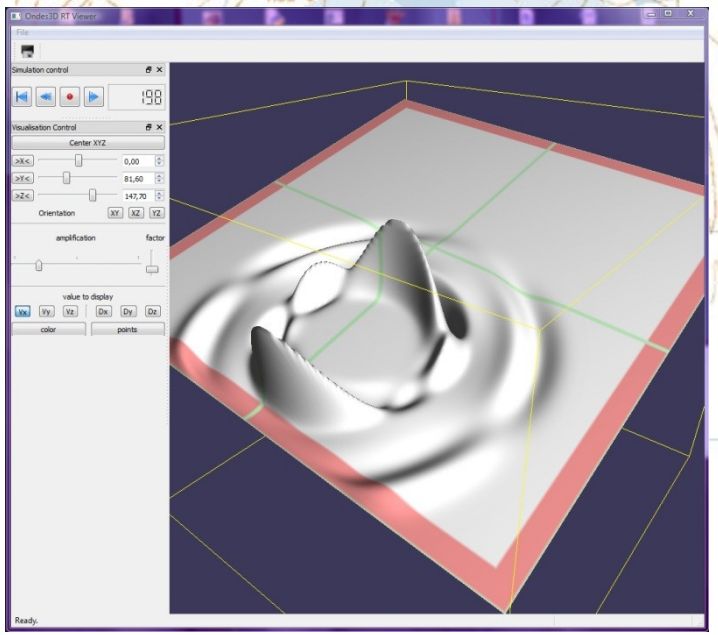
Ondes3D

ONDE3D is a 4th order FD code that simulates seismic wave propagation in an elastic medium.

It implements C-PML absorbing layers from D.Komatitsch and R.Martin (Geophysics, 2007)

It has been written by Hideo Aochi and Ariane Ducellier from BRGM.

It implements C-PML absorbing layers from D.Komatitsch and R.Martin (Geophysics, 2007)

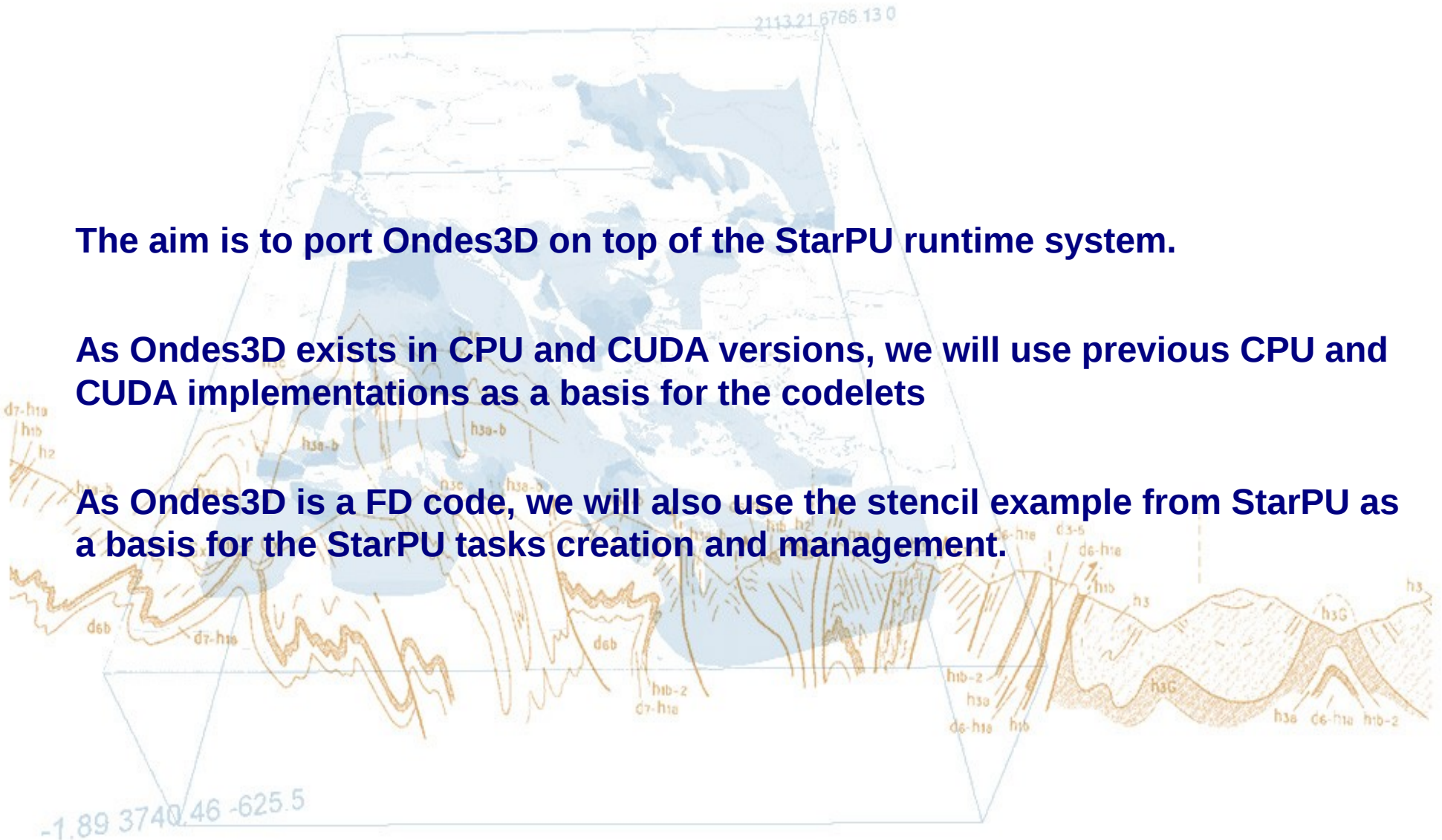


It has been parallelized using MPI & OpenMP and ported to CUDA + MPI (Geophys. J. Int. 2010)

GPGPU version gave a speed-up of ~ 40x (one GPU proc compared to one CPU core).

Geoscience for a sustainable Earth
brgm

Ondes3D / StarPu



The aim is to port Ondes3D on top of the StarPU runtime system.

