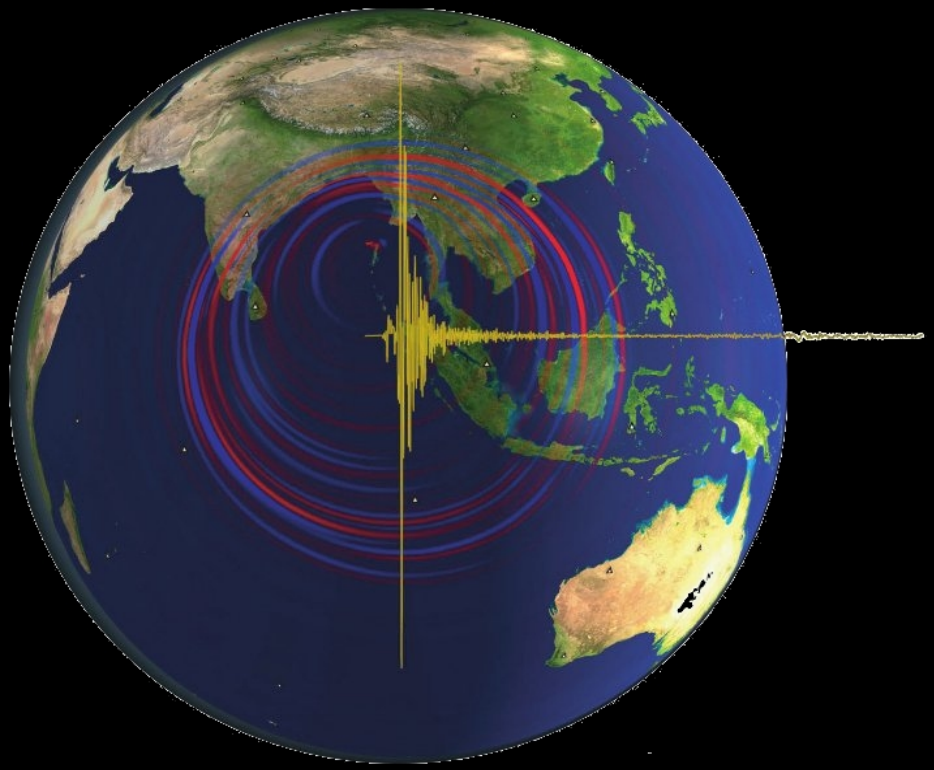
SPECFEM3D_GLOBE



Min Chen
Vala Hjörleifsdóttir
Sue Kientz
Dimitri Komatitsch
Qinya Liu
Alessia Maggi
David Michéa
Brian Savage
Bernhard Schuberth
Leif Strand
Carl Tape
Jeroen Tromp

The SPECFEM3D source code is freely available
for academic non-commercial research at
http://www.gps.caltech.edu/~jtromp/research/downloads/register.html

Mostly developed at Caltech (USA) and University of Pau (France)
History: v1.0: 1999/2000 ; v3.6: 2005; v4.0: today

Brief history of numerical methods

- Seismic wave equation : tremendous increase of computational power $\Rightarrow$ 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

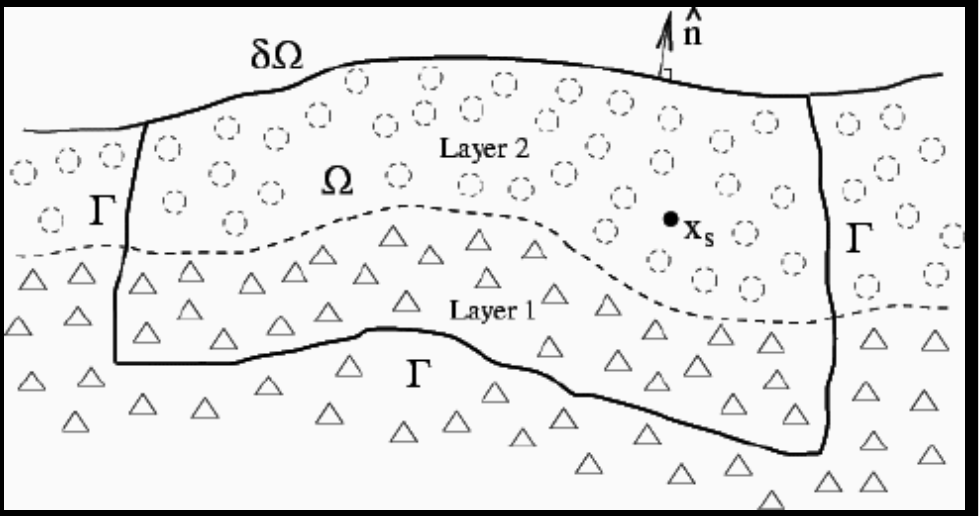
- Boundary-element or boundary-integral methods (Kawase 1988, Sanchez-Sesma et al. 1991) : homogeneous layers, expensive in 3D

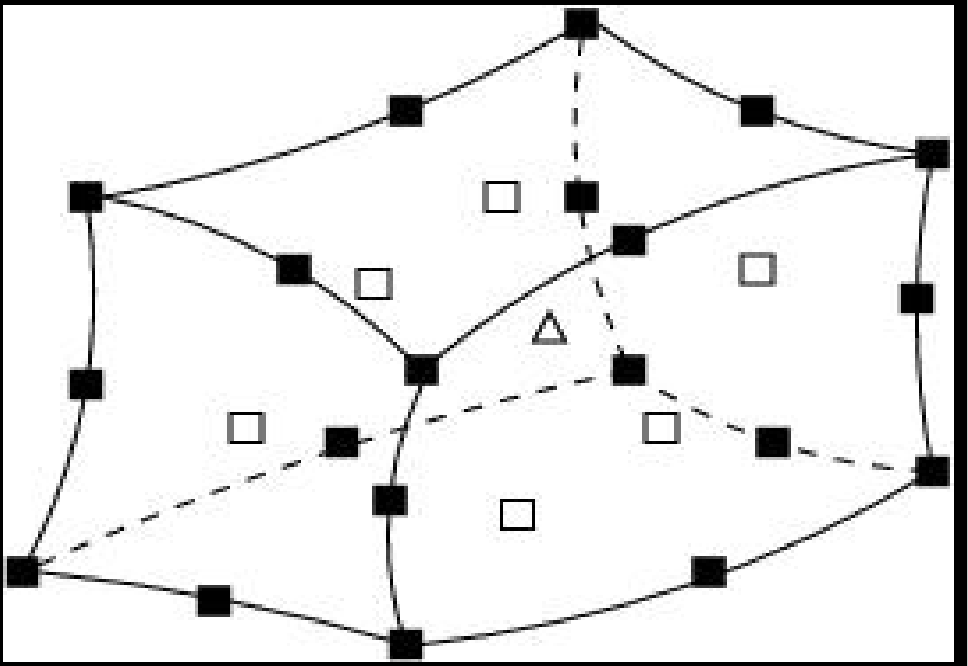
- Spectral and pseudo-spectral methods (Carcione 1990) : smooth topographies and media, difficult for boundary conditions, difficult on parallel computers

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

Spectral-Element Method

- Developed in Computational Fluid Dynamics (Patera 1984)
- Introduced for 3D elastodynamics by Komatitsch et al., Chaljub et al.
- Large curved “spectral” finite-elements with high-degree polynomial interpolation: accuracy of a pseudospectral method, flexibility of a finite-element method
- Mesh honors the main discontinuities (velocity, density) and topography
- Very efficient on parallel computers, no linear system to invert (diagonal mass matrix)





- Curved 27-node elements are mapped to unit cube

$$x(\xi) = \sum_{a=1}^{27} N_a(\xi)\, x_a$$

- provides efficient way of capturing curvature of sphere, and accuracy for surface waves
- High-degree pseudospectral finite elements: Number of integration Gauss Lobatto Legendre (NGLL) points = 5 to 9 usually

Equations of Motion

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

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

We solve the integral or *weak* form:

$$\int \rho w \cdot \partial_t^2 s d^3 r = -\int \nabla \mathbf{w} : \mathbf{T} \, d^3 r + \mathbf{M} : \nabla w(r_s) S(t) - \int_{F-S} w \cdot T \cdot \hat{n} d^2 r$$

+ attenuation (memory variables) and ocean load

- Fluid : Differential or *strong* form:

$$\rho \partial_t v = -\nabla p \qquad \partial_t p = -\kappa \nabla \cdot v$$

We use a generalized velocity potential χ $\quad p = \partial_t x$

the integral or *weak* form is:

$$\int \kappa^{-1} w \partial_t^2 x d^3 r = -\int \rho^{-1} \nabla w \cdot \nabla x d^3 r + \int_{F-S} w \hat{n} \cdot \mathbf{v} \, d^2 r$$

⇒ three times cheaper than in the solid (scalar potential, not vector)
⇒ natural coupling with solid

The global earth

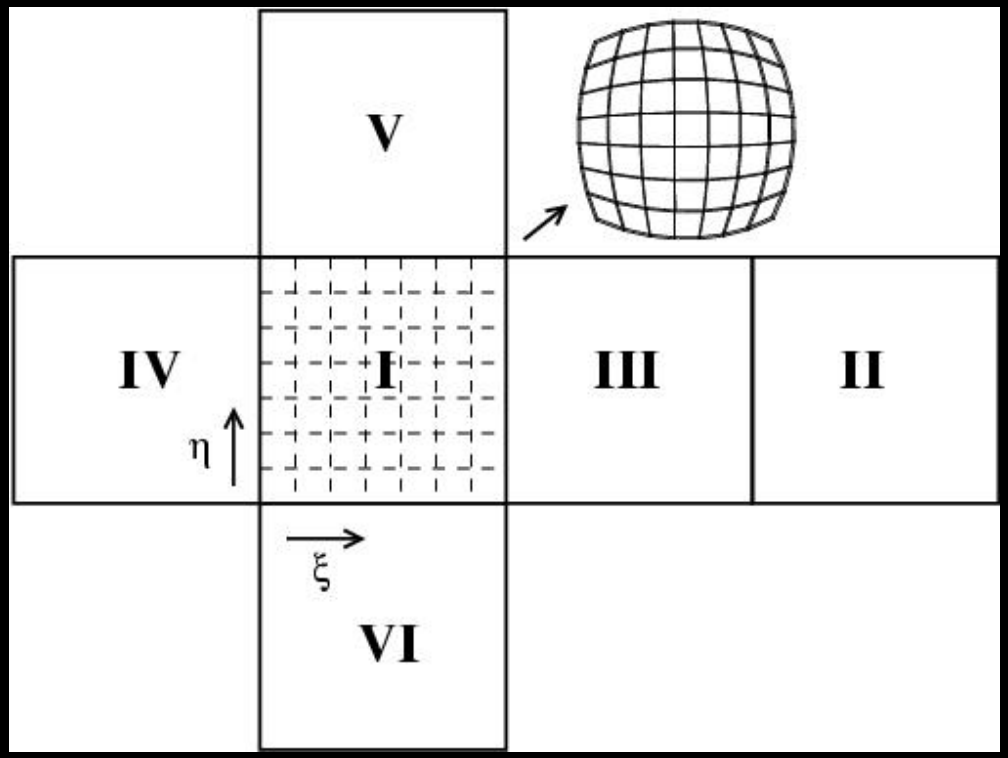
- Need accurate numerical modeling to study Earth structure (global scale)
- Very large models at high frequency (3D Earth)
- Complexity: classical methods (ray tracing, finite difference, pseudo-spectral) *do not work* for this problem (surface waves, anisotropy, fluid/solid interfaces, Earth's crust etc.)
- The challenge of the global earth :
  - A slow, thin, highly variable crust
  - Sharp radial velocity and density discontinuities
  - Fluid-solid boundaries (outer core of the Earth)
  - Anisotropy
  - Attenuation
  - Ellipticity
  - topography and bathymetry
  - Rotation
  - Self-gravitation
  - 3D mantle and crust models (lateral variations)

