

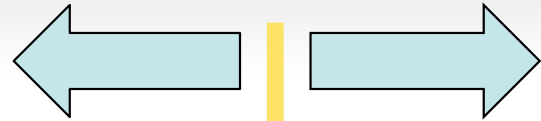
High Performance Computing on commodity PCs

Alfio Lazzaro
CERN openlab

Seminar at Department of Physics
University of Milan
January 14th, 2011

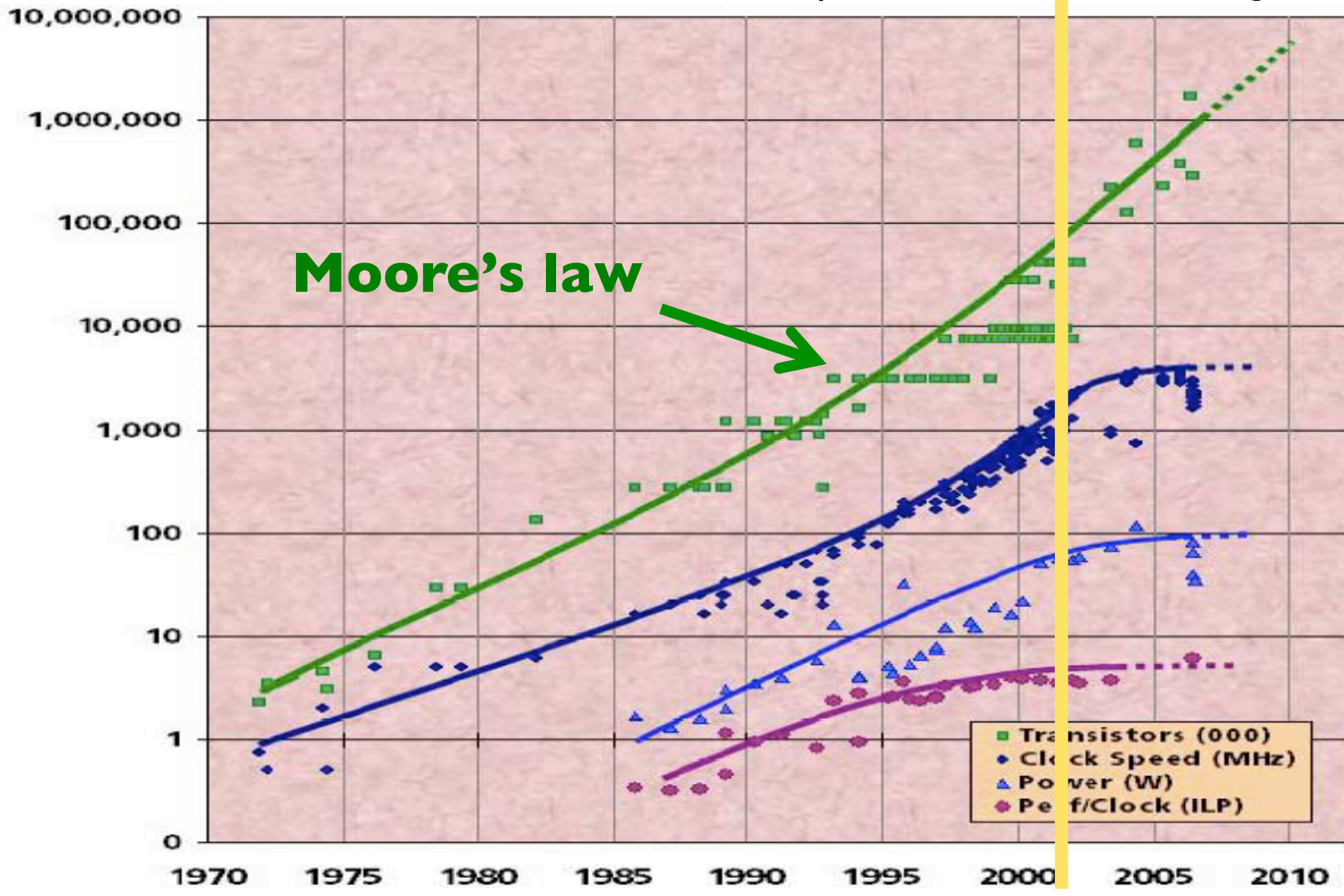


Computing in the years



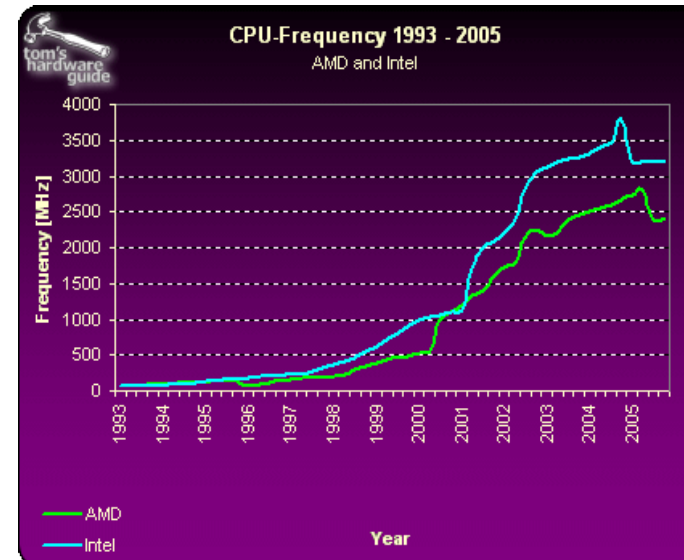
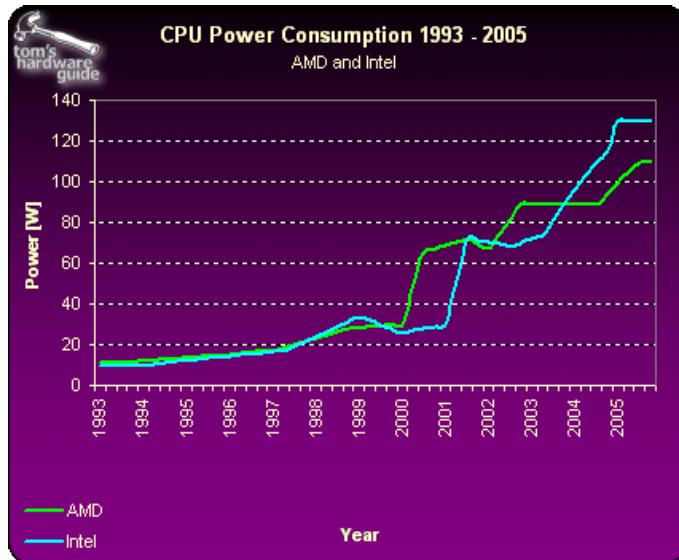
Transistors used to increase *raw-performance*

Increase *global performance*



Frequency scaling and power consumption

- The increase in performance was mainly driven by the increase of the clock frequency
 - Pentium Pro in 1996: 150 MHz
 - Pentium 4 in 2003: 3.8 GHz (~25x!)
- However, this brought to a significant increase in power consumption



<http://www.processor-comparison.com/power.html>

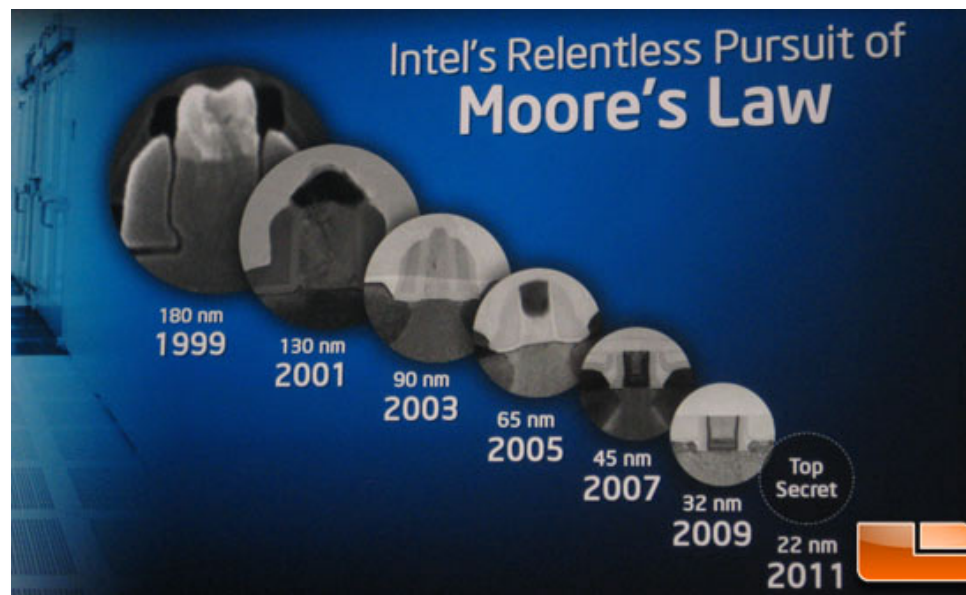
- Pollack's rule ($\text{perf} \approx \text{power}^{1/2}$)
 - 10% more performance costs about 20% more in power

- ❑ Power = EnergyPerInst * InstPerSecond
 - To keep power constant, EPI has to decrease at the same pace as increase in IPS (IPS = performance)
- ❑ $EPI = V_{cc}^2 * C + Leakage$
 - C is the capacitance
 - V_{cc}^2 is the supply voltage
- ❑ V_{cc} needs to be kept as low as possible
 - It cannot be reduced by big margins, since a low voltage level slows down the switching speed and imposes limits on the maximum frequency
- ❑ C is related to the physical properties of the material
 - Not easy to decrease
- ❑ At the time of the Pentium 4 (2003), the increase in frequency was no more possible because of increase in leakage currents

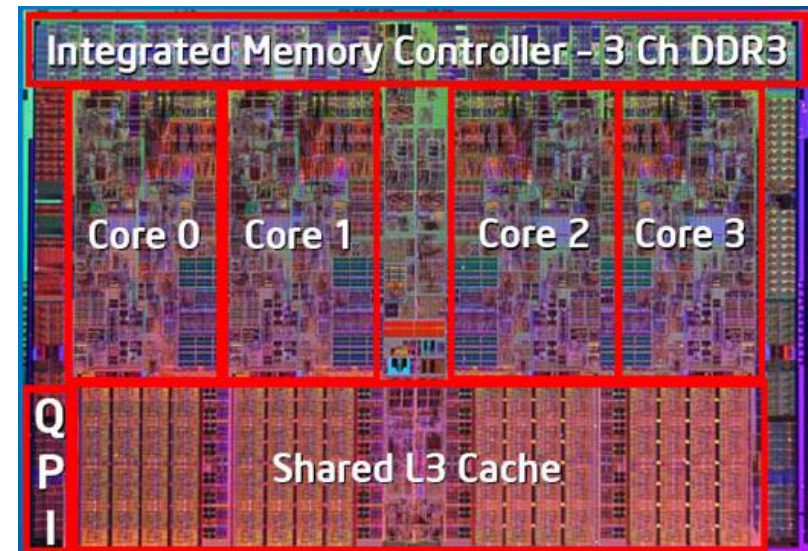
All these factors limited the increase in performance of the single computational unit (and it is very unlikely that the situation will change in the next 5-10 years)

