Parallel Systems
Impact of architectures on Performance

Arnaud Legrand, CR CNRS, LIG/INRIA/Mescal
Jean-Louis Roch, MCF ENSIMAG, LIG/INRIA/Moais

Vincent Danjean, MCF UJF, LIG/INRIA/Moais
Derick Kondo, CR INRIA, LIG/INRIA/Mescal
Jean-François Méhaut, PR UJF, LIG/INRIA/Mescal
Bruno Raffin, CR INRIA, LIG/INRIA/Moais
Alexandre Termier, MCF UJF, LIG/Hadas

What is Parallel Computing?

- **Parallel computing:** using multiple processors/cores in parallel to solve problems more quickly than with a single processor/core
- Examples of parallel machines:
  - A **Chip Multi-Processor** (CMP) contains multiple processors (called cores) on a single chip
  - A **shared memory multiprocessor** (SMP*) by connecting multiple processors to a single memory system
  - A **cluster computer** that contains multiple PCs combined together with a high speed network
  - A **grid** is a cluster of networked, loosely-coupled computers acting to perform very large tasks
- Concurrent execution comes from desire for performance; unlike the inherent concurrency in a multi-user distributed system
- * Technically, SMP stands for “Symmetric Multi-Processor”

Motivations of this first course...

- Details of machine are important for performance
  - Processor, memory system, communication (not just parallelism)
  - Before you parallelize, make sure you're getting good serial performance
  - What to expect? Use understanding of hardware limits
- There is parallelism hidden within processors
  - Pipelining, SIMD, etc
- Locality is at least as important as computation
  - Temporal: re-use of data recently used
  - Spatial: using data nearby that recently used
- Machines have memory hierarchies
  - 100s of cycles to read from DRAM (main memory)
  - Caches are fast (small) memory that optimize average case
- Can rearrange code/data to improve locality

Why Parallel Computing Now?

- Researchers have been using parallel computing for decades:
  - Mostly used in computational science and engineering
  - Problems too large to solve on one computer; use 100s or 1000s
- Many companies in the 80s/90s “bet” on parallel computing and failed
  - Computers got faster too quickly for there to be a large market

Why Parallelism (2008)?

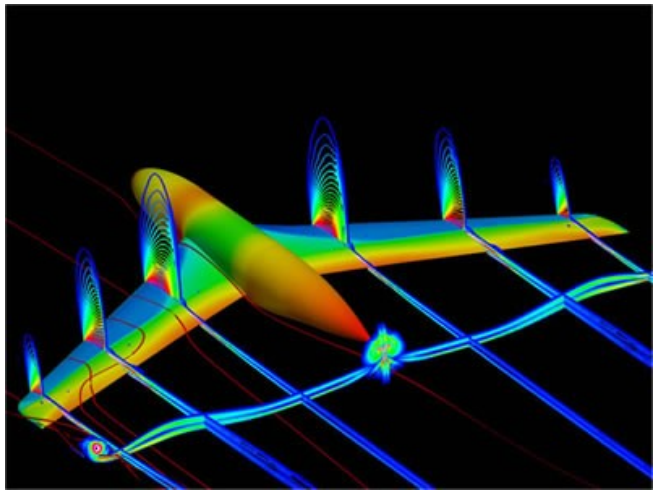
- These arguments are no long theoretical
- All major processor vendors are producing multicore chips
  - Every machine will soon be a parallel machine
  - All programmers will be parallel programmers???
- New software and programming model
  - Want a new feature? Hide the “cost” by speeding up the code first
  - All programmers will be performance programmers???
- Some may eventually be hidden in libraries, compilers, and high level languages
  - But a lot of work is needed to get there
- Big open questions:
  - What will be the killer apps for parallel machines?
  - How should the chips be designed, and how will they be programmed?

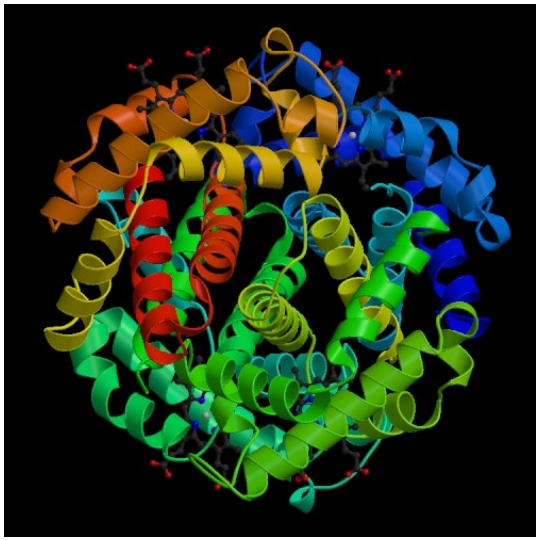
Outline

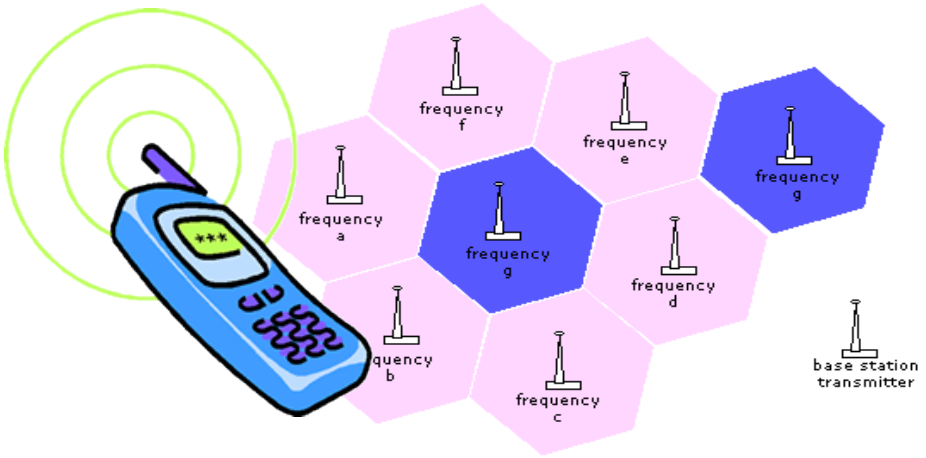
- HPC applications
- Why all computers (including your laptop or MPSoC) must be parallel?
- Why writing (fast) parallel programs is hard?
- Principles of parallel computing performance

HPC Applications

Intensive Computation Applications



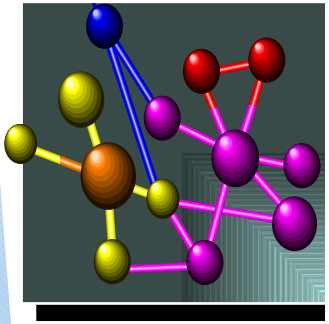


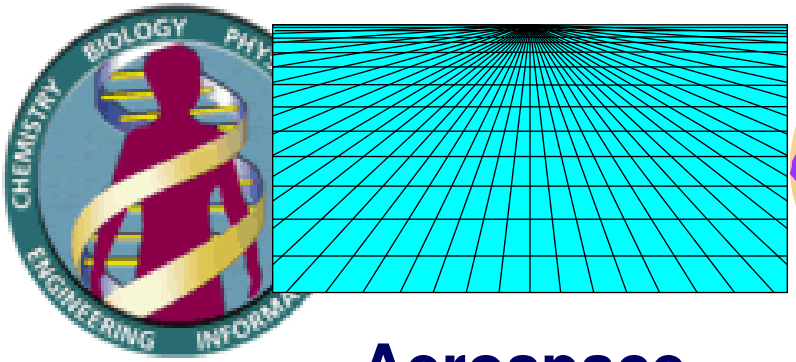


Computing Power Drivers

"Grand Challenge" Applications using computing power and also memory

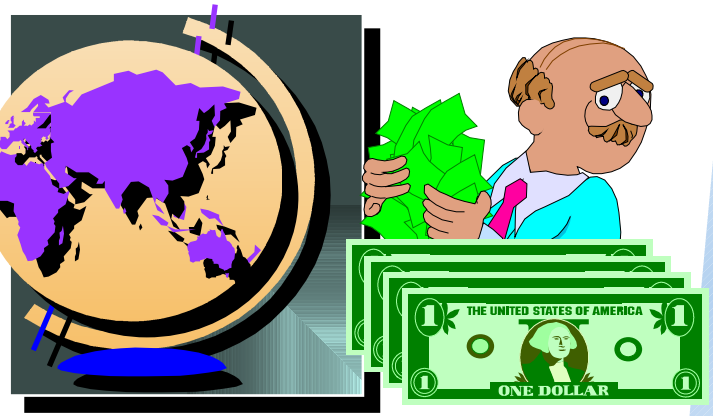
Models, *simulations* and *analysis*



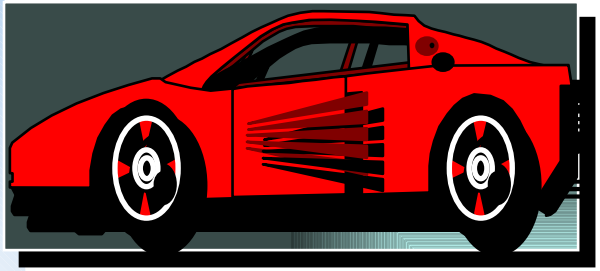


Life Sciences

Aerospace



E-commerce



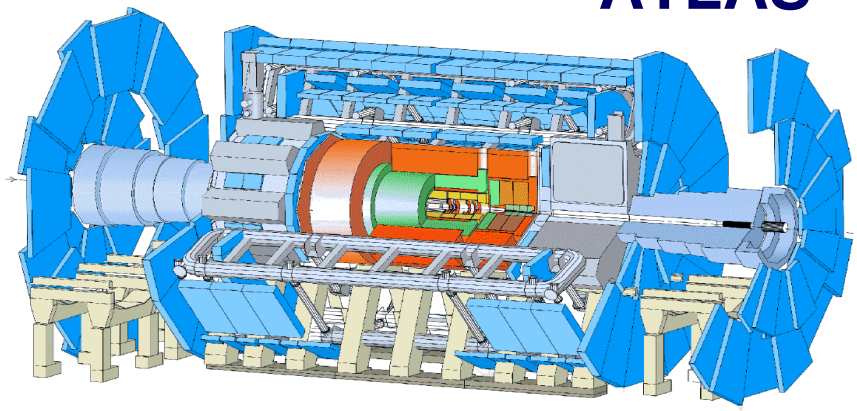
CAD/CAM

Digital Biology

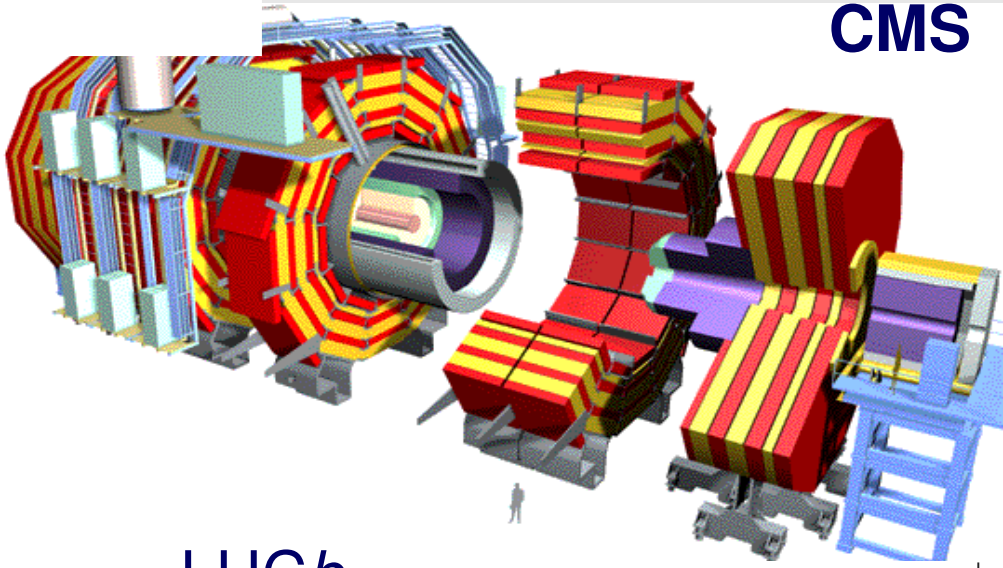
Military Applications

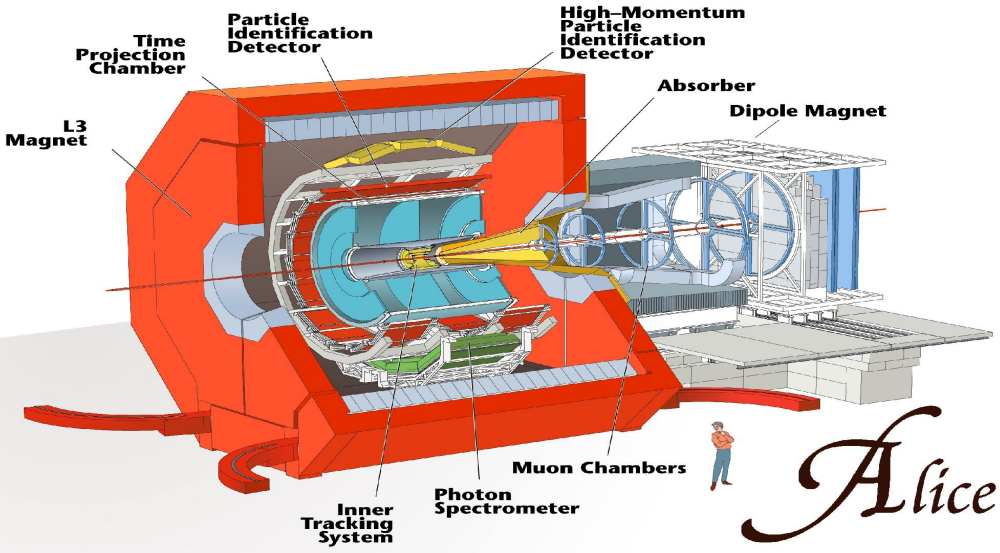
The Large Hadron Collider Project
4 detectors

