Parallel Application Development Issues

G. Colin de Verdière
CEA, DAM, DIF, F-91297, Arpajon, France
guillaume.colin-de-verdiere@cea.fr

Overview

- Presentation of CEA
- Why CEA is doing Parallel Computing
- The various options
- Recommendations

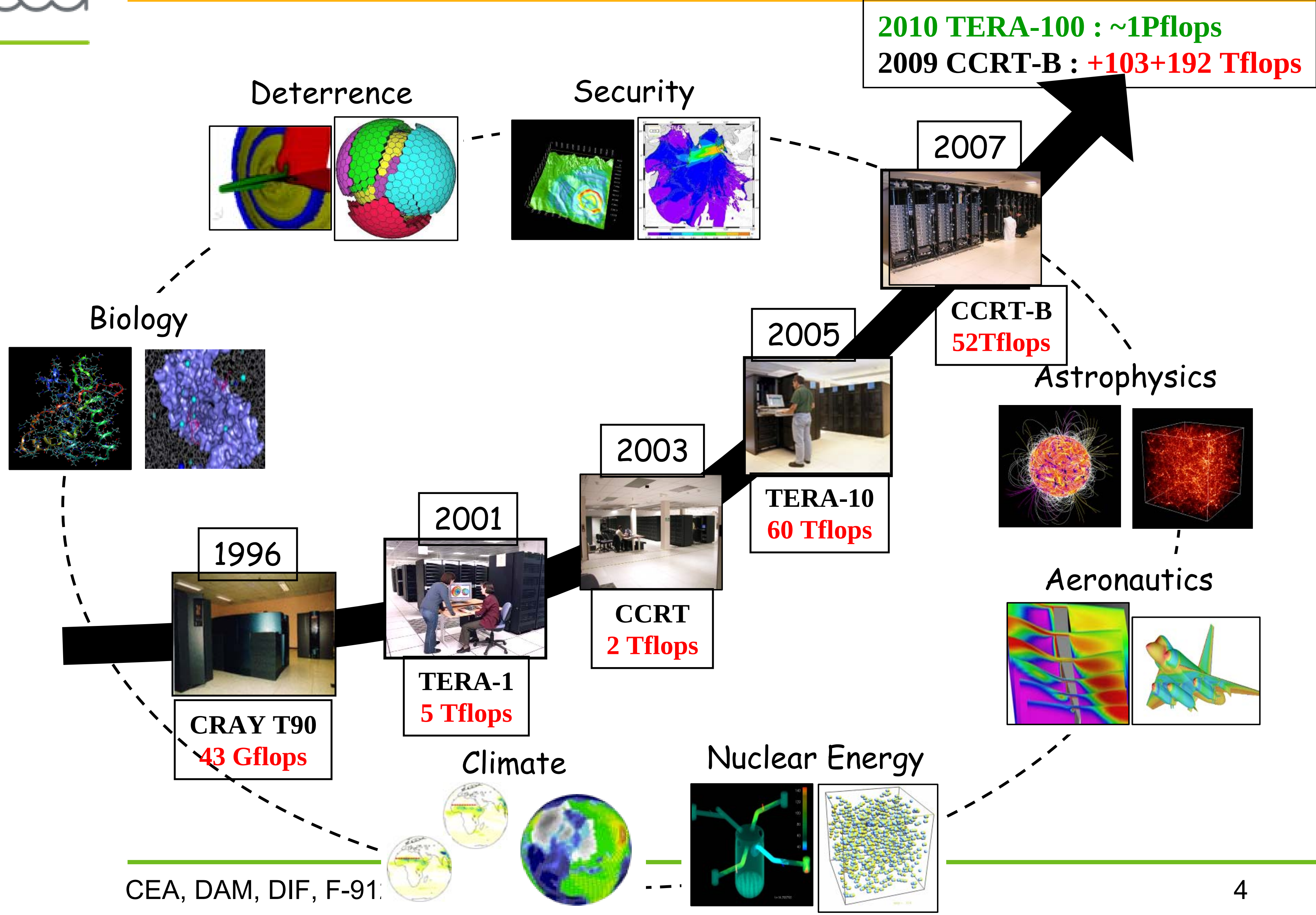
Defense

Energy

Technologies for information and health

The French Atomic Energy Commission

TERA/CCRT : Applications and Evolution



The new CCRT cluster

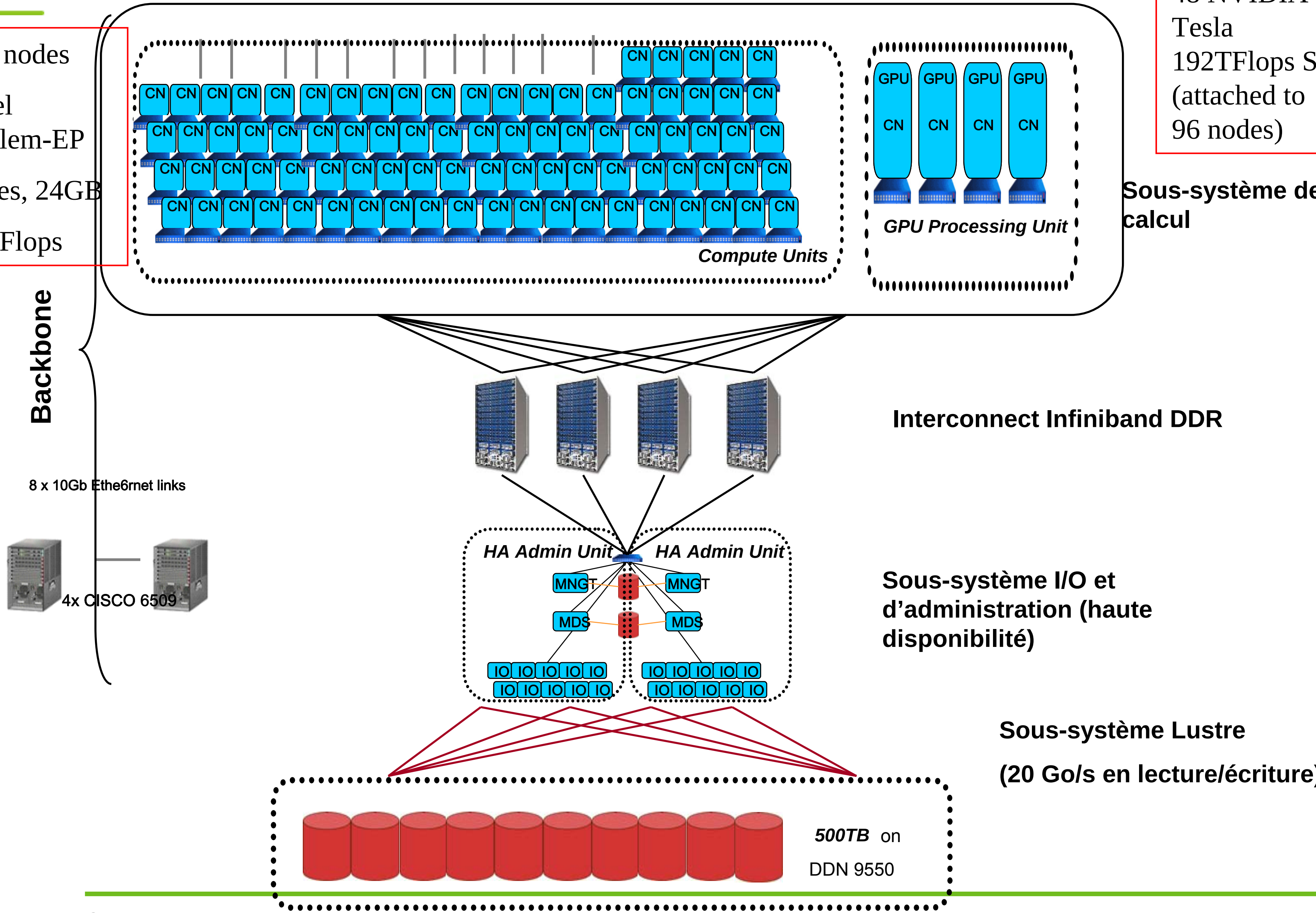
1068 nodes

2 Intel Nehalem-EP

8 cores, 24GB

103TFlops

48 NVIDIA Tesla 192TFlops SP (attached to 96 nodes)



Future machines

- We have to design the next generation of production machines
  - For Research and Technology (CCRT)
  - For Defense programs (TERA)

If petaflops machines will be available soon (2010),

Exascale Computing is the next focus

- We are limited by power resources
  - Is 100+ MW reasonable ?