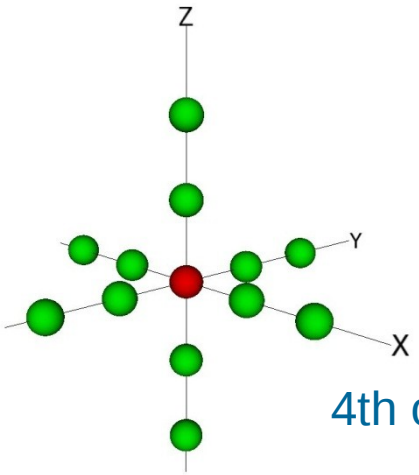
As Ondes3D exists in CPU and CUDA versions, we will use previous CPU and CUDA implementations as a basis for the codelets

As Ondes3D is a FD code, we will also use the stencil example from StarPU as a basis for the StarPU tasks creation and management.

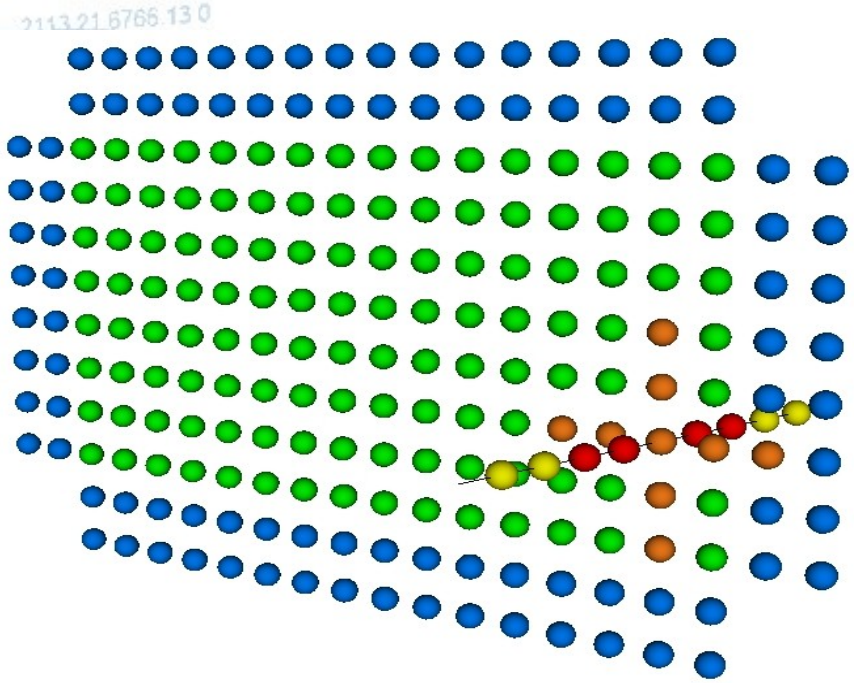
Geoscience for a sustainable Earth

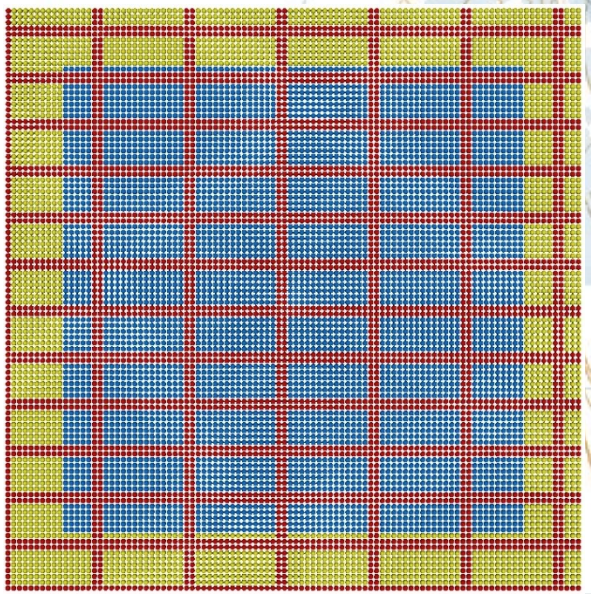
brgm

ONDES3D CUDA version



4th order FD stencil





CUDA Block : a block of threads sharing memory

A sliding window algorithm
(P. Micikevicius 2009)

CUDA Grid : a set of block with asynchronous execution

Geoscience for a sustainable Earth
brgm

Ondes3D / StarPu

StarPU uses task-level parallelism -> we need to decompose all the simulation into basic tasks less dependent as possible of each other.

Use data parallelism to decompose the domain into a set of blocks with asynchronous execution, to increase the number of independant tasks.

Sequential version of Ondes3d : Code structure

```
<initializations/>
<time loop>
        <update source momentum/>
        <compute velocity vector from stress tensor/>
        <compute stress tensor from velocity vector/>
        <record seismograms/>
</time loop>
```

Geoscience for a sustainable Earth

brgm

Parallel ONDES3D : Tasks

Let's look further at the code structure for a parallel implementation (CUDA+MPI): Data are no more located in the same place and boundaries have to be exchanged.

Update source momentum

Receive stress boundaries from neighbours

Load stress boundaries to GPU memory

Compute velocity from stress

Save velocity boundaries from GPU memory to MPI buffers

Send velocity boundaries to neighbours

Receive velocity boundaries from neighbours

Copy velocity boundaries to GPU memory

Compute stress from velocity

Save stress boundaries from GPU memory to MPI buffers

Send stress boundaries to neighbours

Record seismograms

If we decompose the domain into a set of blocks for StarPU, and we can no more assure that the data are located in the same place, these are the tasks we will have to achieve for each iteration.

As the domain we use are often much longer & larger than deep, the domain will be cut in X and Y directions.

Geoscience for a sustainable Earth

brgm

Ondes3D :

With StarPU, no need to think about data as GPU data or as CPU data.

Just define handlers binded to your data so that starPU can manage its placement. Thus, no need to explicitly copy data from or to GPU. It is completely managed by StarPU.

