Hybrid Parallelization of a pure Eulerian finite volume solver for multimaterial fluid flows

M. Peybernes[1,2]
in collaboration with J.-Ph. Braeunig[1], J. Costes[3,5],
C. Delacourt[4], P. Demichel[6], J.-M. Ghidaglia[2,5]

[1] CEA, DAM, DIF, F-91297, Arpajon, France
[2] LRC MESO, ENS de Cachan et CEA, DAM
[3] EUROBIOS, 86, Avenue Lénine, 94250 Gentilly, France
[4] CAPS ENTREPRISE, 4 allée Marie Berhaut 35000 Rennes
[5] CMLA UMR 8536, 65 avenue du Président Wilson, 94235 Cachan cedex

[6] HP, HPC EMEA Competency Centre, Grenoble, France

2 trends are seen in current calculators :

- **Bigger systems** : Number of nodes in clusters increase ; Number of processors in supercomputer increase.
- **More powerful components** : increased number of cores ; Specialized co-processors (GPU,MIC,...).

Middlewares are available to program those machines. Each middleware covers a range of usage. Examples :

- **Distributed machines** : MPI.
- **Multicore architectures** : MPI, OpenMP, CILK, TBB,...
- **GPU** : OpenCL, NVIDIA Cuda, HMPP, ...

Introduction : FVCF-NIP method

The **FVCF-NIP**[1] method is developped since Braeunig Phd Thesis (2007) for compressible multimaterial fluid flows simulation. The main property of this pure Eulerian method is the **sliding condition at the interface between materials**, which is an improvement in the consistency of the discretization with respect to the Euler equations model.The interface calculation NIP has been improved to **Enhanced-NIP by Loubère, Braeunig, Ghidaglia, 2011**.

This method has been **parallelized using a domain decomposition in slices associated with transpositions using the MPI library** (Sonnendrücker et al., GYSELA code), and not in blocs as we are used to do. This is a convenient choice for this **totally directionally splitted method**.

Recently, we have developped **an alternative hybrid parallel algorithm**. Indeed, this MPI decomposition in slices restricts the MPI communications to transpositions at the end of each directional phase, **leaving the algorithm almost sequential**. It is then **as easy as for a sequential code to add OpenMP or HMPP (GPU) directives in between transpositions. This hybrid parallel evolution becomes particularly necessary in the context of hexascale computers**.

1. **J.-P. Braeunig and B. Desjardins and J.-M. Ghidaglia**, Eur. J. Mech. B/Fluids, 2009

Numerical results and performance analysis

Computing Ressources :

- **Titane cluster** built by Bull and located at Bruyères-Le-Chatel (2009) : 1068 nodes (including 2 processors Intel quadri-cores) and 48 NVIDIA Tesla S1070.
- **Curie cluster** built by Bull and located at Bruyères-Le-Chatel (2012) : 3 different fractions of computing ressources.
  - **Thin nodes** : 10080 eight-core processors, 2 processors per node (targeted for full MPI parallel codes).
  - **Fat nodes** : 1440 eight-core processors, 4 processors per node (targeted for MPI+OpenMP parallel codes).
  - **Hybrid nodes** : total of 288 Intel quadri-cores + 288 GPU Nvidia.
- **HP cluster** located at the HPC Competency Center (Grenoble), 2012 : $\approx$ 3000 nodes including Intel (six-cores) and AMD processors.
- **CAPS cluster** located at Rennes.

Outline

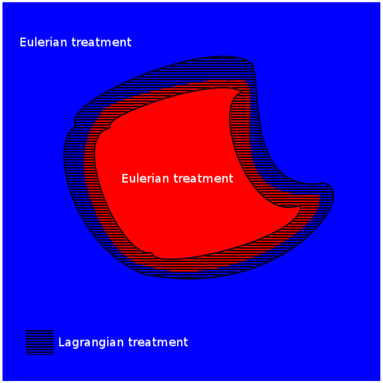
Outline of the talk

- Outline of the NIP method.
- MPI parallelization algorithm using Transpositions.
- Multi-threads parallelization with OpenMP.
- Hybrid parallelization MPI+OpenMP.
- GPU migrating with HMPP.