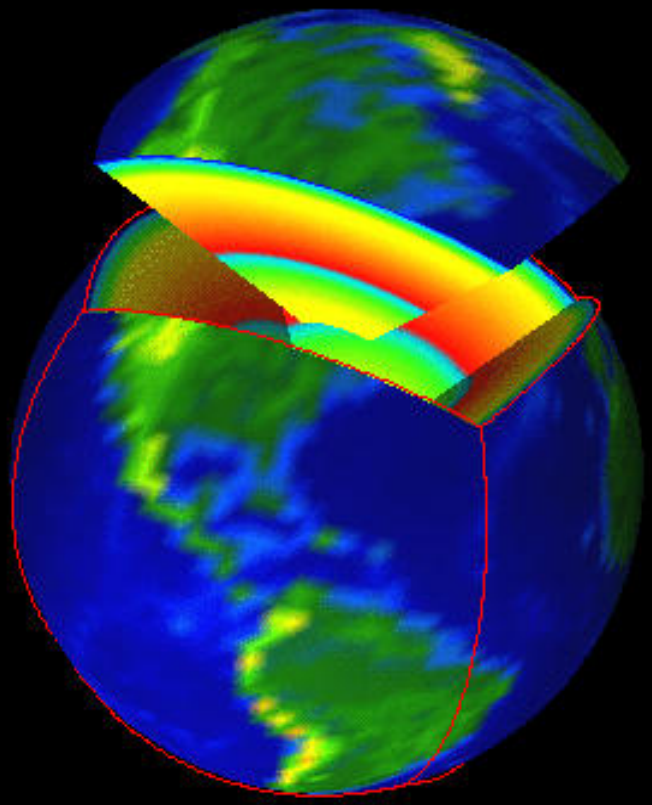
SPECFEM3D GLOBE

- Known problems in version v3.6
  - Very significant load imbalance
    - Imbalance induced by the mesh
    - Imbalance induced by the cache misses
  - Problems with I/O when all the processors read or write large files (topography, seismograms etc) from a remotely-mounted home file system (GPFS, NFS...) : In Marenostrum (Barcelona), we crashed the file system in 2006
  - Overall performance could be improved

- What's new in v4.0?
  - new doubling brick in the mesh, new perfectly load-balanced mesh
  - more flexible routines for mesh design, one more doubling level
  - In the inner core, new inflated central cube with optimized shape
  - far fewer mesh files saved by the mesher
  - Better one layer crust version for 1D models (better stability => less CPU time needed)
  - Better sampling in the crust
  - Far fewer cache misses: program runs faster (up to a factor of 3.3 on Marenostrum, 1.6 on pangu)
  - Same number of cache misses in each slice => no more imbalance induced by cache misses
  - the seismograms are now appended and not rewritten entirely each time, they are sent directly on the master by MPI, thus no more need to collect them on the nodes at the end of the simulation.

The Cubed Sphere

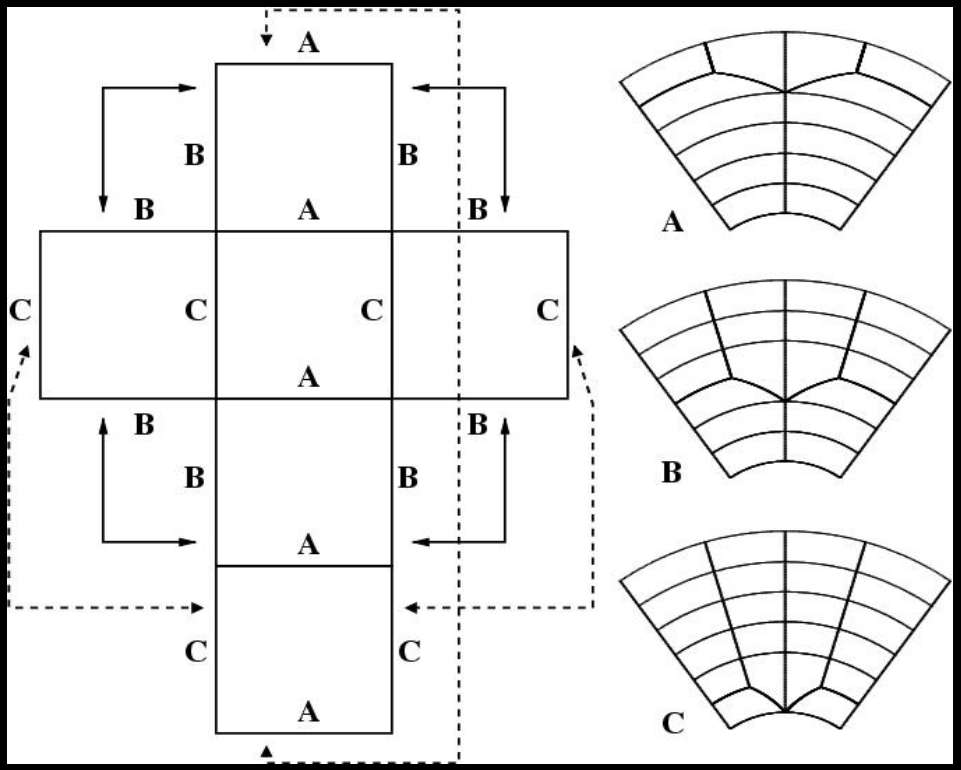


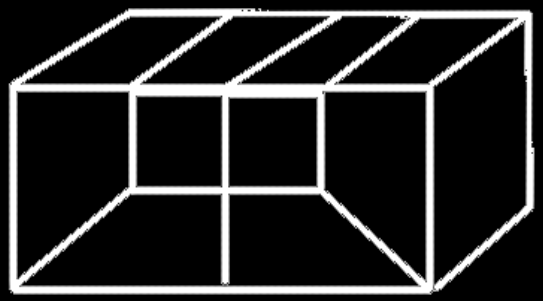


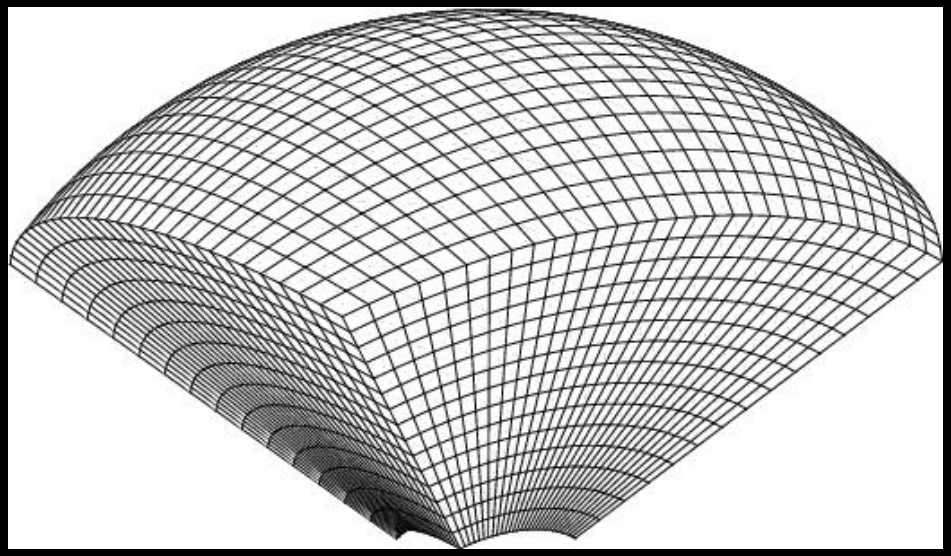
- **“Gnomonic” mapping (Sadourny 1972)**
- **Ronchi et al. (1996), Chaljub (2000)**
- **Analytical mapping from six faces of cube to unit sphere**

SPECFEM3D GLOBE v3.6

- Element size needs to increase with depth because of velocity gradients to keep a similar number of points per wavelength
- Regular mesh does not work near center: the mesh must be adapted => doubling layers
- With the old brick, doubling in two directions in the same layer was not possible, therefore made in differents layers for each type of chunk (A, B & C) => different number of elements = load imbalance
- Phenomenon even worse because one doubling in the anisotropic layer, where 21 arrays are needed

