3D relativistic hydrodynamic computations on graphics processing units

Przemysław Duda[1], Daniel Kikoła[3], Joanna Porter-Sobieraj[2], Marcin Słodkowski[1], Daniel Kowalski[1], Jan Sikorski[1], Piotr Krzyżanowski[1], Natalia Książek[1]

[1] Faculty of Physics
Warsaw University of Technology

[2] Faculty of Mathematics and Information Science,
Warsaw University of Technology

[3] Department of Physics,
Purdue University

July 9, 2012

Contents

- Introduction,
- Motivation,
- NVIDIA CUDA Framework,
- Hydrodynamics on the GPU,
- Future plans

Problem formulation

- Fast and efficient code for hydrodynamics would enable us to study collisions event–by–event in full 3+1D.
- For this a huge amount of processing power is required.
- GPUs (Graphics Processing Units) seem to be a promising and adequate solution.
- Our project aims to implement hydrodynamic algorithms using NVIDIA GPGPU (General Purpose Computing on Graphics Processing Units) solution, namely the CUDA framework.
- Early stage of developement.

Motivation

Main reasons to invest in GPGPU:

- High performance leading to big speedups in parallel problems,
- Lower power consumption per FLOPS,
- Low price per FLOPS (Floating point OPeration per Second)
  typically 0.2–0.3 USD per theoretical GFLOPS for NVIDIA cards
  5–15 USD per GFLOPS for Intel processors

Motivation

Main reasons to invest in GPGPU:

- High performance leading to big speedups in parallel problems,
- Lower power consumption per FLOPS,
- Low price per FLOPS (Floating point OPeration per Second)
  typically 0.2–0.3 USD per theoretical GFLOPS for NVIDIA cards
  5–15 USD per GFLOPS for Intel processors
  not a perfect measure — often memory copying is the bottleneck

CUDA framework

- Programming in a C++—like language.
- A cluster of 128–512 processors which execute the *kernel* in parallel.
- Up to 4 GPUs in a single unit.

Memory types:

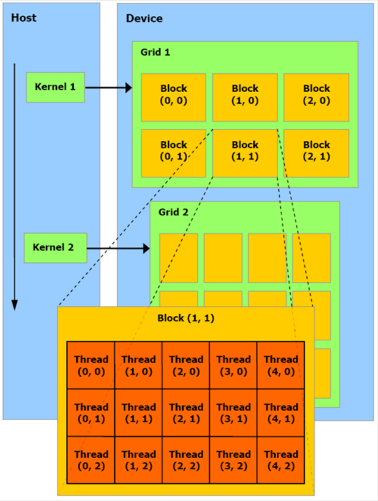
global memory big, slow, common to all threads, accessible by CPU

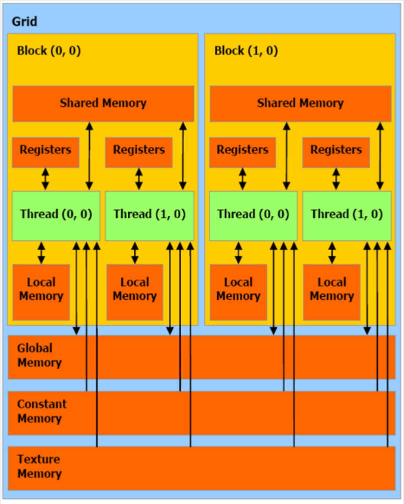
shared memory small, relatively fast, accessed by threads in a block

registers the fastest, only several per thread

Threads are divided into a grid of blocks, and are executed in parallel within a warp.

CUDA framework





images from NVIDIA CUDA Programming Guide

Hydrodynamic calculations

To solve the hydrodynamics equations:

$$\partial_t \mathbf{U} + \nabla \cdot \mathbf{F}(\mathbf{U}) = 0$$

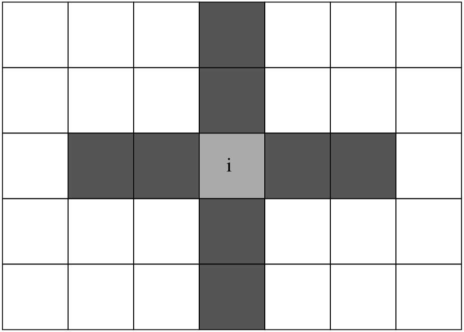
after discretization (in 1+1D):

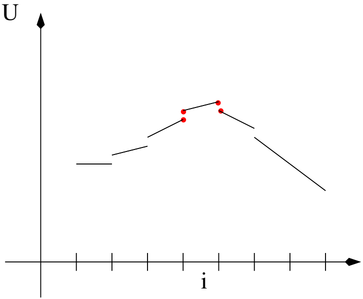
$$U_i^{n+1} = U_i^n + \frac{\Delta t}{\Delta x} \left(F_{i-\frac{1}{2}} - F_{i+\frac{1}{2}} \right)$$

we use MUSCL w/ slope limiting + Musta–Force algorithm, which gives second order accuracy in time and space.
In order to apply this scheme, each cell must access itself in the preceding timestep, and 2 neighbouring cells on each side.

