WARSAW
UNIVERSITY OF
TECHNOLOGY
Faculty of Physics

Applied Physics
Specialization: Nuclear Physics & Technology

Master's thesis

Implementation of algorithms for relativistic hydrodynamics using graphics processing units in CUDA framework

Jan Sikorski
thesis supervisor: Marcin Słodkowski, Ph.D. Eng.

Warsaw 2013

Abstract

Relativistic hydrodynamics became a very useful tool in high-energy physics after Landau's application of this theory for explaining data on proton–proton collisions. It's later application to heavy ion collisions has been very successful in modeling apparent collective behaviour of hot matter produced in such collisions.

This work is a part of an effort of the hydro group at University of Warsaw to create an efficient and robust software tool for solving the equations of relativistic hydrodynamics, designed to be used by the high-energy physics community. Event-by-event calculations, which became a popular field of study recently, require many simulations to be performed. Because of that performance may become a considerable issue.

As a result of this master's thesis a number of numerical algorithms dedicated to solving conservative field equations have been implemented and tested. Especially the WENO schemes proved to be quite efficient and robust. A variety of slope limiting methods have been compared with a MUSTA–FORCE algorithm.

The implementation is designed to run efficiently on contemporary graphics processing units, which have many times more computing power compared to ordinary processors. Benchmarks and comparison to an equivalent implementation of some of these algorithms in C shown speedups of over 2 orders of magnitude.

Contents

Introduction

Relativistic hydrodynamics became a very useful tool in high-energy physics after Landau's application of this theory for explaining data on proton–proton collisions. It's later application to heavy ion collisions has been very successful in modeling apparent collective behaviour of hot matter produced in such collisions.

Today a lot of data is being produced in big particle colliding experiments in Brookhaven, Geneve, and other places. Developing models of events that occur in such experiments is a way of understanding the physics behind it. The hydrodynamic model plays a crucial role as a part of such models that describe the evolution of a fluid-like state of matter, which can be created in the early stage of the collision. Despite it's simplicity, it is the most successful model so far. It is thus important to develop software for evaluating this model and make effort to keep numerical and resource related issues out of the picture, as much as that is possible.

This work is a part of an effort to create an efficient and robust software tool for solving the equations of relativistic hydrodynamics, designed to be used by the high-energy physics community. Event-by-event calculations, which became a popular field of study recently, require many simulations to be performed. Because of that performance may become a considerable issue.

The physical introduction and motivation of this work is presented in chapter 1. Numerical algorithms used for solving the hydrodynamical equations are described in detail in chapter 2.

Chapter 3 introduces the technology used for implementing the algorithms, which makes the code much faster than currently available packages. It is also much cheaper to implement than a traditional computational cluster.

The following chapters 4 and 5 overviews the code and presents results that it gives for test problems and comparison with known analytic solutions.

Chapter 1

Hydrodynamic models of heavy-ion collisions

1.1 High energy physics background

The current Standard Model [1, 42, 49] of particle physics includes 3 out of 4 fundamental interactions: the strong, weak and electromagnetic forces, which govern particle dynamics. It is a theory of coupled quantum fields, which describe interactions between what is considered elementary particles.

The particles include fermions—1/2–spin particles, which constitute matter. There are 12 elementary fermions in total: 6 quarks and 6 leptons, grouped together in 3 generations:

	quarks		leptons	
generation	charge 2/3	charge $-1/3$	charge -1	charge 0
I	u (up)	d (down)	e (electron)	ν_e (e-neutrino)
II	c (charming)	s (strange)	μ (muon)	ν_μ (μ-neutrino)
III	t (top)	b (bottom)	τ (taon)	ν_τ (τ-neutrino)

Table 1.1: Fermions of the Standard Model

Where each quark can carry one of three *colors*—red, green or blue. Additionally, each of the fermions have a corresponding antiparticle, with the same mass and spin, and opposite charge and color.

In the Standard Model the neutrinos are massless. However, the experiment shows that this is not the case. Their masses have been found to be non-zero, although very small.

Gauge bosons are another kind of particles included in the Standard Model. In the perturbative treatment of the theory, they are seen as particles that carry forces—interaction is exchange of the gauge bosons. Gauge bosons can be grouped by the interaction that they carry: the photon γ carrying electromagnetic interaction, $W^{\pm}$ and Z bosons carrying weak interactions, and 8 gluons g mediating strong interactions (3 colors $\times$ 3 anticolors, one of which is a linear combination of the others).

One last particle included in the Standard Model is the recently discovered Higgs boson. It is an excitation of the Higgs field, which provides a mechanism for particles to acquire mass.

The Standard Model is formulated in the framework of gauge quantum field theory, so it has a Lagrangian that controls the kinematics and dynamics of the theory. The Lagrangian can be divided into 3 sectors:

1. the quantum chromodynamics (QCD) sector, governing strong interactions;
2. the electroweak sector, which provides an unified view of electromagnetic and weak interactions;
3. and the Higgs sector.

From the point of view of this thesis, the quantum chromodynamics part is of most interest, as the processes in heavy ion collision physics are dominated by the strong interaction.

Quantum chromodynamics qualitatively predicts stages of evolution of a ultra-relativistic heavy ion collision event (fig. 1.1). The very first are the primary interactions between partons (quarks and gluons), which lead to a highly excited, non-equilibriated state of nuclear matter. Hard scattering processes (processes with high momentum transfer) creates fireballs of hot matter, which by further rescattering cools down and equilibrates.

After a certain amount of time (estimated to be about 1–2 fm/c) the system aquires approximate local thermal equilibrium and forms the strongly coupled quark–gluon plasma (QGP). This highly exotic state of matter consists of deconfined partons, which are quasi–free in high momentum, and strongly interacting in low momentum states.

Due to further expansion the QGP evaporates and enters the hadron gas phase. Here the interactions become less energetic, and a chemical equilibrium is developed—new particles are no longer produced. At sufficiently low temperature and density the interactions stop altogether and particles move away freely. This transition is called the kinetic freezeout.

The phase diagram of nuclear matter gives another look at these processes. Although it is not well known, theoretical methods of, most notably, non-perturbative lattice QCD have given some insight into it's structures.

Facility:	SIS	NICA	SPS	RHIC	LHC
Laboratory:	FAIR GSI	JINR	CERN	BNL	CERN
Experiment:	HADES, CBM	MPD	NA61/SHINE	STAR PHOENIX	ALICE ATLAS CMS
CMS energy [GeV/N+N]:	2.3–8.5	4–11	5.1–17.3	7.7–39	5500 14000 p+p

Table 1.2: A summary of high-energy and heavy ions experiments

At very small net baryon density (a constraint that is currently approximately met only in the largest of colliders), it predicts a phase transformation between the plasma and hadronic matter, at energy density of about 1 GeV/fm^3 and a temperature near 170 MeV [55]. It this regime the transition is cross-over, up until a critical point between $\mu_B = 200$ GeV and $\mu_B = 500$ GeV, where it is expected to turn into a first order phase transition.

A central point of high energy experiments is to explore this diagram in various directions in the $T - \mu_B$ plane. Various experiments probe different regions of the diagram, giving us infomation about properties of matter in that region.

Examples include the NICA project optimized for the highest possible baryon density, RHIC energy scan that probes the region around the critical point and LHC experiments which have access to the highest energies achieved in an accelerator so far. A summary of some of most important experiments is given in table 1.2.

The most relevant observables that could verify this model of a high-energy collision are:

1. J/Ψ and Y suppresion caused by Debye screening in the plasma,
2. jet (high momentum stream of particles in the transverse plane) suppresion in the medium,
3. collective motion—the ellipsoidal flow caused by pressure gradients in peripheral collisions, and higher order harmonics of particle flow,
4. strangeness enhancement,
5. event-by-event fluctuations of various observables,
6. Bose-Einsten correlations, which are a probe for the freezeout process.

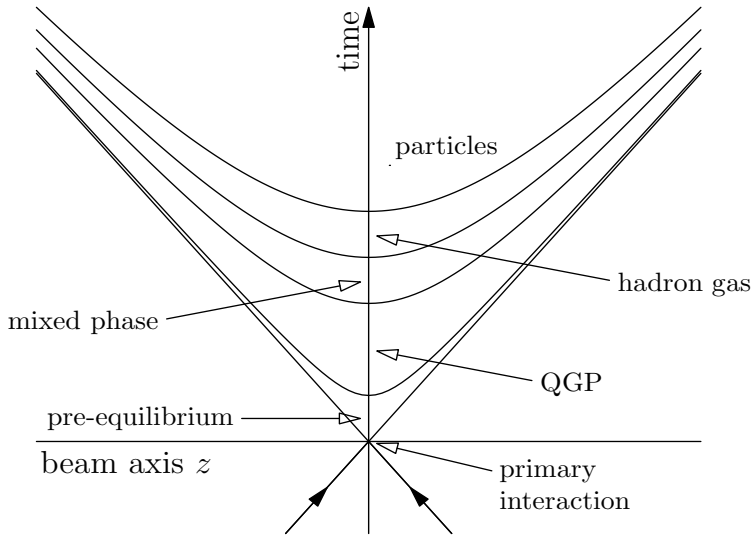


Figure 1.1: Space–time diagram of an ultra-relativistic heavy ion collision event

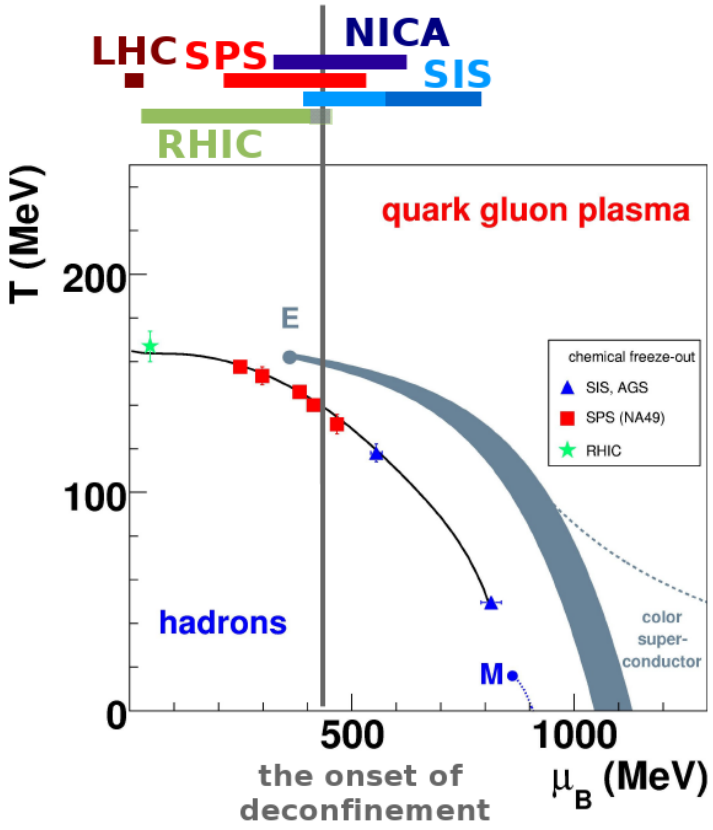
1.2 Thesis' motivation

Quantum chromodynamics has been successful in describing strong interaction in hard processes, ie. those involving large momentum transfer, such as deep inelastic scattering, or jet production. Small coupling strength in such processes, a result of a property of QCD called asymptotic freedom, allow for perturbative treatment of the theory. Despite great consilience of theoretical predictions with experimental results, we are not able to reason in the QCD framework about systems in which soft (low momentum transfer) processes are dominant.

In heavy-ion collision experiments we largely have to deal with the latter situation. After the initial partonic collisions, at sufficiently high energies, a dense system of quarks and gluons is created called the quark gluon plasma (QGP). In such system particles are strongly coupled, and perturbative QCD description falls apart, thus a different way of modeling QGP is required. Currently the most successful one is the hydrodynamical model, which models the QGP as—in its most basic form—a perfect fluid [19, 31, 27, 48, 17].

Unfortunately, even the equations of ideal relativistic hydronamics are difficult to solve analytically, and only a handful of exact solutions are known. To apply it in a realistic case (with a realistic equation of state and initial conditions) one need to resort to numerical algorithms.

To minimize time and resources required for numerical treatment of hydrodynamic systems, the simulation is often done in reduced dimensionality, or in with only several cells in one of the dimensions (usually rapidity). This approach is motivated by certain symmetries that are approximately held in heavy ion collisions. Full 3+1D,



picture based on
`https://na61.web.cern.ch/na61/xc/index.html?O=public/physicsGoals/onsetOfDeconfiment`

Figure 1.2: Phase diagram of nuclear matter

high resolution simulations are currently still too expensive to regularly perform using traditional, single threaded software.

This problem can be solved by employing a recently introduced technology of general purpose computing on graphics processing units (GPGPU), which is described in chapter 3. Thus the motivation of this thesis is lack a functional, efficient program which would perform accurate hydrodynamic computations in a reasonable time and low budget, allowing for good statistics and high spacial and temporal resolution at the same time, designed for the needs of high energy physics.

1.3 Assumptions

The hydrodynamic model considers QGP to be continuous, disregarding any kind of microscopic structure, discrete or otherwise. Evolution is thus described by a set of differential equations, and variables form fields in the spacetime. Since in heavy–ion collisions the final state consists of individual, non–interacting particles, this assumption

is violated at some point and a procedure of translating from hydrodynamic to particle description (called freezeout) is required.

Furthermore, to justify the description of QGP by fields of thermodynamical variables such as temperature and pressure, a local thermal equilibrium must be assumed. Again, in particle collision physics, the system must be given a certain amount of time to develop this equilibrium after initial hard processes. Small deviations from the equilibrium can be modelled using viscous hydrodynamics models, however here only ideal hydrodynamics will be considered.

1.4 Mathematical description

A relativistic fluid is described by the conservation of the energy–momentum tensor:

$$\partial_\nu T^{\mu\nu} = 0 \tag{1.1}$$

which, for an ideal fluid, is defined by

$$T^{\mu\nu} = (\varepsilon + p)u^\mu u^\nu - p g^{\mu\nu} \tag{1.2}$$

where ε is the energy density, p is the pressure and u^μ is the four–velocity:

$$u^\mu = \gamma(1, \mathbf{v}) \tag{1.3}$$

where $\mathbf{v}$ is the velocity vector, $\gamma = \frac{1}{\sqrt{1-\mathbf{v}^2}}$ is the Lorentz factor, and $g^{\mu\nu}$ is the Minkowski metric tensor:

$$g^{\mu\nu} = \begin{pmatrix} 1 & & \cdots & 0 \\ & -1 & & \vdots \\ \vdots & & -1 & \\ 0 & \cdots & & -1 \end{pmatrix} \tag{1.4}$$

The form of energy–momentum tensor given in eq. 1.2 can be obtained by defining the energy–momentum tensor in fluid rest frame ($\mathbf{v} = 0$)

$$T_0^{\mu\nu} = \begin{pmatrix} \varepsilon & & \cdots & 0 \\ & p & & \vdots \\ \vdots & & p & \\ 0 & \cdots & & p \end{pmatrix} \tag{1.5}$$

and applying the Lorentz transformation:

$$T^{\mu\nu} = \Lambda^\mu_\rho \Lambda^\nu_\sigma T_0^{\rho\sigma} \tag{1.6}$$

Additionally to the energy–momentum conservation, one can introduce a conserved charge n, obeying the continuity equation:

$$\partial_\mu (n u^\mu) = 0 \tag{1.7}$$

The charge is usually taken to be baryon density.

We can express eq. 1.2 and 1.7 in a convienient, conservative form

$$\begin{aligned} \partial_t E + \nabla \cdot ((E+p)\mathbf{v}) &= 0 \\ \partial_t \mathbf{M} + \nabla \cdot (\mathbf{M}\mathbf{v} + p\mathbb{I}) &= 0 \\ \partial_t R + \nabla \cdot (R\mathbf{v}) &= 0 \end{aligned} \tag{1.8}$$

where capital letters E, $\mathbf{M}$ and R are respectively energy density, momentum density and charge density in laboratory frame, given by following relations:

$$\begin{aligned} E &\equiv T^{00} = (\varepsilon + p)\gamma^2 - p \\ \mathbf{M} &\equiv T^{0i} = (\varepsilon + p)\gamma^2 \mathbf{v}, \qquad\qquad i = 1,2,3 \\ R &\equiv n u^0 = n\gamma \end{aligned} \tag{1.9}$$

To complete this set of equations one also need the *equation of state*:

$$p = p(e, n) \tag{1.10}$$

At this point it should be noted, that all of the physical information about the system (not including the previously mentioned assumptions) is contained in the equation of state. The form of this equation is still a matter of active research [46, 70, 5, 51].

1.5 Modeling the initial and final stages of the collision

As previously mentioned, models of realistic high energy heavy ion collisions involve more than just hydrodynamic expansion [40]. To complete the picture, hydrodynamic model must be connected to models describing states dominated by other phenomena.

1.5.1 Initial conditions

To start hydrodynamical evolution, an initial state is required as input.

The most basic are parametrizations based e.g. on Glauber-like models in the transverse plane (see [38] for a review), and Bjorken's solution in the longitudinal direction.

Other approaches involve models based on color glass condensate (CGC), which describe a Lorentz contracted and slowed down, fast moving particle; pQCD+saturation model [15], or the string rope model [35].

These models describe a smooth, averaged initial state. However, since the hydrodynamic equations are nonlinear, solution with an averaged initial state is not

equivalent to the average of solutions with fluctuating initial conditions. Because of this, event-by-event calculations became a major point of interest.

Fluctuating initial conditions can be obtained using e.g. Monte–Carlo Glauber [2, 8] or CGC, NeXus [14], and models like EPOS [45] or UrQMD [4]. They have been shown to improve the behaviour of elliptic flow especially in high p_T and large $|\eta|$ regions, and produce other non-trivial effects in agreement with the data [24, 9, 28].

1.5.2 Hadronization, freezeout, final state interactions

Hadronization is a method of recovering hadron distributions. The quark recombination models are best candidates for describing this process [16]. In general, they assume a universal distribution of quarks and then project quark states onto hadronic states either instantaneously, or by dynamic coalescence [40].

Second step is the particle emmision, or freezeout. An overview of freezeout techniques is presented in [20]. The most popular method is a sudden freezeout, which assumes that there is a sudden change from hydrodynamical state into free particles at some point, usually a constant temperature or proper time hypersurface [63] (for a more sophisticated choice of freezeout hypersurface see e.g. [29]). More realistic, continuous emmision models are also considered. THERMINATOR [34], which implement sudden freezeout, also take into account resonance decays.

To account for the interactions between "free" particles obtained by the freezeout procedure (which may not interact strongly, but surely can electromagnetically), the UrQMD [4] software is often used [41].

Chapter 2

Numerical algorithms for relativistic hydrodynamics

2.1 Solving the hydrodynamic equations

Equations of relativistic hydrodynamics (2.1) can be written in a shortened form:

$$\frac{\partial U}{\partial t} + \frac{\partial F(U)}{\partial x} + \frac{\partial G(U)}{\partial y} + \frac{\partial H(U)}{\partial z} = 0 \tag{2.1}$$

where $U = (E, M_x, M_y, M_z, R)$ is a vector of conserved quantities in *laboratory rest frame*, E is the energy density, M_i is the momentum density in the i-th cartesian coordinate and R is a conserved charge density (e.g. baryon number). F, G, H are vectors of fluxes of those quantities in x, y, z directions, defined as:

$$F(U) = \begin{pmatrix} (E+p)v_x \\ M_x v_x + p \\ M_y v_x \\ M_z v_x \\ R v_x \end{pmatrix}, \quad G(U) = \begin{pmatrix} (E+p)v_y \\ M_x v_y \\ M_y v_y + p \\ M_z v_y \\ R v_y \end{pmatrix}, \quad H(U) = \begin{pmatrix} (E+p)v_z \\ M_x v_z \\ M_y v_z \\ M_z v_z + p \\ R v_z \end{pmatrix} \tag{2.2}$$

where v is the velocity and p is pressure, defined by an equation of state: $p = p(e, n)$. e and n are energy and charge density in fluids *rest frame* (i.e. in a frame where velocity vanishes, $v = (0, 0, 0)$).

