Running 3D Finite-difference or Spectral-element Wave Propagation Codes Faster Using a GPU Cluster

Dimitri Komatitsch, Pieyre Le Loher and David Michéa
(Univ Pau, CNRS and INRIA Sud-Ouest Magique3D)

Gordon Erlebacher (Department of Scientific Computing, Florida State University, USA)

Dominik Göddeke (TU Dortmund, Germany)

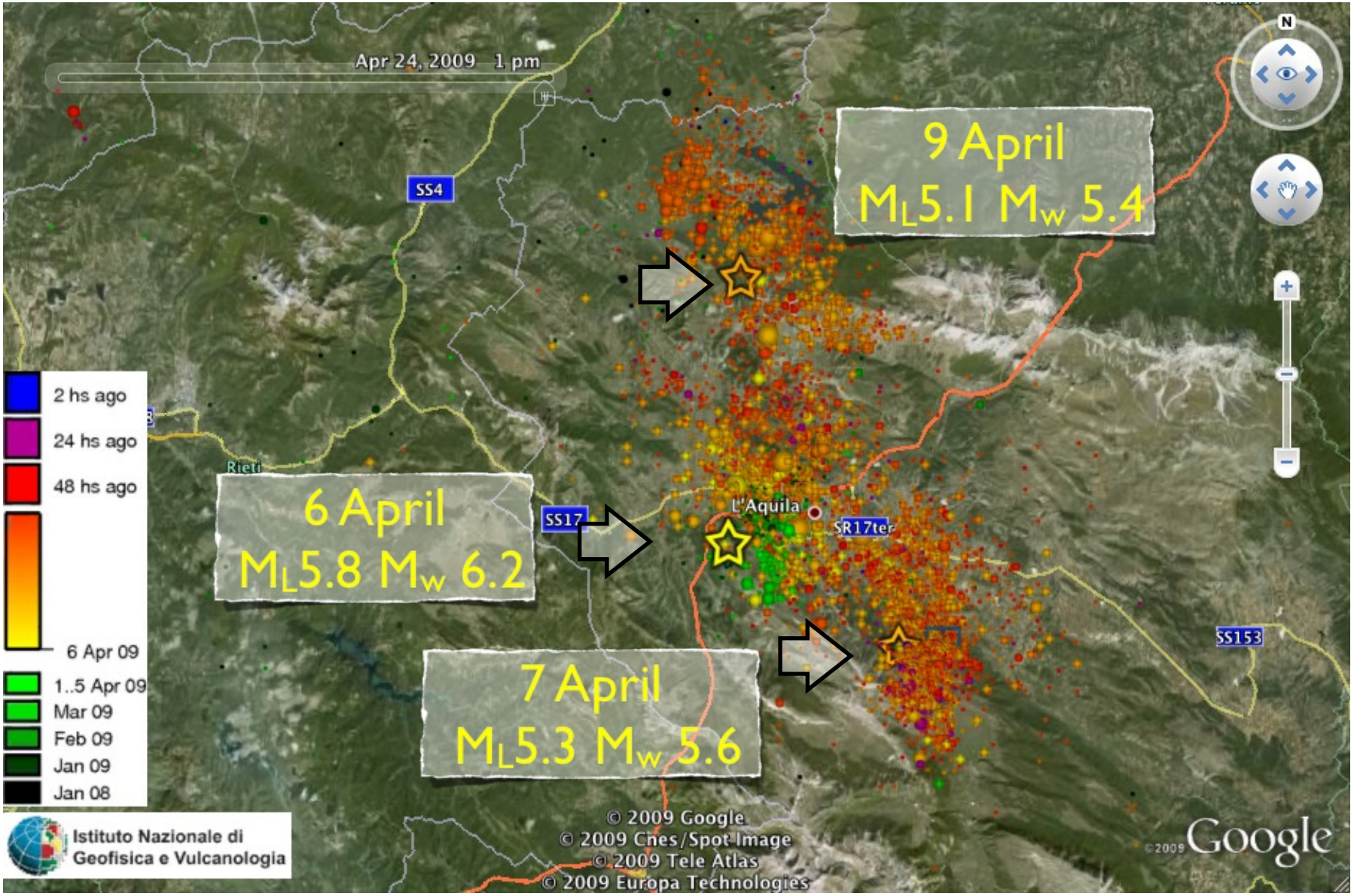
QUEST workshop
September 23, 2010

Outlines

- Why we are involved in HPC
- Porting on GPU a scientific code based upon :
  - The Spectral Element Method
  - Finite Differences

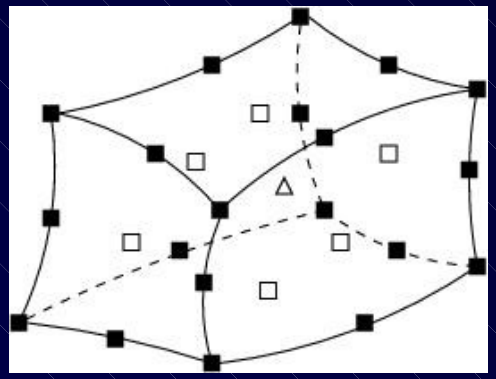
M_w 6.2 L'Aquila

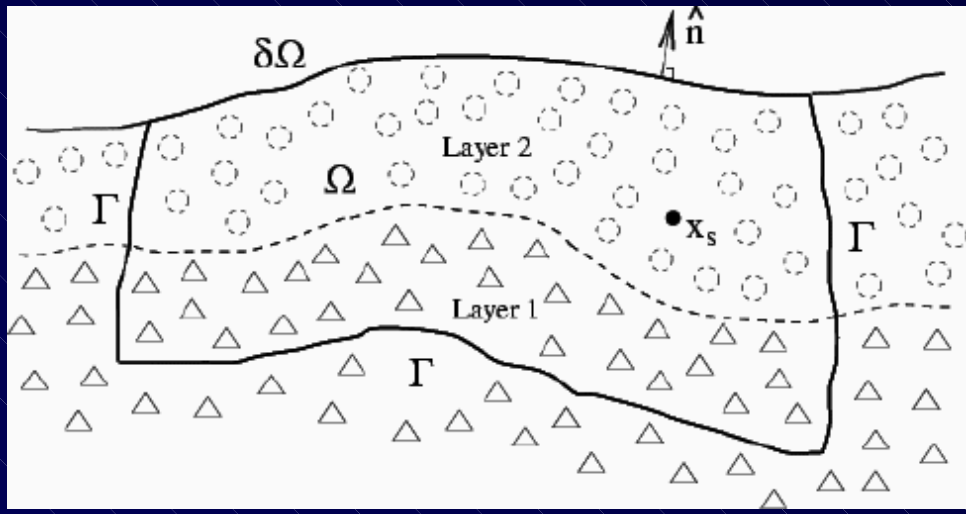
"Mainshocks and Aftershocks"



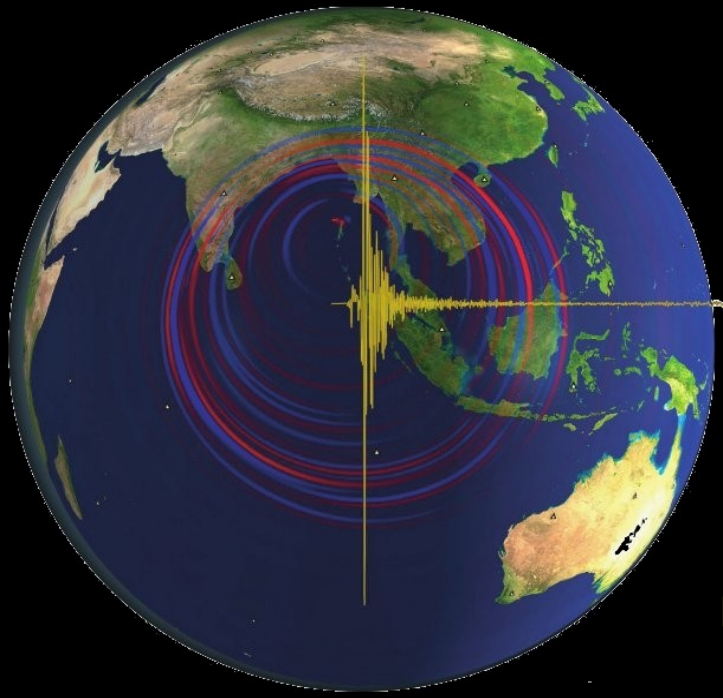
Spectral-Element Method

- Developed in Computational Fluid Dynamics (Patera 1984)
- Accuracy of a pseudospectral method, flexibility of a finite-element method
- Extended by Komatitsch and Tromp, Chaljub et al., Capdeville et al.
- Large curved “spectral” finite-elements with high-degree polynomial interpolation
- Mesh honors the main discontinuities (velocity, density) and topography
- Very efficient on parallel computers, no linear system to





Our SPECFEM3D software package



Dimitri Komatitsch
Jeroen Tromp
Qinya Liu
David Michéa

Min Chen
Vala Hjörleifsdóttir
Jesús Labarta
Nicolas Le Goff
Pieyre Le Loher
Alessia Maggi
Roland Martin
Daniel Peter
Brian Savage
Bernhard Schuberth
Carl Tape
...

