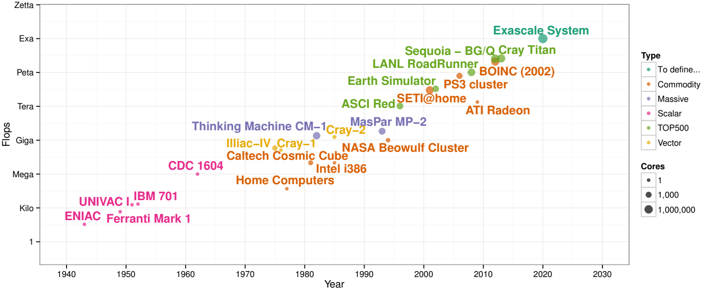
Expected 1 GFlops but reached only **200 MFlops**.

Was somehow the precursor of **vector processing**.



ILLIAC IV

A Journey Through Time

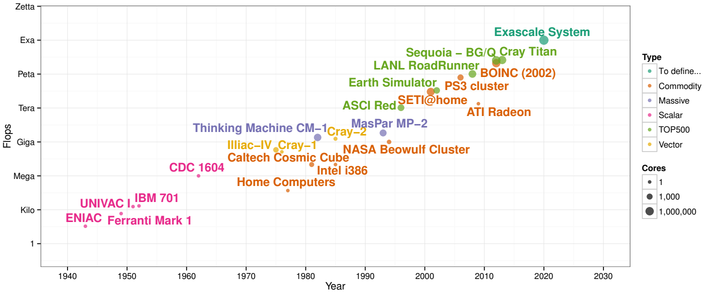


1970–1977: micro-computers

1970	Datapoint 2200
1921	Intel 4004
1972	Intel 8008
1972	Micral-N
1977	Second generation: home computers

Micral-N

A Journey Through Time



1976–1985: the CRAY domination

If you were plowing a field, which would you rather use? Two strong oxen or 1024 chickens? – Seymour Cray

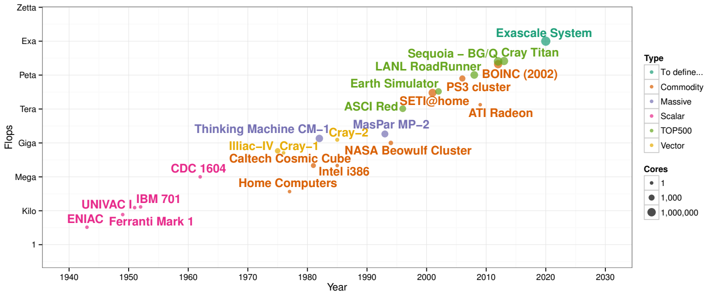
1976 **CRAY-1**. Scalar+vector processor, **133 MFLOPS** for 5 to 8 million $

1982 **Cray X-MP**. **800 MFlops** with 2 to 4 CPUs

1985 **1,900 MFlops CRAY-2**: 4 CPUs

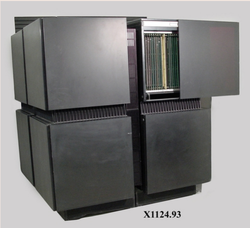
Cray-1

A Journey Through Time



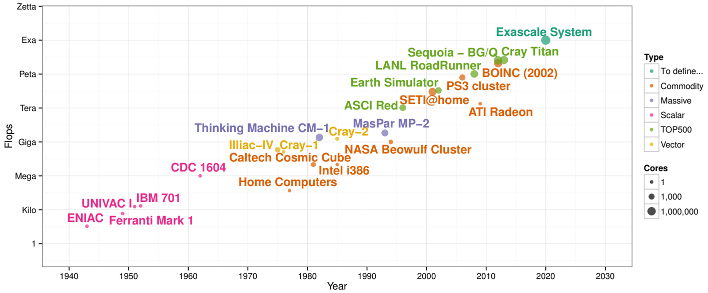
1976–1995: Massive paralelism

- 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

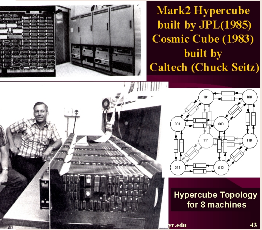


Connexion Machine-1

A Journey Through Time



1976–1995: commodity hardware. DIY!



Cosmic Cube

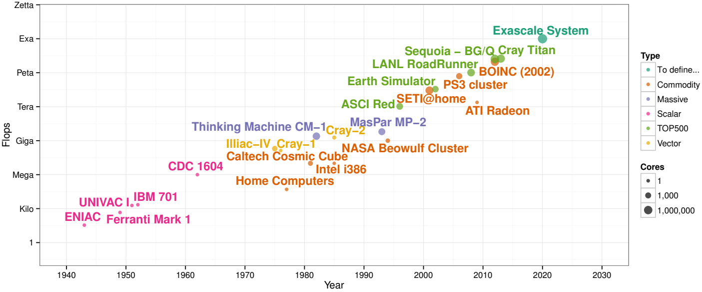
NASA Beowulf Cluster

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 with Ethernet, **1,000 MFlops** for $50,000.

A Journey Through Time



1996–...: distributed/volunteer computing

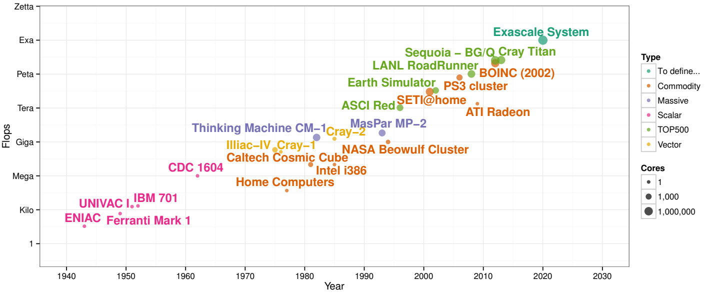
1996 GIMPS

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



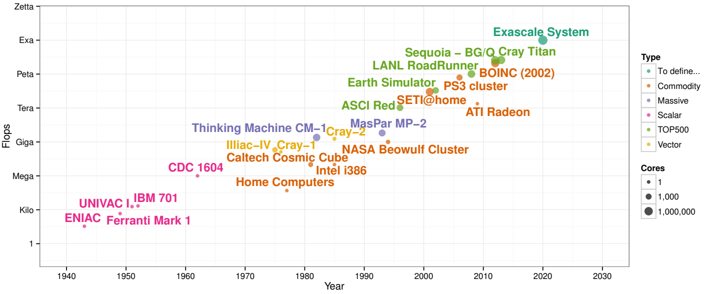
1996–...: Top500 "commodity" hardware

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

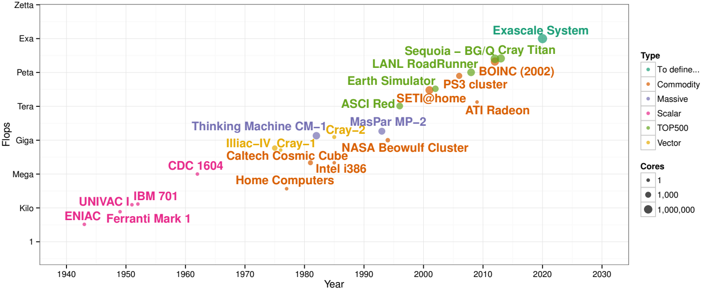


1996–...: commodity hardware

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

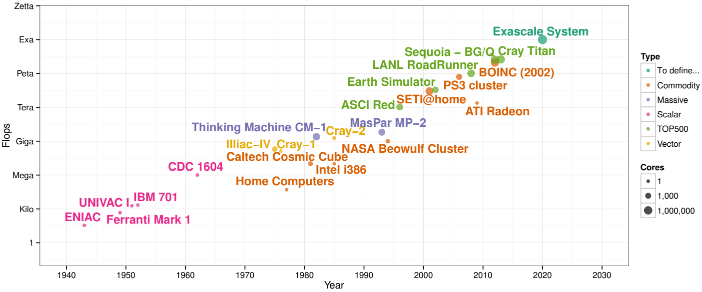
2012 **Sequoia** - BlueGene/Q. 98,304 16-core (1,572,864) Power processors.