In numerical applications, all continuous fields have to be represented in a finite number of degrees of freedom, e.g. on a fixed numerical grid. In our program we use a finite–difference scheme on a cartesian grid. Since non–conservative methods (i.e. methods based on non–conservative variables) have been show to fail (not converge to a correct solution) if a shock wave is present in the solution [30], a conservative method is used.

2.2 Time integration

For time propagation the standard Runge–Kutta methods were employed [10]. For numerical stability only TVD (total variation diminishing) methods are used.

In general, a Runge–Kutta method for eq. 2.1 can be written in the form:

$$\begin{aligned} U^{(i)} &= \sum_{k=0}^{i-1}(\alpha_{ik}U^{(k)} + \Delta t\beta_{ik}L(U^{(k)})), \qquad i = 1,\ldots,m \\ U^{(0)} &= U^{(n)} \\ U^{(m)} &= U^{n+1} \end{aligned} \tag{2.3}$$

where the upper index without parentheses denotes the time step, upper index denotes integration step, L is a numerical recipe to calculate the negative flux gradient in 2.1 and α, β are constant coefficients given for a particular method.

For second order accuracy the following method is used:

$$\begin{aligned} U^{(1)} &= U^n + \Delta t L(U^n) \\ U^{n+1} &= \frac{1}{2}(U^n + U^{(1)} + \Delta t L(U^{(1)})) \end{aligned} \tag{2.4}$$

and for third order accuracy:

$$\begin{aligned} U^{(1)} &= U^{(n)} + \Delta t L(U^n) \\ U^{(2)} &= \frac{3}{4}U^n + \frac{1}{4}U^{(1)} + \frac{1}{4}\Delta t L(U^{(1)}) \\ U^{n+1} &= \frac{1}{3}U^n + \frac{2}{3}U^{(2)} + \frac{2}{3}\Delta t L(U^{(2)}) \end{aligned} \tag{2.5}$$

It can be seen, that apart from additional computational cost due to more evaluations of L, these methods introduce the need for an additional storage register for each of the conserved variables. As this can be an issue for large resolution simulations, a low storage version of the third order method can be used.

2.3 Method of lines

The two standard ways of solving 2.1 numerically are the *dimensional splitting* approach, and the *method of lines*. Derivation of dimensional splitting methods are based on Taylor series expansions and may give incorrect results for discontinuous solutions [59], thus the latter was chosen. For a three dimensional problem, such a scheme reads:

$$\begin{aligned} U^{n+1}_{i,j,k} &= U^n_{i,j,k} + \frac{\Delta t}{\Delta x}\left(F_{i-\frac{1}{2},j,k} - F_{i+\frac{1}{2},j,k}\right) + \frac{\Delta t}{\Delta y}\left(G_{i,j-\frac{1}{2},k} - G_{i,j+\frac{1}{2},k}\right) \\ &\quad + \frac{\Delta t}{\Delta z}\left(H_{i,j,k-\frac{1}{2}} - H_{i,j,k+\frac{1}{2}}\right) \end{aligned} \tag{2.6}$$

where U_i^n, j, k represents a conserved quantity at the discrete time t_n; Δt and Δx, Δy, Δz are time and space steps, respectively, and $F_{i-\frac{1}{2},j,k} \dots H_{i,j,k+\frac{1}{2}}$ are numerical fluxes through cell boundaries.

The central point of a particular scheme is the construction of intercell fluxes ($F_{i-\frac{1}{2},j,k}$ etc.). There are two distinct approaches for this problem: the *upwind* and *centered* schemes.

Main feature of upwind schemes is that they explicitly exploit information about wave propagation contained in the equations, usually by solving a one–dimensional Riemann problem locally. Accuracy of such schemes is highly dependent on the choice of a particular Riemann solver, which should ideally be *complete* (i.e. take all characteristic fields present in the exact solution into account).

On the other hand, centered methods do not solve the Riemann problem directly, and therefore are usually simpler and more general, at the cost of it's accuracy (given that there is a complete Riemann solver available).

2.4 Musta algorithm

In order to obtain a general and accurate algorithm, we use a hybrid Musta (multi–stage) approach [60, 59, 57]. It utilizes a centered flux in a predictor–corrector loop, solving the Riemann problem numerically, i.e. without using a priori information about waves.

The algorithm, in a one dimensional case, is as follows:

1. In order to calculate flux $F_{i+\frac{1}{2}}$ we introduce auxiliary variables $U_L^{(l)}$ and $U_R^{(l)}$ and their fluxes $F_L^{(l)}$ and $F_R^{(l)}$.
2. Set $U_L^0 = U_i$, $U_R^0 = U_{i+1}$.
3. Calculate $F_{i+\frac{1}{2}}^{(l)}$ using a centered flux, $U_R^{(l)}$, $U_L^{(l)}$, $F_L^{(l)}$ and $F_R^{(l)}$. If l reached a predefined value, stop.
4. Solve Riemann problem locally:
$$U_L^{(l+1)} = U_L^{(l)} - \frac{\Delta t}{\Delta x}\left(F_{i+\frac{1}{2}}^{(l)} - F_L^{(l)}\right), \quad U_R^{(l+1)} = U_R^{(l)} - \frac{\Delta t}{\Delta x}\left(F_R^{(l)} - F_{i+\frac{1}{2}}^{(l)}\right) \quad (2.7)$$
5. Go back to step 3.

One drawback of using such algorithm is that it's numerically more expensive than other, more conventional algorithms, for instance Shasta (Sharp And Smooth Transport Algorithm) [7].

2.5 Force flux

As a centered flux we used the FORCE (First ORder CEntered) scheme:

$$F^{\text{force}}_{i+\frac{1}{2}} = \frac{1}{2}\left(F^{\text{lw}}_{i+\frac{1}{2}} + F^{\text{lf}}_{i+\frac{1}{2}}\right) \tag{2.8}$$

where $F^{\text{lw}}_{i+\frac{1}{2}}$ is the Lax–Wendroff type flux (in terms of MUSTA auxilliary variables):

$$F^{\text{lw}}_{i+\frac{1}{2}} = F\left(\frac{1}{2}(U_L + U_R) - \frac{1}{2}\frac{\alpha\Delta t}{\Delta x}(F_R - F_L)\right) \tag{2.9}$$

and $F^{\text{lf}}_{i+\frac{1}{2}}$ is the Lax–Friedrichs type flux:

$$F^{\text{lf}}_{i+\frac{1}{2}} = \frac{1}{2}(F_L + F_R) - \frac{1}{2}\frac{\Delta x}{\alpha\Delta t}(U_R - U_L) \tag{2.10}$$

In three–dimensional case $\alpha = 3$, but other values may also be considered.

2.6 Muscl–Hancock scheme

To achieve second order accuracy in space, we extend our algorithm with MUSCL-Hancock scheme. The basic idea of this scheme is to use more cells to interpolate inter–cell values and evolve them half a time step. The algorithm is:

1. Replace cell average values U^n_i by a piece–wise linear function inside i-th cell:

$$U_i(x) = U^n_i + \frac{(x - x_i)}{\Delta x}\Delta_i \tag{2.11}$$

   In the local coordinates the points $x = 0$ and $x = \Delta x$ correspond to boundaries of the cell $x_{i-\frac{1}{2}}$ and $x_{i+\frac{1}{2}}$. The values at these points are $U^L_i = U^n_i - \frac{1}{2}\Delta_i$ and $U^R_i = U^n_i + \frac{1}{2}\Delta_i$.

2. Propagate U^L_i and U^R_i by a time $\frac{1}{2}\Delta t$:

$$\begin{aligned}
\tilde{U}^L_i &= U^L_i + \frac{1}{2}\frac{\Delta t}{\Delta x}(F(U^L_i) - F(U^R_i)) + \frac{1}{2}\frac{\Delta t}{\Delta x}(G(U^L_i) - G(U^R_i)) \\
&\quad + \frac{1}{2}\frac{\Delta t}{\Delta x}(H(U^L_i) - H(U^R_i)) \\
\tilde{U}^R_i &= U^R_i + \frac{1}{2}\frac{\Delta t}{\Delta x}(F(U^L_i) - F(U^R_i)) + \frac{1}{2}\frac{\Delta t}{\Delta x}(G(U^L_i) - G(U^R_i)) \\
&\quad + \frac{1}{2}\frac{\Delta t}{\Delta x}(H(U^L_i) - H(U^R_i))
\end{aligned}$$

3. Use $\tilde{U}_i^R$ and $\tilde{U}_{i+1}^L$ as U_L^0 and U_R^0 in MUSTA to get $F_{i+\frac{1}{2}}$.

A simple choice for the *slope* Δ_i in 2.11 is:

$$\Delta_i = (U_i^n - U_{i-1}^n) \tag{2.12}$$

which indeed results in a second–order accurate algorithm. However, as predicted by Godunov's theorem, it has an unpleasant property of producing spurious oscillations in the vicinity of strong gradients.

2.7 Slope limiting

To solve this issue, a number of flux limiting and slope limiting methods has been proposed [26, 56, 6]. We employed a slope limiting method: instead of Δ_i as in 2.12 we use $\tilde{\Delta}_i = \xi(r_i)\Delta_i$ in 2.11, where ξ is called the slope limiter and $r_i = \frac{U_{i+1}-U_i}{U_i-U_{i-1}}$.

Then one can calculate U_i^L and U_i^R using the following relations

$$U_i^L = U_i - \frac{1}{2}\xi(1/r_i)(U_{i+1} - U_i),\ U_i^R = U_i + \frac{1}{2}\xi(r_i)(U_i - U_{i-1})$$

There are a number of possible choices for ξ, each with it's own characteristics and features. One possibility is the MINBEE limiter:

$$\xi_{\text{mb}}(r) = \max(0, \min(1, r))$$

and another, called SUPERBEE:

$$\xi_{\text{sb}}(r) = \max(0, \min(2r, 1), \min(r, 2))$$

In fact these are the two most extreme slope limiters, MINBEE being the most dissipative and SUPERBEE the least. Between them is the admissible region for second order total variation diminishing limiters.

In the tests also the van Albada and van Leer limiters were used:

$$\xi_{\text{va}}(r) = \frac{r^2 + r}{r^2 + 1}$$

$$\xi_{\text{vl}}(r) = \frac{r + |r|}{1 + |r|}$$

The limiters are designed to reduce the scheme to first order accuracy near shocks, and keep higher order in smooth areas. Introducing non–linearity in this way reduces spurious oscillations and retains good accuracy of the solution.

Here it should be stressed, that results are very sensitive to the choice of slope in 2.12 and the slope limiter—both the formula for ξ, and the way that it is applied. Here the procedure was applied for each variable separately, but in general it could be applied any other way (e.g. one slope limiter could be chosen for all the variables). There is no general procedure in a multi-dimensional and non-scalar case, and some experimentation may be necessary for both a particular system of equations and maybe even a particular problem. A comparison of various slope limiters is be given in section 5.1.1.

One should also take care that the resulting scheme is symmetric, or the results on the left will differ from these on the right despite initial symmetry in the system.

2.8 The Weno scheme

Due to high numerical cost and complexity of the Musta algorithm, the Weno scheme was also implemented [52, 3, 53, 58, 50, 47, 69]. It is a method of approximating the value of a function $h\left(x + \frac{\Delta x}{2}\right)$, where h is defined implicitly as:

$$\bar{h}_i = \frac{1}{\Delta x} \int_{x_i - \Delta x/2}^{x_i + \Delta x/2} h(\xi)\, \mathrm{d}\xi \tag{2.13}$$

in our context the average $\bar{h}_i$ can either be the cell average $\bar{U}_i$ for a finite volume method, or cell flux $F(U_i)$ for a finite difference method. The output is then the left and right intercell values $U^{\pm}_{i+\frac{1}{2}}$ or the numerical flux $F_{i+\frac{1}{2}}$, respectively.

In this work the latter was chosen, because finite volume methods become complex in multi dimensional case, and their main feature—applicability to non-uniform meshes—is irrelevant when simulating heavy-ion collisions.

This however comes at a cost—this scheme only works upwind, that is for fluxes with non-negative Jacobian eigenvalues, that is

$$\frac{\partial F}{\partial U} \geqslant 0 \tag{2.14}$$

which is typically not the case.

To get around this problem a procedure called *flux splitting* is required. One divides the flux into F^+ and F^-, such that

$$F = F^+ + F^- \tag{2.15}$$

and

$$\frac{\partial F^+}{\partial U} \geqslant 0, \quad \frac{\partial F^-}{\partial U} \leqslant 0. \tag{2.16}$$

The procedure is then applied separately to both of them, mirrored with respect to the point $i + \frac{1}{2}$ in the negative case, and the results are added together.

The most common method for flux splitting is Lax-Friedrichs flux splitting:

$$F^{\pm} = \frac{1}{2}(F \pm \alpha U) \tag{2.17}$$

where $\alpha = \max_U \left|\frac{\partial F}{\partial U}\right|$ is the maximum Jacobian eigenvalue on the current stencil.

Finding these eigenvalues is difficult in 3 dimensional case with a conserved current—with a total of 5 variables, there is a 5th order polynomial to solve. Unfortunately, using a numerical algorithm in this case would likely kill the performance beyond usefulness. The derivation of the Jacobian is available in appendix B.

As a stopgap, I set $\alpha = 1$, which is the maximum possible speed in the system, with decent results (see chap. 5).

A useful thing to do would be to implement the proper eigenvalues in case without any additional conserved charge. Then one could also make the calculations in the characteristic variables, which should make the algorithm more robust.

The algorithm itself is an extension of the ENO scheme. The idea of the ENO scheme is to choose the smoothest stencil from a set of candidates to avoid oscillations near shocks, and use it to approximate the intercell value.

There are several drawbacks to this method. Firstly, in the smooth areas, choosing only one scheme "wastes" the others, which could be used to get a higher order scheme. Secondly, it takes many logical statements to choose the stencil so the efficiency on the GPU would be decreased.

In the WENO scheme instead using one of the stencils, the idea is to use a combination of all of them. Each of the stencils is assigned a weight, so that stencils containing shocks contribute less than the smooth ones. This way the non-oscillatory feature is kept and the scheme has higher order in smooth areas.

To get a $2r-1$ order accurate scheme r stencils of size r are used to approximate the value in point $i + \frac{1}{2}$:

$$S_k = \{i - (r-1) + k, \ldots, i + k\}, \qquad k = 0 \ldots r-1 \tag{2.18}$$

E.g. for $r = 3$:

$$S_0 = \{i-2, i-1, i\}, \quad S_1 = \{i-1, i, i+1\}, \quad S_2 = \{i, i+1, i+2\} \tag{2.19}$$

Then for each stencil S_k an approximation of $F_{i+\frac{1}{2}}$, F_k, is computed. Again for $r = 3$:

$$\begin{aligned} F_0 &= 2F_{i-2} - 7F_{i-1} + 11F_i \\ F_1 &= -F_{i-1} + 5F_i + 2F_{i+1} \\ F_3 &= -F_{i+2} + 5F_{i+1} + 2F_i \end{aligned} \tag{2.20}$$

Each stencil also gets it's smoothness indicator β_k:

$$\begin{aligned}
\beta_0 &= \frac{13}{12}(F_{i-2} - 2F_{i-1} + F_i)^2 + \frac{1}{4}(F_{i-2} - 4F_{i-1} + 3F_i)^2 \\
\beta_1 &= \frac{13}{12}(F_{i-1} - 2F_i + F_{i+1})^2 + \frac{1}{4}(F_{i-1} - F_{i+1})^2 \\
\beta_2 &= \frac{13}{12}(F_i - 2F_{i+1} + F_{i+2})^2 + \frac{1}{4}(3F_i - 4F_{i+1} + F_{i+2})^2
\end{aligned} \tag{2.21}$$

The final approximation $F_{i+\frac{1}{2}}$ is then given by:

$$F_{i+\frac{1}{2}} = \sum_{i=0}^{r-1} \omega_i F_i \tag{2.22}$$

where $\omega_i = \frac{\alpha_i}{\alpha_0 + \ldots + \alpha_{r-1}}$, $\alpha_i = \frac{d_i}{(\varepsilon + \beta_i)^2}$ and d_i are the linear weights, for $r = 3$:

$$d_0 = \frac{1}{10}, \quad d_1 = \frac{3}{5}, \quad d_2 = \frac{3}{10} \tag{2.23}$$

To avoid division by 0, the ε in the definition of α_i is taken to be a numerically small number compared to 1, e.g. 10^{-6}. Parameters for other values of r can be found in the literature. In this work 5th and 7th order schemes ($r = 3$, 4) were implemented.

Computed intercell fluxes 2.22 can then be substituted into equation 2.6.

2.9 Calculationg the hydrodynamic flux

To complete the chapter about numerical methods for relativistic hydrodynamics, one more detail still must be dealt with. It is easy to change from rest frame variables and velocities to conserved variables used in the integration. The inverse transformation, however, is not trivial.

It is needed though each time we compute the flux, since the equation of state is defined as a function of e and n, and not E and R. As in [48], we can invert equations 1.9 and get

$$e = E - Mv \tag{2.24}$$

$$n = R\sqrt{1 - v^2} \tag{2.25}$$

$$v = \frac{M}{E + p(e, n)} = \frac{M}{E + p(E - Mv, R\sqrt{1 - v^2})} \tag{2.26}$$

and then use a numerical algorithm to solve the fixed point equation 2.26 for v. For this purpose we used an algorithm proposed by Sławomir Biegluk, a former student of Warsaw University of Technology.

Chapter 3

General purpose computing on graphics processing units

3.1 Introduction to Gpgpu computing

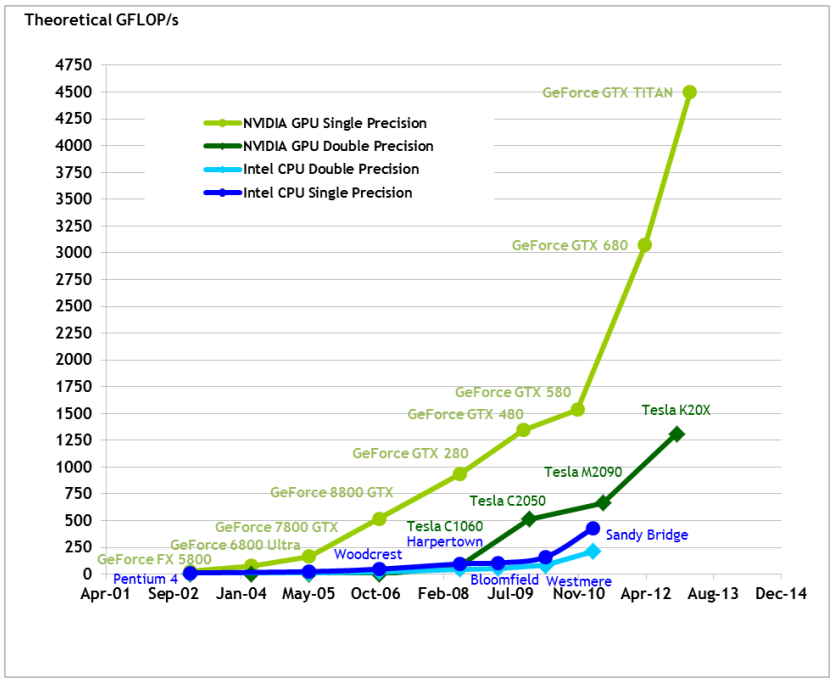
Modern graphics processing units posess an impressive computational power commonly used in 3D computer graphics. Exploiting this capacity for general use has been an active area of research for many years, but only since recently it has been possible on consumer grade hardware, using well developed software frameworks and libraries [11, 39]. Today this technology is being applied in various research areas with great success [25, 23, 62, 44, 32, 66, 43].

Processing graphics consists mainly of applying point-wise *shaders*, i.e. functions that apply pixel effects, transform geometry and compute lightning; to a massive amount of data. These problems can be described as *embarassingly parallel* problems, a term that indicates that they can be easily divided into smaller problems, that can be solved independently of each other. GPUs are thus naturally equipped with many multi-core microprocessors executing the kernel (a program) in unison, each on different input data.

An important subclass of the embarassingly parallel problems are the *stencil* computations. Often the function that will be applied point-wise also requires information about the neighbourhood of the point—e.g. when convoluting an image with a small kernel, a procedure that is commonly used in image processing. More importantly for this thesis, the process of numerically solving the partial differential equations also belong to this class.