So the previous tasks become :

For each iteration :

Update source

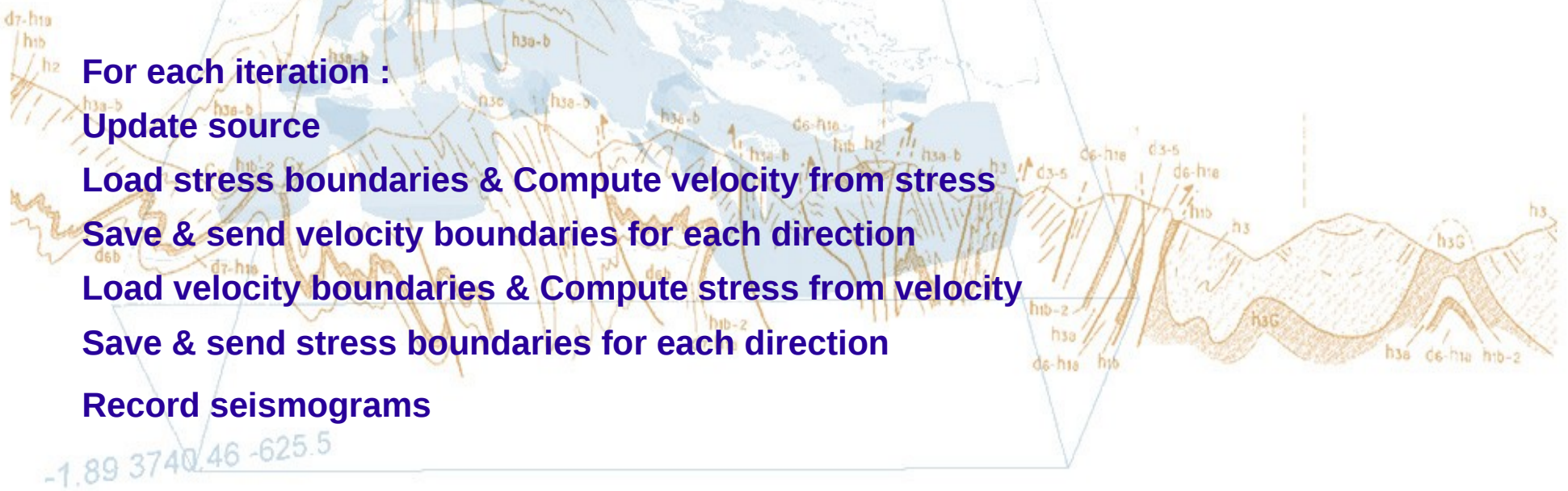
Load stress boundaries & Compute velocity from stress

