Accelerating Euler equations numerical solver on graphics processing units

Pierre Kestener[1], Frédéric Château[1], Romain Teyssier[2]

[1] CEA, Centre de Saclay, DSM/IRFU/SEDI, F-91191 Gif-Sur-Yvette, France
pierre.kestener@cea.fr,
WWW home page: http://irfu.cea.fr/en/index.php
[2] CEA, Centre de Saclay, DSM/IRFU/SAp, F-91191 Gif-Sur-Yvette, France

Abstract. Finite volume numerical methods have been widely studied, implemented and parallelized on multiprocessor systems or on clusters. Modern graphics processing units (GPU) provide architectures and new programing models that enable to harness their large processing power and to design computational fluid dynamics simulations at both high performance and low cost. We report on solving the 2D compressible Euler equations on modern Graphics Processing Units (GPU) with high-resolution methods, i.e. able to handle complex situations involving shocks and discontinuities. We implement two different second order numerical schemes, a Godunov-based scheme with quasi-exact Riemann solver and a fully discrete second-order central scheme as originally proposed by Kurganov and Tadmor. Performance measurements show that these two numerical schemes can achieves x30 to x70 speed-up on recent GPU hardware compared to a mono-thread CPU reference implementation. These first results provide very promising perpectives for designing a GPU-based software framework for applications in computational astrophysics by further integrating MHD codes and N-body simulations.

1 Introduction

We report on implementing different numerical schemes for solving the Euler equations on massively parallel architectures available today in graphics cards hardware. Euler equations govern inviscid flow and are the fundamental basis of most computational fluid dynamics (CFD) problems, which often require large computing resources due to the dimensions of the domain (space and time). Modern GPU provide efficient cost-effective computing power to potentially solve large problems and prepare for running on capability supercomputer. The purpose of this paper is to show one can efficiently perform high-order numerical schemes simulations of Euler equations on a single GPU system.

GPU used to be graphics tasks dedicated co-processors. Before the advent of the Nvidia CUDA architecture (2006), some deep knowledge of the graphics pipeline model and low-level architecture was required to adapt a CPU code to run on the GPU. In 2005, Hagen *et al.* [1] implemented the Lax-Friedrichs Euler solver using the graphics pipeline approach, and designed shaders programs in

Cg language to harness the growing computing power of the vertex and fragment processors. They obtained speedup ranging from 10 to 30 when solving a shock-bubble problem on grid with up to 1024^2 cells. Nvidia CUDA is a parallel computing architecture which introduced a new programing model based on high-level abstraction levels which avoid the former graphics pipeline concepts and ease the porting of a scientific CPU application. More recently Brandvik *et al.* [2] compared a CUDA and a BrookGPU implementation of a 3D Euler numerical scheme, using a 300,000 grid-cells domain. They report runtime speedups of 16 for the GPU implementation (running on Nvidia GTX8800) versus the reference CPU implementation (running on Intel Core2 Duo, 2.33GHz).

Let us finaly mention the ambitious and impressive work of Schive *et al.* [3] which presents a GPU-accelerated adaptive-mesh-refinement code for astrophysics applications. Overall speed-up factors of ~ 10 are demonstrated for large (4096^3 and 8192^3) effective grid size. The hydrodynamics part of this code uses a Riemann solver-free relaxation scheme.

In section 2, we briefly describe the numerical schemes used to solve the 2D Euler equations the finite volume framework. First the Godunov scheme using a quasi-exact Riemann solver is presented. Then we recall basics of the Riemann solver-free Kurganov-Tadmor scheme. Details of the GPU implementation using the Nvidia CUDA tools are given in section 3, then we report on a comparative CPU/GPU performance analysis in section 4.

2 Finite volume numerical schemes for solving the compressible Euler equations

Let us consider the two-dimensional Euler equations of hydrodynamics for an ideal polytropic gas expressing the conservation of mass, momentum and energy:

$$\partial_t \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ E \end{bmatrix} + \partial_x \begin{bmatrix} \rho u \\ \rho u^2 + p \\ \rho u v \\ u(E+p) \end{bmatrix} + \partial_y \begin{bmatrix} \rho v \\ \rho u v \\ \rho v^2 + p \\ v(E+p) \end{bmatrix} = 0, \tag{1}$$