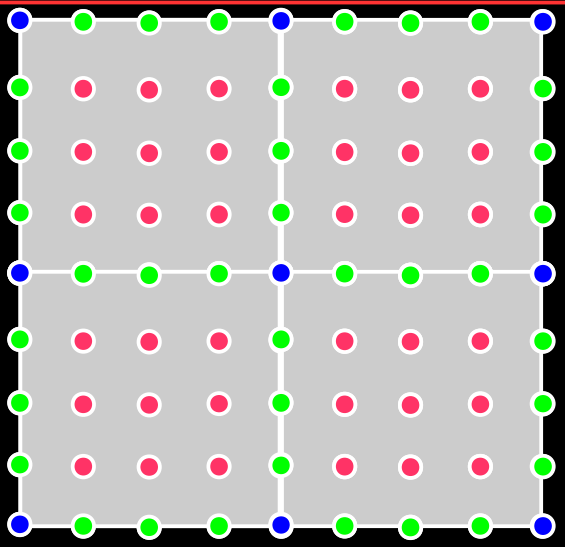
Goal: modeling seismic wave propagation in the full Earth or in densely populated regions following large earthquakes

The SPECFEM3D source code is open (GNU GPL v2)

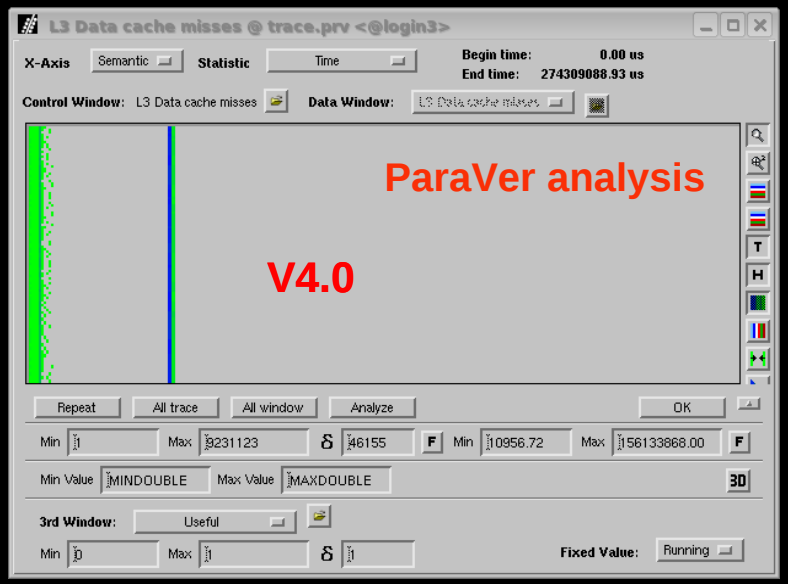
Mostly developed by Dimitri Komatitsch and Jeroen Tromp at Harvard University, Caltech and Princeton (USA) and University of Pau (France) since 1996.

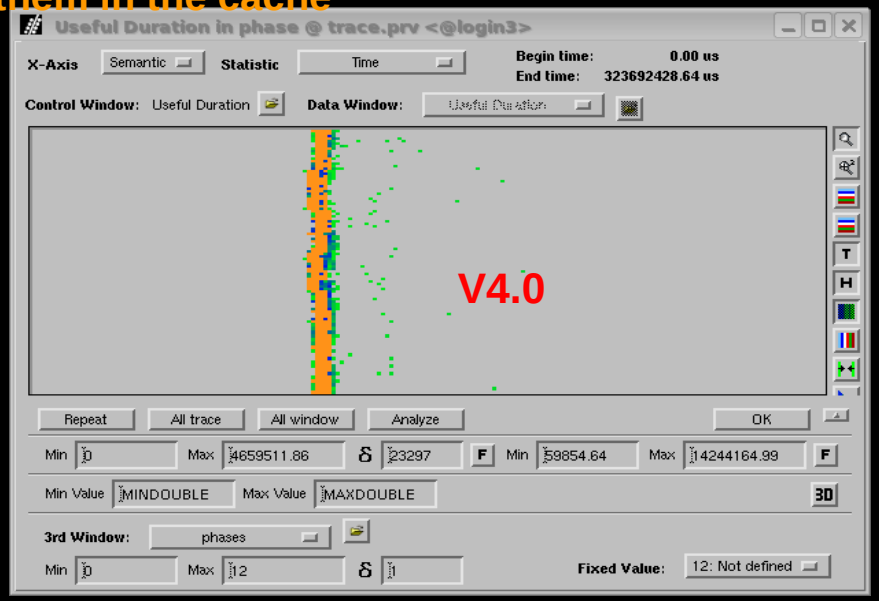
Improved with the Barcelona Supercomputing Center, Spain (Jesús Labarta et al.)
and David Michéa (INRIA, Pau, HPC-Europa program, 2007),

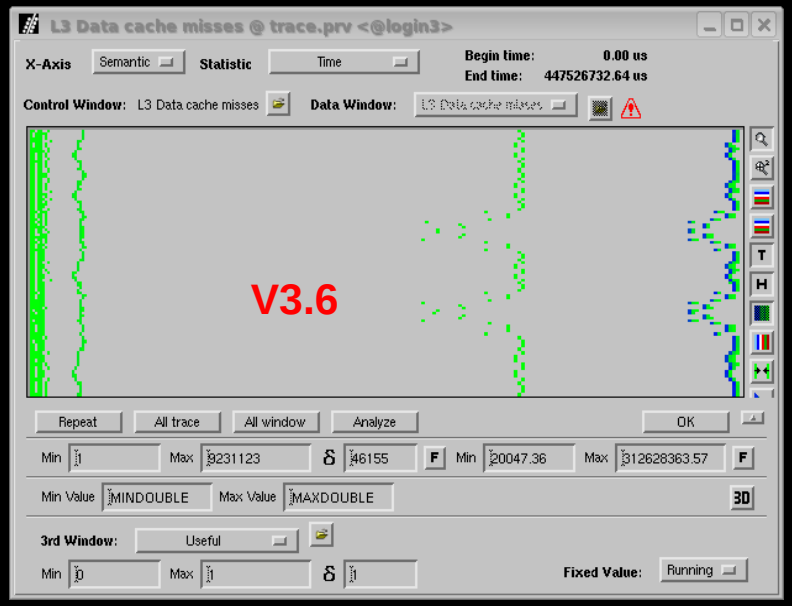
Results for load balancing: cache misses (J. Labarta, BSC)



=> it is crucial to reuse common points by keeping
them in the cache

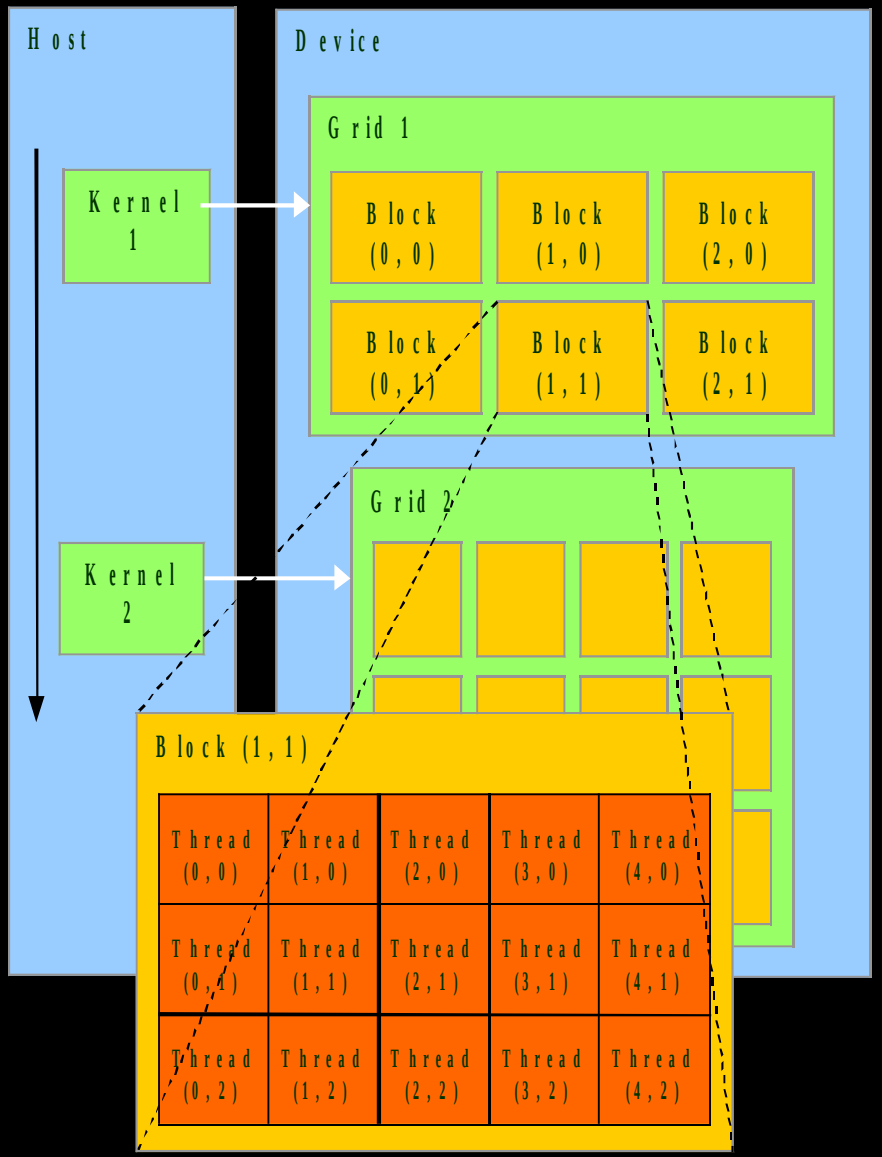






Graphics cards

NVIDIA GeForce 8800 GTX



GPU = Graphics Processing Unit

Minimize CPU ↔ GPU data transfers

- CPU ↔ GPU memory bandwidth much lower than GPU memory bandwidth
  - Use page-locked host memory (`cudaMallocHost()`) for maximum CPU ↔ GPU bandwidth

- Minimize CPU ↔ GPU data transfers by moving more code from CPU to GPU
  - Even if that means running kernels with low parallelism computations
  - Intermediate data structures can be allocated, operated on, and deallocated without ever copying them to CPU memory