Fig. 3.1 shows the motivation to invest in graphics processing units. On the vertical axis is a measure of raw number crunching power, the theoretical achieveable bilions of floating point operations per second. The difference is already vast in favor of GPUs

and increasing. Already now the investment is affordable, building a cluster of GPUs is going to be much cheaper than building a cluster of equivalent computing power using CPUs.



picture taken from [21]

Figure 3.1: Comparison of theoretical giga-floating operations per second for GPUs and CPUs

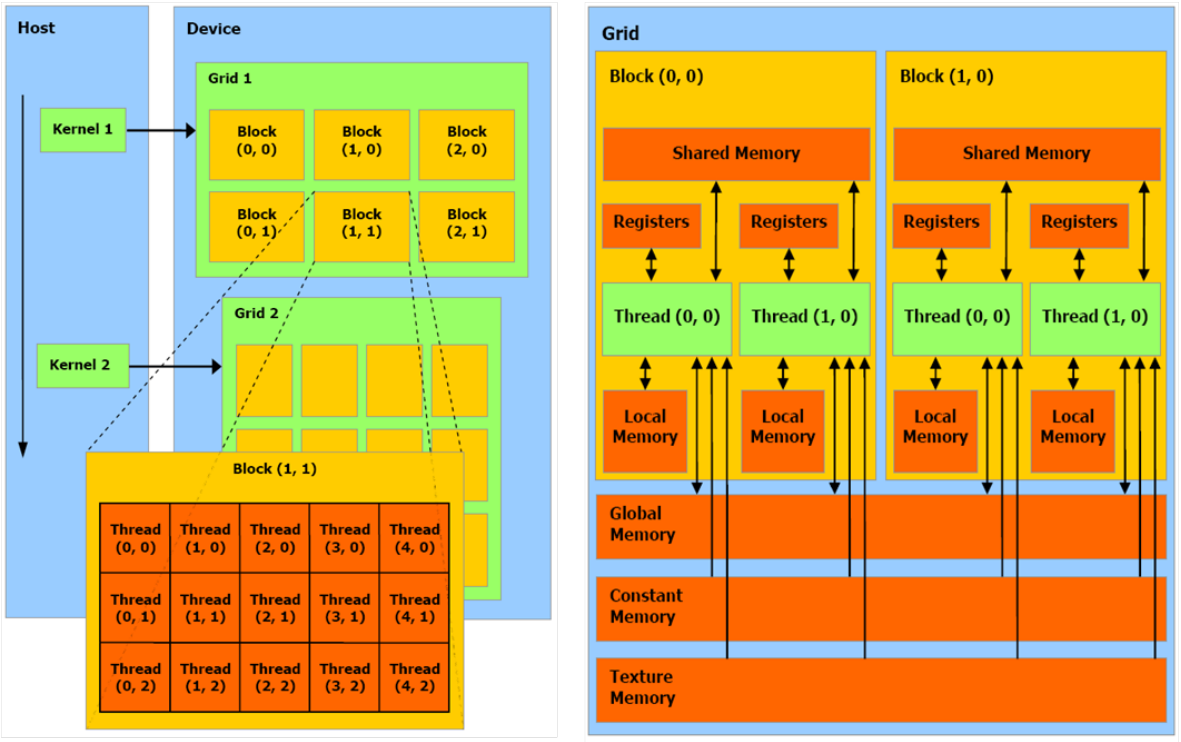
3.2 Gpu architecture

Fig. 3.2 shows an overview of the execution of host and GPU code (on the left), and types and scope of memory that is available (on the right).

The kernel is executed by the library on a grid of threads, which is divided into blocks. The importance of this division lies in a fact that threads in a block run concurrently on the device, while blocks themselves may execute sequentially. Because of that, synchronization between blocks is not possible.

From the user's point of view, the framework provides several global variables which can be used inside device (GPU) code which indicate "current" thread's index in the block, block's index in the whole grid and grid dimensions. Using these indices one can decide what should a thread do (e.g. compute memory address and fetch the data).

Another thing to point out in this section are the memory types available to device code. There are three basic types of memory storage:



picture taken from [21]

Figure 3.2: Overview of GPU architecture

registers Graphics cards usually have about 16–64 thousands of registers per multiprocessor. This number is divided by the number of threads executing on the multiprocessor, and typically a single thread has about a dozen registers allocated for use. This is the fastest and the most limited in quantity type of memory available. It is therefore of great importance to the efficiency that the program is optimized in terms of register usage.

shared memory Shared memory is an on-chip type of memory which is allocated per block. Typically a multiprocessor will have about 14–48 KB of memory of this kind available. Due to the per block allocation, this memory can only be accessed by threads within a particular block. It is often described as a manual cache and as such can provide significant optimization opportunities.

device memory This is the main bulk storage memory for storing large amount of data. Access to it is the slowest, however with the right access patterns it is possible to make a coalesced read or write operation. There are several ways to manage this memory pool, each tailored for a particular use case:

local memory this address space is per-thread, for storing intermediate calculation results that does not fit elsewhere. It is the only type of device memory

that is not globally shared by all threads;

global memory is the generic address space for exchanging data between the host and the device;

texture memory is a read-only address space optimized for spacially localized access;

surface memory similiar texture memory, but supports both read and write operations, available on newer devices;

constant memory immutable read-only memory, optimized for simultaneous access to one location.

3.3 Cuda programming framework

The Cuda platform [21, 64, 36, 65, 68, 67] provides a compiler, a set of developement tools for writing, debugging and optimizing the code, and some numerical libraries for basic tasks.

The language in which device code is written is a subset of C++, with some restrictions regarding the usage of pointers or recursion, especially in earlier versions of Cuda.

Chapter 4

Implementation of `hydro` program

4.1 Overview

The code, my contribution to which is the product of this thesis, is written in C++ as a set of classes that implement the algorithms described in chapter 2, data types to operate on and configuration tools. A program for performing the simulation and collecting results is also provided.

In this chapter the code functionality and structure will be briefly outlined. Because the code is still being developed, some of the technical details will likely change in future. Sample pieces are presented in appendix A. For more details, refer directly to the source code.

4.2 Input and output

The input consists of the configuration file with all simulation parameters except the algorithm of choice, dimension and the equation of state, which are fixed during the compilation phase and can be specified in the Makefile.

The output, apart from the final state of the fluid, can consist of a set of sections, projections, total system energy and wall clock time taken from intermediate time steps.

4.2.1 Hydrodynamic fields data

The state of the system is stored as 5 numbers per lattice cell—the energy density, conserved charge density, and 3 momenta density; all in laboratory reference frame.

The disk representation of this data is a plain text file with 5 columns of floating point numbers (energy, charge, x, y, and z momentum), separated by a space, with x being the fastest changing dimension and z the slowest. The files also contain a one line

header with information about the lattice size, which for a $x = 100$, $y = 120$, $z = 80$ lattice would look like this:

```
# 100 120 80
```

4.2.2 Parameters

Run-time parameters are given to the program in a configuration file written in a simple INI format. These include lattice dimensions, time step, number of iterations, type of border conditions and some algorithm specific parameters. An example configuration file for a simulation using the MUSTA-FORCE algorithm is shown below.

```
[main]
steps = 500
name = simulation1

[fluid]
xDim = 128
yDim = 128
zDim = 128
dx = 0.02
dy = 0.02
dz = 0.02
dt = 0.005
epsilon = 1e-6
mustaSteps = 4
border = copy

[files]
in = initial.dat
out = output.dat
```

4.2.3 Equation of state

The equation of state is an arbitrary function of the energy density and conserved charge density in the rest frame. It has to be compiled to be executed in the device code (the function signature is float eos(float e, float n), where the first argument is the energy and second the charge and the function returns pressure).

The expression for pressure can be specified in the Makefile, or directly in the source code if a more complicated procedure is required. Currently a table lookup equation of state is not supported.

4.3 Gpu–side code

For each lattice cell a number of neighbouring cells must be included in the computation. The kernel could simply read all required data from the global memory, but since operations on the global memory are expensive, some method of caching is necessary. Two different memory access patterns have been implemented for comparison. Both of them will be described in this section.

4.3.1 Shared memory version

In this version the shared memory is used as an explicit cache for holding lattice cell data. The threads are mapped to the lattice one-to-one, omiting the lattice border. The first threads in the block also load the halo around the block.

The situation is depicted on fig. 4.1 in a simplified, one-dimensional case with a block of size 4 and one-cell border.

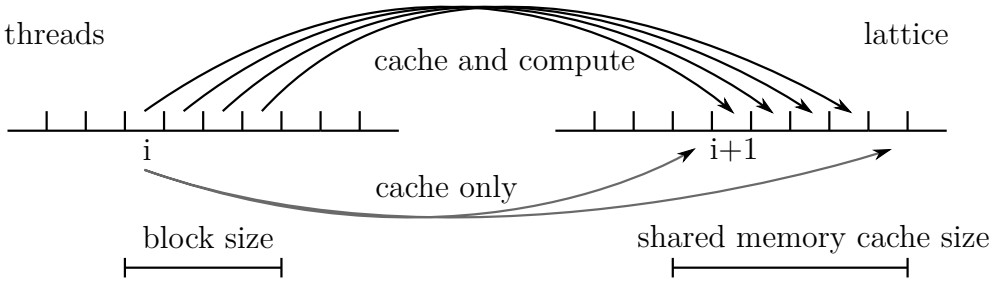


Figure 4.1: Schematic view of the threads–lattice mapping of shared memory caching and computation.

In the full 3 dimensional case the mapping takes place in the $x - y$ plane—the thread grid is 2 dimensional, and each thread loops over the z dimension. Necessary neighbouring cells in the z dimension are kept and rotated in the registers. This way both the usage of registers (less threads in a block) and shared memory is reduced.

4.3.2 Surface memory version

Using the surface memory is rather easier than managing manual cache. All threads just read the data they require from the surface object and caching is done automatically in the texture cache.

A great advantage of this is that we are not limited by the shared memory size—the stencil can have arbitrary size and shape. This feature is crucial if a very high order scheme is desired. It also lowers the register pressure caused by caching the z dimension.

The mapping is similiar as in the shared memory version—the thread grid is 2 dimensional, and each thread loops over the third dimension.

Chapter 5

`hydro` verification and performance analysis

5.1 Test problems

To verify the simulation reliability, the algorithms were tested against three analytic solutions to relativistic hydrodynamics. For each of them plots of chosen variables are presented together with the theoretical curves. The tested algorithms are Musta–Force with three different slope limiters (Minbee, Superbee, and van Albada) and without one, and two Weno schemes (5th and 7th order accurate). Third order accurate Runge–Kutta method was used for time integration.

5.1.1 Sod shock tube

First of the tests is a solution to the Riemann problem. This is a one dimensional solution, whose initial state are two regions of stationary fluid with a charge and pressure discontinuity in the middle.

When the discontinuity is big enough, a relativistic shock wave appears in the solution. The initial conditions (given in table 5.1 together with other parameters) were chosen to produce such a shock wave. The solution is divided into waves: the shock wave, the contact discontinuity, and the rarefaction wave, see fig. 5.1.

The results are presented in figs. 5.2–5.22.

For Musta–Force, a variety of slope limiters was tested. From the least diffusive cases:

no limiter Oscillations at the contact points of waves are becoming visible. The shock is also smeared out from the left side.

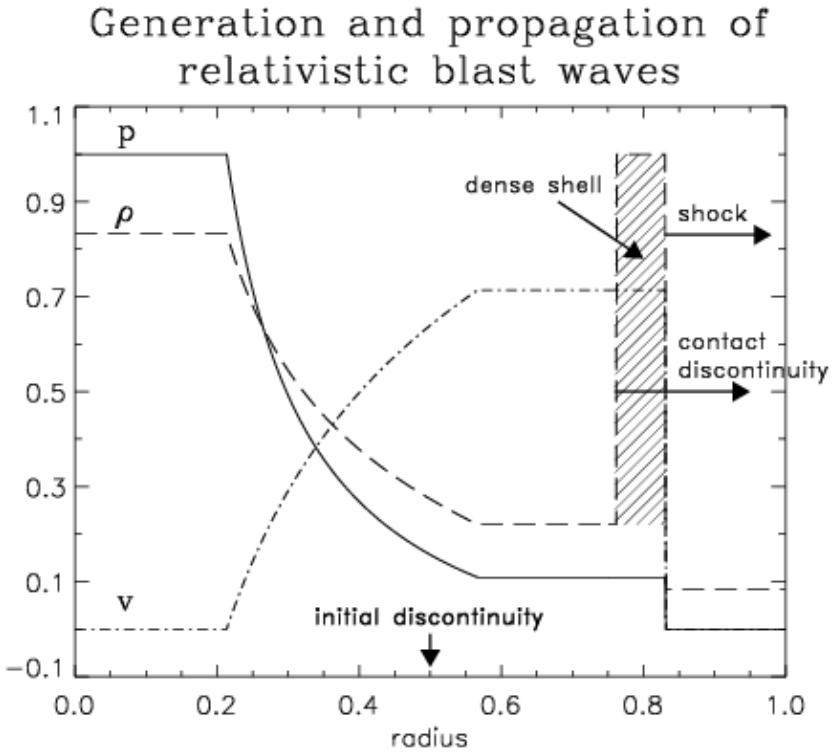


Figure 5.1: An overview of the solution of the relativistic Riemann problem (picture taken from [37])

SUPERBEE The most compressive limiter. The shocks are sharpened, but additional oscillations are introduced. Also the overshoot clearly visible in the velocity plot is enhanced.

van Leer Here the shocks are still sharp and the oscillations are gone, but the overshoot is still significant.

van Albada This limiter trades some sharpness for better rendition of the velocity profile.

MINBEE Now the overshoot is almost entirely gone, at the cost of some visible diffusion, especially in the shock wave region.

To sum up, van Albada and the Minbee limiter seem to be closest to the analytic solution, and those two will be used in rest of the tests.

Weno schemes seem to fare much better. The shock wave is sharper and more accurate. Also the overshoot is nicely supressed, especially in the 7th order scheme. The 7th order scheme though develops some slight oscillations in the contact plateau region (best visible on the velocity plot).

grid size	grid spacing	time step	p_L	p_R	ρ_L	ρ_R
500	0.02	0.005	$13\frac{1}{3}$	0	10	1

Table 5.1: Parameters of the Sod shock tube simulations

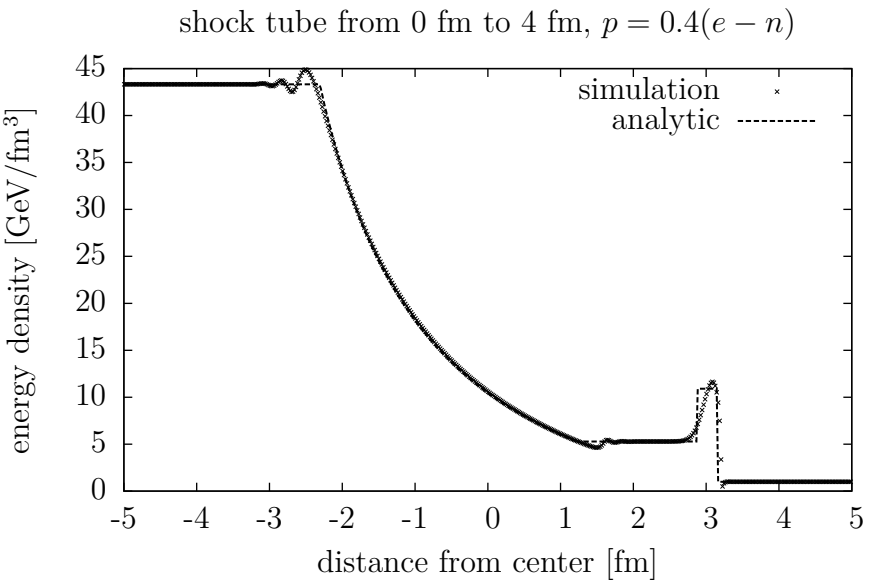


Figure 5.2: Sod shock tube, energy density in local coordinates, MUSTA-FORCE with no limiter.

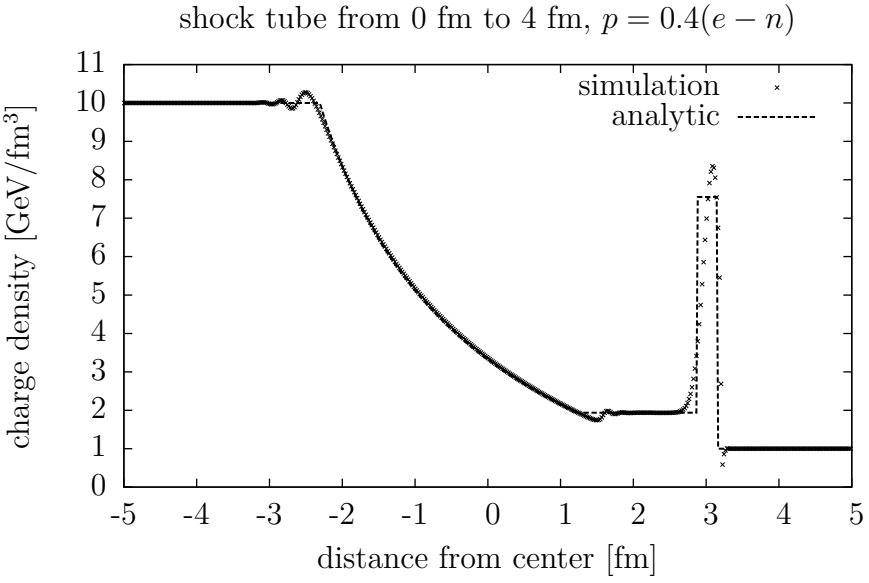


Figure 5.3: Sod shock tube, charge density in local coordinates, MUSTA-FORCE with no limiter.

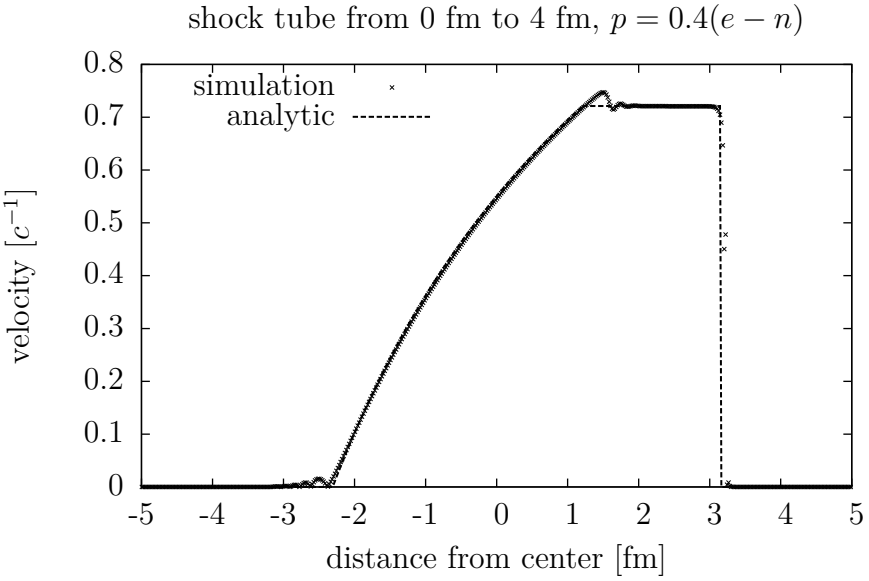


Figure 5.4: Sod shock tube, velocity, MUSTA-FORCE with no limiter.

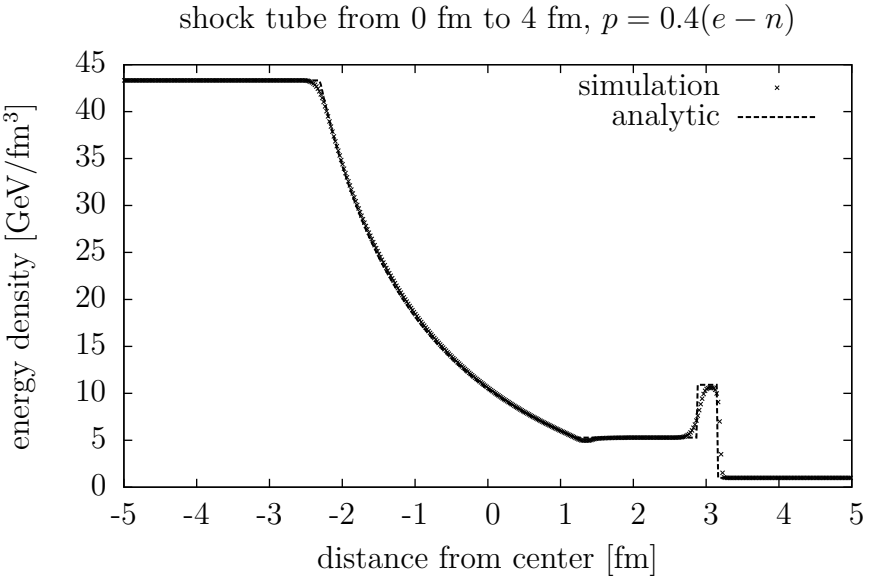


Figure 5.5: Sod shock tube, energy density in local coordinates, MUSTA-FORCE with MINBEE limiter.