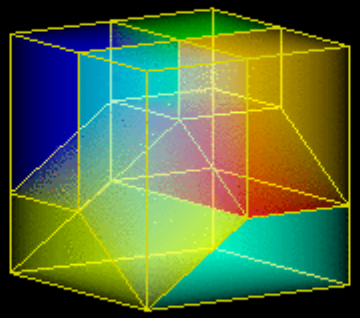




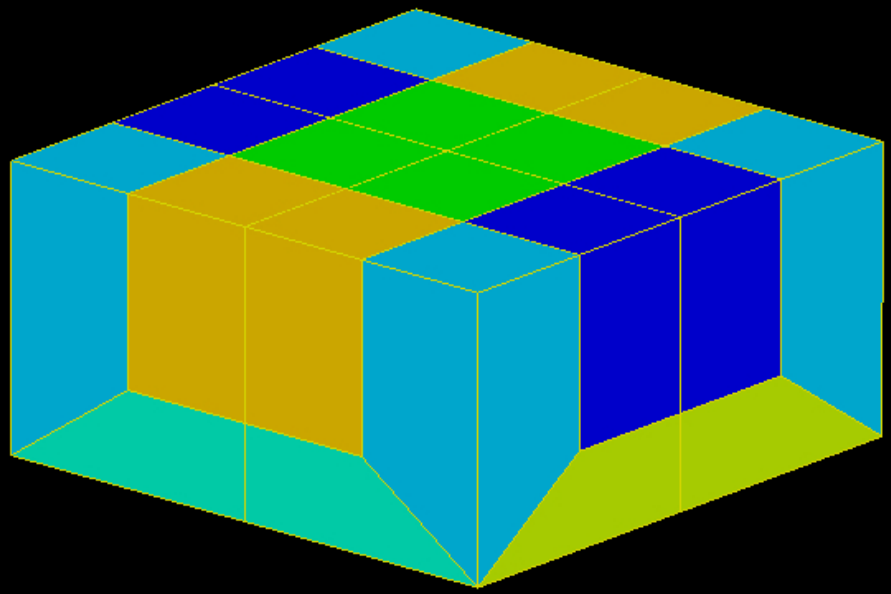


SPECFEM3D GLOBE v4.0

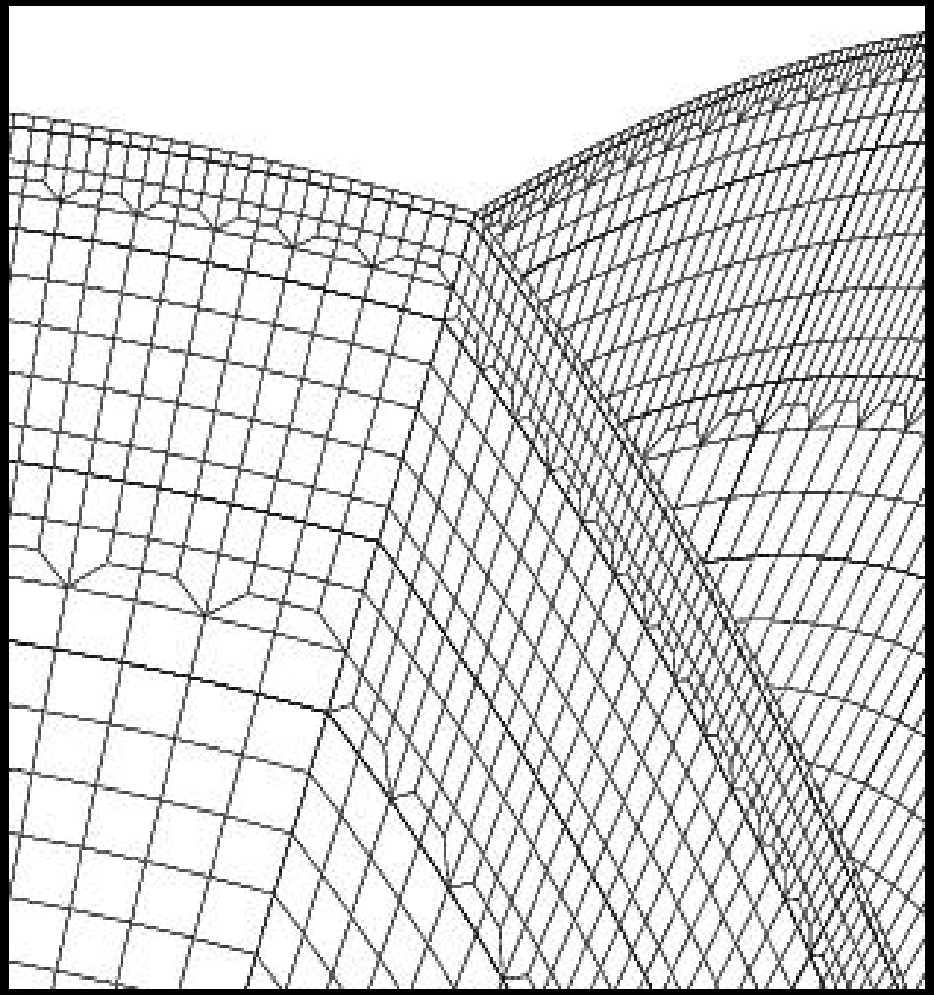
- A new mesh brick allows us to get rid of the three types of mesh chunks (A, B and C) and use the same number of elements in all the slices instead



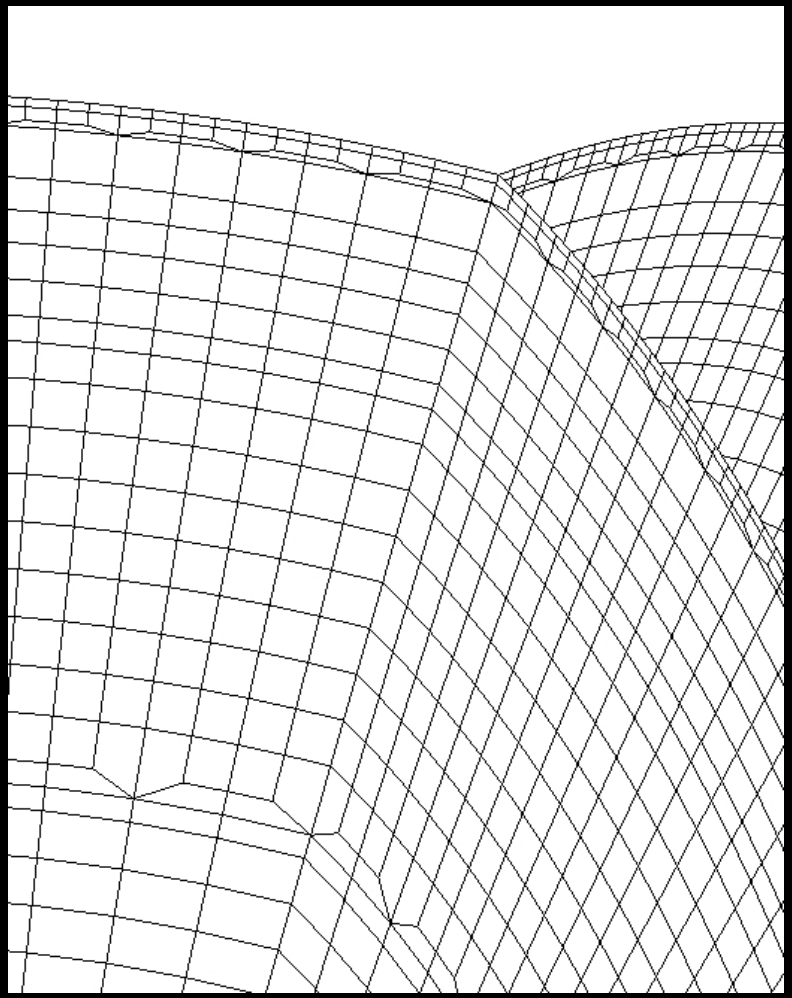
- The full doubling brick is obtained by symmetry based on the basic brick
- Makes it possible to carry out the doubling on two directions in the same layer
- Thus the number of operations performed in each block is the same => load balancing very significantly improved and close to perfect.