Save & send velocity boundaries for each direction

Load velocity boundaries & Compute stress from velocity

Save & send stress boundaries for each direction

Record seismograms



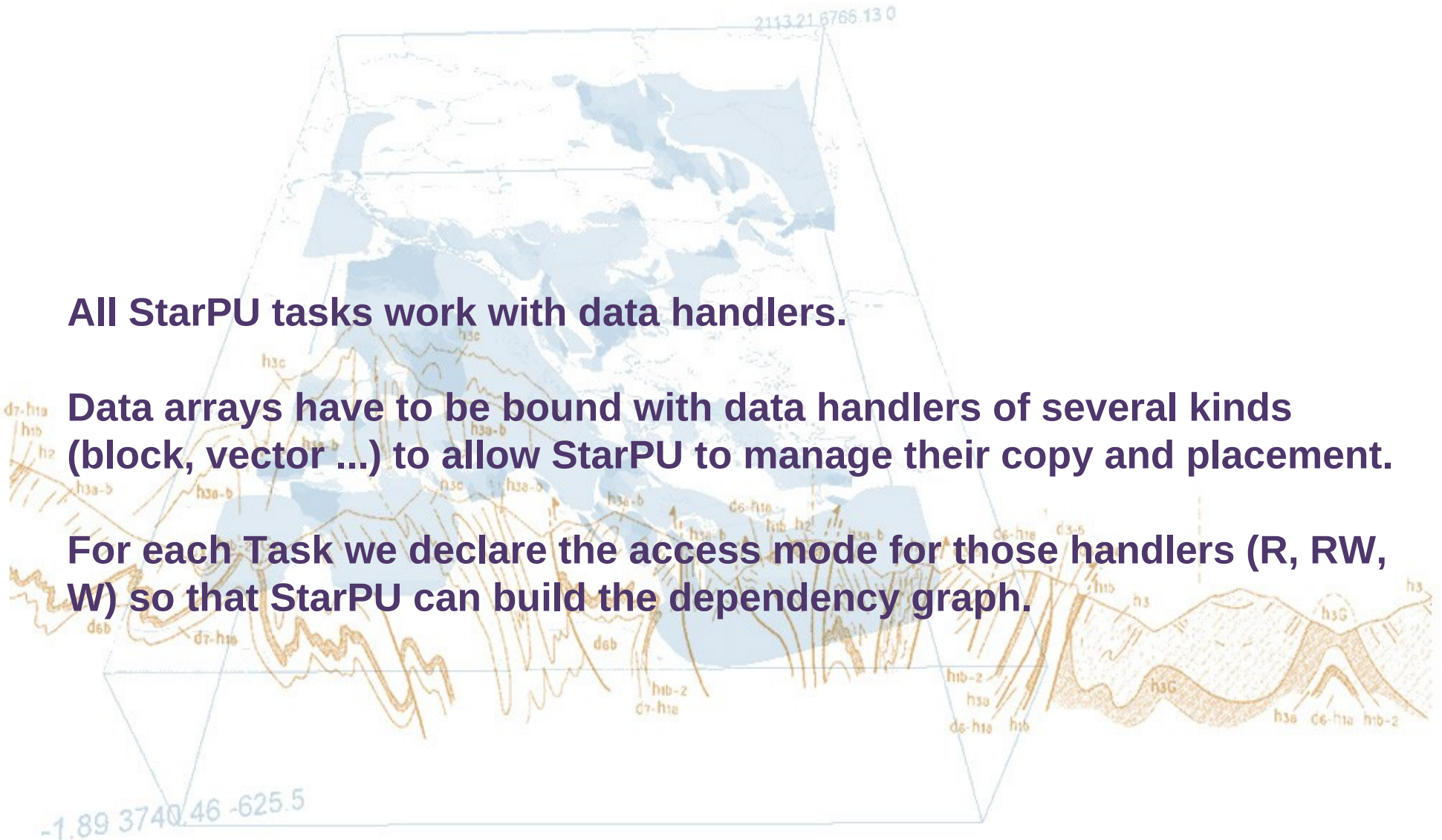
Geoscience for a sustainable Earth

brgm

Tasks creation

```
for (iter = 0; iter <= niter; iter++)
{   for (bx = 0; bx < nbx; bx++)
    {   for (by = 0; by < nby; by++)
        {
            struct block_description * block = get_block_description(bx, by);
            // I) sources update
            if (block->mpi_node == rank && block->nb_src > 0)
                create_task_update_sources(iter, bx, by, rank);

            // II) Kernel_1 : V = f(T)
            if (block->mpi_node == rank)
                create_task_compute_veloc(iter, bx, by, rank);

            // III) save boundaries(V)
            if ((block->mpi_node == rank) || (get_block_mpi_node(bx+1, by) == rank))
                create_task_save(iter, bx, by, XP, rank, VELOC);
            if ((block->mpi_node == rank) || (get_block_mpi_node(bx-1, by) == rank))
                create_task_save(iter, bx, by, XM, rank, VELOC);
            if ((block->mpi_node == rank) || (get_block_mpi_node(bx, by+1) == rank))
                create_task_save(iter, bx, by, YP, rank, VELOC);
            if ((block->mpi_node == rank) || (get_block_mpi_node(bx, by-1) == rank))
                create_task_save(iter, bx, by, YM, rank, VELOC);

            // IV) Kernel_2 : T = f(V)
            if (block->mpi_node == rank)
                create_task_compute_stress(iter, bx, by, rank);

            // V) save boundaries(T)
            if ((block->mpi_node == rank) || (get_block_mpi_node(bx+1, by) == rank))
                create_task_save(iter, bx, by, XP, rank, STRESS);
            if ((block->mpi_node == rank) || (get_block_mpi_node(bx-1, by) == rank))
                create_task_save(iter, bx, by, XM, rank, STRESS);
            if ((block->mpi_node == rank) || (get_block_mpi_node(bx, by+1) == rank))
                create_task_save(iter, bx, by, YP, rank, STRESS);
            if ((block->mpi_node == rank) || (get_block_mpi_node(bx, by-1) == rank))
                create_task_save(iter, bx, by, YM, rank, STRESS);

            // VI) record seismos
            if (block->mpi_node == rank && ((iter < niter && block->nb_sta > 0) || (iter == niter)))
                create_task_record_seismo(iter, bx, by, rank);
        }
    }
}
```

Ondes3D : data handlers



All StarPU tasks work with data handlers.

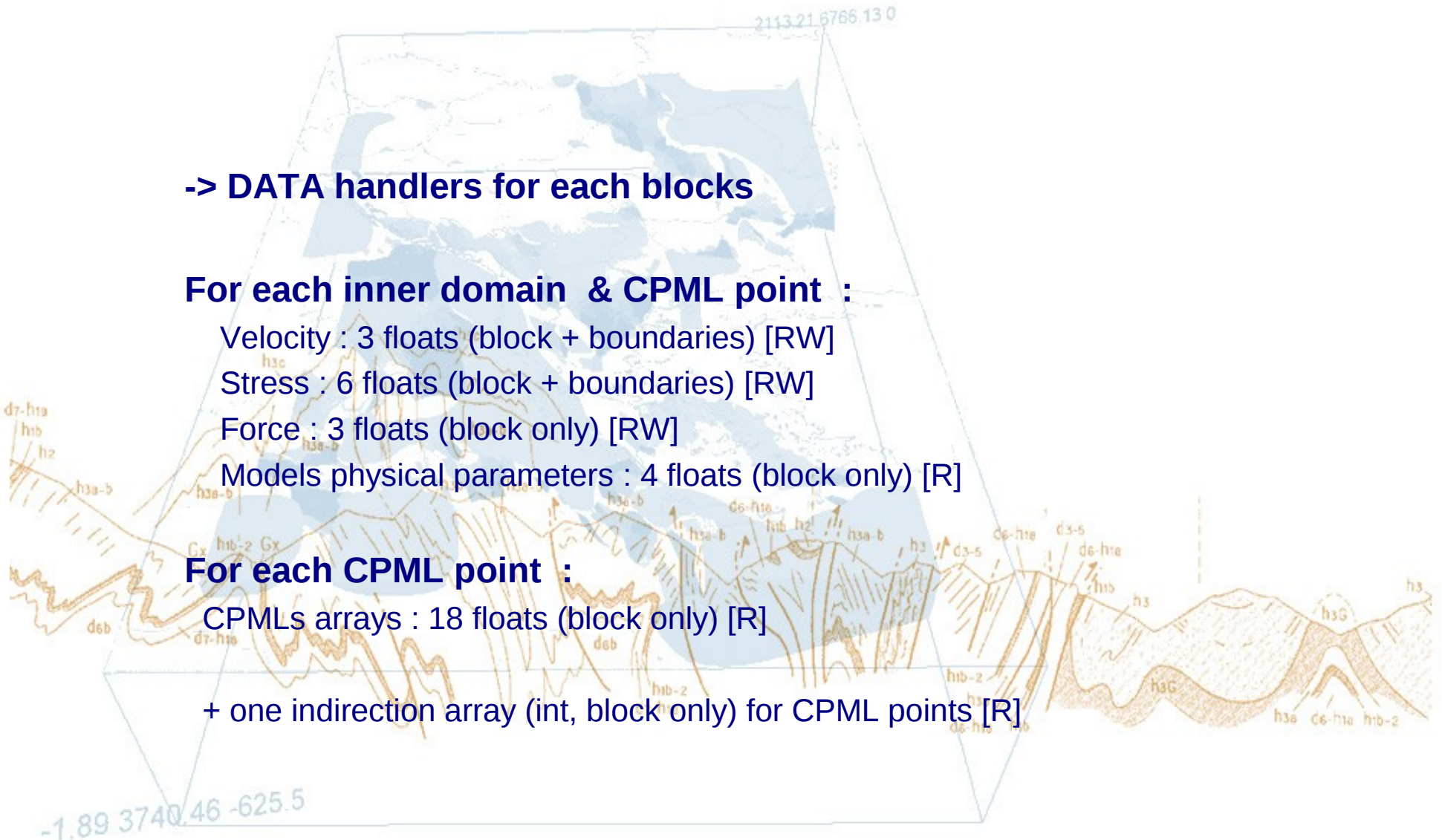
Data arrays have to be bound with data handlers of several kinds (block, vector ...) to allow StarPU to manage their copy and placement.