Hydrodynamic calculations

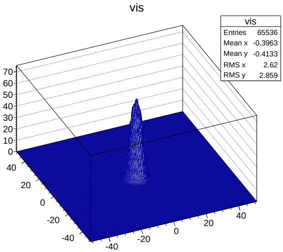
MUSCL makes a linearization inside the cells, edges (red dots) are propagated 1/2 of a timestep and given to Musta–Force, which computes final inter–cell values.

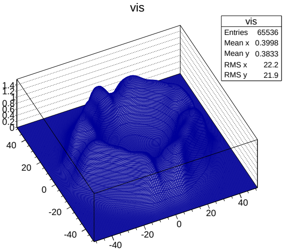


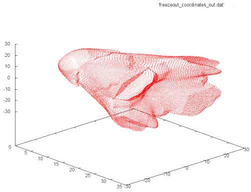


Preliminary results for 2D with UrQMD generated initial conditions

Plots show initial energy density, energy density after freezeout, and freezeout coordinates.



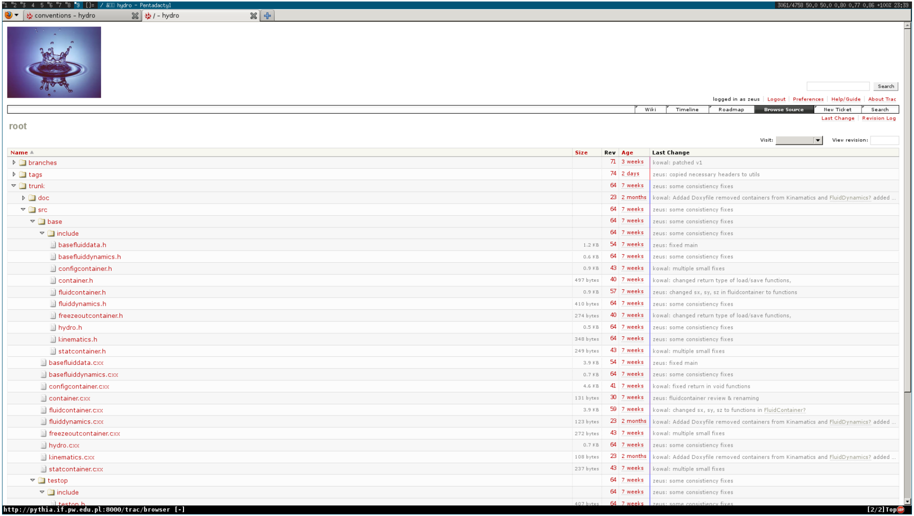




GPU implementation

- We have to map cells $\rightarrow$ threads.
- Each thread corresponds to a point in XY plane. The kernel loops over Z axis.
- In order to minimize redundant global memory reads, data is cached in shared memory and in registers.
- XY planes are cached in shared memory, neighbours in Z direction in registers.
- Blocks of threads must overlap by 4 threads in XY plane, as a border condition of size 2 is always required.

Current state of affairs



conventions – hydro | / – hydro

Search

logged in as zeus | Logout | Preferences | Help/Guide | About Trac

Wiki | Timeline | Roadmap | Browse Source | New Ticket | Search

Last Change | Revision Log

root

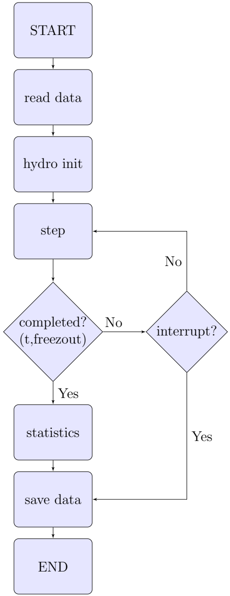
Visit: View revision:

Name	Size	Rev	Age	Last Change
branches		71	3 weeks	kowal: patched v1
tags		74	2 days	zeus: copied necessary headers to utils
trunk		64	7 weeks	zeus: some consistency fixes
doc		23	2 months	kowal: Addad Doxyfile removed containers from Kinamatics and FluidDynamics? added …
src		64	7 weeks	zeus: some consistency fixes
base		64	7 weeks	zeus: some consistency fixes
include		64	7 weeks	zeus: some consistency fixes
basefluiddata.h	1.2 KB	54	7 weeks	zeus: fixed main
basefluiddynamics.h	0.6 KB	64	7 weeks	zeus: some consistency fixes
configcontainer.h	0.9 KB	43	7 weeks	kowal: multiple small fixes
container.h	497 bytes	40	7 weeks	kowal: changed return type of load/save functions,
fluidcontainer.h	0.9 KB	57	7 weeks	zeus: changed sx, sy, sz in fluidcontainer to functions
fluiddynamics.h	410 bytes	64	7 weeks	zeus: some consistency fixes
freezeoutcontainer.h	274 bytes	40	7 weeks	kowal: changed return type of load/save functions,
hydro.h	0.5 KB	64	7 weeks	zeus: some consistency fixes
kinematics.h	348 bytes	64	7 weeks	zeus: some consistency fixes
statcontainer.h	249 bytes	43	7 weeks	kowal: multiple small fixes
basefluiddata.cxx	3.9 KB	54	7 weeks	zeus: fixed main
basefluiddynamics.cxx	0.7 KB	64	7 weeks	zeus: some consistency fixes
configcontainer.cxx	4.6 KB	41	7 weeks	kowal: fixed return in void functions
container.cxx	131 bytes	30	7 weeks	zeus: fluidcontainer review & renaming
fluidcontainer.cxx	3.9 KB	59	7 weeks	kowal: changed sx, sy, sz to functions in FluidContainer?
fluiddynamics.cxx	123 bytes	23	2 months	kowal: Addad Doxyfile removed containers from Kinamatics and FluidDynamics? added …
freezeoutcontainer.cxx	272 bytes	43	7 weeks	kowal: multiple small fixes
hydro.cxx	0.7 KB	64	7 weeks	zeus: some consistency fixes
kinematics.cxx	108 bytes	23	2 months	kowal: Addad Doxyfile removed containers from Kinamatics and FluidDynamics? added …
statcontainer.cxx	237 bytes	43	7 weeks	kowal: multiple small fixes
testop		64	7 weeks	zeus: some consistency fixes
include		64	7 weeks	zeus: some consistency fixes
testop.h	407 bytes	64	7 weeks	zeus: some consistency fixes

http://pythia.if.pw.edu.pl:8000/trac/browser [-]

Future plans

Encapsulate in a flexible, object–oriented C++ interface that allows easy integration with other models and software as a library or stand–alone program.



Draft UML diagram

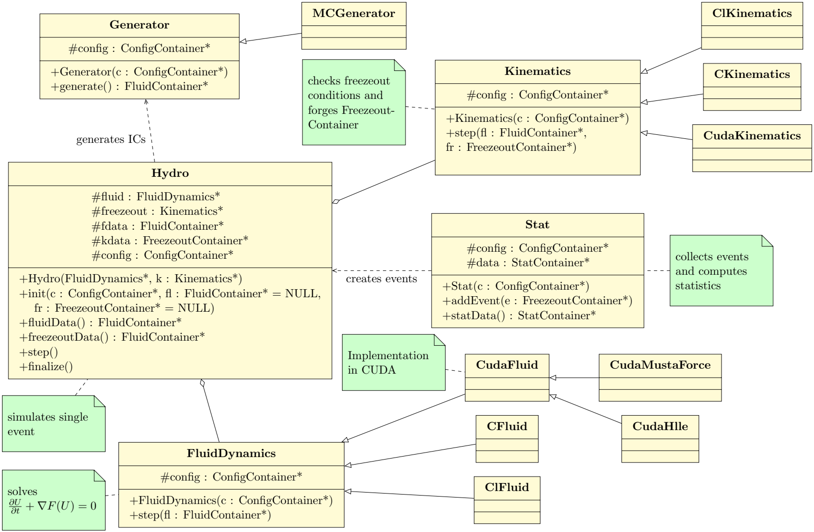


Figure 1: Class hierarchy

Thank you

Equations

Hydrodynamics equations:

$$\partial_t E + \nabla \cdot \left[(E + p)\, \vec{v} \right] = 0 \quad (1)$$

$$\partial_t \vec{M} + \nabla \cdot \left[\vec{M}\vec{v} + p\hat{I} \right] = 0 \quad (2)$$

$$\partial_t R + \nabla \cdot \left[R\, \vec{v} \right] = 0 \quad (3)$$

where:

$$E = (e + p)\gamma^2 - p \quad (4)$$

$$\vec{M} = (e + p)\gamma^2\, \vec{v} \quad (5)$$

$$R = n\gamma \quad (6)$$

In short:

$$\partial_t \mathbf{U} + \nabla \cdot \mathbf{F}(\mathbf{U}) = 0 \quad (7)$$

LAB frame variables: E, $\vec{M}$, R, $\vec{v}$.