Consequence of the Moore's Law

- Hardware continues to follow Moore's law
 - More and more transistors available for computation
 - More (and more complex) execution units: hundreds of new instructions
 - Longer SIMD (Single Instruction Multiple Data) vectors
 - More hardware threading
 - More and more cores



- ❑ To overcome the power problem, a turning point was reached and a new technology emerged: **multi-core**
 - Increase the “global” performance by adding new computational units (cores) on the same die (up to 12 cores currently)
 - Each core are complete processing units
 - Keep low frequency and consumption
- ❑ Dedicated architectures (accelerators):
 - GPGPU (NVIDIA, AMD, Intel MIC)
 - IBM CELL
 - FPGA (Reconfigurable computing)



The Challenge of Parallelization

- ❑ Keep in mind that the performance of a single core are not increasing as in the past
 - Applications which run on a single core (sequential) takes a very little benefit from multicore
 - Of course we can think to run more applications in parallel using the different cores, but still each application runs at the same speed
- ❑ A single application can take benefit from multi-core only if exhibits parallelism
 - **Think parallel!**
 - Write/rewrite your application using parallel concepts: very challenging in case of legacy software

When we want to parallelize

- ❑ **Reduction of the wall-time:** we want to achieve better performance, defined as (results response/execution) times
- ❑ **Memory problem:** large data sample, so we want to split in different sub-samples

Typical problem suitable for parallelization

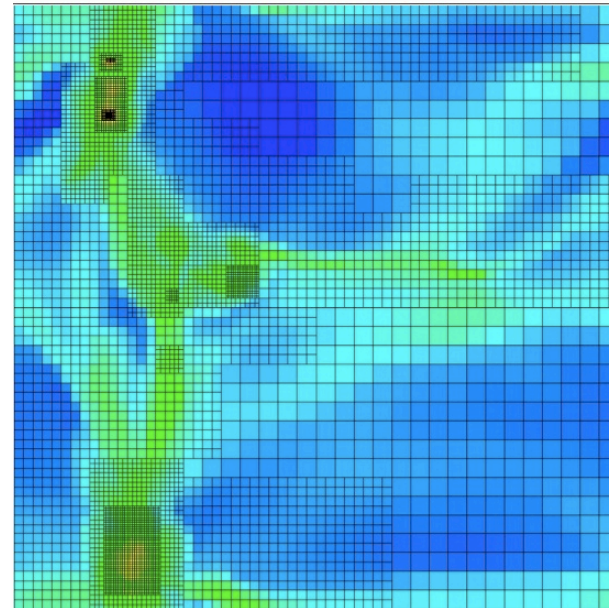
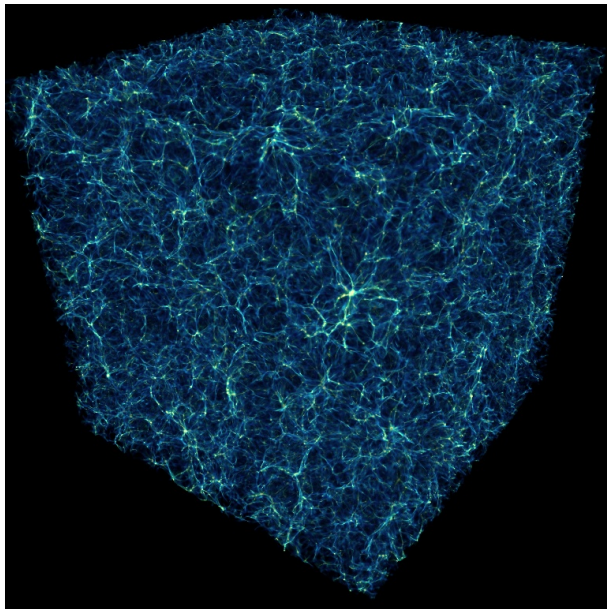
- ❑ The problem can be broken down into subparts (embarrassing parallelism):
 - Each subpart is independent of the others
 - No communication is required, except to split up the problem and combine the final results
 - **Ex: Monte-Carlo simulations**
- ❑ Regular and Synchronous Problems:
 - Same instruction set (regular algorithm) applied to all data
 - Synchronous communication (or close to): each processor finishes its task at the same time
 - **Ex: Algebra (matrix-vector products), Fast Fourier transforms**

Example of parallel problems

- ❑ Physics events are independent of each other
 - Embarrassing parallelism
 - For example, if we need to simulate 1'000'000 events, for instance, we can split the work in 10'000 jobs processing 100 events each, or 100 jobs processing 10'000 events each (or any other combination for that matter) and simply join the output files at the end
- ❑ Simulation of complex physics and chemical processes, universe simulation, brain simulation, ...
 - The parallel processes need to communicate each other many times during the execution
- ❑ Computing games: maybe the best example of application which has profited from parallel systems

Example: Galaxy formation

- Galaxy formation (<http://www.isgtw.org/?pid=1001250>)
 - a total of about **one billion individual grid cells**
 - adaptive mesh refinement



The 3D domain (2 billion light years of side).
Colors represent the density of the gas

Scalability issue in parallel applications

– Ideal case

- » our programs would be written in such a way that their performance would scale automatically
- » Additional hardware, **cores/threads or vectors**, would automatically be **put to good use**
- » Scaling would be as expect:
 - If the number of cores double, scaling (speed-up) would be 2x (or maybe 1.99x), but certainly not 1.05x

– Real case

- » Much more complicated situation...

- Parallelization introduces specific concepts (race conditions, data sharing, synchronization...) which are difficult to manage
 - Parallel programming is at least an order of magnitude more complex than sequential one
 - Parallel version of the code is much more difficult to optimize and debug
 - Parallel implementations can require rethinking the algorithms in a completely different way (for example for the accelerators)

- Handling existing complex and dispersed “legacy” software
 - Difficult to manage/share/tune resources (memory, I/O): better to rely in the support from OS and compiler
 - Coding and maintaining thread-safe software at user-level is hard
 - Need automatic tools to identify code to be made thread-aware
 - Example Geant4: 10K lines modified! (thread-parallel Geant4)

Speed-up (Amdahl's Law)

□ Definition:

S → speed-up

N → number of parallel processes

T_1 → execution time for sequential algorithm

T_N → execution time for parallel algorithm with N processes

$$S(N) = \frac{T_1}{T_N}$$

- Remember to balance the load between the processes. Final time is given by the slowest process!

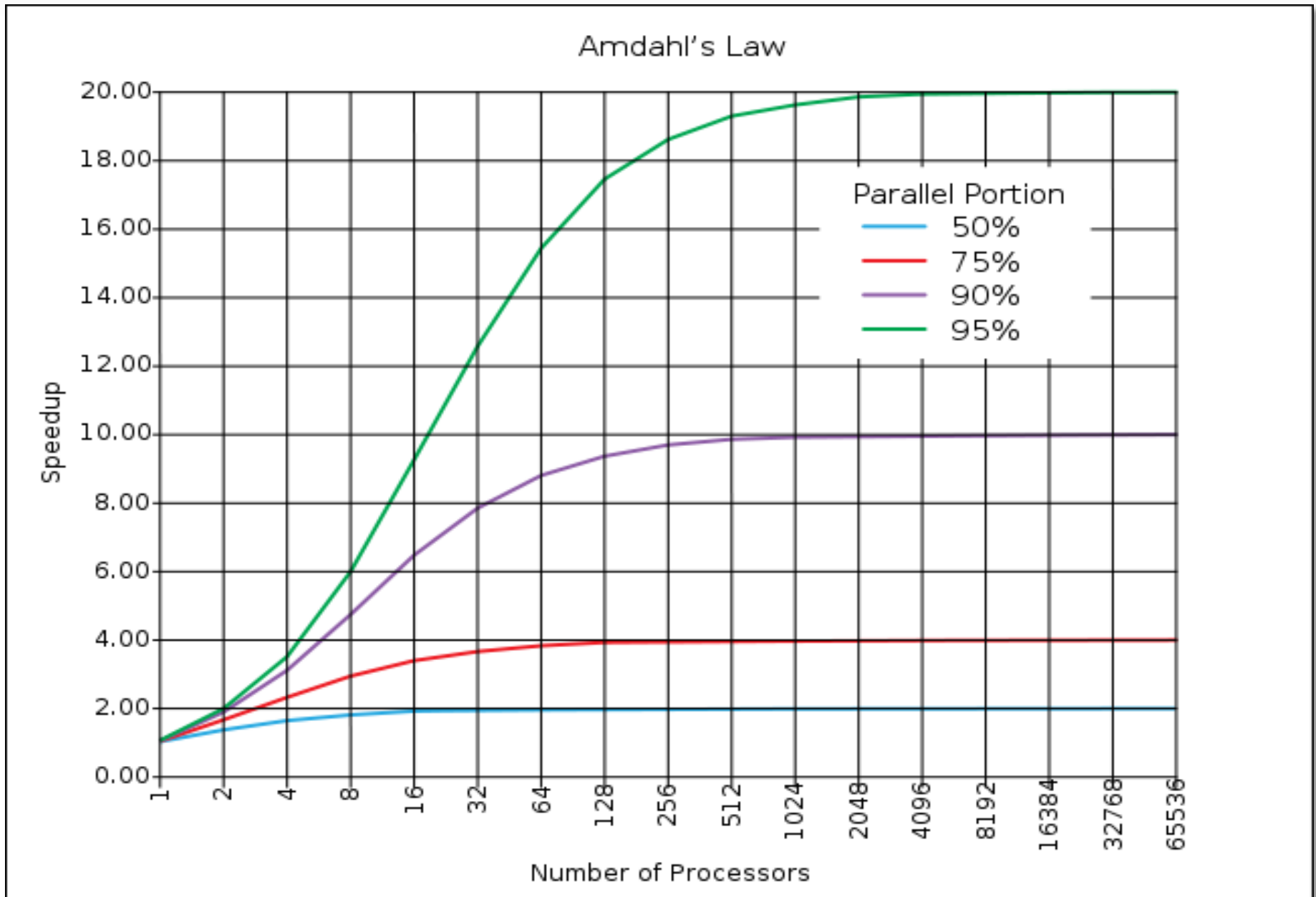
□ Maximum theoretical speed-up: **Amdahl's Law**

P → portion of code which is parallelized

$$S_{max}(N) = \frac{1}{(1 - P) + \frac{P}{N}}$$

- Implication: $S_{max}(N \rightarrow \infty) = \frac{1}{(1 - P)}$

- Need to find good algorithms to be parallelized!