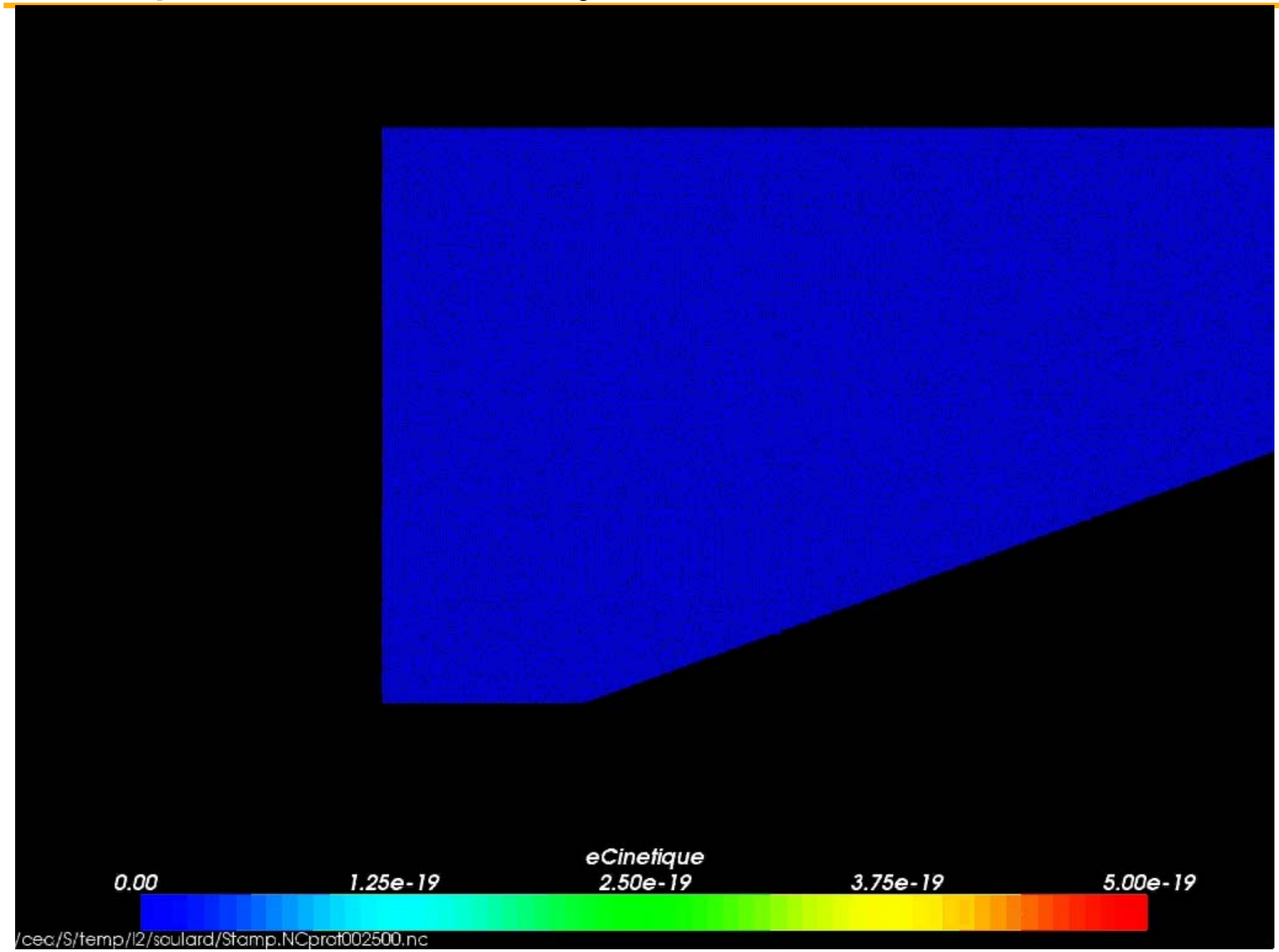
- There's no foreseen breakthrough in processor technology
  - Our goal is to provide General Purpose machines

- How to program those machines?

CEA Context: codes

- Types of codes
  - Research for physics
    - Special purpose, advanced methods
  - R&D for production
    - Highly coupled physics models
    - Finite elements
    - Structured and un structured meshes, AMR, PIC, …
    - Has to be validated to guarantee our designs
- 50+% F90, remainder is C/C++
- Runs are large
  - 10^6 – 10^9 cells
  - Produce > 20TB / day

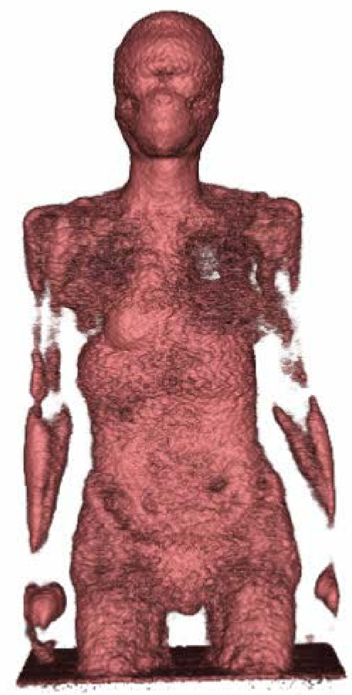
Example: Molecular Dynamics © CEA/DAM



- 7 millions molecules, 300000 time steps (75ns)
- 1000 processors, 100h, Code STAMP
- Credit: Laurent Soulard

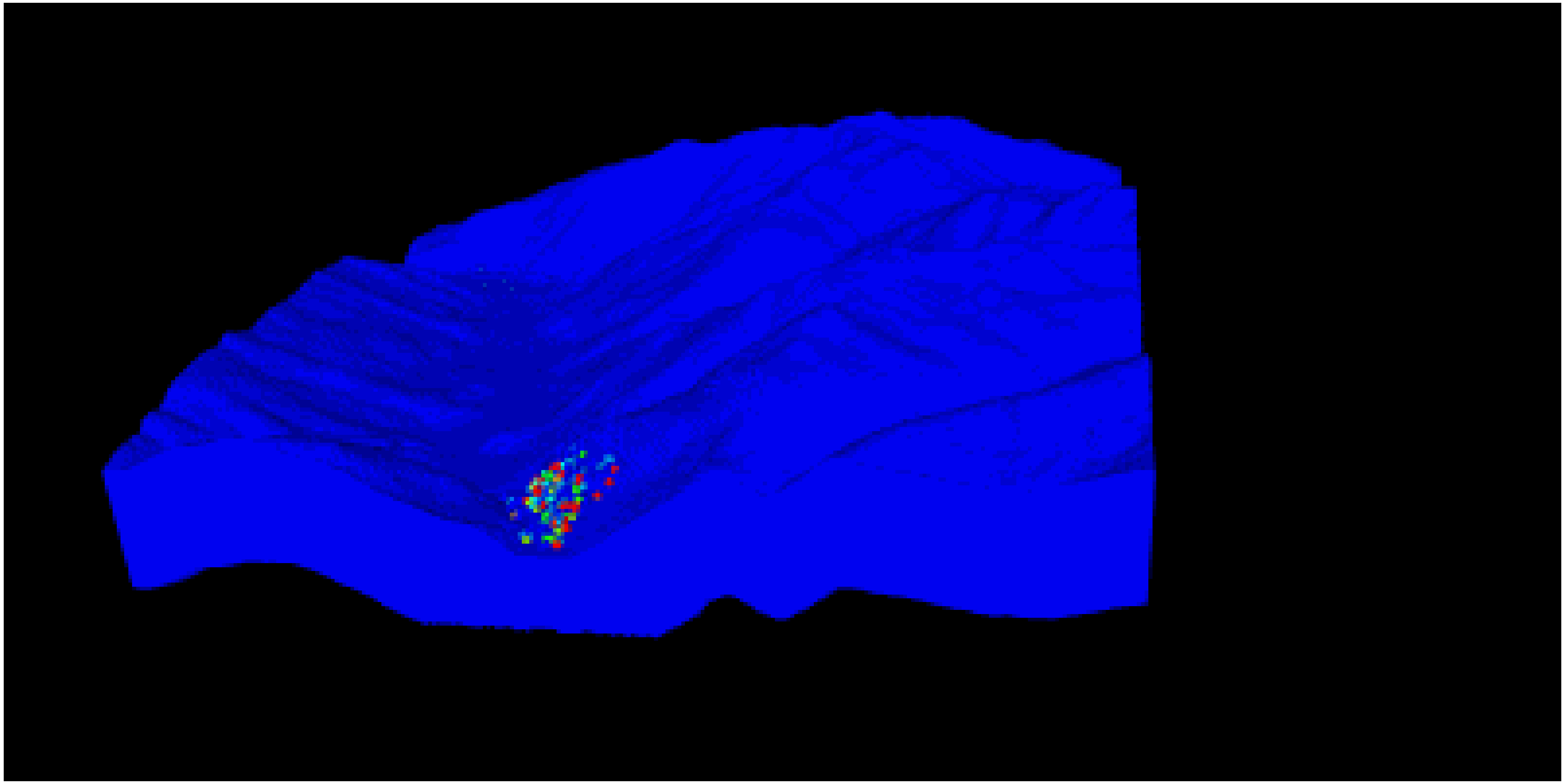
- Explosion front in a explosive
- Grand Challenge on TERA 10

