Development of hybrid software to run in GPUs and CPUs - Implementation of the high-order grid based Piecewise Parabolic Method[1]

Nuno Carvalho[1]

Departamento de matemática da Universidade de Évora

Miguel de Avillez

Departamento de Matemática da Universidade de Évora
Technische Universität Berlin

[1]Funded by the Partnership for Advanced Computing in Europe - Second Implementation Phase (PRACE-2IP) Work Package 8

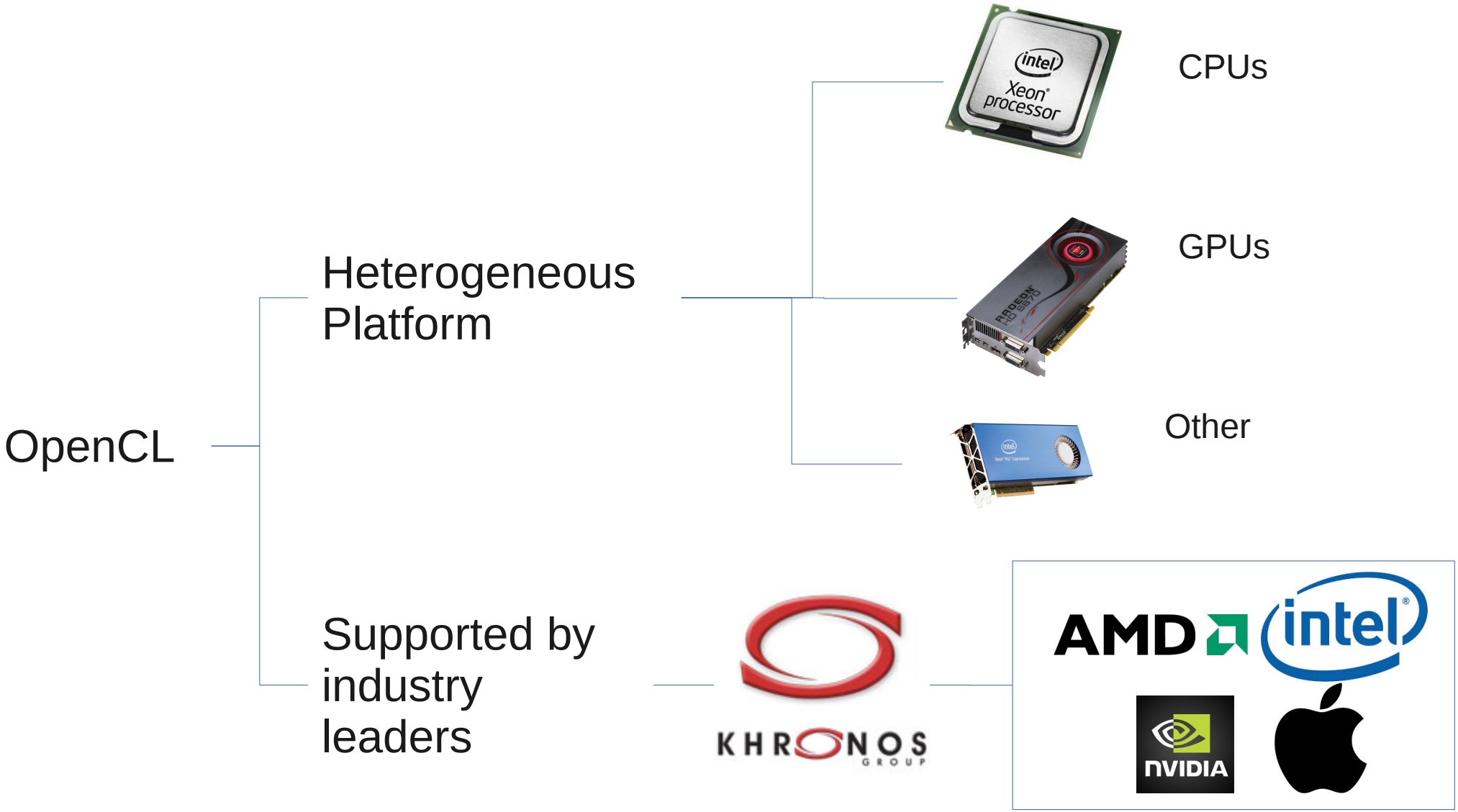
The Objective

Create a highly parallel grid-based hydrodynamical code that makes use of the evolving GPU technology, while retaining the capability to use existing CPUs.