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**

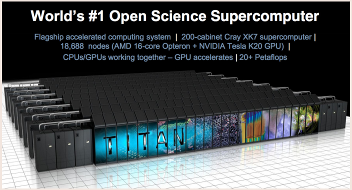
Sequoia

A Journey Through Time



2012–2013: Peta-scale systems

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

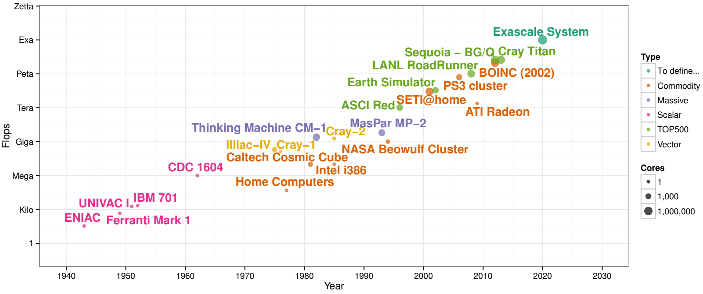


World's #1 Open Science Supercomputer

Flagship accelerated computing system | 200-cabinet Cray XK7 supercomputer | 18,688 nodes (AMD 16-core Opteron + NVIDIA Tesla K20 GPU) | CPUs/GPUs working together – GPU accelerates | 20+ Petaflops

Titan

A Journey Through Time

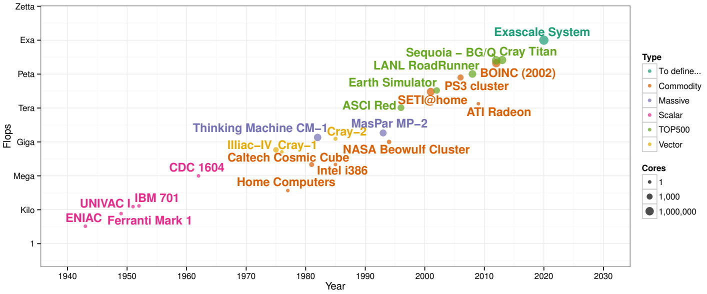


2012–2013: Peta-scale systems

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 1.7 GFLOPS / Watt.

- GPU-based but many other accelerators are possible
- ARM-based (Mont-blanc project)

Recap

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

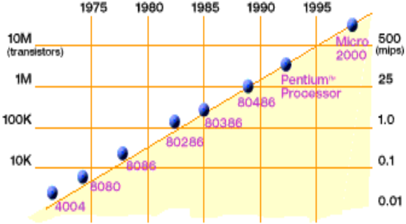
- Most companies died
- Research ideas make their way to mass production
  - vector processors, accelerators
  - pipelining
  - instruction level parallelism
  - multi-threading
- Some research ideas did not make their way because technology was not ready...
- ...or because there was no market for mass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

All computers are parallel.

Outline

1. Introduction to the lecture
    - Organization Forewords
    - Computational Science and Digital Revolution
    - Distributed Computing infrastructures: Technology, Engineering and Research
    - A Brief History of Parallel and Distributed Computing
2. Why All Computers Have to be Parallel
    - Moore Law and Computing Limits
    - Multiple Cores Save Power
    - The Memory Wall
3. Parallelism at the CPU level
    - Vector Processing
    - Pipelining
    - Instruction Level Parallelism
    - Multi-Threading
4. When One is Not Enough
    - SMPs, Multi-cores, NUMAs
    - General Purpose GPUs
    - Clusters
    - Grid/Desktop/Internet/Cloud Computing

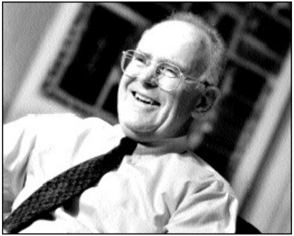
Moore's Law: microprocessor capacity



2X transistors/Chip Every 1.5 years

Called "Moore's Law"

Microprocessors have become smaller, denser, and more powerful.

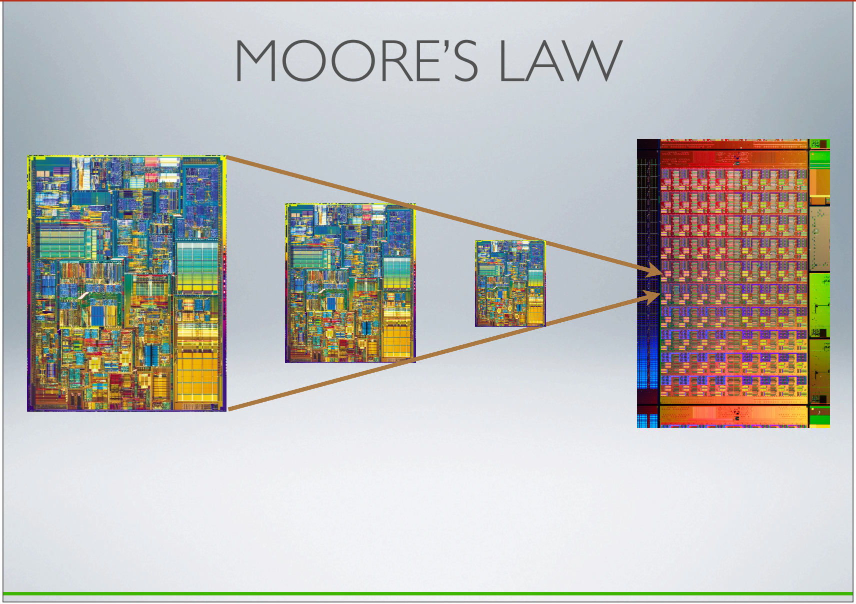


Gordon Moore (co-founder of Intel) predicted in 1965 that the transistor density of semiconductor chips would double roughly every 18 months.

Slide source: Jack Dongarra

Courtesy of Jean-François Méhaut

Moore’s Law

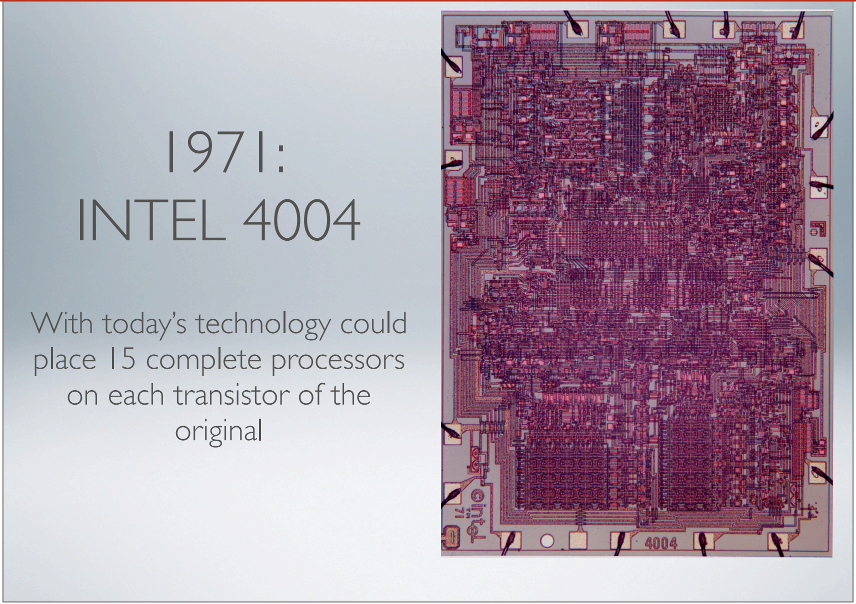


MOORE’S LAW

Sunday, 24 January 2010

Courtesy of Jez Wain (BULL)