Example: Life science © CEA/DSV



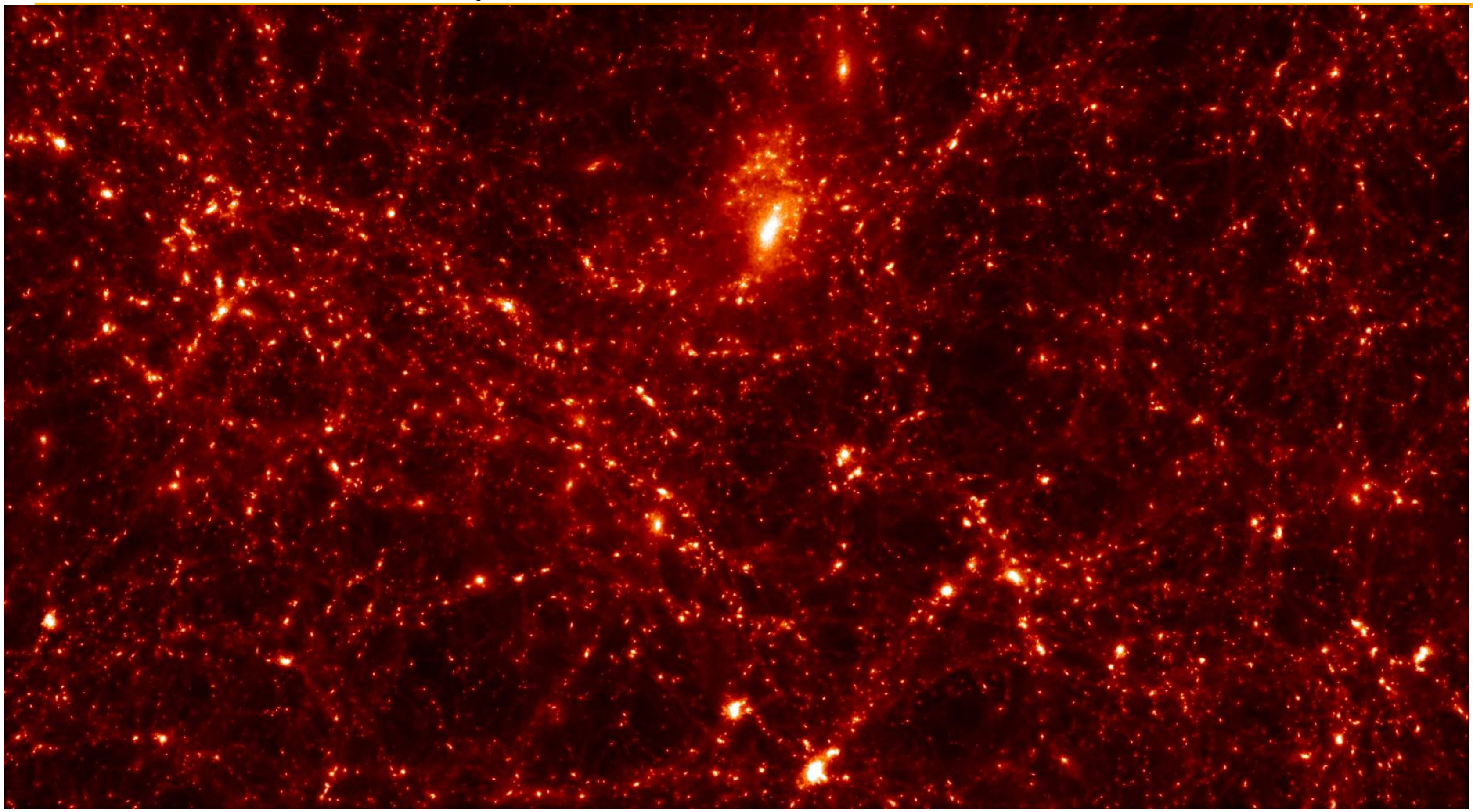
- A few 10^{11} positons to simulate
- Code OpenGATE
- 2.5h, 7000 processors
- Credit: S. Jan
- TEP simulation
- Grand Challenge on TERA 10
- Visualization: 8 cores, 4h

Example: Risk prevention © CEA/DAM



- 11 km x 11 km x 2 km, 500 processors, 40h
- Code MKa3D
- Credit: C. Mariotti
- 5.5 magnitude seism simulation
- Visualization: 16 CPU, 500h

Example: astrophysics © CEA/IRFU



- 70 billion particles, 140 billion cells
- 6144 processors, 18TB RAM, 2 months,
- Code RAMSES
- Credit: Romain Teyssier et al.

- Formation of structures in the universe.
- Grand Challenge on CCRT

Parallelism is required

- Every new application MUST be parallel
  - Problem setup, simulation, post-processing
- What are the options?
  - Multicore
    - OpenMP
    - pthreads
  - Cluster
    - MPI
  - GPU (manycore)
- What level of parallelism?
  - Depends on the granularity possible
    - Embarrassingly parallel problems
    - Domain decomposition
      - Ghost cells
    - Loop level
- You have to verify parallelism
  - Results must be exactly the same in parallel or in sequential
    - Beware of random generators (Monte Carlo)