Outline of the NIP method

Outline of the NIP method

The underlying idea of the FVCF-ENIP method is to **hybridize** both Lagrangian and Eulerian descriptions and take advantage of each :
Far from the interface, that is where the flow is locally **monofluid**, it is convenient to use a Eulerian description.
On the contrary, **in the neighborhood of the interface**, a Lagrangian description provides a natural framework for treating the interface :



Outline of the NIP method

The FVCF-ENIP method is an extension of the **FVCF method** [2] to multifluid simulations.
It is a **colocated Finite Volume method** (no staggered mesh is used), based on a **directional splitting**.
Major features of this method :

- fluids are **not miscible** $\rightarrow$ there is no mixing zone ;
- **no numerical diffusion** between materials and **an exact conservation** of mass, momentum and total energy is granted ;
- interfaces are **reconstructed** (using Young's method [3]) ;
- the **condensate** concept allows to consider material interfaces as edges of **Lagrangian control volumes** $\rightarrow$ **explicit description** of the interface ;
- **perfect sliding** of materials along the interface.

For most physical models the treatment of material interfaces is a **major issue** and is (almost) **always crucial** regarding precision and quality of numerical simulations.

2. **J.-M. Ghidaglia and A. Kumbaro and G. Le Coq**, Eur. J. Mech. B/Fluids (20), 2001
3. **D.L. Youngs**, Numerical Methods for Fluid Dynamics

Outline of the NIP method

Basics : from the continuous flow to numerical representation

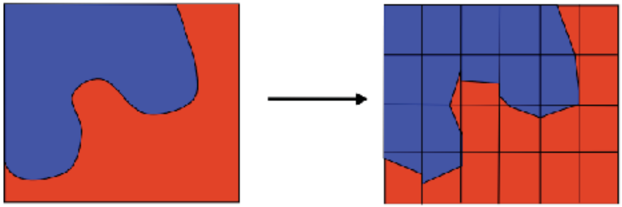
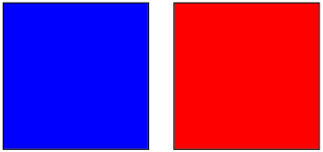


FIGURE: Real situation *vs.* numerical representation.

Real situation can be **as complicated as imaginable** (fragmented interface, non connected fluid subdomains, $\cdots$).
However, numerical representation is **systematic**, we distinguish between :

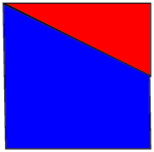
- pure cells (containing one fluid) ;
- mixed cells (containing two fluids).

Outline of the NIP method



Pure cells :

- cell centered variables ;



Mixed cells :

- more than 2 materials ;
- interface is represented by a straight line ;
- volume fraction given for each material ;
- "partial volume" centered variables.

Basics : directional splitting over a cartesian mesh

We solve **systems of Partial Differential Equations** of the type :

$$\frac{\partial v}{\partial t} + \frac{\partial F(v)}{\partial x} + \frac{\partial G(v)}{\partial y} = 0,$$

with an **alternating direction scheme** (*e.g.* a Strang second order splitting) :

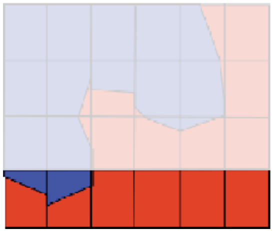


FIGURE: Solving $\frac{\partial v}{\partial t} + \frac{\partial F(v)}{\partial x}$ on each horizontal slice.

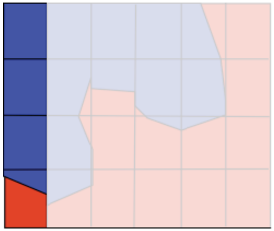
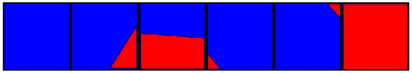


FIGURE: Solving $\frac{\partial v}{\partial t} + \frac{\partial G(v)}{\partial y}$ on each vertical slice.