$$p = (\gamma - 1).\left[E - \frac{\rho}{2}(u^2 + v^2)\right], \tag{2}$$

where ${}^T\mathbf{U} = (\rho, \rho u, \rho v, E)$ is the vector of conservative variables. ρ, u, v, p and E are the density, the x- and y- velocities, the pressure and the total energy respectively. γ denotes the adiabatic index, i.e. the ratio of specific heats. The value $\gamma = 1.4$ (for H_2 at temperature 100^oC) is often used in astrophysics simulation. Equation (1) can be rewritten as $\partial_t\mathbf{U} + \partial_x\mathbf{F}(\mathbf{U}) + \partial_y\mathbf{G}(\mathbf{U}) = 0$. $\mathbf{F}$ and $\mathbf{G}$ are the flux vectors. The standard approach of finite volume methods is to discretize the integral form of the system of conservation laws. This allows the discrete approximation to satisfy the conservation property. The space cell-average conserved variables vector is:

$$\mathbf{U}_{i,j}(t) = \frac{1}{|\Omega_{i,j}|}\int_{\Omega_{i,j}} \mathbf{U}(x,y,t)dxdy \tag{3}$$

where $\Omega_{i,j}$ is the elementary grid cell. In case of a cartesian grid, $\Omega_{i,j}$ is simply a square which center is $(x = i, y = j)$ of sizes Δx, Δy. An overview of modern high resolution schemes using the finite volume framework can be found in the following references [4, 5]. We will only summarize the main features of the two schemes considered here.

2.1 Multidimensional Godunov scheme

The two dimensional Euler equations in integral (conservative) form are discretized in the finite volume framework as follow:

$$\mathbf{U}_{i,j}^{n+1} = \mathbf{U}_{i,j}^{n} + \frac{\Delta t}{\Delta x}\left(\mathbf{F}_{i+1/2,j}^{n+1/2} - \mathbf{F}_{i-1/2,j}^{n+1/2}\right) + \frac{\Delta t}{\Delta y}\left(\mathbf{G}_{i,j+1/2}^{n+1/2} - \mathbf{G}_{i,j-1/2}^{n+1/2}\right) , \quad (4)$$

where the flux functions are now time and space averaged. Algorithm 1 summarizes the Godunov scheme with splitting direction technique

Algorithm 1 *Directional splitting Godunov scheme* algorithm

```
initialize $U_{i,j}^0$ buffer
initialize $n_{step} = 0$ (discrete time variable)
while $t < t_{end}$ do
  dt=computeDt(); //compute time step
  if $n_{step}\%2 == 0$ then
    Godunov(X,dt); Godunov(Y,dt);
  else
    Godunov(Y,dt); Godunov(X,dt);
  end if
  if $n_{step}\%n_{output} == 0$ then
    output_U(); //dump fluid variables arrays into a file
  end if
end while
Generate timing report
return
```

and algorithm 2 shows the pseudo-code of the main routine implementing Eq. (4) to update fluid cells $U_{i,j}$. Each time step, routine `Godunov` is called twice, one for each direction.

Algorithm 2 *Godunov time step routine (pseudo-code)* `Godunov(integer dir, float dt)`

```
apply boundary conditions to $U$
for $(i, j) \in$ {computing cells indexes} do
    • get state $U(i, j)$, compute primitive variables $^{T}W(i, j) = (\rho, u, v, p)$
    • solve Riemann problem at current cell interfaces along direction dir, i.e. compute
      Godunov state
    • compute incoming fluxes $\mathbf{F}^{n+1/2}_{i+1/2,j}$ (resp. $\mathbf{G}^{n+1/2}_{i,j+1/2}$) from Godunov state
    • update $U(i, j)$ (see Eq. 4)
end for
```

2.2 Kurganov-Tadmor central scheme

Kurganov and Tadmor [6, 7] introduced a class of Riemann solvers-free schemes based on a central approach: the solution of the Riemann problem is computed on a staggered cell, before being averaged back on the standard grid. The numerical solution is updated on the edges of the staggered grid, where it is smooth, and can be computed via a Taylor expansion, with no need to solve the actual Riemann problem.