Do not forget Amdahl's law

n=∞

f=0.00 s=1

f=0.10 s=1.1

f=0.20 s=1.25

f=0.30 s=1.42

f=0.40 s=1.66

f=0.50 s=2

f=0.60 s=2.5

f=0.70 s=3.33

f=0.80 s=5

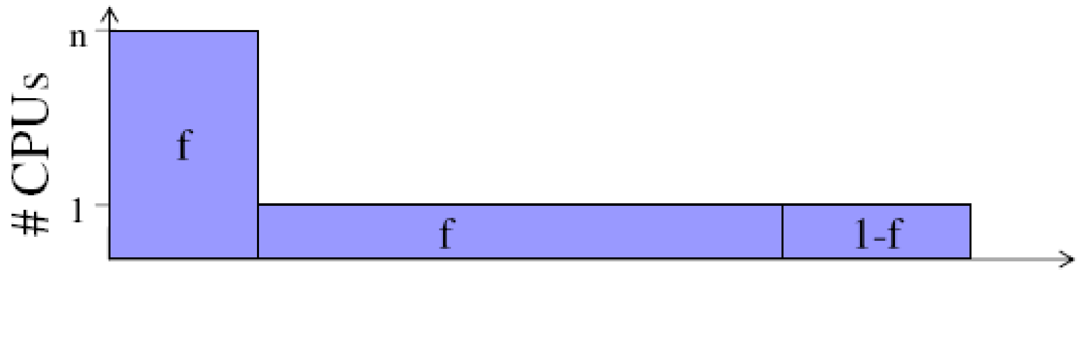
f=0.90 s=10

f=0.95 s=20

f=0.99 s=100

f=0.999 s=1000

Amdahl's Law



f – fraction that can run in parallel

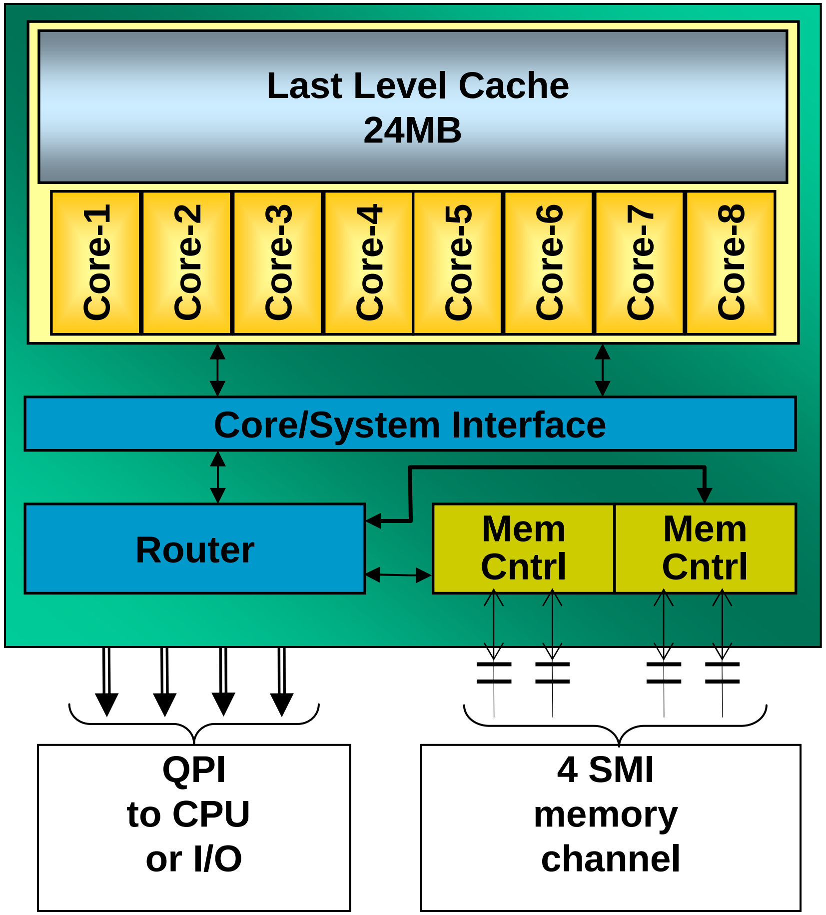
1-f – fraction that must run serially

$$Speedup = \frac{1}{(1-f) + \frac{f}{n}}$$

$$\lim_{n \to \infty} \frac{1}{1-f+\frac{f}{n}} = \frac{1}{1-f}$$

Think massively parallel!