Speed-up: Gustafson's Law

- Any sufficiently large problem can be efficiently parallelized

$$S_{max}(N) = 1 + P(N - 1)$$

S → speed-up

N → number of parallel processes

P → portion of code which is parallelized

- **Amdahl's law VS Gustafson's law**

- Amdahl's law is based on **fixed workload** or **fixed problem size**. It implies that the sequential part of a program does not change with respect to machine size (i.e, the number of processors). However the parallel part is evenly distributed by N processors
- Gustafson's law removes the fixed problem size on the parallel processors: instead, he proposed a **fixed time concept** which leads to scaled speedup for larger problem sizes

- ❑ Systems for high performance computing (HPC)
 - Basically massive parallel execution on several computational nodes connected by fast networks
 - Site www.top500.org lists the 500 most powerful systems (Top500)

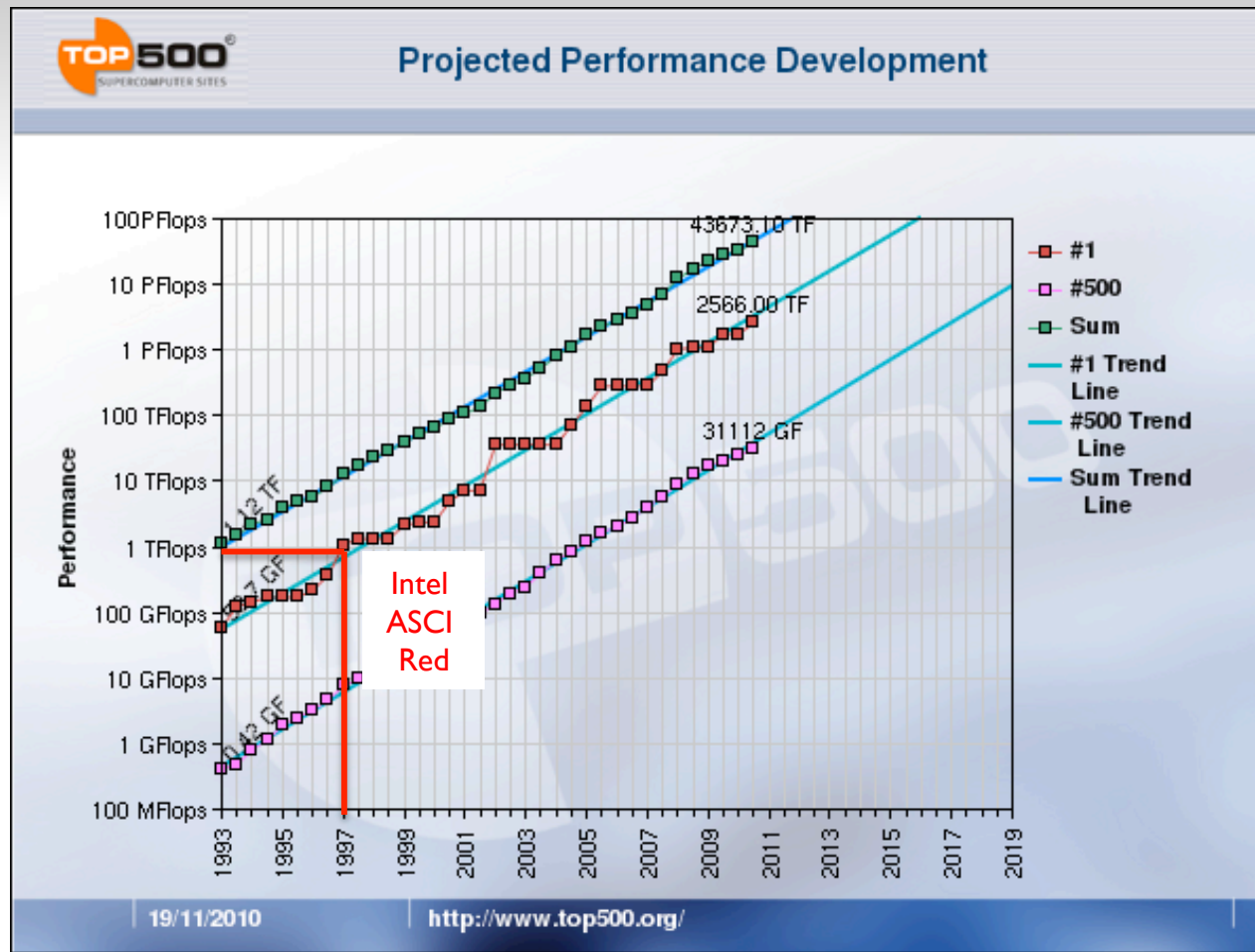
Rank	Site	Computer/Year Vendor	Cores	R_{max}	R_{peak}	Power
1	National Supercomputing Center in Tianjin China	Tianhe-1A - NUDT TH MPP, X5670 2.93Ghz 6C, NVIDIA GPU, FT-1000 8C / 2010 NUDT	186368	2566.00	4701.00	4040.00
2	DOE/SC/Oak Ridge National Laboratory United States	Jaguar - Cray XT5-HE Opteron 6-core 2.6 GHz / 2009 Cray Inc.	224162	1759.00	2331.00	6950.60
3	National Supercomputing Centre in Shenzhen (NSCS) China	Nebulae - Dawning TC3600 Blade, Intel X5650, NVidia Tesla C2050 GPU / 2010 Dawning	120640	1271.00	2984.30	2580.00
4	GSIC Center, Tokyo Institute of Technology Japan	TSUBAME 2.0 - HP ProLiant SL390s G7 Xeon 6C X5670, Nvidia GPU, Linux/Windows / 2010 NEC/HP	73278	1192.00	2287.63	1398.61
5	DOE/SC /LBNL/NERSC United States	Hopper - Cray XE6 12-core 2.1 GHz / 2010 Cray Inc.	153408	1054.00	1288.63	2910.00

Nov. 2010 list

- ❑ Very expensive systems for specific users (e.g. military agency)!!!!

Jaguar @ Oak Ridge (USA)





exaFLOPS

Intel ASCI Red @
Sandia Labs
First system @
1 teraFLOPS

9298 Pentium Pro @
200MHz
104 cabinets
230 m²
850 kW of power (not
including air
conditioning)

- ❑ Current trend foresees an exaFLOPS system by the end of 10s
 - Useful for some applications, e.g. real-time brain simulation and weather forecasting (Grand Challenges)
- ❑ **Nobody knows how to build such a monster!**
 - Maximum ~20MW is considered reasonable
 - Billions of parallel processes if extrapolating the current systems!

exaFLOPS and commodity computing

- ❑ The research on exaFLOPS systems involves big companies and research centers
 - Huge efforts and quantity of money
 - ❑ Although exaFLOPS systems are not directly connect to commodity systems, we should consider that the research on these systems can influence the entire computing systems world
 - Goal is to have commodity petaFLOPS systems
 - Normal users can use these systems for their research, but without paying “an arm and a leg”
 - ❑ 3 important parameters:
 - Performance ↑
 - Power consumption ↓
 - Cost ↓
- Let's focus on systems that maximize performance over power consumption and cost**
- So the performance must be normalize for the other two parameters
- ❑ Other parameters to take in account are: manageability, programmability, reliability, which are not easy to quantify...

- ❑ CPUs are for general tasks, like running operating systems
- ❑ Parallelism at different levels
 - Inside core using pipelined execution units, superscalar execution, vector units (SIMD) and hardware multi-threading
 - Currently vector units (128bit) support 4 single precision or 2 double precision operations in one go. Already this year new CPUs will double this number of operations (256bit units)
 - Inside the CPU using multicore: 2, 4, 6, 8, 12 cores
 - Already announced 16 cores by AMD and the number will increase in the future (many-core systems)
 - Between CPUs on the same motherboard (multi-socket):
 - 2 sockets are the standard for many users; 4 sockets and 8 sockets still too much expensive for general users
 - Complex configuration for CPUs connections and memory (NUMA)
- ❑ Potentially soon each computational node will have the possibility to run hundreds of parallel processes!

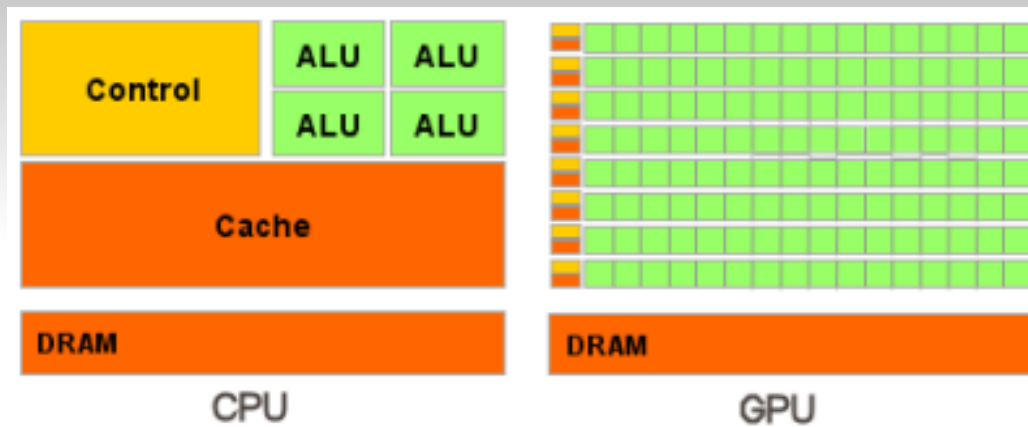
Common CPU Architectures

- Different architectures for CPUs available in the market
 - x86 and x86-64 (Intel and AMD): 15 – 150 Watts
 - The most common architectures
 - Intel Itanium (native 64bit): 130 – 185 Watts
 - Specific applications which require high performance (expensive)
 - Power-derived architecture from IBM: 10 – 200 Watts
 - Common in HPC (e.g. BlueGene) and in specific applications
 - Base for a lot of CPUs used in many fields, like Xenon CPU (Microsoft Xbox 360) and Broadway (Nintendo Wii)
 - SPARC architecture from Oracle: ~140 Watts
 - Specific applications which require high performance (expensive)

Common Low-power CPU Architectures

- Mainly targeting for mobile market and embedded systems, where power consumption is the main concern
 - ARM (32bit): ~1 Watt
 - 75% of mobile market (e.g. cellular phones)
 - Interesting architecture with impressive ratio performance/power consumption, but limited versatility
 - Several projects to build systems with hundreds (thousands) of ARM for massive parallelization
 - VIA Nano (64bit): ~ 3 Watts
 - Compatible with x86-64 instructions, the main target is ultra-mobile laptop (netbook) PCs
 - Intel Atom (64bit): 0.7 – 8 Watts
 - Compatible with x86-64 instructions
 - Several projects to build clusters of Atom CPUs for massive parallelization with high performance/consumption ratio

- Used as co-processors with the CPU for specialized tasks
 - Generally for intensive floating point operations
 - Demand for computing power grows faster than the compute capabilities of modern processors
 - Example of applications: 3D graphics
 - Increase performance, but reduce versatility
- The prominent example is graphics co-processors (GPUs)
 - Mostly for gaming and interactive entertainment
 - It is now very common to use GPUs for HPC (GPGPU)
 - Very attractive solution to have “cheap” FLOPS
 - Increasing accessibility (every PC has a GPU...)
 - Good ratio performance/power at good price
- Several other accelerators
 - Intel MIC (x86-64 compatible), expected in 2012
 - CELL processor
 - FPGAs