- Group data transfers
  - One large transfer much better than many small ones

- Fit all the arrays on the GPU card to avoid costly CPU ↔ GPU data transfers

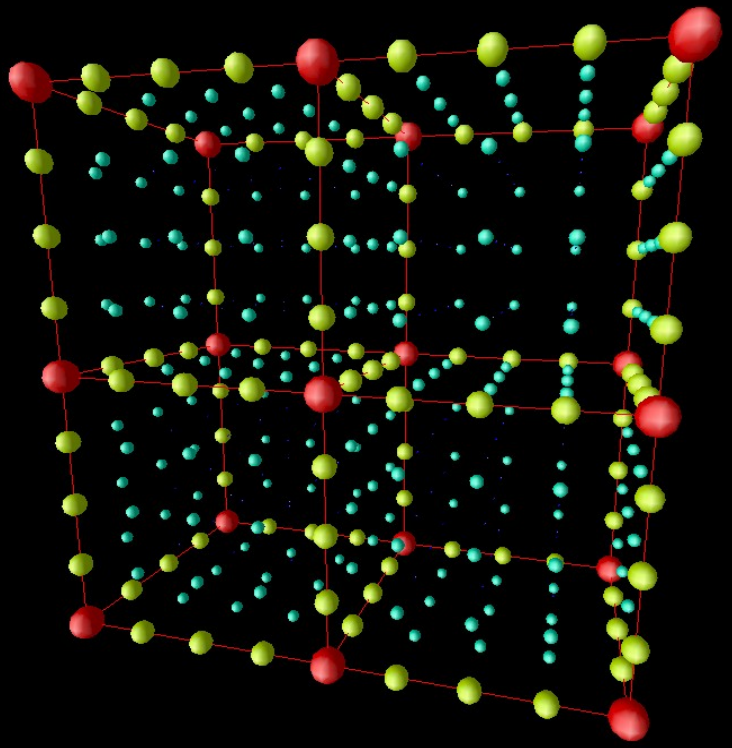
- But of course the MPI buffers must remain on the CPU, therefore we can not avoid a small number of transfers (of 2D cut planes)

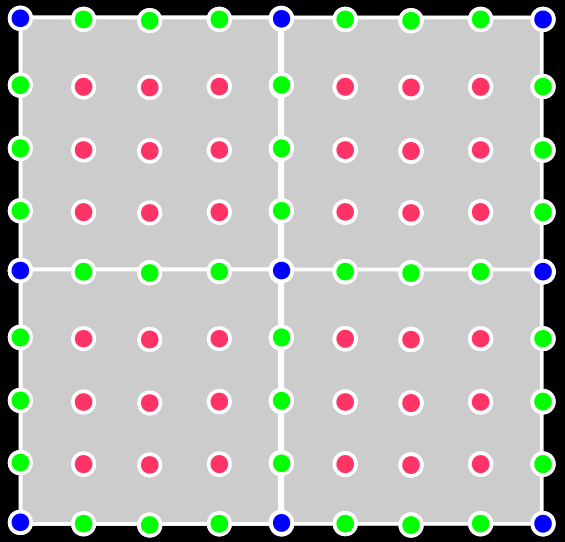
Porting SPECFEM3D on CUDA

- **At each iteration of the serial time loop, three main types of operations are performed:**
  - **update (with no dependency) of some global arrays composed of the unique points of the mesh**
  - **purely local calculations of the product of predefined derivative matrices with a local copy of the displacement vector along cut planes in the three directions (i, j and k) of a 3D spectral element**
  - **update (with no dependency) of other global arrays composed of the unique points of the mesh**

Porting SPECFEM3D on CUDA: global numbering versus local numbering

- In 3D and for NGLL = 5, for a regular hexahedral mesh there are:
  - 125 GLL integration points in each element
  - 27 belong only to this element
  - 98 belong to several elements

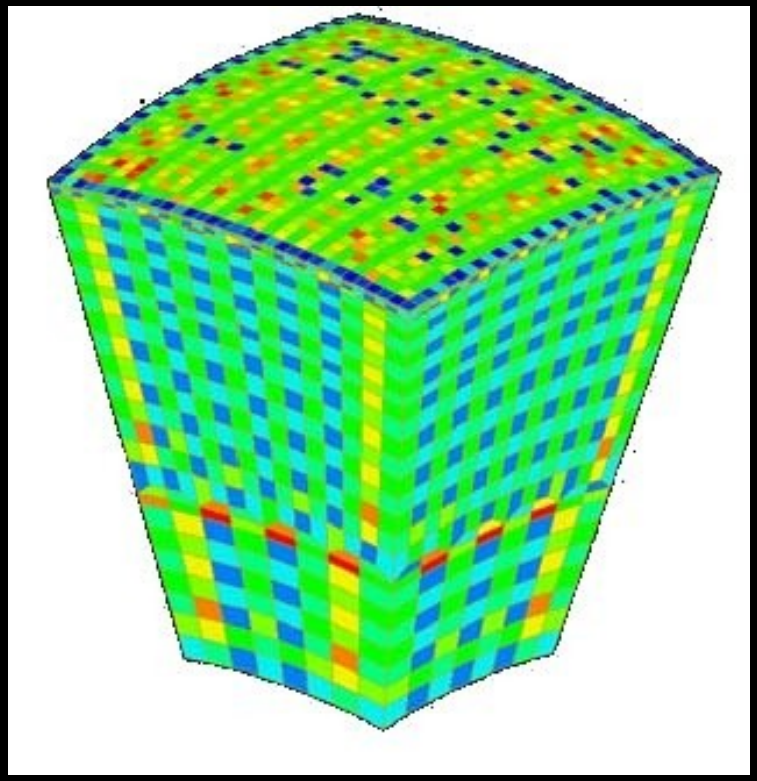




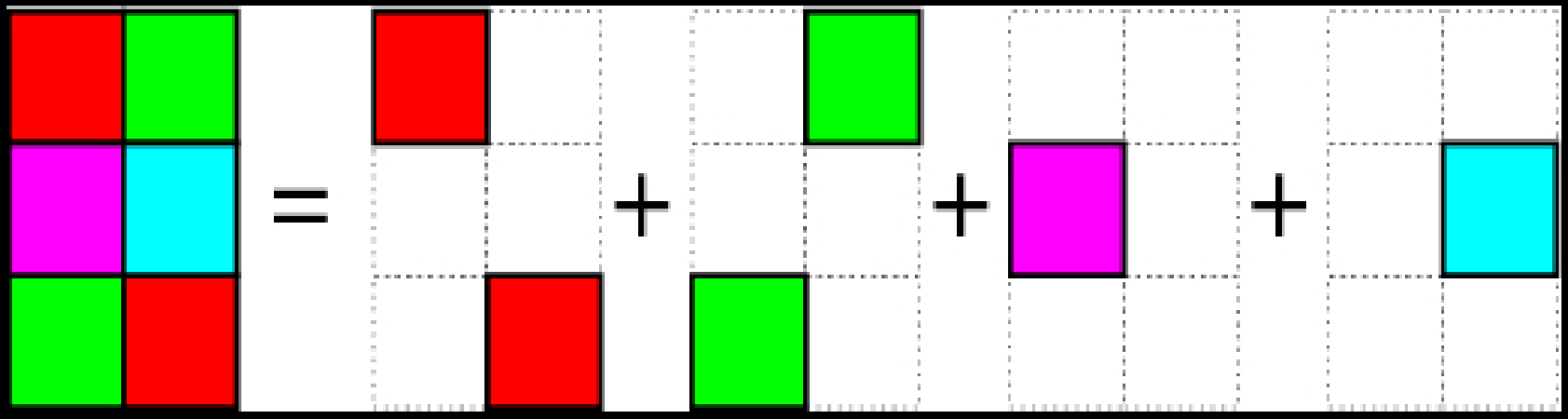
=> one thread per grid point (i.e., 125 threads per finite element)

Porting SPECFEM3D on CUDA: mesh coloring

- Key challenge: ensure that contributions from two local nodes never update the same global value from different warps

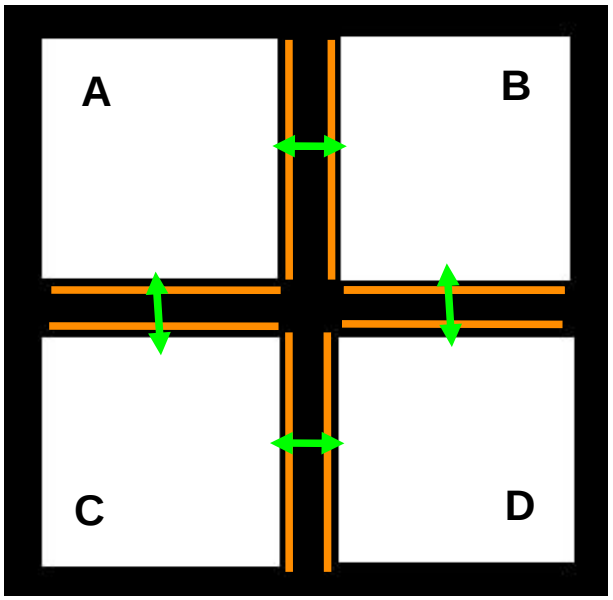


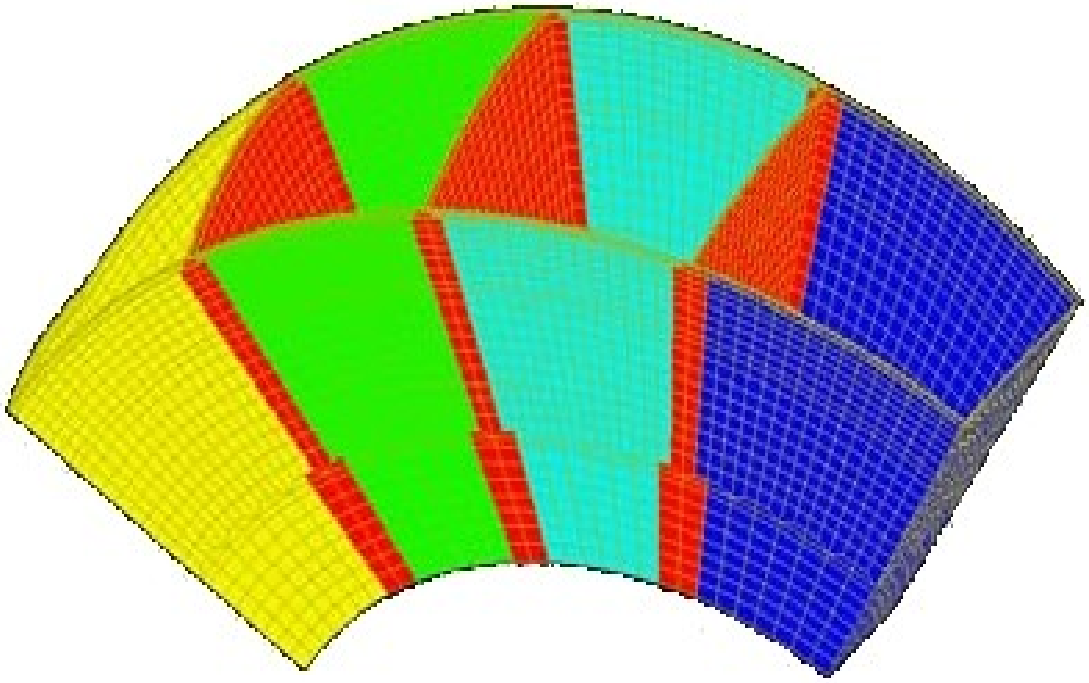
- Use of mesh coloring: suppress dependencies between mesh points inside a given kernel



