Exploring XMP programming model applied to Seismic Imaging application

Introduction

Total at a glance:

- **96 000 employees** in more than **130 countries**
- **Upstream operations** (oil and gas exploration, development and production, Liquefied Natural Gas)
- **Downstream operations** (refining, marketing and the trading and shipping of crude oil and petroleum products)
- **Base chemicals** (petrochemicals and fertilizers) and **specialty chemicals**
- **Renewable energies** (especially solar and biomass)

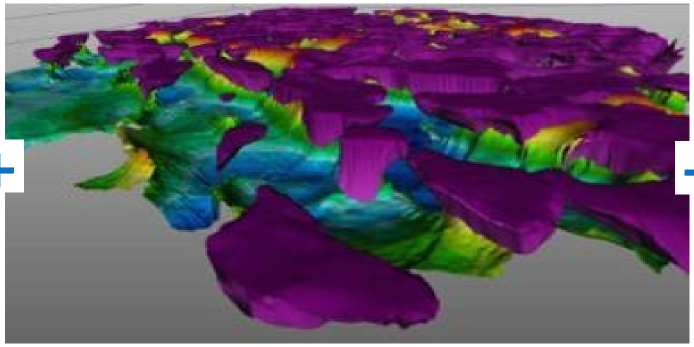
Contents

Seismic imaging and RTM principle

Seismic imaging and RTM principle

Seismic imaging:

Data + **Velocity model (inverse problem)** + **Algorithms + HPC**

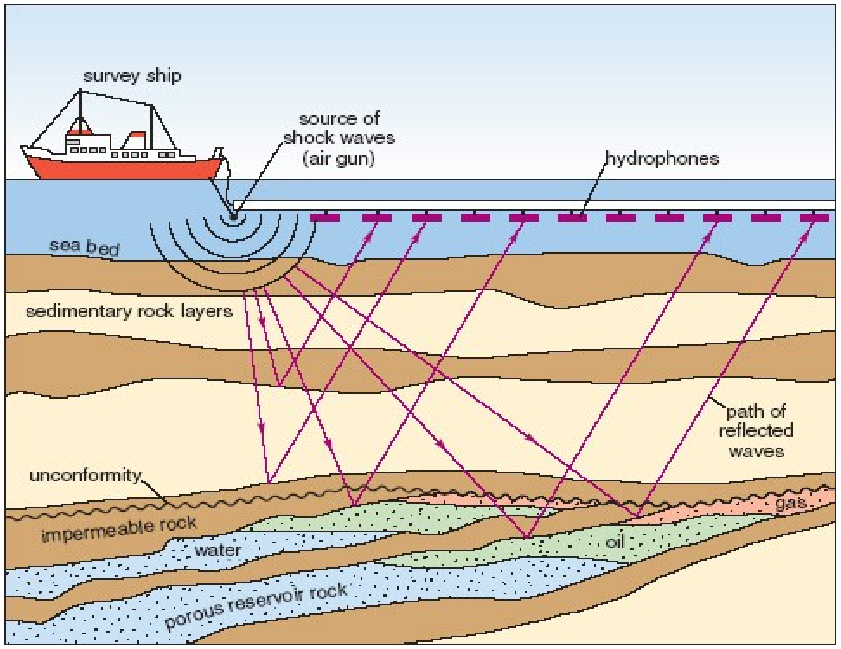


One algorithm: the Reverse Time Migration.

Seismic imaging and RTM principle

Seismic acquisition:

Contrast in terms of seismic waves velocity and density causes **reflections** of seismic waves.