Work supported by the PRACE-2IP WP8.

OpenCL

The platform chosen for the code development was the Open Computing Language (OpenCL)



Numerical methods

Grid based hydrodynamics

Problem is divided in a grid.

Physical quantities are represented as volume averaged values in the center of each cell.

The flux between each cell is calculated (Riemann problem).

The value at the cell center is then updated.

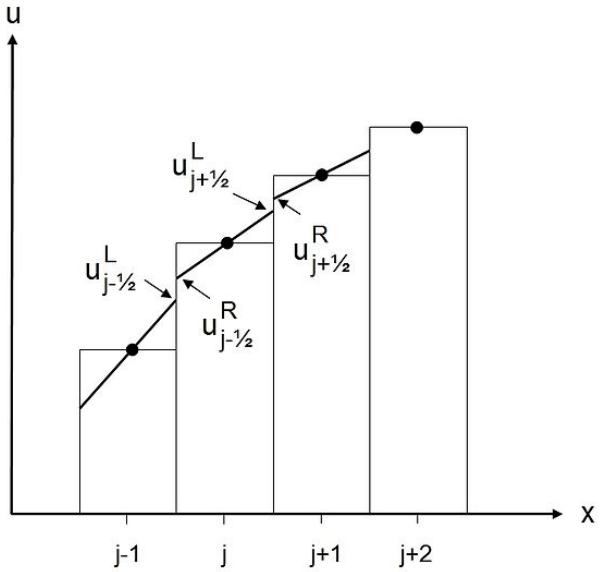
Piecewise Parabolic Method

To calculate the flux at the boundaries we need the value of the physical quantity at that boundary.

The value at the center of the cell is considered constant in all the cell. → First order methods

To achieve better results we can interpolate the values so that the boundary value is a better representation of the real value.

Piecewise linear interpolation →



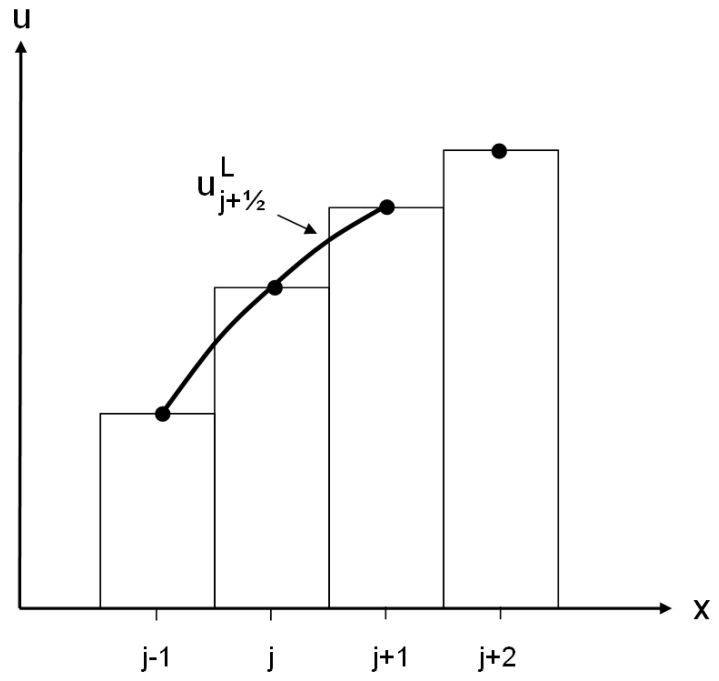
Numerical methods

We can do better than linear interpolation.