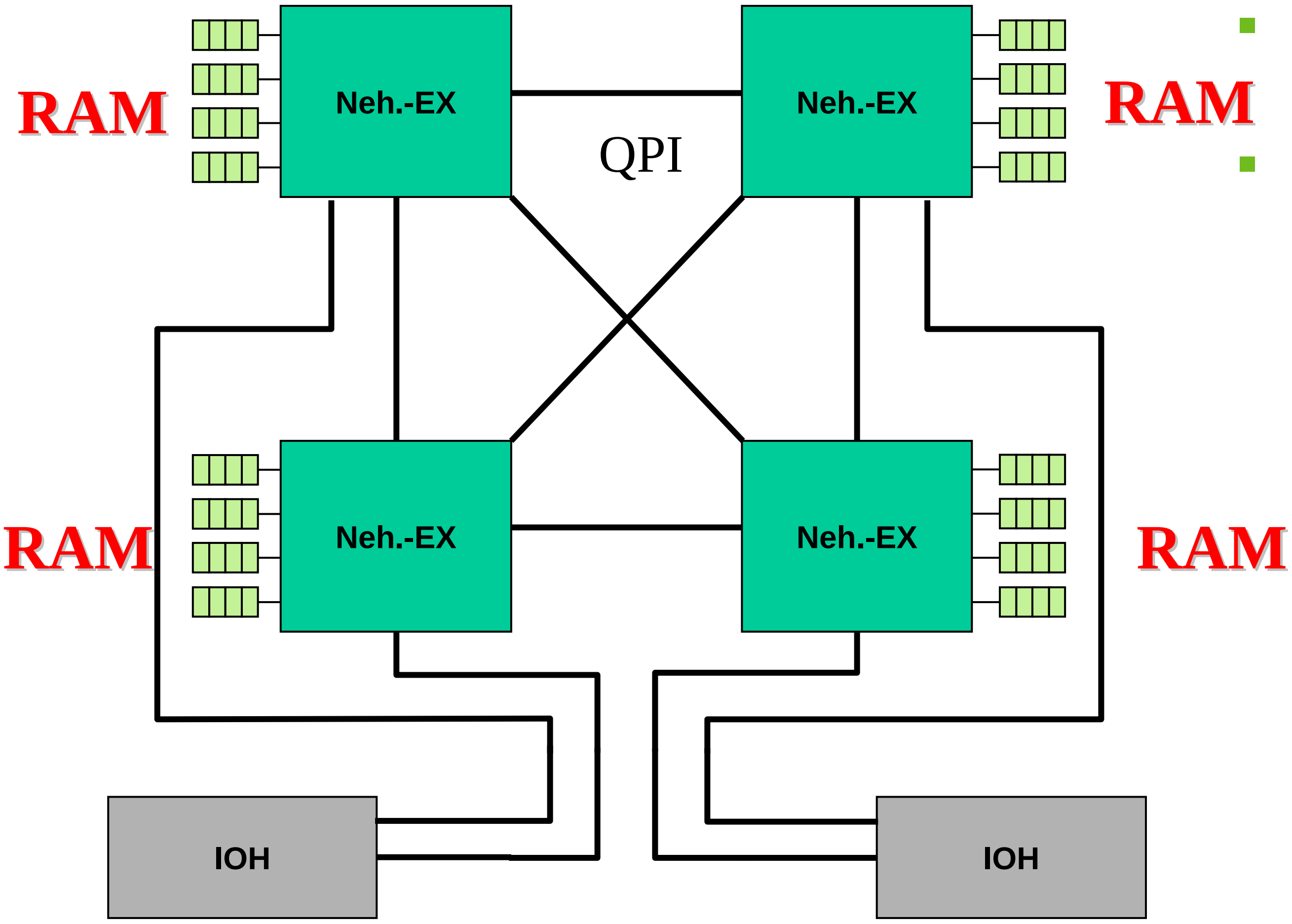
Hardware Architecture: a processor



Nehalem-EX

- Memory is banked
  - Reuse old practices from CRAY vector machines

Hardware Architecture: a node with 4 sockets

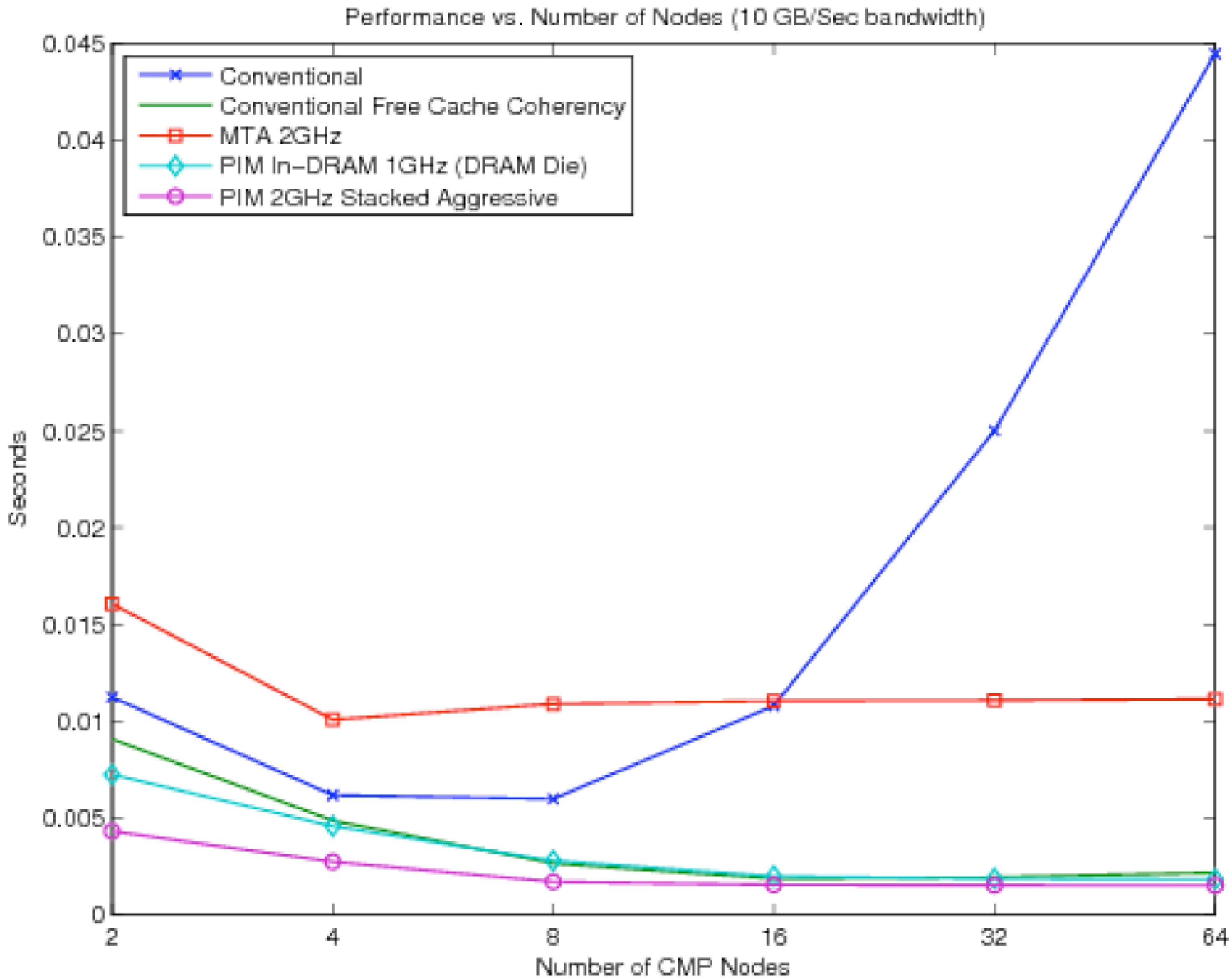


- All RAM positions are
  - Available to each processor
  - not equal in access time
    - Lock process to CPU
    - Lock allocated memory to a CPU

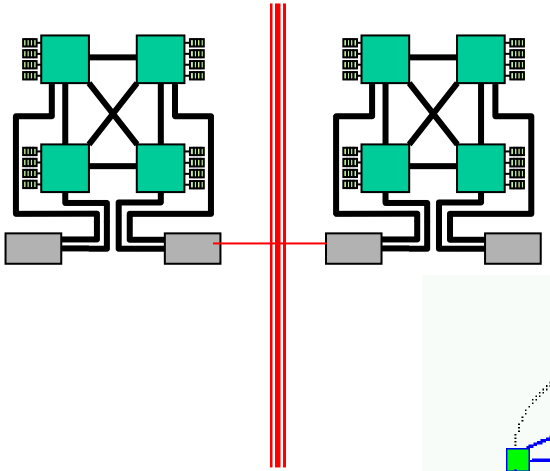
Multicore (N < 32)

- Multithread and OpenMP
  - two different ways to exploit multicore on the node
- Multithread
  - Well suited for graphic user interfaces
  - Many implementations
    - Pthreads is the most widely used
    - Intel TBB – for C++ codes
  - Hard to program
- OpenMP
  - Easy to learn
  - Well adapted to loop level parallelism
  - `#pragma` (ftn) or `!$` (c) => leaves the code unchanged
- MPI
  - Works also;
- Impact on libraries
    - Should be threads safe + re-entrant

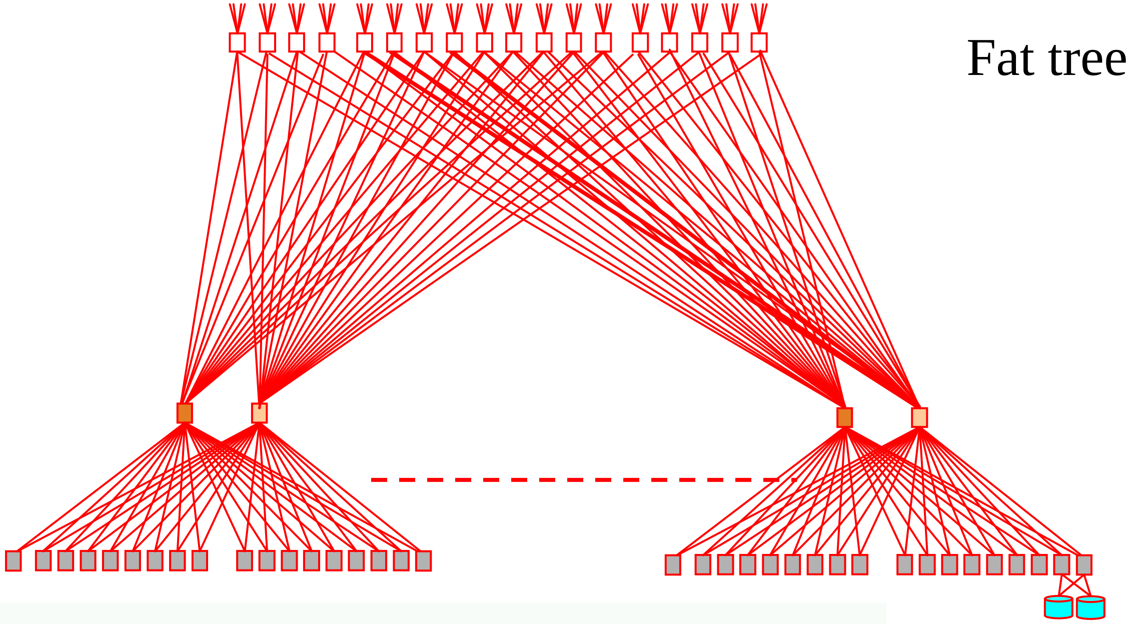
http://www.sandia.gov/news/resources/releases/2009/multicore.html