Moore's Law



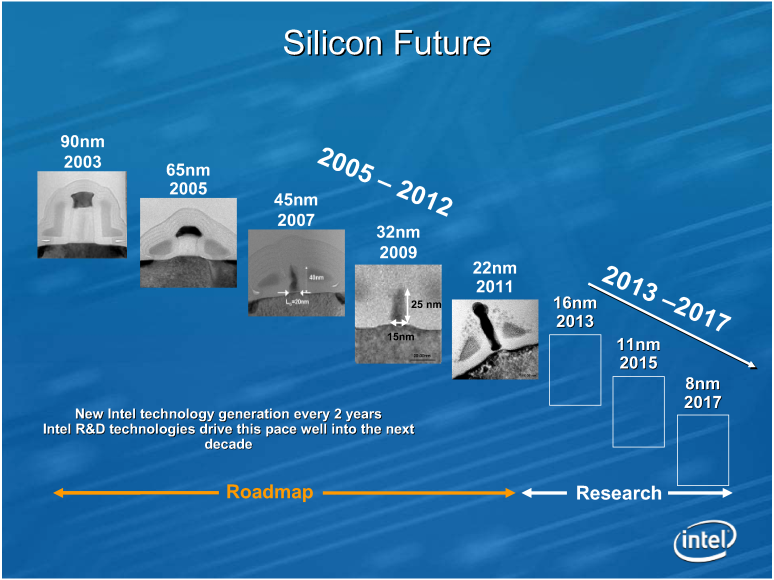
1971: INTEL 4004

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

Sunday, 24 January 2010

Courtesy of Jez Wain (BULL)

Moore’s Law



Silicon Future

90nm
2003

65nm
2005

45nm
2007

2005 – 2012

32nm
2009

22nm
2011

16nm
2013

2013 –2017

11nm
2015

8nm
2017

New Intel technology generation every 2 years
Intel R&D technologies drive this pace well into the next decade

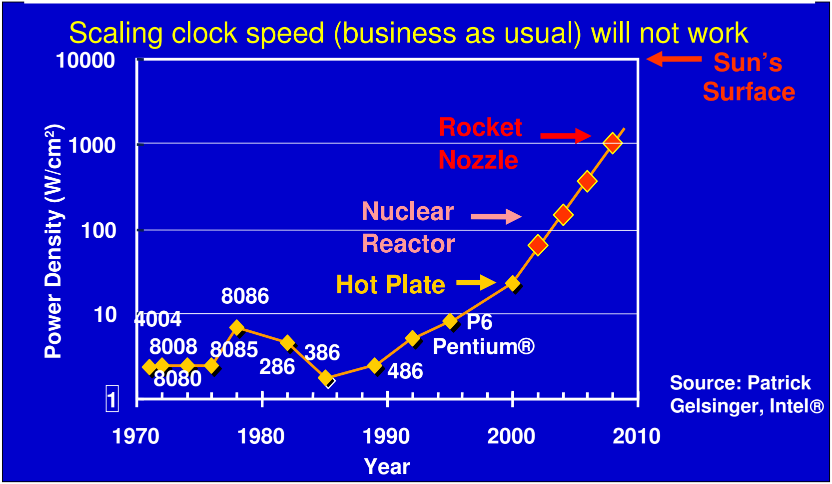
Roadmap

Research

Courtesy of Intel

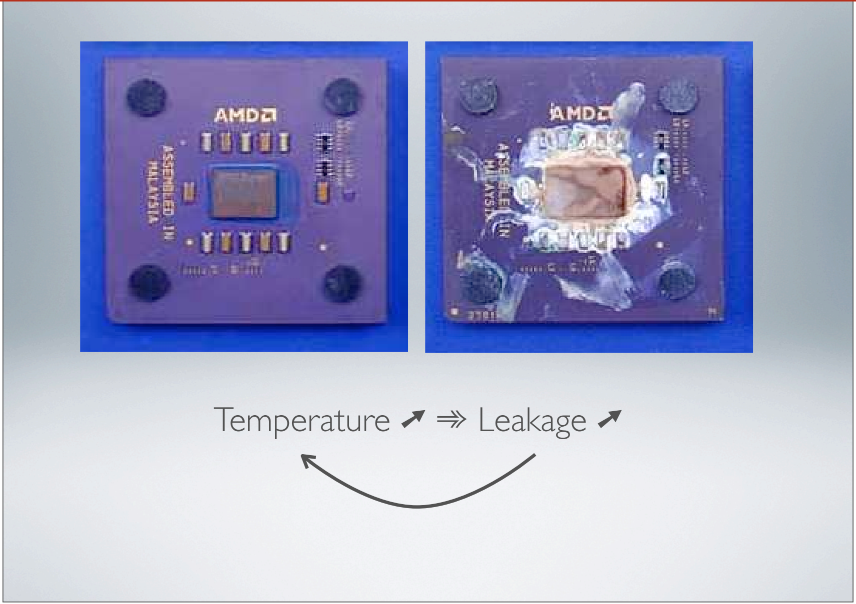
Increasing Frequency Does not Help

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



Courtesy of Jean-François Méhaut

Increasing Frequency Does not Help



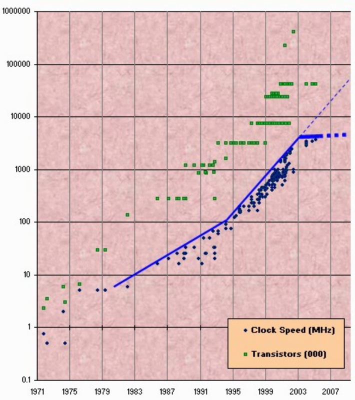
Temperature $\nearrow \Rightarrow$ Leakage $\nearrow$

Sunday, 24 January 2010

Courtesy of Jez Wain (BULL)

Moore's Law again

- 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

No More Moore ?

- 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

Outline

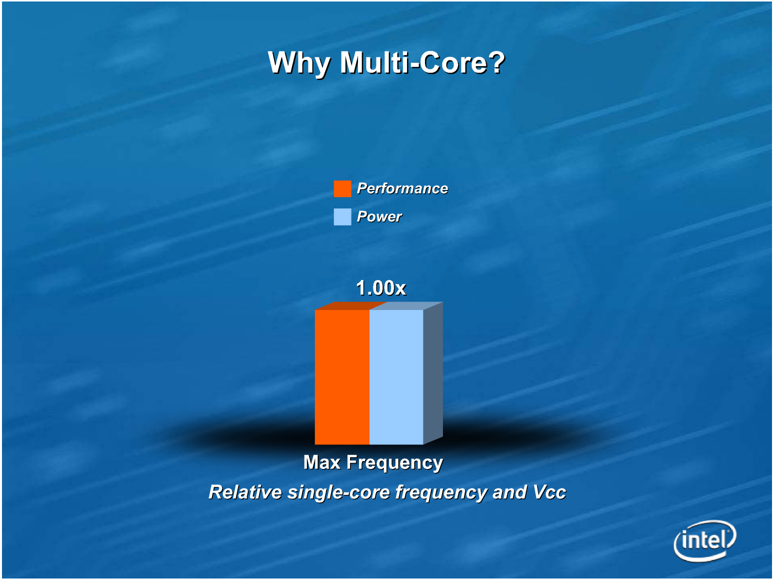
Parallelism Saves Power

- 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

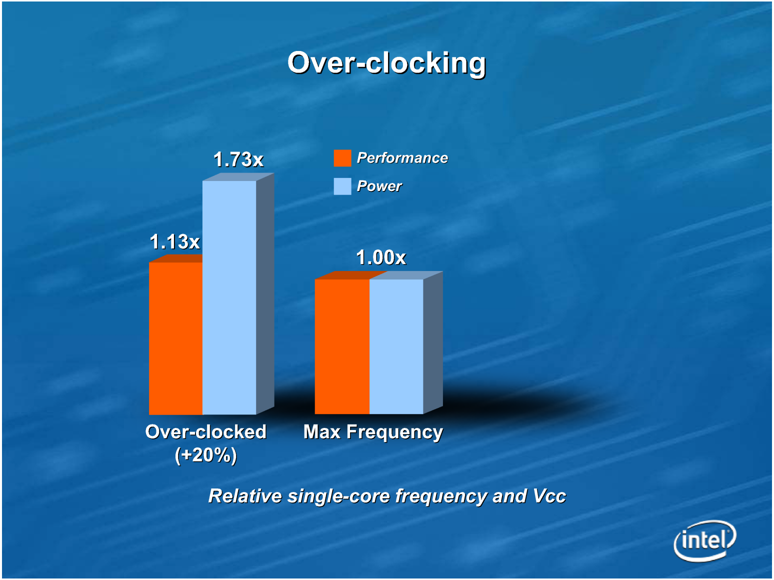
Parallelism Saves Power



Why Multi-Core?