Therefore, we will only consider a **generic object** of the type :



Parallelization with MPI

MPI Parallelization algorithm using transpositions

Parallelization with MPI

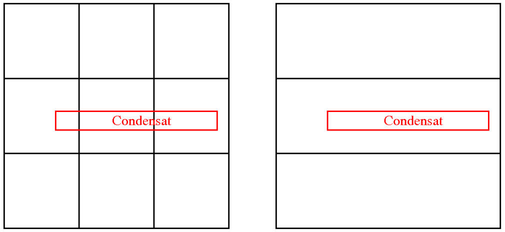


FIGURE: Rectangular subdomain decomposition (on the left) and slice subdomain decomposition (on the right)

Parallelization with MPI

- With classical rectangular subdomain decomposition, a condensate can cross several subdomains and then it has to be computed by several processors. Therefore, this kind of decomposition is not well suited to distributed memory system.
- To take advantage of the 1D directional splitting, we use a subdomain decomposition in **horizontal or vertical slices**.
- Each subdomain (slice) is computed by one proc, such that all condensates of the slice are known and computed by this one.
- A data transfer is needed to go from the x step to the y : we perform **Transpositions** using MPI communications.
- **This method does not use ghost cells** as in classical rectangular subdomain parallelization. The number of degrees of freedom communicated per iteration is constant with respect to the number of procs. **It saves memory when using a large number of procs.**

Slice subdomain decomposition

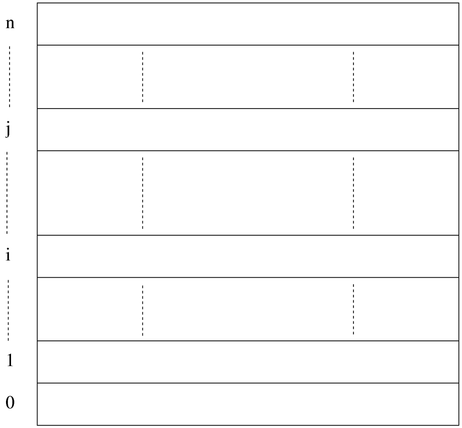


FIGURE: x-step : decomposition in $(n+1)$ horizontal slices for $(n+1)$ procs, on a distributed memory system

Slice subdomain decomposition

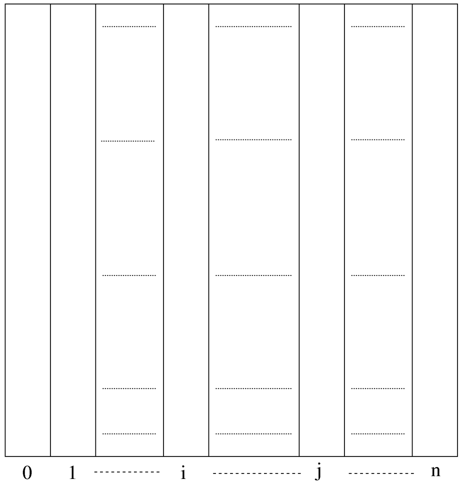


FIGURE: y-step : decomposition in $(n+1)$ vertical slices for $(n+1)$ procs, on a distributed memory system

Transposition $x \rightarrow y$ algorithm

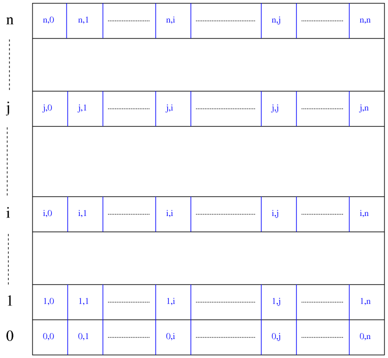


FIGURE: $(n+1)^2$ blocks decomposition of $(n+1)$ horizontal slices (x step), each one associated with one proc.