In 1984 Colella & Woodward developed the Piecewise Parabolic Method.

Interpolation is done using second degree polynomials.

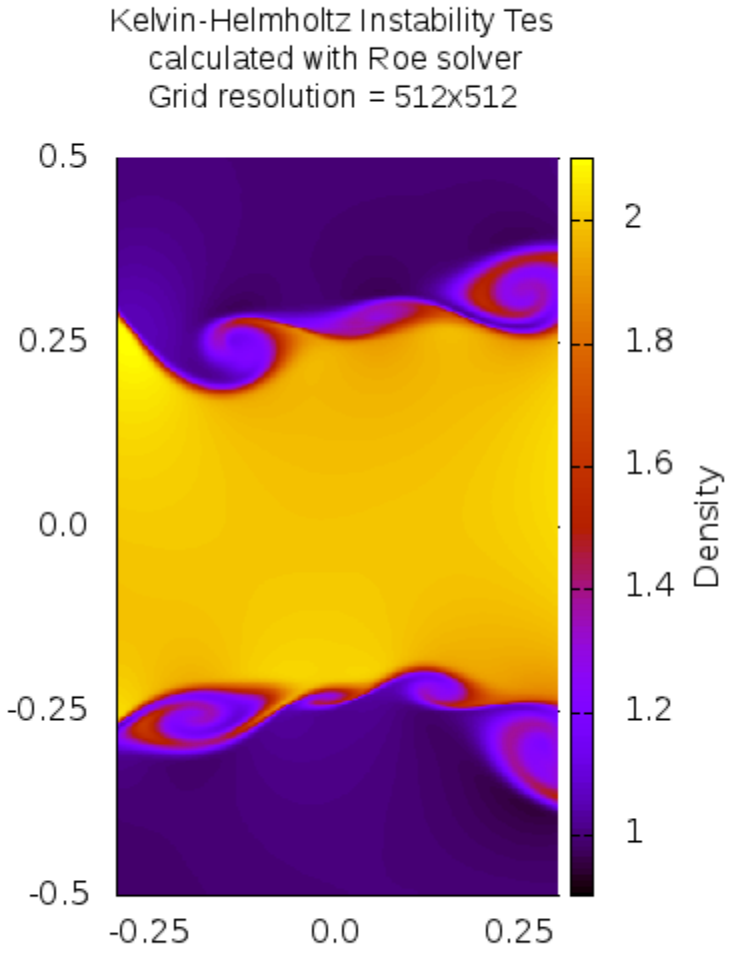
This greatly increases the accuracy of the numerical solution.