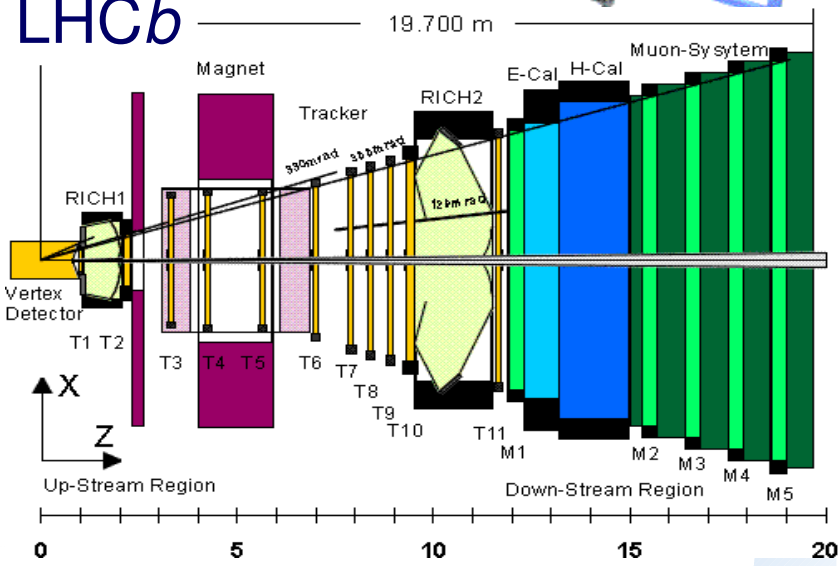
ATLAS



CMS





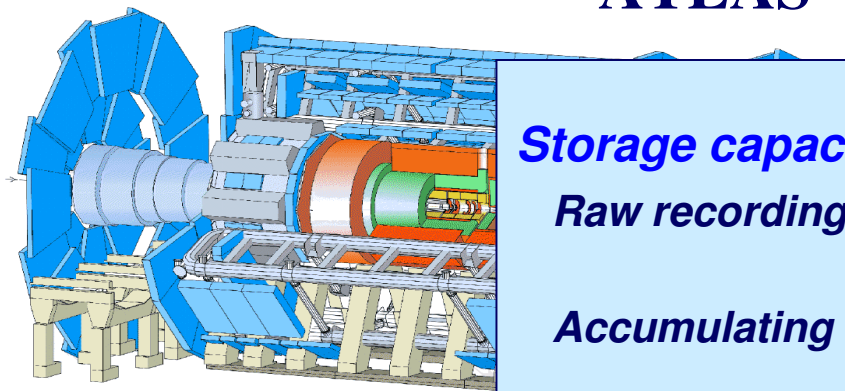


CERN

The Large Hadron Collider Project
4 detectors

ATLAS

CMS



Storage capacity–

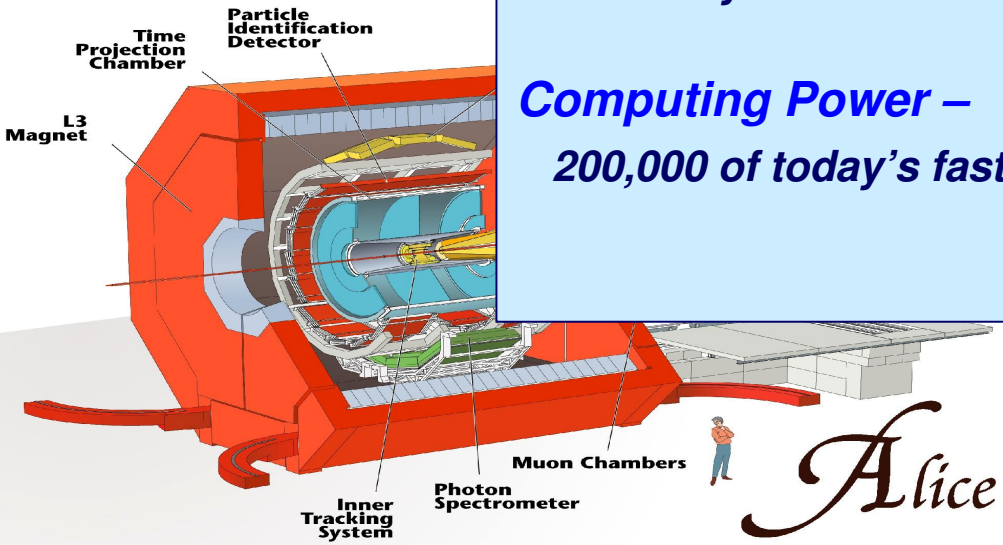
Raw recording rate 0.1 – 1 GBytes/sec

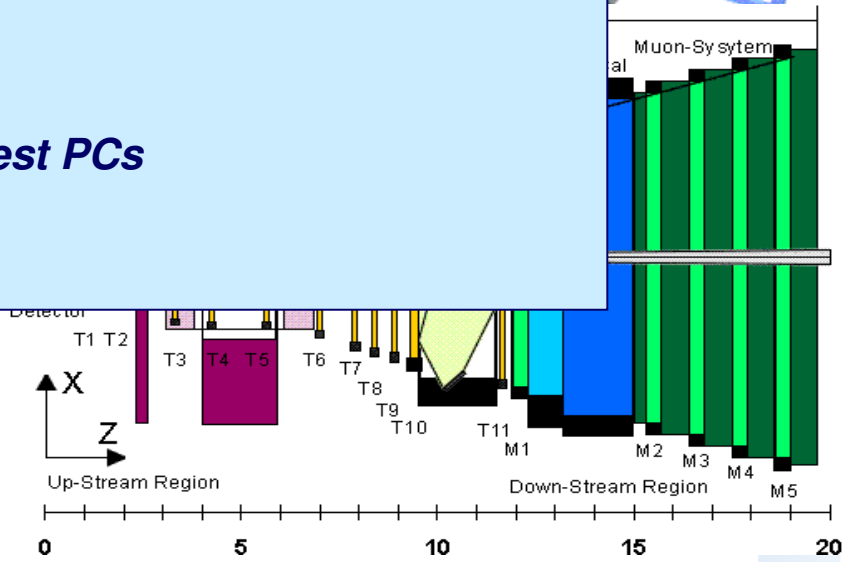
Accumulating at 5-8 PetaBytes/year

10 PetaBytes of disk

Computing Power –

200,000 of today's fastest PCs





CERN

Earthquake Hazard Assessment

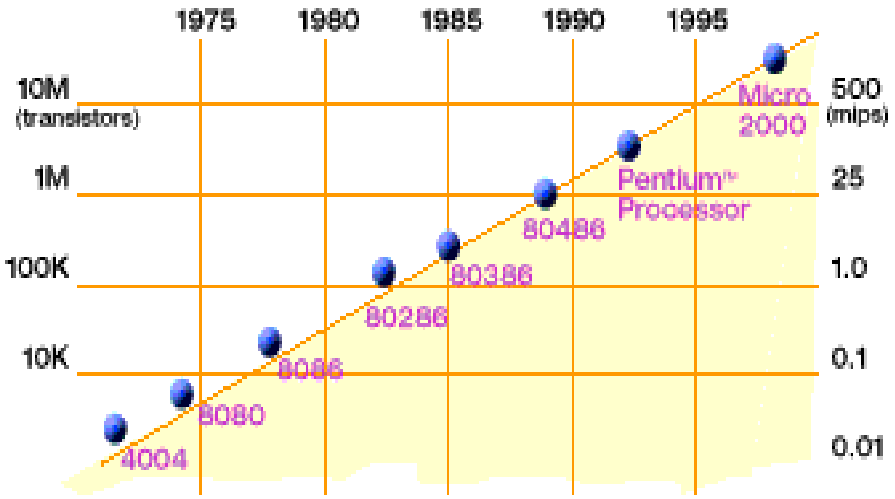
2001 Gujarati (M 7.7) Earthquake, India

- 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

Why all computers must be parallel?

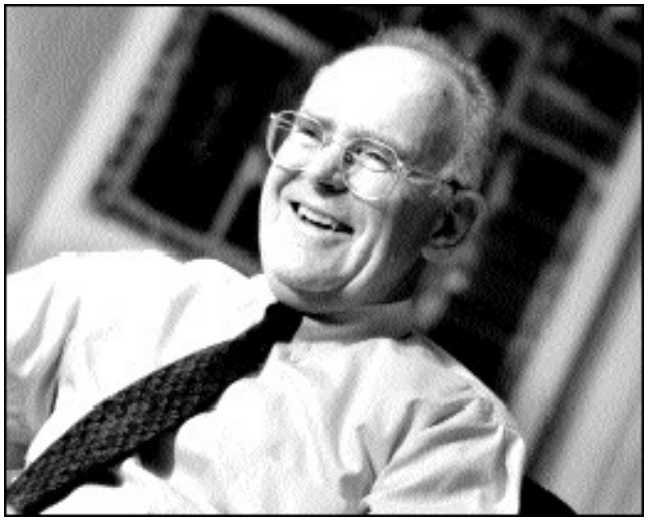
Technology Trends: Microprocessor Capacity



2X transistors/Chip Every 1.5 years

Called "Moore's Law"

Microprocessors have become smaller, denser, and more powerful.



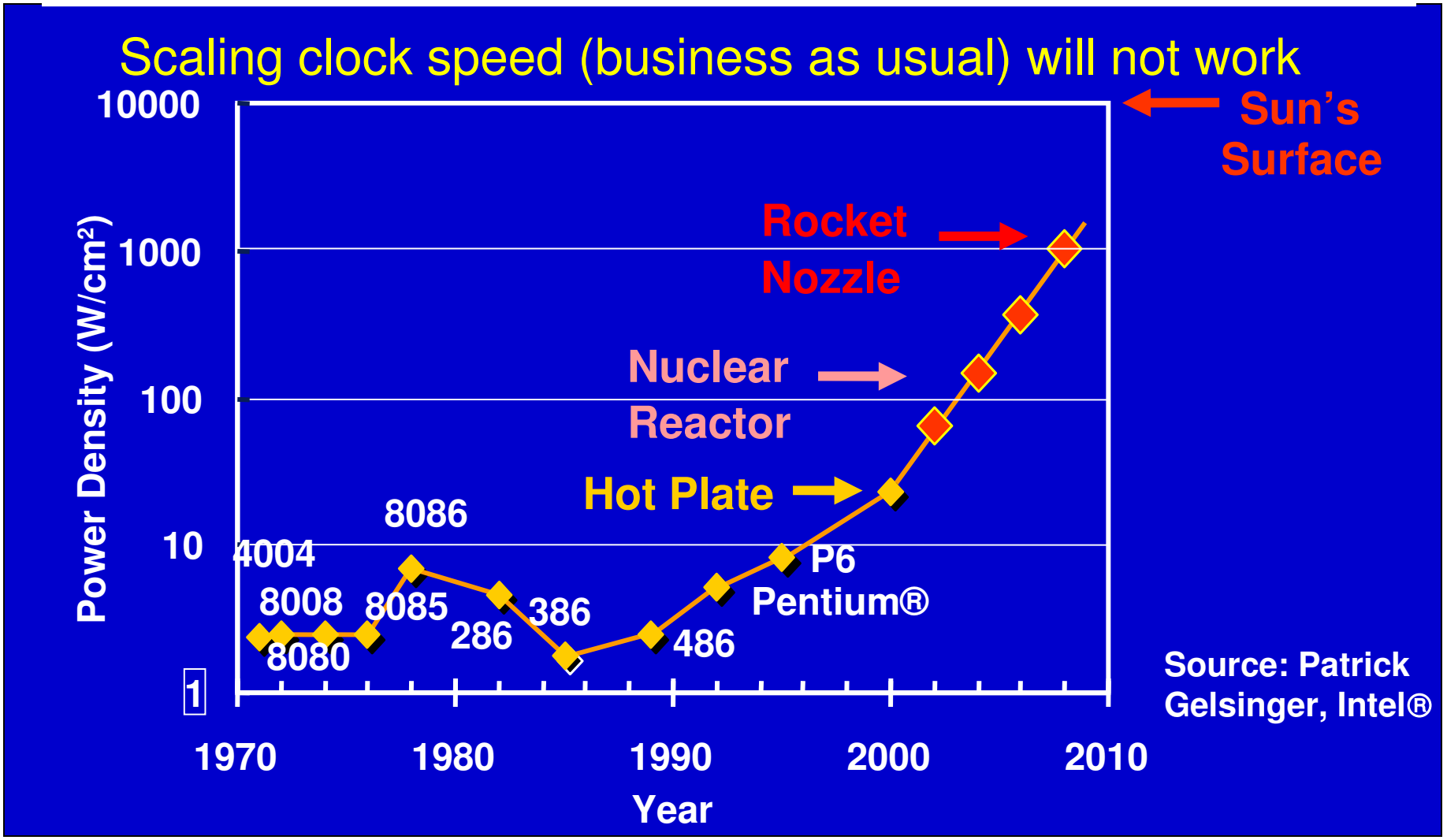
Gordon Moore (co-founder of Intel) predicted in 1965 that the transistor density of semiconductor chips would double roughly every 18 months.

Slide source: Jack Dongarra

Limit #1: Power density

Can soon put more transistors on a chip than can afford to turn on.

-- Patterson '07



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

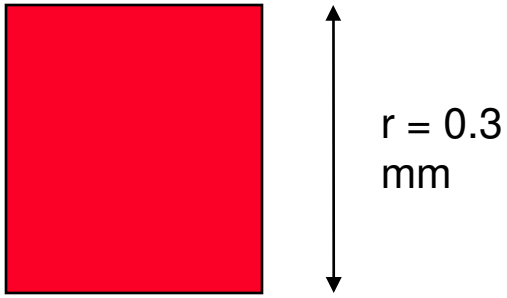
Limit #2: Hidden Parallelism Tapped Out

- 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

Limit #3: Speed of Light (Fundamental)

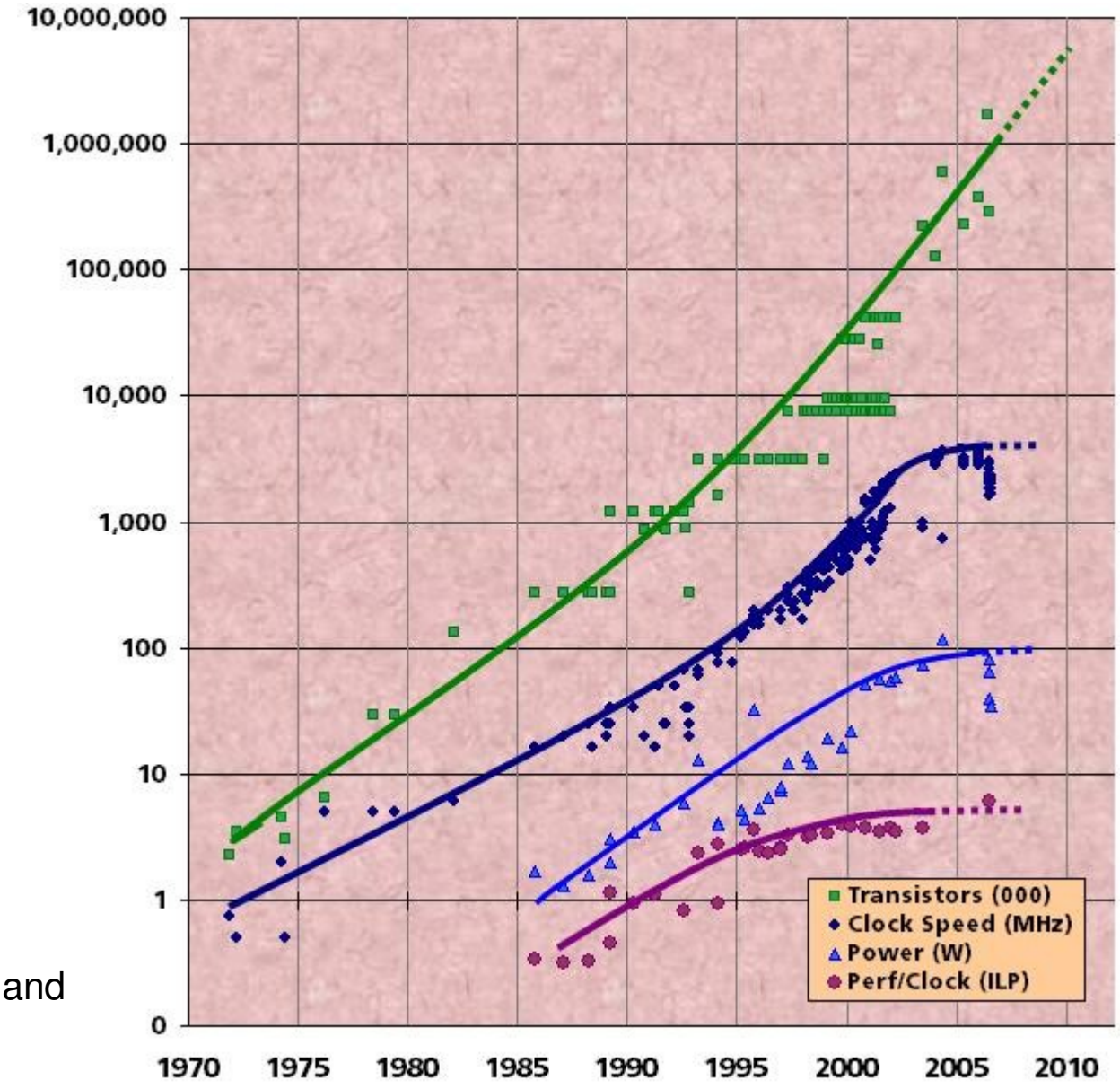
1 Tflop/s, 1 Tbyte sequential machine



- 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 = $3x10^8$ 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