Porting SPECFEM3D on CUDA: adding MPI

Non blocking MPI: update done in communication buffers (for outer mesh elements first)





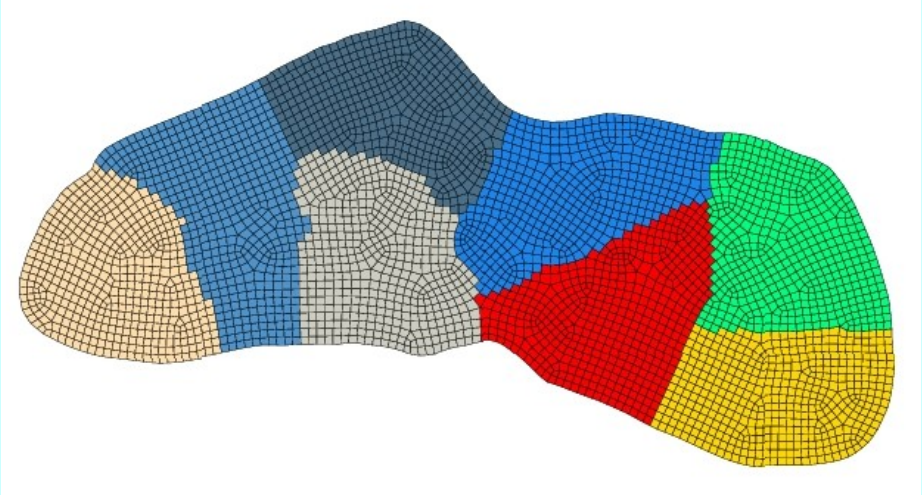
MPI communications cost on GPU version ~ 5%,

> We need to use non-blocking MPI communications.

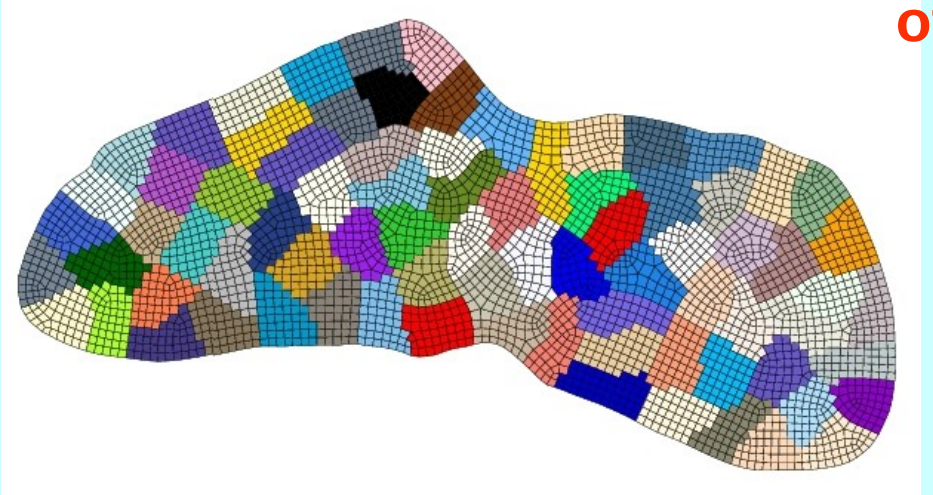
> MPI communications are very well overlapped by computations on the GPU.

Use non-blocking MPI

Classical overlapping, see e.g. Danielson and Namburu (1998)



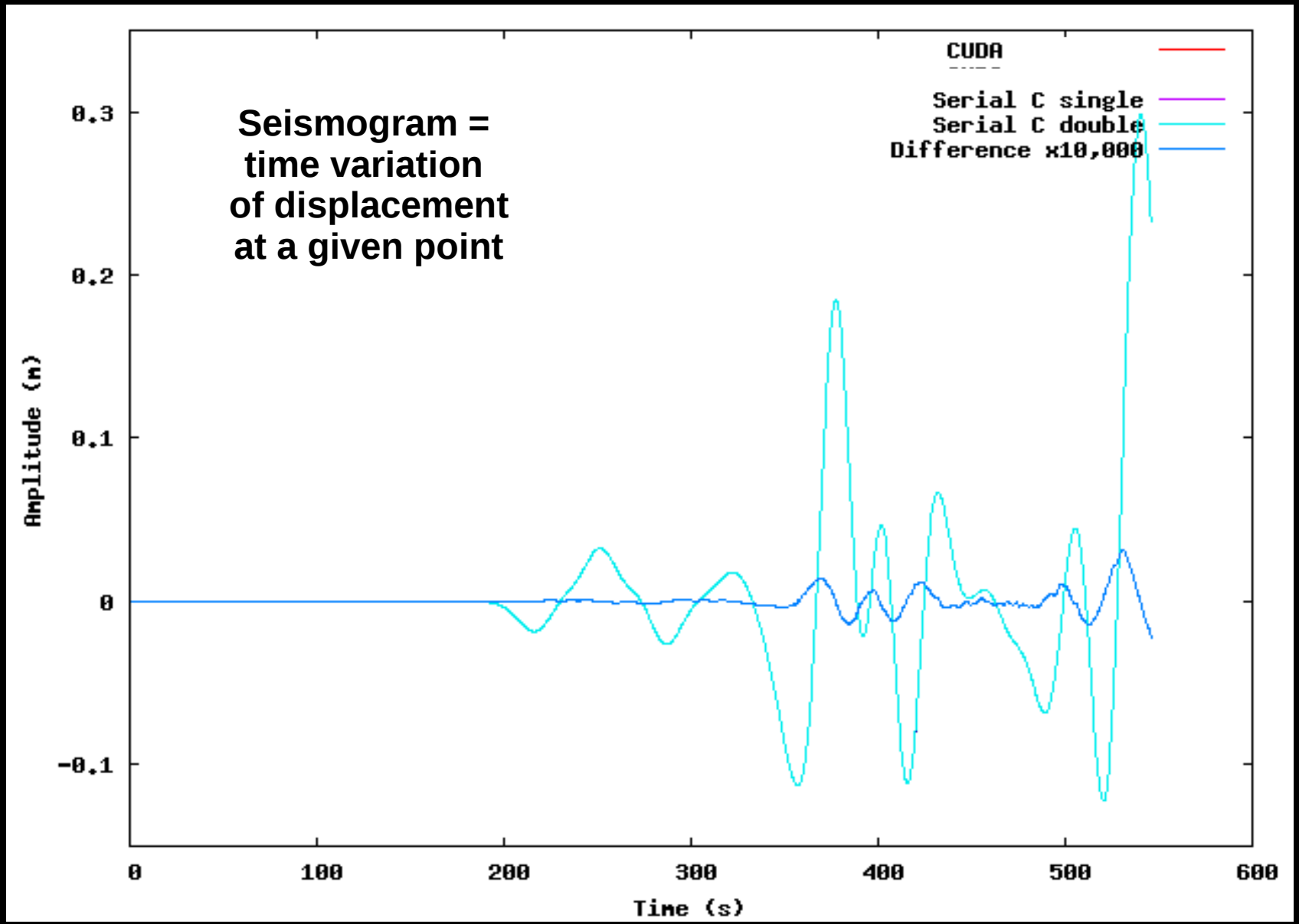
Works fine only if the interior of each domain is big enough.