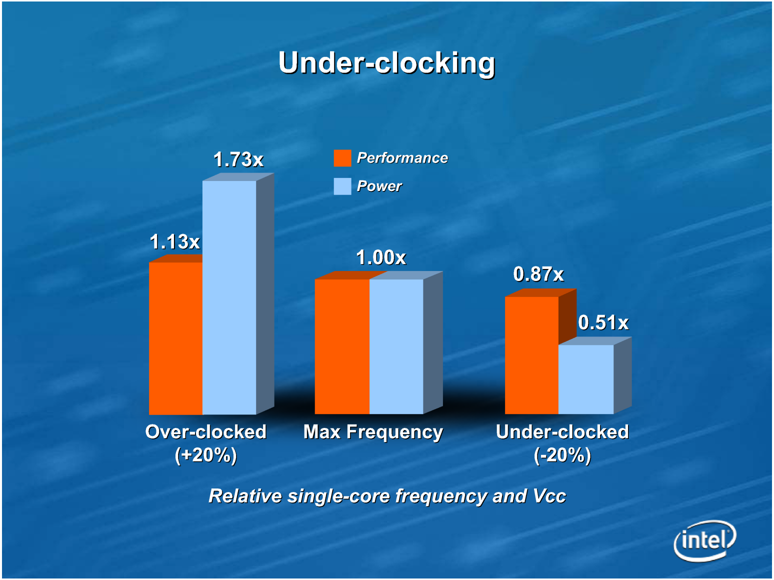
Courtesy of Intel

Parallelism Saves Power



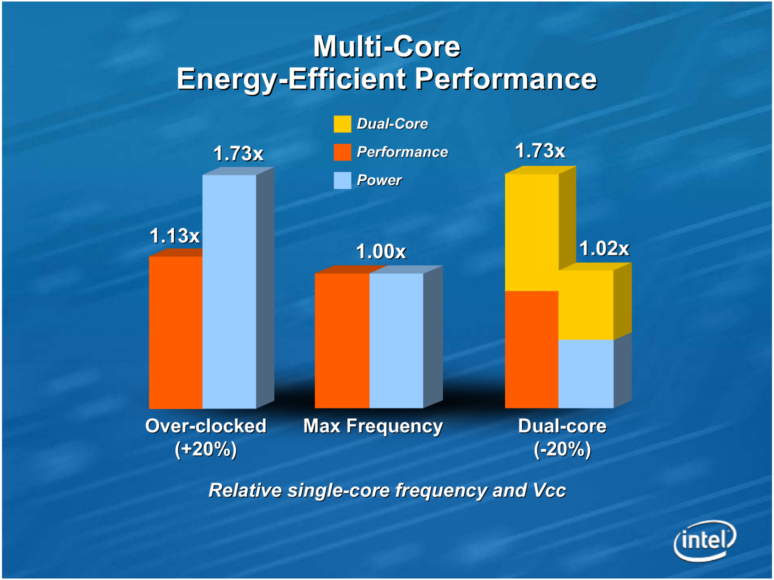
Courtesy of Intel

Parallelism Saves Power



Courtesy of Intel

Parallelism Saves Power



Courtesy of Intel

Power Saving

- 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

A Breathtaking Evolution...

EVOLUTION: TERAFLOP 1996

Sunday, 24 January 2010

Courtesy of Jez Wain (BULL)

A Breathtaking Evolution…

1,000 FLOPS PER WATT

Sunday, 24 January 2010

Courtesy of Jez Wain (BULL)

A Breathtaking Evolution...

EVOLUTION: 2.4 TERAFLOPS
2009

Sunday, 24 January 2010

Courtesy of Jez Wain (BULL)

A Breathtaking Evolution...

1,600,000 FLOPS PER WATT

Sunday, 24 January 2010

Courtesy of Jez Wain (BULL)

Multicore as The ultimate Solution?

- “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

Outline

1. Introduction to the lecture
    - Organization Forewords
    - Computational Science and Digital Revolution
    - Distributed Computing infrastructures: Technology, Engineering and Research
    - A Brief History of Parallel and Distributed Computing
2. Why All Computers Have to be Parallel
    - Moore Law and Computing Limits
    - Multiple Cores Save Power
    - The Memory Wall
3. Parallelism at the CPU level
    - Vector Processing
    - Pipelining
    - Instruction Level Parallelism
    - Multi-Threading
4. When One is Not Enough
    - SMPs, Multi-cores, NUMAs
    - General Purpose GPUs
    - Clusters
    - Grid/Desktop/Internet/Cloud Computing

The Memory Wall

The memory is a very common bottleneck that beginning programmers often don't think about

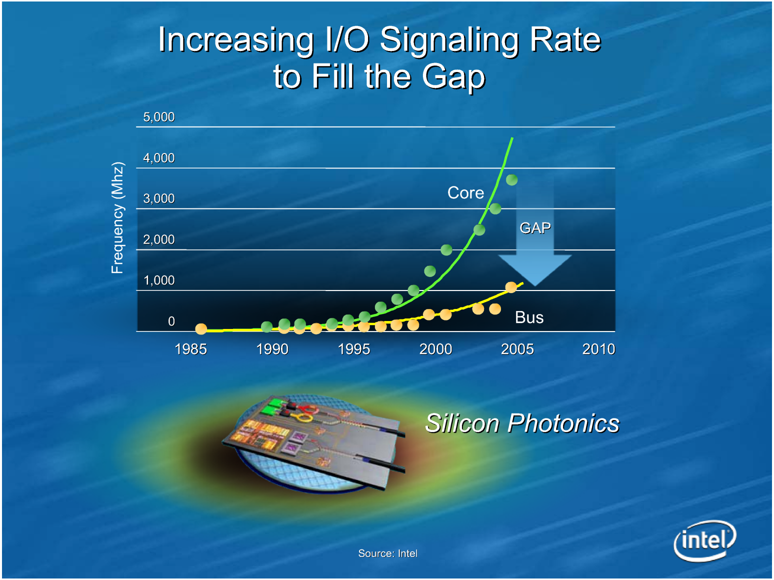
- When you look at code, you often pay more attention to computation

  $a[i] = b[j] + c[k]$

- 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
- 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

A Gap That Keeps Increasing



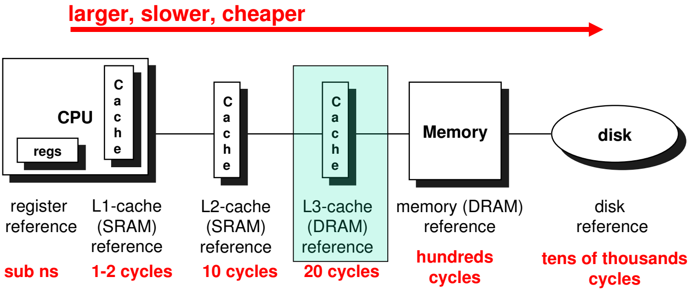
Increasing I/O Signaling Rate to Fill the Gap