SPECFEM3D GLOBE



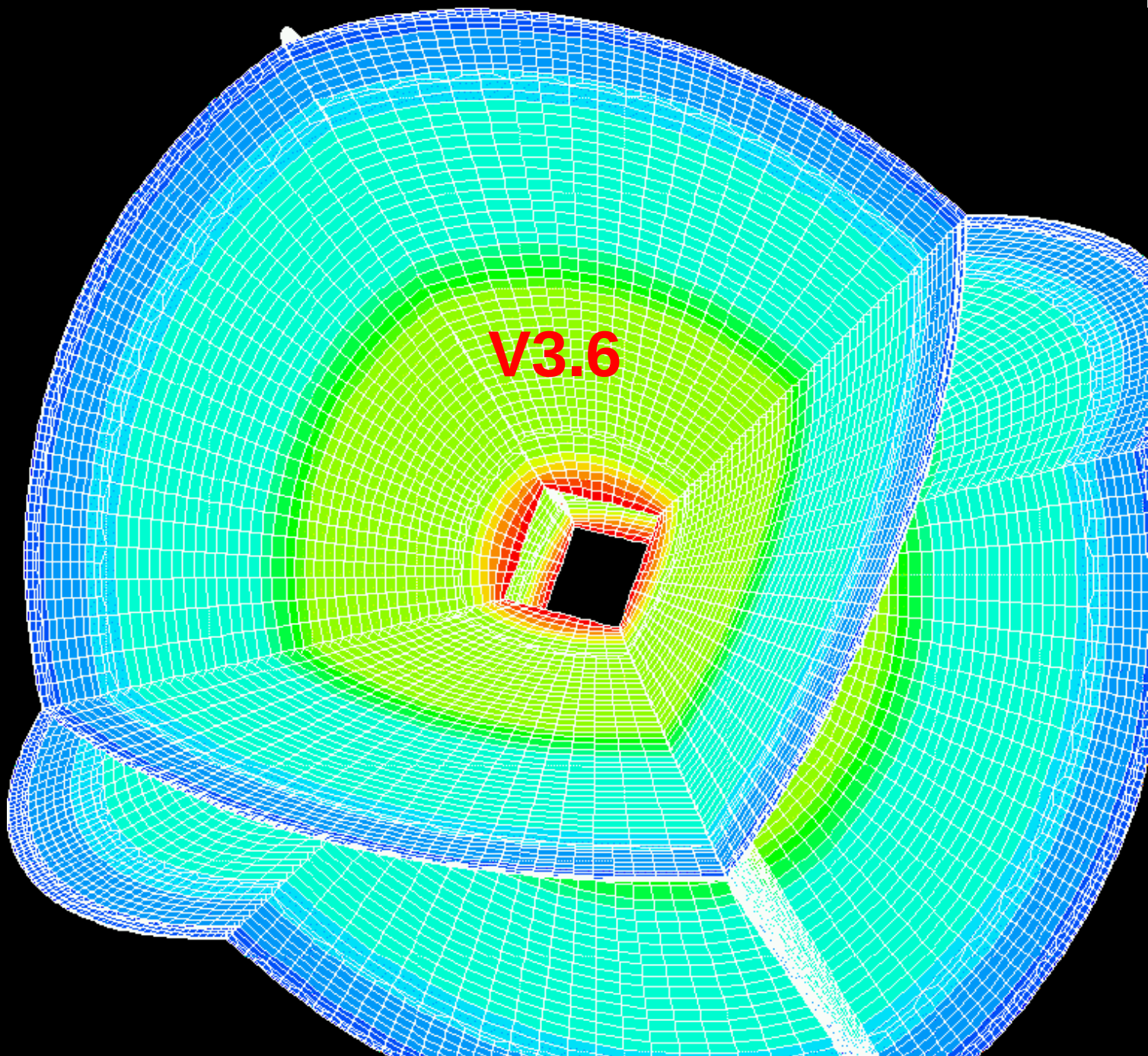
detail of the v3.6 mesh



detail of the v4.0 mesh

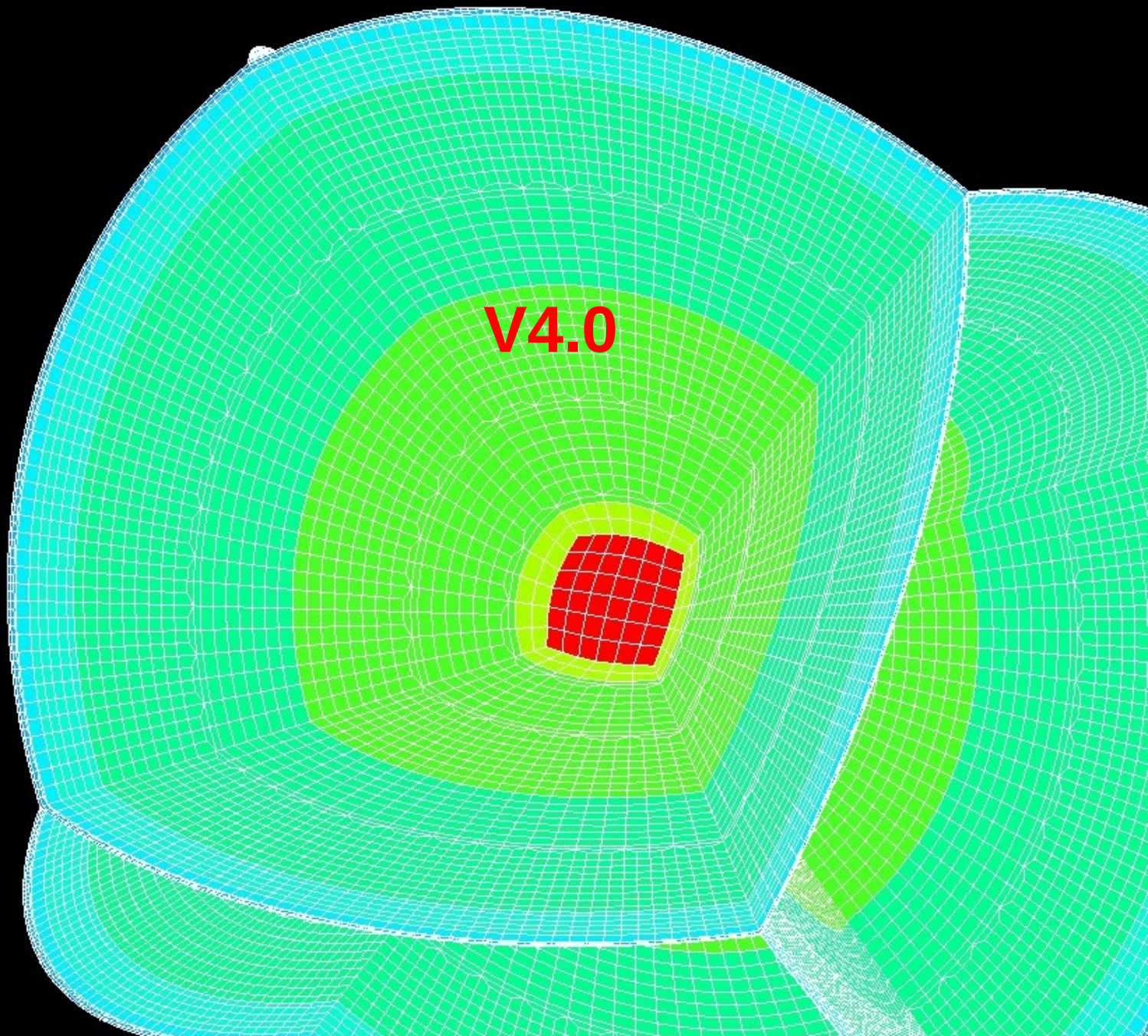
Mesh of the v3.6

- **Only 3 doubling layers: not optimal because oversampling at the top of the inner core**
- **Second doubling near the d670 discontinuity, not optimal at all because subsampling in the middle of the mantle and lower mantle**
- **Central cube not inflated => bad skewness and aspect ratio for the elements near the central cube**



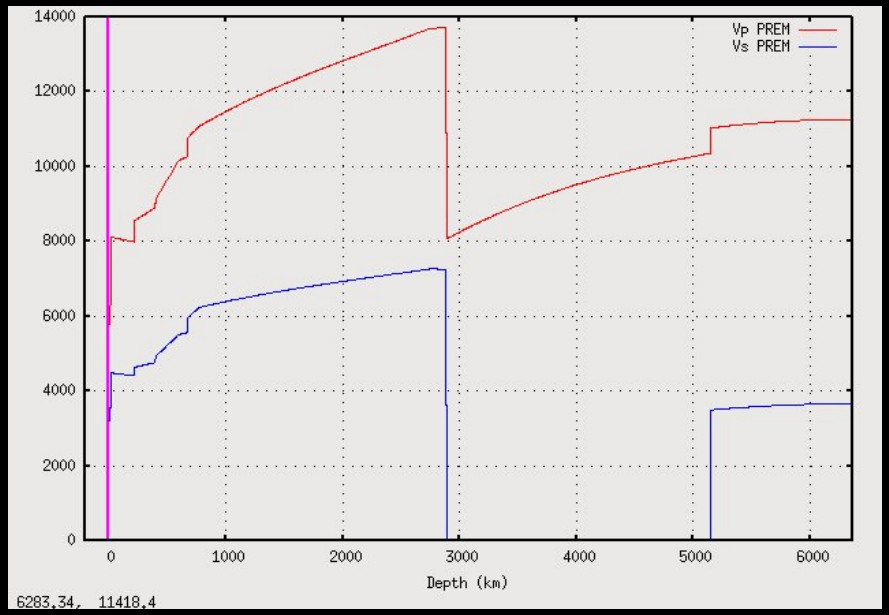
Mesh of the v4.0

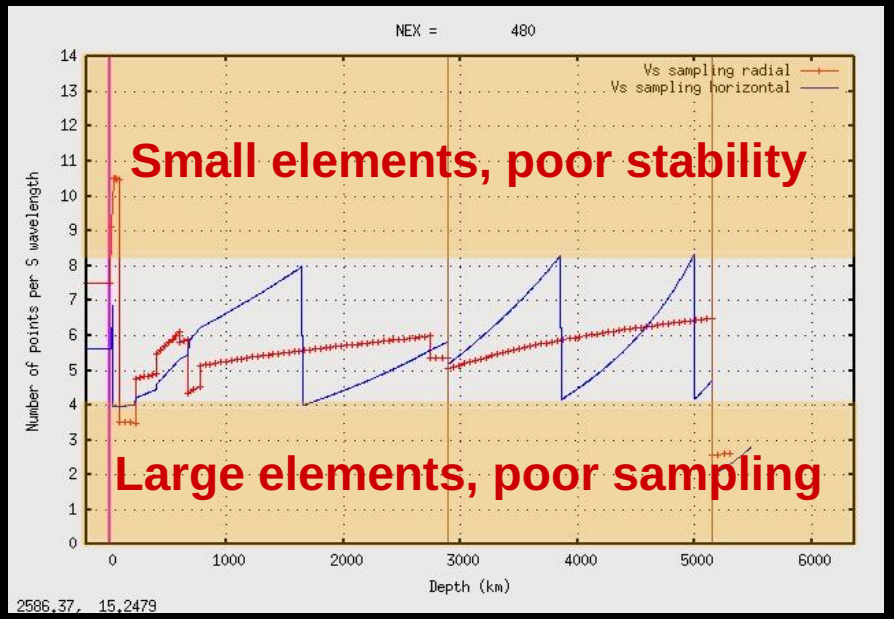
- 4 doubling layers: much better stability condition at the top of the inner core
- Perfectly balanced number of elements in all chunks
- New inflated central cube => better skewness and aspect ratio for the elements near the central cube
- Depth of the doubling layers better adapted to the model to guarantee best possible sampling

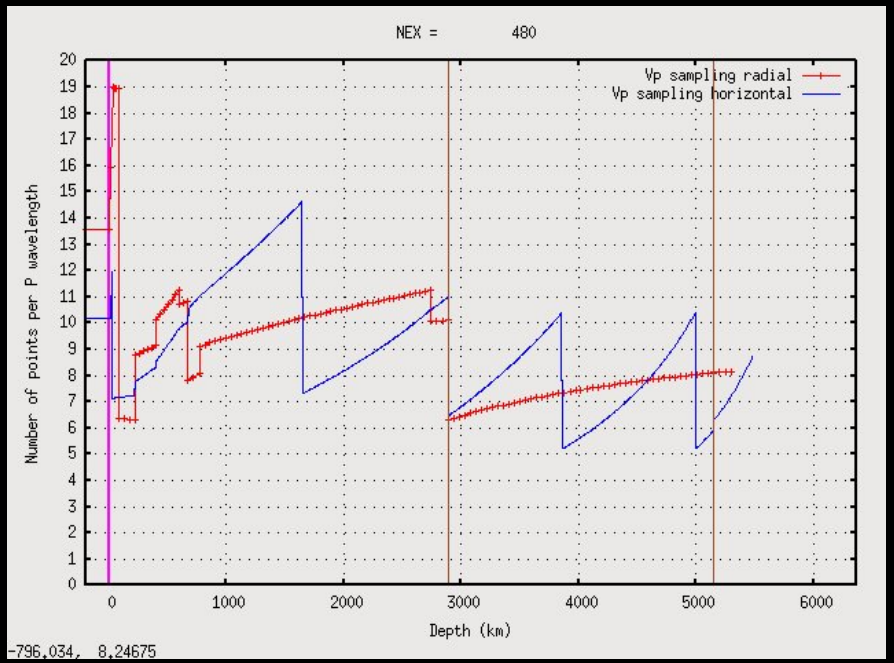


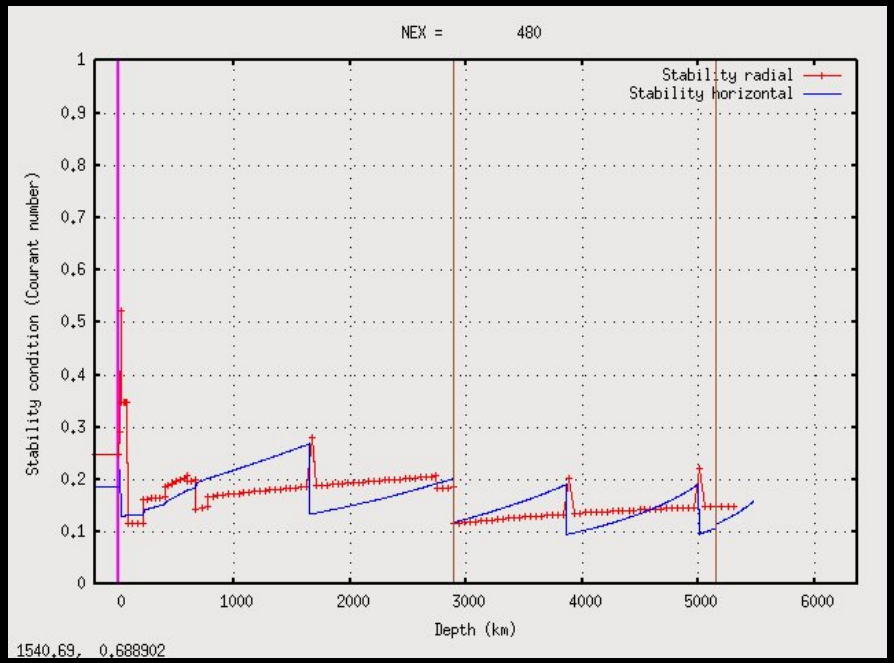
Mesh of the v4.0

- **Depth of the doubling layers better adapted to the model to guarantee best possible sampling**

