A two-fluid finite-volume solver based on OpenCL

Jonathan Jung

A two-fluid finite-volume solver based on OpenCL

Jonathan Jung*

April 20, 2015

Abstract

In this paper, we propose a new very simple numerical method for solving liquid-gas compressible flows on two dimensional cartesian meshes. For achieving high performance, the scheme is tested on recent multi-core processors and Graphics Processing Units (GPU), using the OpenCL environment. We describe how to install and to run the code for computing a shock-bubble interaction on your GPU.

1 Mathematical model and numerical strategy

1.1 Model

We are studying the numerical resolution of a two-fluid compressible fluid flow. The model is the Euler equations with an additional transport equation of a color function φ. The function φ is equal to 1 in the gas and 0 in the liquid. It allows to locate the two-fluid interface. We consider the system

$$\partial_t W + \partial_x F(W) + \partial_y G(W) = 0, \tag{1.1}$$

where

$$\begin{aligned} W &= (\rho, \rho u, \rho v, \rho E, \rho\varphi)^T, \\ F(W) &= (\rho u, \rho u^2 + p, \rho uv, (\rho E + p)u, \rho u\varphi)^T, \\ G(W) &= (\rho u, \rho uv, \rho v^2 + p, (\rho E + p)v, \rho v\varphi)^T. \end{aligned}$$

The pressure law p is a stiffened gas pressure law

$$p(\rho, e, \varphi) = (\gamma(\varphi) - 1)\rho e - \gamma(\varphi) p_\infty(\varphi), \tag{1.2}$$

where $e = E - \frac{u^2+v^2}{2}$, and

$$(\gamma, p_\infty)(\varphi) = \begin{cases} (\gamma_{gas}, p_{\infty,gas}), \text{ if } \varphi = 1, \\ (\gamma_{liq}, p_{\infty,liq}), \text{ if } \varphi = 0. \end{cases} \tag{1.3}$$

1.2 Mesh

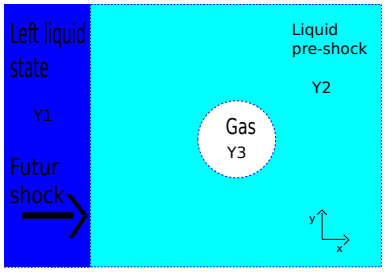
We want solve (1.1) on the study domain $[a;b] \times [c;d]$ between time $t = 0$ and time $t = T > 0$. We consider regular subdivisions $(x_{i-\frac{1}{2}})_{0 \le i \le N_x+1}$ of $[a;b]$ and $(y_{j-\frac{1}{2}})_{0 \le j \le N_y+1}$ of $[c;d]$

$$\begin{aligned} x_{i-\frac{1}{2}} &= a + ih_x,\ i = 0 \cdots N_x + 1, \\ y_{j-\frac{1}{2}} &= c + ih_y,\ j = 0 \cdots N_y + 1, \end{aligned}$$

where $h_x = \frac{b-a}{N+1}$ and $h_y = \frac{d-c}{M+1}$ are space steps in the x and y direction. The cell $C_{i,j}$ is the volume

$$C_{i,j} = \left]x_{i-\frac{1}{2}}; x_{i+\frac{1}{2}}\right[\times \left]y_{j-\frac{1}{2}}; y_{j+\frac{1}{2}}\right[,$$

*EFREI, 30-32 avenue de la République, 94800 Villejuif, France and Université Pierre et Marie Curie, LRC Manon and LJLL, 4 place Jussieu, 75252 Paris, cedex 05, France, jonathan.jung@ljll.math.upmc.fr



Quantities	Y1	Y2	Y3
$\rho(kg.m^{-3})$	1030.9	1000	1
$u(m.s^{-1})$	300	0	0
$v(m.s^{-1})$	0	0	0
$p(Pa)$	3.0e9	1.0e5	1.0e5
φ	0	0	1
γ	4.4	4.4	1.4
$p_\infty(Pa)$	6.8e8	6.8e8	0

Figure 1.1: Shock bubble interaction. Description of the initial conditions on the left and initial data on the right.

centrer on (x_i, y_j) with

$$x_i = \frac{x_{i-\frac{1}{2}} + x_{i+\frac{1}{2}}}{2},\ i = 1 \cdots N_x,$$
$$y_j = \frac{y_{j-\frac{1}{2}} + y_{j+\frac{1}{2}}}{2},\ j = 1 \cdots N_y.$$

The cell $C_{i,j}$ for $i = 0$, $i = N_x$, $j = 0$ and $j = N_y$ are used to apply the boundary conditions.
We consider also a sequence of times t_n, $n \in \mathbb{N}$ such that the time step $\Delta t_n := t_{n+1} - t_n > 0$.