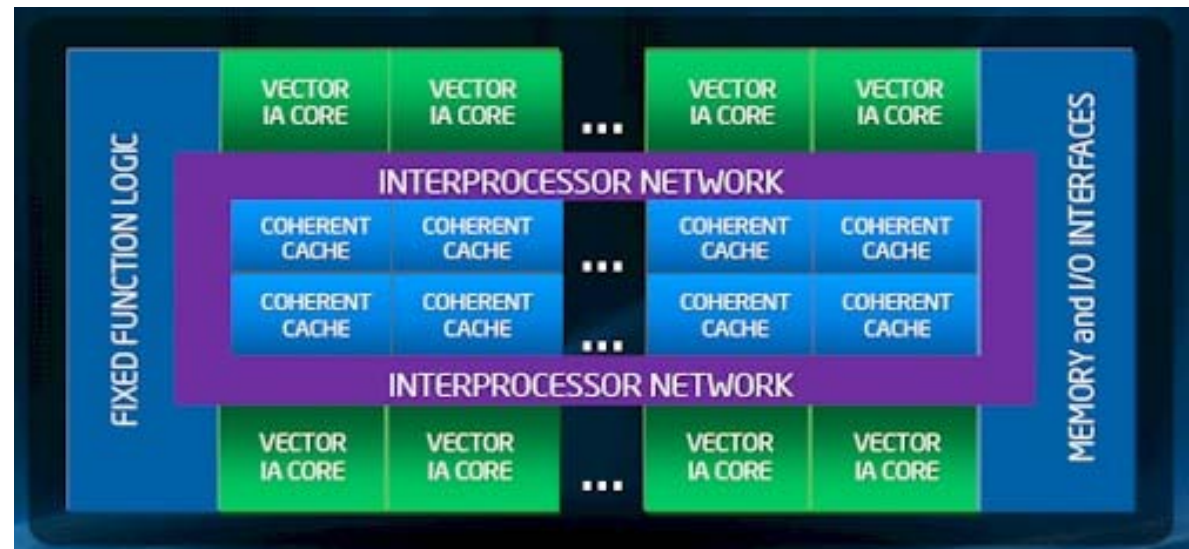
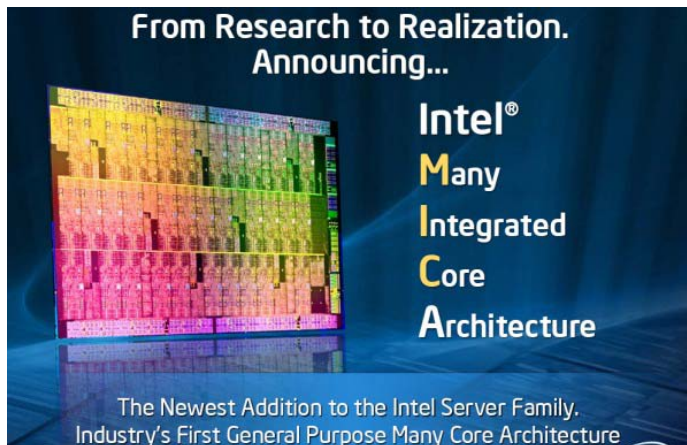
- ❑ A lot of interest is growing around GPUs for HPC
 - 4 out of top 10 supercomputers in Top500 have GPUs
 - 70% of performance in mixed CPU-GPU computers is provided by GPUs
 - If considering the ratio performance/power consumption (Green500 list), then 8 out of the top 10 supercomputers have GPUs
- ❑ Impressive performance (3x – 7x than a multi-core CPU), but high power consumption (up to 250Watts)
- ❑ Great performance using single floating point precision (IEEE 754 standard): up to **1 teraFLOPS** (w.r.t ~150 GFLOPS of a multicore CPU): same performance of the ASCI Red supercomputer!!!!
- ❑ Completely different software paradigm!
 - **Need to rewrite most of the code to benefit of this massive parallelism (thread parallelism), especially memory usage: it can be not straightforward...**

- Two main vendors
 - NVIDIA with “Fermi” architecture: peak 1TFLOP Single Precision (50% Double Precision) @ ~225W (Tesla M2050)
 - GeForce GPUs have same SP performance, half for DP
 - AMD with “Cypress” architecture: peak 2.72 TFLOPS SP (544 GFLOPS DP) @ ~220W (AMD 5870)
- Intel has its GPUs series (limited performance)
- PCIe form factor (data transfer across PCIe can be a bottleneck for most applications)
 - AMD and Intel propose to integrate GPUs on the same CPU die for fast GPU ↔ CPU connection
- Deviations from the IEEE754 floating point standard
 - Denormals, NaNs, rounding, Precision lower

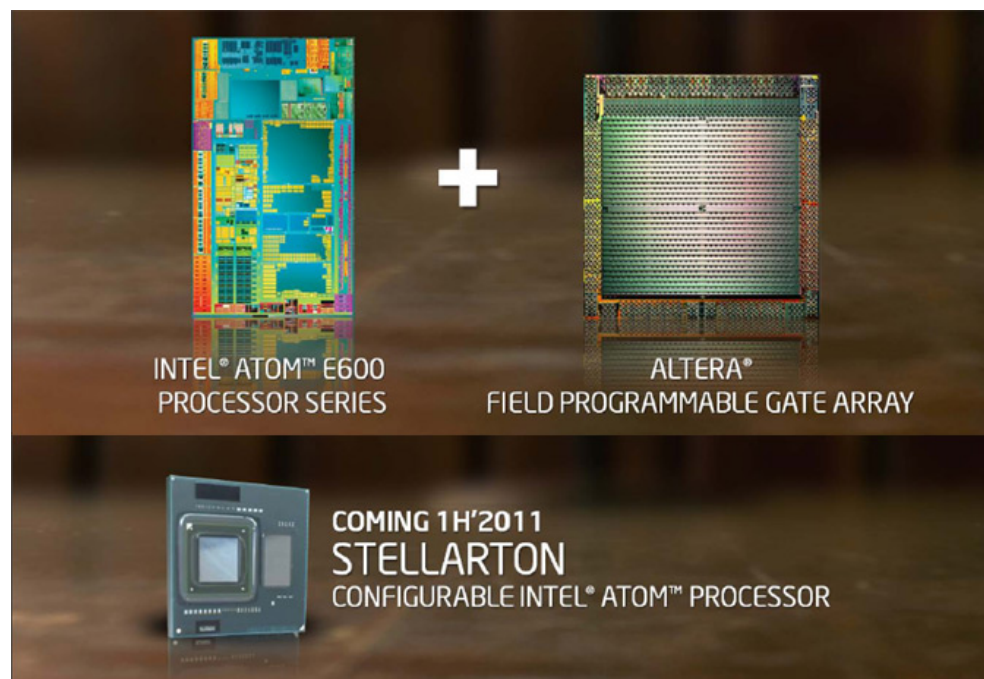


Intel “Many Integrated Cores” architecture

- ❑ Announced at ISC10 (June 2010)
- ❑ A research processor, originally conceived as a GPU
 - x86-64 instruction set
 - 32 cores @ 1.2 GHz + 4-way hardware multithreaded + 512-bit vector units: ~ 1TFLOPS SP (50% DP)
 - Limited memory: up to 2GB
 - PCIe card
- ❑ Commercial version in 2012(?): 22nm (?)
 - Many-core (>50 cores) + 4-way hardware multithreaded + 512-bit vectors
 - In project a complete independent system, i.e. more than a just simple accelerator...

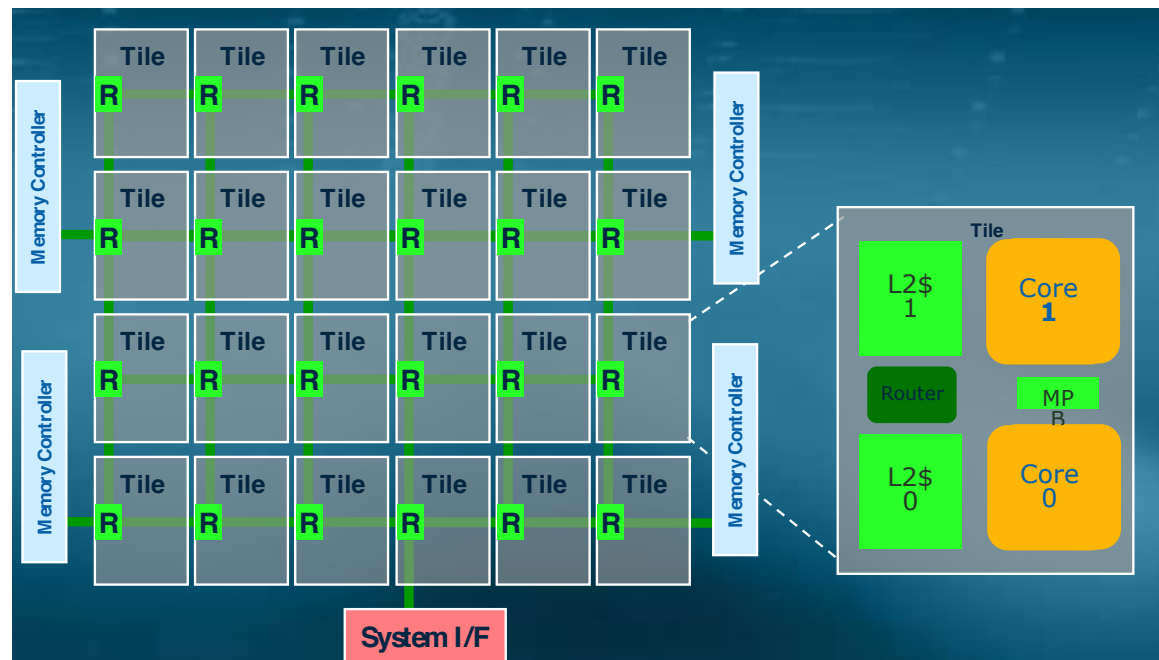
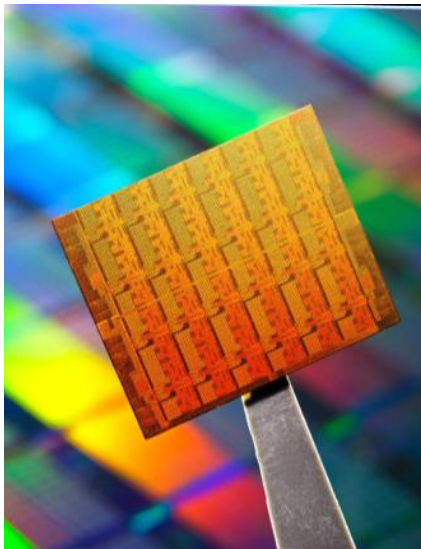


- ❑ Hardware programming
 - FPGA can be considered as raw system of transistors that can be programmed via software
 - Very hard to program for general tasks
 - Great performance for very specialized tasks, with very low power consumption
- ❑ FPGAs can be used to help CPUs in several tasks
 - Some research project to build HPC systems (e.g. Maxwell at EPCC (Edinburgh), Janus by INFN in Italy)
 - Interesting proposal by Intel last year: Atom CPU + FPGA