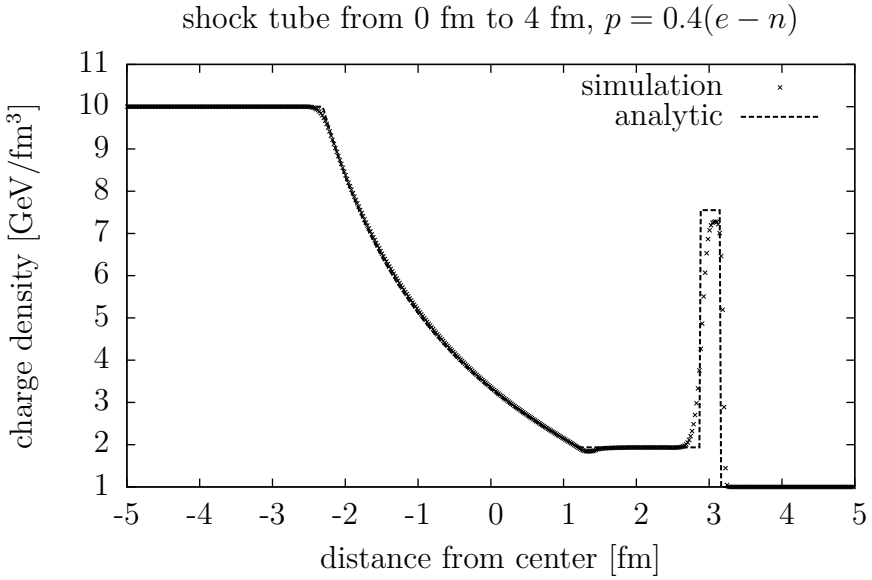


Figure 5.6: Sod shock tube, charge density in local coordinates, MUSTA-FORCE with MINBEE limiter.

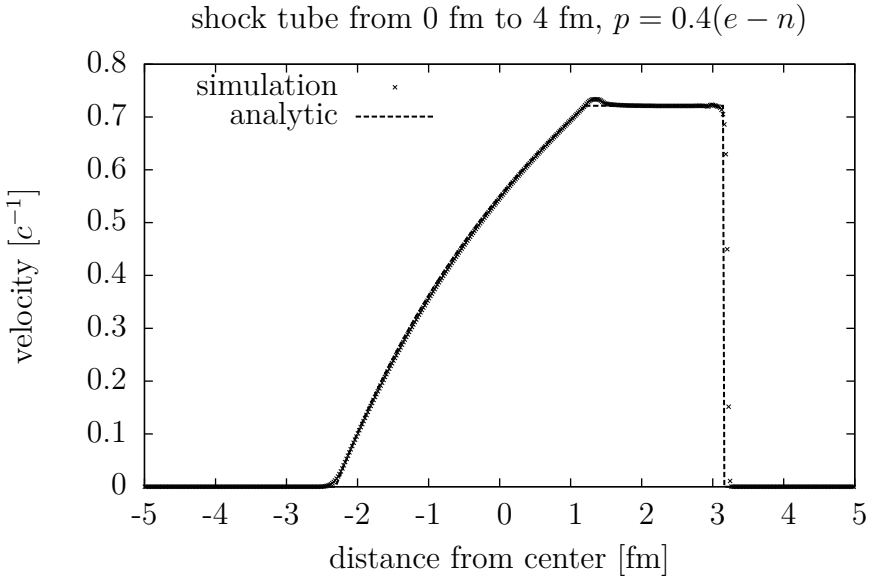


Figure 5.7: Sod shock tube, velocity, MUSTA-FORCE with MINBEE limiter.

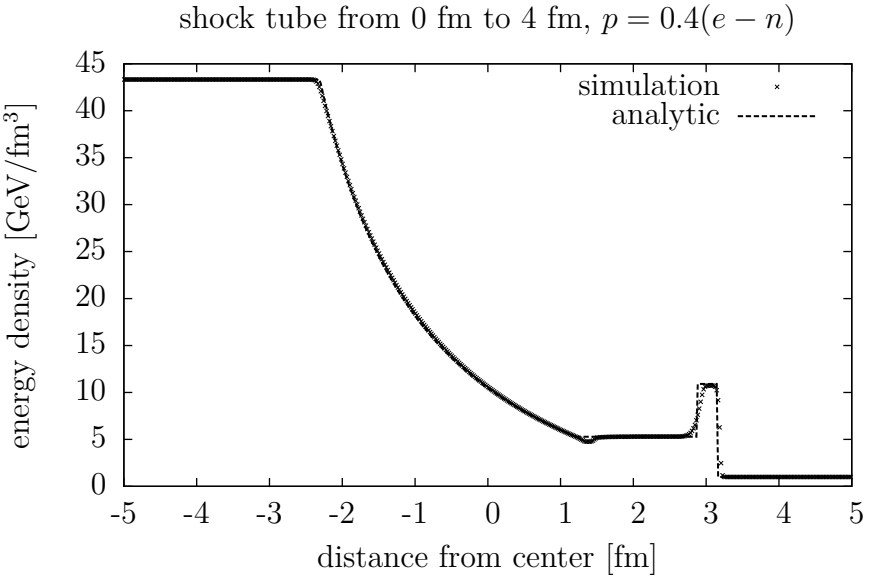


Figure 5.8: Sod shock tube, energy density in local coordinates, MUSTA-FORCE with van Albada limiter.

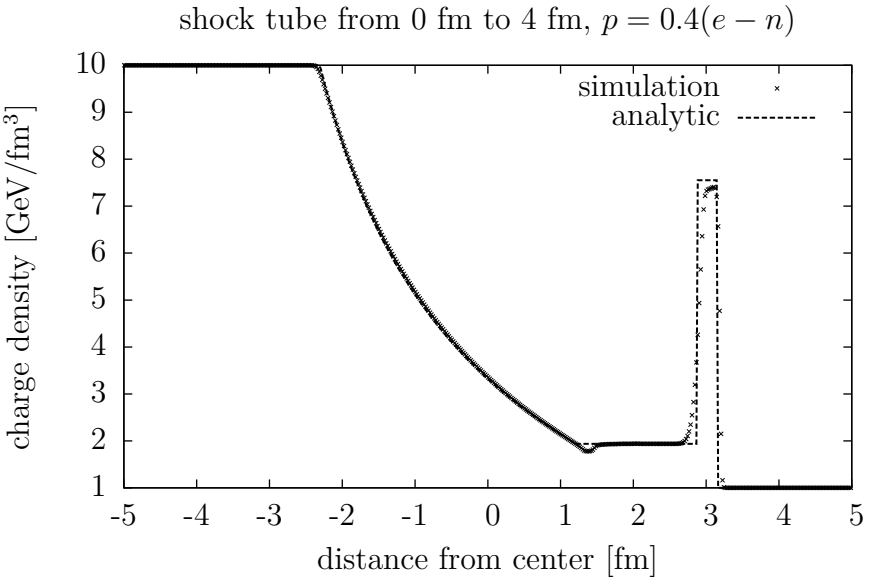


Figure 5.9: Sod shock tube, charge density in local coordinates, MUSTA-FORCE with van Albada limiter.

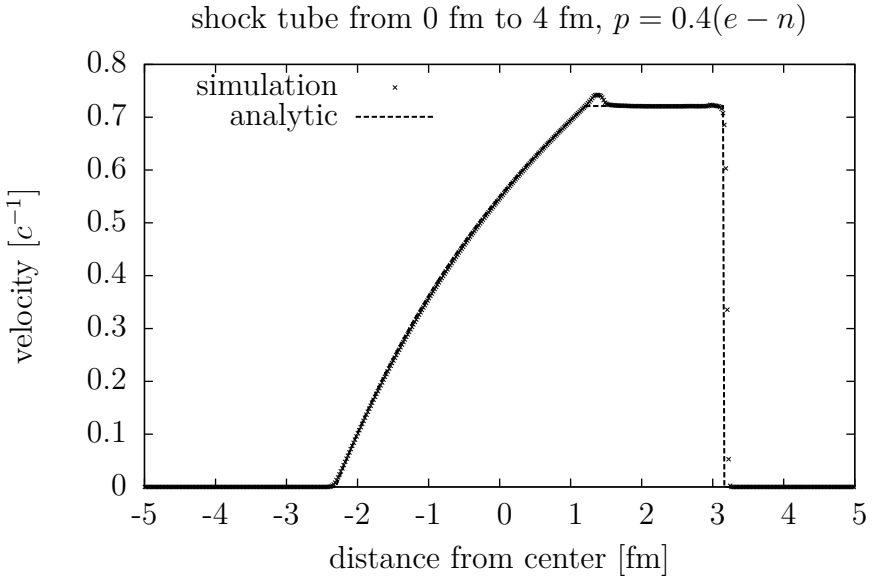


Figure 5.10: Sod shock tube, velocity, MUSTA-FORCE with van Albada limiter.

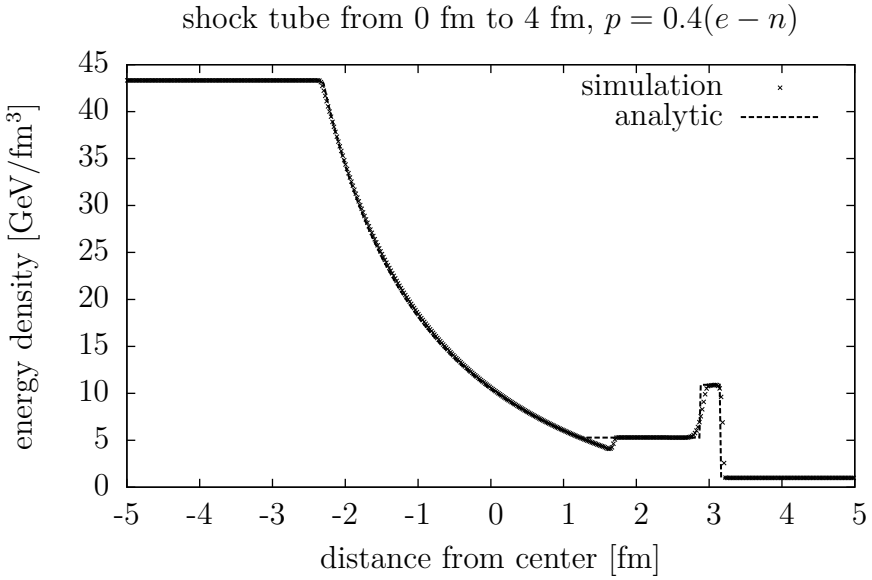


Figure 5.11: Sod shock tube, energy density in local coordinates, MUSTA-FORCE with van Leer limiter.

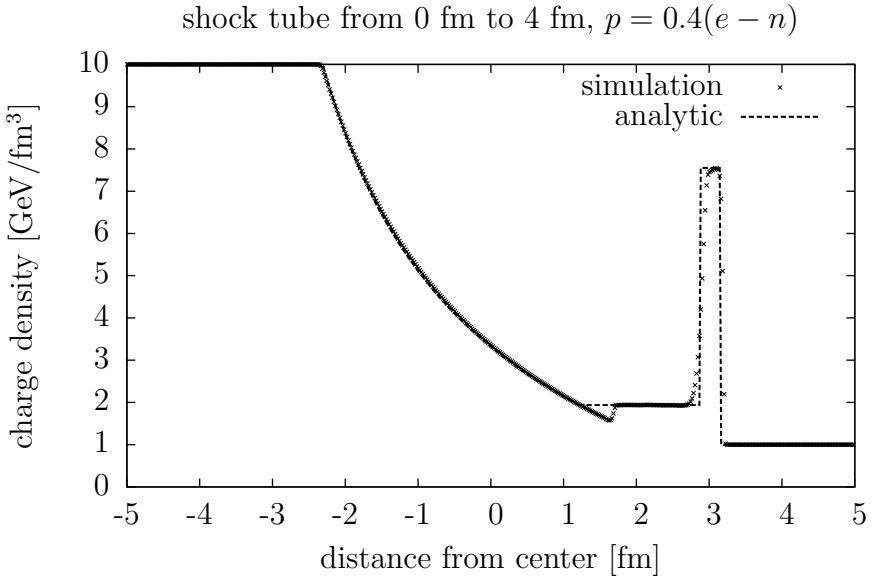


Figure 5.12: Sod shock tube, charge density in local coordinates, MUSTA-FORCE with van Leer limiter.

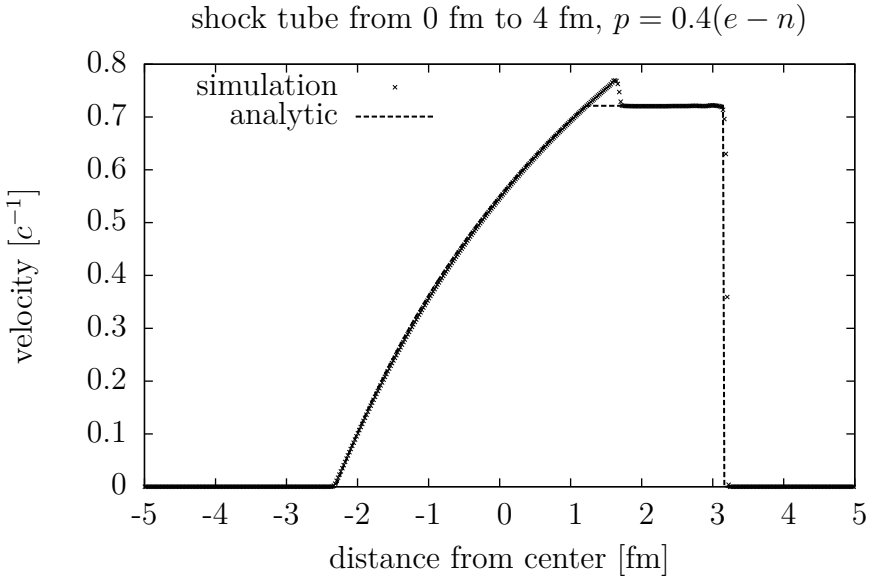


Figure 5.13: Sod shock tube, velocity, MUSTA-FORCE with van Leer limiter.

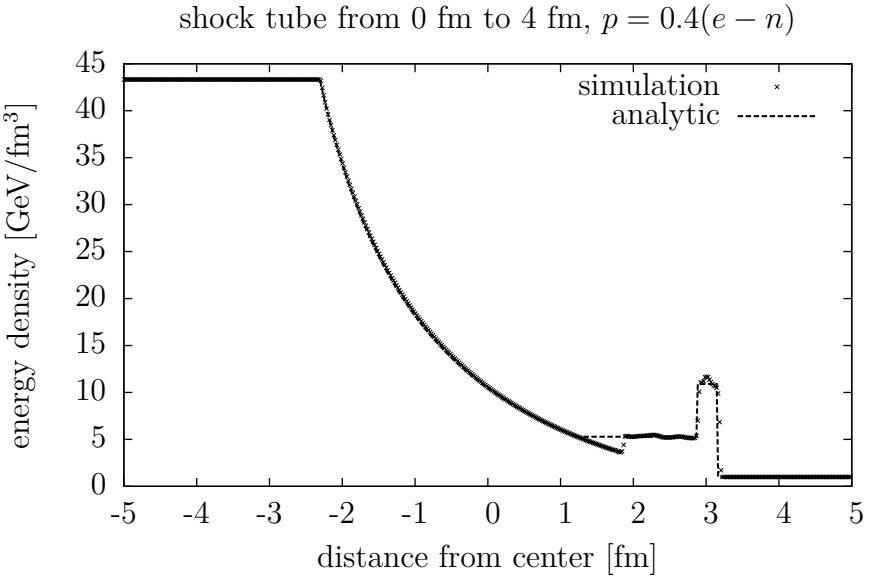


Figure 5.14: Sod shock tube, energy density in local coordinates, MUSTA-FORCE with SUPERBEE limiter.

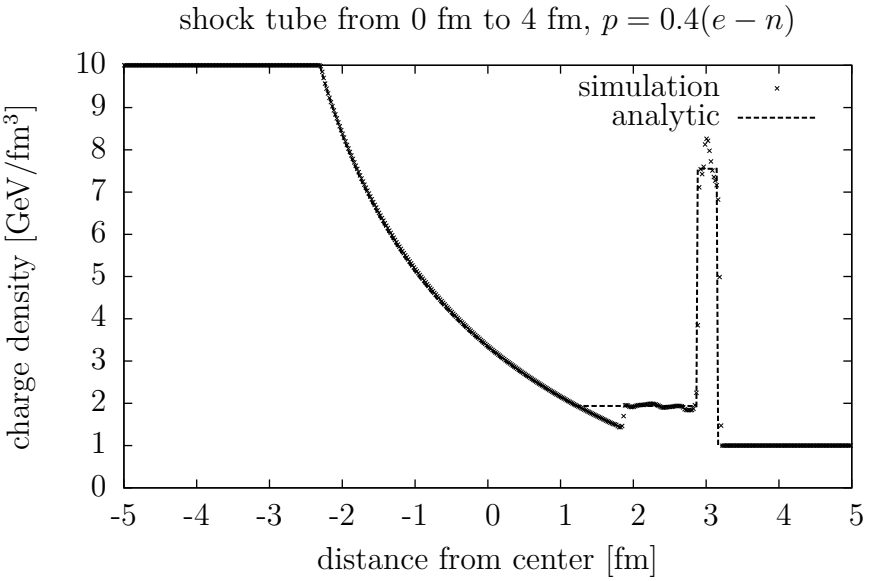


Figure 5.15: Sod shock tube, charge density in local coordinates, MUSTA-FORCE with SUPERBEE limiter.

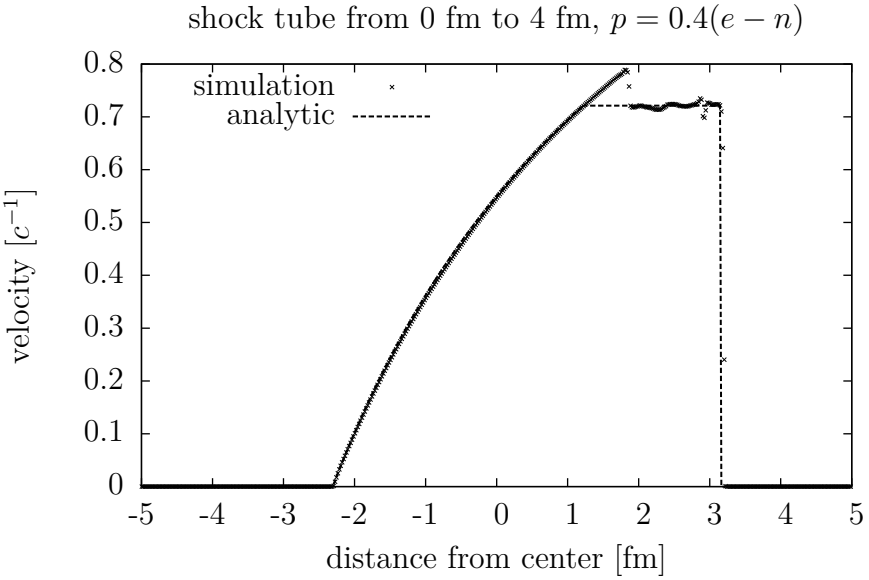


Figure 5.16: Sod shock tube, velocity, MUSTA-FORCE with SUPERBEE limiter.