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)

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,

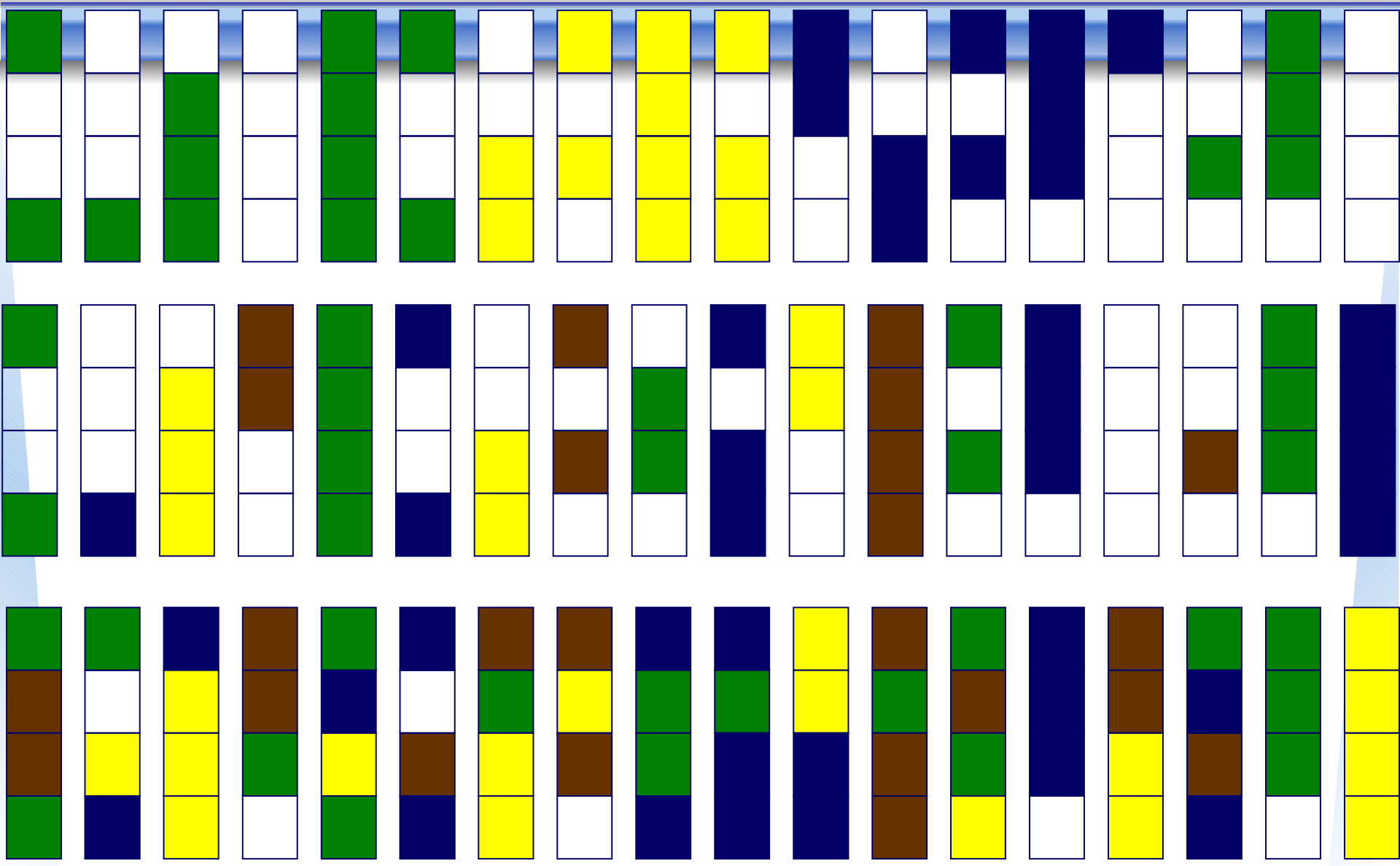
- The STI Cell processor (PS3) has 8 cores
- The latest NVidia Graphics Processing Unit (GPU) has 128 cores
- Intel has demonstrated the TeraScale processor (80-core), research chip

Tunnel Vision by Experts

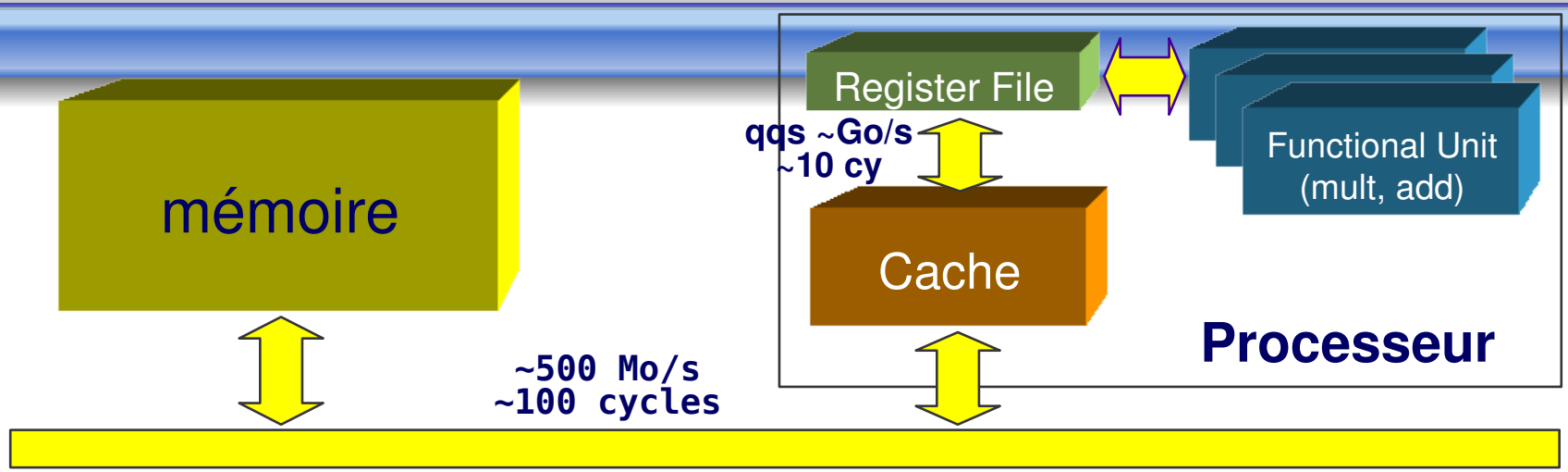
- “On several recent occasions, I have been asked whether parallel computing will soon be relegated to the trash heap reserved for promising technologies that never quite make it.”
  - Ken Kennedy, CRPC Directory, 1994
- “640K [of memory] ought to be enough for anybody.”
  - Bill Gates, chairman of Microsoft,1981.
- “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.

Hyperthreading, SMT, NUMA

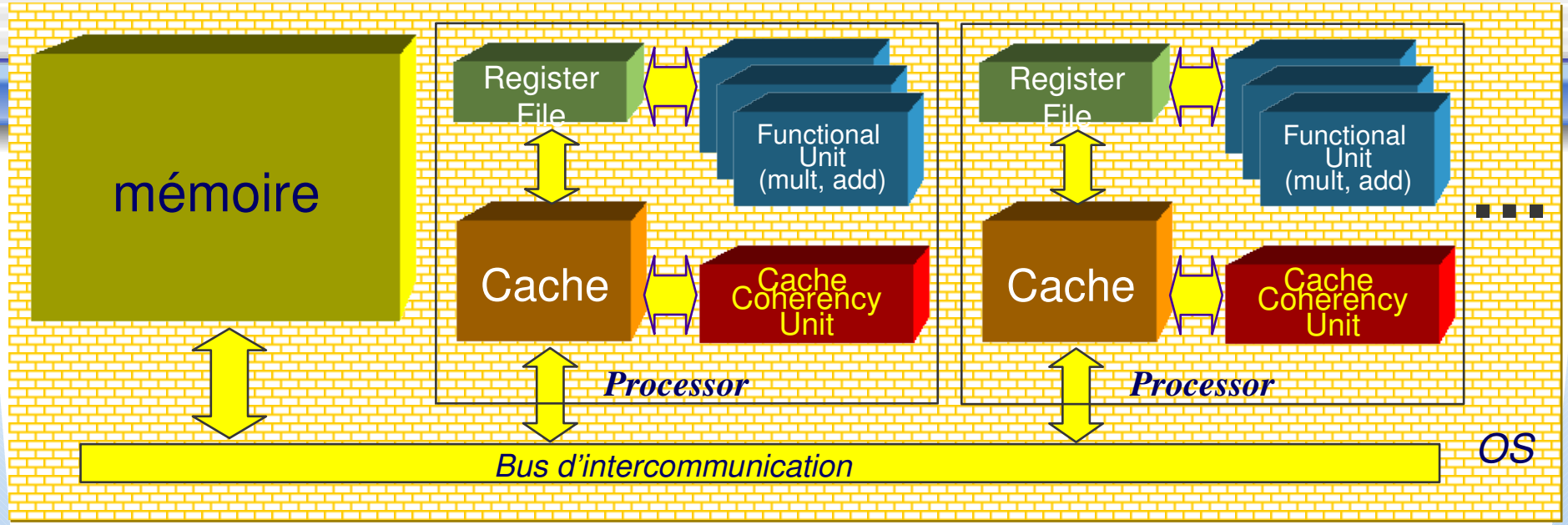


Architecture Scalaire



- Reduced Instruction Set (RISC) Architecture:
  - Les instructions load/store font référence à la mémoire
  - Les unités fonctionnelles travaillent sur des données stockées dans les registres
  - Hiérarchie mémoire dans une architecture scalaire :
    - Les éléments utilisés récemment sont copiés dans le cache,
    - Les accès au cache sont plus rapides que les accès à la mémoire.

Architecture SMP UMA



- **Modèle de programmation**
  - **Extension du modèle de programmation monoprocesseur**
- **Bus d'interconnexion entre la mémoire et les processeurs**
- **Faible nombre de processeurs**

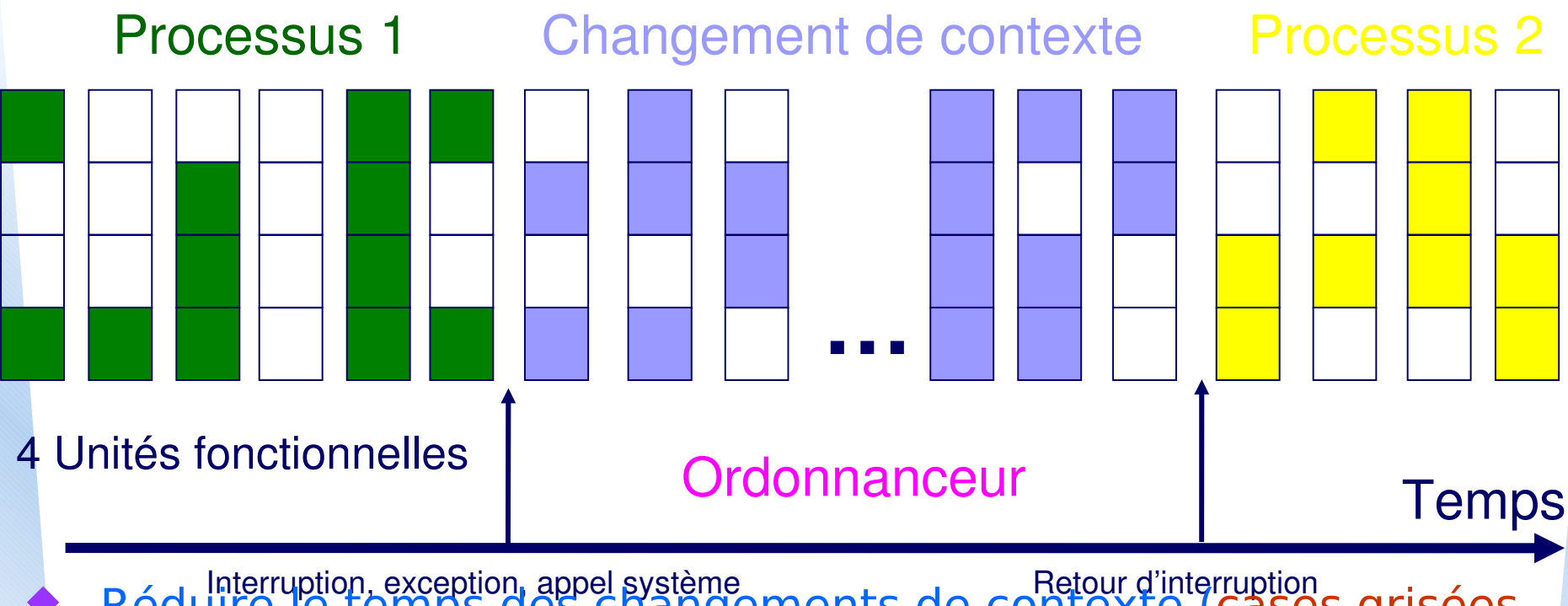
Programmation Mono-Threadée

- Exécution séquentielle d'un programme
  - Instruction par instruction
  - Instructions: Calcul, Mémoire, Branchement, appel de procédure,…
- Processus, processus Lourd
  - Structuration des systèmes d'exploitation
  - Multi-programmation, temps-partagé
- Caractéristiques
  - Entité active directement supportée par l'OS
    - Flot d'exécution
    - Pile des contextes de procédure
    - Espace d'adressage privé
    - Ressources systèmes
- Coût de gestion élevé
  - Allocation des ressources (mémoire,…)
  - Appels systèmes (Fork, exec, …)