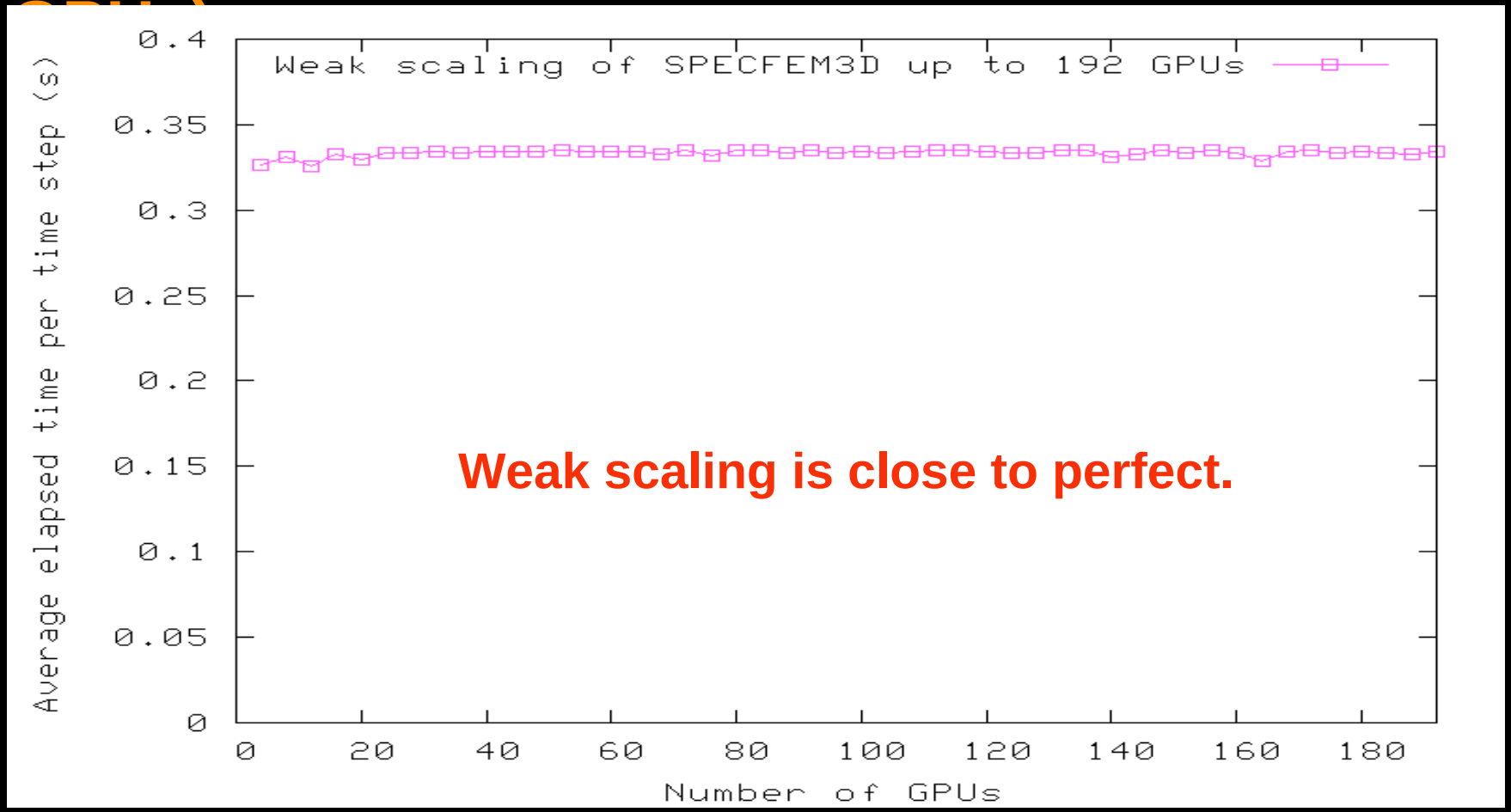
Collaboration with Roland Martin and Nicolas Le Goff (Univ of Pau, France)

Porting SPECFEM3D on CUDA: validation

- Validation and single precision efficiency:



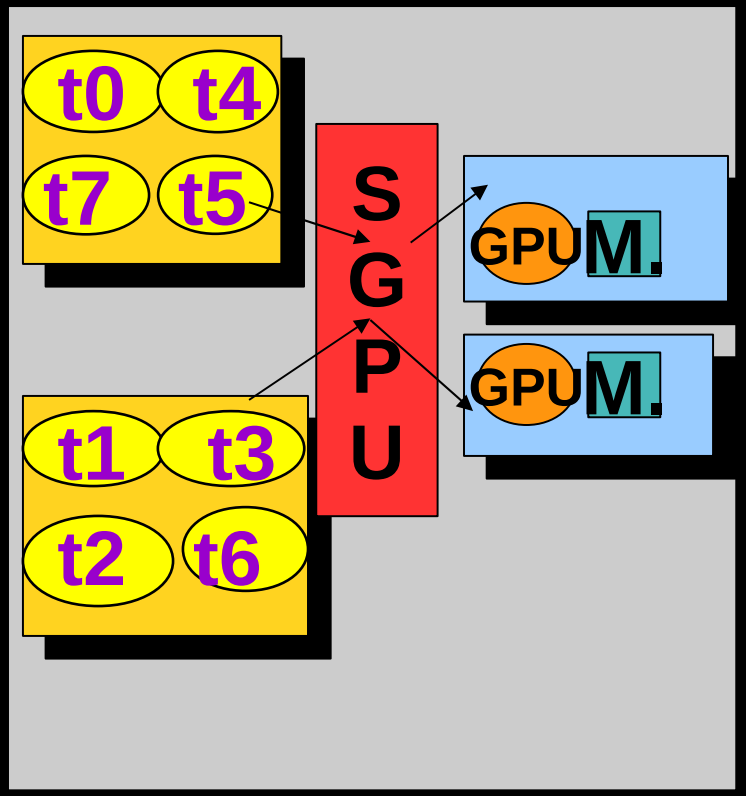
Multi-GPU weak scaling (up to 192 GPUs)



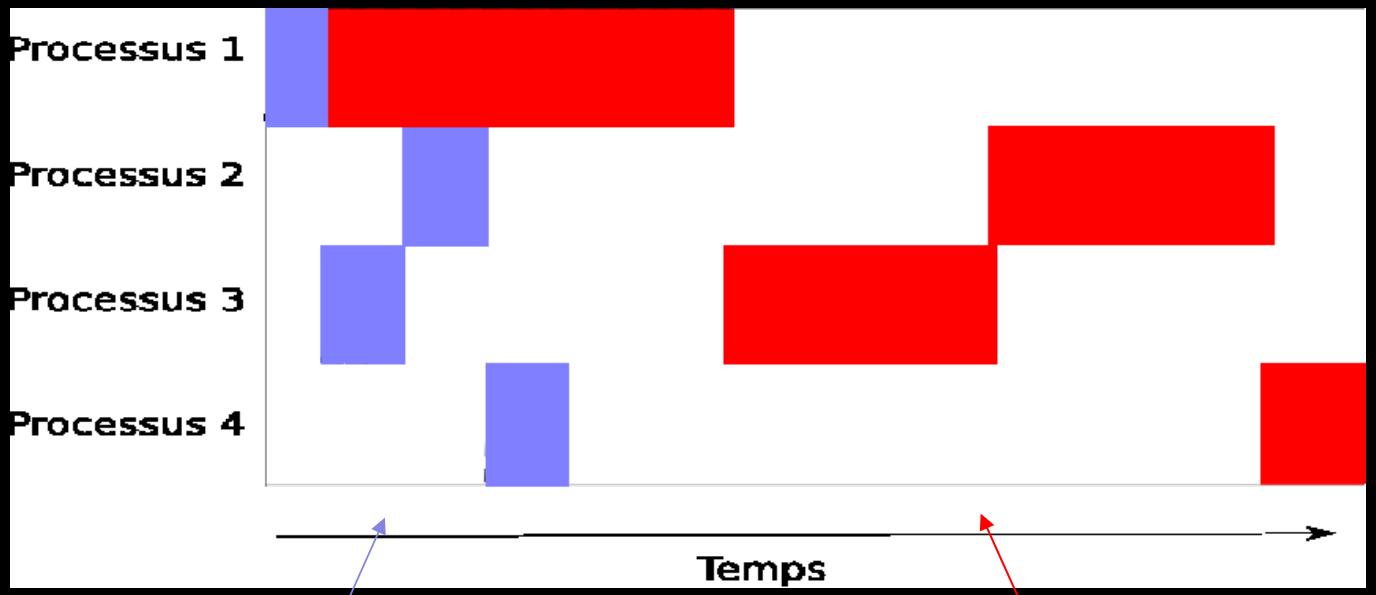
- It is difficult to define speedup: versus what?
- For us, on the CEA/CCRT/GENCI GPU/Nehalem cluster, about 20x for one GPU versus one CPU core.

The S_GPU library

- Implemented by Matthieu Ospici, Jérôme Reybert, Jean-François Méhaut (INRIA Grenôble – MESCAL)
- Virtualization : 1 GPU visible per CPU core
- Instructions scheduling done by S_GPU, not by CUDA
- Memory transfers / computations overlapping
- Written in C++ and CUDA, binding Fortran
- Limited intrusion in the source code



S_GPU : running traces



Memory transfer CPU - GPU

Computation (GPU)

So far, up to 20 % faster than CUDA

(CUDA 3.1 , Tesla T10)

Porting the FD code ONDES3D to CUDA (done with David Michéa from BRGM, France).

Implementing Finite-Difference code on GPU :

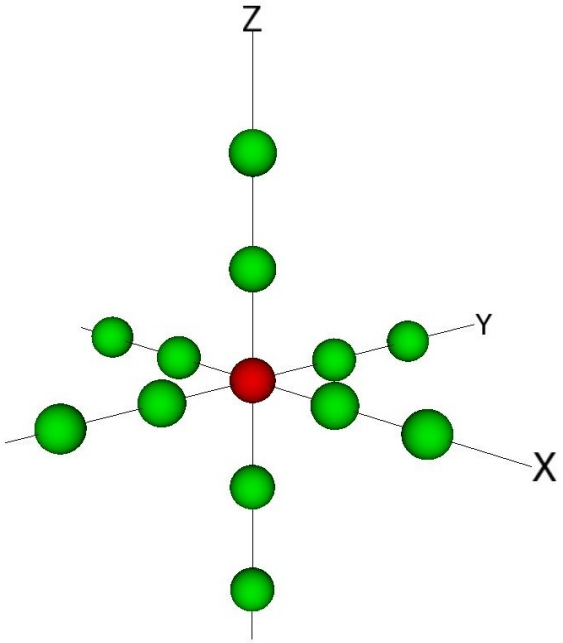
> Computation stencil leads to high reading redundancy

> Necessity to massively use shared memory

> Reduce the ratio halo_points / block_points to reduce redundancy

> Intuitive approach : 3D blocks that minimize the ratio halo/points

> problem: not enough shared memory to have 3D blocks big enough to get a good ratio.



