Hybrid parallelization of a pure Eulerian finite volume solver for multi-material fluid flows

M. Peybernes\textsuperscript{1}, J.-Ph. Braeunig\textsuperscript{1}, and J.M. Ghidaglia\textsuperscript{2}
\textsuperscript{1}CEA, DAM, DIF, F-91297 Arpajon Cedex, France
\textsuperscript{2}CMLA, ENS Cachan, 61, Avenue du Président Wilson, 94235 Cachan Cedex, France

Abstract—The FVCF-NIP method has been developed since the work of Braeunig et al. \cite{1} for compressible multi-material 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. In this paper, we propose a parallelization of this method using a domain decomposition in slices associated with a transposition using the MPI library, and not in blocks as in usual domain decomposition techniques. This is a convenient and efficient choice for this totally directionally splitted method. Then a hybrid parallel algorithm is introduced using multithreading (with OpenMP) and GPU migration (with HMPP) into each slice.

Keywords: Finite volume; Multi-material; parallel computing; MPI; OpenMP

1. Introduction: the FVCF-NIP method

This method is termed as pure Eulerian Finite Volumes in the sense it does not use an operator splitting as in the Lagrange-Remap scheme for instance. This method is built for compressible multi-material fluid flows simulation. The underlying single-phase scheme is FVCF \cite{2}. The interface between materials is sharp and is approximated by a piecewise linear curve. In a mixed cell, i.e. a cell containing more than one material, the interface is then a straight line in 2D, which separates partial volumes containing each a pure material, with no mixing at all. The method is totally cell centred, so each partial volume, as a pure cell, has its own volume centred pressure, velocity vector, density, energy and corresponding EOS. This method conserves locally the mass, the momentum and the total energy, since we write conservation laws on pure cells and even on each partial volume. This is made possible by introducing a 1D data structure called condensate, in the context of a directional splitting, which is a merge of neighboring mixed cells in one direction of the mesh, see Figure 1. The evolution of this set of cells is computed no more considering cell faces but considering interfaces between materials as Lagrangian interfaces. A conservation law is written on partial volumes in the condensate by computing fluxes through these Lagrangian interfaces. It should be noticed that these fluxes are computed in such a way a sliding condition is imposed between materials, by writing a Riemann problem in the interface normal vector direction. The new state of the condensate is then remapped on the mesh. This technique is restricted to rectangular mesh since the directional splitting is necessary to compute the interface motion using condensates.

One interesting feature of this method for parallel computing is that each phase of the directional splitting in a generic direction $x$ does not need informations from other directions (however, the 2D interfaces normal vectors are used to compute the fluxes and to impose the sliding condition at interfaces). That makes the computation of each cell line of the mesh independent from the others during one step of the direction splitting, so an interesting property for parallel computing. Indeed the parallel algorithm presented here takes advantage of these 1D features using a domain decomposition in slices.

On the other hand, to take full advantage of the largest and fastest computers employing both shared and distributed memory architectures, we propose a hybrid model which combines message passing and shared memory programming.

The remainder of this article is organized as follows. In section 2, we present the MPI parallel algorithm with the associated transposition, and the hybrid parallel algorithm. Then we propose a significative multi-material test in section 3 to evaluate the efficiency of the parallel code.

2. MPI + (OpenMP or GPU) parallelization

2.1 MPI Decompostion in slices

With classical rectangular subdomain decomposition parallelization, a condensate (defined in section 1) can cross several subdomains and then it has to be computed by several processors. Therefore, this method FVCF-NIP using condensates is not well adapted to this kind of parallelization.

The parallel algorithm presented here allows to compute each condensate by a single processor and to take advantage of the 1D directional splitting used in the FVCF-NIP method. During the computation of the $x$ step of the directional splitting, the 2D domain is decomposed in horizontal slices (Figure 2). Each slice is computed by a processor $P_i$ on a distributed memory system. In the same way, we use vertical
Fig. 1: Extraction of neighboring mixed cells from the grid to become a condensate during $x$ direction step.

slices to decompose the domain during the $y$ step. Thus, each slice contains the same number of cells equal to the total number of cells ($NbCells$) divided by the number of processors ($n$), what allows a good load balancing between processors.

However, with this kind of parallel algorithm, a processor (attached to a MPI processus) has to compute a different subset of cells (vertical or horizontal slice) at each step ($x$ or $y$) of the directional splitting. Thus, to allow a given processor to compute the appropriate subset of cells following the direction change, we propose a “transposition algorithm” (see section 2.2). This transposition allows to transfer the necessary data using the Message Passing Interface (MPI).

### 2.2 Transposition

Between two steps of the directional splitting, the transposition allows to move from a horizontal slice decomposition to a vertical slice decomposition and reciprocally. The data transfer is performed using MPI communications. During transposition $x \rightarrow y$ (Figure 4), each slice is split in $n$ blocks. Then each processor communicates each block $j$, $j \neq i$, to processor $i$. MPI nonblocking messages are used to obtain an overlap of communications and computations.

To limit the communication cost of the transposition, we communicate the smallest amount of data needed by the algorithm: for each cell, the number of materials, the material volumes, and the conservative quantities per material $V_m := (\rho, \rho u_x, \rho u_y, \rho E)$.

Other necessary quantities to the algorithm are deduced from the restricted set of communicated ones thanks to equations of state. Fortunately, this supplementary computations are scalable since they are proportional to the number of cells in a slice ($NbCells/n$).

### 2.3 Hybrid parallel algorithm: multithreading or manythreading into a slice

Modern multicore HPC clusters have hierarchical levels of parallelism. They contain multicore processors (for instance, processor intel Xeon 7500 Nehalem in the TERA100\(^1\) cluster is an eight-cores) with shared memory and graphics processors (GPU’s).