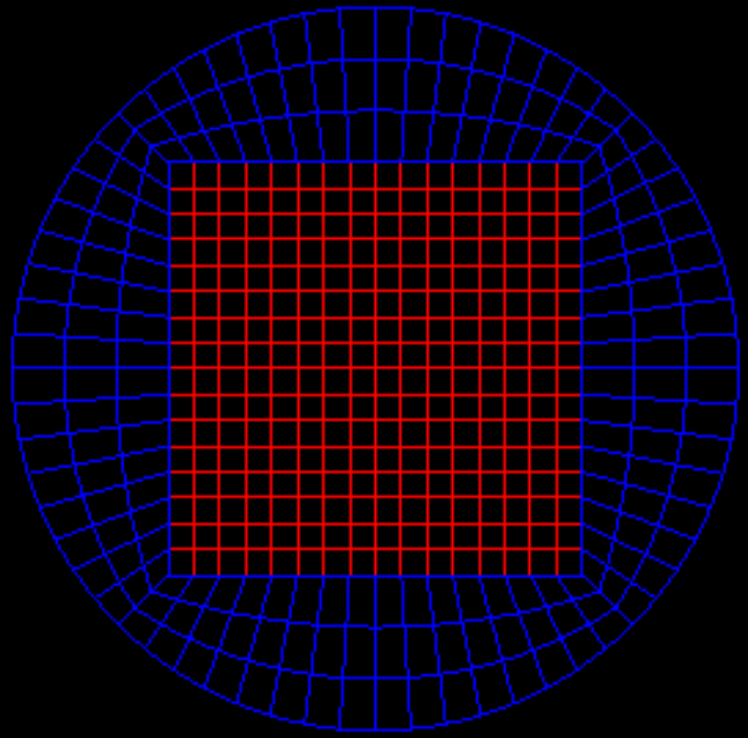


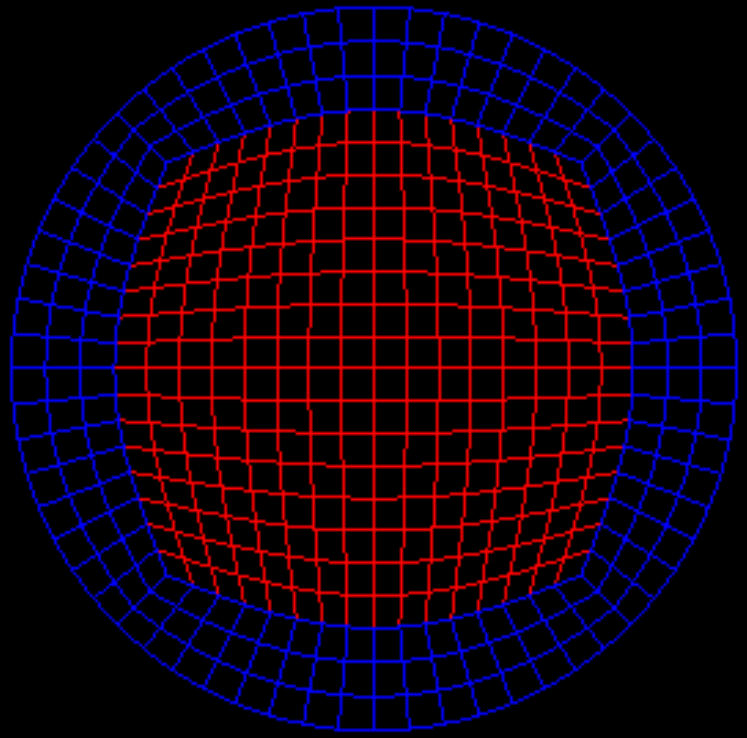




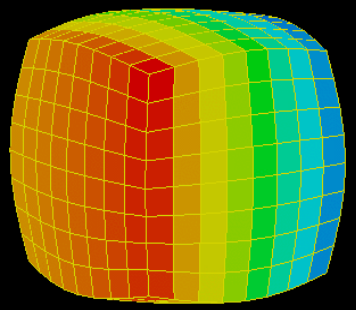


Inflated central cube

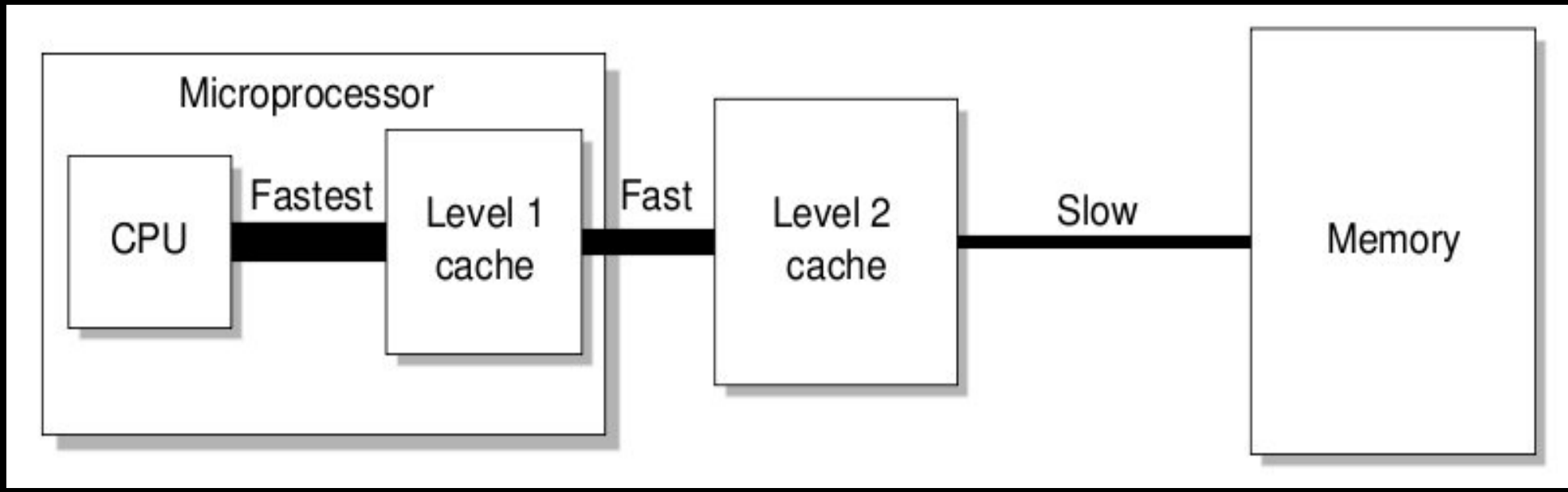


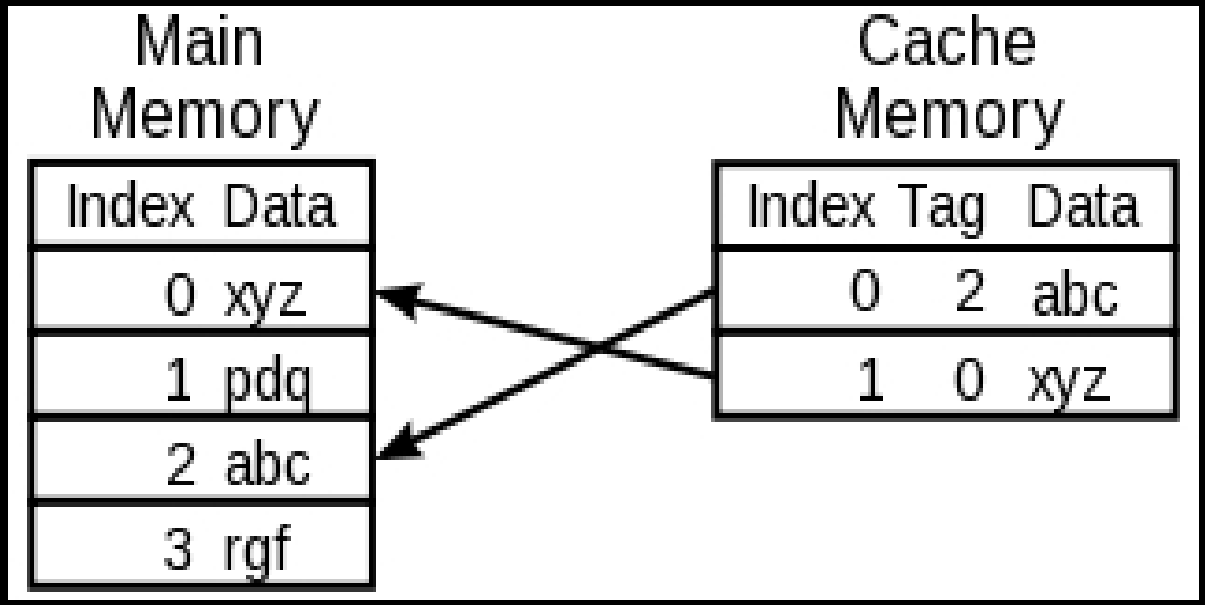


- **Shape of central cube has been optimized using two variables**
  - **« radius » i.e. size of the central cube**
  - **its « inflate » factor**
- **... and based on four criteria**
  - **skewness (average & worst)**
  - **aspect ratio (average & worst)**



The cache hierarchy on modern processors





Memory: principle of locality

The principle of locality deals with the process of accessing a single resource multiple times

- Temporal locality: a resource referenced at one point in time will be referenced again sometime in the near future
- Spatial locality: the likelihood of referencing a resource is higher if a resource in the same neighborhood has been referenced
- Sequential locality : memory is accessed sequentially

Locality must be optimized in loops that tend to reference arrays or other data structures by indices.

Increasing and exploiting locality of memory references is an important optimization technique

Optimization of global addressing

In 3D and for NGLL=5, for a regular hexahedral mesh there are:

125 GLL integration points in each element

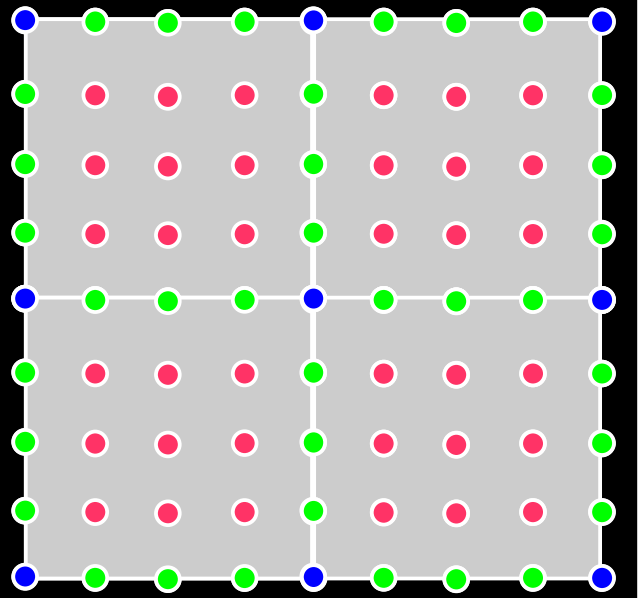
27 belong only to this element (21.6%)

54 belong to 2 elements (43.2%)

36 belong to 4 elements (28.8%)

8 belong to 8 elements (6.4%)

=> 78.4% of the GLL integration points belong to at least 2 elements



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

The mesh is created element by element, then the common points are identified, and a global addressing is created:

1 <= global_addressing(num_element, i, j, k) <= total number of GLL points

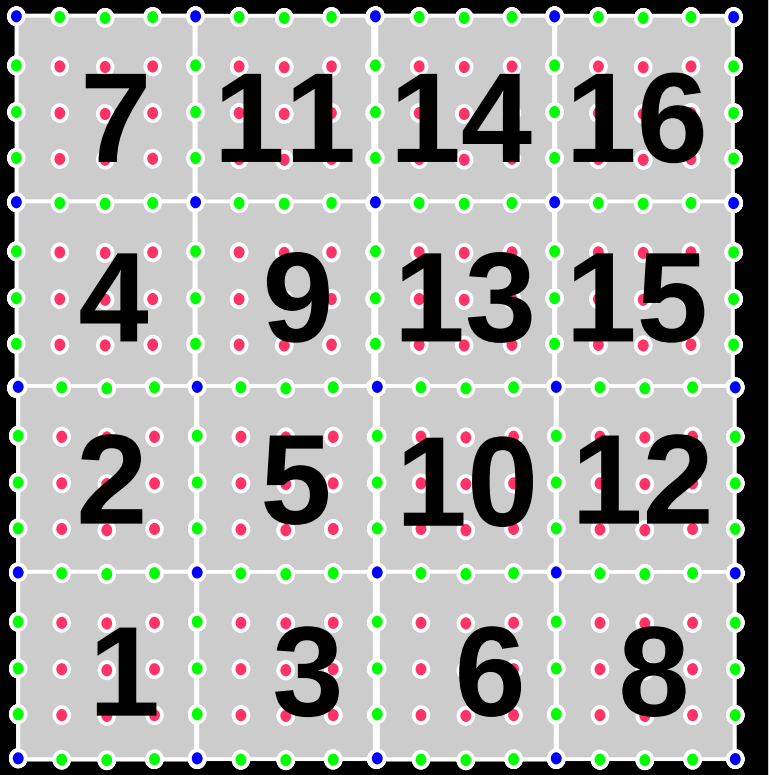
This array must be reordered once and for all in the mesher to optimize the future memory access order of the points and elements in the solver, in order to maximize spatial and temporal locality

Sorting of the elements

To increase spatial and temporal locality for the global access of the points that are common to several elements, the order in which we access the elements can be optimized.