Transposition $x \to y$ algorithm

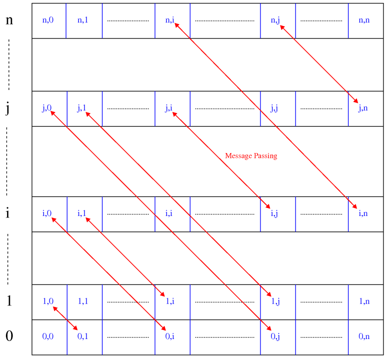


FIGURE: Transposition : blocks communication.

Transposition $x \to y$ algorithm

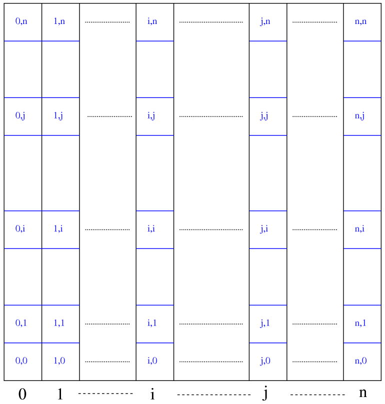


FIGURE: New data organization : $(n+1)$ vertical slices (y step), each one associated with one proc.

Transposition : block description

Communicated quantities :

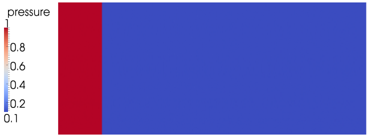
- Nb_m := Number of materials in the cell
- Nu_m := Materials tags
- Vol_m := Materials volumes in the cell
- $V_m := (\rho, \rho u_x, \rho u_y, \rho u_z, \rho E) = 5$ quantities per material in the cell

Dimension of the communicated quantities in a block :

- $\text{dimension}(Nb_m) = \frac{NbCells}{n^2}$
- $\text{dimension}(Nu_m) = \frac{NbPure+NbMix}{n^2}$
- $\text{dimension}(Vol_m) = \frac{NbPure+NbMix}{n^2}$
- $\text{dimension}(V_m) = 5 \times \frac{NbPure+NbMix}{n^2}$

with n the number of computational procs.

Performances MPI, triple point shock tube



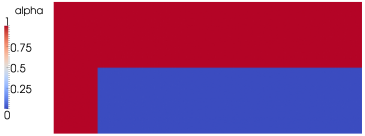
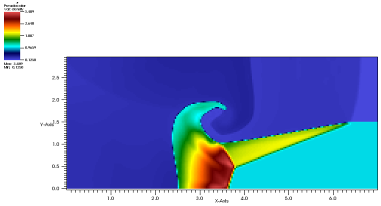


FIGURE: Pressure (left) and volume fraction (right) at initial time, triple point shock tube.



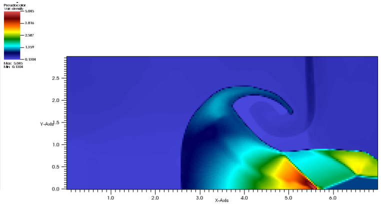
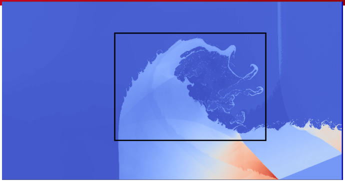


FIGURE: density at time 3.3 (left) and density at final time 5 (right), triple point shock tube with mesh 210 $\times$ 90.

Performances MPI, triple point shock tube



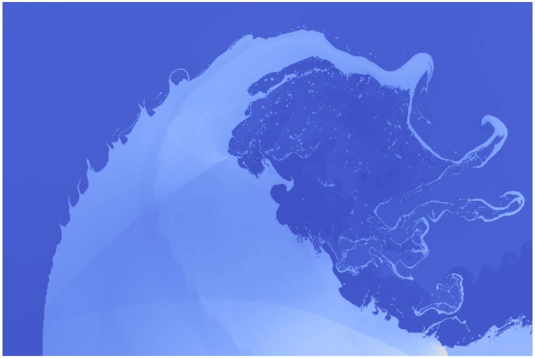


FIGURE: Density full geometry (up) and zoom on small structures (down), mesh 6144x2048 triple point shock tube.