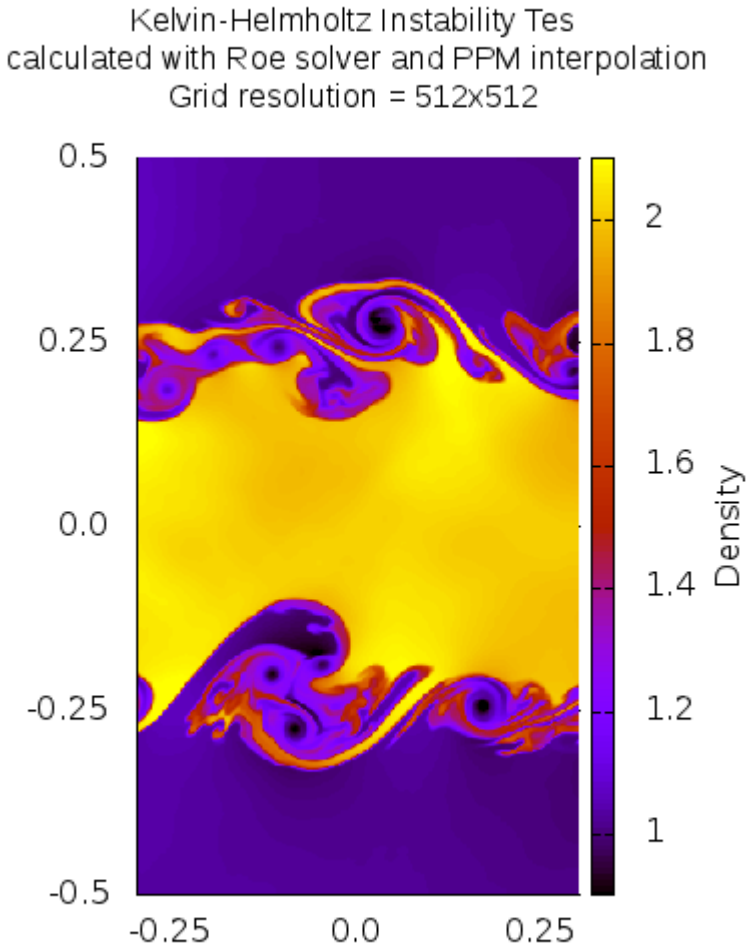


Piecewise parabolic reconstruction

Numerical methods

Example of the difference in the simulation result when using PPM or a simple first order method.





Code description

Implemented methods

Roe & HLL Riemann solvers

Piecewise Parabolic Method (PPM, Colella & Woodward 1984)

Unsplit first order and PPM methods using Corner Transport Upwind (CTU) algorithm (Colella, 1990) (in development)

Parallelization methods

Dimensional splitting

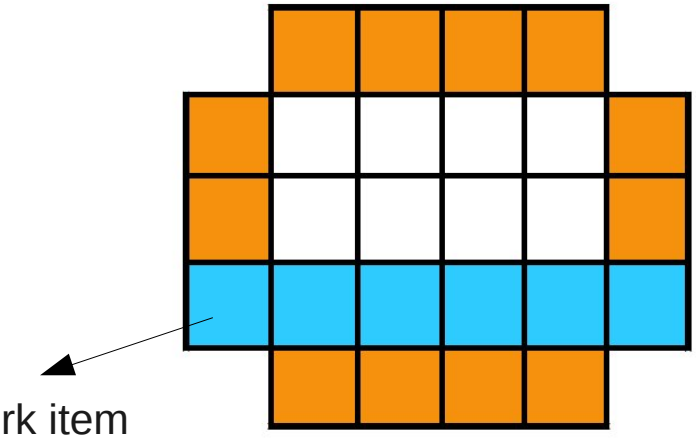
Uses only global memory – each element is only read once and stored in private memory