Given the cell averages $U^n_{i,j}$, the fully discrete second order Kurganov-Tadmor scheme is a two-step predictor-corrector method. Let us define the reconstructing piecewise linear polynomial of the form:

$$\tilde{U}^n_{i,j}(x, y) = U^n_{i,j} + (x - i)U^{x,n}_{i,j} + (y - j)U^{y,n}_{i,j} \tag{5}$$

where $U^{x,n}_{i,j}$ and $U^{y,n}_{i,j}$ are partial derivative approximates. By considering averages over staggered cell (centered around $(x = i + 1/2, y = j + 1/2)$), one gets [8]

$$\begin{aligned} U^n_{i+\frac{1}{2},j+\frac{1}{2}} = \frac{1}{4}\left(U^n_{i,j} + U^n_{i+1,j} + U^n_{i,j+1} + U^n_{i+1,j+1}\right) + \\ \frac{1}{16}\left(U^{x,n}_{i,j} - U^{x,n}_{i+1,j} + U^{x,n}_{i,j+1} - U^{x,n}_{i+1,j+1}\right) + \\ \frac{1}{16}\left(U^{y,n}_{i,j} - U^{y,n}_{i,j+1} + U^{y,n}_{i+1,j} - U^{y,n}_{i+1,j+1}\right). \end{aligned} \tag{6}$$

The predictor step estimates the half time steps values $U^{n+1/2}_{i,j} = U^n_{i,j} - \frac{\Delta t}{2\Delta x}(F^n_{i+1,j} - F^n_{i,j} + F^n_{i+1,j+1} - F^n_{i,j+1}) - \frac{\Delta t}{2\Delta y}(G^n_{i,j+1} - G^n_{i,j} + G^n_{i+1,j+1} - G^n_{i+1,j})$ which are used in the corrector step to update U:

$$\begin{aligned} U^{n+1}_{i,j} = \frac{1}{4}\left(U^n_{i,j} + U^n_{i+1,j} + U^n_{i,j+1} + U^n_{i+1,j+1}\right) + \\ + \frac{1}{16}(U^{x,n}_{i,j} - U^{x,n}_{i+1,j}) - \lambda_x\left[F(U^{n+\frac{1}{2}}_{i+1,j}) - F(U^{n+\frac{1}{2}}_{i,j})\right] + \dots \end{aligned} \tag{7}$$

Let us note that updating values $U^{n+1}_{i,j}$ in the Kurganov-Tadmor scheme requires information in a larger neighborhood (5×5) compared to the Godunov scheme (3×3) due to the different ways the fluxes are calculated.

3 GPU implementation

Over the past few year, the ever growing computing power of GPUs makes them and interesting candidate for high performance general purpose computation (GPGPU). By unifying the different shaders processors, Nvidia CUDA architecture provides a new data parallel programming model which to not require graphics rendering technics knonledge. NVIDIA also introduced a C-like environment [9] much easier to use for designing scientific applications running on hybrid CPU/GPU systems. The currentGPU architecture, e.g. Tesla S1070, has 4 devices, each equipped with 240 32-bits cores working at 1.44Ghz. This system delivers up to 4×1037 Giga Floating Point Operations Per Second (GFLPOS). In addition, each device can access a 4GBytes GDDR3 memory at 110 GBytes/s. The CUDA programing model provides two high level key abstractions: a virtual hierarchy of thread blocks and the shared memory space, that make data and thread paralelism explicit. A CUDA kernel, defined as the entry point for executing parallel code on GPU are parametrized by the grid of block and block of threads dimensions. Each thread of a block has access to a common on-chip low latency memory space named *shared memory*. One of the major asset of this kind of architecture is the cross-device scalability, which makes a program blind to the actual hardware ressources on GPU device (number of multiprocessors per chip, ...). Let us mention that the advent of OpenCL language which essentially uses the same programming model concepts as CUDA allow our results to apply on other GPUs.

We developped a GPU CUDA-based implementation of the two numerical schemes described in section 2 using the same parallel programming pattern: the actual computational domain is splitted into overlapping sub-domains. The width of the ghost region clearly depends on the complexity of the numerical scheme; the Godunov scheme only requires one surrounding ghost-cell per subdomain whereas the Kurganov-Tadmor requires two. In the Godunov scheme, each inner cell only requires information from 3×3 neighborhood to solve the local Riemann problem. We also implemented kernels for computing time step as a parallel reduction and computing boundary conditions so that no transfert of data between CPU and GPU memory during a simulation time step is required except at initialization and at the end of the simulation.

4 Performance analysis

The performance of the numerical scheme is evaluated on two systems whose GPU specifications are listed in Table 1. The performance in GFLOPS is calculated by the following formula:

$$kN_xN_yN_{ts}/t * 10^{-9} \tag{8}$$

where t is the execution time, k is a numerical prefactor (340 for the Godunov scheme and 320 for the Kurganov-Tadmor scheme), N_x and N_y are the domain sizes and N_{ts} is the number of time steps of the simulation run.