Source: Intel

Courtesy of Intel

Caches! 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

Illustration with matrix multiplication

Consider the simple three nested loops algorithm

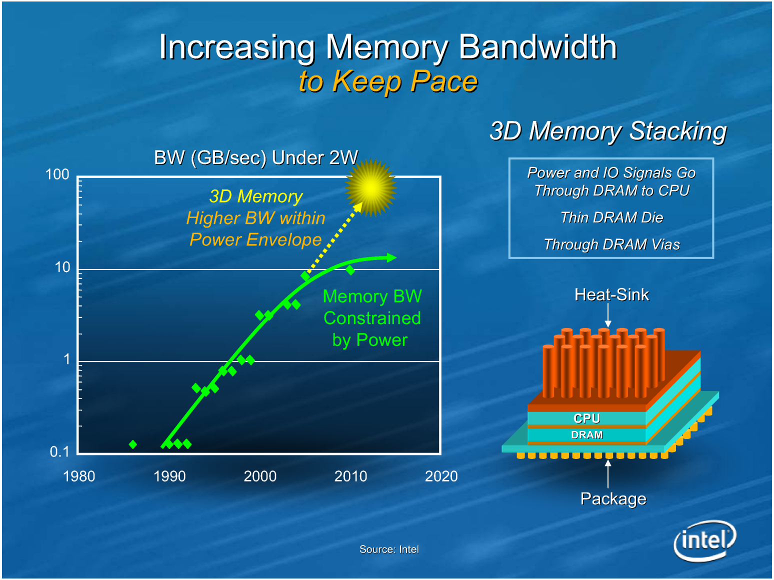
```
for(i=0;i<N; i++)
  for(j=0;j<N; j++)
    for(k=0;k<N; k++)
       C[i*N+j] += A[i*N+k] * B[k*N+j]
```

1. Simulate what happens with data access and cache usage
2. Change granularity by hand to improve locality
3. How would a divide and conquer algorithm behave ?...

Locality

- The memory hierarchy is useful because of “locality” (data is brought in bulk)
- **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
- The compiler can do some work for us regarding locality but unfortunately not everything. . .
- Programmers should keep a mental picture of the memory layout of the application, and reason about locality (**cache-aware/cache oblivious algorithms**).
- When writing concurrent code on a multi-core architecture, one must also thing of which caches are shared/private, which can be extremely complex.

3D Memory?



Increasing Memory Bandwidth

to Keep Pace

3D Memory Stacking

Power and IO Signals Go Through DRAM to CPU

Thin DRAM Die

Through DRAM Vias

Source: Intel

Courtesy of Intel

Conclusion

Exponential Growth $\rightsquigarrow$ **My laptop is a 10 years old supercomputer!** (and your phone is a 10 years old desktop)

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 algorithms (cache-aware/cache-oblivious; parallelism-aware/parallelism-oblivious).

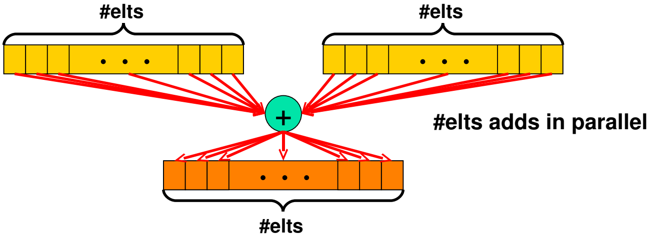
Outline

Outline

1. Introduction to the lecture
    - Organization Forewords
    - Computational Science and Digital Revolution
    - Distributed Computing infrastructures: Technology, Engineering and Research
    - A Brief History of Parallel and Distributed Computing
2. Why All Computers Have to be Parallel
    - Moore Law and Computing Limits
    - Multiple Cores Save Power
    - The Memory Wall
3. Parallelism at the CPU level
    - **Vector Processing**
    - Pipelining
    - Instruction Level Parallelism
    - Multi-Threading
4. When One is Not Enough
    - SMPs, Multi-cores, NUMAs
    - General Purpose GPUs
    - Clusters
    - Grid/Desktop/Internet/Cloud Computing

Vector Units

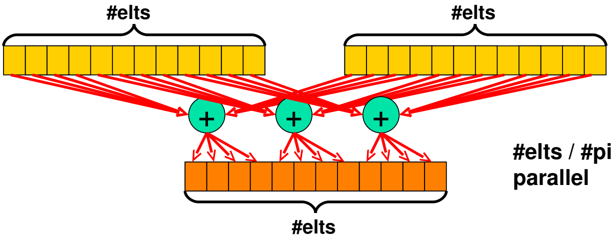
- A functional unit that can do elt-wise operations on entire vectors with a single instruction, called a vector instruction
  - These are specified as operations on vector registers
  - A “vector processor” comes with some number of such registers
    - MMX extension on x86 architectures



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



#elts / #pipes adds in parallel

Courtesy of Henri Casanova

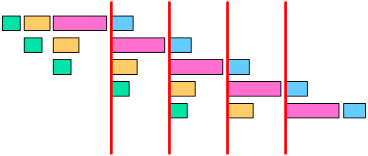
Vector processing (aka SIMD)

- Vector instruction specifies **multiple independent operations**. You **save the decoding part**
- Scientific computing uses tons of arrays (to represent mathematical vectors) and often does regular computation with these arrays. Hence, scientific code is "easy" to **vectorize**, i.e., to generate assembly that uses the vector registers and the vector instructions
- Pioneered in supercomputers and dominated supercomputer design through the 1970s into the 90s, notably the various **Cray platforms**
- Niche processors though. The rapid rise in the price-to-performance ratio of conventional microprocessor designs led to the vector supercomputer's demise in the later 1990s.
- Yet, the technique has been **integrated in off-the-shelf processors**:
  - Examples: VIS, MMX, SSE, AltiVec and AVX.
  - Also found in video game console hardware and graphics accelerators.
  - Cell processor 2000: IBM, Toshiba and Sony = 1 scalar processor + 8 vector processor
  - GPUs are somehow vector processors

Outline

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)

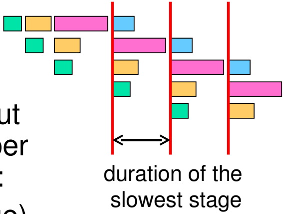


Time to do 1 task = 9
Time to do 2 tasks = 13
Time to do 3 tasks = 17
Time to do 4 tasks = 21
Time to do 10 tasks = 45
Time to do 100 tasks = 409

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)

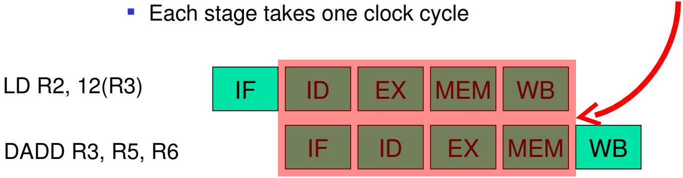
- Therefore, in an ideal pipeline, all stages would be identical (balanced pipeline)
- Question: Can we make computer instructions all consist of the same number of stage, where all stages take the same number of clock cycles?

Courtesy of Henri Casanova

Pipelining principle

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

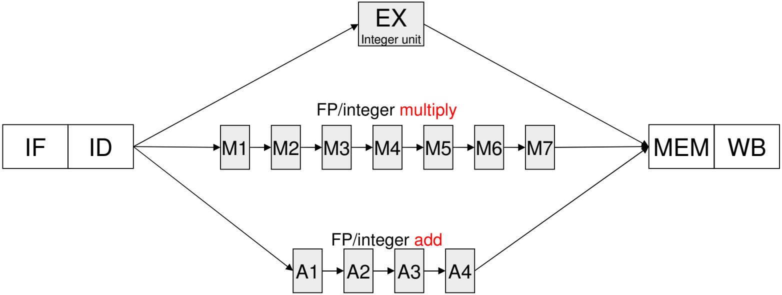


Concurrent execution of two instructions

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
  - **RISC and CISC appear in early 1970s**. But CISC are easier to compile and result in smaller program sizes, hence and fewer (slow) main memory accesses, which at the time resulted in a tremendous savings on memory, storage, as well as faster execution
  - 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
  - Itanium architecture uses pipelining
  - RISC processors are still around (ARM) and have an **excellent flop/W performance**...
- Bottom line: **pipelining is a pervasive** (and conveniently hidden) **form of concurrency** in computers today
  - Takes a computer architecture course to know all about it