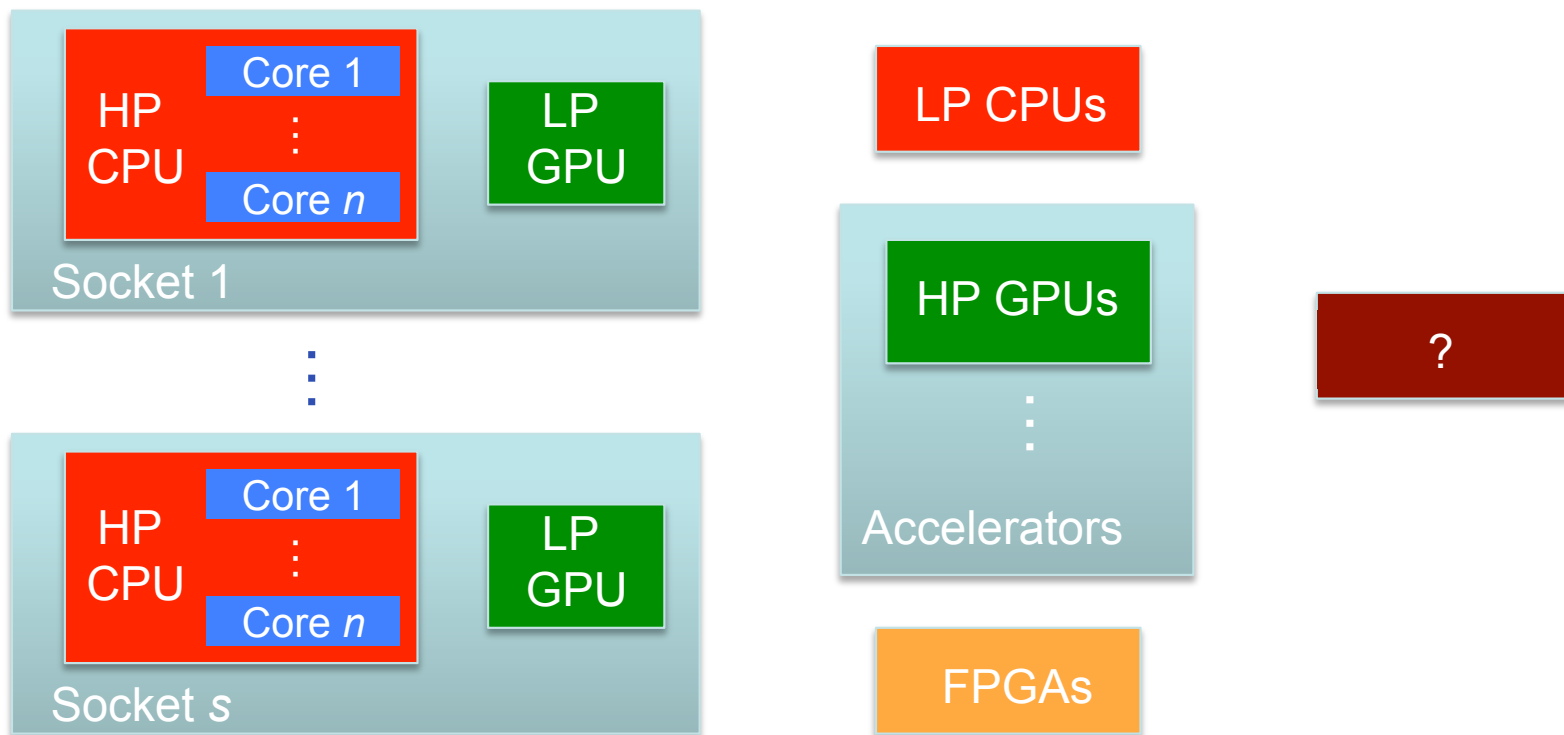
Single-chip Cloud Computer

- ❑ 48 Core Research Microprocessor
 - Experimental Research Processor – Not A Product
- ❑ “Cluster-on-die” architecture (new concept)
 - 48 Pentium Processor cores
- ❑ Interesting possibilities: a lot of parameters can be configured via software, such as operational voltage and frequency



Heterogeneous systems

- All systems give the best performance for specific tasks
 - There is not a unique system which is suitable for everything!
- It is a common understanding that future systems for computation will be an “heterogeneous” systems, where each sub-system will properly perform his part of execution

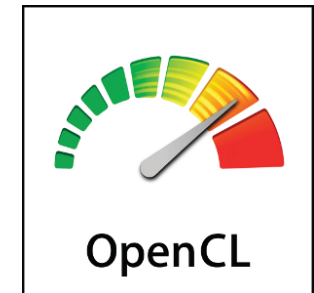


HP = High Performance; LP = Low Performance

Parallelization in the code languages

- ❑ **Very challenging!!!**
- ❑ **Automatic parallelization** of a sequential program by a compiler is the holy grail of parallel computing
 - automatic parallelization has had **only limited success** so far...
- ❑ Parallelization must be **explicitly declared** in a program (or at the best partially implicit, in which a programmer gives the compiler directives for parallelization)
 - Some languages define parallelization as own instructions
 - High Performance Fortran
 - Chapel (by Cray)
 - X10 (by IBM)
 - C++1x (the new C++ standard)
 - In most cases parallelization relays on external libraries
 - Native: pthreads/Windows threads
 - OpenMP (www.openmp.org)
 - Intel Threading Building Blocks (TBB)
 - OpenCL (www.khronos.org/ocl)
 - CUDA (by NVIDIA, for GPU programming)

- ❑ I don't think there will an unique language to program everything
 - Good projects is OpenCL, an open standard for programming heterogeneous parallel processors
 - Many prominent members of the working group: AMD, ARM, IBM, Intel, NVIDIA and many others
 - Possibility to run the same code on many platforms
 - Based on the C99 standard
 - Suited for writing computation kernels
 - Task-based and data-based parallelism
 - More at <http://www.khronos.org/opencvl/>
- ❑ More thinking is required at every software level, starting from the operating systems



- ❑ Hardware is definitely changing
 - More than a normal evolution, not yet a revolution (for the moment...)
 - The situation will not change in future (and at least for the next 5 years)
 - More and more parallelism in the hardware
 - Research is ongoing
- ❑ Some communities have successfully parallelized their code
 - High Performance Computing applications, mainly based on algebra applications
 - Game companies
- ❑ Needs to change to think algorithm: **think parallel, write parallel!**
 - Need to teach parallel techniques just as normal computing course
 - It maybe the case that current software will not properly work in the future hardware
 - Some tools can alleviate the migration, but it can be not enough...
 - Huge effort from the software side
 - Maybe we are already behind the schedule...
 - Users contribution is critical! Be aware and start to parallelize your code as soon as possible...

Q & A



CERN
openlab

Foster's Design Methodology

□ Partition

- Divide problem into tasks

□ Communicate

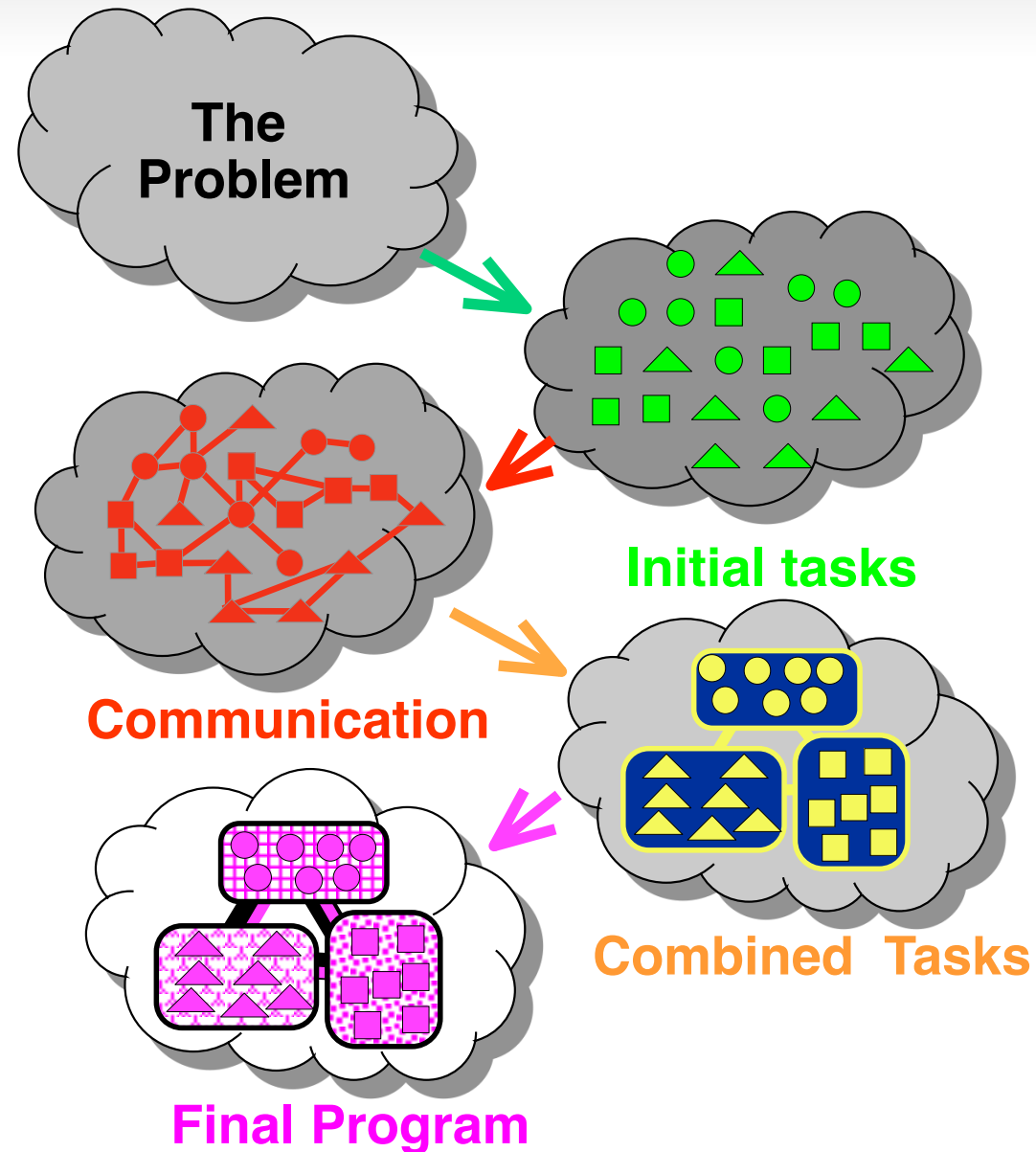
- Determine amount and pattern of communication

□ Agglomerate

- Combine tasks

□ Map

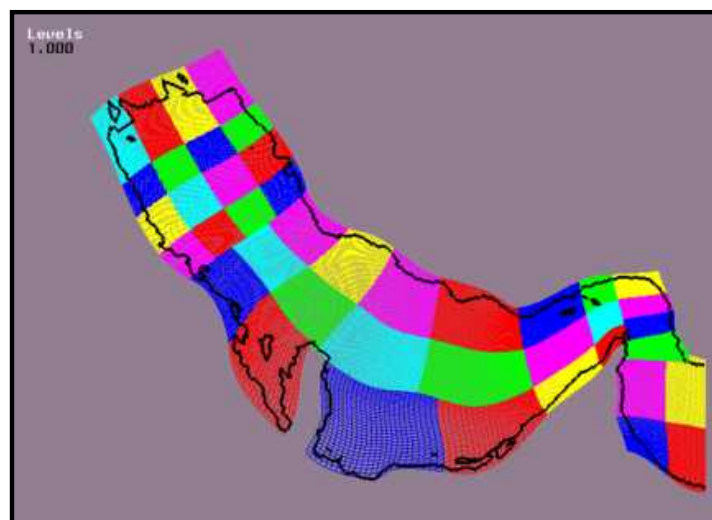
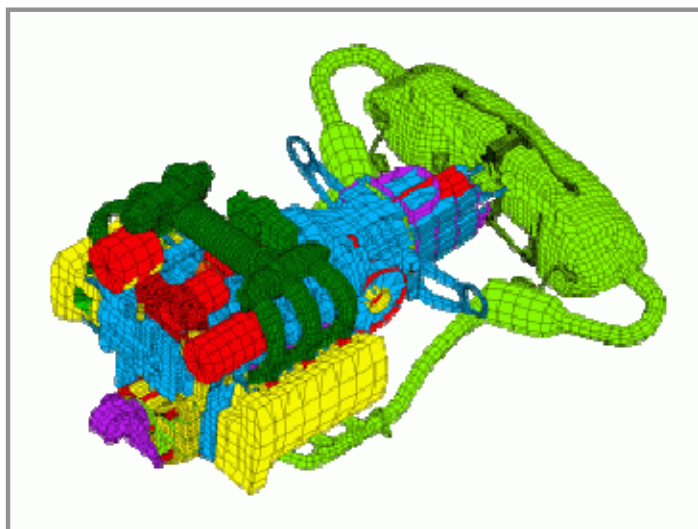
- Assign agglomerated tasks to created threads