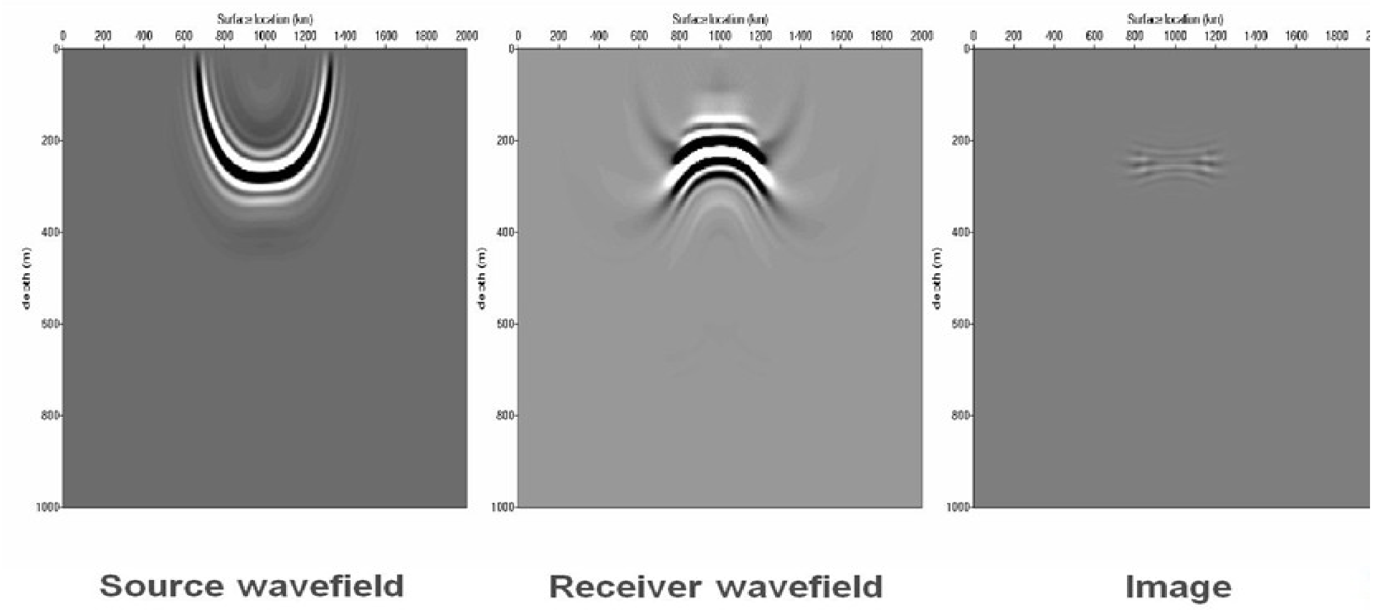


Seismic imaging and RTM principle

RTM: Consists in solving the **wave equation**.

$$[1/c2 \ \ \partial 2 /\partial t2 - \Delta]\, u(\mathbf{x},t) = 0$$

1. Propagation of the source wave-field (t=0,T)
2. Retro-propagation of the receivers wave-field (t=T,0)
   Reversibility of the wave equation!
3. Imaging condition: cross-correlation



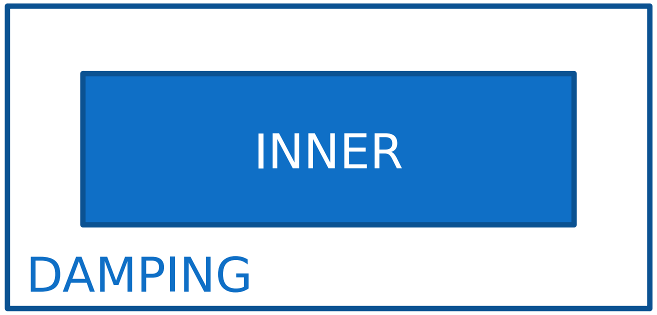
Seismic imaging and RTM principle

<u>Infinite physical domain</u>
(underground)

<u>Finite computational domain</u>
(3D rectangular box)

Damping effect (artificial absorbing layer near the walls to avoid artificial reflections)