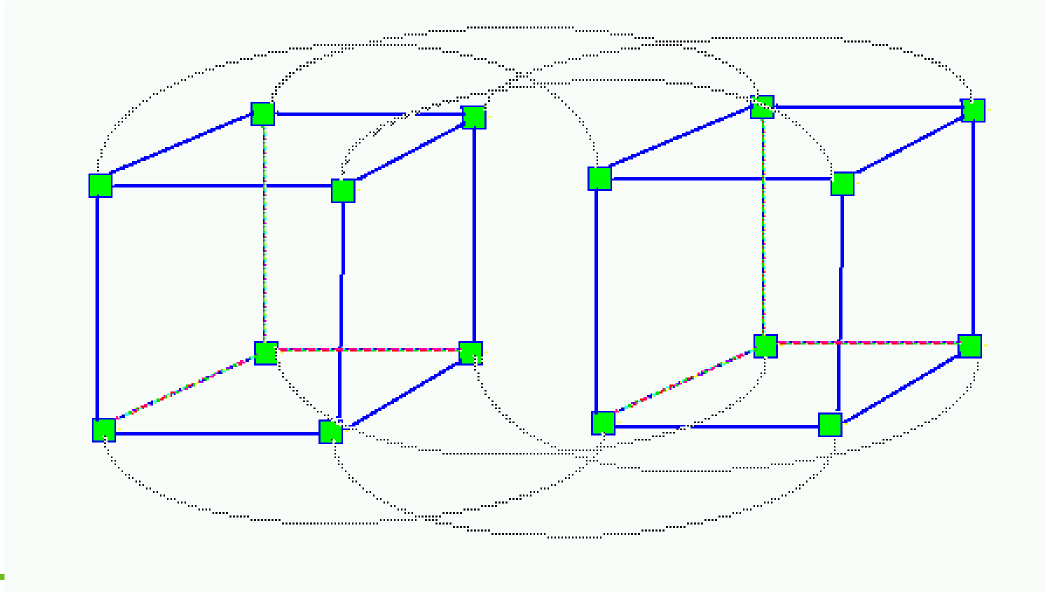
Hardware Architecture: clustering nodes



network



Fat tree



4D
hypercube

Cluster

- MPI (Message Passing Interface): the most widely used library on clusters
  - Many implementations optimized for the hardware
  - Easy to program
  - Scalable (should still work for exascale computers)
  - Can be adapted to the network topology
- Requires a careful study of communication patterns
  - The memory structure of the code must be well understood
  - Communications can take most of the time
- Master / slave structures don't scale
- Impact on libraries
  - None
  - If the library uses MPI: should have its own communicator

OpenMP + MPI or OpenMP / MPI

- OpenMP and MPI can coexist peacefully
  - May require some tuning : cpuset / memset
  - Put MPI in a single OpenMP thread
- What to choose ?
  - If MPI on a node is efficient then MPI only
  - It all depends on the largest problem size which will be computed
  - Study the memory layout of the application
- MPI
  - Good for problems which can be divided by blocks
  - Very intrusive work, no automatic process
  - Excellent for embarrassingly parallel jobs
- OpenMP
  - If needs a lot of memory but hard to parallelize then OpenMP on fat nodes
  - Loop oriented
    - Scalable up to the number of nodes (at most if it goes well)
  - Minimal work and code almost unchanged
    - If the original code was properly developed

Parallelism: an active field of research

- Algorithms
- Processor architecture
  - Intel Larrabee, AMD Fusion

- Languages: Partitioned Global Adress Space
  - UPC, Co-array Fortran, Fortress, Chapel, X10
  - Interesting notions, still under development
    - Far from being productions tools
    - May need some extensive rewrite
      - What should be done with million of lines of legacy codes ?

In our production

- MPI mostly
  - Took time to do the migration
- OpenMP not performing as well as expected
  - Position should be revised with Intel’s Nehalem
- Very few multithreaded codes
  - Active field of investigation though

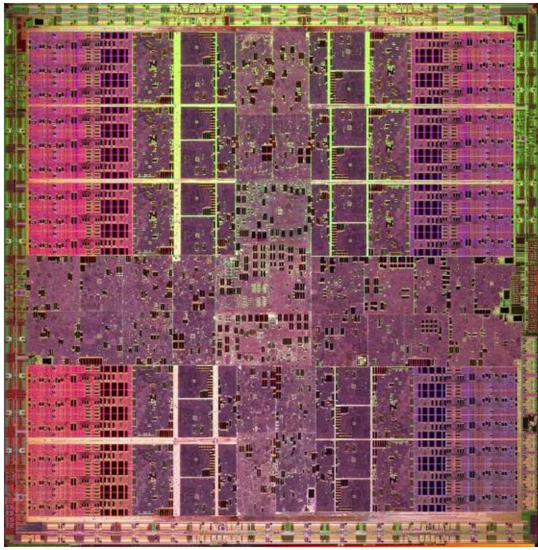
Manycore (N>32)

- As of 2009, we think that the future architectures will be accelerated
  - Good ratio power / performance
  - Good ratio density / performance
- GPU Computing is our current compromise/answer
  - In terms of performances and power
  - We wait for the market to get clearer (Larrabee ?)
- Constraints of this type of hardware as of today
  - No ECC
  - Debugging tools
  - A choice of non standard languages

Available configurations at CEA

Tesla S1070 credit NVIDIA

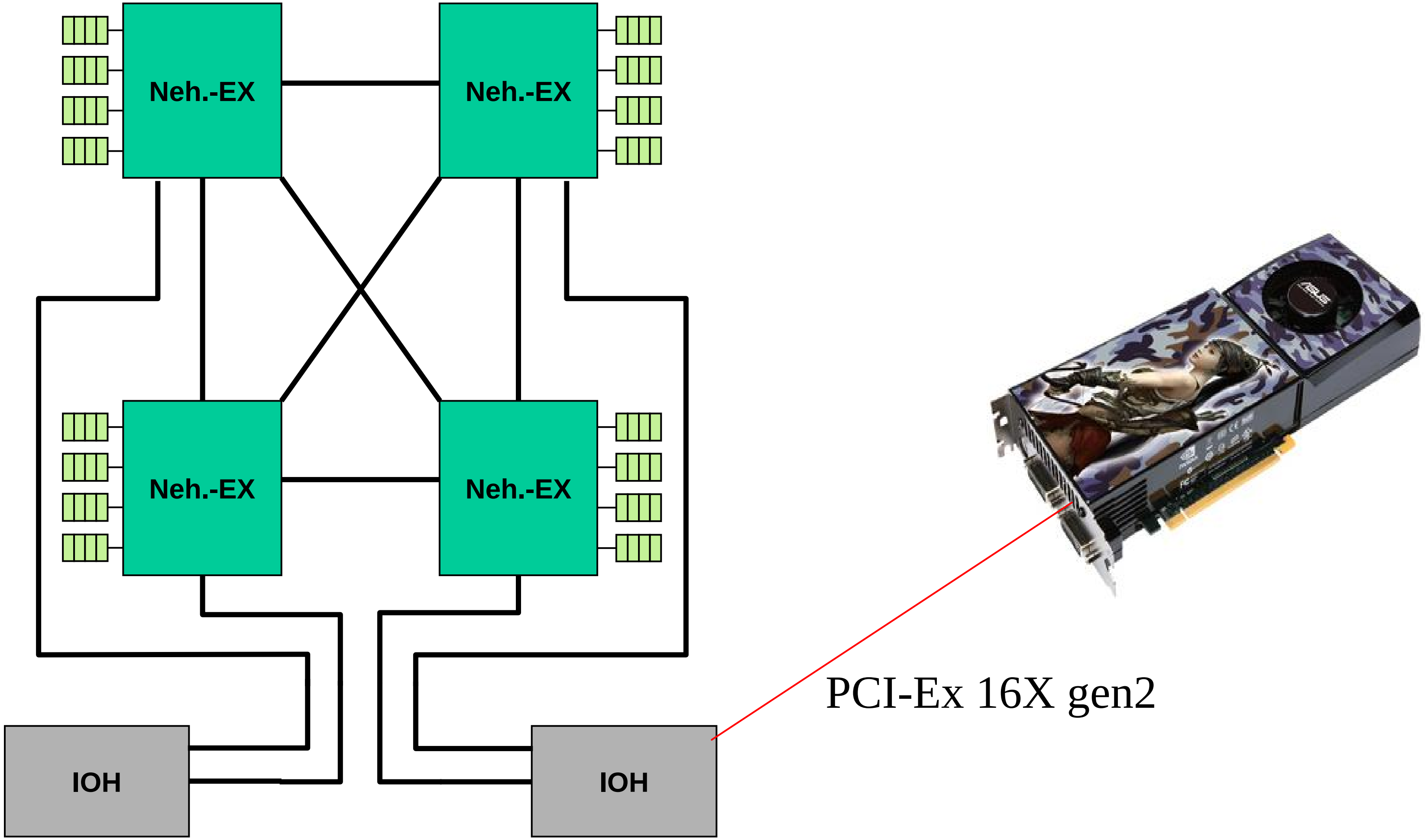
- Various test machines
  - As close to the user's network as possible
  - Each test machine
    - 2 Bull servers – 2 Haperton, 8GB
    - 2 **NVIDIA Tesla S1070**
    - IB DDR