Then, to improve performance of our full MPI version of VFFC-NIP code on these clusters, we have introduced a shared memory threading into each slice using OpenMP. Thus each slice is attached to a processor though a MPI processus and each sub-slice is computed by a core though a thread.

On the other hand, to take advantage of GPU’s and following the previous OpenMP algorithm using directives, we use HMPP directives to allow the migration of a slice computation to a GPU. HMPP is a Heterogeneous Multicore Parallel Programming workbench with compilers, developed by CAPS entreprise, that allows the integration of heterogeneous hardware accelerators in a non-intrusive manner while preserving legacy codes.

![Diagram](image)

Fig. 2: Decomposition in $n$ horizontal slices for the $x$-step, for $n$ processors on a distributed memory system

This hybrid parallel algorithm is efficient since MPI communications are all localized in the transposition, between the two steps of the directional splitting (Figure 5). Actually, each cell line can be computed independently from the others, as it has been said in section 1. So during each directional step, a processor makes all the computations for the slice needed by this finite volume method (flux computation, interfaces motion computation...) independently from the others, so without any communication. Then, the “OpenMP region” and “HMPP region” do not contain any MPI communication as we can see in figure 5. Moreover,

\(^{1}\)The TERA100 cluster is located at Bruyeres-le-Chatel, CEA/DAM-Ile de France center. Built by Groupe Bull, it has a peak processing speed of 1050 teraflops, making it the fastest supercomputer in Europe as of 2011.
Fig. 3: Decomposition in $n$ vertical slices for the $y$-step, for $n$ processors on a distributed memory system.

Fig. 4: Transposition $x \rightarrow y$: blocks communications in opposition to classical subdomain parallelization, this method does not need ghost cells. This memory saving is particularly appreciable for supercomputers that tend to have less memory per core.

3. Numerical results and performance analysis

The numerical simulations presented in this paper have been performed on the Titane cluster (CCRT: Centre de Calcul Recherche et Technologie, located at Bruyères-le-Chatel, CEA/DAM-Ile de France center). This massively hybrid parallel computer is composed of a fast network (Infiniband) connecting classical nodes with multicore processors Intel Xeon 5570 (quadri-cores) and accelerators (GPUs Tesla from NVIDIA).

The test we have chosen to evaluate the efficiency of the parallelization algorithm is a two material “SOD like” shock tube with an initial geometry containing a triple point and three different states. A vortex will be created by the difference of shock waves velocity between horizontal layers in such a way materials will roll up. Equations of state are of type perfect gas with different gamma coefficients between layers. This test is convenient for parallel performance analysis since it creates enough mixed cells to be demonstrative for the interface capturing cost.

In tables 1 and 2, one can see that the efficiency is good ($> 0.6$) up to 64 cores for a 1 million cells test. The data size of blocks communicated during transpositions is decreasing fast by a $n^2$ factor. Of course, the number of communications follows a $n^2$ factor as well, so a good balance should be found function of the number of cores. When the data size of blocks decreases, the use of OpenMP allows to keep a
good efficiency up to 256 cores, compared to the full MPI algorithm which has a bad efficiency under 16,000 cells per block. This improvement with MPI+OpenMP is illustrated in Table 3 too, for 4.2 million cells and 512 cores.

<table>
<thead>
<tr>
<th>Nb cores</th>
<th>1</th>
<th>2</th>
<th>16</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI</td>
<td>1800</td>
<td>910</td>
<td>150</td>
<td>80</td>
</tr>
<tr>
<td>Efficiency</td>
<td>0.98</td>
<td>0.75</td>
<td>0.70</td>
<td></td>
</tr>
<tr>
<td>Cells/core</td>
<td>1048K</td>
<td>524K</td>
<td>65K</td>
<td>32K</td>
</tr>
</tbody>
</table>

Table 1: Efficiency with 1 million cells

<table>
<thead>
<tr>
<th>Nb cores</th>
<th>64</th>
<th>256</th>
<th>256</th>
</tr>
</thead>
<tbody>
<tr>
<td>64MPI</td>
<td>47</td>
<td>94</td>
<td>17</td>
</tr>
<tr>
<td>Efficiency</td>
<td>0.60</td>
<td>0.13</td>
<td>0.41</td>
</tr>
<tr>
<td>Cells/core</td>
<td>16K</td>
<td>4K</td>
<td>4K</td>
</tr>
</tbody>
</table>

Table 2: Efficiency with 1 million cells

<table>
<thead>
<tr>
<th>Nb Cores</th>
<th>512</th>
<th>512</th>
<th>512</th>
</tr>
</thead>
<tbody>
<tr>
<td>512MPI</td>
<td>650</td>
<td>440</td>
<td>320</td>
</tr>
<tr>
<td>Cells/core</td>
<td>8K</td>
<td>8K</td>
<td>8K</td>
</tr>
</tbody>
</table>

Table 3: Efficiency with 4.2 million cells

4. Conclusion
We have described a MPI parallelization of the FVCF-NIP method based on a domain decomposition in slices associated with a transposition. It is well adapted for methods using a directional splitting. The independence (without any communications) of each MPI process during a directional step allows to easily add in it multi-threaded shared memory parallelism. The resulting hybrid code (MPI+OpenMP and MPI+GPU with HMPP) allows to take full advantage of these method features and of heterogeneous architectures of our current and future HPC clusters (massively hybrid parallel clusters).
Fig. 8: Density full geometry (up) and zoom on small structures (down) with 12 Million cells, triple point shock tube.

References