<u>Two different wave equations</u> are discretized depending on the domain

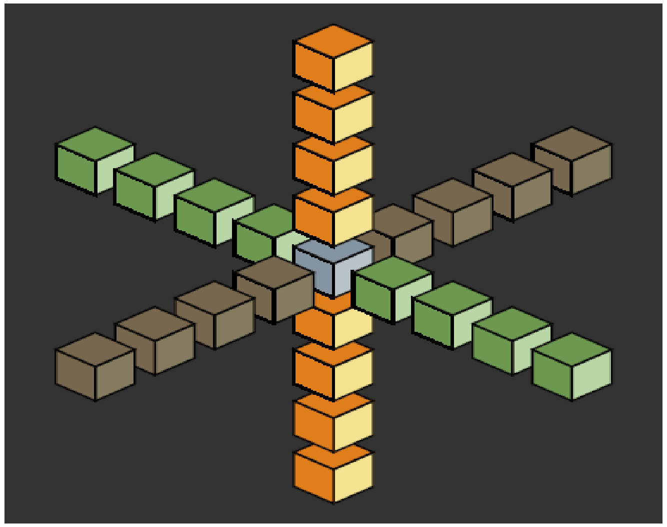


Seismic imaging and RTM principle

RTM code consists in a Finite Difference discretization, explicit time scheme

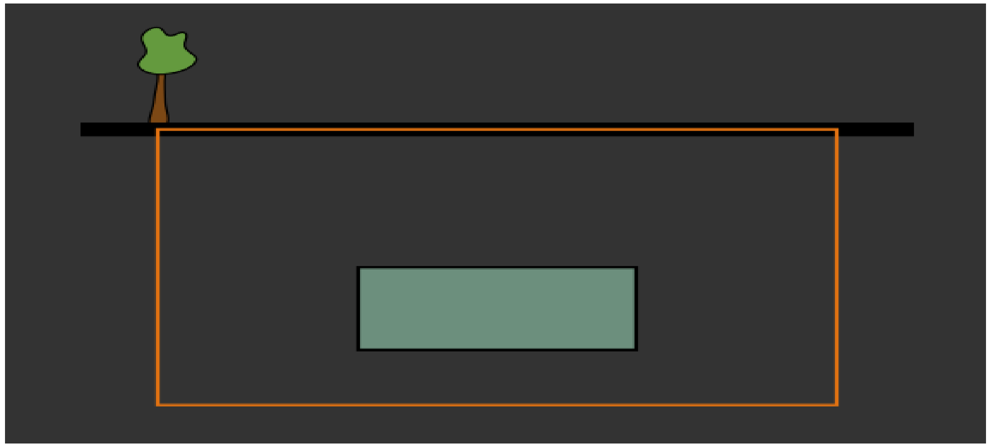
Stencil:

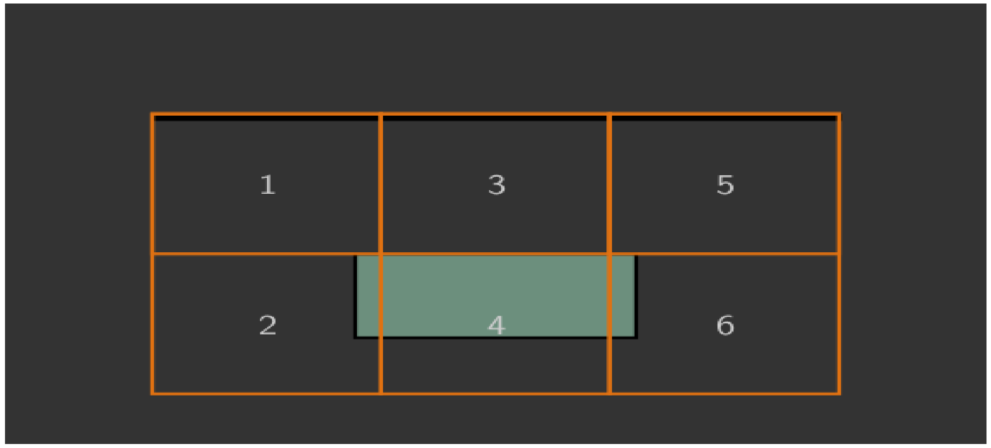
- two points in time
- up to eight points in each direction in space