The goal is to find an order that minimizes the memory strides for the global arrays.

We use the classical reverse Cuthill-McKee (1969) algorithm, which consists in renumbering the vertices of the graph to reduce the bandwidth of the adjacency matrix



Sorting the elements with the Cuthill-McKee algorithm before renumbering the global index table also increases the spatial and temporal locality:

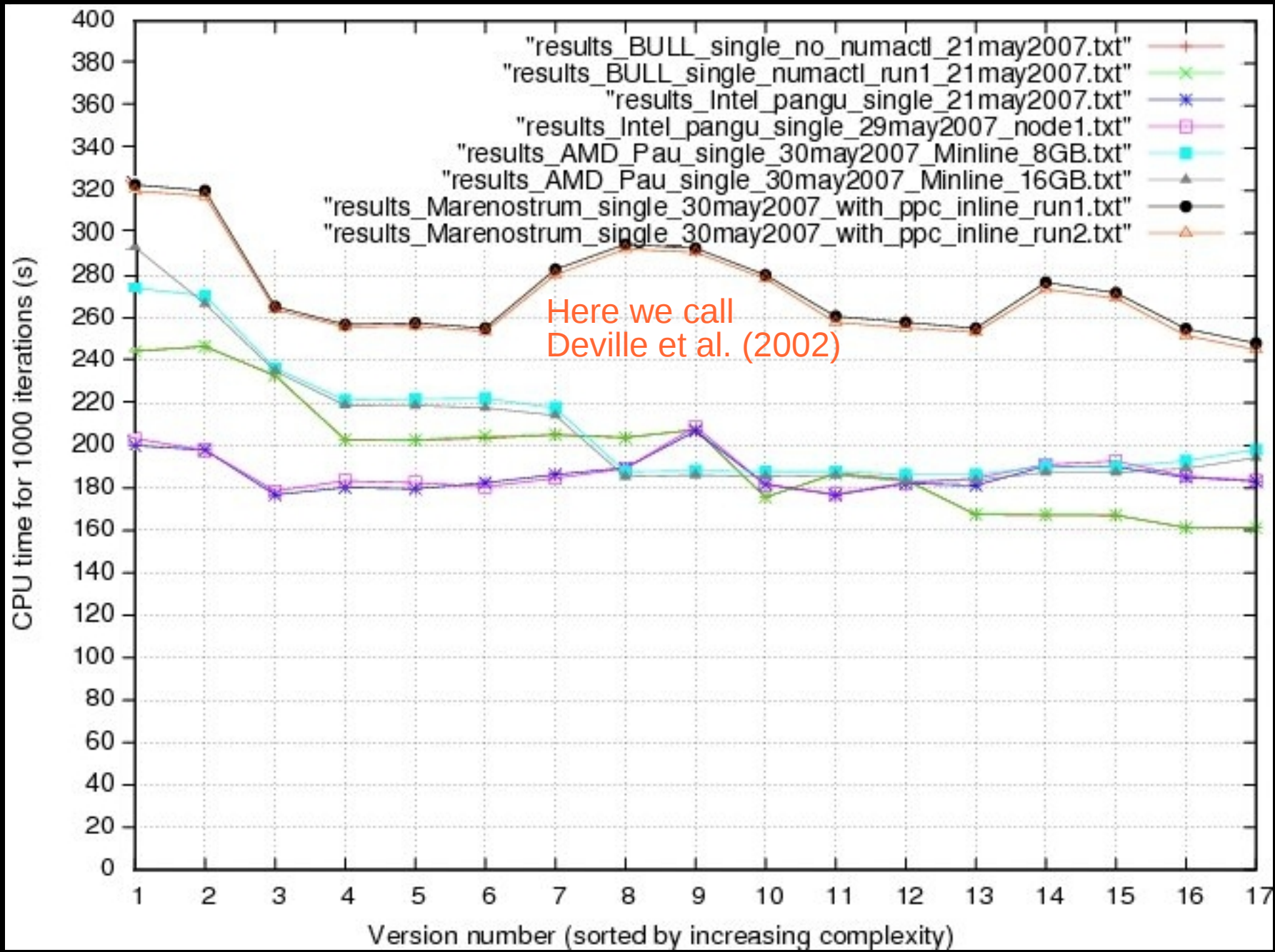
- spatial locality, because the common points of the connected elements will be stored statistically closer in memory

- temporal locality, because these common points will be reaccessed sooner

Loop splitting

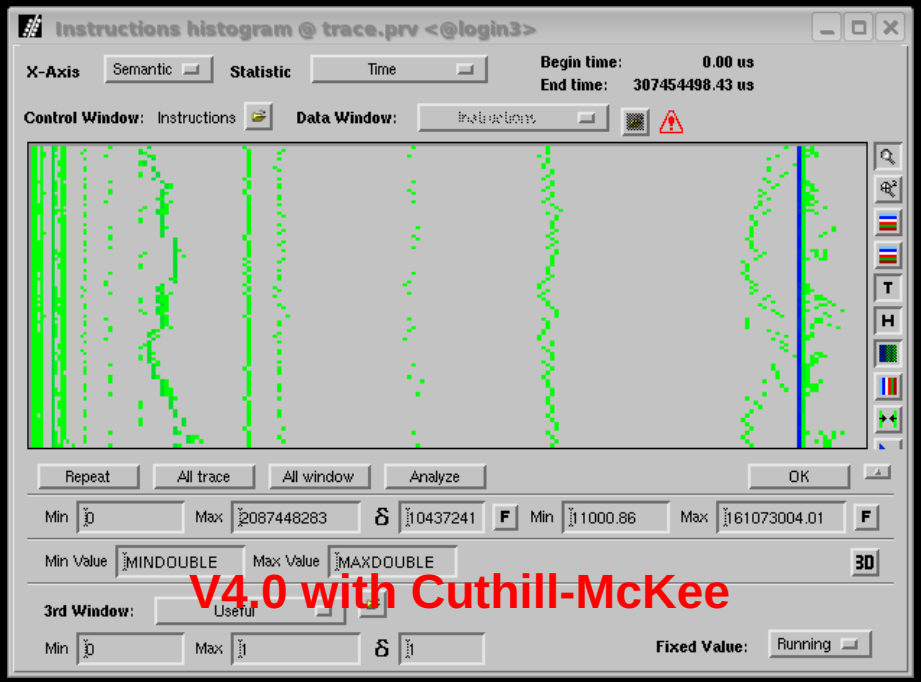
```
! ************************************************
!   big loop over all spectral elements in the solid
! ************************************************

! set acceleration to zero
  accel(:,:) = 0._CUSTOM_REAL

  do ispec = 1,NSPEC_CRUST_MANTLE
    do k=1,NGLLZ
      do j=1,NGLLY
        do i=1,NGLLX

                ... ! work on arrays(ispec,i,j,k)

        enddo
      enddo
    enddo

...

! sum contributions from each element to the global mesh and add gravity terms
    do k=1,NGLLZ
      do j=1,NGLLY
        do i=1,NGLLX
          iglob = ibool(i,j,k,ispec)
          accel(1,iglob) = accel(1,iglob) + sum_terms(1,i,j,k)
          accel(2,iglob) = accel(2,iglob) + sum_terms(2,i,j,k)
          accel(3,iglob) = accel(3,iglob) + sum_terms(3,i,j,k)
        enddo
      enddo
    enddo
  enddo ! spectral element loop
```

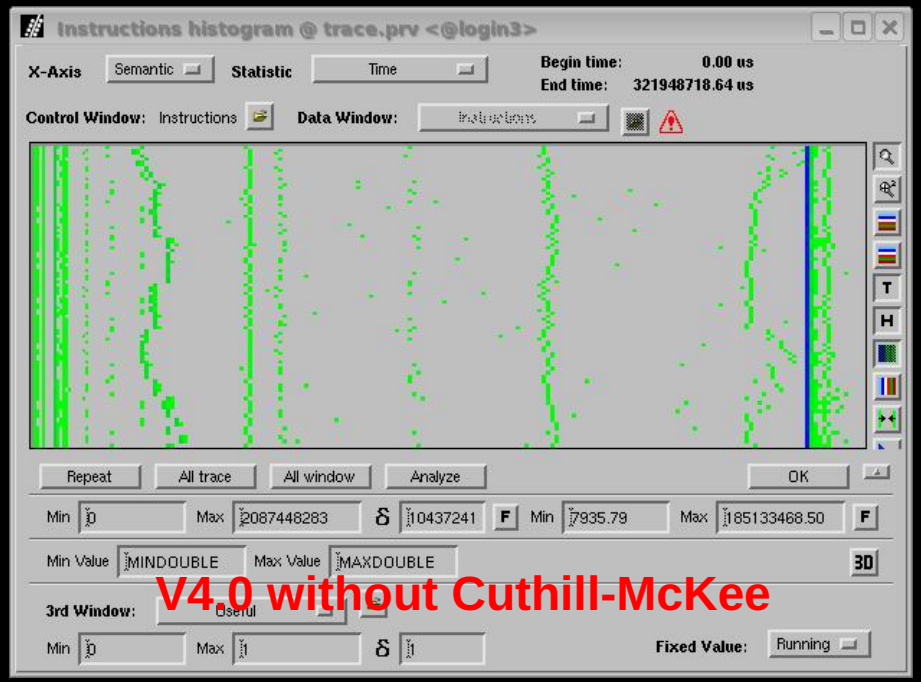
Impact of loop inlining / splitting / reordering



Results for load balancing: instructions

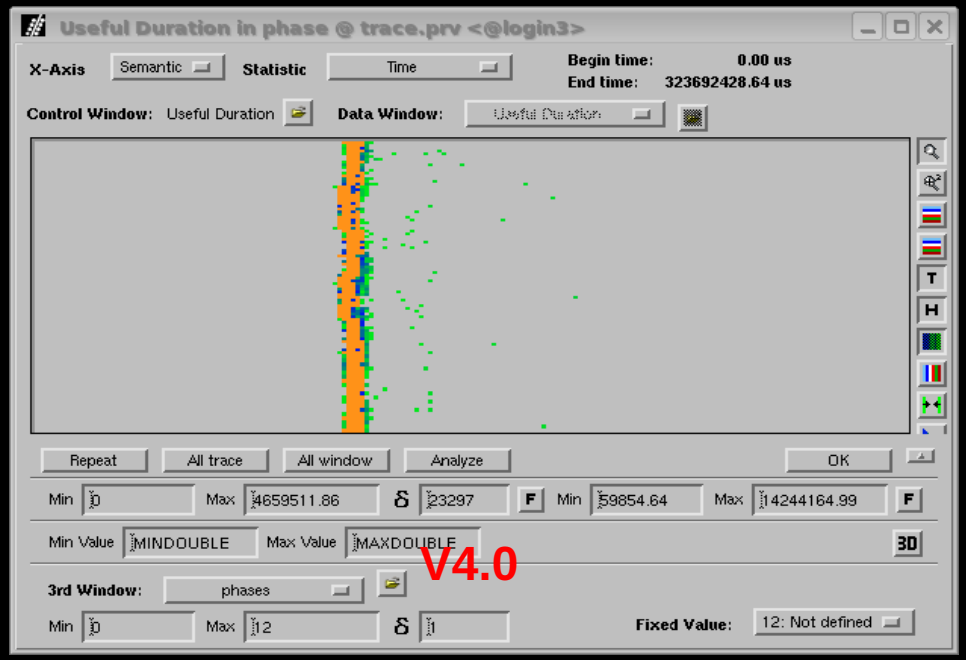
Analysis of parallel execution performed with Prof. Jesús Labarta in Barcelona (Spain) using his Paraver software package