Fluid element frame variables: e, p, n.

Integration algorithm

The scheme for equation

$$\partial_t \mathbf{U} + \nabla \cdot \mathbf{F}(\mathbf{U}) = 0$$

is

$$U_i^{n+1} = U_i^n + \frac{\Delta t}{\Delta x}\left(F_{i-\frac{1}{2}} - F_{i+\frac{1}{2}}\right) \tag{8}$$

We need to know $F_{i+\frac{1}{2}}$ and $F_{i-\frac{1}{2}}$.

Computing $F_{i+\frac{1}{2}}$ and $F_{i-\frac{1}{2}}$ — Musta–Force method

In first iteration $U_L = U_i$ and $U_R = U_{i+1}$.
Step 1:

$$F_L = F(U_L), F_R = F(U_R)$$

$$U_M = \frac{1}{2}[U_L + U_R] - \frac{1}{2}\frac{\Delta t}{\Delta x}[F_L - F_R] \quad (9)$$

$$F_M = F(U_M) \quad (10)$$

$$F(U) = \begin{pmatrix} (E+p)\vec{v} \\ \vec{M}\vec{v} + p\hat{I} \\ R\vec{v} \end{pmatrix} \quad (11)$$

$$F_{i+\frac{1}{2}} = \frac{1}{4}\left[F_L + 2F_M + F_R - \frac{\Delta x}{\Delta t}(U_R - U_L)\right] \quad (12)$$

$$(13)$$

To compute $F(U)$ we additionally need to know $\vec{v}$ and p.

Computing $F_{i+\frac{1}{2}}$ and $F_{i-\frac{1}{2}}$ — Musta–Force method

Step 2:

$$U_L^{new} = U_L - \frac{\Delta t}{\Delta x}\left[F_{i+\frac{1}{2}} - F_L\right] \tag{14}$$

$$U_R^{new} = U_R - \frac{\Delta t}{\Delta x}\left[F_R - F_{i+\frac{1}{2}}\right] \tag{15}$$

We then substitute new $U_{L,R}$ in step 1.

SLIC/MUSCL scheme

In this method we create a linear approximation of U(x) inside $\left[x_{i-\frac{1}{2}}, x_{i+\frac{1}{2}}\right]$, so that on limit points:

$$U_L = U_i - \frac{1}{2}\Delta_i \tag{16}$$

$$U_R = U_i + \frac{1}{2}\Delta_i \tag{17}$$

where Δ_i is the slope vector.
Next step is to evolve $U_{L,R}$ by a half step:

$$\bar{U}_{L,R} = U_{L,R} + \frac{1}{2}\frac{\Delta t}{\Delta x}\left[F(U_L) - F(U_R)\right] \tag{18}$$

Then we use Musta–Force, using U_R in i–th cell and U_L in i+1–th cell as initial conditions.

SLIC/MUSCL scehem — choosing Δ_i

The formula is:

$$\Delta_i = \frac{1}{2}(1+\omega)(U_i - U_{i-1}) + \frac{1}{2}(1-\omega)(U_{i+1} - U_i), \quad \omega \in [-1, 1] \quad (19)$$

To avoid oscillations we use *slope limiters* — we change $\Delta_i \to \bar{\Delta}_i = \xi(r)\Delta_i$, where $r = \frac{U_i - U_{i-1}}{U_{i+1} - U_i}$

One of choices for ξ is called MINBEE/MINMOD:

$$\xi(r) = \max[0, \min(1, r)] \quad (20)$$

Getting $\vec{v}$ and p

Using

$$M = (E + p)v$$

we have:

$$v = \frac{M}{E + p} \tag{21}$$

We obtain:

$$\begin{aligned} e &= E - Mv && (22) \\ n &= R\sqrt{1 - v^2} && (23) \end{aligned}$$

Pressure p is computed using eos $p = p(e, n)$:

$$v = \frac{M}{E + p(E - Mv, R\sqrt{1 - v^2})} \tag{24}$$

$$\vec{v} = v\frac{\vec{M}}{M} \tag{25}$$