One work item solves the 1D problem along a single row

A x*y*z problem launches y*z work items

No read/write conflicts in global memory

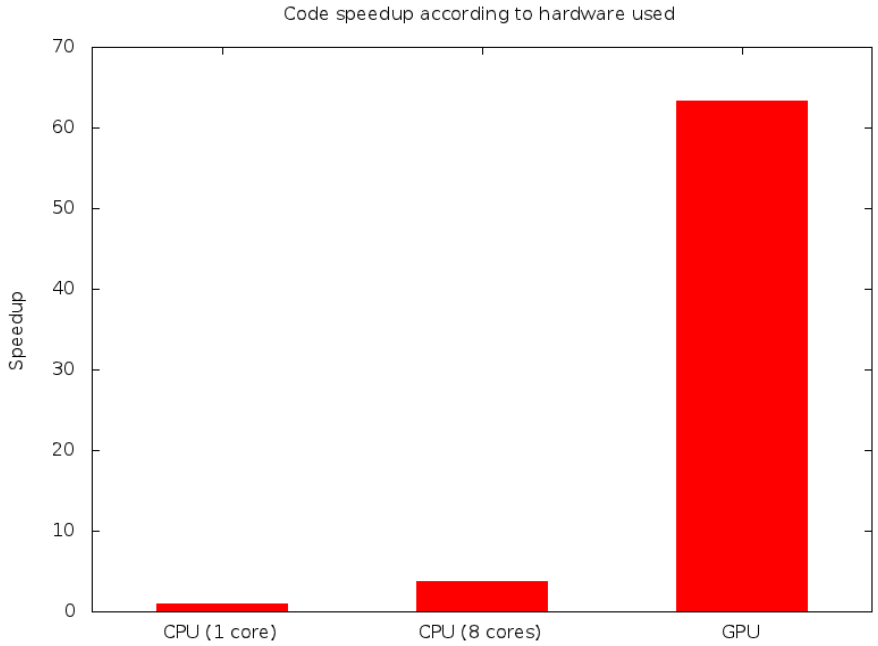
High amounts of work done per work item



Code description

Using one GPU we can obtain a speedup of about 63x.

When comparing CPU speedups we obtain values of 3.7x, very close to the theoretical maximum!



CPU used: Intel core2 Q6600
Clock speed 2.4 GHz
Number of cores: 4

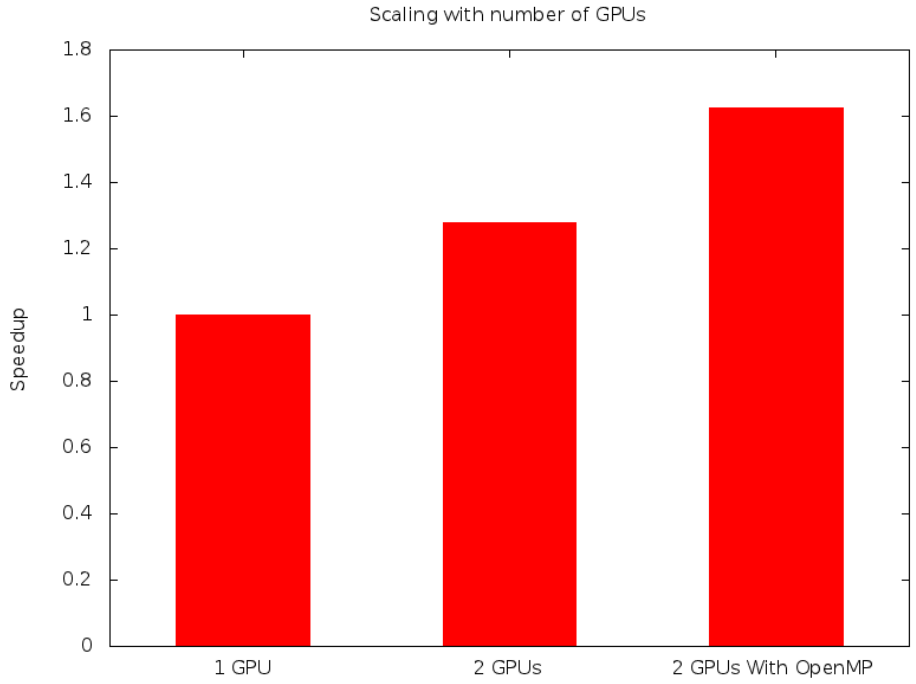
GPU used: AMD HD 7970
32 compute units (2048 stream processors)
Up to 925MHz Engine Clock

Code description

OpenCL is not enough!

Due to the large size of most scientific simulations, using only one GPU will not be enough because of memory constraints.

When using more than one device we must use other parallelization methods on the host code.



We can see that when using 2 GPUs without parallelization in the host we obtain only a speedup of about 35%.

With a simple implementation of OpenMP directives we can improve that to a speedup of 65%.

Future work

Extend the number of methods (unsplit 3D HD)

Include MHD

Implement self gravity

Improve host code parallelization

Use the code to run simulations of the local interstellar medium