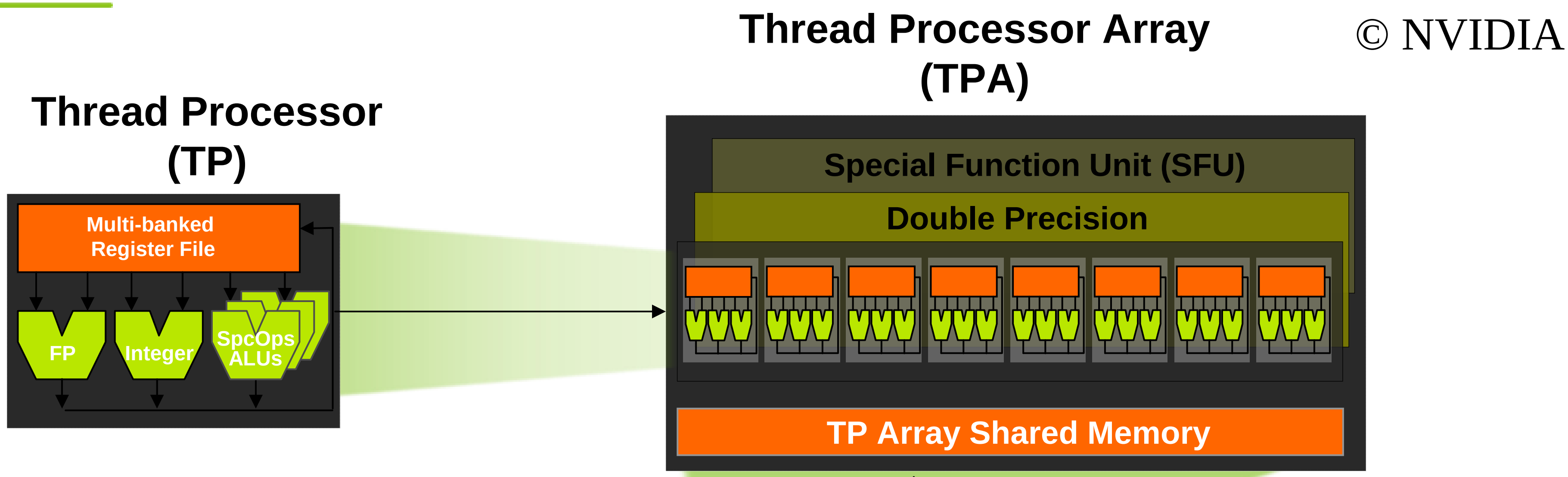
T10 credit NVIDIA

- CCRT Graphic cluster
  - 40 Quadro FX 5800 (8 cores Haperton, 64GB/128GB)
    - T10 based : will be used for GPGPU too.
- A new CCRT machine
  - Two partitions in 1068 nodes
    - Standard production : 972 nodes for 103TFlops
    - Hybrid partition : 96 servers + **48 NVIDIA Tesla** for GPU Computing **192 TFlops**
  - Servers = BULL Novascale R42x
    - Intel Nehalem-EP, 8GB, IB DDR

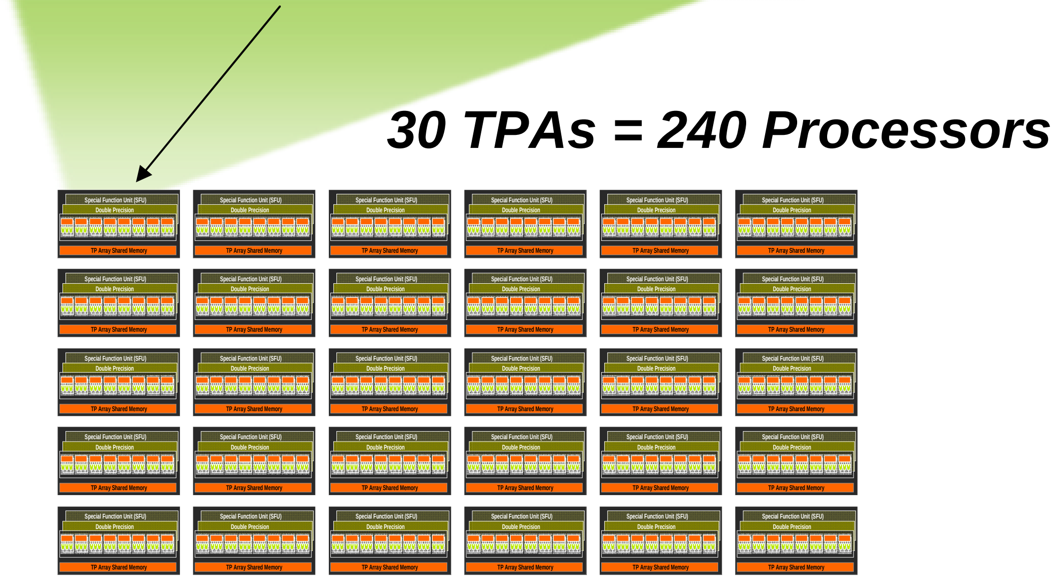
Hardware Architecture: node integration of a GPU



Tesla T10: The Processor Inside



- 240 thread processors
- Full scalar processor with integer and floating point units
- IEEE 754 floating point
  - *Single and Double*



Programming languages for GPUs (1/2)

- Cuda
  - For prototyping or “Kleenex” codes
  - For highly tuned libraries (if needed)
  - Status of FORTRAN ?

- OpenCL
  - The great unknown – for the portability
  - Should be tested on AMD/ATI cards at some points too
  - NO FORTRAN

- Rapidmind
  - Has to be experimented
  - NO FORTRAN
  - Seen as very intrusive

Programming languages for GPUs (2/2)

- HMPP
  - Our KEY solution for LEGACY codes
    - Ease of use for the FORTRAN community
    - Capitalize on our MILLIONS of line of code
  - Main advantages for CEA
    - FORTRAN, C
      - Java, C++ support soon
    - Multiple targets (NVIDIA, ATI, SSE, …)
    - Keeps the codes’ portability
    - Low learning curve
- First actions : Create a user base
  - Important to ease the change of paradigm
    - Parallelism is NOT a widespread knowledge
      - Massive parallelism is for some experts only
    - Spread the knowledge from within the teams