From "Designing and Building Parallel Programs" by Ian Foster

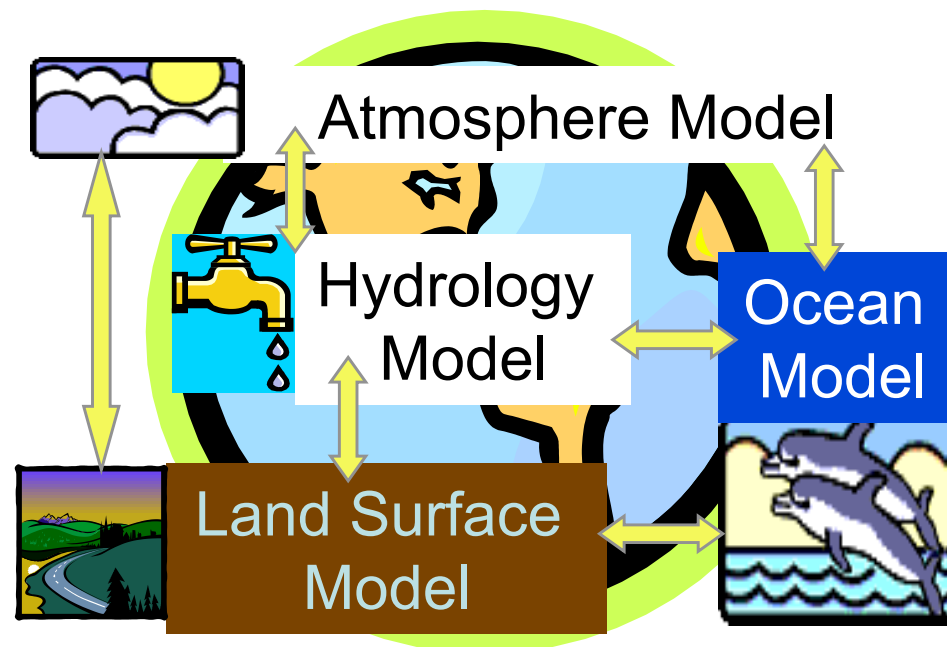
Domain (Data) Decomposition

- ❑ Exploit large datasets whose elements can be computed independently
 - Divide data and associated computation amongst threads
 - Focus on largest or most frequently accessed data structures
 - **Data parallelism: same operation(s) applied to all data**



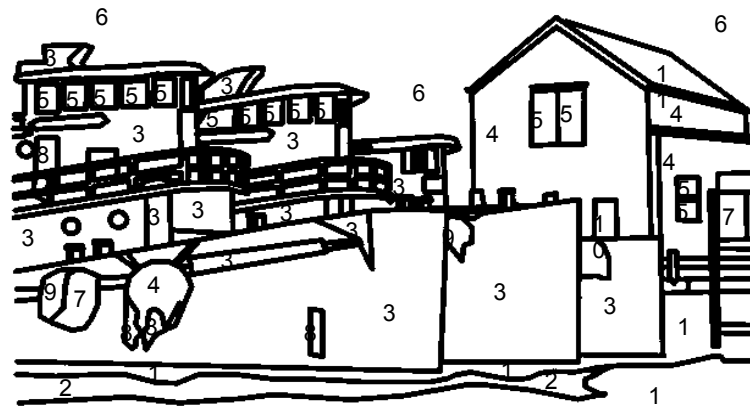
Functional Decomposition

- Divide computation based on a natural set of independent functions
 - Predictable organization and dependencies
 - Assign data for each task as needed
 - Conceptually a single data value or transformation is performed repeatedly



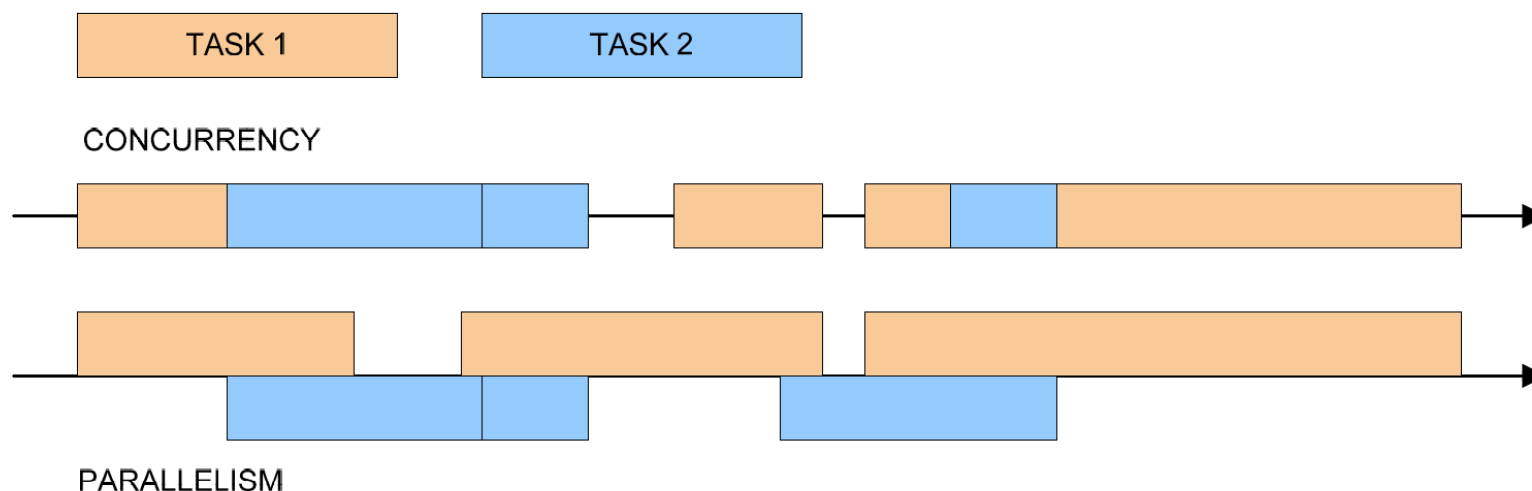
Activity (Task) Decomposition

- Divide computation based on a natural set of independent tasks
 - Non deterministic transformation
 - Assign data for each task as needed
 - Little communication
- Example: Paint-by-numbers
 - Painting a single color is a single task



Definition of concurrency/parallelism

- ❑ **Concurrent programming:** the program can be logically split in independent parts (threads)
 - Concurrent programs can be executed sequentially on a **single CPU** by interleaving the execution steps of each computational process
 - Benefits can arise from the use of I/O resources
 - Example: a thread is waiting for a resource reply (e.g. data from disk), so another thread can be executed by the CPU
 - Keep CPU busy as much as possible
- ❑ **Parallel execution:** Independent parts of a program execute simultaneously



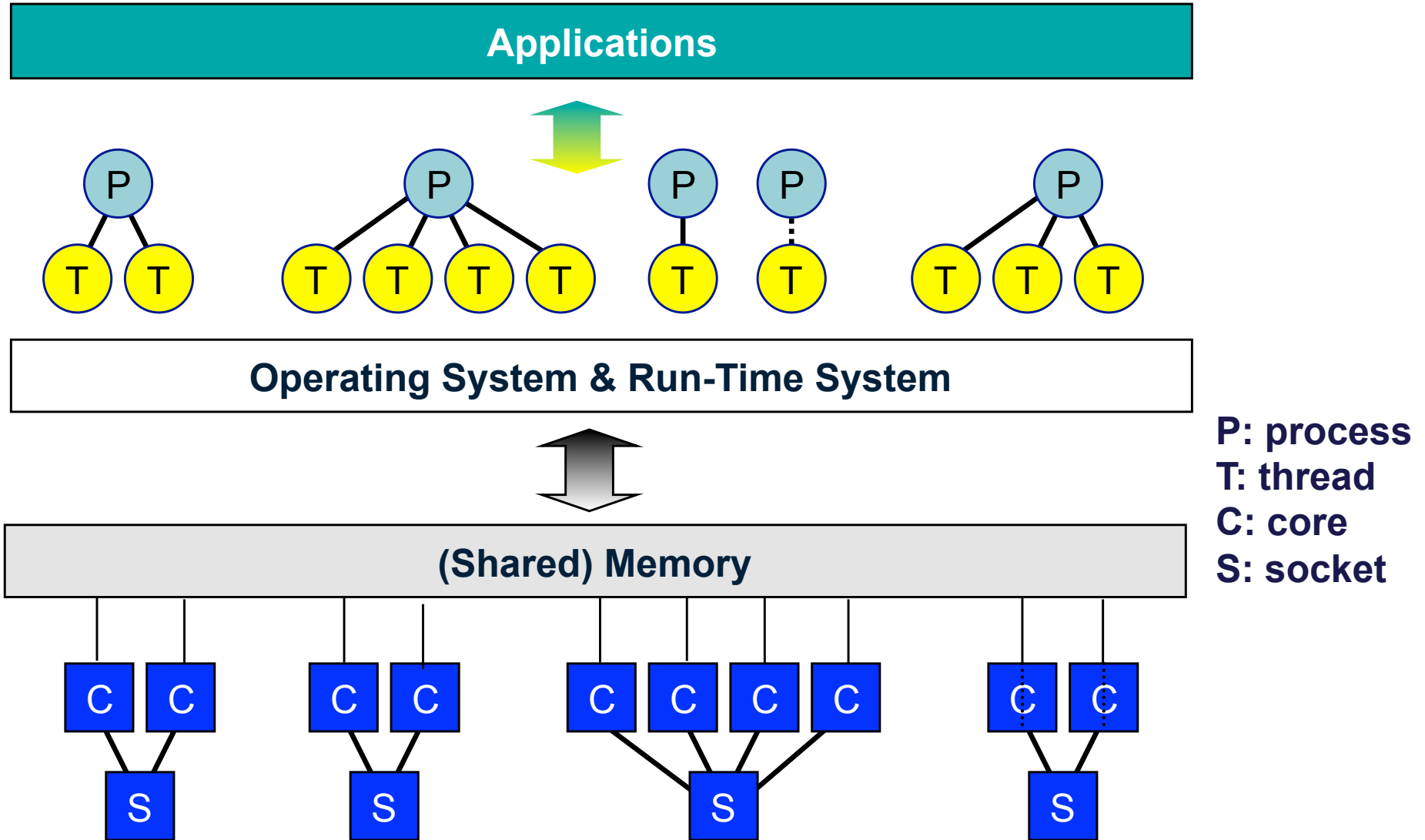
Other some basic definitions

Software level

- ❑ **Process**: an instance of a computer program that is being executed (sequentially). It contains the program code and its current activity: its own “address space” with all the program code and data, its own file descriptors with the operating system permission, its own heap and its own stack.
- ❑ **SW Thread**: a process can *fork* in different threads of execution. These threads run in the same address space, share the same program code, the operating system resources as the process they belong to. Each thread gets its own stack.