Seismic imaging and RTM principle

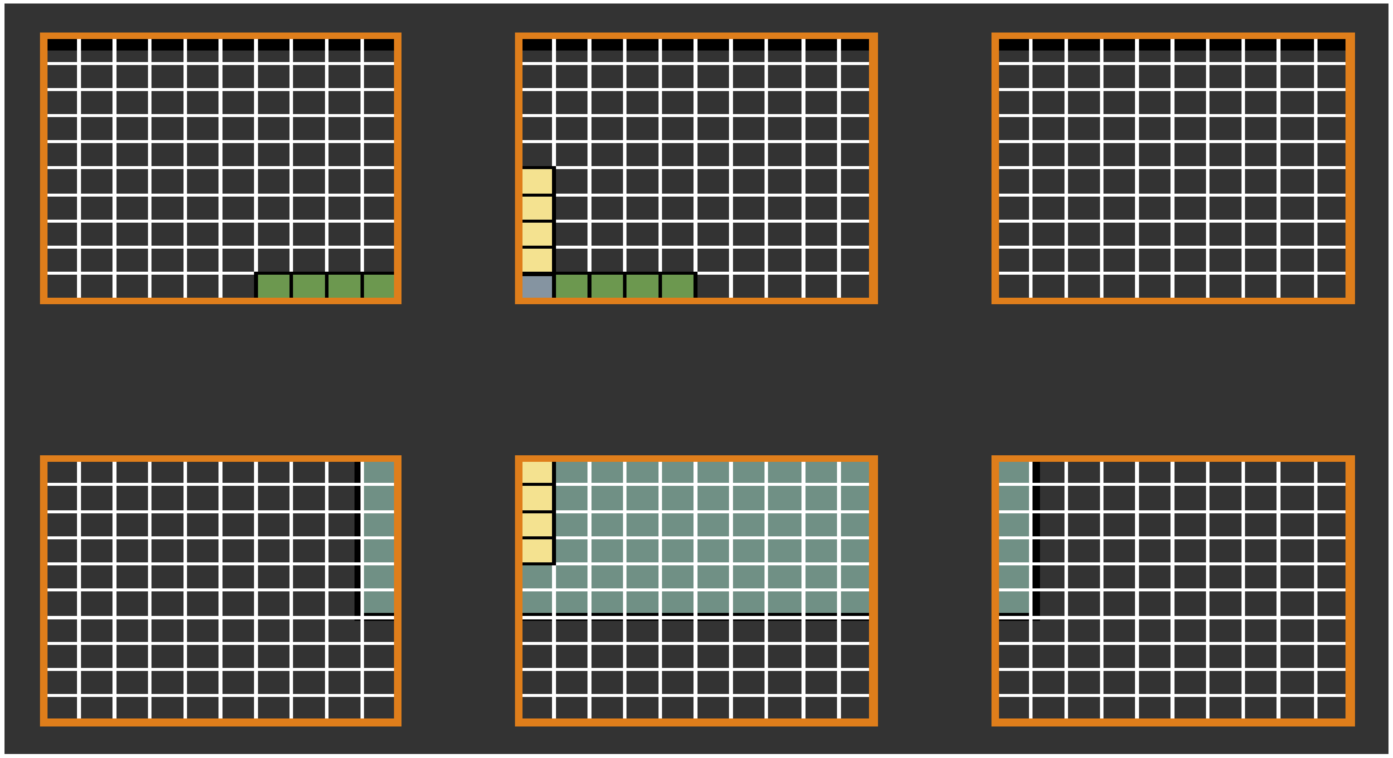
Grid decomposition:





Seismic imaging and RTM principle

Communications:



Seismic imaging and RTM principle

Finite Difference scheme:

- Explicit time scheme.
- Forward or Backward: same discretization but reverse time loop (t=0,T or t=T,0).
- The domains (inner and damping) are determined at the beginning of each shot, no « if » statement in the Finite Difference subroutines.

For XMP:

- Regular 3D grid-decomposition
- Communications only between two neighbors

XMP

XMP advantages

XMP:

- Easy to transform a sequential program into a parallel program.

- Optimized to the architecture, and no necessity to rewrite the code to change the parallelism language (MPI,...)

Two XMP versions

In my case: XMP FORTRAN

I already had a parallel code so I am doing two different versions using XMP:

- I came back to a sequential version and all the parallelism is implemented with XMP (grid decomposition, loops, communication)
- I kept the parallel code and only the communications are performed with *reflect*.

Two XMP versions

In my case: XMP FORTRAN

I already had a parallel code so I am doing two different versions using XMP:

- I came back to a sequential version and all the parallelism is implemented with XMP (grid decomposition, loops, communication)

- I kept the parallel code and only the communications are performed with *reflect*.

XMP at a glance

How to parallelize a code using XMP?

Initialization of XMP:

```
!$xmp nodes node(*,*,*)     ! defined by environment variables
!$xmp template temp_3d(:,:,:)
!$xmp distribute temp_3d(gblock(*),gblock(*),gblock(*)) onto node
```

Declaration of the vectors distributed over the grid:

```
real, dimension (:, :, :), allocatable :: u0
!$xmp align (i,j,k) with temp_3d(i,j,k) :: u0
!$xmp shadow u0(SHADOWX, SHADOWY, SHADOWZ)     ! macros
allocate (u0 (xcompmin_g : xcompmax_g, ...) )
```

XMP computational directives:

```
!$xmp array on temp_3d(xcompmin_g:xcompmax_g, ...)
!$xmp loop(i,j,k) on temp_3d(i,j,k)
!$xmp task on temp_3d(ix:jx,iy:jy,iz:jz)
```

XMP communication directives:

```
!$xmp reflect (u0)
!$xmp bcast (xmax,ymax,zmax) on node
```

XMP programming model

Fortran 95 base language:

Some intrinsic modules do not exist.