1.3 Numerical strategy

We solve the two dimensional equations (1.1) on cartesian meshes with a dimensional splitting. Each step of the splitting is solved with a finite volume scheme. We use a Arbitrary-Lagrangian-Eulerian (ALE) method coupled with a remap step. It allows us to schwith between a Lagrangian approach at the liquid-gas interface and an Eulerian approach in the pure phases. The ALE step used a relaxation solver described in [7] and the remap step is done randomly. For more details, we refer to [1, 7, 8].

1.4 Initial conditions

We present a two-dimensional test that consists in simulating the interaction of a shock wave propagating in a liquid and interacting with a gas cylinder (bubble). It is a difficult problem, both numerically and physically. The initial conditions are depicted in Figure 1.1: the bubble of gas is surrounded by liquid at rest in atmospheric conditions. The computational domain is 2 meters long and 1 meter high. The gas is inside a disk (bubble) whose center is is located at $(0.5, 0.5)$. The initial radius of the bubble is 0.4 m. A piston hits the left side at the velocity of 300 $m.s^{-1}$ yielding a shock pressure of about 3×10^9 Pa. The initial data of this problem are given in the table of the Figure 1.1.

1.5 Boundary conditions

Top and bottom boundary conditions are set to solid walls while we use constant state boundary conditions for the left and right boundaries.

2 The code CLBubble

For performance reasons, we decided to implement the 2D scheme on recent multicore processor architectures, such as a Graphic Processing Unit (GPU).

2.1 What is a GPU?

A modern GPU is made of a global memory ($\approx$ 1 GB) and compute units ($\approx$ 28). Each compute unit is made of processing elements ($\approx$ 10) and a local memory ($\approx$64 kB). The same program (a kernel) can be executed on all the processing elements at the same time. There are some rules to respect. All the processing elements have access to the global memory but have only access to the local memory of their compute unit. The access to the

global memory is slow while the access to the local memory is fast. The access to global memory is much faster if two neighboring processing elements read (or write) into two neighboring memory locations, in this case we speak about "coalescent memory access".

2.2 OpenCL and GPU implementation

The OpenCL [6] implementation is described in [8, 7]. We recall only the main lines.

- We compute the time step Δt_n. We compute a local time step $(\Delta t_n)_{i,j}$ on each cell and we use a *reduction algorithm* (see [2]) to compute $\Delta t_n = \min_{i,j}(\Delta t_n)_{i,j}$.
- We perform the ALE-projection update in x-direction. We compute the fluxes balance in the x-direction for each cell of each row of the grid: a row or a part of a row is associated to one compute unit and one cell to one virtual processor. As of October 2012, the OpenCL implementations generally imposes a limit (typically 1024) for the number of work-items inside a work-group [6]. This forces us to split the rows for some large computations. The values in the cells are then loaded into the local cache memory of the compute unit. It is then possible to perform the ALE-projection algorithm with all the data into the cache memory in order to achieve the highest performance. The memory access are coalescent for reading and writing.
- We transpose the data matrix (exchange x and y) with an optimized memory transfer algorithm [9]. The optimized algorithm includes four steps:
  - the data matrix are splitted into smaller tables of size 32×32. Each sub-table is associated to a compute unit,
  - each sub-table is copying line by line from the global memory to the local memory of the compute unit. Memory access are coalescents because two successive processors read in two neighboring memory locations,
  - we transpose each sub-table 32×32 in the local memory,
  - each sub-table is copying line by line from the local memory to the global one. The memory access are coalescent for writing.
- We perform the ALE-projection update in y-direction. The memory access are coalescent because of the transposition,
- We transpose again the data matrix for the next time step.