Outline

1. Introduction to the lecture
    - Organization Forewords
    - Computational Science and Digital Revolution
    - Distributed Computing infrastructures: Technology, Engineering and Research
    - A Brief History of Parallel and Distributed Computing
2. Why All Computers Have to be Parallel
    - Moore Law and Computing Limits
    - Multiple Cores Save Power
    - The Memory Wall
3. Parallelism at the CPU level
    - Vector Processing
    - Pipelining
    - Instruction Level Parallelism
    - Multi-Threading
4. When One is Not Enough
    - SMPs, Multi-cores, NUMAs
    - General Purpose GPUs
    - Clusters
    - Grid/Desktop/Internet/Cloud Computing

Instruction Level Parallelism

Instruction Level Parallelism is the set of techniques by which performance of a pipelined processor can be pushed even further (e.g., by overlapping the execution of multiple instructions or by changing the order in which instructions are executed)

```
1.   e = a + b
2.   f = c + d
3.   m = e * f
```

- ILP can be done by the hardware
  - Dynamic instruction scheduling
  - Dynamic branch predictions and speculative execution
  - Multi-issue superscalar processors: multiple parallel pipelines, each of which is processing instructions simultaneously from a single instruction thread.
- ILP can be done by the compiler
  - Static instruction scheduling
  - Register renaming
  - Multi-issue VLIW processors
  - "Loop unrolling"

This technique is also widespread

- Yet, **no more instruction reordering without compromising correctness**

Outline

1. Introduction to the lecture
    - Organization Forewords
    - Computational Science and Digital Revolution
    - Distributed Computing infrastructures: Technology, Engineering and Research
    - A Brief History of Parallel and Distributed Computing
2. Why All Computers Have to be Parallel
    - Moore Law and Computing Limits
    - Multiple Cores Save Power
    - The Memory Wall
3. Parallelism at the CPU level
    - Vector Processing
    - Pipelining
    - Instruction Level Parallelism
    - Multi-Threading
4. When One is Not Enough
    - SMPs, Multi-cores, NUMAs
    - General Purpose GPUs
    - Clusters
    - Grid/Desktop/Internet/Cloud Computing

Multi-Threading ?

One of the “cool” innovations in the last decade has been the concept of a **Multi-threaded Architecture**

Here we’re talking about Hardware Support for threads:

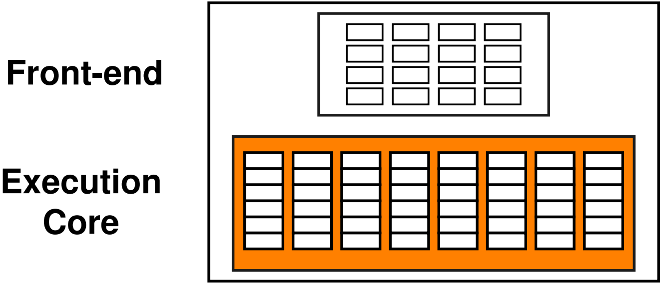
- Simultaneous Multi Threading (SMT)
- SuperThreading
- HyperThreading

Let’s start with a “simple” single-threaded processor:

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

From SMT to HyperThreading

Simplified Example CPU

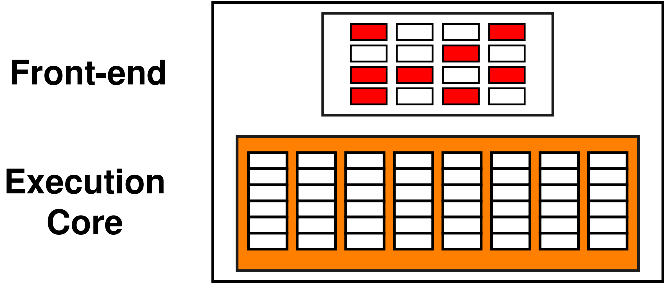


- The front-end can issue four instructions to the execution core simultaneously
  - 4-stage pipeline
- The execution core has 8 functional units
  - each a 6-stage pipeline

Courtesy of Henri Casanova

From SMT to HyperThreading

Simplified Example CPU

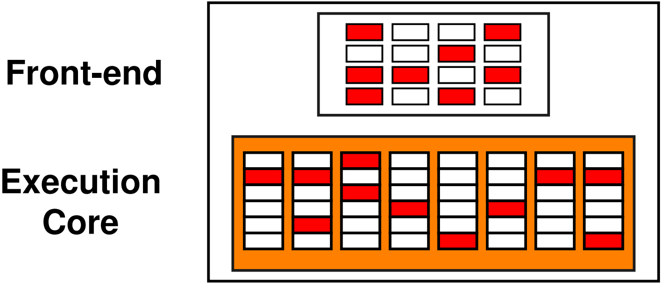


- The front-end is about to issue 2 instructions
- The cycle after it will issue 3
- The cycle after it will issue only 1
- The cycle after it will issue 2
- There is complex hardware that decides what can be issued

Courtesy of Henri Casanova

From SMT to HyperThreading

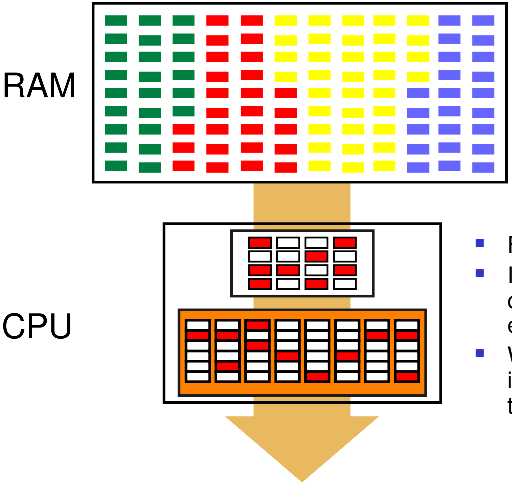
Simplified Example CPU



- 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

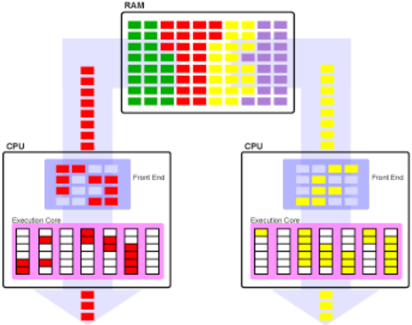
Multiple Threads in Memory



- 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?



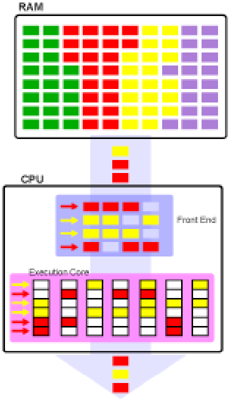
- Two threads execute at once, so threads spend less time waiting
- The number of “bubbles” is also doubled
- ➔ Twice as much speed and twice as much waste