Instruction Level Parallelism

- Programme séquentiel
  - N'y aurait-il pas des instructions indépendante qui pourraient être exécutées en parallèle?
- Comment générer de l'ILP?
  - Pipe-line du processeur
    - Recouvrement d'exécution d'instructions
    - Limité par la divisibilité de l'instruction
  - Superscalaire
    - Plusieurs unités fonctionnelles
    - Limité par le parallélisme intrinsèque du programme seq.
- Comment accroître l'ILP?
  - Prédictions sur les branchements conditionnels
  - Réordonnancer les instructions (Out of Order Execution)
- Recherche
  - Domaines de l'Architecture-compilation

Processeur SuperScalaire



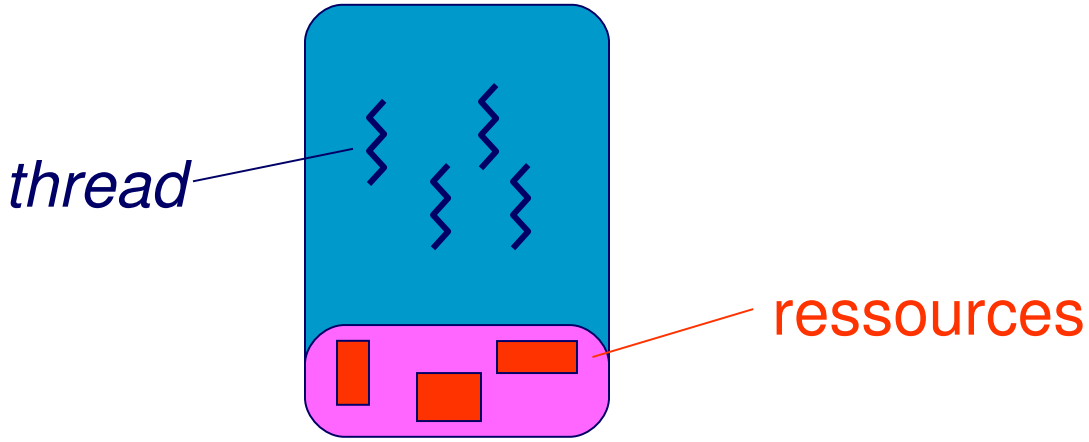
- Réduire le temps des changements de contexte (cases grisées-bleues)
- Accroître l'utilisation des unités d'exécution (cases blanches)

Processus Légers/Threads

- Objectifs
  - Mener plusieurs activités indépendantes au sein d'un processus
  - Exploitation des architectures SMP
  - Améliorer l'utilisation du processeurs (context-switch)
- Exemples
  - Simulations
  - Serveurs de fichiers
  - Systèmes d'exploitation (!)
- Solution sans l'aide du multithreading
  - Automate à états finis implanté « à la main » (sauvegardes d'états)

Les processus légers

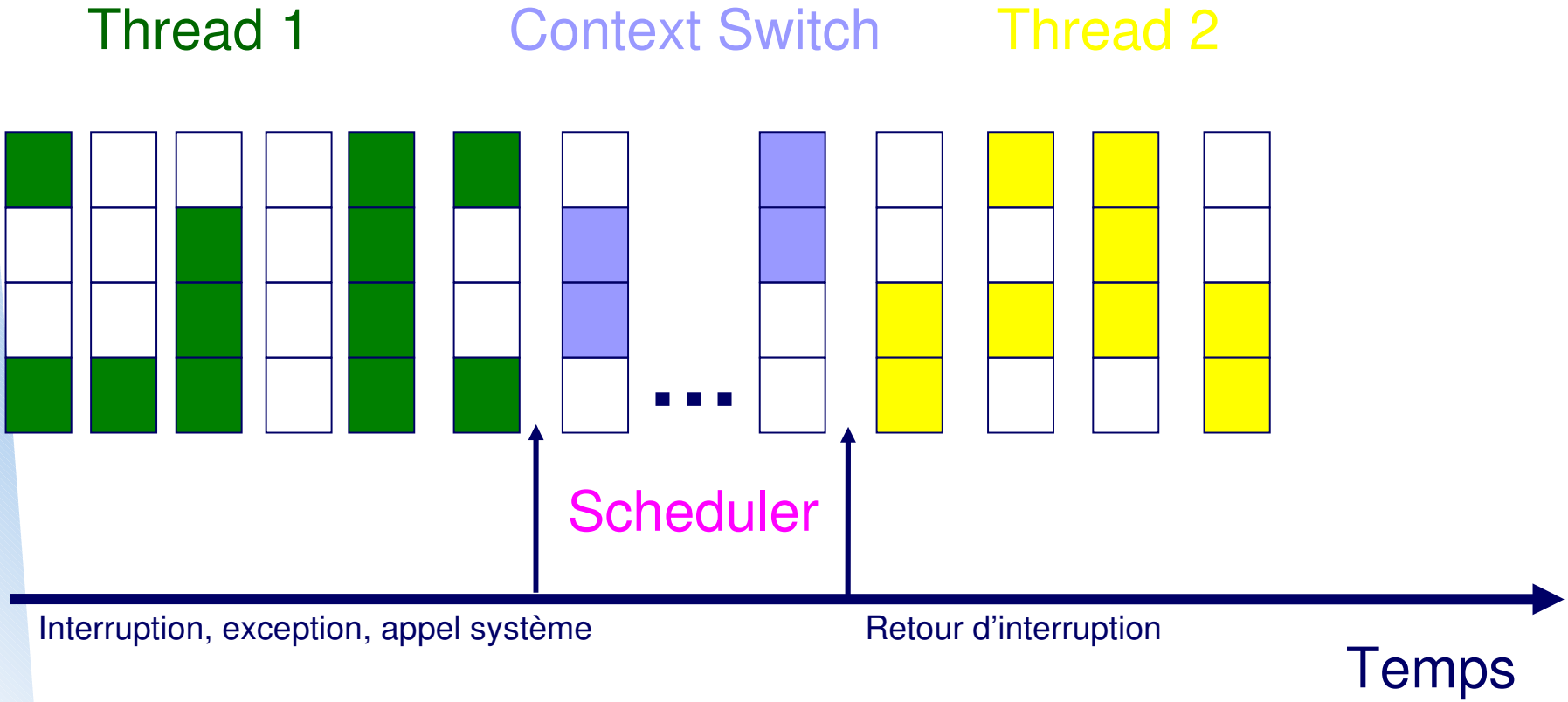
- Principe
  - Détacher flot d'exécution et ressources



- Introduits dans divers langages & systèmes
  - Programmation concurrente
  - Recouvrement des E/S
  - Exploitation des architectures SMP

Multithreading et Processeur SS

Processeur SuperScalaire



Hyperthreading

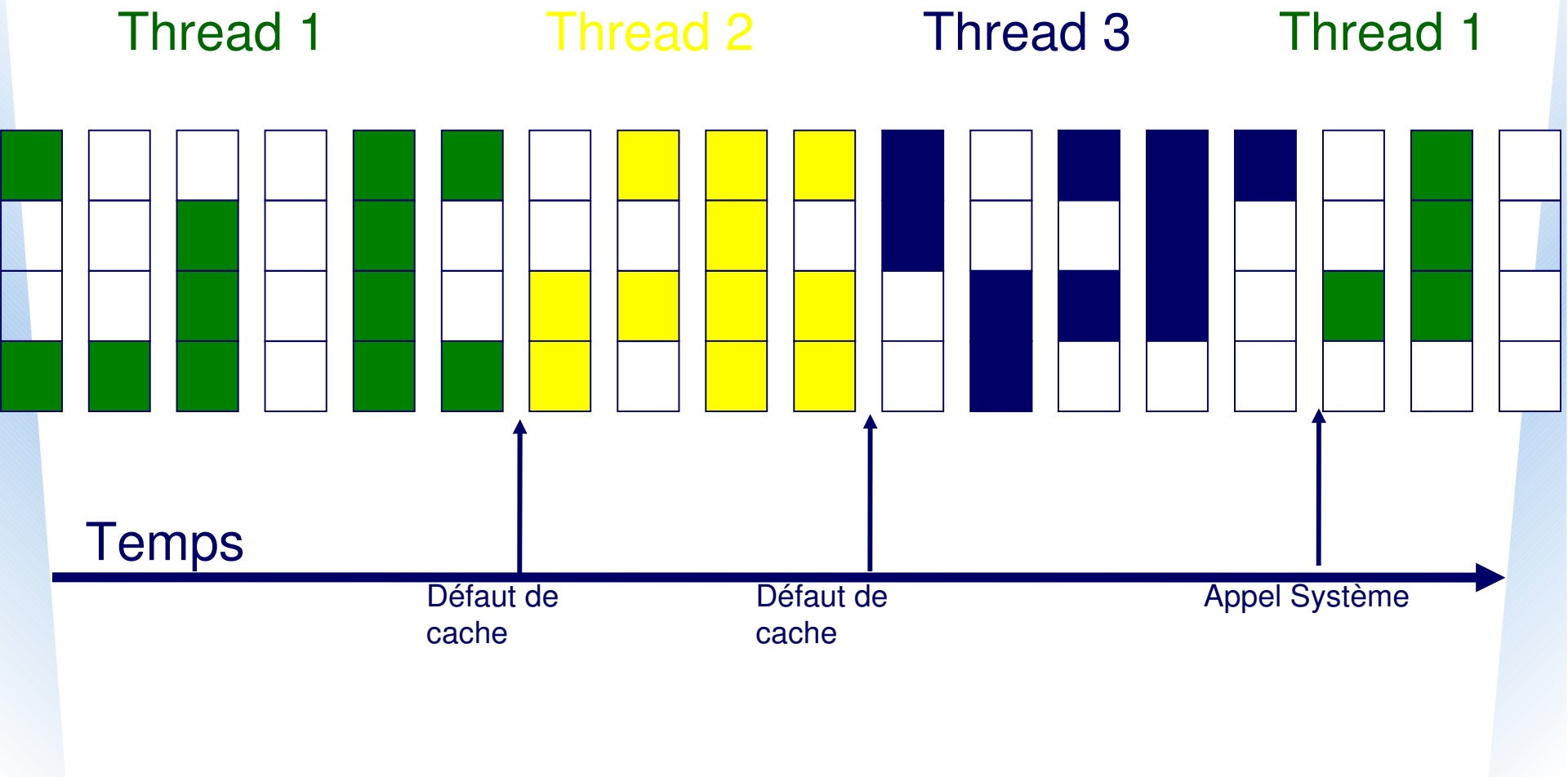
Evolution des architectures de processeurs

Phase 1: Processeurs Multithreadés

- Modification de l'architecture du processeur
  - Incorporer au processeur deux (ou plus) jeux de registres pour les contextes des threads
    - Registres généraux
    - Program Counter (PC), registre d'instruction
    - Process Status Word (PSW), registre d'état
  - A tout instant, un thread et son contexte sont actifs
  - Changement du contexte courant instantané
    - Appel système, IT
    - Défaut de cache
- Processeur IBM PowerPC RS 64
  - Recherche, non commercialisé
- Processeur Intel Xeon Hyperthreading
  - Serveur Bi-processeur Xeon Hyperthreadé
    - Vue de Linux ou Windows, 4 processeurs

Phase 1: Processeurs Multithreadés (2)

Multithread à Gros Grain (Coarse-Grained Multi-threaded)



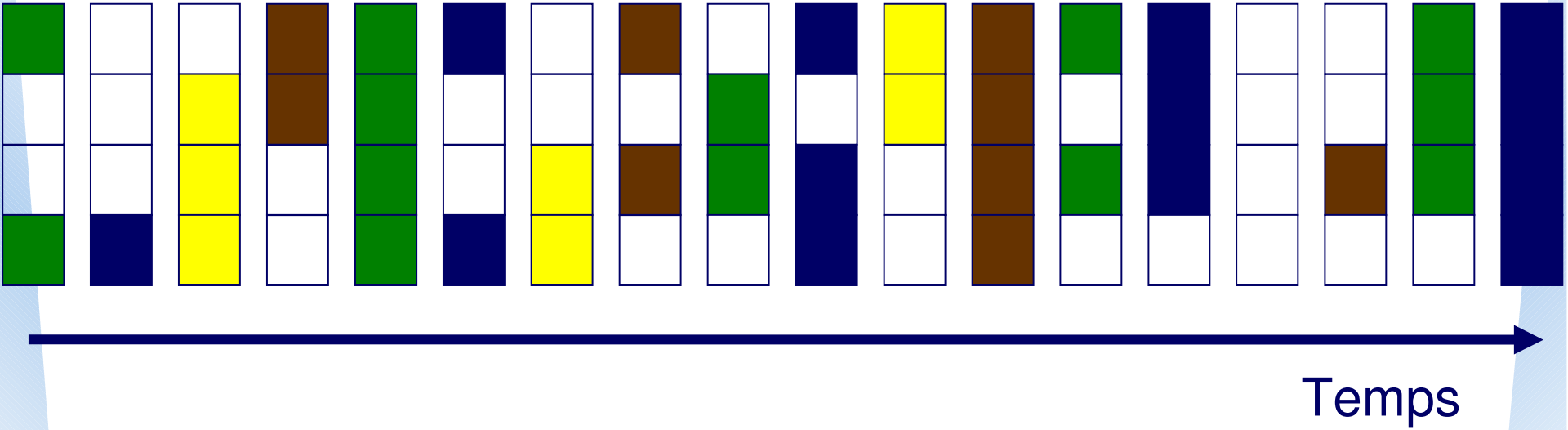
Phase 2: Processeurs Multithreads (1)

- Modification de l'architecture du processeur
  - Incorporer au processeur N jeux de registres pour les contextes des threads
    - Registres généraux
    - Program Counter (PC), registre d'instruction
    - Process Status Word (PSW), registre d'état
  - A tout instant, un thread et son contexte sont actifs
  - Changement de contexte à chaque cycle
    - Chaque thread dispose de son ratio du processeur (1/N)
- Processeur TERA

Phase 2: Processeurs Multithreadés (2)

Multithread à Grain Fin (Fine-Grained Multi-threaded)

4 registres de threads : ¼ temps processeur par thread



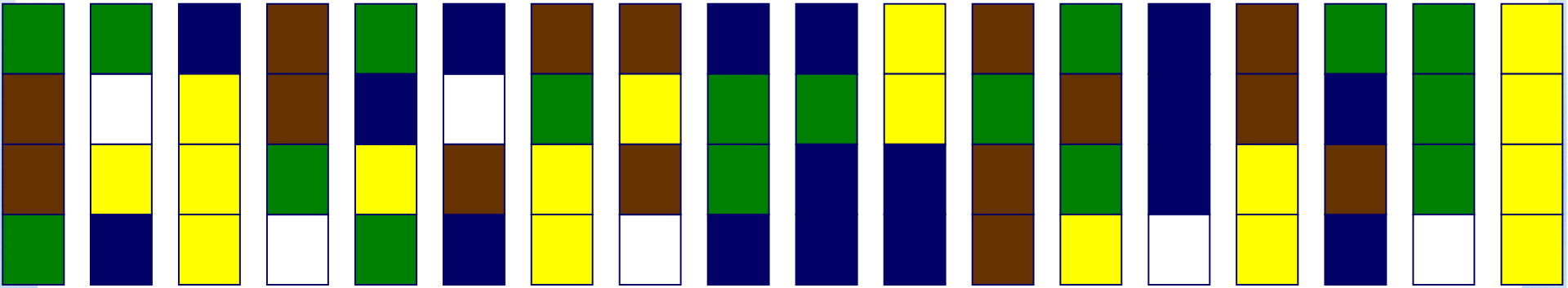
Temps

Thread 1 Thread 2 Thread 3 Thread 4

Processeurs HyperThreadés (1)