For each Task we declare the access mode for those handlers (R, RW, W) so that StarPU can build the dependency graph.

Geoscience for a sustainable Earth

brgm

Ondes3D : data handlers



-> DATA handlers for each blocks

For each inner domain & CPML point :

Velocity : 3 floats (block + boundaries) [RW]

Stress : 6 floats (block + boundaries) [RW]

Force : 3 floats (block only) [RW]

Models physical parameters : 4 floats (block only) [R]

For each CPML point :

CPMLs arrays : 18 floats (block only) [R]

+ one indirection array (int, block only) for CPML points [R]

Geoscience for a sustainable Earth

brgm

Ondes3D : data handlers

```
void create_task_update_sources(unsigned iter, unsigned x, unsigned y, unsigned local_rank)
{

    struct starpu_task *task = starpu_task_create();

    struct block_description *descr = get_block_description(x,y);

    // data handles
    task->handles[0] = descr->force_handle[0];
    task->handles[1] = descr->force_handle[1];
    task->handles[2] = descr->force_handle[2];

    task->cl = &cl_update_source;
    task->cl_arg = arg;

    starpu_task_submit(task);
}

struct starpu_codelet cl_update_source =
{
    .where = 0 |
#ifdef STARPU_USE_CUDA
        STARPU_CUDA|
#endif
        STARPU_CPU,
    .cpu_funcs = {update_sources_cpu, NULL},
#ifdef STARPU_USE_CUDA
    .cuda_funcs = {update_sources_cuda, NULL},
#endif
    .model = &cl_update_source_model,
    .nbuffers = 3,
    .modes = {STARPU_RW, STARPU_RW, STARPU_RW}
};

static void update_sources_cpu(void *descr[], void *arg_)
{
    struct block_description* block = (struct block_description*)arg_;
    int workerid = starpu_worker_get_id();

    // get Data pointers
    struct starpu_block_interface *b_fx = (struct starpu_block_interface *)descr[0];
    struct starpu_block_interface *b_fy = (struct starpu_block_interface *)descr[1];
    struct starpu_block_interface *b_fz = (struct starpu_block_interface *)descr[2];

    float *fx = (float *)b_fx->ptr;
    float *fy = (float *)b_fy->ptr;
    float *fz = (float *)b_fz->ptr;

    ...

}
```

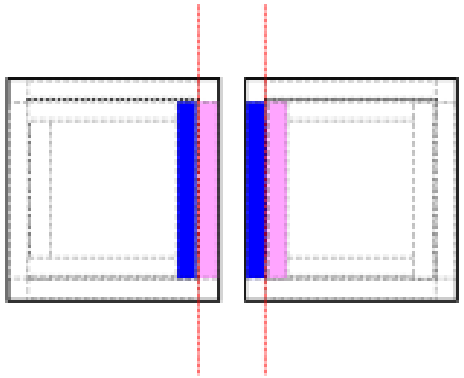
We attach to task data handlers (previously allocated with starpu_malloc)

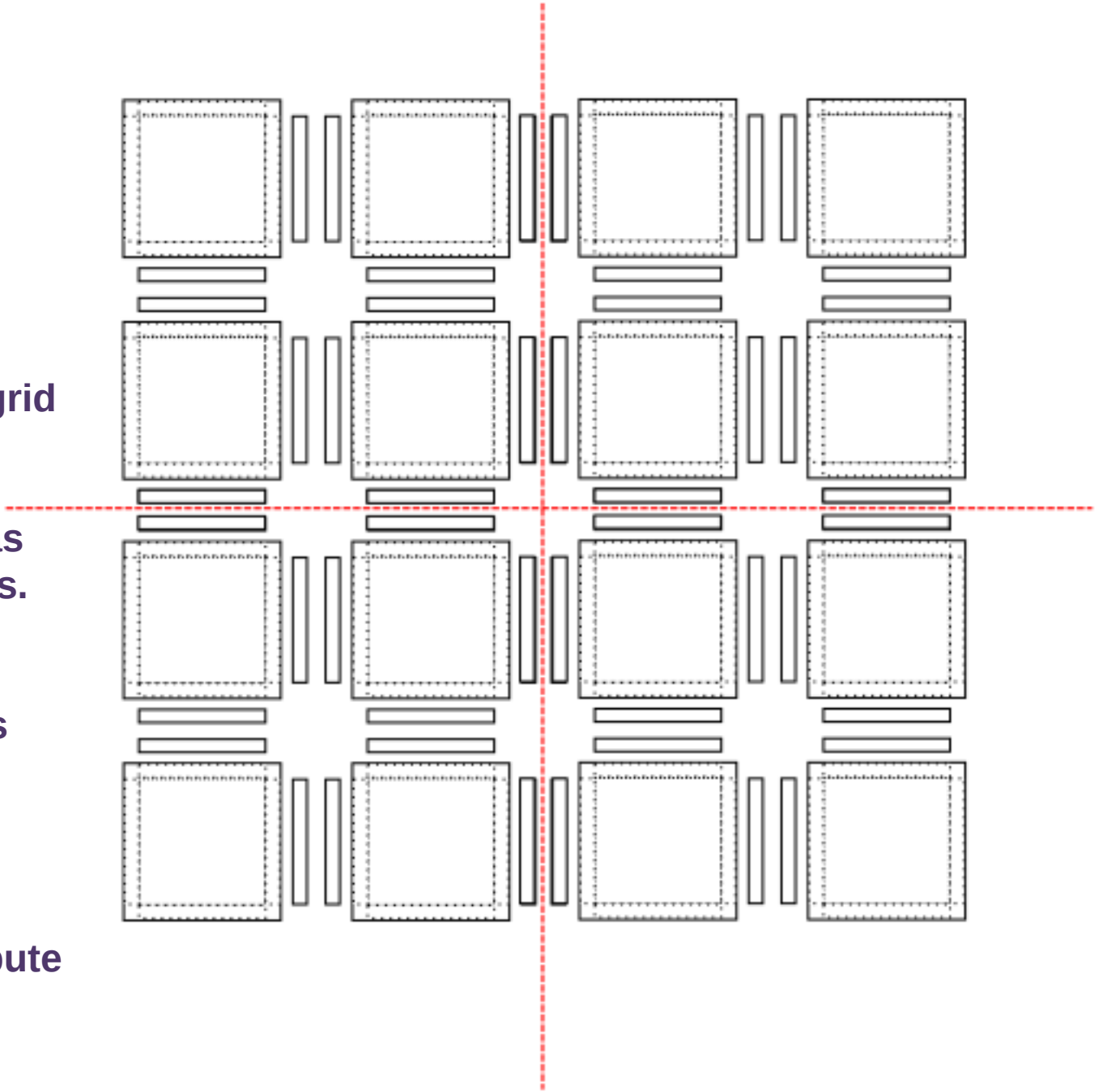
We recover pointers on the data through these handlers