Courtesy of Henri Casanova

From SMT to HyperThreading

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

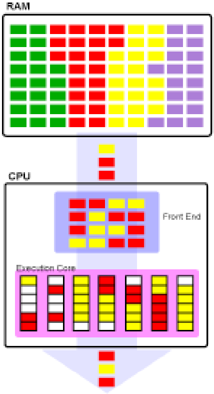


Courtesy of Henri Casanova

From SMT to HyperThreading

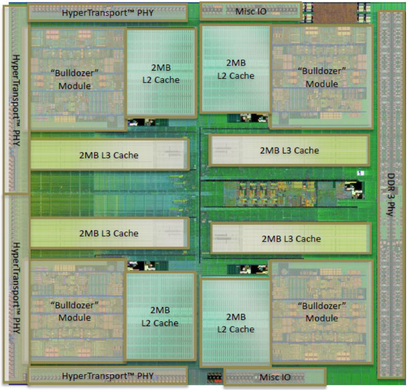
Hyper-threading

- 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

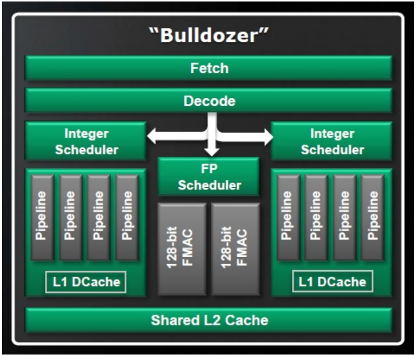


Courtesy of Henri Casanova

A Modern Off-the-shelf Processor

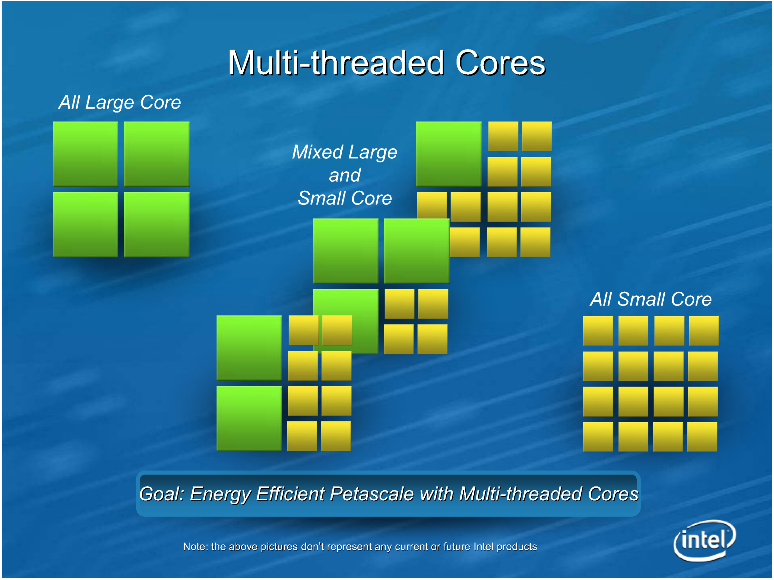


AMD FX-8150



Bulldozer module

And The Picture is Unlikely to get Simpler



Multi-threaded Cores

All Large Core

Mixed Large and Small Core

All Small Core

Goal: Energy Efficient Petascale with Multi-threaded Cores

Note: the above pictures don't represent any current or future Intel products

Courtesy of Intel

Outline

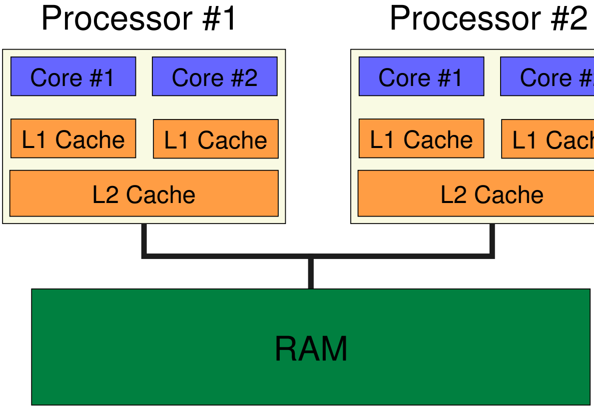
1. Introduction to the lecture
    - Organization Forewords
    - Computational Science and Digital Revolution
    - Distributed Computing infrastructures: Technology, Engineering and Research
    - A Brief History of Parallel and Distributed Computing
2. Why All Computers Have to be Parallel
    - Moore Law and Computing Limits
    - Multiple Cores Save Power
    - The Memory Wall
3. Parallelism at the CPU level
    - Vector Processing
    - Pipelining
    - Instruction Level Parallelism
    - Multi-Threading
4. When One is Not Enough
    - SMPs, Multi-cores, NUMAs
    - General Purpose GPUs
    - Clusters
    - Grid/Desktop/Internet/Cloud Computing

Outline

1. Introduction to the lecture
    - Organization Forewords
    - Computational Science and Digital Revolution
    - Distributed Computing infrastructures: Technology, Engineering and Research
    - A Brief History of Parallel and Distributed Computing
2. Why All Computers Have to be Parallel
    - Moore Law and Computing Limits
    - Multiple Cores Save Power
    - The Memory Wall
3. Parallelism at the CPU level
    - Vector Processing
    - Pipelining
    - Instruction Level Parallelism
    - Multi-Threading
4. When One is Not Enough
    - SMPs, Multi-cores, NUMAs
    - General Purpose GPUs
    - Clusters
    - Grid/Desktop/Internet/Cloud Computing

Cache Structure in SMPs with Multi-Cores

Multi-proc & multi-core systems



Processor #1

Core #1

Core #2

L1 Cache

L1 Cache

L2 Cache

Processor #2

Core #1

Core #2

L1 Cache

L1 Cache

L2 Cache

RAM

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

Going further: NUMA

- Before multicore chips, Symmetric Multiprocessor (**SMP**) and Non Uniform Memory Access (**NUMA**) systems were popular multiprocessor architectures.
- They are still in use in servers, clusters, or supercomputers.

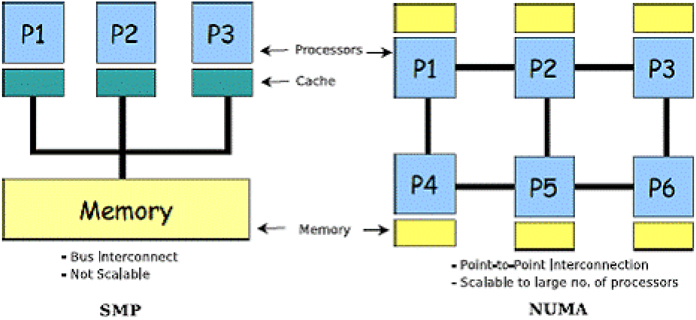
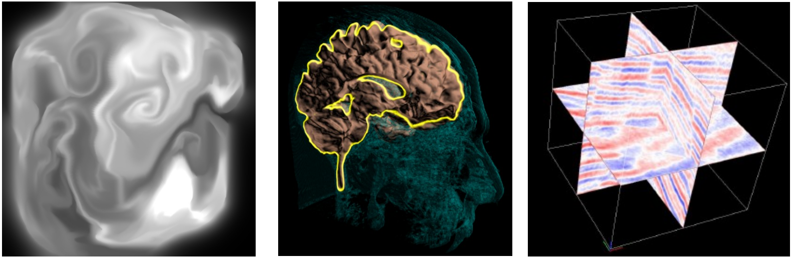


Figure 1: SMP versus NUMA

Outline

General Purpose GPUs

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

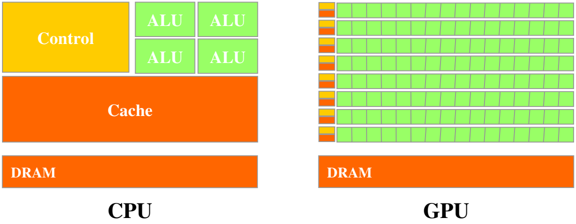


Courtesy Jens Krueger and Aaron Lefohn

Courtesy of Jean-François Méhaut

General Purpose GPUs

- 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

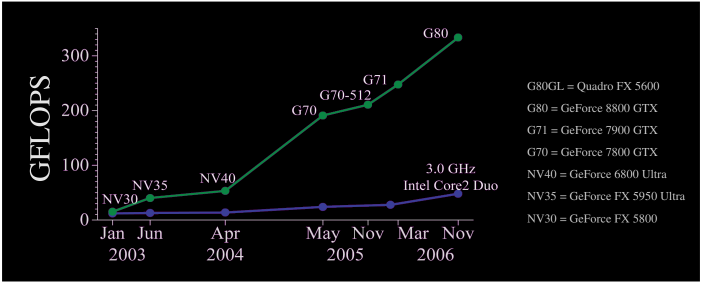


Courtesy NVIDIA

Courtesy of Jean-François Méhaut

General Purpose GPUs

- 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)



G80GL = Quadro FX 5600
G80 = GeForce 8800 GTX
G71 = GeForce 7900 GTX
G70 = GeForce 7800 GTX
NV40 = GeForce 6800 Ultra
NV35 = GeForce FX 5950 Ultra
NV30 = GeForce FX 5800