- Modification de l'architecture du processeur
  - Incorporer au processeur N jeux de registres pour les contextes des threads
    - Registres généraux
    - Program Counter (PC), registre d'instruction
    - Process Status Word (PSW), registre d'état
  - A un instant donné, les unités du processeurs peuvent être partagées entre plusieurs threads
- Eviter de stresser les mêmes ressources
  - Conflit d'accès à certaines unités d'exécution
- Processeur ALPHA EV8

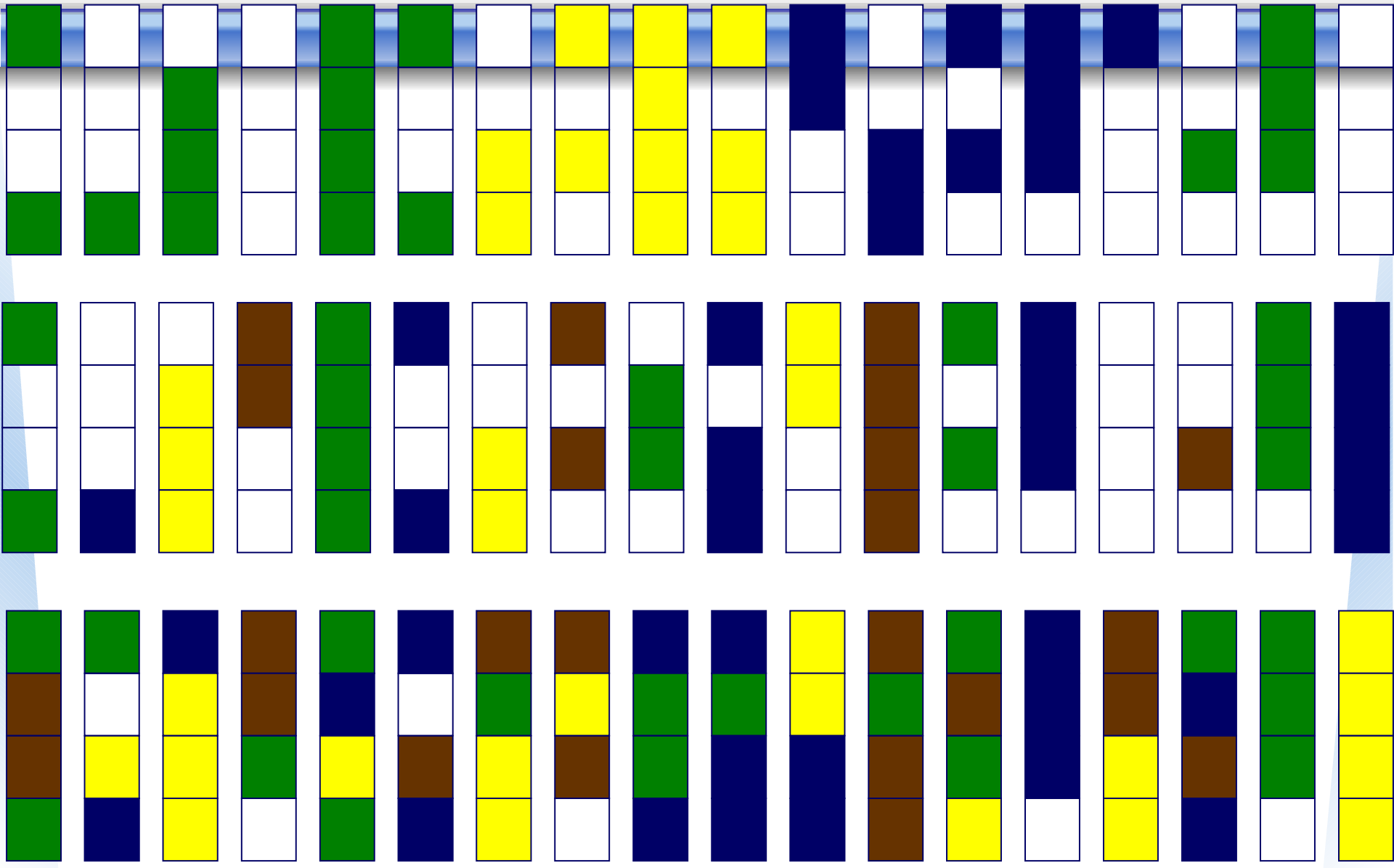
Processeurs Hyper-Threadés (2)



Thread 1 Thread 2 Thread 3 Thread 4

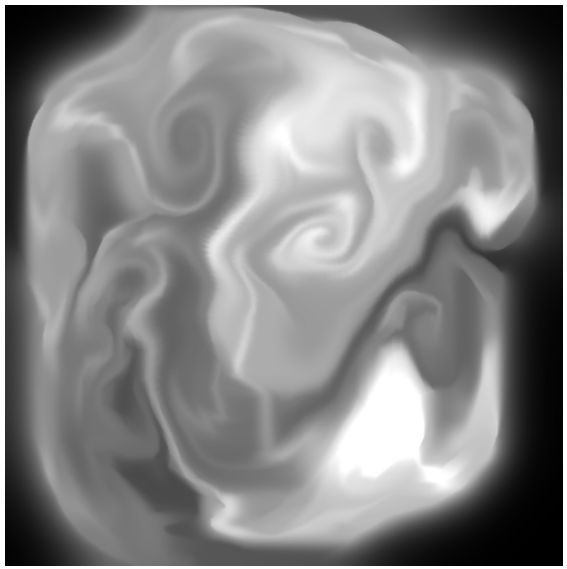
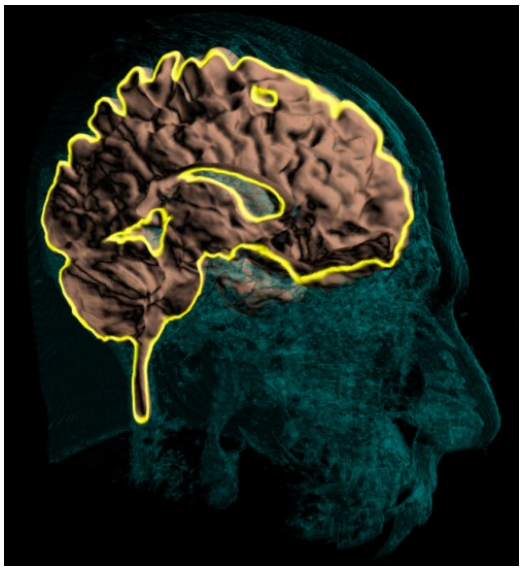
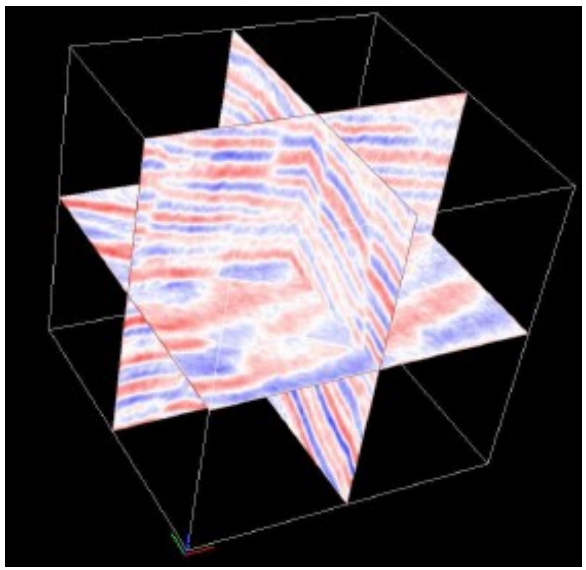
Temps

Résumé



GPGPU

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

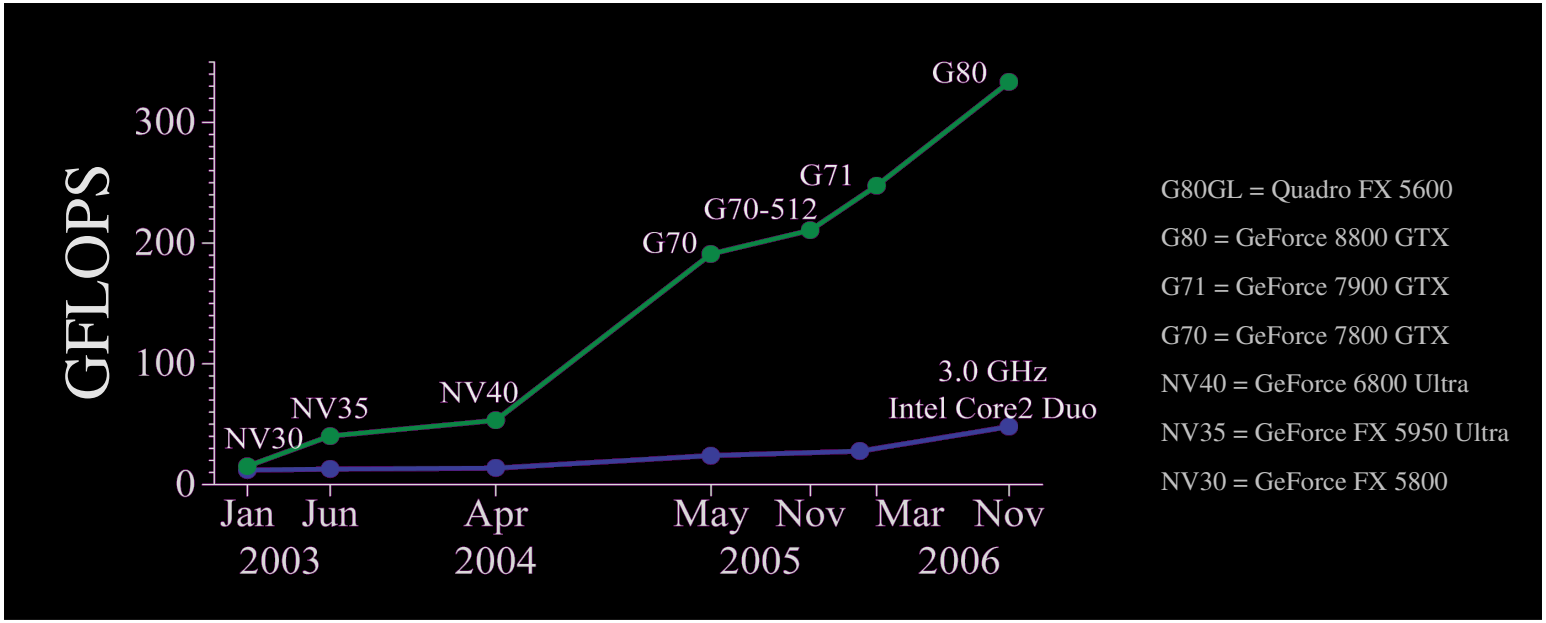
Courtesy Jens Krueger and Aaron Lefohn

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

Why GPU for computing?

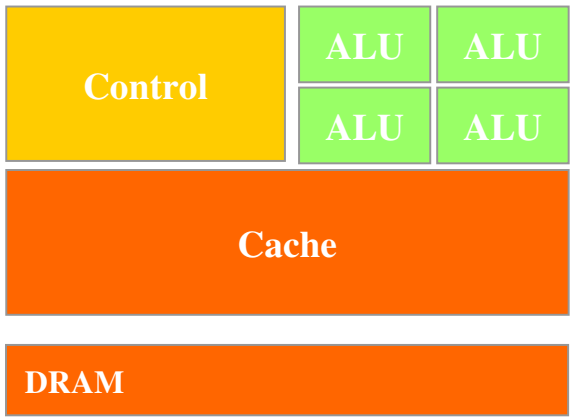
- Inexpensive supercomputer
  - Two NVIDIA Tesla D870 : 1 TFLOPS
- GPU hardware performance increases faster than CPU
  - Trend : simple, scalable architecture, interaction of clock speed, cache, memory (bandwidth)



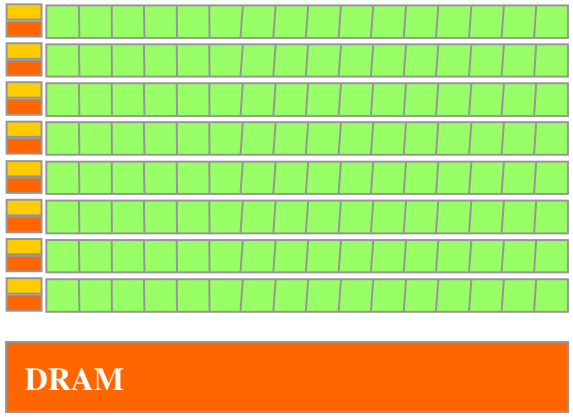
Courtesy NVIDIA

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



CPU



GPU

Courtesy NVIDIA

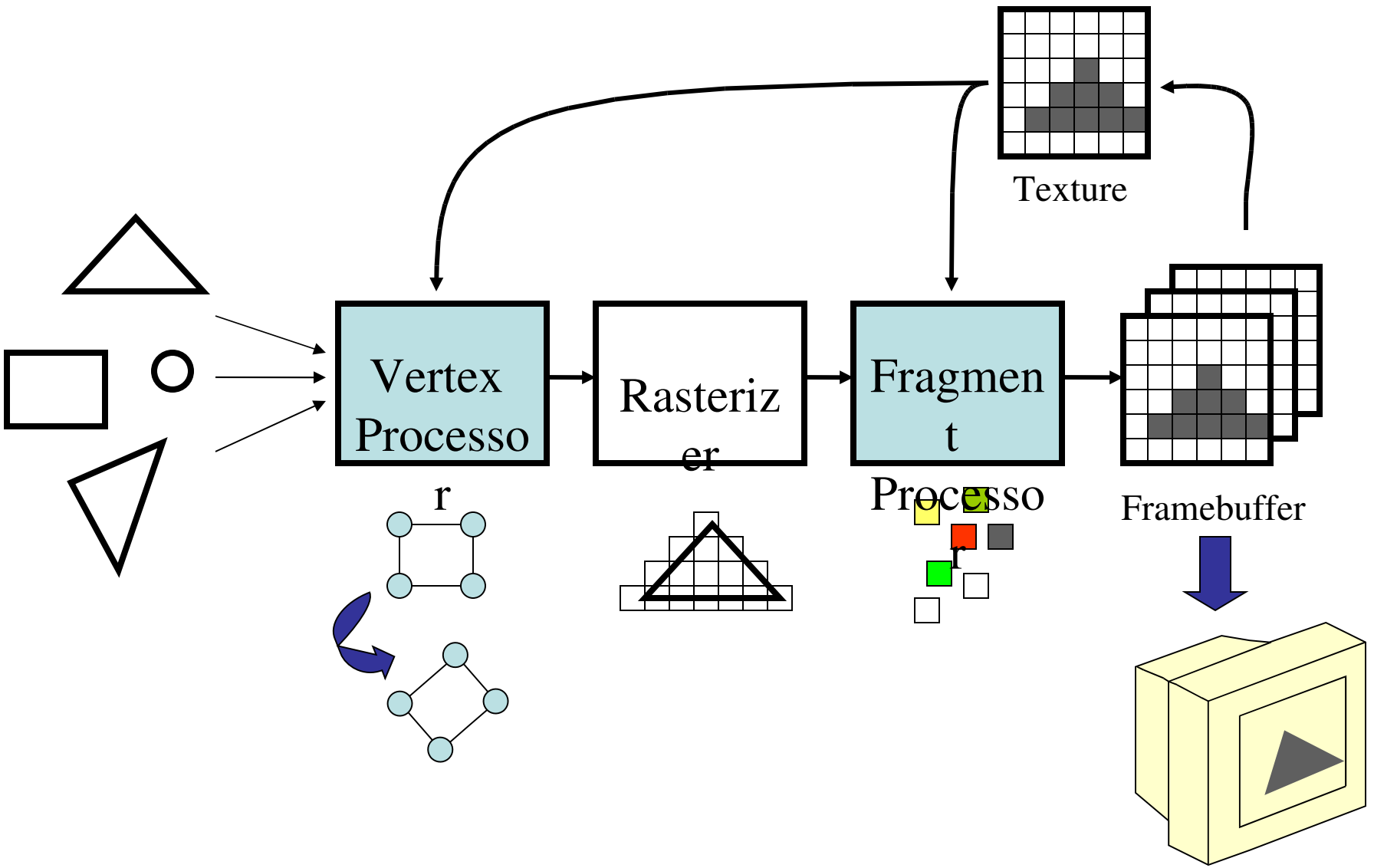
GPU-friendly Problems

- 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)