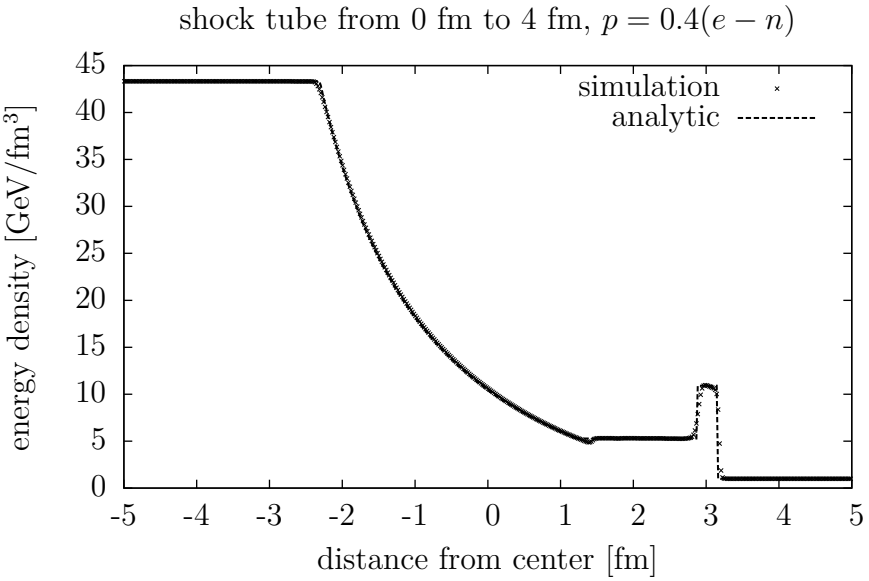


Figure 5.17: Sod shock tube, energy density in local coordinates, 5th order WENO.

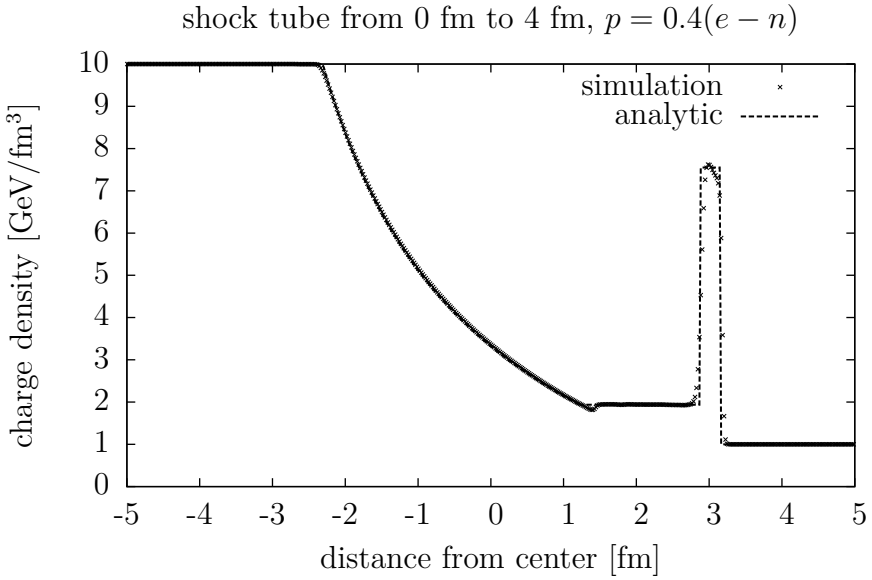


Figure 5.18: Sod shock tube, charge density in local coordinates, 5th order WENO.

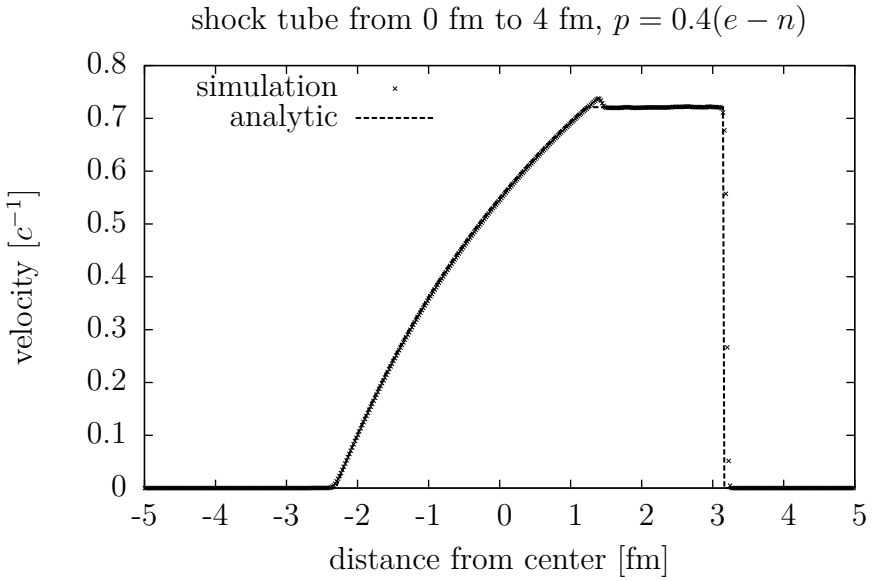


Figure 5.19: Sod shock tube, velocity, 5th order WENO.

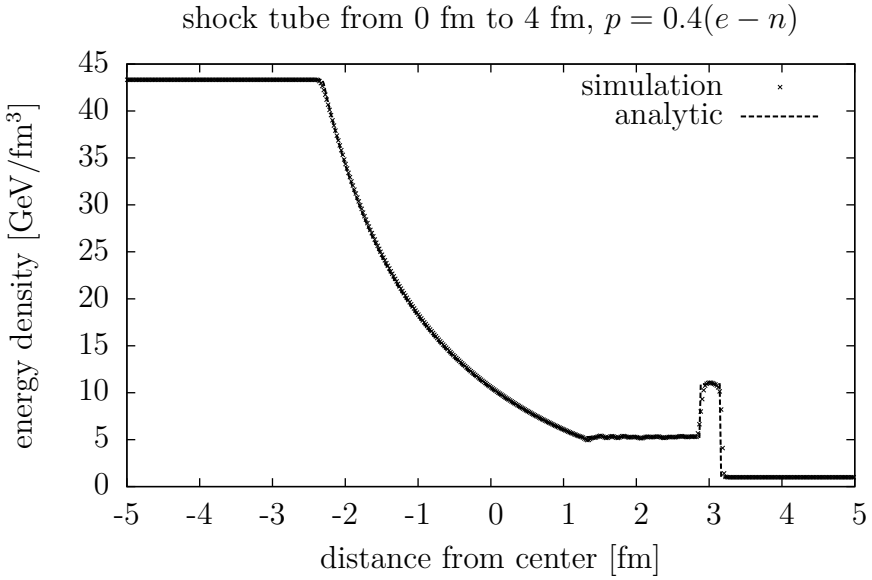


Figure 5.20: Sod shock tube, energy density in local coordinates, 7th order WENO.

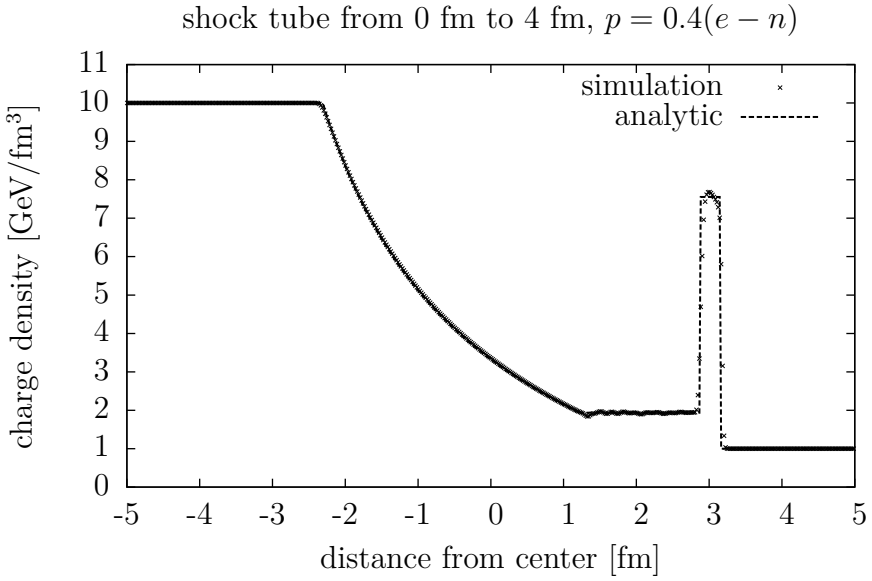


Figure 5.21: Sod shock tube, charge density in local coordinates, 7th order WENO.

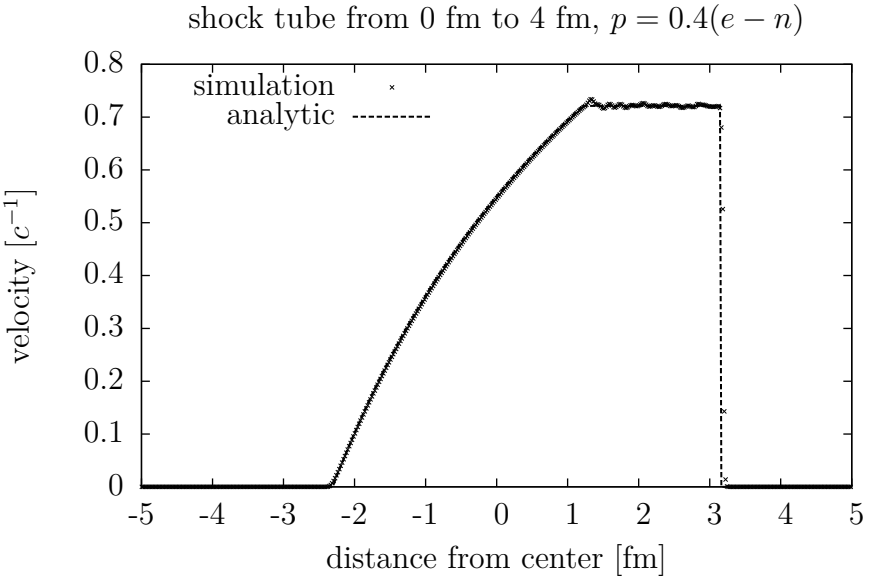


Figure 5.22: Sod shock tube, velocity, 7th order WENO.

5.1.2 Hubble–like expansion

This is a three dimensional, spherically symmetric solution of matter that expands uniformly. The velocity is proportional to distance from the center $v = \frac{\vec{r}}{t}$. The energy density is given by:

$$e = e_0 \left(\frac{\tau_0}{\sqrt{t^2 - r^2}} \right)^{3(1+c_s^2)} \tag{5.1}$$

In our case the solution is well defined for $r < t$. For the test we set $r < t - 0.5$ fm and put vacuum ($e = v = 0$) outside this region. This means that the solution is exact only in the central area—on the periphery the matter will expand into vacuum, so a rarefaction wave is expected. The solution uses ultra relativistic equation of state $p = c_s^2 e$.

Initial parameters are given in table 5.2.

grid size	grid spacing	time step	e_0	c_s^2	τ_0	t_0
120^3	0.1	0.02/0.03*	1	$\frac{1}{3}$	4	2

*0.02 for WENO, and 0.03 for MUSTA-FORCE simulations

Table 5.2: Parameters of the Hubble–like expansion simulations

The results are presented in figs. 5.23–5.34. Shown are $y = z = 0$ sections through the three dimensional solution.

The results are similiar as in previous test. All of the schemes were accurate in the middle region, and WENO schemes have shown less diffusion than MUSTA–FORCE on the sides.

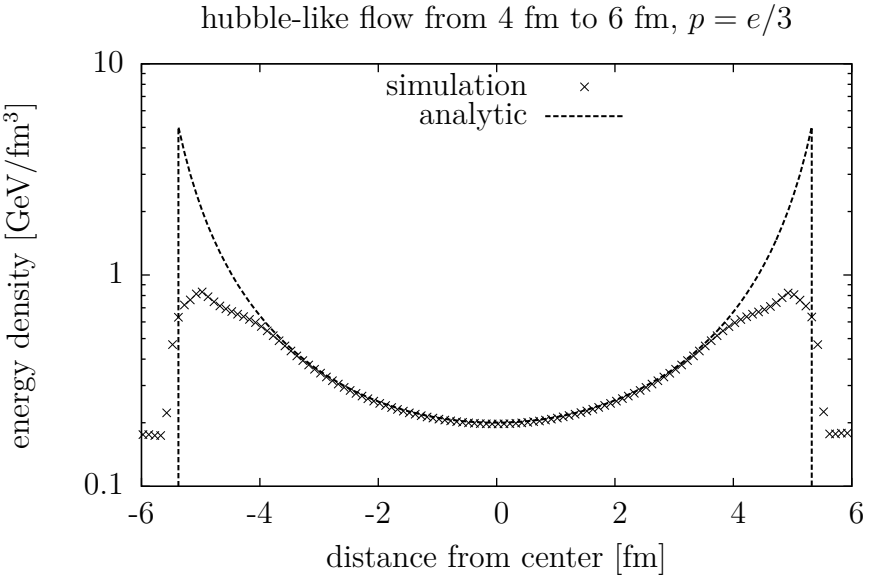


Figure 5.23: Hubble–like expansion, energy density in local coordinates, MUSTA-FORCE with MINBEE limiter.

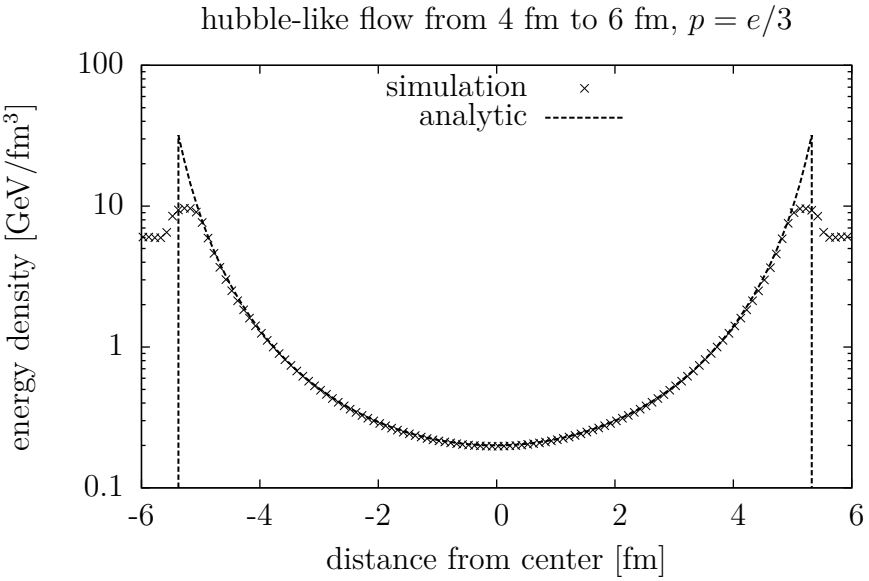


Figure 5.24: Hubble–like expansion, energy density in laboratory coordinates, MUSTA-FORCE with MINBEE limiter.

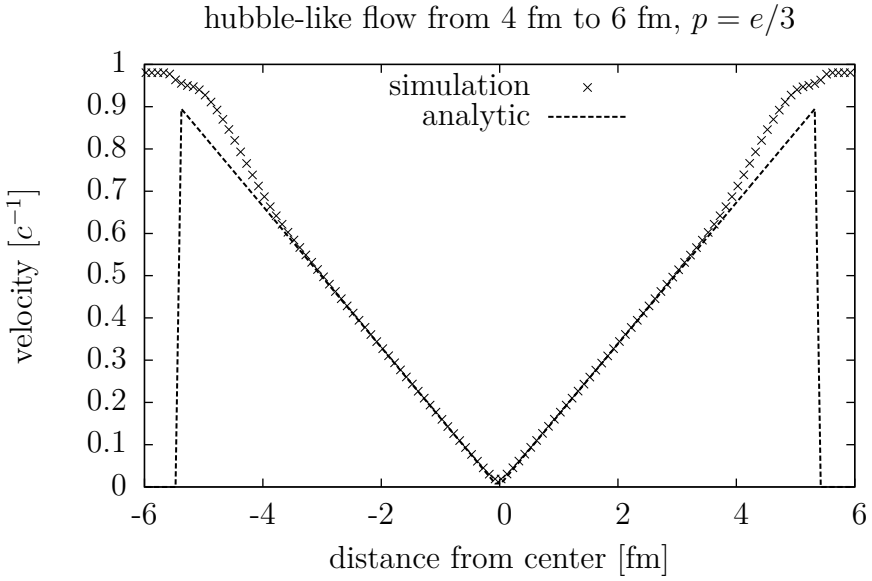


Figure 5.25: Hubble–like expansion, velocity, MUSTA-FORCE with MINBEE limiter.

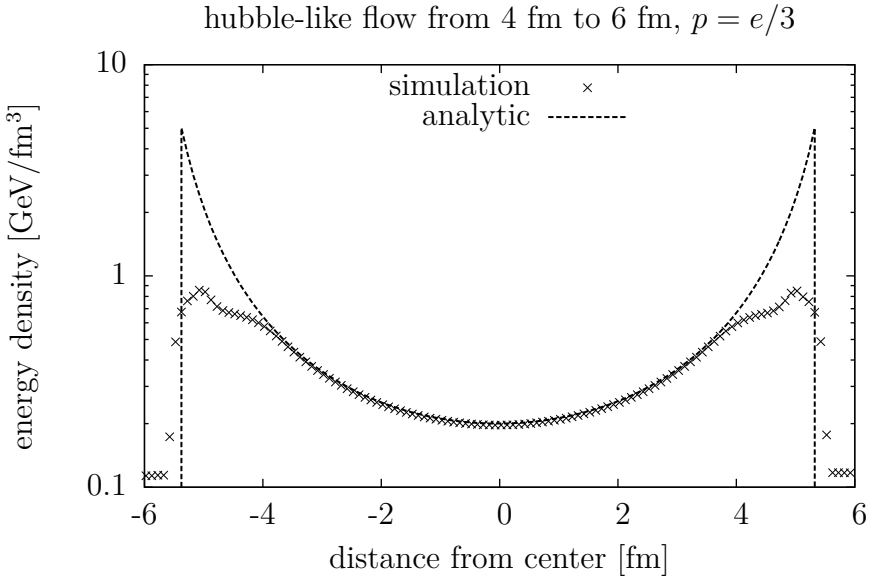


Figure 5.26: Hubble–like expansion, energy density in local coordinates, MUSTA-FORCE with van Albada limiter.

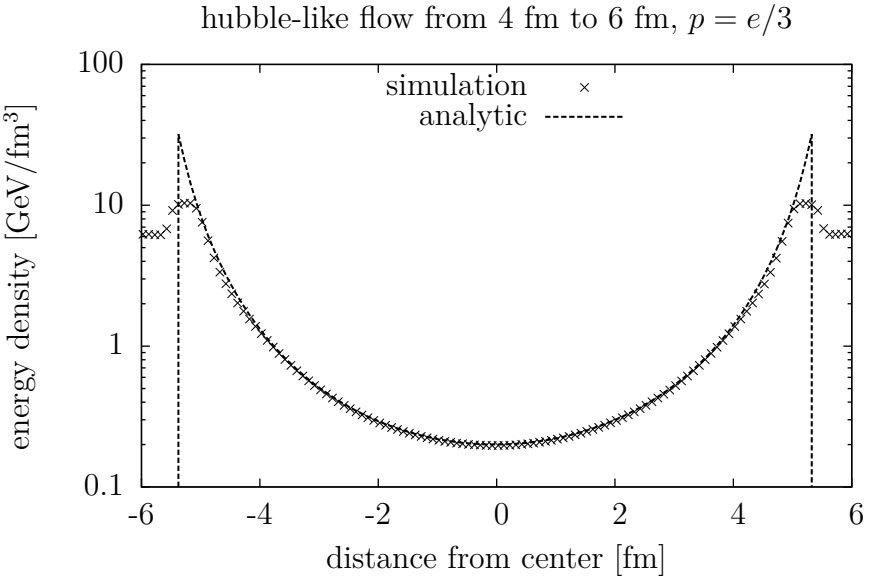


Figure 5.27: Hubble–like expansion, energy density in laboratory coordinates, MUSTA-FORCE with van Albada limiter.