Hardware level

- ❑ **Core**: unity for executing a software process or thread: execution logic, cache storage, register files, instruction counter (IC)
- ❑ **HW Thread**: addition of a set of register files plus IC



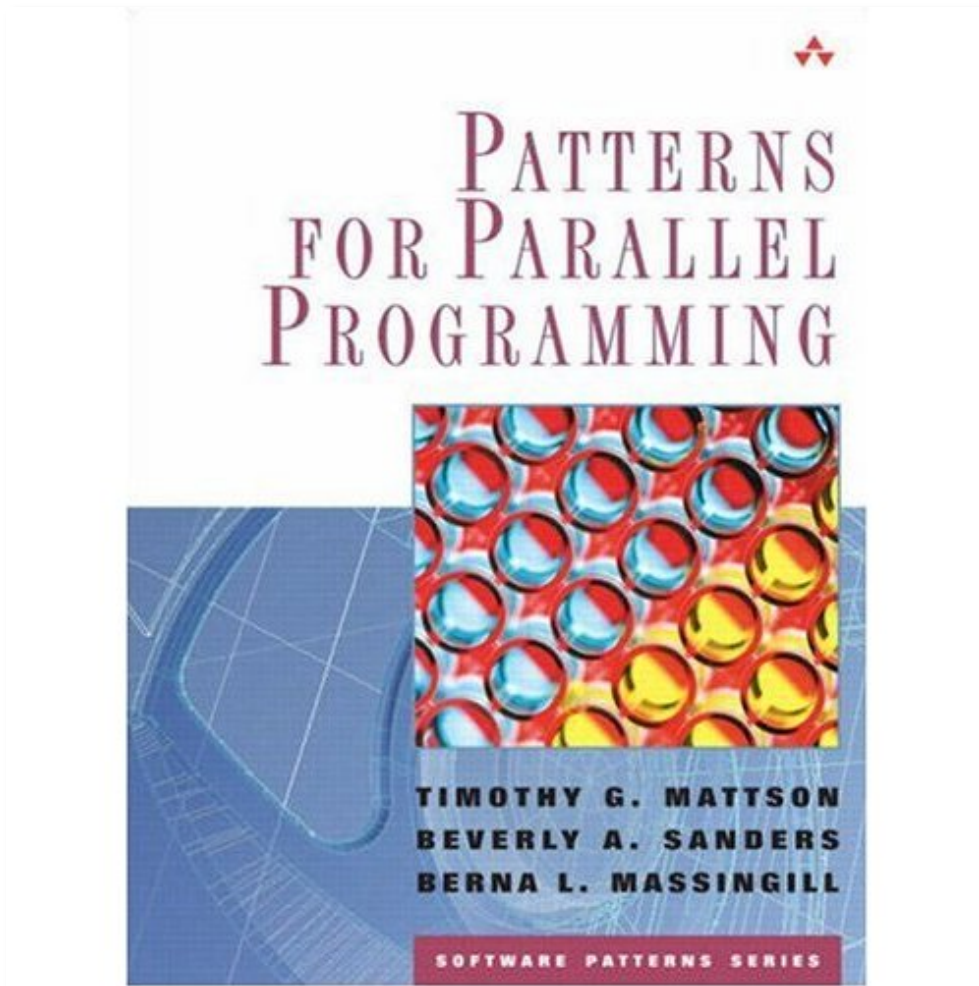
Schematic overview

Parallelization in High Energy Physics

- **Event-level parallelism** mostly used
 - » Compute one event after the other in a single process
 - » **Advantage:** large jobs can be split into N efficient processes, each responsible for process M events
 - Built-in scalability
 - » **Disadvantage:** memory must be made available to each process
 - With 2 – 4 GB per process, with a dual-socket server with Quad-core processors we need 16 –32 GB (or more)
 - Memory is expensive (power and cost!) and the capacity does not scale as the number of cores
 - A lot of recent efforts in this area (see CHEP presentations at Tapei)
- **Algorithm parallelization**
 - » Prototypes using posix-thread, OpenMP, CUDA, and parallel gcclib
 - **Online:** track finding and fitting
 - **Data analysis software**
 - » Effort to provide basic thread-safe/multi-thread library components

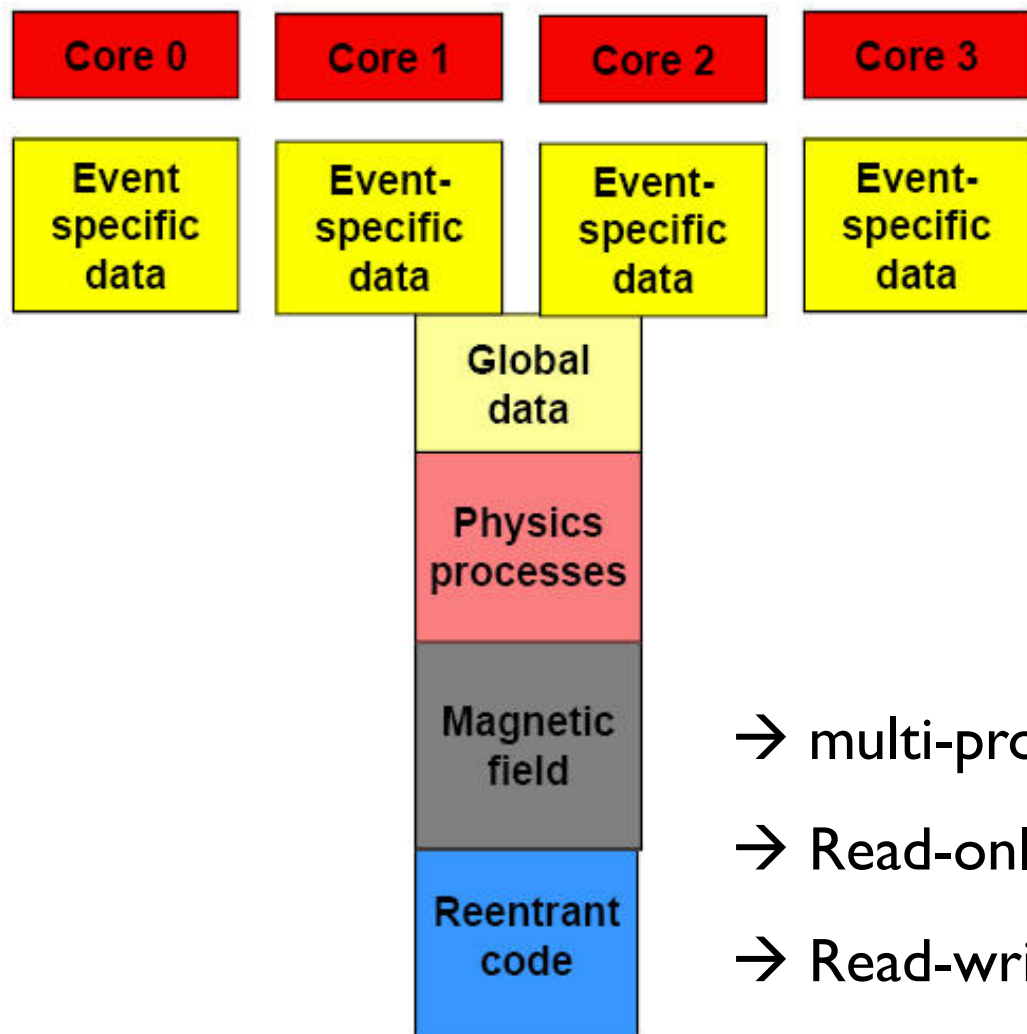
Patterns for Parallel Programming

- ❑ In order to create complex software it is necessary to compose programming patterns
- ❑ Examples:
 - Pipes and filters
 - Layered systems
 - Agents and Repository
 - Event-Based Systems
 - Puppeteer
 - Map/Reduce



Opportunity: Reconstruction Memory-Footprint shows large condition data

How to share common data between different process?



CMS:

1 GB total Memory
Footprint

Event Size 1 MB

Sharable data 250MB

Shared code 130MB

Private Data 400MB !!



- multi-process vs multi-threaded
- Read-only: Copy-on-write, Shared Libraries
- Read-write: Shared Memory, sockets, files

“The experiment offline systems after one year”

LANCASTER UNIVERSITY 10/18/10 RWL Jones CHEP2010 26

Related developments

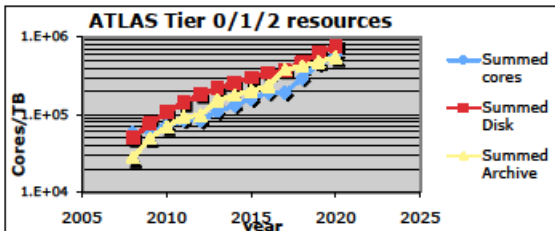
- IO challenges being (partly) addressed by fast merging
- Re-write of Gaudi with stronger memory model planned
- Down the line, we may need to parallize the code
 - This could be either for many-core processors or for Graphical Processing Units – but the development might address both
 - GPUs having big success & cost savings in other fields
 - Harder for us to use, but funders will continue to ask
 - We need the R&D to know which path to take
 - Developments require O(3 years) to implement
 - This includes Geant4 – architectural review this year


LANCASTER UNIVERSITY 10/18/10 RWL Jones CHEP2010 2

Future Challenges

- We assume we can use growth in CPU
 - But this implies changing architectures
 - And handle the data throughput



- Experiments already working to deal with multi cores
 - Many cores and GPGPUs are down the line
- We need to use them or be very clear why we cannot

“How to harness the performance potential of current Multi-Core CPUs and GPUs”

Today:
Seven dimensions of multiplicative performance

First three dimensions:

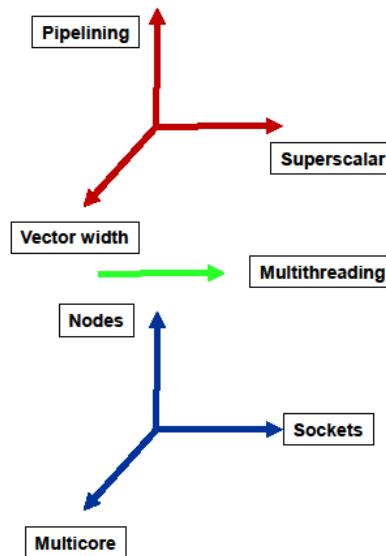
- Pipelined execution units
- Large superscalar design
- Wide vector width (SIMD)

Next dimension is a “pseudo” dimension:

- Hardware multithreading

Last three dimensions:

- Multiple cores
- Multiple sockets
- Multiple compute nodes



SIMD = Single Instruction Multiple Data

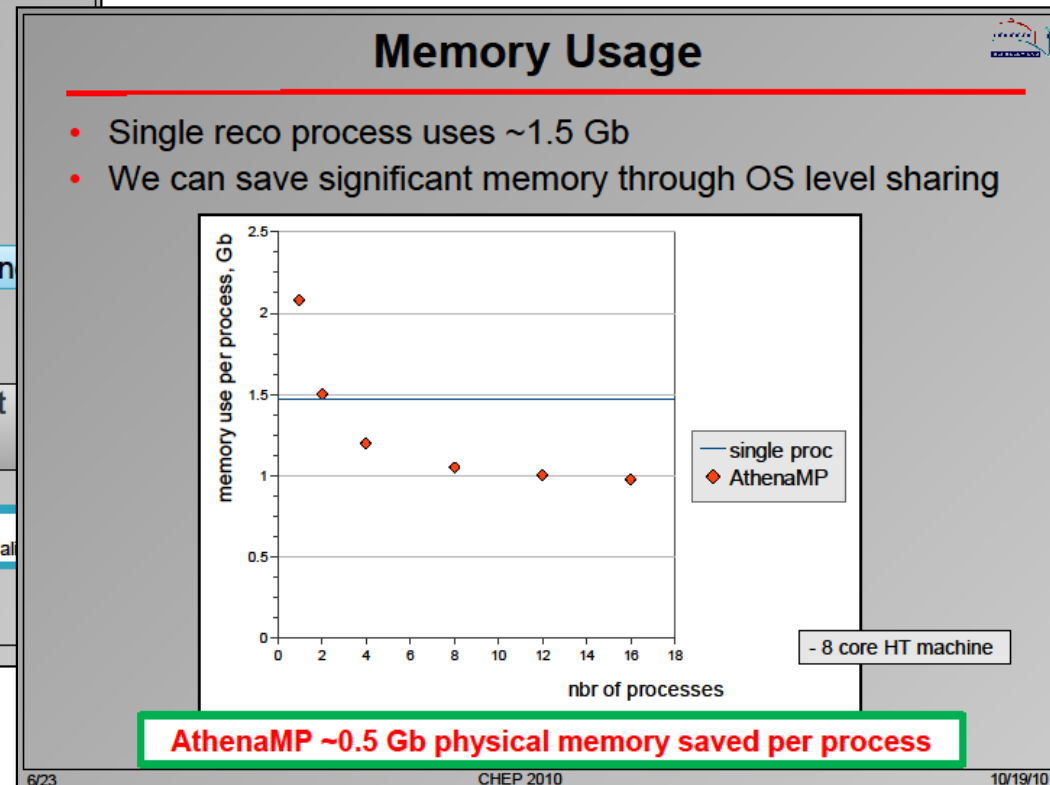
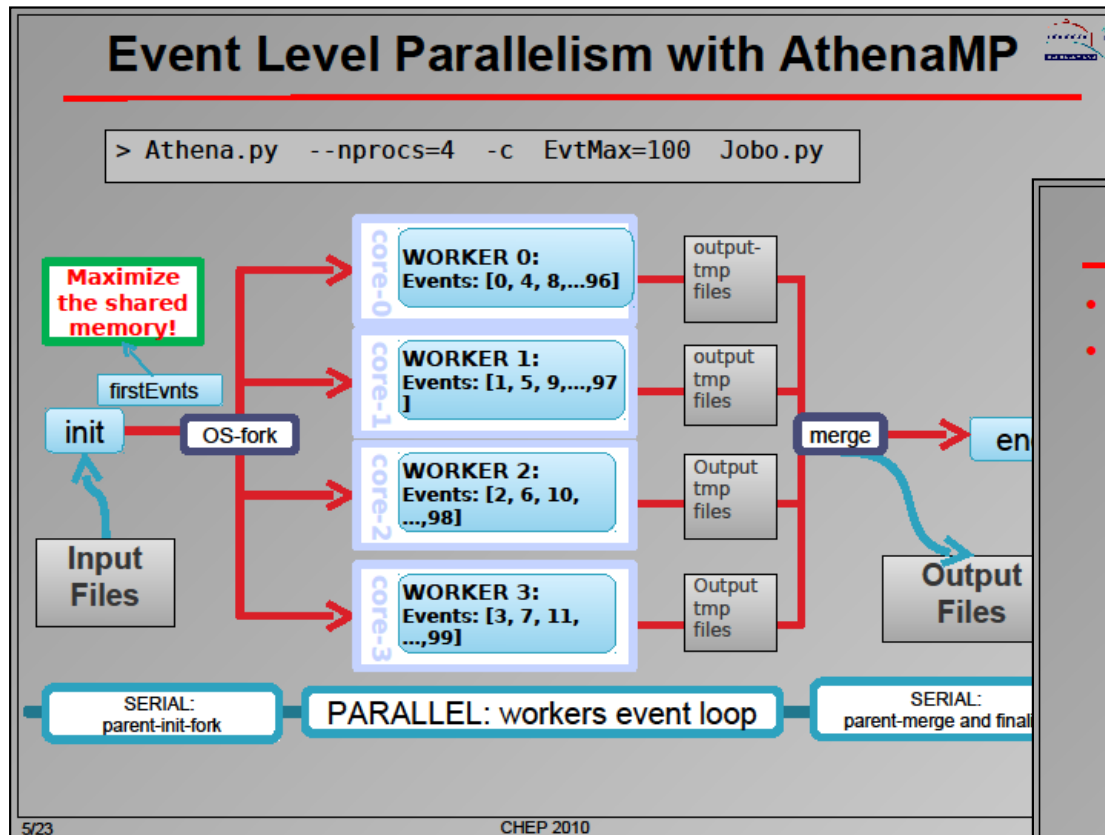
Sverre Jarp - CERN

What are the multi-core options?

There is a discussion in the community about the best way(s) forward:

- 1) Stay with event-level parallelism (and entirely independent processes)
 - Assume that the necessary memory remains affordable
 - Or rely on tools, such as KSM, to help share pages
- 2) Rely on forking:
 - Start the first process; Run through the first “event”
 - Fork N other processes
 - Rely on the OS to do “copy on write”, in case pages are modified
- 3) Move to a fully multi-threaded paradigm
 - Still using coarse-grained (event-level) parallelism
 - But, watch out for increased complexity

“Parallelizing Atlas reconstruction and simulation on multi-core platforms”



“Multi-core aware Applications in CMS”

Why Bother?

HEP processing is naturally parallelizable

We have billions of events

Each event can be processed independently

Memory is becoming a limitation

Historically GB/US\$ increases at the same rate as number of transistors in a CPU

<http://www.jcmit.com/memoryprice.htm>

Funding levels are not guaranteed to stay this high

We can afford 2GB/core now but may not in the future

Opportunistic use of grid sites improves if we lower our memory requirements

Not all grid sites have 2GB/core

Technical limitations on connecting many cores to shared system memory

<http://www.intel.com/technology/itj/2007/v11i3/3-bandwidth/7-conclusion.htm>

Multi-core aware applications can improve memory sharing

Threading

All threads share the same address space but have to worry about concurrent usage

Forking

Each child process gets its own address space

Untouched memory setup by the parent is shared between the child processes

Multi-core Aware Applications in CMS

3

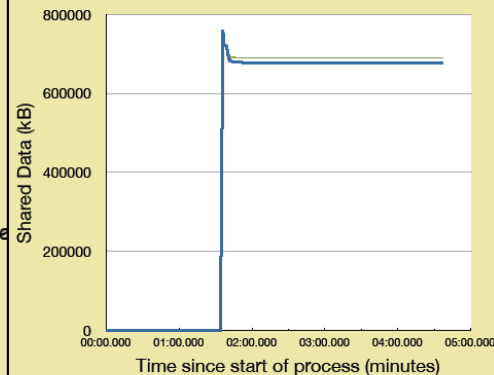
CHEP



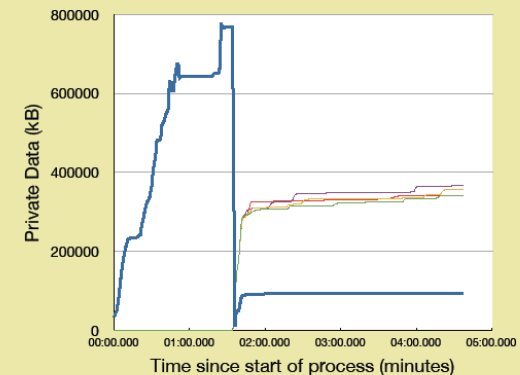
Memory Sharing



Shared Data vs Time



Private Data vs Time



Measurements done using reconstruction with 64bit software on 4 CPU, 8 core/CPU 2GHz AMD Opteron(tm) Processor 6128

Shared memory per child: ~700MB

Private memory per child: ~375MB

Total memory used by 32 children: 13GB

Total memory used by 32 separate jobs: 34 GB

Saved 62% of memory

Multi-core Aware Applications in CMS

11

CHEP 2010

“Maximum likelihood fits using GPUs”

Test environment

- PCs
 - CPU: Nehalem @ 3.2GHz: 4 cores – 8 hw-threads
 - OS: SLC5 64bit - GCC 4.3.4
 - ROOT trunk (October 11th, 2010)

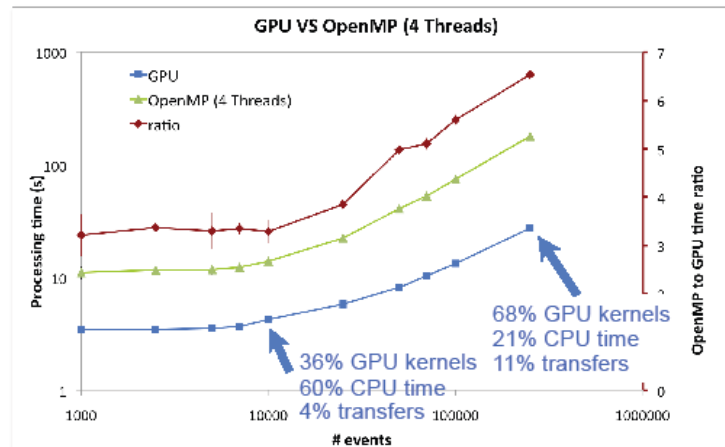
- GPU: ASUS nVidia GTX470 PCI-e 2.0
 - Commodity card (for gamers)
 - Architecture: GF100 (Fermi)
 - Memory: 1280MB DDR5
 - Core/Memory Clock: 607MHz/837MHz
 - Maximum # of Threads per Block: 1024
 - Number of SMs: 14
 - CUDA Toolkit 3.1 06/2010
 - Developer Driver 256.40
 - Power Consumption 200W
 - Price ~\$340



Alfio Lazzaro (alfio.lazzaro@cern.ch)

PDF-event-base: GPU VS OpenMP

- Fair comparison
 - Same algorithm
 - Algorithm on CPU optimized and parallelized (4 threads)
 - CPU does the final sum of the *NLL* and normalization integral calculations
- Check that the results are compatible: asymmetry less than 10^{-12}



- Speed-up increases with the dimension of the sample, taking benefit from the data streaming on GPU and the integral calculation only on the CPU
- ~3x for small samples, up to ~7x for large samples

Alfio Lazzaro (alfio.lazzaro@cern.ch)