Traditional GPGPU Model

- GPU as a streaming processor (SIMD)
  - Memory
    - Textures
  - Computation kernel
    - Vertex / fragment shaders
  - Programming
    - Graphics API (OpenGL, DirectX), Cg, HLSL
- Example
  - Render a screen-sized quad with a texture mapping using a fragment shader

Graphics Pipeline



Problems of Traditional GPGPU Model

- Software limitation
  - High learning curve
  - Graphics API overhead
  - Inconsistency in API
  - Debugging is difficult
- Hardware limitation
  - No general memory access (no scatter)
    - B = A[i] : gather (O)
    - A[i] = B : scatter (X)
  - No integer/bitwise operations
  - Memory access can be bottleneck
    - Need coherent memory access for cache performance

NVIDIA G80 and CUDA

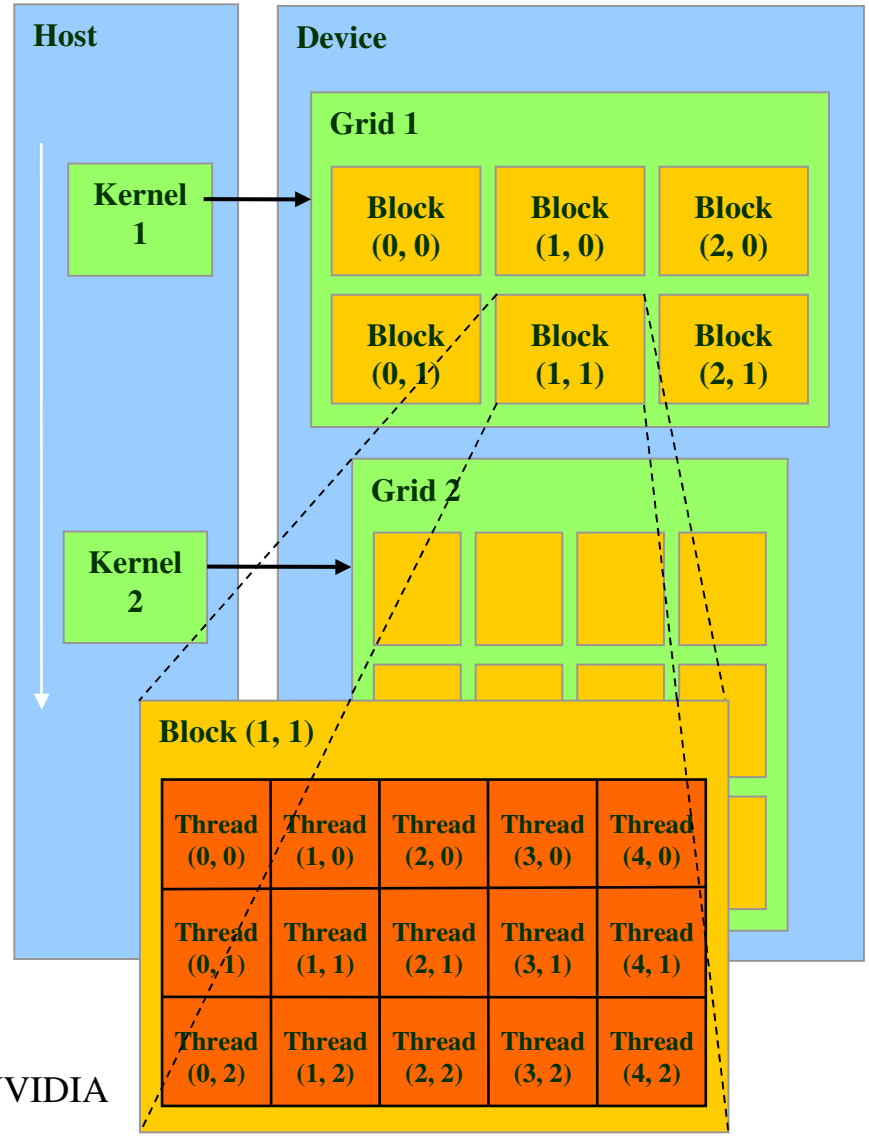
- New HW/SW architecture for computing on the GPU
  - GPU as massively parallel multithreaded machine : one step further from streaming model
  - New hardware features
    - Unified shaders (ALUs)
    - Flexible memory access
    - Fast user-controllable on-chip memory
    - Integer, bitwise operations
  - New software features
    - Extended C programming language and compiler
    - Support debugging option (through emulation)

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

Programming Model: SPMD + SIMD

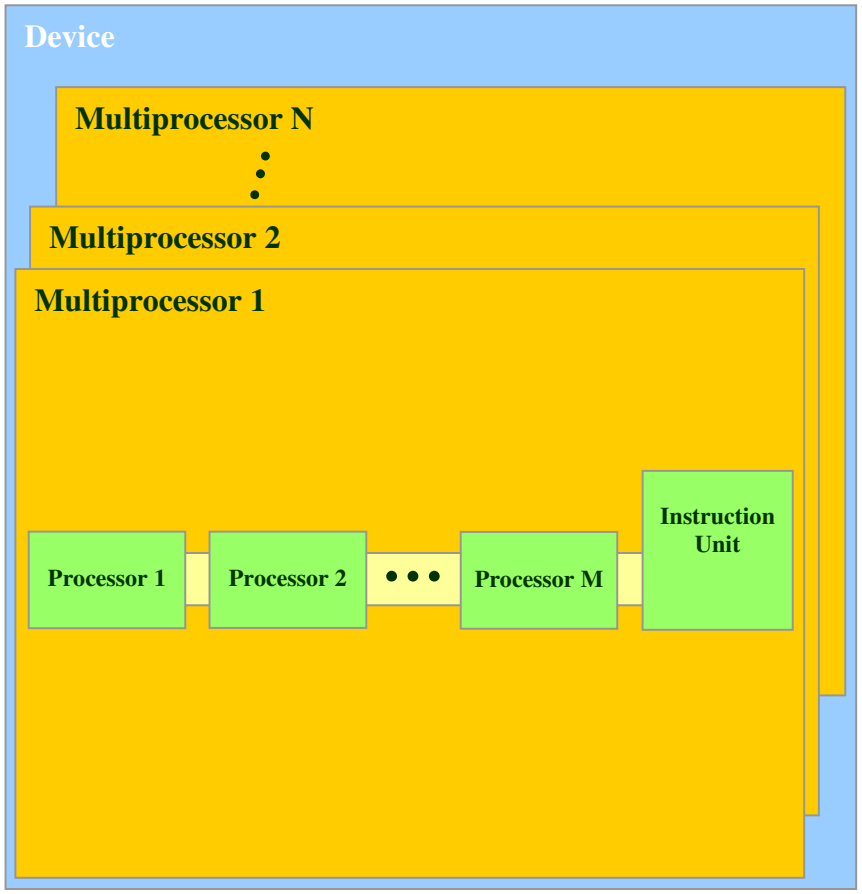
- Hierarchy
  - Device = Grids
  - Grid = Blocks
  - Block = Warps
  - Warp = Threads
- Single kernel runs on multiple blocks (SPMD)
- Single instruction executed on multiple threads (SIMD)
  - Warp size determines SIMD granularity (G80 : 32 threads)
- Synchronization within a block using shared memory



Courtesy NVIDIA

Hardware Implementation : a set of SIMD Processors

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

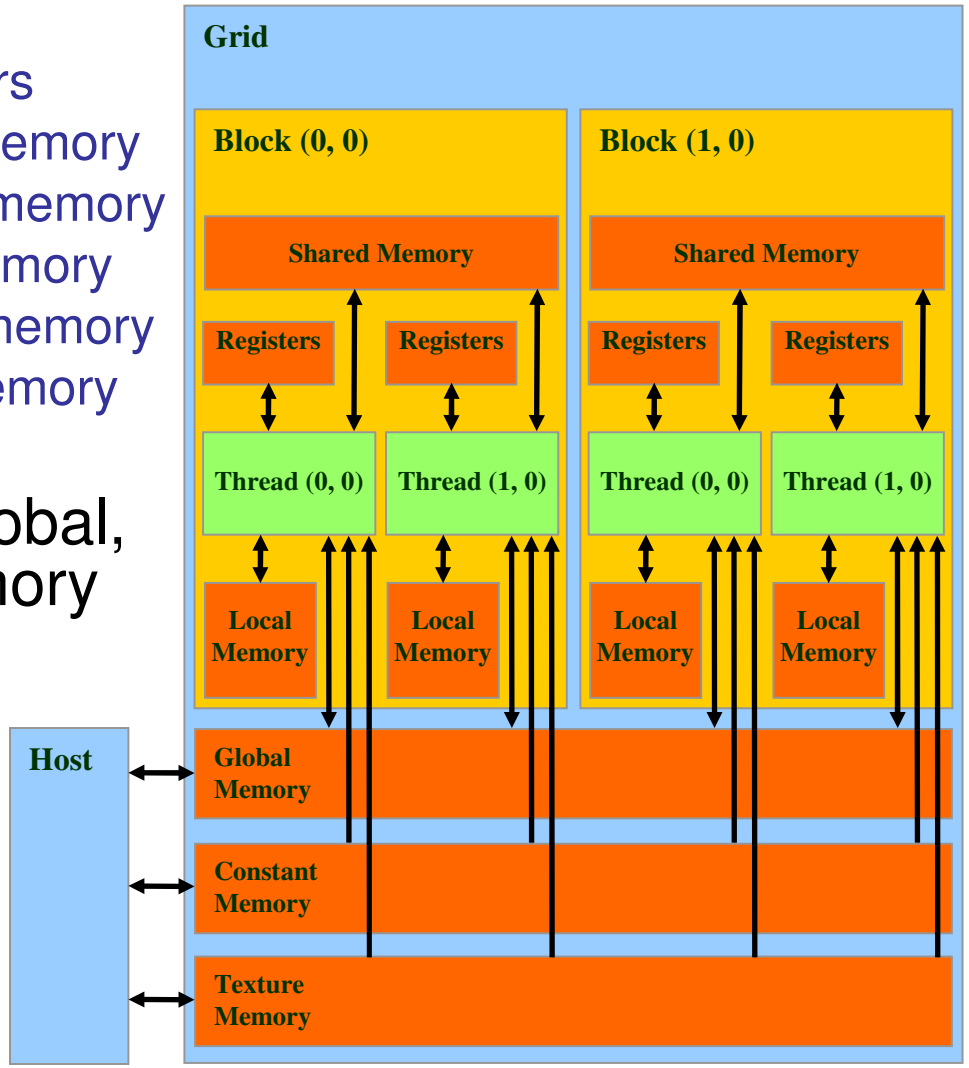


Courtesy NVIDIA

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

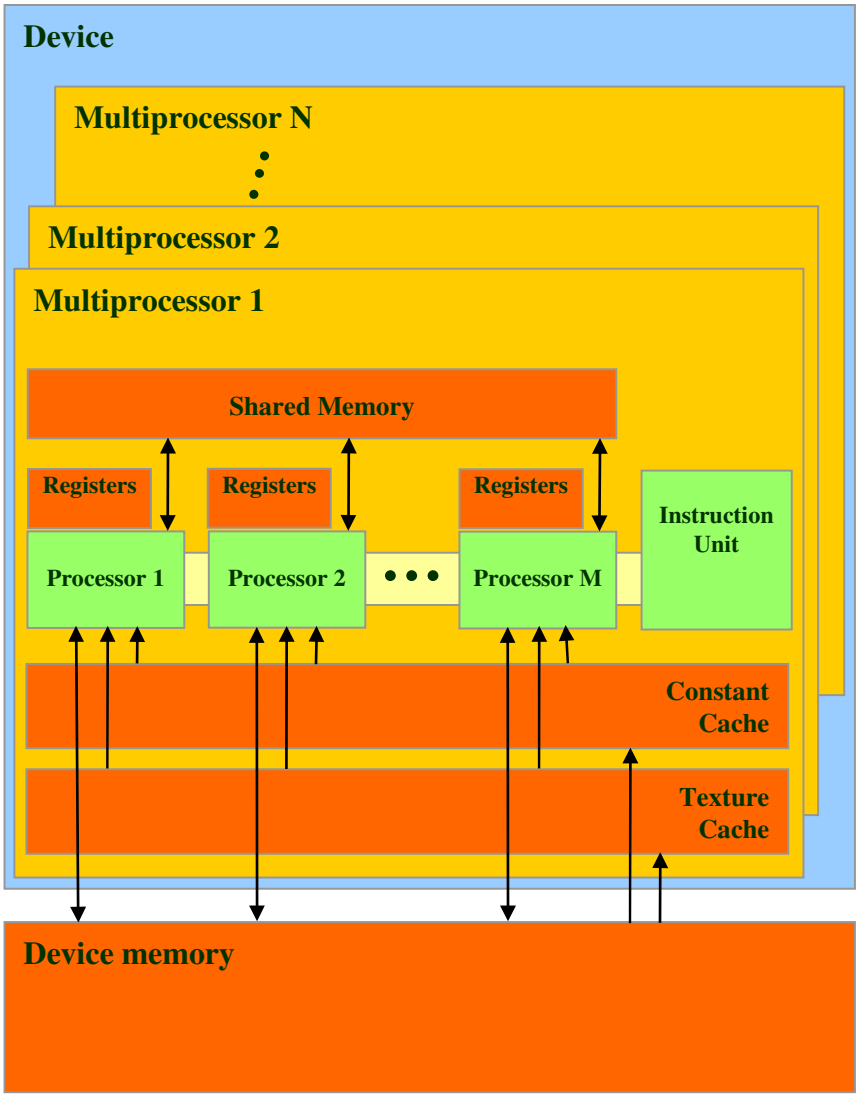
- The host can read/write global, constant, and texture memory