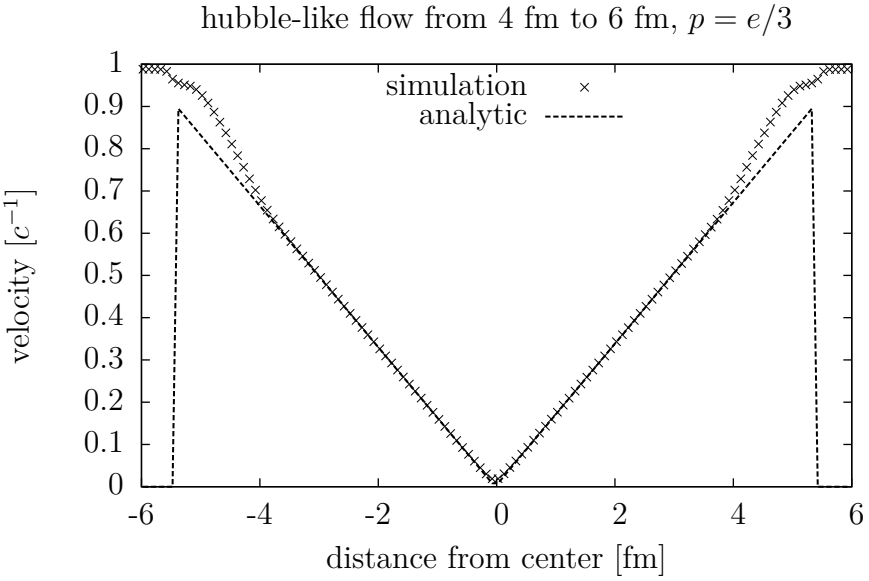


Figure 5.28: Hubble–like expansion, velocity, MUSTA-FORCE with van Albada limiter.

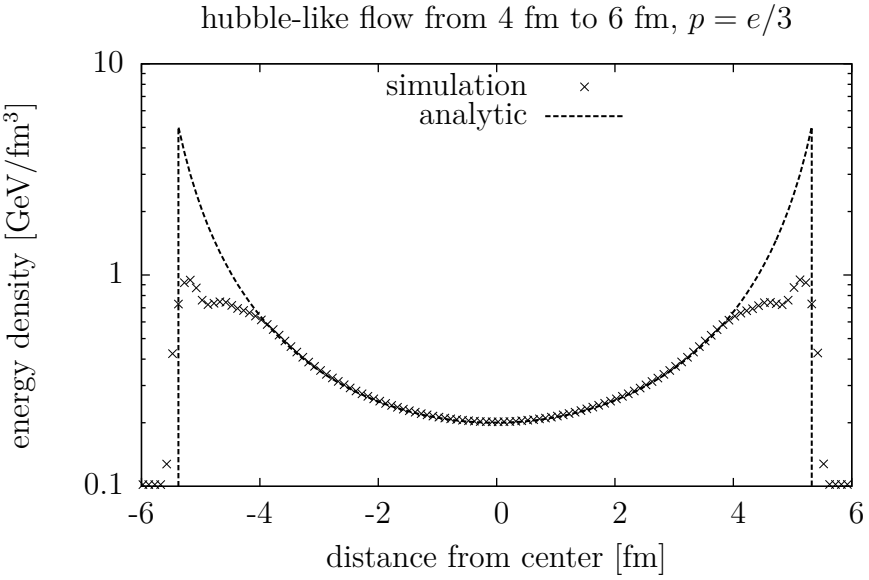


Figure 5.29: Hubble–like expansion, energy density in local coordinates, 5th order WENO.

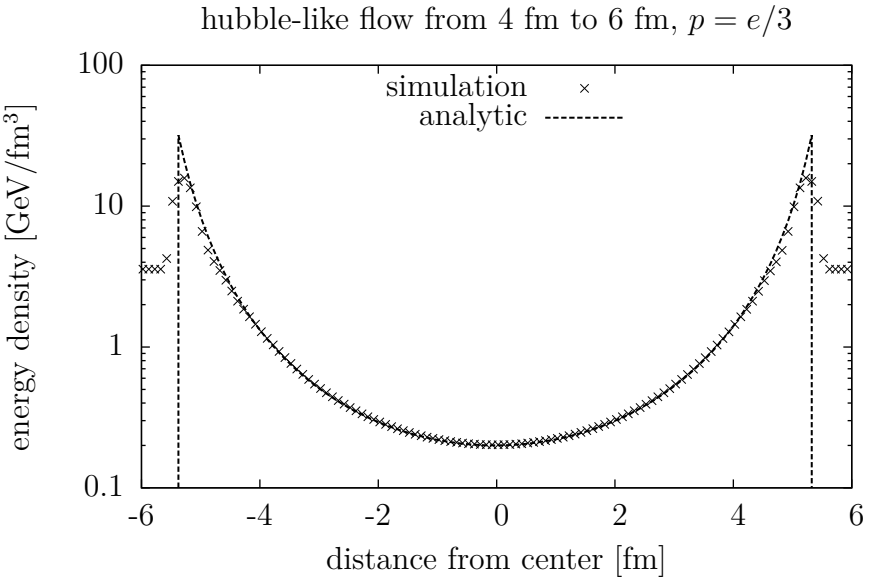


Figure 5.30: Hubble–like expansion, energy density in laboratory coordinates, 5th order WENO.

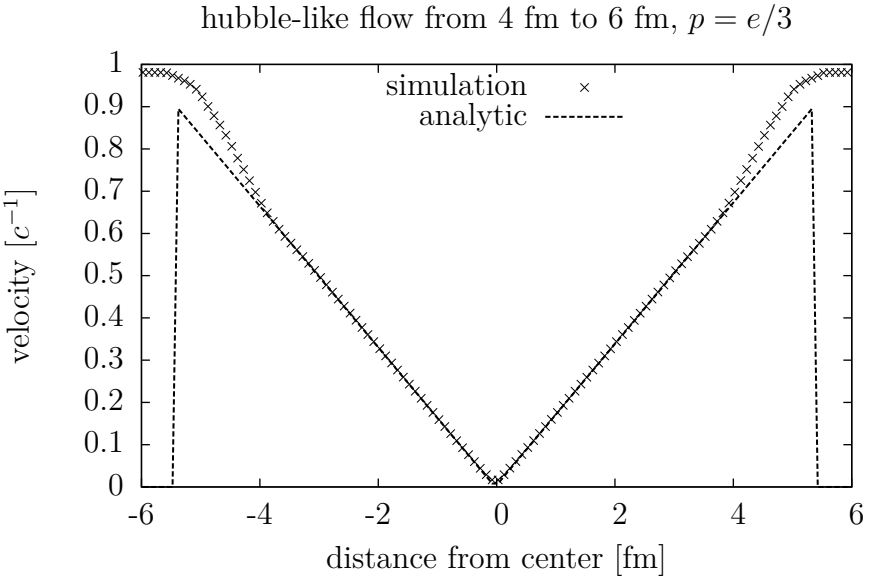


Figure 5.31: Hubble–like expansion, velocity, 5th order WENO.

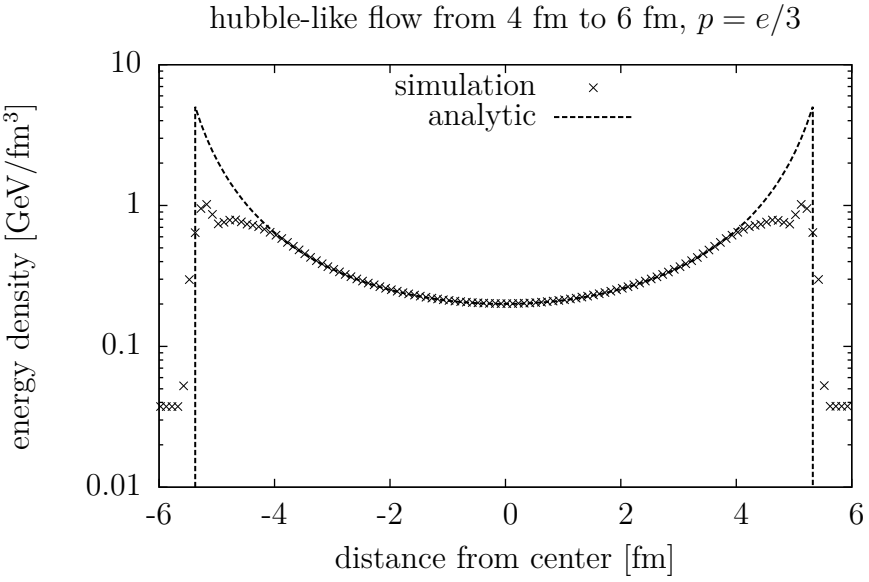


Figure 5.32: Hubble–like expansion, energy density in local coordinates, 7th order WENO.

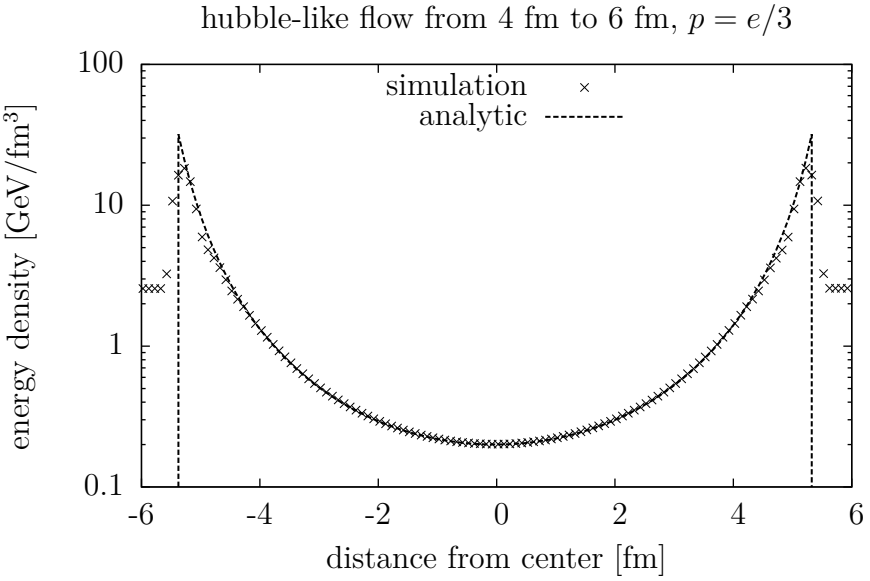


Figure 5.33: Hubble–like expansion, energy density in laboratory coordinates, 7th order WENO.

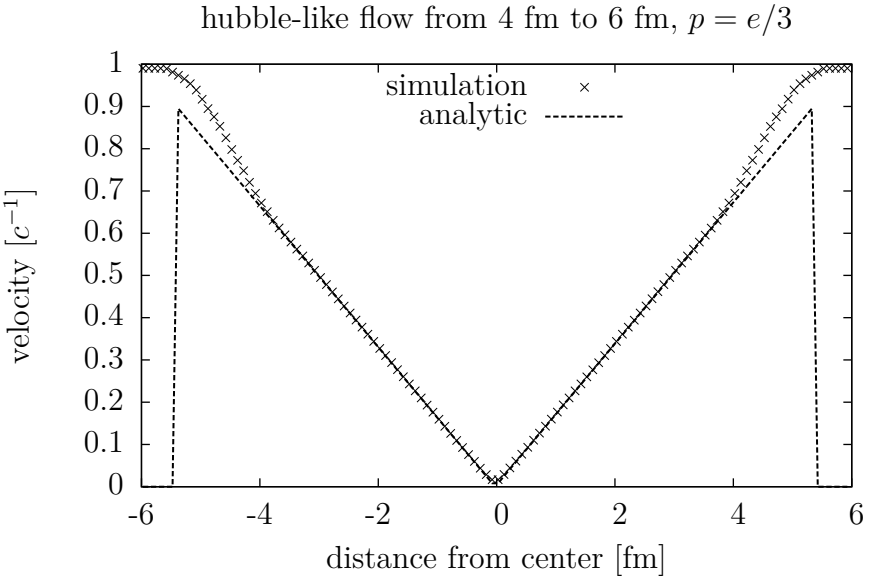


Figure 5.34: Hubble–like expansion, velocity, 7th order WENO.

5.1.3 Ellipsoidal flow

The last test uses the ellipsoidal solutions [54], which is a generalized Hubble–like solution (with velocity proportional to $\vec{r}$), a gaussian profile and vanishing pressure $p = 0$. The variables are given by the following equations:

$$e = \frac{C_e}{\prod_i (t + T_i)} \exp\left(-b_e^2 \frac{t^2}{\tau^2}\right) \tag{5.2}$$

$$n = \frac{C_n}{\prod_i (t + T_i)} \exp\left(-b_n^2 \frac{t^2}{\tau^2}\right) \tag{5.3}$$

$$\vec{v} = \left(\frac{a_1(t)x}{t}, \frac{a_3(t)y}{t}, \frac{a_2(t)z}{t}\right) \tag{5.4}$$

where $\tau = \sqrt{t^2 - \sum_i a_i^2 x_i^2}$, $a_i \equiv a_i(t) = t/(t + T_i)$ and C_e, C_n, b_e, b_n, T_i are constants; $i = 1, 2, 3$.

Initial parameters are given in table 5.3.

grid size	grid spacing	time step	t_0	C_e	C_n	b_e	b_n	T_1	T_2	T_3
120^3	0.1	0.02	2	2	0.75	1	1	0.4	0.6	0.8

Table 5.3: Parameters of the ellipsoidal flow simulations

The results are presented in figs. 5.35–5.42. Shown are $y = z = 0$ sections through the three dimensional solution.

Suprisingly, Musta–Force had some difficulties in this test. With both slope limiters it developed strong oscillations in the solution. This is rather unfortunate, since this solution despite being pressureless, resembles the most a physically relevant situation.

On the other hand, the Weno schemes had no such difficulties. The results almost perfectly fit the analytical solution, except a little velocity disturbance on the edge of the matter region.

A two dimensional section comparing both schemes is shown on fig. 5.43. Musta–Force shows clear anisotropy—the oscillations are aligned with x and y axes. This effect is absent in the case of Weno scheme. In fact such anisotropy has been observed only in this test case.

5.2 Conservation of energy

Plots of total energy on the lattice are shown on fig. 5.44.

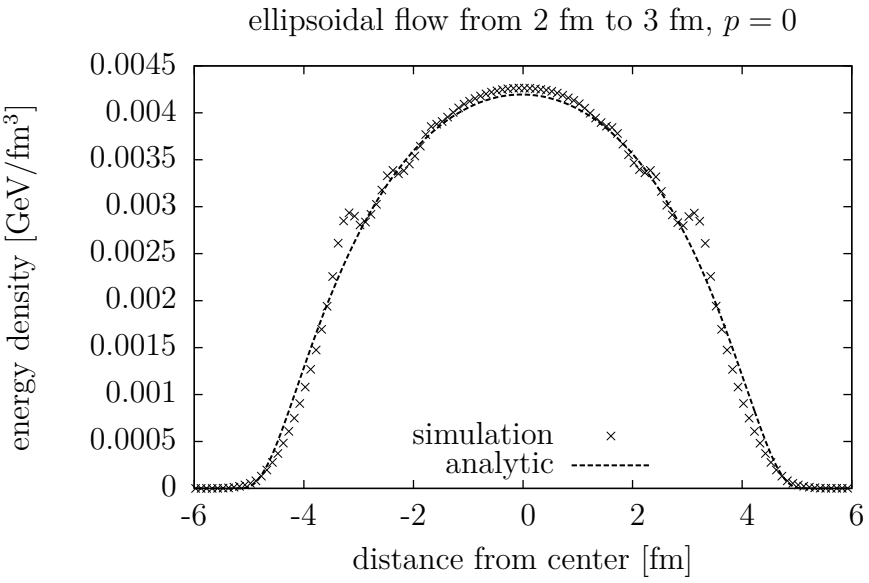


Figure 5.35: Ellipsoidal flow, energy density in local coordinates, Musta-Force with Minbee limiter.

In the case of ellipsoidal flow 7th order Weno performs the best with a deviation of the order of 0.01%. Slightly worse is Musta-Force, getting up to an 0.04% excess of energy. Here the 5th order Weno performed the worst, sliding down 0.1%.

In the hubble-like test the situation is somewhat different: here 5th order Weno performs best with about 0.03% deviation. Musta-Force is second again with a dropdown about 2 times as big. 7th order Weno is the worst, it's deviation gets up to 0.3%.

Overall the energy behaviour, despite being somewhat unpredictable, is mostly good. If, however, it becomes a problem, one should switch to a different, better behaving time integration scheme.

5.3 Performance analysis

A detailed analysis of the performance is discussed in related works [12, 13]. In this section I will present a summary of these results.

For grids bigger than about 64^3 the Gpu version turned out to be more than 200 times faster than an equivalent algorithm executed on the Cpu (comparison was done using Intel Pentium B960, a 2.2 GHz processor and a NVIDIA GeForce 610 graphics card with 1 GB of memory and Compute Capability 2.1).

A comparison of shared memory and surface memory implementations was also conducted. Although in principle the shared memory is faster, the results favor the

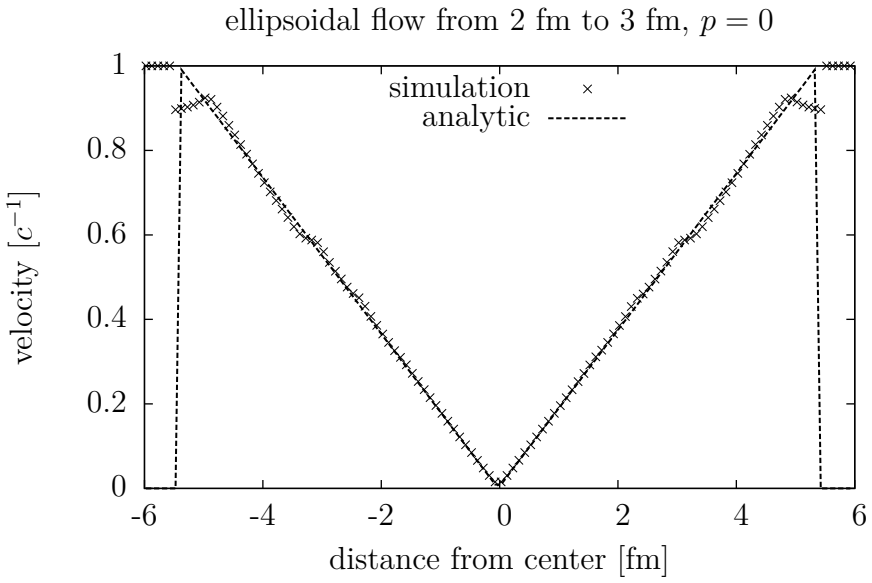


Figure 5.36: Ellipsoidal flow, velocity, MUSTA-FORCE with MINBEE limiter.

surface memory version. It is mostly due to reduced register pressure—kernels that run out of registers put some of the intermediate data in local memory (this situation is called *register spilling*). This has great effect on the performance, and as the profiling of the application has shown, in the shared memory version some register spilling indeed takes place, making the execution speed of both versions comparable.

Apart from lower register usage, the surface version also shows bigger occupancy (a statistic that tells to what extent in average the cores are being used), and a slightly smaller branching (a measure of divergence between threads running concurrently, which cause serialization).

5.3.1 Algorithm performance comparison

During the simulations durations of the kernel execution were measured. In table 5.4 the averages of kernel execution time for the ellipsoidal flow test are given. The simulations were performed on a 120^3 grid.

Due to it's simplicity, the finite difference WENO scheme, despite being higher order, was about 54% and 41% faster than MUSTA–FORCE for 5th and 7th order respectively. The MUSTA–FORCE simulation was performed with 4 MUSTA iterations, as in all the tests.