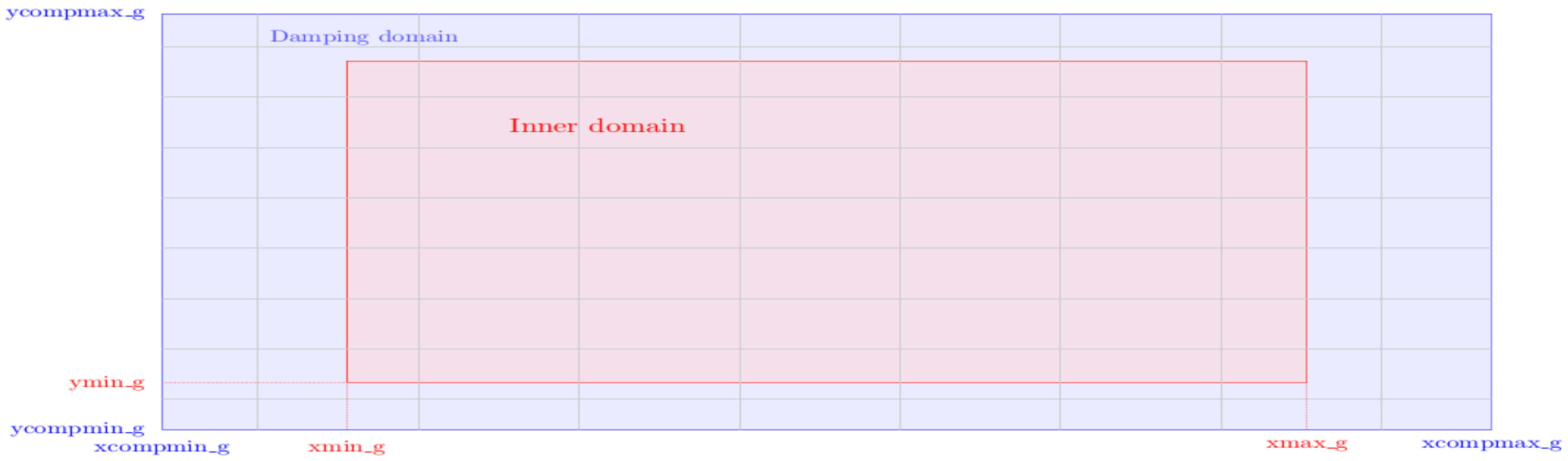
Mixed FORTRAN and C programming (ISO_C_BINDING and BIND are not available).

Allocatable array cannot be passed to subroutines. Necessity to declare them as global variables, and copy of the subroutines which were called with different allocatable arrays.

XMP programming model

Initialization of XMP:

!$xmp nodes node(,*,*)* ***! defined by environment variables***
!$xmp template temp_3d(:,:,:)
!$xmp distribute temp_3d(gblock(*),gblock(*),gblock(*)) onto node