Geoscience for a sustainable Earth

brgm

Ondes3D : partitionning





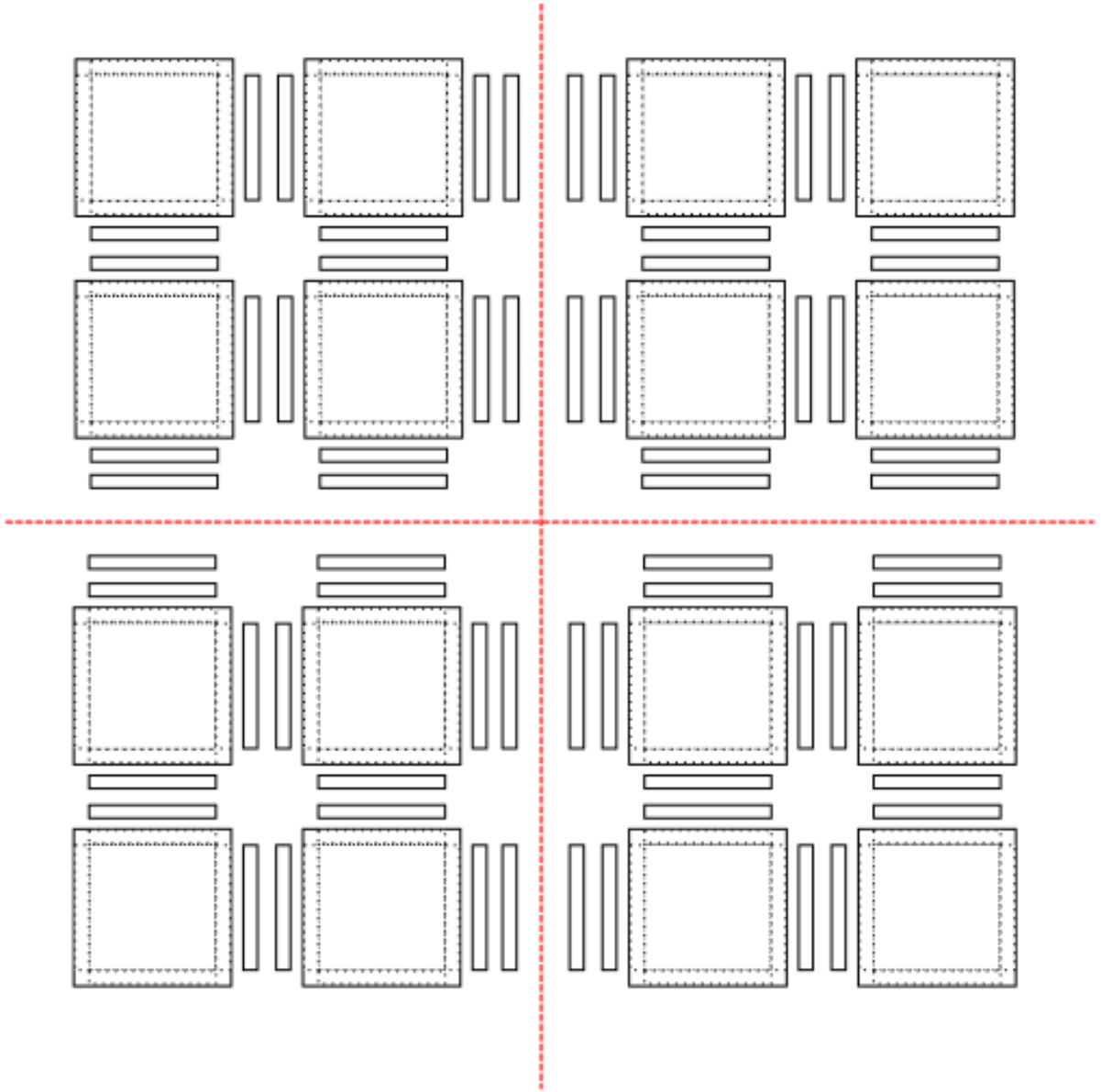
The domain is divided into a grid of block.

Stencil code => each block has ghost points on its boundaries.

Boundaries need to be exchanged between adjacents blocks at each iteration.

Buffers have been added to uncouple the dependency between save and load&compute for 2 neighbours blocks.