Courtesy NVIDIA

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

Memory Access Strategy

Copy data from global to shared memory

Synchronization

Computation (iteration)

Synchronization

Copy data from shared to global memory

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

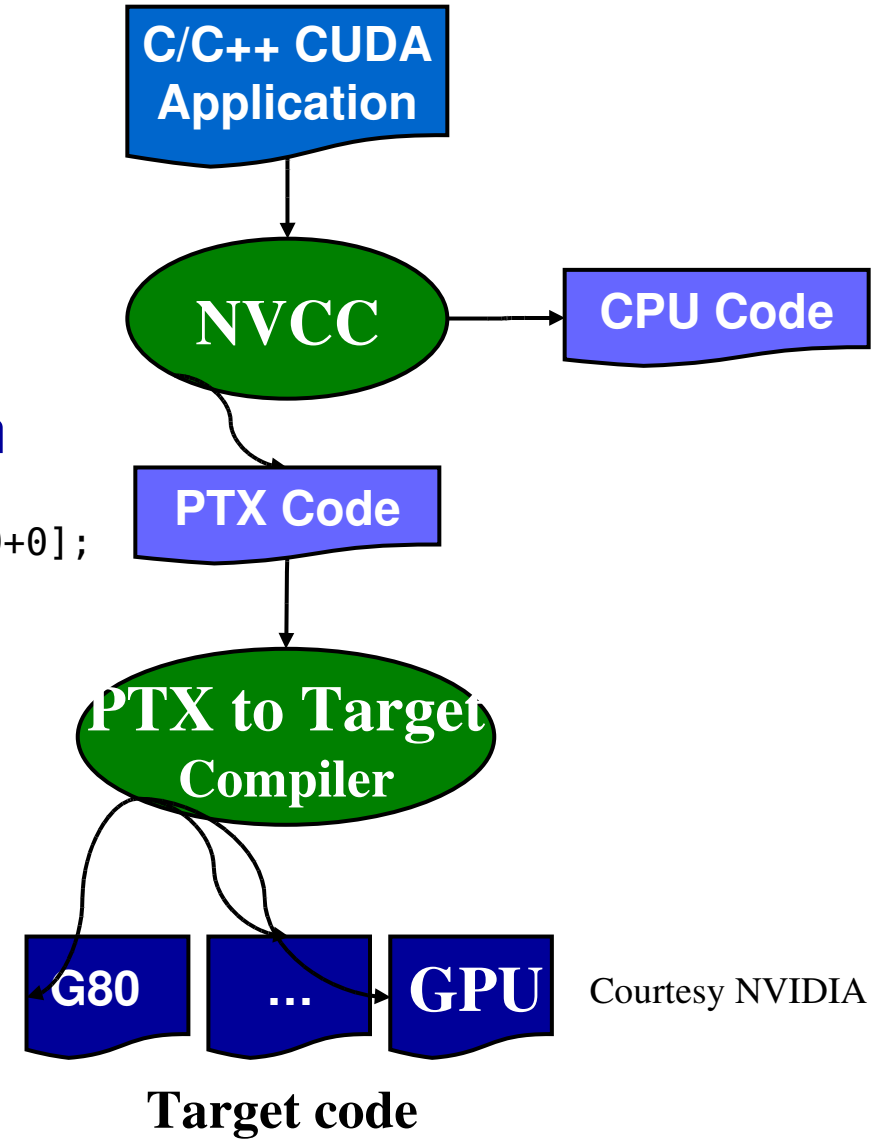
CUDA

- 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

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

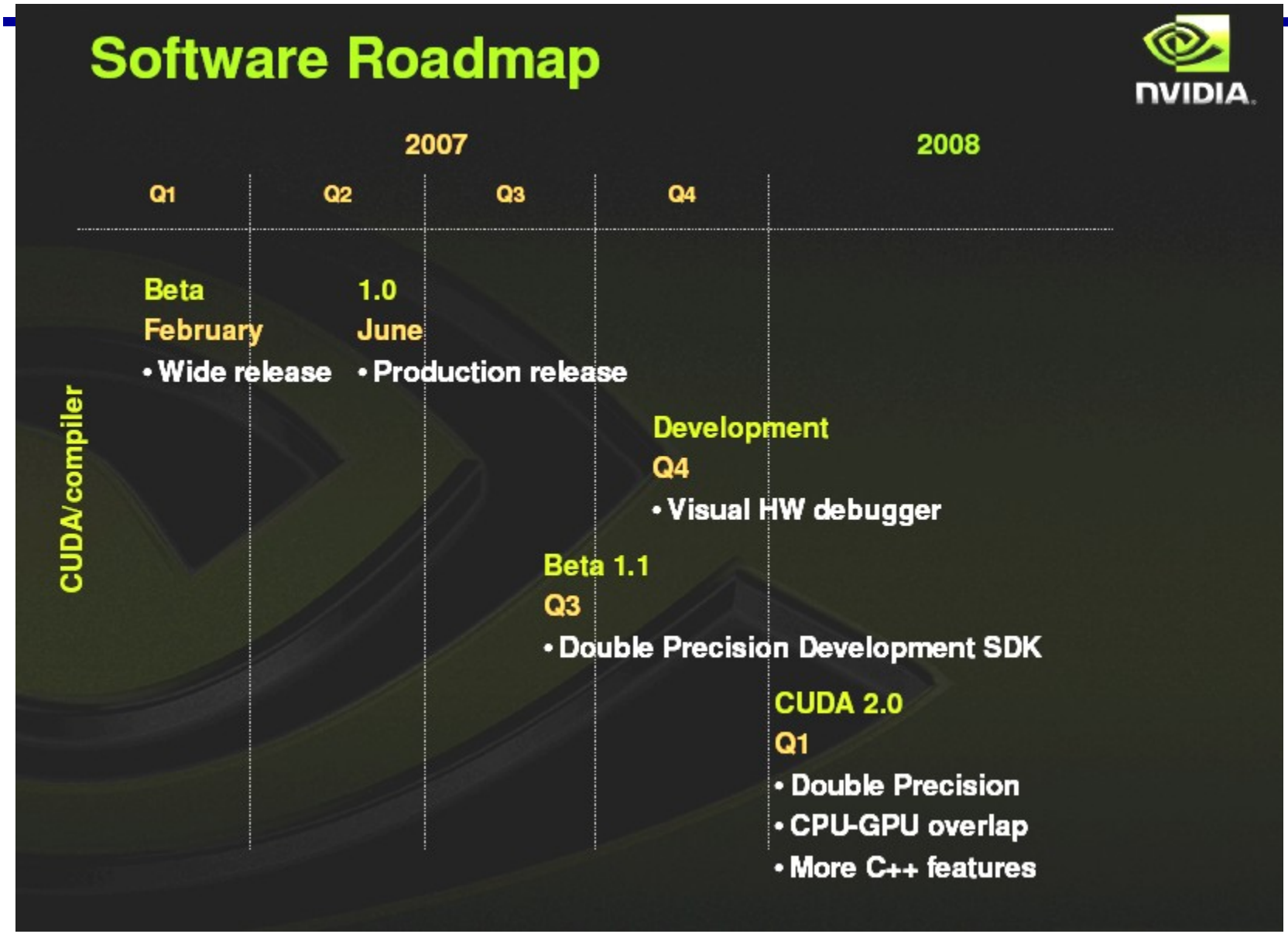
Optimization Tips

- Avoid shared memory bank conflict
  - Shared memory space is split into 16 banks
    - Each bank is 4 bytes (32bit) wide
    - Assigned round-robin fashion
  - Any non-overlapped parallel bank access can be done by a single memory operation
- Coalesced global memory access
  - *Contiguous* memory address is fast
    - a = b[thread_id]; // coalesced
    - a = b[2*thread_id]; // non-coalesced

CUDA Enabled GPUs / OS

- Supported OS
  - MS Windows, Linux
- Supported HW
  - NVIDIA GeForce 8800 series
  - NVIDIA Quadro 5600/4600
  - NVIDIA Tesla series

Software Roadmap



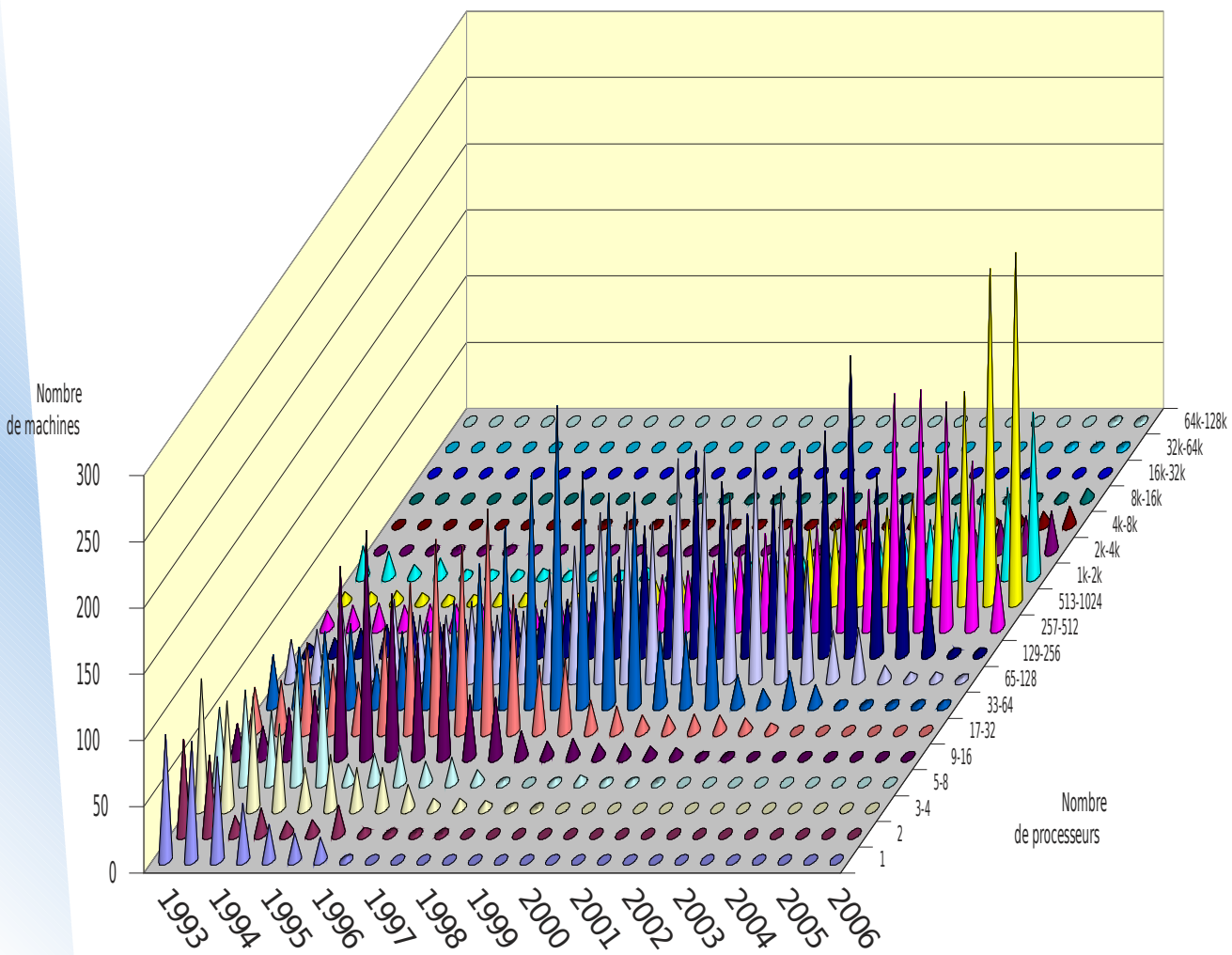
Courtesy NVIDIA

What is a cluster?

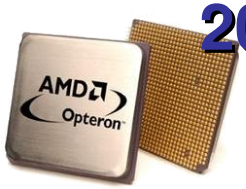
- A cluster is a type of parallel or distributed processing system (MIMD),
  - which consists of a collection of interconnected stand-alone/complete computers cooperatively working together as a single, integrated computing resource.
- A typical cluster:
  - Network: Faster, closer connection than a typical network (LAN)
  - Low latency communication protocols
  - Looser connection than SMP
- Cluster 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