The repartition of the computational time on each kernel is the following: the ALE-projection update represents 80%, the transposition 11% and the time step computation 9% of the global time computation.
We observe high speedups (see [8, 7]) with the GPU implementation. The efficiency is explained by two important points. We used an optimized transposition algorithm to have coalescent access in x and y directions and we also used a relaxation solver. With this solver, fluxes have a simpler expression than the exact Godunov's flux.

2.3 Documentation

We perform a documentation of the code with Doxygen [4]. To obtain an HTML and a LATEX documentation, go in the folder Doxygen.

```
cd Doxygen/
```

and run:

```
doxygen Doxyfile
```

The HTML documentation is build in html/index.html and the LATEX documentation is in latex/refman.tex

3 Run the shock bubble interaction on GPU

In this section, we explain how to install the code. We choose CMake [3] to build the executable file. Then, you need CMake on your computer.

3.1 Environnement and GPU

The code was tested on different hardware. It works on LINUX and MACOS environnement. There are two important points for the code, the first one is OpenCL and the second one is the drivers for your GPU. The code can be execute on Graphics Processing Units from different brand (Nvidia, AMD, Intel).

3.2 How to compile and run the code?

The code run on LINUX or MACOS, the way to compile it is the same. There are two steps. The first one is to detect all OpenCL devices and the second one is the run the code on the chosen OpenCL device.

3.2.1 Choose the OpenCL device

To found all OpenCL devices, go in the folder
"Found_OpenCL_Devices"

```
cd Found_OpenCL_Devices/Make/
```

and you run:

```
cmake .
```

It look if OpenCL is install and construct all the link for the OpenCL libraries. After that, run

```
make
```

It construct an executable file "cldevices" to list all OpenCL devices. You can show the list with

```
./../Binary/cldevices
```

The answer is something like this

```
CL_PLATFORM_NAME = Apple
CL_PLATFORM_VERSION = OpenCL 1.2 (Jun  3 2014 12:43:41)
2 devices found
Device #0 name = Intel(R) Core(TM) i5-4250U CPU @ 1.30GHz
        Driver version = 1.1
        Global Memory (MB):     8192
        Global Memory Cache (MB):       0
        Local Memory (KB):      32
        Max clock (MHz) :       1300
        Max Work Group Size:    1024
        Number of parallel compute cores:       4
Device #1 name = HD Graphics 5000
        Driver version = 1.2(Jun  9 2014 13:24:19)
        Global Memory (MB):     1536
        Global Memory Cache (MB):       0
        Local Memory (KB):      64
        Max clock (MHz) :       1200
        Max Work Group Size:    512
        Number of parallel compute cores:       280
```

In this example, the GPU device is the number 1. Then, to run the code on the GPU, you choose DEVICEID = 1.

3.2.2 Run the code on the chosen device

To compile the code, go in the folder "Make"

```
cd ../../Make/
```

and run:

```
cmake . -DDEVICEID:INT=1
```

where 1 is the number of the chosen OpenCL device. If the number of your GPU device is 0, replace 1 by 0. After that, you can compile the code with

```
make
```

You can run the code on the device number "DEVICEID" with

```
./../Binary/clbubble
```

3.3 How to run the code with an interactive visualisation (OPENGL)?

As the computation is done on the GPU, it is also possible to visualize the data during the computation. For an interactive visualization, OpenGL, GLEW and GLUT have to be installed in your computer. Moreover, you should also have at least the version 2.8.10 of cmake to found the GLEW libraries. To compile the code, go in the folder "Make" and run:

```
cmake . -DOPENGL:BOOL=TRUE -DDEVICEID:INT=1
```

where 1 is the number of the chosen OpenCL device.

Remark 1. It is possible that Cmake returns an error. Sometimes, it does not found the direction for the header (for example GLEW_INCLUDE_DIRS) or a library (for example GLEW_LIBRARIES). In this case, go in the file CMakeCache.txt and put the good link at the right place. You could found the good link with a "locate". When the CMakeCache.txt is completed, run again "cmake" to be sur that everything is good.