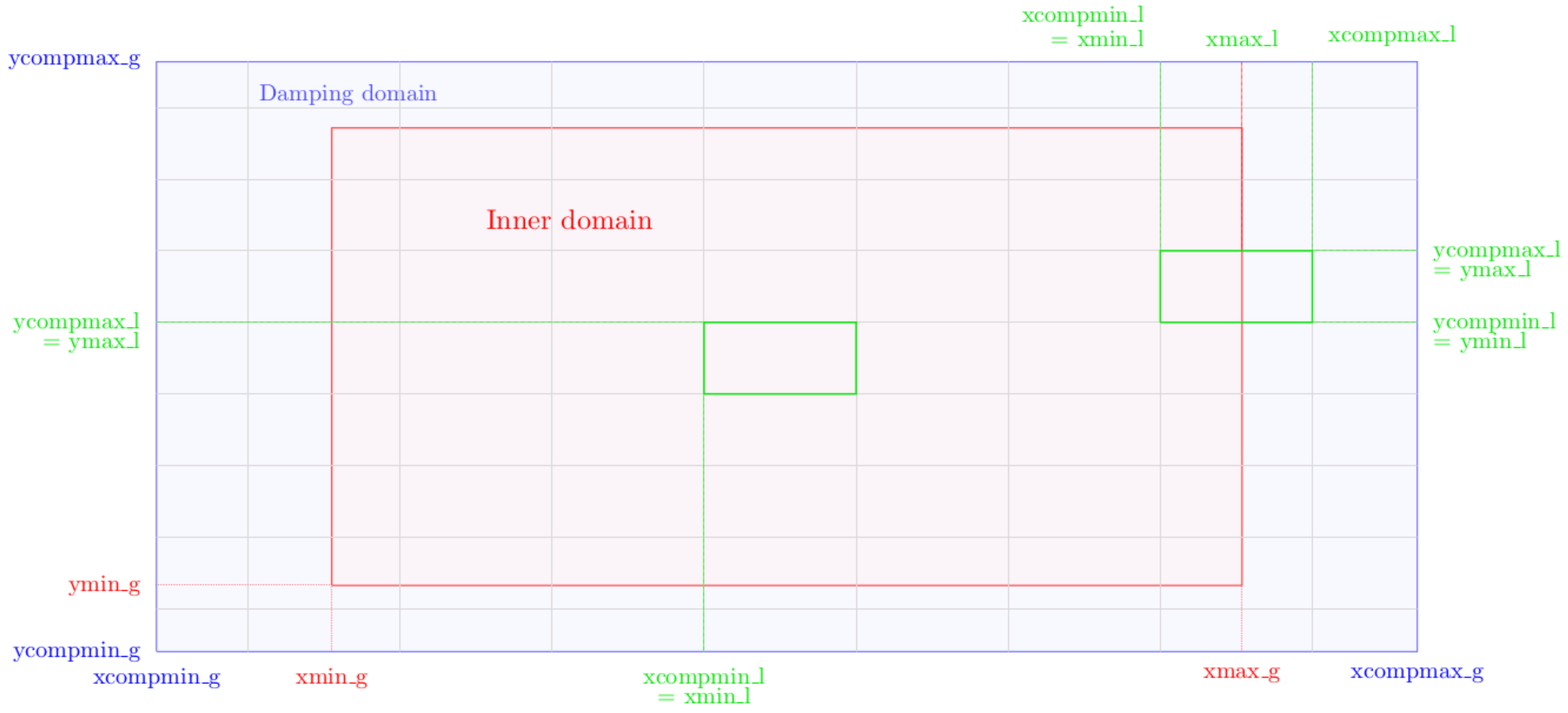
After the determination of the domain size:

!$xmp template_fix(gblock(Mx), gblock(My), gblock(Mz)) &
temp_3d(xcompmin_g:xcompmax_g,ycompmin_g:ycompmax_g,...)

XMP programming model

MPI domain description.

Definition of global and local variables to describe the grid decomposition : **local view programming**.



XMP programming model

To distribute a vector, in the MPI version, the vector is allocated with the local sizes; whereas with XMP the vector is aligned with XMP construct and allocated with the global size.

MPI:

real, dimension(:, :, :), allocatable, save :: u0
allocate (u0 (xmemmin_l : xmaxmem_l, ...))

XMP:

```
real, dimension (:, :, :), allocatable, save :: u0
!$xmp align (i,j,k) with temp_3d(i,j,k) :: u0
!$xmp shadow u0(SHADOWX, SHADOWY, SHADOWZ)
allocate (u0 (xcompmin_g : xcompmax_g, ...) )
```