Performances MPI, triple point shock tube

Number of procs	1	2	4	8	16	32	64	128	256
CPU time (s)	128246	64123	32050	16272	7997	4039	2154	1199	741
Speed up	1	2	4	7.88	16	31.75	59.53	106	173
Efficiency	1	1	1	0.98	1	0.99	0.93	0.83	0.67
Cells/proc (n)	$12.6M$	$6.3M$	$3,1M$	$1.6M$	$800K$	$400K$	$200K$	$100K$	$50K$
Cells/block (n^2)	$12.6M$	$3.1M$	$800K$	$200K$	$50K$	$12K$	$3K$	800	200

TABLE: Efficiency for the triple point shock tube with mesh 6144x2048 = 12.6 M cells

Speed up, triple point shock tube

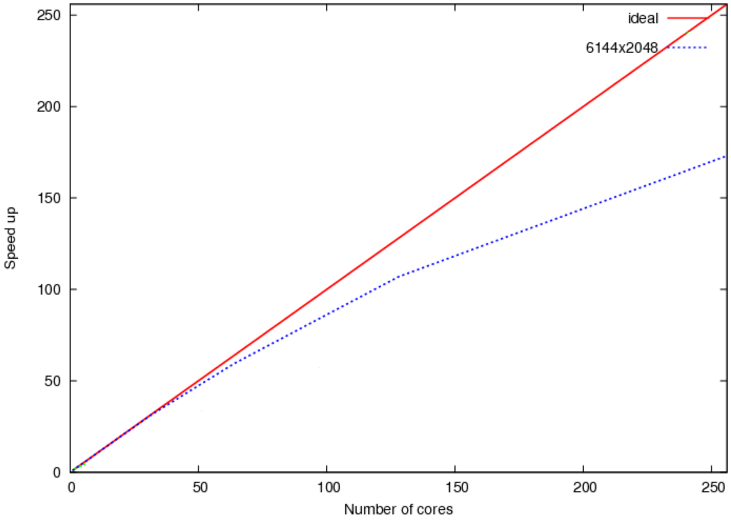


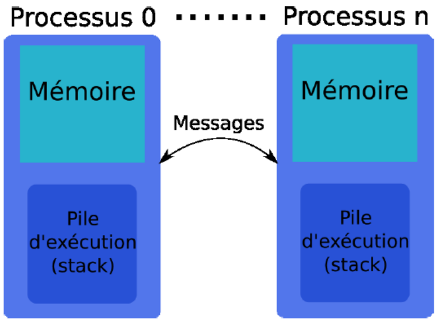
FIGURE: Speed up, triple point shock tube

Summary of the MPI parallelization

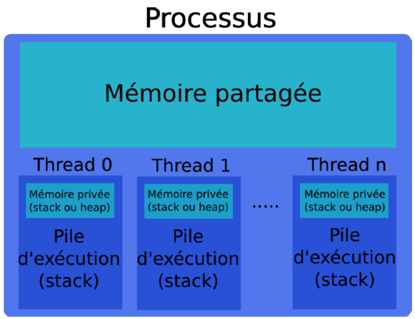
- No communication between two steps of the directional splitting ;
- This parallel algorithm allows to separate completely the numerical part and the MPI communication part of the code ;
- These MPI communications are localized but each processor communicates with all others, which can be too heavy for massively parallel architecture ;
- MPI 3 : non-blocking collective operations into the MPI standard ;
- No need of ghost cells : they become more and more memory consuming on modern architectures ;
- This MPI algorithm is less effective under $\approx 40\,000$ cells per processor.

Multi-threads parallelization with OpenMP

Distributed memory system



Shared memory system


Processus
Mémoire partagée
Thread 0
Thread 1
Thread n
Mémoire privée (stack ou heap)
Mémoire privée (stack ou heap)
Mémoire privée (stack ou heap)
.....
Pile d'exécution (stack)
Pile d'exécution (stack)
Pile d'exécution (stack)