After that, you compile the code with

```
make
```

You can run the code on the device number "DEVICEID" with

```
./../Binary/clbubble
```

To stop the computation and build an output file, press "Escape".

3.4 Output

The code construct a Gmsh [5] file in the folder "Output". The file contains the value of ρ, u, v, p and φ on the domain at the final time.

3.5 How to modify the mesh?

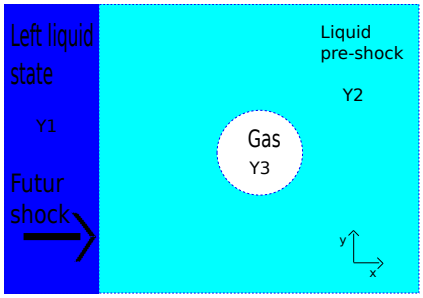
You can increase or decrease the number of cell of the computation in each direction. To increase (resp. decrease) the number of cell in the x direction, increase (resp. decrease) the variable _NBLOCKSX at line 41 of file "CLFunctions.hpp". For the y direction, we can modify the variable _NBLOCKSY at line 47 of file "CLFunctions.hpp". You could also increase the number of cells in the x (resp. y) direction by increasing the number of work-item _NBWORKSX (resp. _NBWORKSY). If you modify this number, you have to be careful because there is a maximal size that is different for each GPU. The maximal size is given in the information at the beginning of the execution. It is the number referenced after

```
Nb Works Max=
```

The code should be more efficient if you put this maximal value in _NBWORKSX (resp. _NBWORKSY).

3.6 How to modify the initial condition?

You can change the initial condition by modifying the density ρ (_RHO), the x-velocity u (_U), the y-velocity v (_V), the pressure p (_P) and the gas mass fraction φ (_PHI) in the file "CLFunctions.hpp". The initial configuration is the following



The parameters γ_{gas}, γ_{liq}, $p_{\infty,gaz}$ and $p_{\infty,liq}$ of the stiffened gas pressure law (1.2)-(1.3) in each phase are fixed by _GAMGAZ, _GAMLIQ, _PINFGAZ and _PINFLIQ. The variable φ (_PHI) allows to fix the phase (liquid or gas) of each states at the initial time. φ is equal to 1 in the gas and to 0 in the liquid phase.

3.7 How to change the pressure law?

It is possible to change the pressure law. You have to change three functions. The first one is the functions W2Y where we use the expression of the pressure p as a function of the density ρ, the internal energy e and the gas mass fraction φ. The second one is Y2W where we use the expression of the internal energy e as a function of the density ρ, the pressure p and the gas mass fraction φ. The third one is the expression of the sound_speed where we describe the sound speed c as a function of the density ρ, the internal energy e and gas mass fraction φ.

4 Conclusion

In this paper, we describe how you can install and run on your GPU the code CLBUBBLE to compute liquid-gas compressible flows on a two dimensional cartesian meshe.

Acknowledgements

The author wish to thank Philippe Helluy for many fruitful discussions.

References

[1] M. Bachmann, P. Helluy, J. Jung, H. Mathis, and S. Müller. Random sampling remap for compressible two-phase flows. *Computers and Fluids*, 86:275–283, 2013.

[2] G.E. Blelloch. Scans as primitive parallel operations. *IEEE Transactions on Computers*, 38(11):1526–1538, 1989.

[3] CMake. Cmake online documentation. `http://www.cmake.org`.

[4] Doxygen. Doxygen online documentation. `www.doxygen.org`.

[5] Gmsh. Gmsh online documentation. `http://www.geuz.org/gmsh`.

[6] Khronos Group. Opencl online documentation. `http://www.khronos.org/opencl/`.

[7] P. Helluy and J. Jung. Opencl simulations of two-fluid compressible flows with a random choice method. *IJFV International Journal On Finite Volumes*, 10:1–38, 2013.

[8] J. Jung. *Schémas numériques adaptés aux accélérateurs multicoeurs pour les écoulements bifluides*. PhD thesis, University of Strasbourg, 2013.

[9] G. Ruetsch and P. Micikevicius. Optimizing matrix transpose in cuda. *NVIDIA GPU Computing SDK*, pages 1–24, 2009.