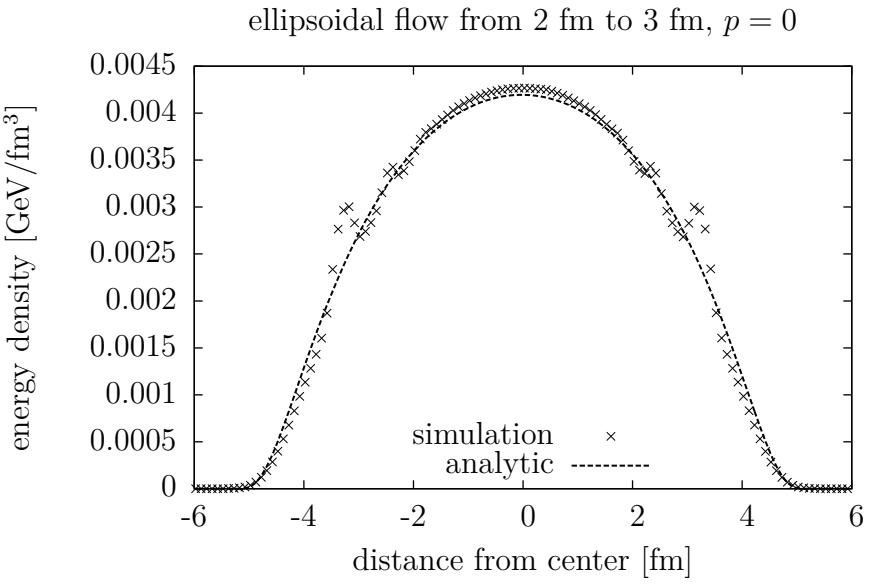


Figure 5.37: Ellipsoidal flow, energy density in local coordinates, MUSTA-FORCE with van Albada limiter.

MUSTA–FORCE	WENO-5	WENO-7
817	375	479

Table 5.4: Average time of kernel execution for different schemes, given in milliseconds

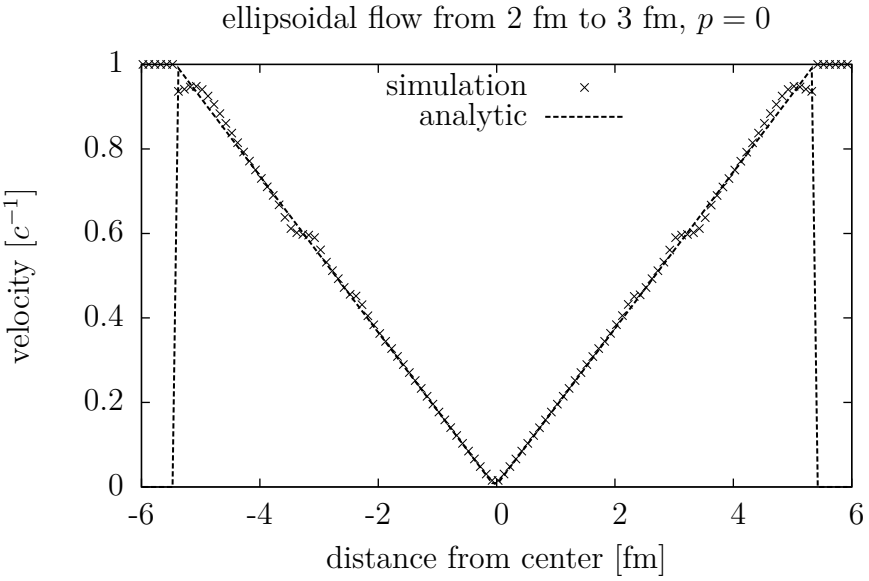


Figure 5.38: Ellipsoidal flow, velocity, MUSTA-FORCE with van Albada limiter.

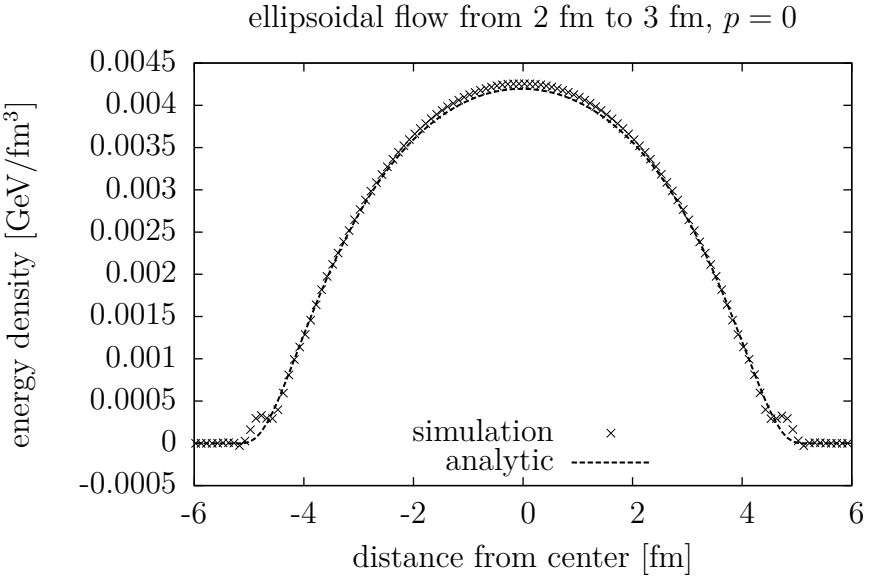


Figure 5.39: Ellipsoidal flow, energy density in local coordinates, 5th order WENO.

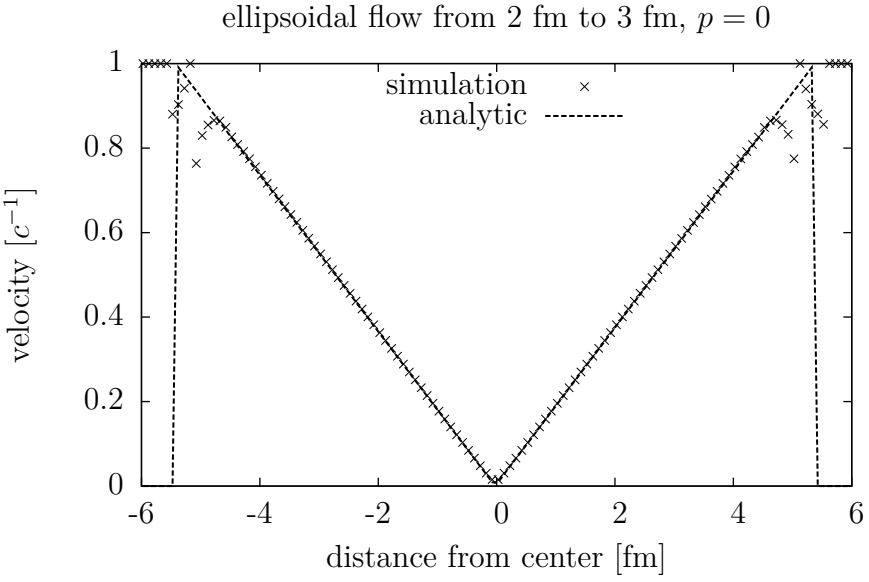


Figure 5.40: Ellipsoidal flow, velocity, 5th order WENO.

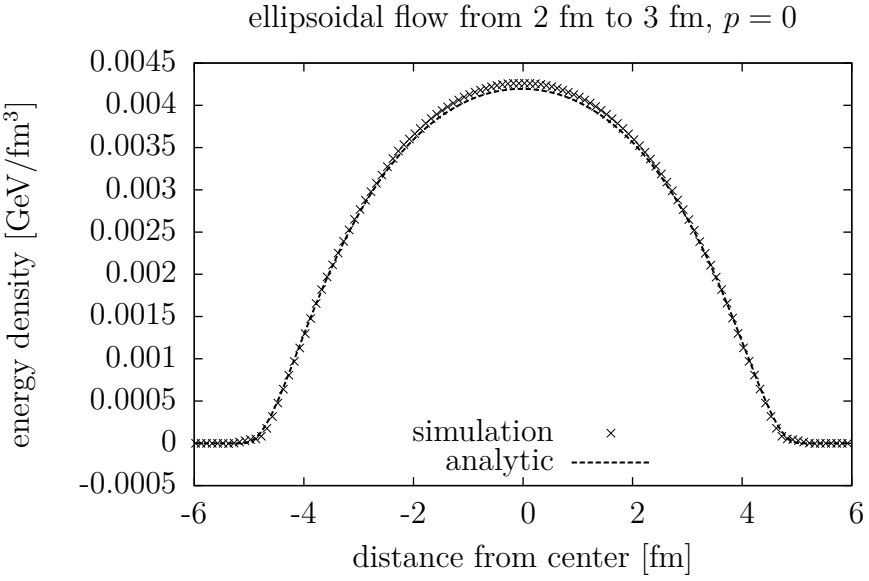


Figure 5.41: Ellipsoidal flow, energy density in local coordinates, 7th order WENO.

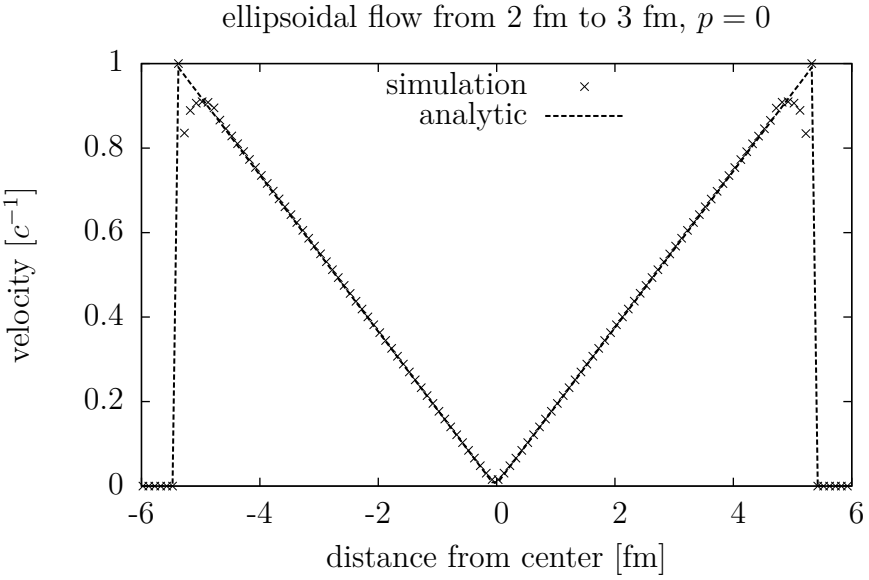


Figure 5.42: Ellipsoidal flow, velocity, 7th order WENO.

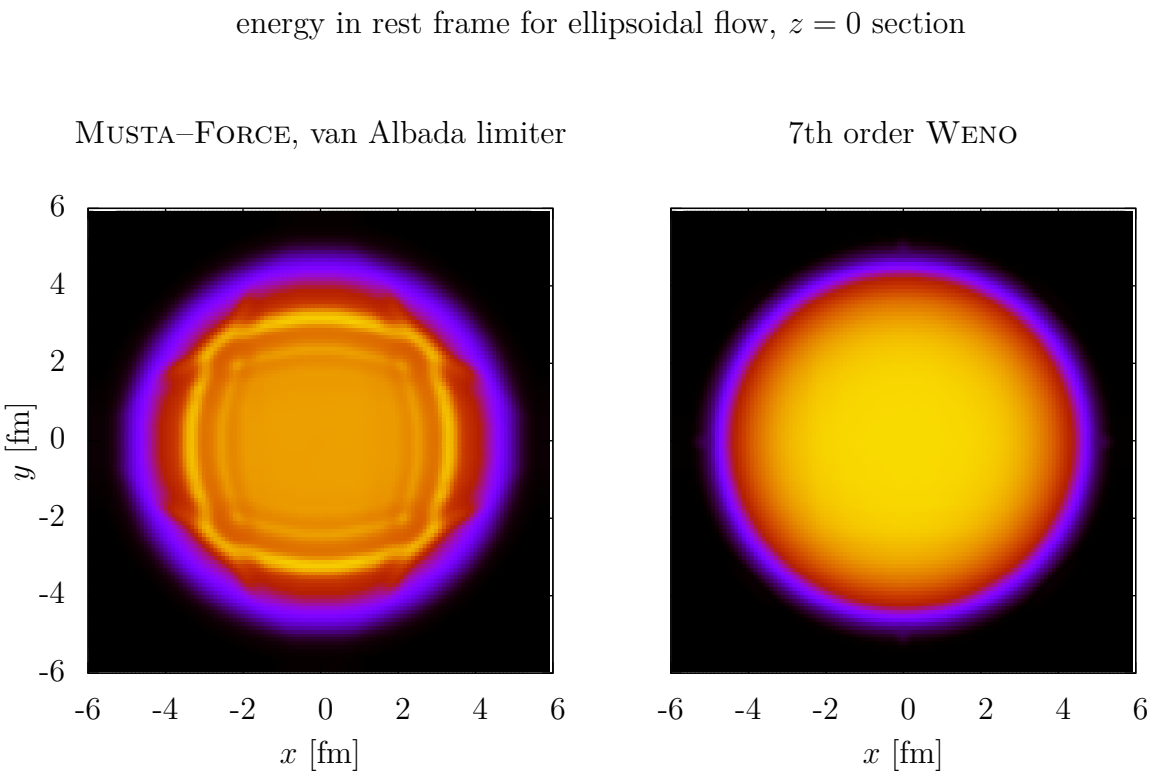


Figure 5.43: Ellipsoidal flow, energy density in local coordinates.

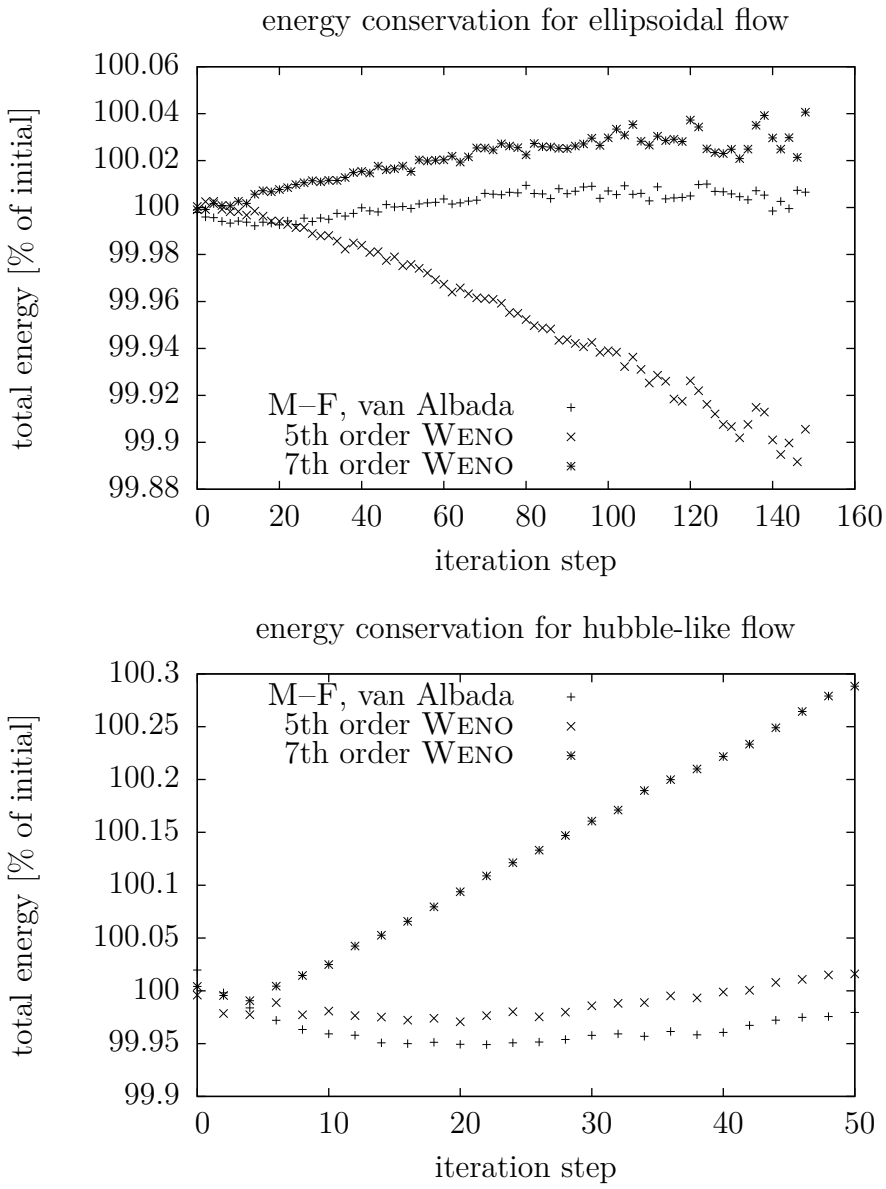


Figure 5.44: Conserservation of energy for MUSTA-FORCE and WENO schemes

Summary

As a result of this master's thesis a number of numerical algorithms dedicated to solving conservative field equations have been implemented and tested. Especially the WENO schemes proved to be quite efficient and robust. A variety of slope limiting methods have been compared with a MUSTA–FORCE algorithm.

The implementation is designed to run efficiently on contemporary graphics processing units, which have many times more computing power compared to ordinary processors. Benchmarks and comparison to an equivalent implementation of some of these algorithms in C shown speedups of over 2 orders of magnitude.

The code as is (without sources and viscosity), when coupled with software for other states of matter, could be used to simulate heavy ion collision events in energy scales where viscosity does not play a major role (e.g. RHIC energies). Thanks to it's speed, event-by-event analyses would benefit the most from it's use.

It could also be useful to study jet interactions with the medium, especially with the addition of sources (which should not pose a big problem and are expected to be implemented soon). Modern graphics cards have enough memory to conduct high resolution simulations, which are the ones with highest speedup.

Implementation of viscosity could strongly influence the efficiency—it would require several additional registers per lattice cell, and register spilling would likely occur. With some clever use of shared memory though one may be able to reduce the register spilling and maintain the speedup.

There are still a lot of things that could be done to extend this code. It would be interesting to combine both schemes, i.e. use WENO in a finite volume setting instead of MUSCL—perhaps with operator splitting for simplicity, and then use MUSTA-FORCE as the Riemann solver.

One great usability improvement would be to make the code more modular, and make it possible to construct the full algorithm from different schemes, perhaps even during run-time. Another important feature is the equation of state input. Currently only a parametrization is supported, but a table lookup support is also planned.

Appendix A

Source code

The whole software project is the result of a collaborative work of the hydro research group at Warsaw University of Technology. It is based on a previous work by Daniel Kikoła et al. [33]. The project is kept in a shared git repository, which can be used to quantify (extremely roughly!) author's amount of work on the code: added lines: 7745, removed lines: 3901, with the main version codebase totaling to just over 5000 lines of code.

Below is the function `weno7` attached, taken verbatim from the code. It returns the interpolated value between `fc` and `fn`. Thanks to class features that are available in CUDA code, all algorithmic code is just as clear and easy to read and write.

```
__device__ U weno7(U fppp, U fpp, U fp, U fc, U fn, U fnn, U fnnn)
{
    REAL g1, g2, g3, g4, eps = 1e-6;

    U f1, f2, f3, f4,
      b1, b2, b3, b4,
      w1, w2, w3, w4, w1a, w2a, w3a, w4a, ws;

    f1 = -0.25 * fppp
       + 13.0/12.0 * fpp
       - 23.0/12.0 * fp
       + 25.0/12.0 * fc;

    f2 = 1.0/12.0 * fpp
       - 5.0/12.0 * fp
       + 13.0/12.0 * fc
       + 0.25 * fn;

    f3 = -1.0/12.0 * fp
       + 7.0/12.0 * fc
       + 7.0/12.0 * fn
       - 1.0/12.0 * fnn;
```

```
    f4 = 0.25 * fc
       + 13.0/12.0 * fn
       - 5.0/12.0 * fnn
       + 1.0/12.0 * fnnn;

    g1 = 1.0/35.0;
    g2 = 12.0/35.0;
    g3 = 18.0/35.0;
    g4 = 4.0/35.0;

    b1 = fppp * (547*fppp - 3882*fpp + 4642*fp - 1854*fc)
       + fpp  * (7043*fpp - 17246*fp + 7042*fc)
       + fp   * (11003*fp - 9402*fc) + 2107*fc*fc;

    b2 = fpp * (267*fpp - 1642*fp + 1602*fc - 494*fn)
       + fp  * (2843*fp - 5966*fc + 1922*fn)
       + fc  * (3443*fc - 2522*fn)
       + 547*fn*fn;

    b3 = fp  * (547*fp - 2522*fc + 1922*fn - 494*fnn)
       + fc  * (3443*fc - 5966*fn + 1602*fnn)
       + fn  * (2843*fn - 1642*fnn) + 267*fnn*fnn;

    b4 = fc  * (2107*fc - 9402*fn + 7042*fnn - 1854*fnnn)
       + fn  * (11003*fn - 17246*fnn + 4642*fnnn)
       + fnn * (7043*fnn - 3882*fnnn)
       + 547*fnnn*fnnn;

    w1a = eps + b1;
    w1a = g1 / (w1a*w1a);

    w2a = eps + b2;
    w2a = g2 / (w2a*w2a);

    w3a = eps + b3;
    w3a = g3 / (w3a*w3a);

    w4a = eps + b4;
    w4a = g4 / (w4a*w4a);

    ws = w1a + w2a + w3a + w4a;

    w1 = w1a/ws;
    w2 = w2a/ws;
    w3 = w3a/ws;
    w4 = w4a/ws;

    return w1*f1 + w2*f2 + w3*f3 + w4*f4;
}
```

Appendix B

Jacobian

The Jacobian could be useful for implementing other time stepping schemes, or for a proper flux splitting and transformation to characteristic variables.