=> 13 global memory accesses / thread for each iteration

Porting the FD code ONDES3D to CUDA

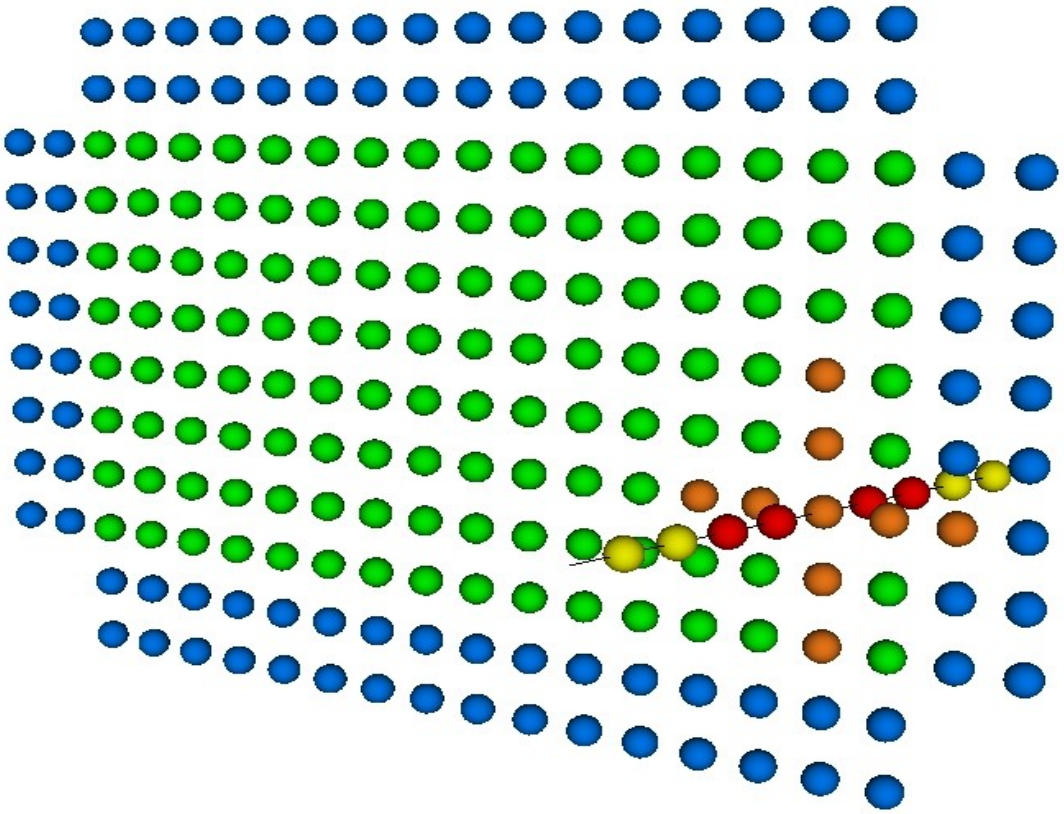
The approach of P. Micikevicius from NVIDIA (2009):
a sliding window algorithm

> use 2D tile instead of 3D blocks and loop along z axis

> in 2D, the amount of shared memory is sufficient to have tiles big enough to obtain a good ratio between halo points and computed points.

> data along z axis are shifted in registers at each iteration in a pipeline way => except for halos, only one global memory data need to be loaded at each iteration for a point.

Porting the FD code ONDE3D to CUDA

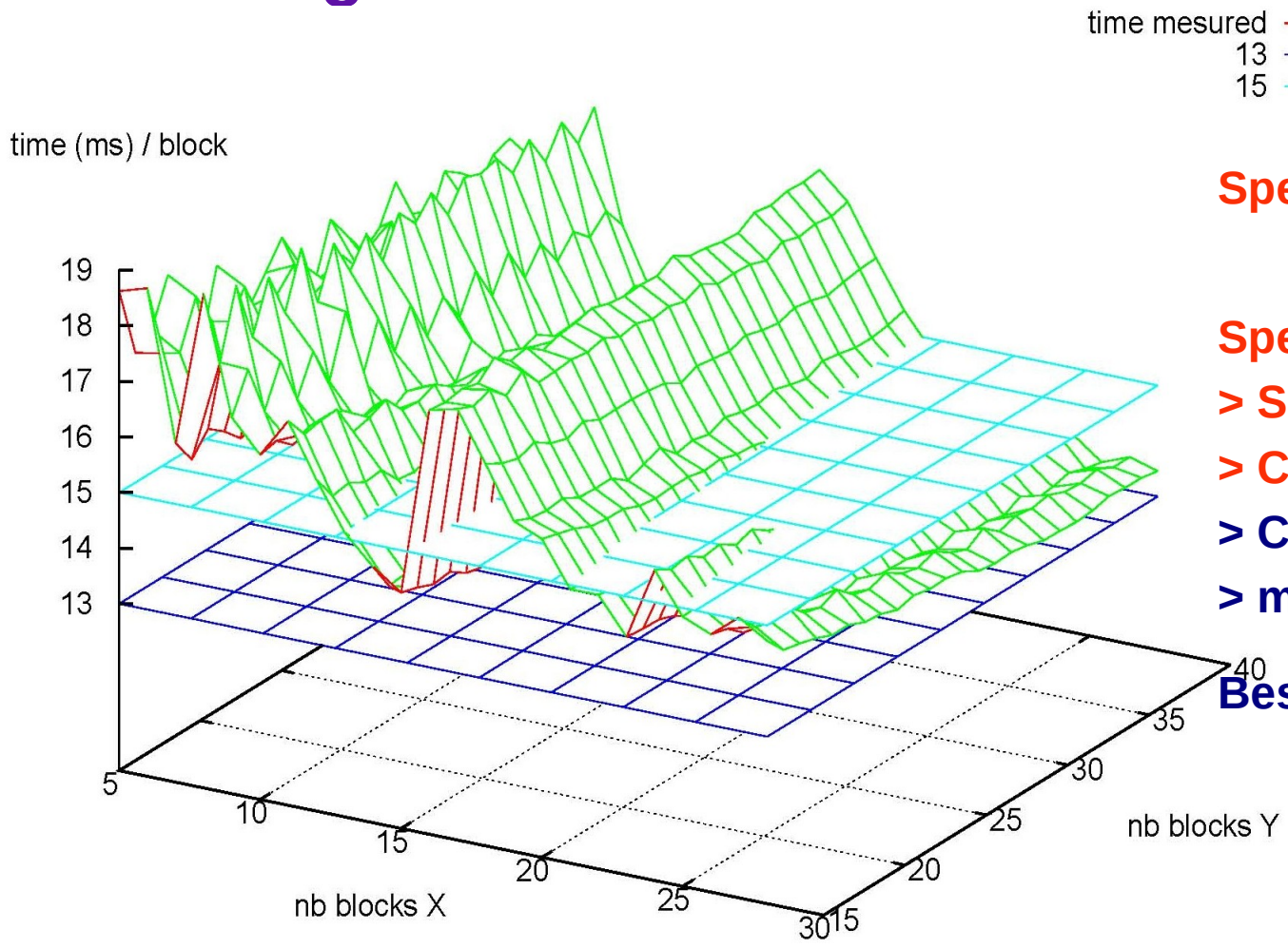


We use one thread per point in the *xy* plane and we tile it with blocks of 16x8 threads.

For a block : 4*16 + 4*8 + 16*8 data loaded by 16*8 threads

=> 1.75 global memory accesses / thread for each iteration (compared to 13 accesses before, i.e., 7.5 times fewer)

Porting the FD code ONDE3D to CUDA



Speedup : from 20x to 60x

Speedup depends on :

> Speedup definition!

> Compilers

> C-PML thickness

> model size

Best performance is obtained for C-PML thickness = 16 and model width multiple of 16.

- **Scaling along Z is almost linear.**
- **Despite the coalesced memory accesses, scaling along X is not regular and suffers from pathological cases**

Conclusions and future work

Dimitri Komatitsch, David Michéa and Gordon Erlebacher, Porting a high-order finite-element earthquake modeling application to NVIDIA graphics cards using CUDA, Journal of Parallel and Distributed Computing, vol. 69(5), p. 451-460, doi: 10.1016/j.jpdc.2009.01.006 (2009).

David Michéa and Dimitri Komatitsch, Accelerating a 3D finite-difference wave propagation code using GPU graphics cards, Geophysical Journal International, vol. 182(1), p. 389-402, doi: 10.1111/j.1365-246X.2010.04616.x, (2010).

- We get a speedup of 20x on a cluster of GPUs for spectral elements with non-blocking MPI, and 20x to 60x in the case of finite differences
- It is crucial to use larger domains to compensate for the very high speedup
- In future work, we could use OpenCL (to support other hardware)
- Need for some kind of OpenMP for GPUs: CAPS HMPP, StarSs, StarPU...

Thank you
for your attention

SLIDES FOR QUESTIONS

methods

Seismic wave equation : tremendous increase of computational power

⇒ development of numerical methods for accurate calculation of synthetic seismograms in complex 3D geological models has been a continuous effort in last 30 years.