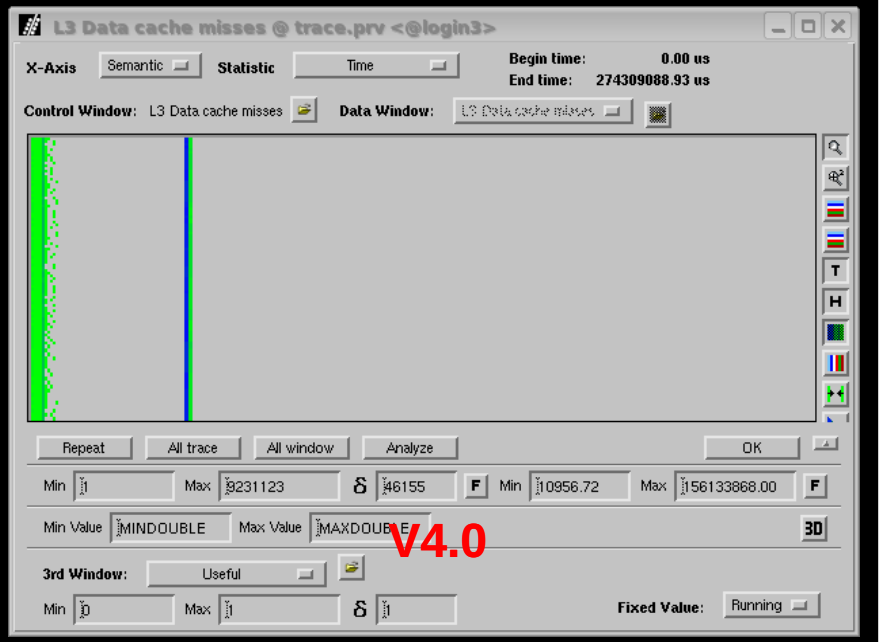




- Number of instructions executed in each slice is now well balanced
- Cuthill-McKee has almost no effect on that because we use high-order finite elements (of Q4 type), each of them fits in the L1 cache and for any such element we perform a very large number of operations using data that is already in L1
- Cuthill McKee sorting has been removed from v4.0 because of side effects (bugs); it will be reintroduced in v4.1 once these bugs are resolved, but performance improvement should be small (maybe even negligible?)

Results for load balancing: cache misses

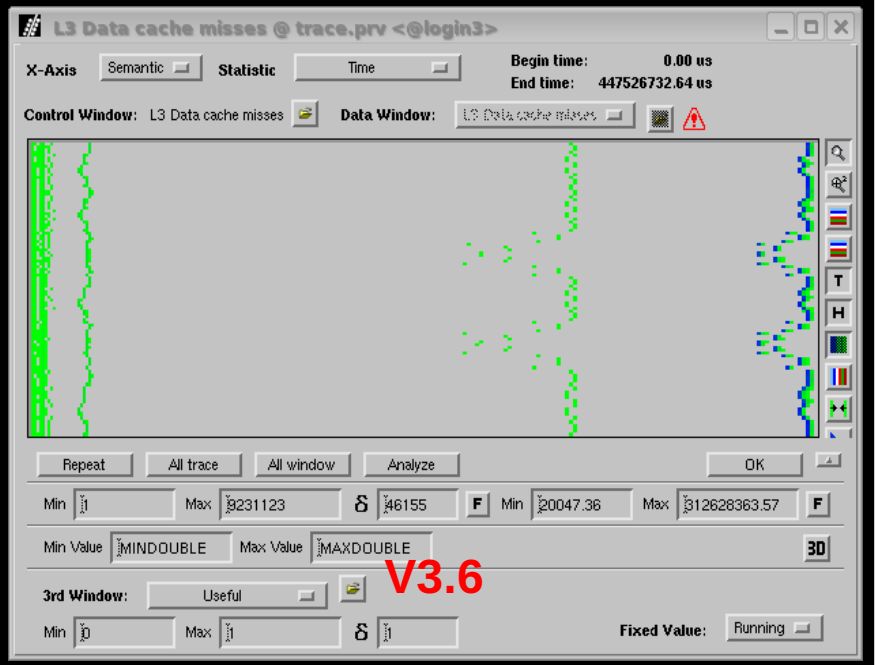




After adding Cuthill-McKee sorting, global addressing renumbering and loop reordering we get a perfectly straight line for cache misses, i.e. same behavior in all the slices and also almost perfect load balancing.

The total number of cache misses is also much lower than in v3.6

CPU time (in orange) is also almost perfectly aligned



Results of cache use optimization on Marenostrum

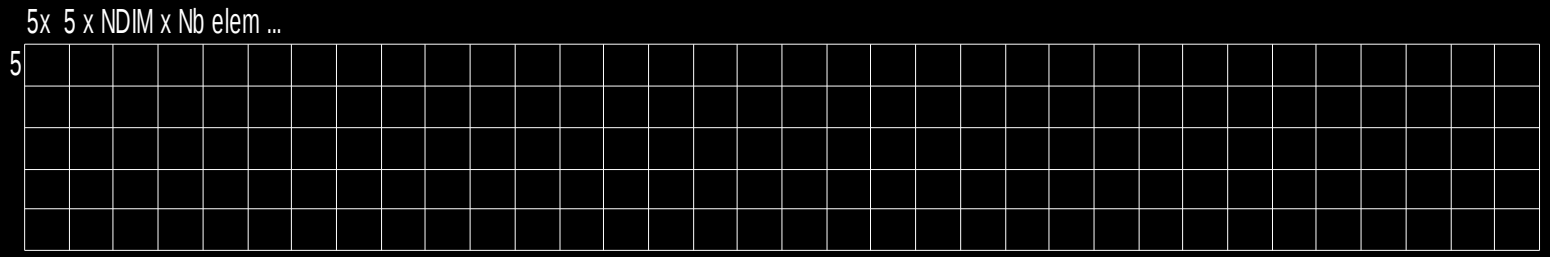
Summary of results of “time” command for a serial run, with several improvements implemented or not.

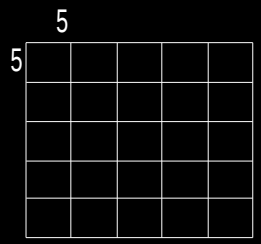
Global array sorted	Cuthill McKee	Split loop	User	Sys
No	No	No	4776.00	4.77
No	No	Yes	4259.18	5.45
Yes	No	No	1589.29	2.16
Yes	Yes	No	1509.71	2.40
Yes	Yes	Yes	1466.24	2.09

(measured time can fluctuate in a range of ± 2% because of system load)

We gain a factor of 1.55 in CPU time on pangu (Intel Itanium) and on AMD Opteron, and a factor of 3.3 (!!) on Marenostrum (the IBM PowerPC is very sensitive to cache misses)

BLAS 3
(Basic Linear Algebra Subroutines)





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

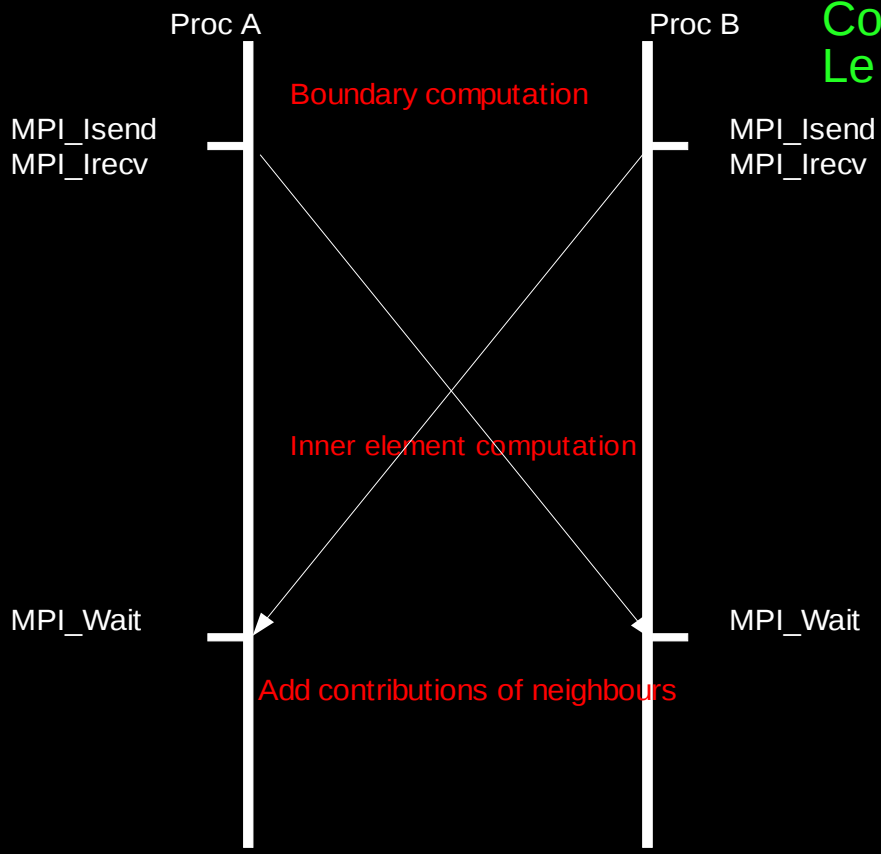
Can we use highly optimized BLAS matrix matrix products (90% of computations)?

- 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.

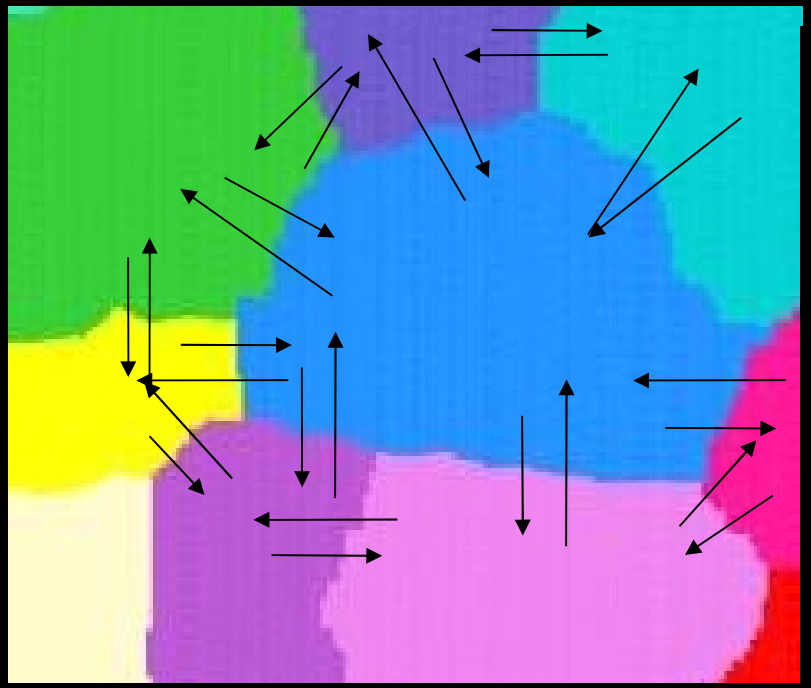
=> No need to, and cannot easily use BLAS

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

Non-blocking MPI



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



Another way to optimize MPI code is to overlap communications with computations using non-blocking MPI. But, for our code, the overall cost of communications is very small (< 5%) compared to CPU time.