XMP programming model

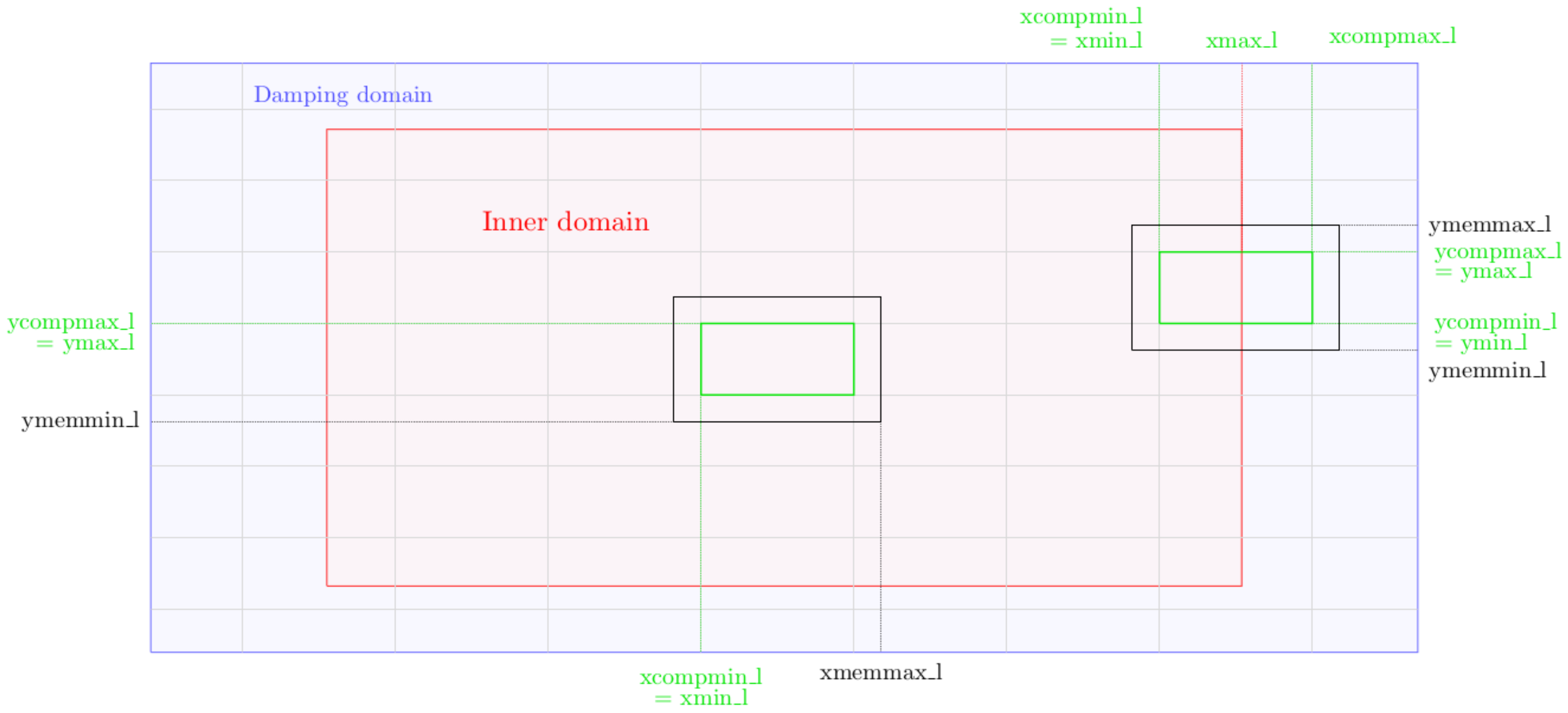
XMP computational directives:

```
!$xmp array on temp_3d(xmin_g:xmax_g, ymin_g:ymax_g, ...)
u0 (xmin_g:xmax_g, ymin_g:ymax_g, ...) = 1.0

!$xmp loop(i,j,k) on temp_3d(i,j,k)      !really easy for inner/damping
do i = xmin_g, xmax_g
   do j = ymin_g, ymax_g
     do z = zmin_g, zmax_g
       ....
     end do
  end do
end do

!$xmp task on node(1,1,1)
  open(ius, FILE=my_file)
   ...
!$xmp end task
```

XMP communications

MPI Communications.

More local variables are defined and *MPI_Isend*.