Ondes3D : MPI partitionning

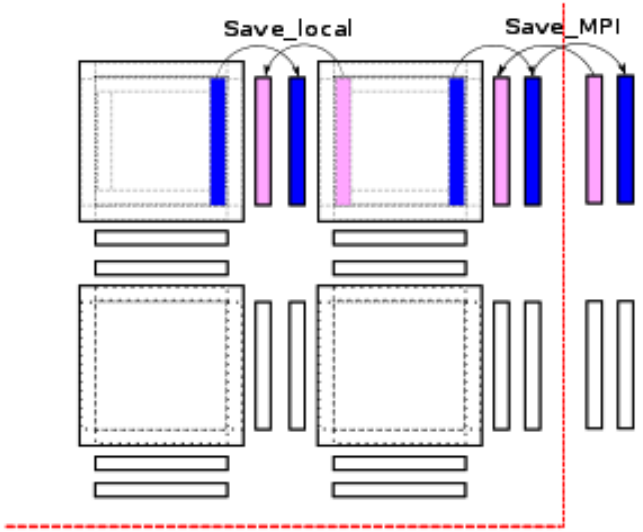


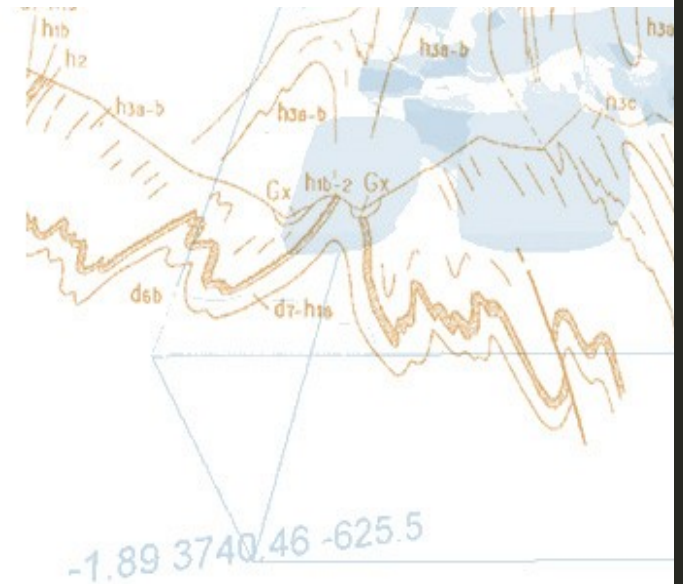
Buffers of the neighbours blocks have been added on the MPI boundary, to uncouple local save process and save process through MPI.

Geoscience for a sustainable Earth

brgm

Ondes3D : save task





```
void create_task_save(unsigned iter, unsigned x, unsigned y, int dir, unsigned local_rank, data_type type)
{
    // DGN : ATTENTION aux bords du domaine !!!!
    unsigned node = get_block_mpi_node(x, y);
    int node_d;
    int time_to_stop = 0;

    int xd = x, yd = y;
    switch(dir)
    {
        case XP :   xd++;
                    break;
        case XM :   xd--;
                    break;
        case YP :   yd++;
                    break;
        case YM :   yd--;
                    break;
    }

    // no boundary saving if on domain's border
    if (!(xd < 0 || yd < 0 || xd >= get_nbx() || yd >= get_nby()))
    {
        node_d = get_block_mpi_node(xd, yd);

#ifdef STARPU_USE_MPI
        if (node == local_rank)
        {
            /* Save data from update */
            create_task_save_local(iter, x, y, dir, local_rank, type);
            if (node_d != local_rank)
            { /* We have to send the data */
                create_task_save_mpi_send(iter, x, y, dir, local_rank, type);
            }
        }
        else
        {   /* node_d != local_rank, this MPI node doesn't have the saved data */
            if (node_d == local_rank)
            {
                create_task_save_mpi_recv(iter, x, y, dir, local_rank, type);
            }
            else
            {
                STARPU_ASSERT(0);
            }
        }
#else /* !STARPU_USE_MPI */
        STARPU_ASSERT((node == local_rank) && (node_d == local_rank));
        create_task_save_local(iter, x, y, dir, local_rank, type);
#endif /* STARPU_USE_MPI */
    }
}
```

Optimisations : work in progress ...

- partitionning

models rather plate-shaped : cutting in XY direction instead of Z :

- decreases the overall volume of copy and the size of MPI messages
- increase the number of save tasks (better for scheduling?)
- but : copy of non adjacent memory chunks : increase overhead cost of memcpy.

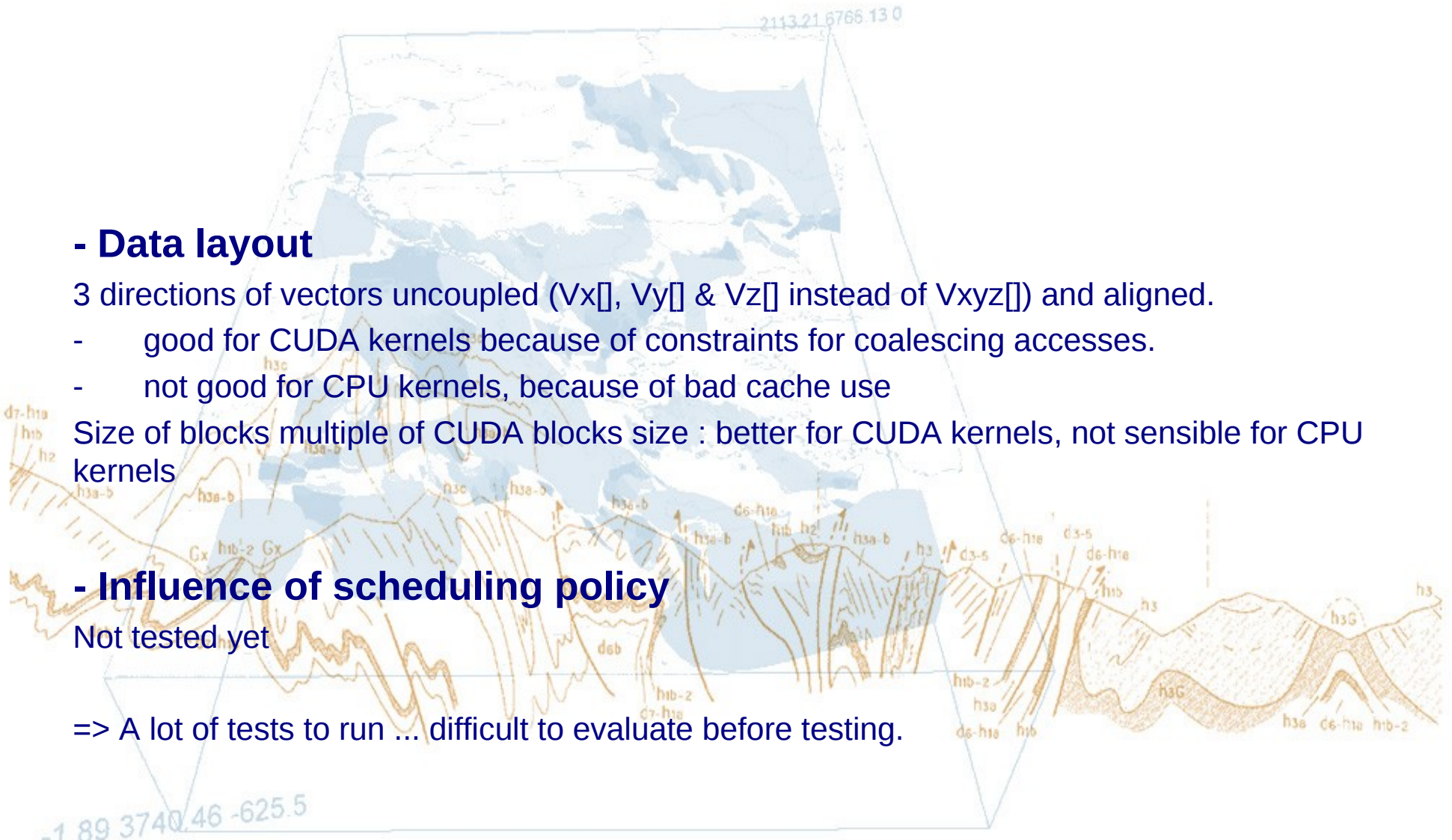
- size of blocks :