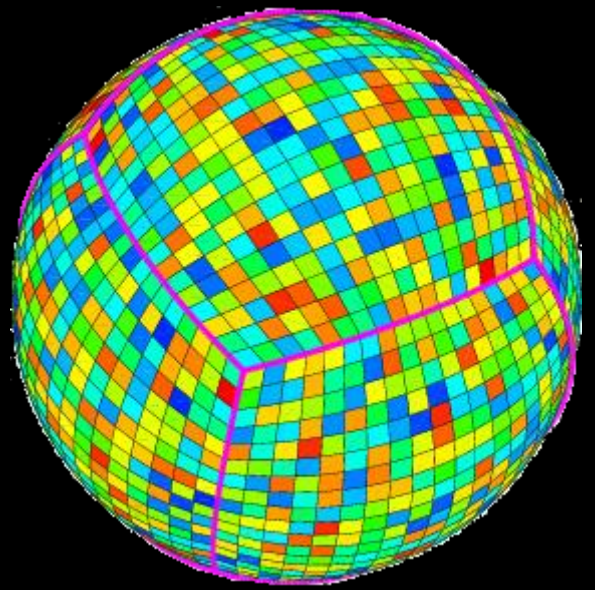
Also, looping on boundary elements contradicts Cuthill-McKee order and therefore causes cache misses.

=> No need to use non blocking MPI because potential gain is comparable to overhead

=> Tested in 2D, and we did not gain anything significant

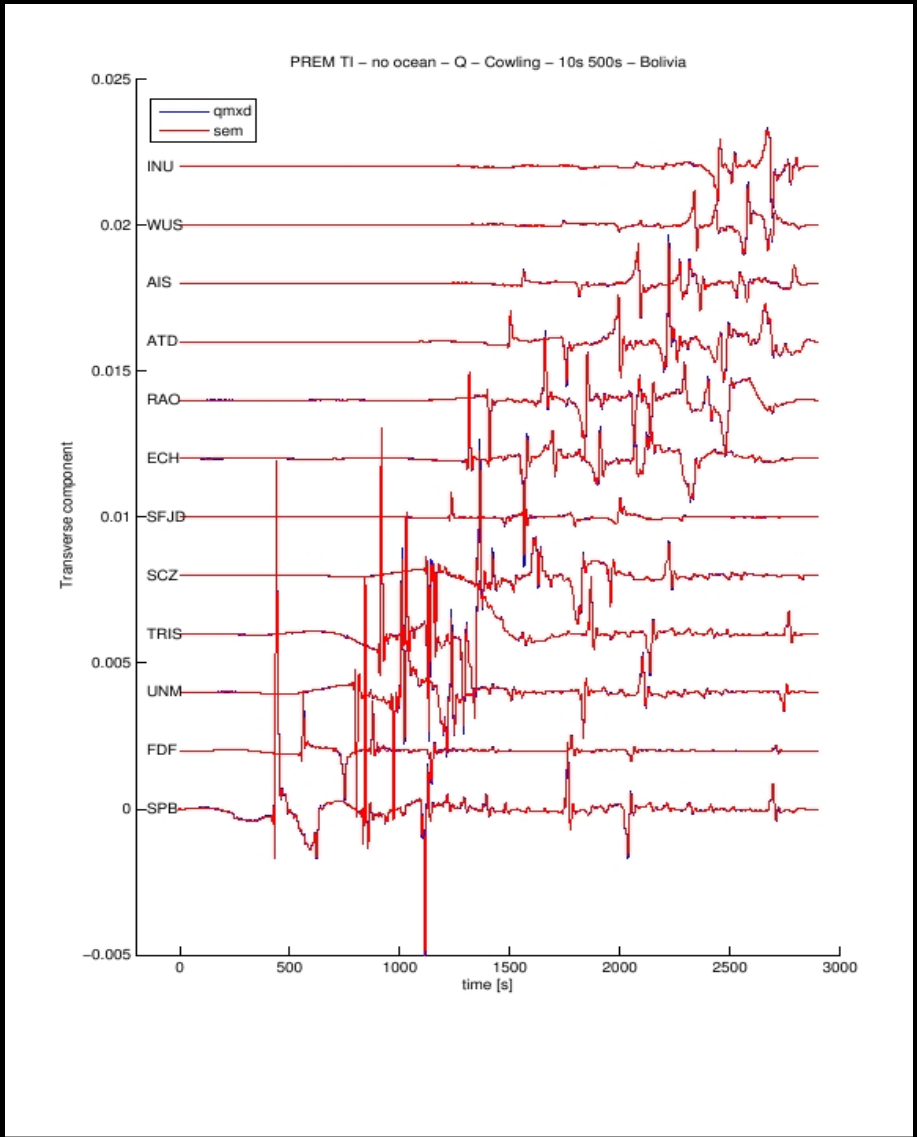
A very large run for PKP phases at 2 seconds

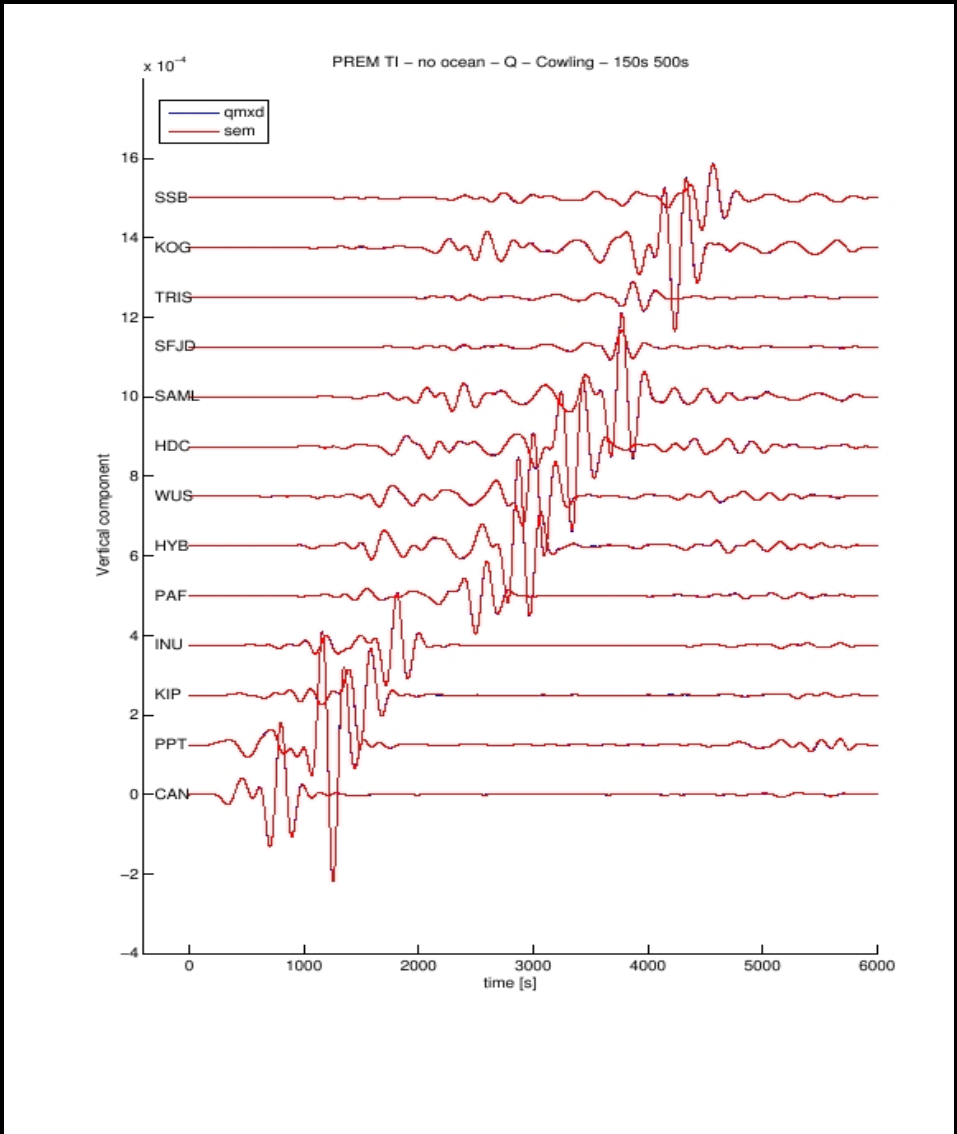
- The goal is to compute differential effects on PKP waves
- Very high resolution needed (2 to 3 seconds typically)
- Too big for current big machines (pangu: 4000 processors, Marenostrum 10000 processors) therefore we convert the mantle to acoustic instead of elastic and remove the crust because we are only interested in differential effects on P phases
- We keep an elastic anisotropic medium in the inner core only
- We can then design a mesh that is accurate down to periods of 2 seconds for P waves and that fits on 2166 processors (6 blocks of N x N slices, with N = 19)



- The mesh contains 126 billion points (the “equivalent” of a 5000 x 5000 x 5000 grid); We use 50000 time steps in 60 hours of CPU on 2166 processors in Barcelona. Total memory used is 3.5 terabytes.
- Calculations with v4.0 (acoustic mantle, no crust) are finished, the code performed well and performance levels were very satisfactory; the analysis of the seismograms is under way

Normal mode benchmark





Conclusions and future work

- Conclusions :
  - New balanced and optimized mesh
  - Balanced code (number of instructions and cache misses)
  - Far fewer cache misses => program runs faster
  - No more I/O problems on distributed file systems
  - No need for BLAS3 or non-blocking MPI
- Future work :
  - 4 doublings => need 2 basic bricks too construct one symmetrized doubling brick => 32 elements block limitation => NEX_XI / NPROC_XI = 32*x. In v4.1, we will get rid of this limitation to obtain blocks of 16 elements as in v3.6.
  - Split central cube in 2 with MPI to have better load balancing and remove bottleneck (many slices send to one processor)
  - Merge the mesher and the solver (not simple: static arrays => problem of memory reuse).
  - Debug Cuthill-McKee side effects.

SPECFEM3D Portal

COMPUTATIONAL INFRASTRUCTURE FOR GEODYNAMICS
CALIFORNIA INSTITUTE OF TECHNOLOGY

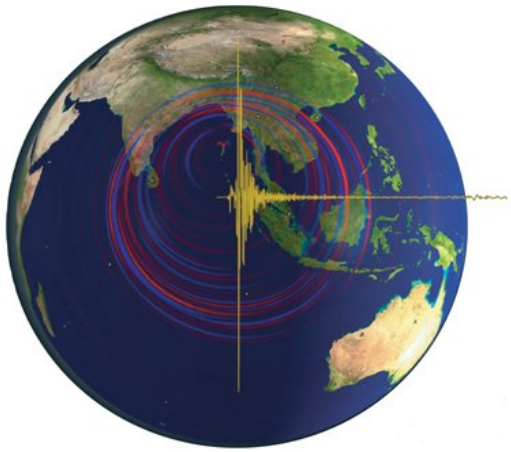
SPECFEM 3D GLOBE

Web Portal

Version 3.6.1

Login

Are you a new user? If so, please register.



Sponsored by:

CIG

Seismological Laboratory
California Institute of Technology

NSF

TeraGrid