Table 1. The specifications of CUDA-capable systems.

CPU	GPU	# of SP	# of SM	SP clock	GLFOPS	Mem. B/W	Mem. Capacity
Intel Core2 Q6600	GTX8800	128	16	1.35GHz	518	86.4 GBytes/s	768 MBytes
Intel Xeon L5420	Tesla S1070	240	30	1.44GHz	1037	110 GBytes/s	4.0 Gbytes

In Fig. 1 are reported the timing measurements of a 200 time steps simulation run for the two numerical schemes on both CPU (Intel Xeon L5420) and GPU (Tesla S1070). Note that the timing measurements include memory transferts between host and the graphics accelerator. By examining Fig. 1, one can notice that the CPU timings for the two numerical schemes have different scaling behaviors as simulation domain size increases. The Godunov scheme simulations behaves as expected from the algorithm complexity, i.e. $t_{simu} \sim N^2$ ($N_x = N_y = N$). This is illustrated in Fig. 1 where Godunov timing curve plotted with log-log axes has a slope of 1.95 whereas the Kurganov-Tadmor corresponding plot is characterized by the slope 2.26 significantly larger than 2. This is due to the fact that the CPU version of Kurganov-Tadmor scheme is based upon software package CentPack [3] which is not optimized regarding memory storage. However the GPU version do not need to store full grid intermediate variables because it uses the on-chip shared memory space. For small domain sizes ($N \leq 256$), the GPU runtime is almost flat. This can be explained by the fact that the GPU occupancy factor is very low (not enough block of threads to fully load the device).

In Fig. 2 are shown CPU versus GPU speed-ups corresponding to timing shown in Fig. 1. The Godunov scheme reachs a maximun speed-up of 70 for domain size larger than 1000^2 on the Tesla-based system. The Kurganov-Tadmor have very high speed-up for domain size larger than 500^2 this can be explained by the fact that corresponding CPU timings scale as N^α with alpha larger than 2 whereas the GPU timing scales as N^2. In Fig. 3 are shown the effective GFLOPS measured for the numerical schemes. Let us notice that the CPU version of the Kurganov-Tadmor scheme has a decreasing GFLOPS count as the domain size increases. Once again, this is due to the fact that corresponding CPU timings scale as N^α with alpha larger than 2.

5 Future work

This work is the first step in parallelizing astrophysical simulation codes. It is shown that that compressible Euler equations solvers can be efficiently implemented on modern GPU and speed-up above 70 can be achieved compared to a single-threaded CPU program. Although, at present only a 2D Euler solver is implemented, we believe further extension to 3D and to other fields (Poisson sover, magnetohydrodynmics,...) will provide a framework for developping new high performance simulations for astrophysics.

[3] `http://www.cscamm.umd.edu/centpack/`

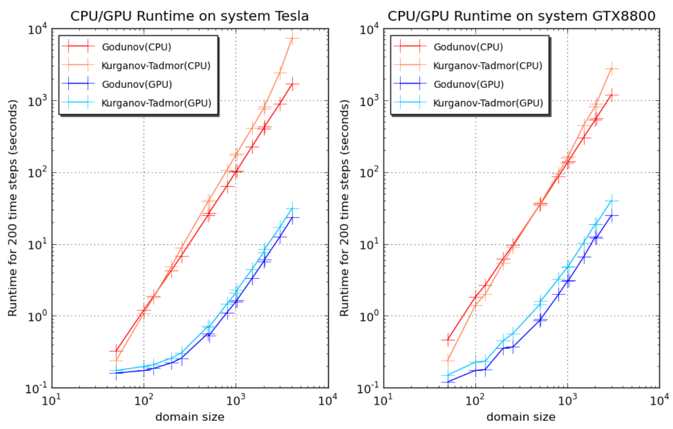

Fig. 1. Runtime (in seconds) versus grid size for a 200 time step simulation. Execution time t_{CPU} and t_{GPU} are measured on the two different hybrid systems listed in Table 1. Runtime includes buffer transfert from host memory at initialization and to host memory at the end of simulation for saving data to file on the harddrive. Left: Runtime for the Tesla-based system. Right: Runtime for the GTX8800-based systems. The red and orange plots correspond to runtime measured on CPU for the Godunov and the Kurganov-Tadmor scheme. The blue and light-blue plots are the corresponding runtime measured on GPU.