- Large enough to keep the number of CUDA blocks large enough to cover the global memory accesses.
- Small enough to keep a large number of tasks so that scheduling can be efficient
- As the size of blocks decrease, the memory use increase dramatically

=> influence of granularity on performance -> need to run much more tests ...

Geoscience for a sustainable Earth
brgm

Optimisations : work in progress ...



- Data layout

3 directions of vectors uncoupled (Vx[], Vy[] & Vz[] instead of Vxyz[]) and aligned.

- good for CUDA kernels because of constraints for coalescing accesses.
- not good for CPU kernels, because of bad cache use

Size of blocks multiple of CUDA blocks size : better for CUDA kernels, not sensible for CPU kernels

- Influence of scheduling policy

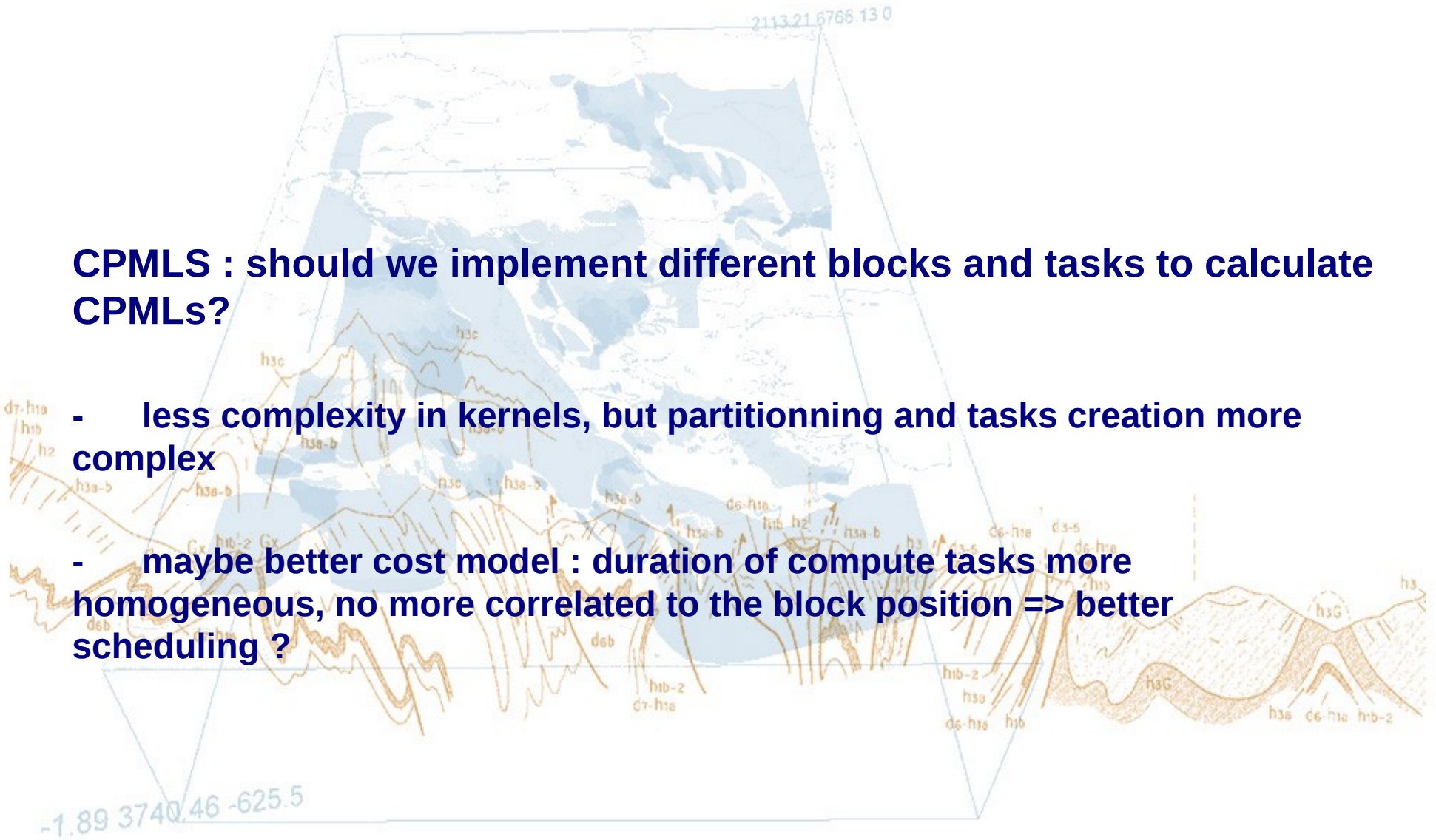
Not tested yet

=> A lot of tests to run ... difficult to evaluate before testing.

Geoscience for a sustainable Earth

brgm

Optimisations : work in progress ...



CPMLS : should we implement different blocks and tasks to calculate CPMLs?

- **less complexity in kernels, but partitionning and tasks creation more complex**

- **maybe better cost model : duration of compute tasks more homogeneous, no more correlated to the block position => better scheduling ?**

Geoscience for a sustainable Earth

brgm