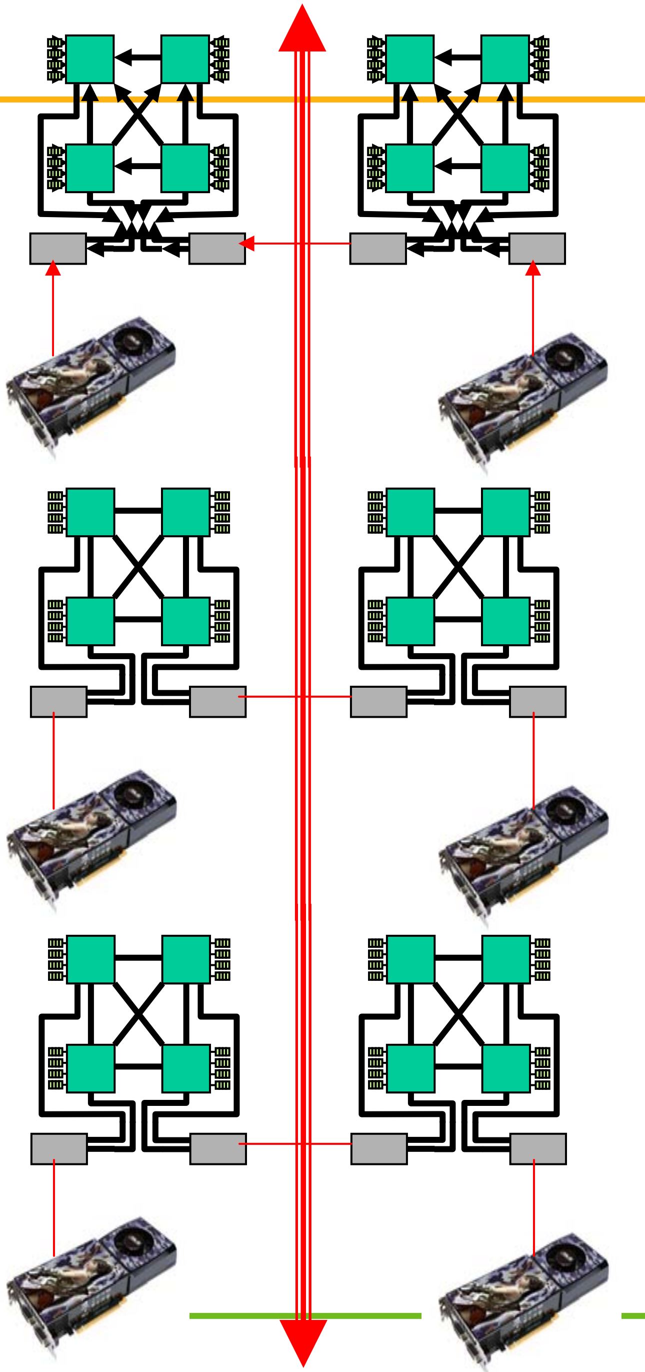
GPU: First results

- Libraries call (0D) can yield significant results
  - EOS on GPU => 3 fold speed up for the whole run

- Code architecture might not be suitable for GPU usage **as is**
  - First make sure that the code is REALLY optimized
  - Then make sure that the parallelization is well done
    - Reuse some of the vector programming habits
    - Reuse the study of the memory
      - Memory is far far away !
  - Then locate what can remain on the GPU as long as possible
  - Then rewrite some portions of the code
    - **`outvec[i] = outvec[i] + matvals[j] * invec[indx[j]]`**
      - **`Indirect addressing + reduction`**

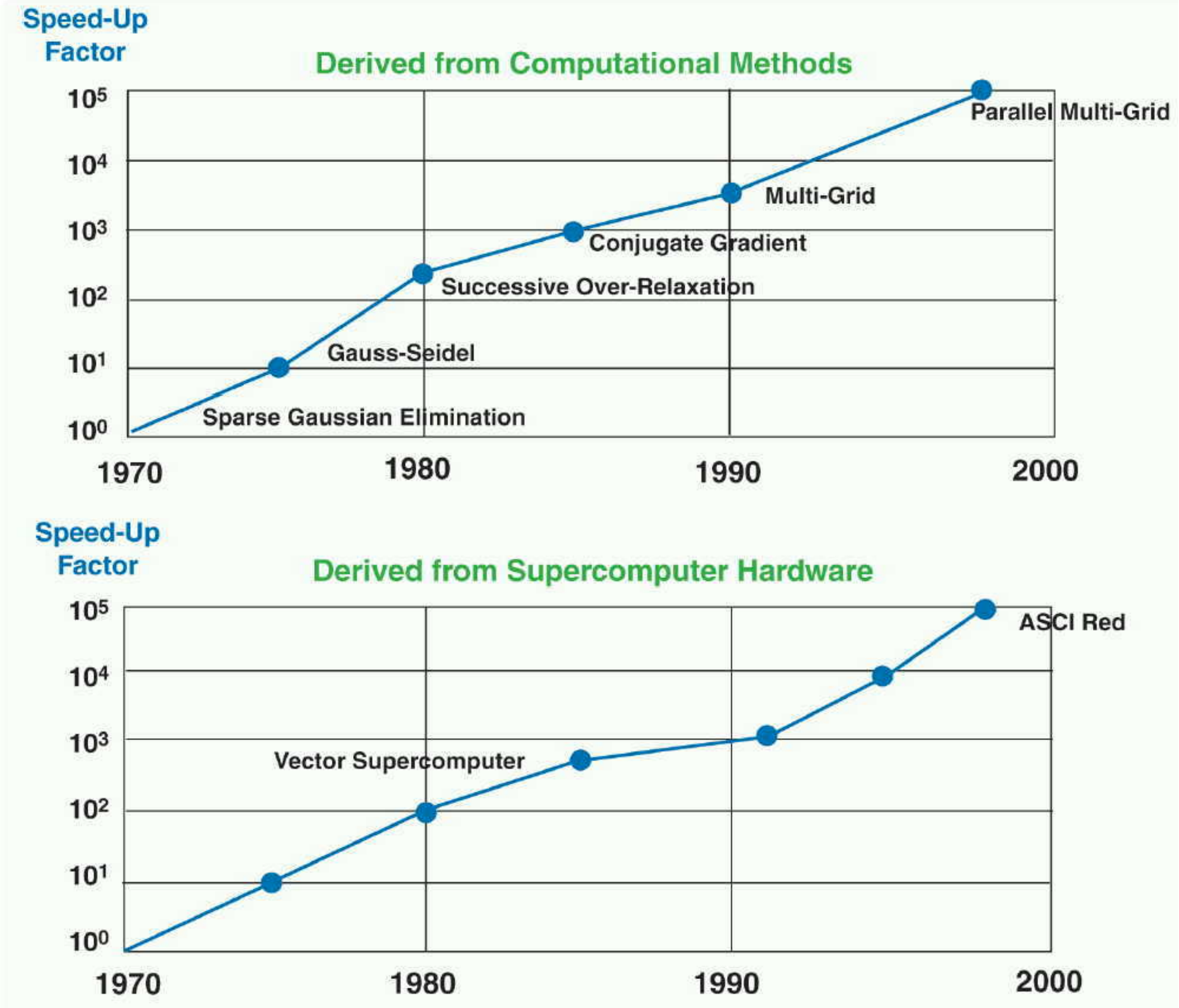
Everything together ?

- Adding Cuda to OpenMP + MPI is possible
  - One MPI task per node
  - One OpenMP thread per core
  - One thread specialized for Cuda
- HMPP allows for a mix of GPU Computing along with MPI
  - Hybrid programming is here
- Big increase in code complexity
  - importance of code architecture
  - KISS
- Keep the code readable and easy to maintain
- Optimize the algorithm is more efficient than modify the existing unoptimized code
  - Teams of experts : computer scientist + physicist + mathematician



Don’t forget algorithms ;-)

"Is There A Moore’s Law For Algorithms?"
David E. Womble
Sandia National Laboratories
Presented at Salishan
April 19, 2004



From SIAM Review, 2001

Other parallelism having an impact on applications

- I/O
  - MPI-I/O versus home made
  - Lustre
    - File system level
      - Hopefully transparent for the user

- Visualization
  - Tools must handle parallel outputs of codes
  - Recent tools use parallelism
    - MPI and multithread
  - Can use multiple displays or graphic cards
    - For performances
    - For higher resolution

Other issues with parallelism: debugging

- Multicore
  - Tools exist, a GUI helps a lot
    - Gdb, DDT, Totalview, …

- Clusters
  - MPI integration usually good
    - DDT, Totalview
  - Usability iffy with hundreds to thousands of tasks

- Manycore
  - NVIDIA just provided gdb for its hardware
  - Allinea and Totalview are working on providing at least basic features.

- General issue
  - How to reproduce unpredictable sequences/events

Other issues with parallelism: profiling

- Multicore
  - Tools exist, a GUI helps a lot
    - gprof, valgrind, …
- Clusters
  - MPI integration usually good
  - Usability iffy with hundreds to thousands of tasks
- Manycore
  - NVIDIA just provided a profiler for its hardware
    - Usability? To be experimented in depth
- General issue
  - How to reproduce unpredictable sequences/events

Recommendations

- Every application should be parallel as of now
- Unless special needs, start with MPI
  - Less risks in the long run on large configurations
- Try to think in terms of millions of tasks
  - Good for MPI as well as GPU usage
- Get ready for new hardware
  - Multicore and GPU
    - HMPP is our favorite solution
- Take the time to work at the algorithm level
- Don’t forget Amdahl’s law ☺

Questions?