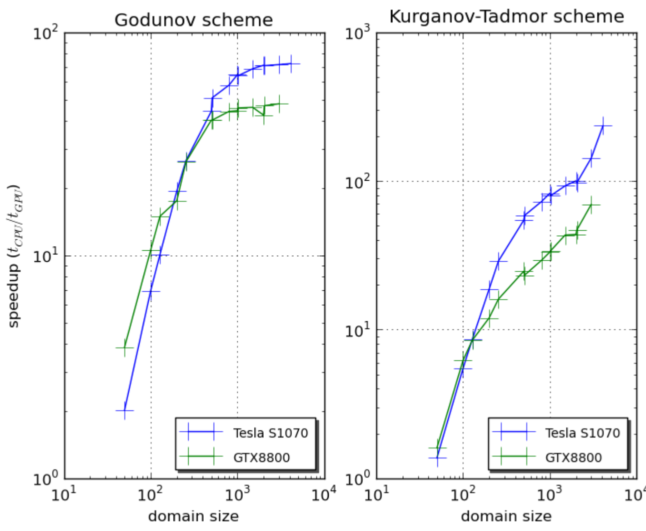

Fig. 2. Speed-up ($\mathbf{t_{CPU}/t_{GPU}}$) versus grid size. Speed-ups are computed using timings shown in Fig. 1. Left: Speed-up for the Godunov scheme simulation. Right: Speed-up for the Kurganov-Tadmor scheme.

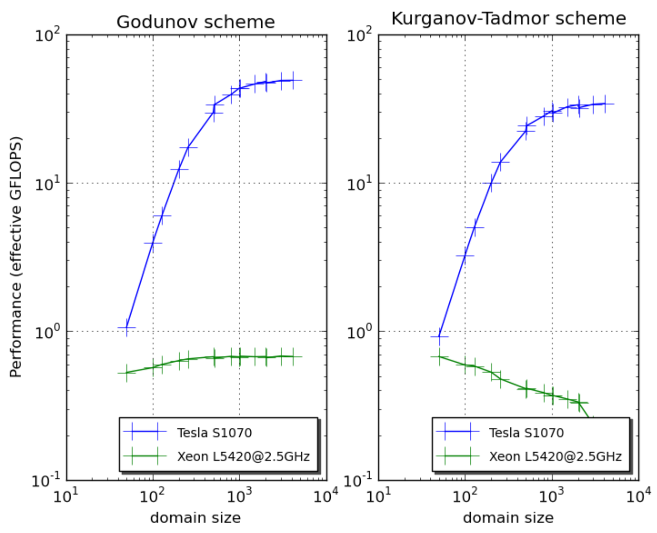

Fig. 3. Effective GFLOPS comparison.. GFLOPS are computed using Eq. (8). Left: GFLOPS for the Godunov scheme simulation. Right: GFLOPS for the Kurganov-Tadmor scheme.

References

1. Hagen, T.R., Henriksen, M.O., Hjelmervik, J.M.: How to solve systems of conservation laws numerically using the graphics processor as a highperformance computational engine. In: Quak (Eds.), Geometric Modelling, Numerical Simulation, and Optimization: Industrial Mathematics at SINTEF, Springer-Verlag (2005)
2. Brandvik, T., Pullan, G.: Acceleration of a 3d euler solver using commodity graphics hardware. In: 46th AIAA Aerospace Sciences Meeting, Reno, NV (2008)
3. Schive, H.Y., Tsai, Y.C., , Chiueh, T.: Gamer: A graphic processing unit accelerated adaptive-mesh-refinement code for astrophysics. The Astrophysical Journal Supplement Series **186**(2) (2010) 457–484
4. Toro, E.: Riemann solvers and numerical methods for fluid dynamics. A practical introduction. 2nd edn. Springer-Verlag (1999)
5. Leveque, R.: Finite Volume Methods for Hyperbolic Problems. Cambridge University Press (2002)
6. Kurganov, A., Tadmor, E.: New high-resolution central schemes for nonlinear conservation laws and convection-diffusion equations. Journal of Computational Physics **160** (2000) 241–282
7. Kurganov, A., Tadmor, E.: Solution of two-dimensional riemann problems for gas dynamics without riemann problem solvers. Numer. Methods Partial Differential Equations **18** (2002) 548–608
8. Jiang, G.H., Tadmor, E.: Nonoscillatory central schemes for multidimensional hyperbolic conservation laws. SIAM J. Sci. Comput. **19**(6) (1998) 1892–1917
9. NVIDIA: Cuda. `http://developer.nvidia.com/object/gpucomputing.html`