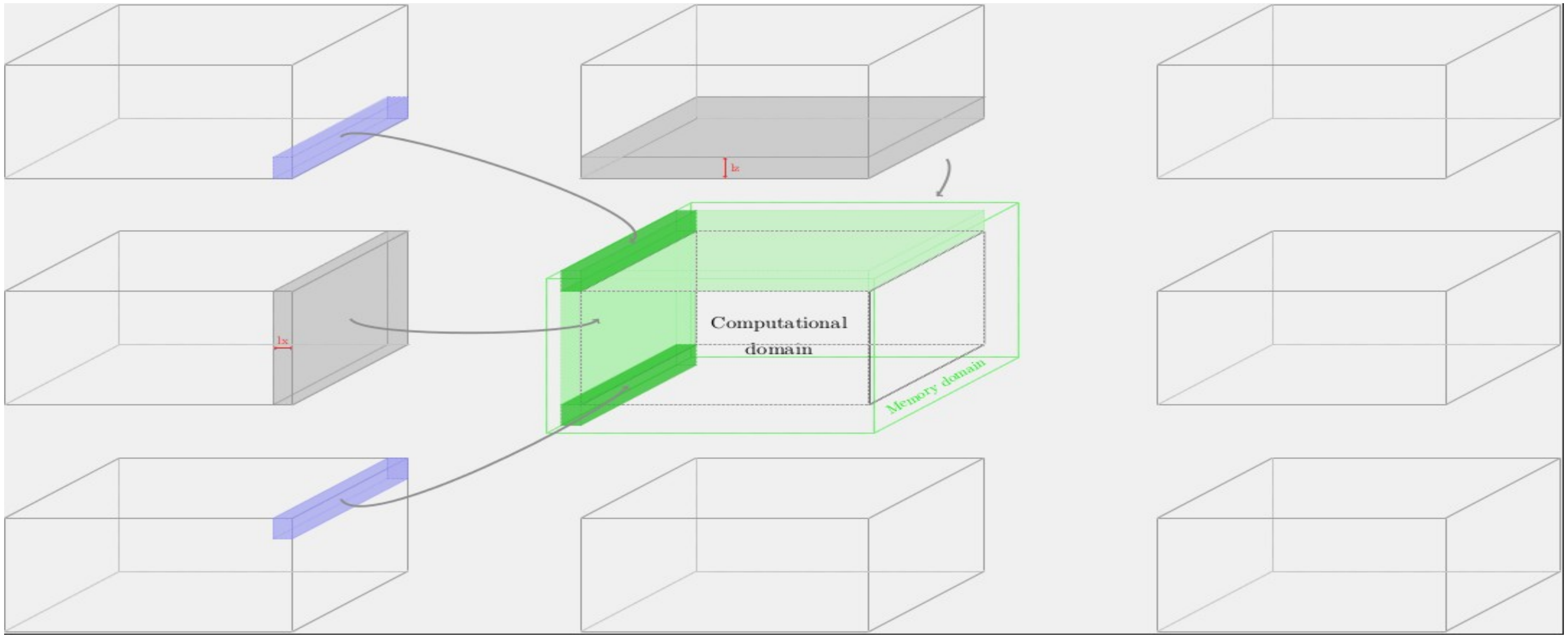


XMP communications

XMP Communications.

Shadow: specifies the width of the shadow area used to communicate the neighbor elements.

Reflect: updates the shadow elements.



Summary

A few and precise directives to parallelize a code. But it can be complicated in some case to do specific operation with XMP.

For example, how to initialize the shadow areas at the global lower and upper bounds without periodic reflect?

It is not possible to use an aligned array without an XMP directive. Even when local variables are used, it is mandatory to precise a directive like “!$xmp task on”.

Two XMP versions

In my case: XMP FORTRAN

I already had a parallel code so I am doing two different versions using XMP:

- I came back to a sequential version and all the parallelism is implemented with XMP (grid decomposition, loops, communication)
- I kept the parallel code and only the communications are performed with *reflect*.

XMP only in communications

The RTM code is written such that the kernel is independent of the language of parallelism.

During the communications: the vector is copied into a sending buffer, then it is send with MPI or CAF, and finally the buffer is copied back into the vector.

The same is performed for this version with XMP. One array is aligned, and is used when one vector needs to update its shadow.

Copy || reflect || copy

Results

<u>My results:</u> Comparison of MPI/XMP.

	MPI Fortran 2003 mpifrtpx	**MPI Fortran 95 xmpf90**	**XMP Fortran 95 xmpf90**
Inner	94s< . <257s	10s< . <263s	11s< . <258s
Damping	143s< . <357s	141s< . <368s	138s< . <355s
Kernel	315s< . <405s	325s< . <404s	325s< . <397s
Copy Send	3.0s< . <6.3s	3.3s< . <6.1s	**8.5s< . <84s**
Transfer	2.1s< . <100s	2.5s< . <89s	**13s< . <14s**
Copy Recv	1.4s< . <4.7s	1.7s< . <4.6s	**9.4s< . <10s**
Total	**423s**	**422s**	**435s**

==> Synchronization in !$xmp task

Feedbacks

<u>What Total has interests in</u>:

- Is it difficult/long to implement a code using XMP?

I work on it since March (not full time, and I have waited for some functions to be implemented).

- And compared to MPI?

XMP might be really easier to implement than MPI. Under the condition that a more detailed documentation exists.

- Is XMP compatible with their programming standards?

Most of it, yes. But some distinctions: Fortran 95; “Implicit none” not possible in some cases.

- What about the performances?

I don't have significant results yet.

- Relative tools (profiler, debugging tool,...)?

Conclusion

Conclusion

I like the idea of XMP. It can be really easy to use, and I like the idea that it is optimized for the machine and that it is not necessary to rewrite the code (XMP changes, not my code).

I had (have) some difficulties to implement with XMP because of the immaturity of XMP FORTRAN (but not because of the philosophy of XMP). Also, the documentation should be more detailed.

What about the performances of the RTM using XMP Fortran? I cannot say, I don't have significant results yet.

Future Work

- YML/XMP
- Co-Array Fortran.

Questions?