Calcul hautes performances
Évolution du nombre de processeurs

1998
ASCI RED
1TFlop
10k Ppro@333

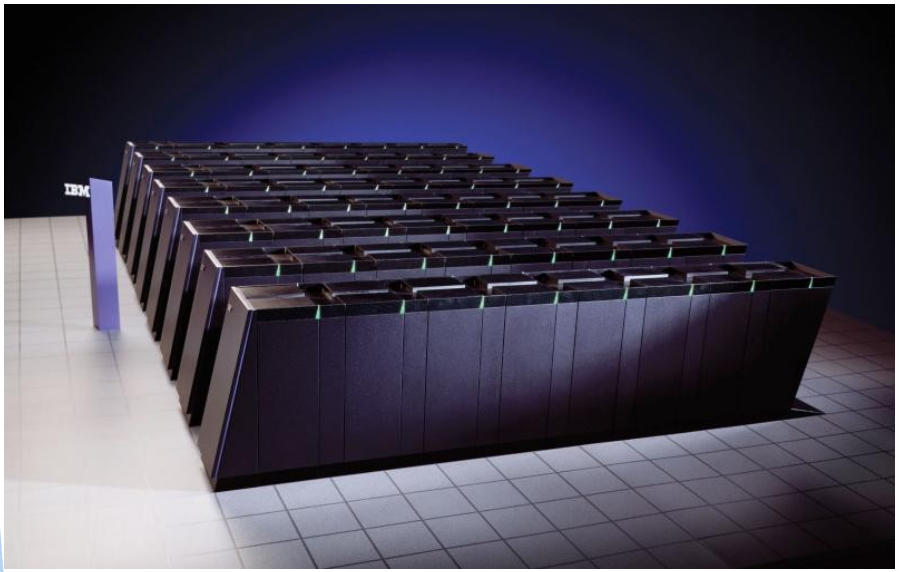


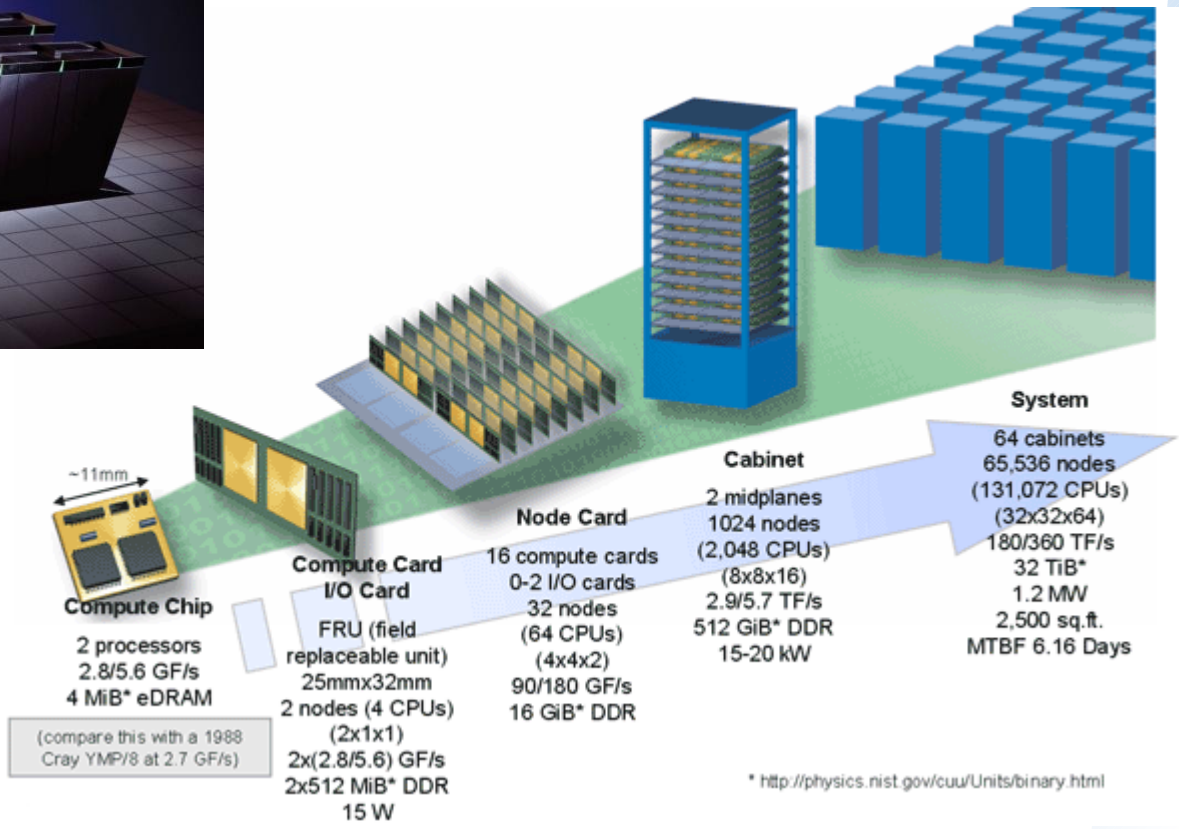
2006 Xbox 360
1TFlop
1 GPU



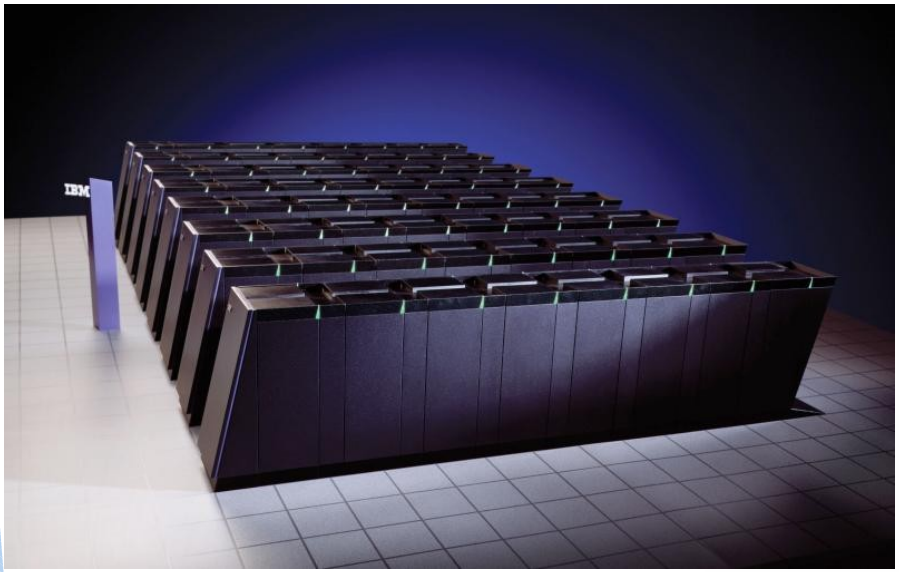
2006 AMD
Opteron
3Gflop

Calcul hautes performances
Programmation des machines parallèles

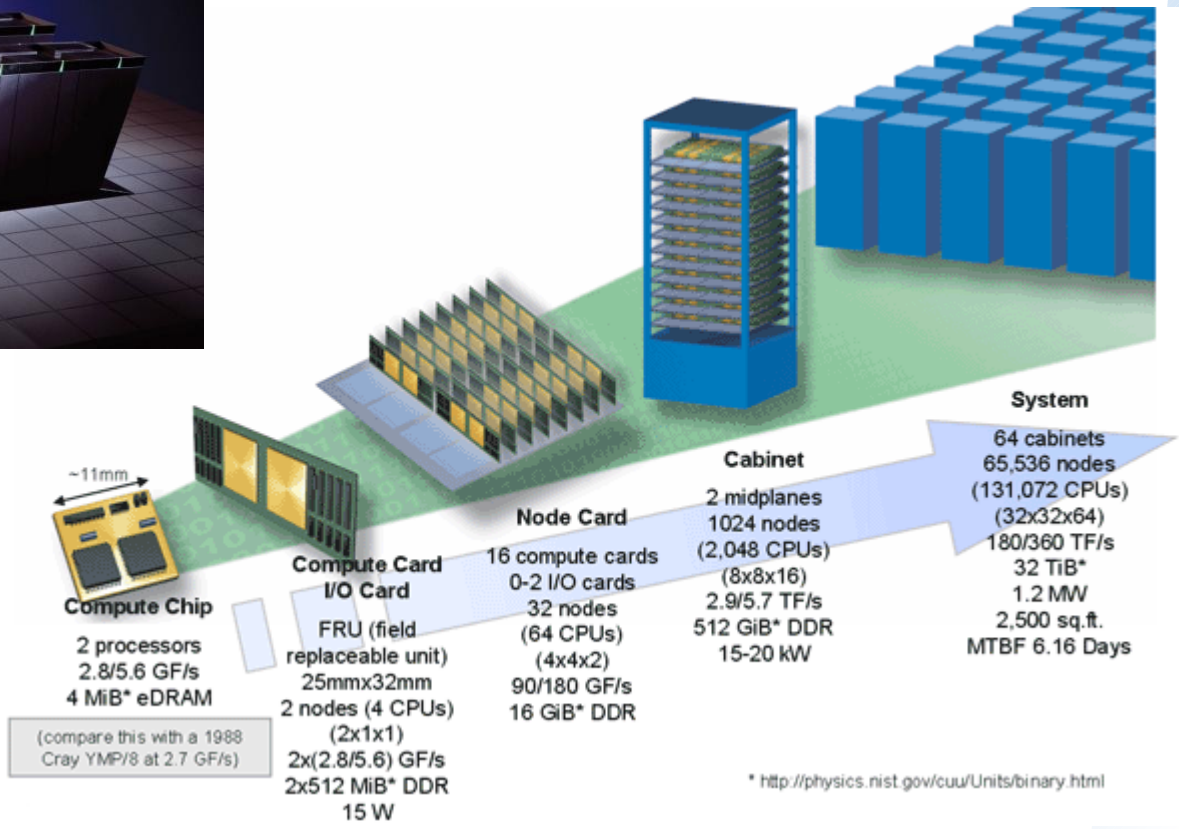




Calcul hautes performances
Programmation des machines parallèles



Communications entre processus
Mémoire partagée : OpenMP
Passage de message : MPI



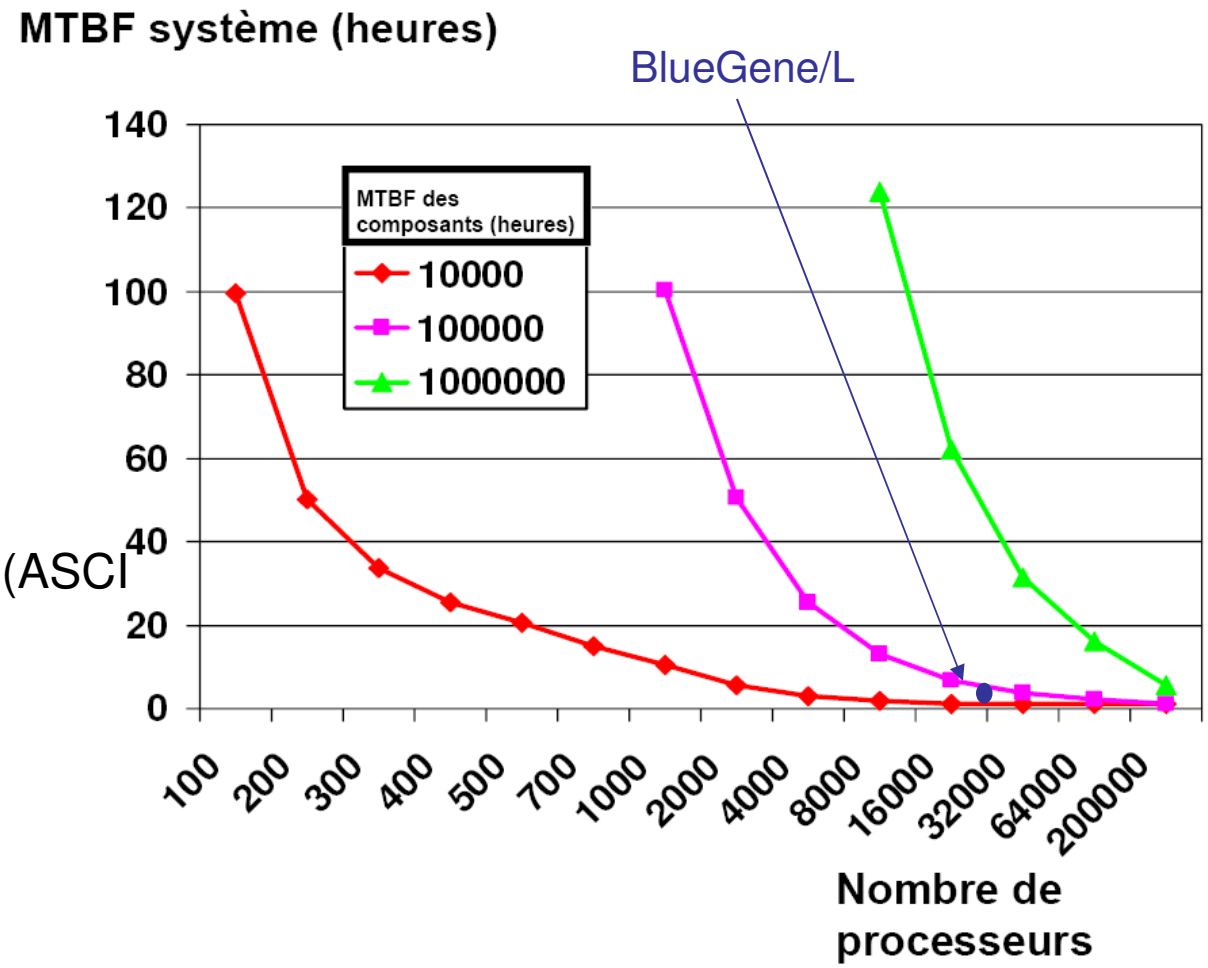
Calcul hautes performances
Évolution du MTBF dans le futur

1 PetaFlop =

200k 5Gflop CPU

Avec du matériel fiable actuel (ASCI white), une machine de cette dimension subit

1 défaillance par heure

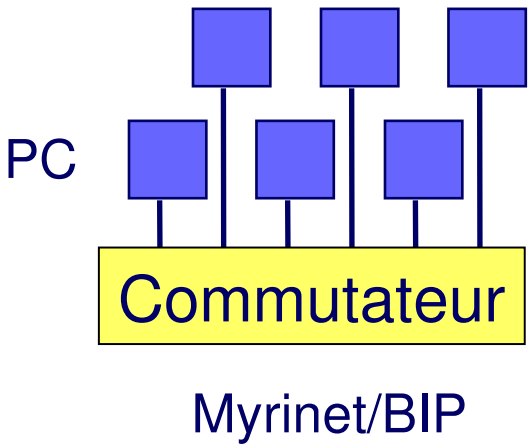


What is a cluster?

- 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.
- A typical cluster:
  - Network: Faster, closer connection than a typical network (LAN)
  - Low latency communication protocols
  - Looser connection than SMP
- Cluster 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

Cluster computing

- Architecture homogène, faiblement hétérogène
  - Grappes (Cluster, COW), machines //
    - PC, stations de travail
    - SCI, Myrinet, Giganet, MPC, ...
  - Protocoles de communication
    - BIP, SISCI, SciOS, VIA, TCP, UDP, ...



81.6 Gflops (216 nodes) + top 500 (385) June 2001
http://clic.mandrakesoft.com

HP Cluster : 104 HPBi Itanium 2

Example Clusters: Berkeley NOW

- 100 Sun UltraSparcs
  - 200 disks
- Myrinet SAN
  - 160 MB/s
- Fast comm.
  - AM, MPI, ...
- Ether/ATM switched external net
- Global OS
- Self Config

Motivation for using Clusters

- The communications bandwidth between workstations is increasing as new networking technologies and protocols are implemented in LANs and WANs.

- Workstation clusters are easier to integrate into existing networks than special parallel computers.

Motivation for using Clusters

- Surveys show utilisation of CPU cycles of desktop workstations is typically <10%.
- Performance of workstations and PCs is rapidly improving
- As performance grows, percent utilisation will decrease even further!
- Organisations are reluctant to buy large supercomputers, due to the large expense and short useful life span.

Motivation for using Clusters

- 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 clusters are a cheap and readily available alternative to specialised High Performance Computing (HPC) platforms.
- Use of clusters of workstations as a distributed compute resource is very cost effective - incremental growth of system!!!

Cluster Components...1a Nodes

- Multiple High Performance Components:
  - PCs
  - Workstations
  - SMPs (CLUMPS)
- Cluster is mainly homogeneous
  - Node (processor, memory, cache, disk)
  - Operating System
  - Network

Cluster Components...1bProcessors

- There are many (CISC/RISC/VLIW/Vector..)
  - Intel: Pentiums, Xeon, Merceed….
  - Sun: SPARC, ULTRASPARC
  - HP PA
  - IBM RS6000/PowerPC
  - SGI MIPS
  - Digital Alphas
- Integrate Memory, processing and networking into a single chip
  - IRAM (CPU & Mem): (http://iram.cs.berkeley.edu)
  - Alpha 21366 (CPU, Memory Controller, NI)