Courtesy NVIDIA

Courtesy of Jean-François Méhaut

General Purpose GPUs

```
Copy data from global to shared memory

Synchronization

Computation (iteration)

Synchronization

Copy data from shared to global memory
```

Courtesy of Jean-François Méhaut

General Purpose GPUs

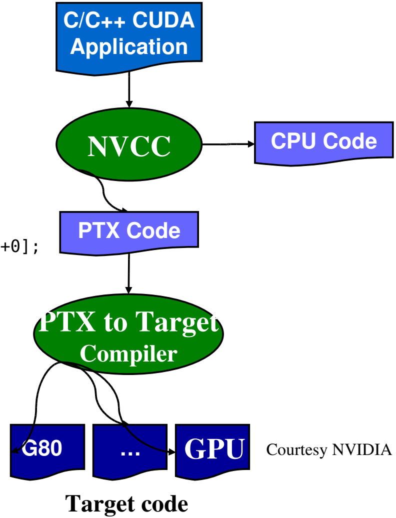
- C-extension programming language
  - No graphics API
    - Flattens learning curve
    - Better performance
  - Support debugging tools
- Extensions / API
  - Function type : __global__, __device__, __host__
  - Variable type : __shared__, __constant__
  - cudaMalloc(), cudaFree(), cudaMemcpy(),…
  - __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

General Purpose GPUs

- 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

1. Introduction to the lecture
    - Organization Forewords
    - Computational Science and Digital Revolution
    - Distributed Computing infrastructures: Technology, Engineering and Research
    - A Brief History of Parallel and Distributed Computing
2. Why All Computers Have to be Parallel
    - Moore Law and Computing Limits
    - Multiple Cores Save Power
    - The Memory Wall
3. Parallelism at the CPU level
    - Vector Processing
    - Pipelining
    - Instruction Level Parallelism
    - Multi-Threading
4. When One is Not Enough
    - SMPs, Multi-cores, NUMAs
    - General Purpose GPUs
    - **Clusters**
    - Grid/Desktop/Internet/Cloud Computing

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:

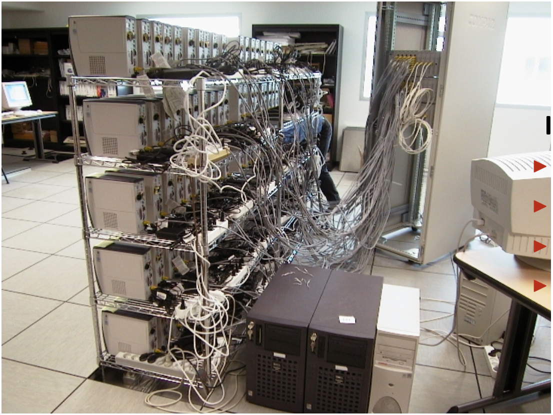
- computing nodes;
- I/O nodes;
- front (interactive) node.

A few examples

Berkeley NOW (1997)

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

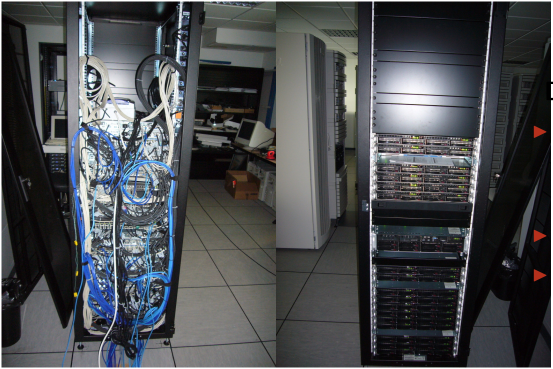
A few examples



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)

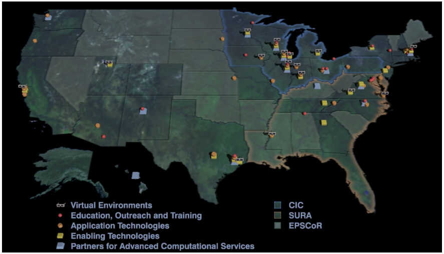
- 34 nodes (2 xeon quad cores $\rightsquigarrow$ 272 cores) with $2 \times 8Gb$ of RAM and $2 \times 160Gb$ of HD each.
- Infiniband.
- Giga Ethernet.

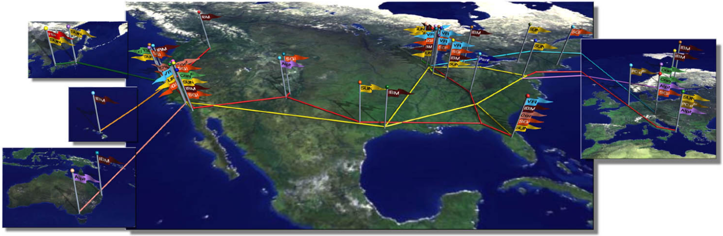
Outline

1. Introduction to the lecture
   - Organization Forewords
   - Computational Science and Digital Revolution
   - Distributed Computing infrastructures: Technology, Engineering and Research
   - A Brief History of Parallel and Distributed Computing
2. Why All Computers Have to be Parallel
   - Moore Law and Computing Limits
   - Multiple Cores Save Power
   - The Memory Wall
3. Parallelism at the CPU level
   - Vector Processing
   - Pipelining
   - Instruction Level Parallelism
   - Multi-Threading
4. When One is Not Enough
   - SMPs, Multi-cores, NUMAs
   - General Purpose GPUs
   - Clusters
   - **Grid/Desktop/Internet/Cloud Computing**

Computing Grids

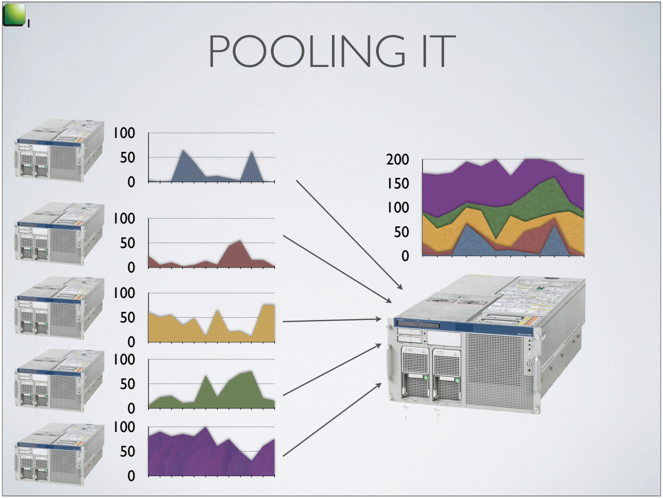
- You don't know where the energy comes from when you turn on your coffee machine.
- You don't need to know where your computations are done.





Cloud Computing. Eh wait!

- 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.



Sunday, 24 January 2010

Conclusion

In parallel computing, **Research**, **Technology**, and **Mass production** are tightly connected

- Research prototypes make their way to mass production
- Research ideas did not make their way because technology was not ready
- Some technology did not make their way because there was no market for mass production
- Mass production influences the way research is done

In this domain of computer science, research requires to anticipate technology (r)evolutions, market needs, and societal needs.
A few questions/comments to think about:

- Can we make general statements about systems whose technology evolves constantly ?
- Technological revolution or Societal revolution are not necessarily research revolution. How to discriminate novelty frome hype ?

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, Head of CRPC, 1994

[640K [of memory] ought to be enough for anybody.]

– Bill Gates

[There is no reason for any individual to have a computer in their home.]

– Ken Olson, president and founder of DEC, 1977

[I think there is a world market for maybe five computers.]

Thomas Watson, chairman of IBM, 1943.

I have not the smallest molecule of faith in aerial navigation other than ballooning or of expectation of good results from any of the trials we hear of.

– Lord Kelvin

[There is nothing new to be discovered in physics now. All that remains is more and more precise measurement.]

– Lord Kelvin