Finite-difference methods : Yee 1966, Chorin 1968, Alterman and Karal 1968, Madariaga 1976, Virieux 1986, Moczo et al, Olsen et al..., difficult for boundary conditions, surface waves, topography, full Earth

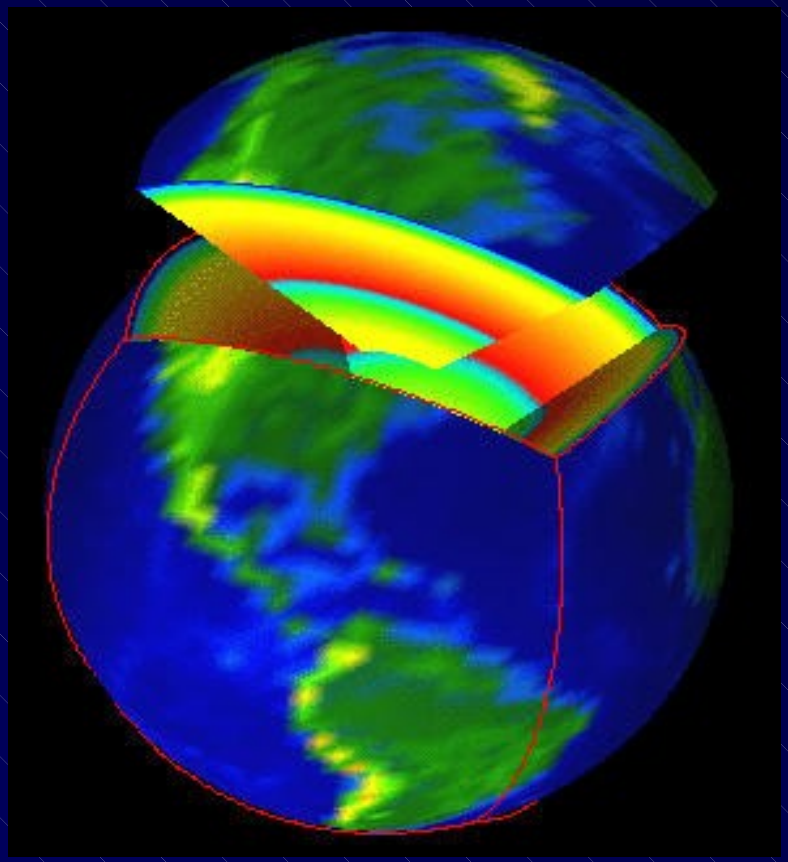
Boundary-element or boundary-integral methods (Kawase 1988, Sanchez-Sesma et al. 1991) : homogeneous layers, expensive in 3D

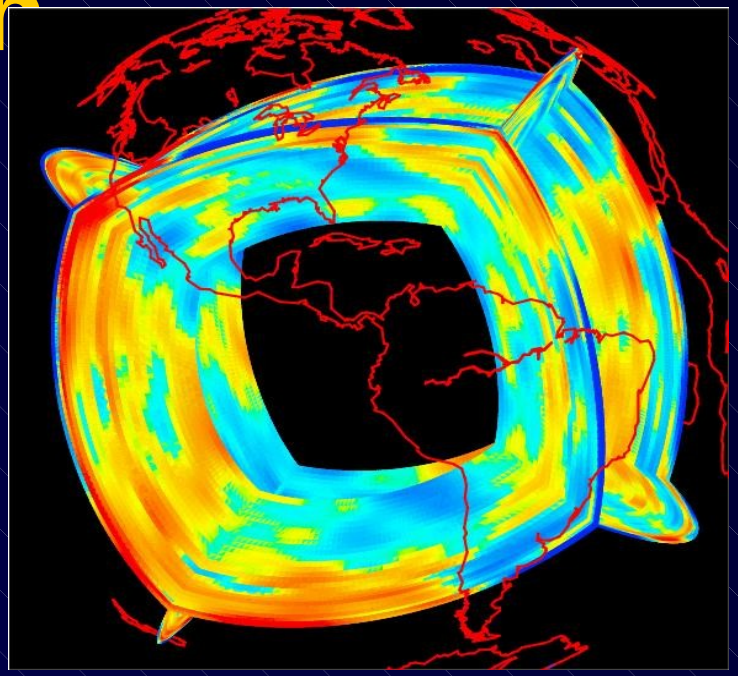
Spectral and pseudo-spectral methods (Carcione 1990) : smooth media, difficult for boundary conditions, difficult on parallel computers

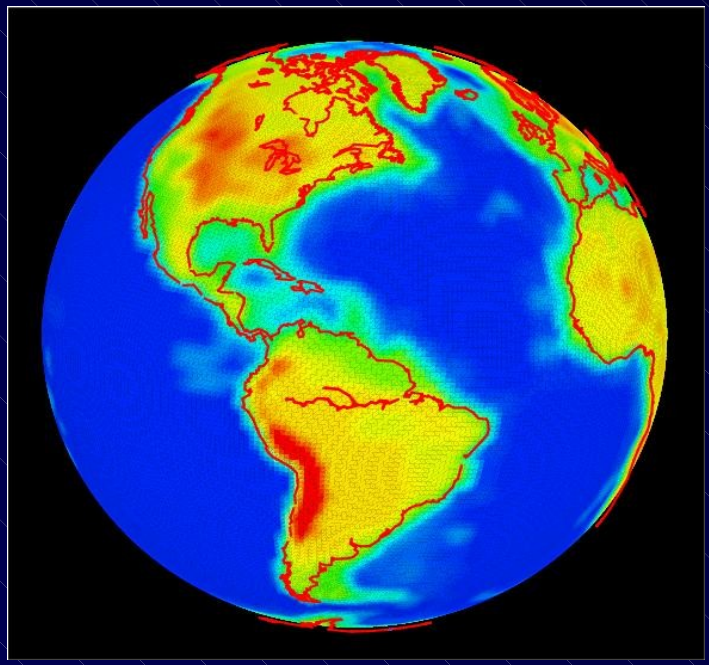
Classical finite-element methods (Lysmer and Drake 1972, Marfurt 1984, Bielak et al 1998) : linear systems, large amount of numerical dispersion

Let us combine the advantages of the last two techniques.

Final mesh of the Earth

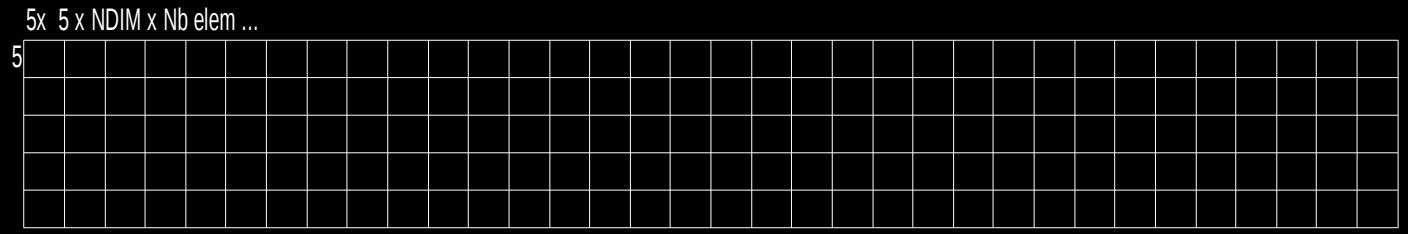


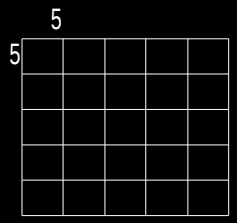




- “Gnomonic” mapping (Sadourny 1972)
- Ronchi et al. (1996), Chaljub (2000)
- Analytical mapping from six faces of cube to unit sphere

BLAS 3 (Basic Linear Algebra Subroutines)





Can we use highly optimized BLAS matrix/matrix products (90% of computations)?

- **For one element: matrices (5x25, 25x5, 5 x matrices of (5x5)), BLAS is not efficient: overhead is too expensive for matrices smaller than 20 to 30 square.**
- **If we build big matrices by appending several elements, we have to build 3 matrices, each having a main direction (x,y,z), which causes a lot of cache misses due to the global access because the elements are taken in different orders, thus destroying spatial locality.**
- **Since all arrays are static, the compiler already produces a very well optimized code.**

=> No need to, and cannot easily use BLAS

=> Compiler already does an excellent job for small static loops

Global Simulations: SPECFEM3D_GLOBE

Open source: geodynamics.org

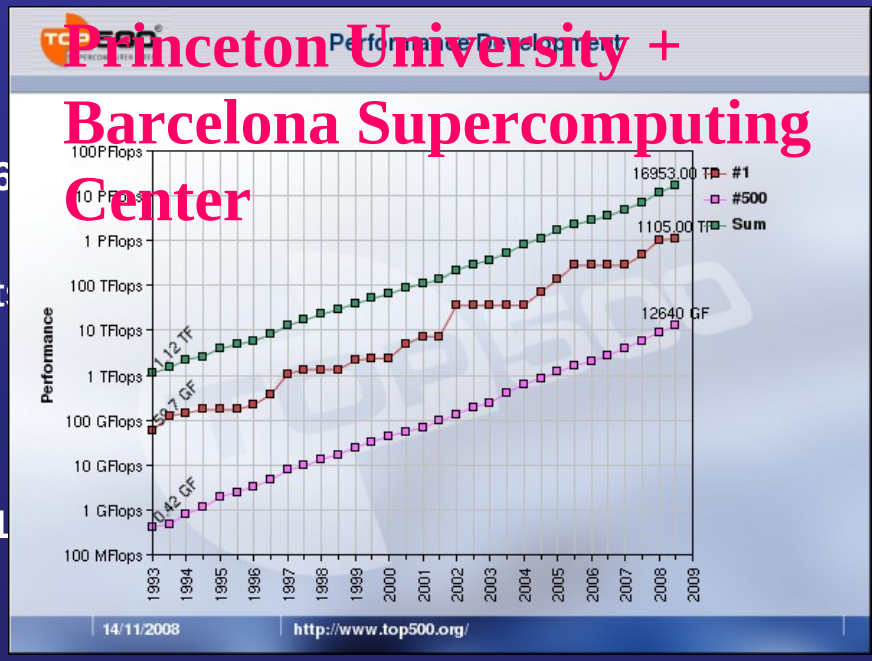
On-demand TeraGrid applications:

- Automated, near real-time simulations of all M>6 earthquakes
- Analysis of past events (more than 20,000 event
- Seismology Web Portal (geodynamics.org)

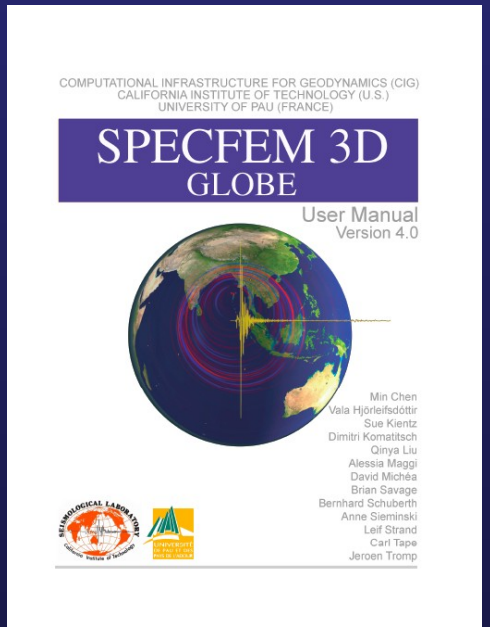
Petascale simulations:

- Global simulations at 1-2 Hz
- Reached 1.15 s period on 149,784 cores at ORNL
- Moving towards global 'adjoint tomography'

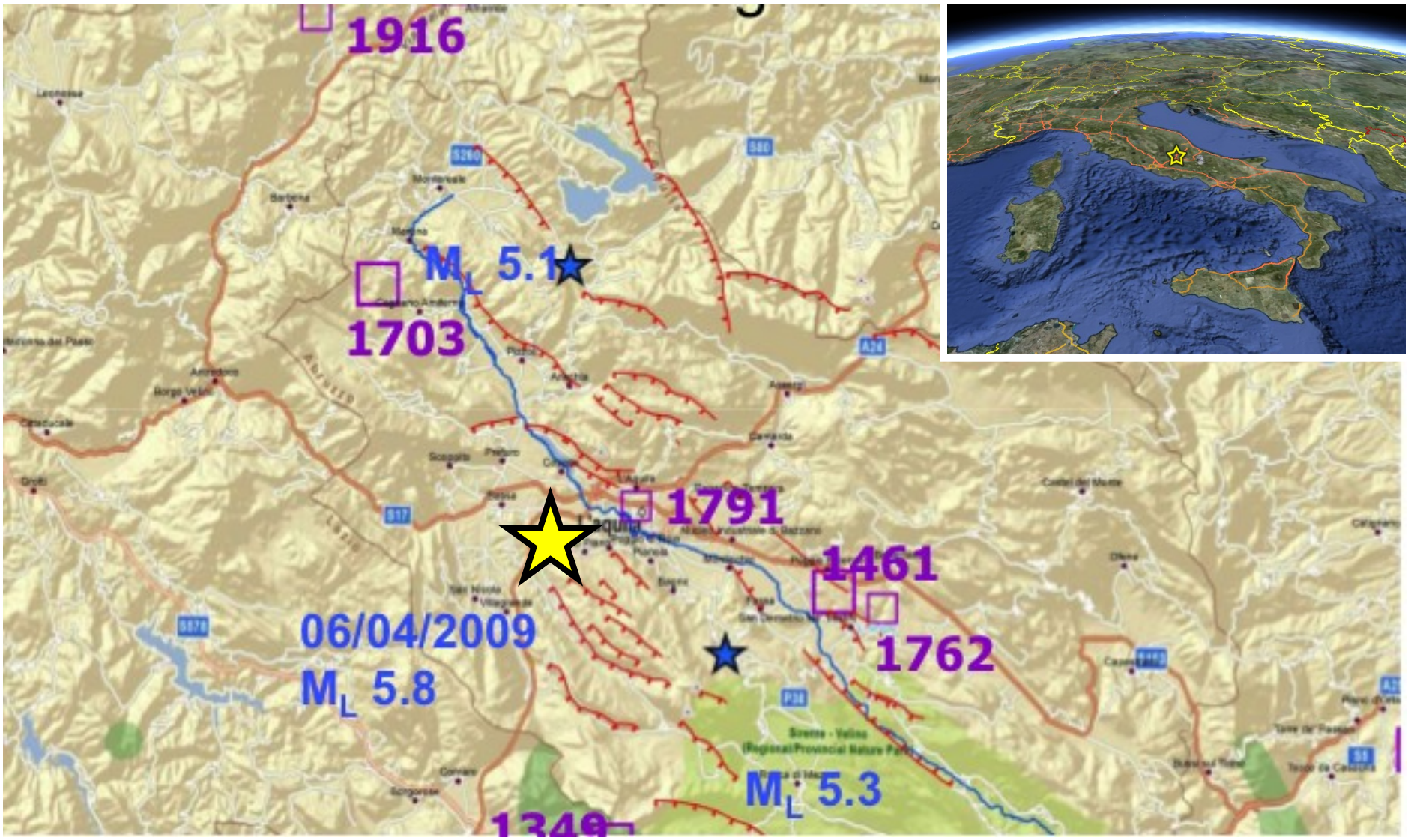
Princeton University + Barcelona Supercomputing Center



SPECFEM3D_GLOBE Users Map

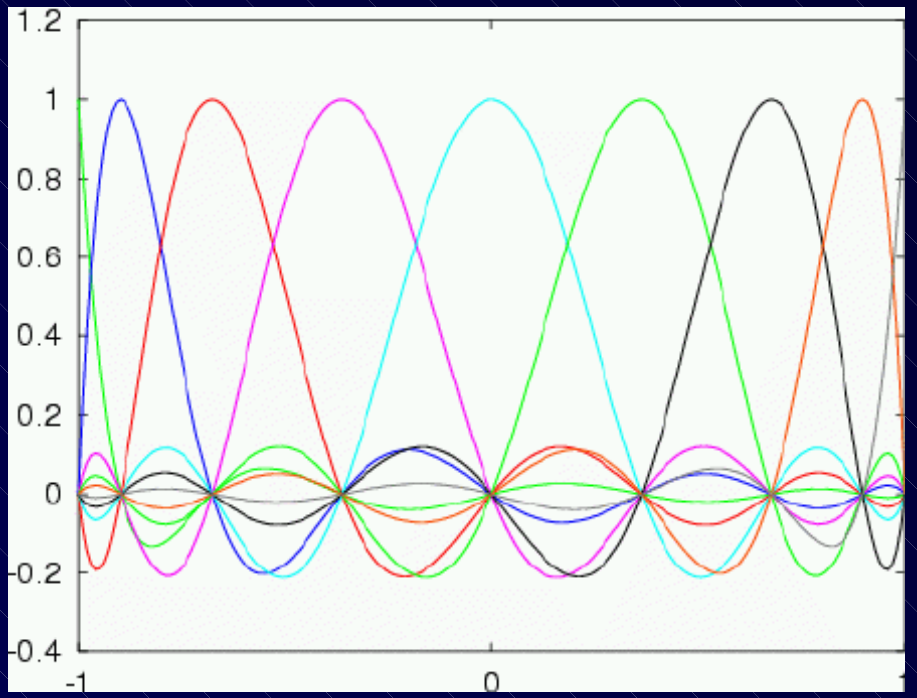


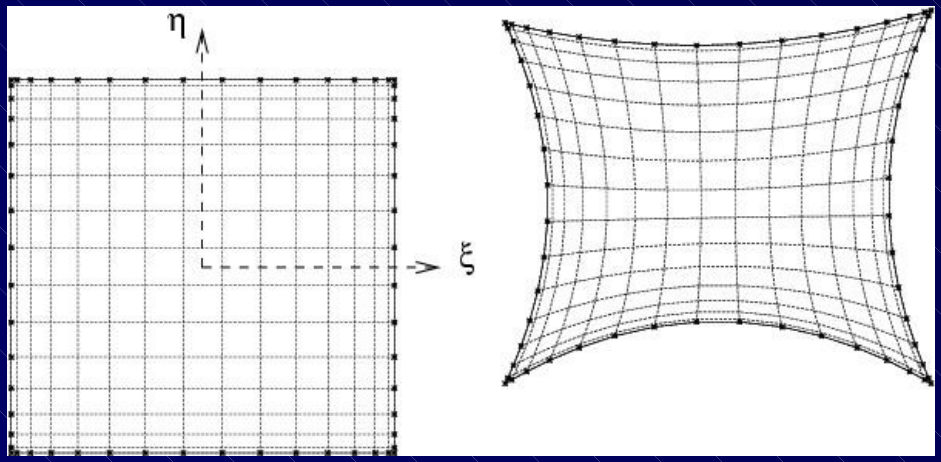
Where

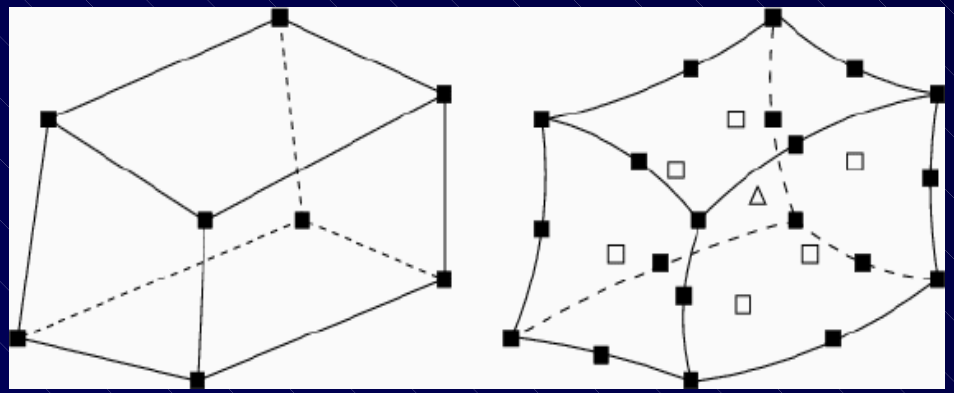


Finite elements

- High-degree pseudospectral finite elements
- N = 4 to 12 usually
- *Exactly* Diagonal mass matrix
- No linear system to invert







Equations of motion (solid)

Differential or *strong* form (e.g., finite differences):

$$\rho \partial_t^2 \mathbf{s} = \nabla \cdot \mathbf{T} + \mathbf{f}$$

We solve the integral or *weak* form:

$$\int \rho \mathbf{w} \cdot \partial_t^2 \mathbf{s} \, d^3\mathbf{r} = -\int \nabla \mathbf{w} : \mathbf{T} \, d^3\mathbf{r}$$

$$+ \mathbf{M} : \nabla \mathbf{w}(\mathbf{r}_s) S(t) - \int_{F-S} \mathbf{w} \cdot \mathbf{T} \cdot \hat{\mathbf{n}} \, d^2\mathbf{r}$$

+ attenuation (memory variables) if needed for viscoelasticity.