The total flux is given by:

$$\vec{F} = \begin{pmatrix} (E+p)\vec{v} \\ \vec{M}\vec{v} + p\mathbb{I} \\ R\vec{v} \end{pmatrix} \tag{B.1}$$

We can calculate the Jacobian of the total flux

$$\frac{\partial \vec{F}}{\partial(E, \vec{M}, R)} = \begin{pmatrix} \frac{\partial \vec{v}}{\partial E}(E+p) + \vec{v}\left(1 + \frac{\partial p}{\partial E}\right) & \frac{\partial \vec{v}}{\partial \vec{M}}(E+p) + \vec{v}\frac{\partial p}{\partial \vec{M}} & \frac{\partial \vec{v}}{\partial R}(E+p) + \vec{v}\frac{\partial p}{\partial R} \\ \vec{M}\frac{\partial \vec{v}}{\partial E} + \frac{\partial p}{\partial E}\mathbb{I} & \vec{M}\frac{\partial \vec{v}}{\partial \vec{M}} + \left(\vec{v} + \frac{\partial p}{\partial \vec{M}}\right)\mathbb{I} & \vec{M}\frac{\partial \vec{v}}{\partial R} + \frac{\partial p}{\partial R}\mathbb{I} \\ R\frac{\partial \vec{v}}{\partial E} & R\frac{\partial \vec{v}}{\partial \vec{M}} & R\frac{\partial \vec{v}}{\partial R} + \vec{v} \end{pmatrix} \tag{B.2}$$

To get derivatives of $\vec{v}$ and p, we can use the relation

$$v = \frac{M}{E + p\left(E - Mv, R\sqrt{1-v^2}\right)} \tag{B.3}$$

and differentiate it w.r.t. our variables

$$\frac{\partial v}{\partial E} = -\frac{M\left(\frac{\partial p}{\partial E} + 1\right)}{(E+p)^2} \tag{B.4}$$

for pressure use the chain rule

$$\frac{\partial p}{\partial E} = \frac{\partial p}{\partial e}\left(1 - M\frac{\partial v}{\partial E}\right) - \frac{\partial p}{\partial n}\frac{v\frac{\partial v}{\partial E}R}{\sqrt{1-v^2}} \tag{B.5}$$

Solving the system of equations B.4 and B.5, we get the derivative of velocity

$$\frac{\partial v}{\partial E} = \frac{\left(\frac{\partial p}{\partial e} + 1\right)v^2\sqrt{1-v^2}}{\frac{\partial p}{\partial n}v^3 R + \sqrt{1-v^2}M\left(\frac{\partial p}{\partial e}v^2 - 1\right)} \tag{B.6}$$

and pressure

$$\frac{\partial p}{\partial E} = -\frac{\frac{\partial p}{\partial n}v^3R + \sqrt{1-v^2}\frac{\partial p}{\partial e}(v^2+1)M}{\frac{\partial p}{\partial n}v^3R + \sqrt{1-v^2}\left(\frac{\partial p}{\partial e}v^2 - 1\right)M} \tag{B.7}$$

Similiar procedure can give us rest of the derivatives

$$\frac{\partial v}{\partial M_i} = -\frac{\sqrt{1-v^2}v_i\left(\frac{\partial p}{\partial e}v^2+1\right)}{\frac{\partial p}{\partial n}v^3R + \sqrt{1-v^2}\left(\frac{\partial p}{\partial e}v^2 - 1\right)M} \tag{B.8}$$

$$\frac{\partial p}{\partial M_i} = \frac{v_i\left(\frac{\partial p}{\partial n}vR + 2\frac{\partial p}{\partial e}\sqrt{1-v^2}M\right)}{\frac{\partial p}{\partial n}v^3R + \sqrt{1-v^2}\left(\frac{\partial p}{\partial e}v^2 - 1\right)M} \tag{B.9}$$

$$\frac{\partial v}{\partial R} = -\frac{\frac{\partial p}{\partial n}v^2(v^2-1)}{\frac{\partial p}{\partial n}v^3R + \sqrt{1-v^2}\left(\frac{\partial p}{\partial e}v^2 - 1\right)M} \tag{B.10}$$

$$\frac{\partial p}{\partial R} = \frac{\frac{\partial p}{\partial n}(v^2-1)M}{\frac{\partial p}{\partial n}v^3R + \sqrt{1-v^2}\left(\frac{\partial p}{\partial e}v^2 - 1\right)M} \tag{B.11}$$

In order to get derivatives of velocity components the following relation can be used

$$v_i = \frac{M_i}{M}v \tag{B.12}$$

Note that to compute the Jacobian, partial derivatives of pressure are needed. These could be either computed numerically, or given as input.

An explicit expression for the Jacobian is quite verbose and will be omitted. Below a Maxima worksheet that computes all the derivatives and the explicit Jacobian is attached.

```
(%i1)  load("lrats")$
       load("vect");

(%i3)  depends(v, [E,Mi,R]); depends(M, Mi);

(%i5)  e : E - M*v; n : R*sqrt(1-v^2);
```

E derivatives

```
(%i7)  vE_soln0 : [diff(v,E) = diff(M/(E+p(e, n)),E)];

(%i8)  pE_soln0 : [diff(p(e,n),E) = pe*diff(e,E) + pn*diff(n,E)];

(%i9)  vE_soln1 : ratsimp(solve(subst(pE_soln0, vE_soln0), diff(v,E)));
```

```
(%i10) vE_soln2 : ratsubst(M/v, p(e,n)+E, vE_soln1);

(%i11) pE_soln1 : combine(rat(subst(vE_soln2, pE_soln0)));
```

M derivatives

```
(%i12) vMi_soln0 : [diff(v,Mi) = diff(M/(E+p(e, n)),Mi)];

(%i13) pMi_soln0 : [diff(p(e,n),Mi) = pe * diff(e,Mi) + pn*diff(n,Mi)];

(%i14) MMi_soln : [diff(M,Mi) = Mi/M];

(%i15) vMi_soln1 : ratsimp(solve(subst(MMi_soln,subst(pMi_soln0, vMi_soln0)), diff(v,Mi)));

(%i16) vMi_soln2 : ratsubst(M/v, p(e,n)+E, vMi_soln1);

(%i17) pMi_soln1 : combine(rat(subst(MMi_soln, subst(vMi_soln2, pMi_soln0))));
```

R derivatives

```
(%i18) vR_soln0 : [diff(v,R) = diff(M/(E+p(e, n)),R)];

(%i19) pR_soln0 : [diff(p(e,n),R) = pe * diff(e,R) + pn*diff(n,R)];

(%i20) vR_soln1 : ratsubst(M/v, p(e,n)+E,solve(subst(pR_soln0, vR_soln0),diff(v,R)));

(%i21) pR_soln1 : ratsimp(factor(subst(vR_soln1, pR_soln0)));
```

Jacobian

```
(%i22) depends(v,[E,Mx,My,Mz,R]); depends(M, [Mx, My, Mz]);

(%i24) F0 : [(E+p(e,n))*vx, Mx*vx+p(e,n), My*vx, Mz*vx, R*vx];
       G0 : [(E+p(e,n))*vy, Mx*vy, My*vy+p(e,n), Mz*vy, R*vy];
       H0 : [(E+p(e,n))*vz, Mx*vz, My*vz, Mz*vz+p(e,n), R*vz];

(%i27) vi_solns : [vx = Mx/M*v, vy = My/M*v, vz = Mz/M*v];

(%i28) F1 : subst(vi_solns, F0);
       G1 : subst(vi_solns, F0);
       H1 : subst(vi_solns, F0);
```

```
(%i31) U : [E, Mx, My, Mz, R];

(%i32) subs : append( pR_soln1, vR_soln1, pE_soln1, vE_soln2,
       subst([Mi=Mx], pMi_soln1),subst([Mi=My], pMi_soln1),subst([Mi=Mz], pMi_soln1),
       subst([Mi=Mx], vMi_soln2),subst([Mi=My], vMi_soln2),subst([Mi=Mz], vMi_soln2),
       subst([Mi=Mx], MMi_soln), subst([Mi=My], MMi_soln),subst([Mi=Mz], MMi_soln));

(%i37) J : combine(rat(subst(subs, [jacobian(F1, U), jacobian(G1, U), jacobian(H1, U)])));
```

List of Figures

Bibliography

[1] E. Abers and B. Lee. Gauge theories. *Physics Reports*, 9:1–2, November 1973.

[2] B. Alver, M. Baker, C. Loizides, and P. Steinberg. The PHOBOS Glauber Monte Carlo. 2008.

[3] D. Balsara and C. W. Shu. Monotonicity Preserving Weighted Essentially Non-oscillatory Schemes with Increasingly High Order of Accuracy. *Journal of Computational Physics*, 160(2):405–452, May 2000.

[4] S.A. Bass, M. Belkacem, M. Bleicher, M. Brandstetter, L. Bravina, et al. Microscopic models for ultrarelativistic heavy ion collisions. *Prog.Part.Nucl.Phys.*, 41:255–369, 1998.

[5] Viktor V. Begun, Mark I. Gorenstein, and Oleg A. Mogilevsky. Equation of State for the Quark Gluon Plasma with the Negative Bag Constant. *Ukr.J.Phys.*, 55:1049–1052, 2010.

[6] Marsha Berger, Scott M. Murman, and Michael J. Aftosmis. Analysis of slope limiters on irregular grids.

[7] Jay P Boris and David L Book. Flux-corrected transport. i. shasta, a fluid transport algorithm that works. *Journal of Computational Physics*, 11(1):38 – 69, 1973.

[8] Wojciech Broniowski, Maciej Rybczynski, and Piotr Bozek. GLISSANDO: Glauber initial-state simulation and more.. *Comput.Phys.Commun.*, 180:69–83, 2009.

[9] Rupa Chatterjee, Hannu Holopainen, Ilkka Helenius, Thorsten Renk, and Kari J. Eskola. Elliptic flow of thermal photons from event-by-event hydrodynamic model. 2013.

[10] Emil Constantinescu and Adrian Sandu. Explicit time stepping methods with high stage order and monotonicity properties. In *Proceedings of the 9th International Conference on Computational Science*, ICCS 2009, pages 293–301, Berlin, Heidelberg, 2009. Springer-Verlag.

[11] Thomas Scott Crow, Dr. Frederick, and C. Harris. Evolution of the graphical processing unit, 2004.

[12] Sebastian Cygert, Daniel Kikoła, Joanna Porter-Sobieraj, Jan Sikorski, and Marcin Słodkowski. Towards an efficient multi-stage riemann solver for nuclear physics simulations. In *Proceedings of the Federated Conference On Computer Science And Information Systems*, FEDCSIS 2013. Springer LNCS, 2013.

[13] Sebastian Cygert, Daniel Kikoła, Joanna Porter-Sobieraj, Jan Sikorski, and Marcin Słodkowski. Using gpus for parallel stencil computations in relativistic hydrodynamic simulation. In *Proceedings of the 10th International Conference on Parallel Processing and Applied Mathematics*, PPAM 2013. Springer LNCS, 2013.

[14] H.J. Drescher, S. Ostapchenko, T. Pierog, and K. Werner. Initial condition for QGP evolution from NEXUS. *Phys.Rev.*, C65:054902, 2002.

[15] K J Eskola, H Niemi, and P V Ruuskanen. Elliptic flow from pqcd + saturation + hydro model. *J. Phys. G*, 35(arXiv:0705.2114):054001–52. 2 p, May 2007.

[16] R.J. Fries, M. He, and R. Rapp. Quark Recombination and Heavy Quark Diffusion in Hot Nuclear Matter. *J.Phys.*, G38:124068, 2011.

[17] Charles Gale, Sangyong Jeon, and Bjoern Schenke. Hydrodynamic modeling of heavy-ion collisions. Technical Report arXiv:1301.5893, Jan 2013. Comments: 27 pages, 7 figures, invited review for a special issue of International Journal of Modern Physics A.

[18] S. K. Godunov. A difference method for numerical calculation of discontinuous solutions of the equations of hydrodynamics.

[19] E. Gourgoulhon and E. Gourgoulhon. An introduction to relativistic hydrodynamics. *EAS Publications Series*, 21:43–79, 0 2006.

[20] F. Grassi. Particle emission in hydrodynamics: a problem needing a solution. *Brazilian Journal of Physics*, 35:52 – 59, 03 2005.

[21] NVIDIA CUDA Programming Guide. `http://nvidia.com`.

[22] M Gyulassy, D H Rischke, and B Zhang. Hot spots and turbulent initial conditions of quark-gluon plasmas in nuclear collisions. *Nucl. Phys. A*, 613(nucl-th/9609030. CU-TP-707):397–434. 28 p, Sep 1996.

[23] Johannes Habich, Christian Feichtinger, Harald Köstler, Georg Hager, and Gerhard Wellein. Performance engineering for the lattice boltzmann method on gpgpus: Architectural requirements and performance results. *CoRR*, abs/1112.0850, 2011.

[24] Yogiro Hama, R.P.G. Andrade, F. Grassi, W.-L. Qian, and T. Kodama. Fluctuation of the Initial Conditions and Its Consequences on Some Observables. *Acta Phys.Polon.*, B40:931–936, 2009.

[25] Mark J. Harris, Greg Coombe, Thorsten Scheuermann, and Anselmo Lastra. Physically-based visual simulation on graphics hardware. In *HWWS '02: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 109–118, Aire-la-Ville, Switzerland, Switzerland, 2002. Eurographics Association.

[26] A. Harten and S. Osher. Uniformly High-Order Accurate Nonoscillatory Schemes. I. *SIAM Journal on Numerical Analysis*, 24:279–309, April 1987.

[27] Tetsufumi Hirano. Elliptic flow based on a relativistic hydrodynamic model. 1999.

[28] H Holopainen, H Niemi, and K J Eskola. Elliptic flow from event-by-event hydrodynamics. *Journal of Physics G: Nuclear and Particle Physics*, 38(12):124164, 2011.

[29] Hannu Holopainen and Pasi Huovinen. Dynamical freeze-out in event-by-event hydrodynamics. *Journal of Physics: Conference Series*, 389(1):012018, 2012.

[30] Thomas Y. Hou and Philippe G. Le Floch. Why nonconservative schemes converge to wrong solutions: error analysis. Technical Report RI 255, Ecole polytechnique. Palaiseau, 1992.

[31] P Huovinen and P V Ruuskanen. Hydrodynamic models for heavy ion collisions. Technical Report nucl-th/0605008, May 2006.

[32] Xun Jia, Yifei Lou, Ruijiang Li, William Y. Song, and Steve B. Jiang. Gpu-based fast cone beam ct reconstruction from undersampled and noisy projection data via total variation. *MEDICAL PHYSICS*, 37:1757, 2010.

[33] Daniel Kikola, Wiktor Peryt, Yuri M. Sinyukov, Marcin Slodkowski, and Marek Szuba. The New computer program for three dimensional relativistic hydrodynamical model. *Acta Phys.Hung.*, A27:373–378, 2006.

[34] Adam Kisiel, Tomasz Taluc, Wojciech Broniowski, and Wojciech Florkowski. THERMINATOR: THERMal heavy-IoN generATOR. *Comput.Phys.Commun.*, 174:669–687, 2006.

[35] V.K. Magas, L.P. Csernai, and D. Strottman. Effective string rope model for the initial stages of ultra-relativistic heavy ion collisions. *Nuclear Physics A*, 712(1–2):167 – 204, 2002.

[36] NVIDIA CUDA Debugger User Manual. `http://nvidia.com`. (accessed 10 August 2013).

[37] José Maria Martí and Ewald Müller. Numerical hydrodynamics in special relativity. *Living Reviews in Relativity*, 6(7), 2003.

[38] Michael L. Miller, Klaus Reygers, Stephen J. Sanders, and Peter Steinberg. Glauber modeling in high energy nuclear collisions. *Ann.Rev.Nucl.Part.Sci.*, 57:205–243, 2007.

[39] John Nickolls and William J. Dally. The gpu computing era. *IEEE Micro*, 30:56–69, 2010.

[40] Chiho Nonaka and Masayuki Asakawa. Modeling a realistic dynamical model for high energy heavy ion collisions. 2012.

[41] Chiho Nonaka, Masayuki Asakawa, and Steffen A. Bass. The 3D hydro + UrQMD model with the QCD critical point. *J.Phys.*, G35:104099, 2008.

[42] S.F. Novaes. Standard model: An Introduction. 1999.

[43] Gilles Pagès and Benedikt Wilbertz. Gpgpus in computational finance: Massive parallel computing for american style options. *ArXiv eprints*, 2011.

[44] James C. Phillips and John E. Stone. Probing biomolecular machines with graphics processors. *Commun. ACM*, 52:34–41, October 2009.

[45] T. Pierog, Iu. Karpenko, S. Porteboeuf, and K. Werner. New Developments of EPOS 2. 2010.

[46] M. Plümer, S. Raha, and R. M. Weiner. Effect of confinement on the sound velocity in a quark-gluon plasma. *Physics Letters B*, 139:198–202, May 1984.

[47] Jianxian Qiu and Chi-Wang Shu. Finite difference weno schemes with lax–wendroff-type time discretizations. *SIAM J. Sci. Comput.*, 24(6):2185–2198, June 2002.

[48] Dirk H. Rischke. Fluid dynamics for relativistic nuclear collisions. In *in Hadrons in Dense Matter and Hadrosynthesis, edited by J. Cleymans et al., Springer Lecture Notes in Physics 516*, pages 21–9809044. Springer Verlag, 1999.

[49] D.P. Roy. Basic constituents of matter and their interactions: A Progress report. 1999.

[50] Guang shan Jiang, Chi-Wang Shu, and In L. Efficient implementation of weighted eno schemes. *J. Comput. Phys*, 126:202–228, 1995.

[51] Bijan Sheikholeslami-Sabzevari. Equation of state for hot quark-gluon plasma transitions to hadrons with full qcd potential. *Phys. Rev. C*, 65:054904, Apr 2002.

[52] Yiqing Shen and Gecheng Zha. chapter A Robust Seventh-Order WENO Scheme and ITS Applications. Aerospace Sciences Meetings. American Institute of Aeronautics and Astronautics, Jan 2008. 0.

[53] Chi-Wang Shu. High order finite difference and finite volume weno schemes and discontinuous galerkin methods for cfd. *International Journal of Computational Fluid Dynamics*, 17:107–118, 2001.

[54] Yu. M. Sinyukov and Iu. A. Karpenko. Quasi-inertial ellipsoidal flows in relativistic hydrodynamics. 2005.

[55] Reinhard Stock. Relativistic Nucleus-Nucleus Collisions and the QCD Matter Phase Diagram. 2008.

[56] P. K. Sweby. High Resolution Schemes Using Flux Limiters for Hyperbolic Conservation Laws. *SIAM Journal on Numerical Analysis*, 21:995–1011, October 1984.

[57] V. A. Titarev and E. F. Toro. Musta schemes for multi-dimensional hyperbolic systems: analysis and improvements. *International Journal for Numerical Methods in Fluids*, 49(2):117–147, 2005.

[58] V.A. Titarev and E.F. Toro. Finite-volume WENO schemes for three-dimensional conservation laws. *J. Comput. Phys.*, 201(1):238–260, 2004.

[59] E. F. Toro. *Riemann Solvers and Numerical Methods for Fluid Dynamics: A Practical Introduction*. Springer, Berlin, 2nd edition, 1999.

[60] E. F. Toro. Musta: a multi-stage numerical flux. *Appl. Numer. Math.*, 56(10):1464–1479, October 2006.

[61] Giorgio Torrieri, Barbara Betz, Jorge Noronha, and Miklos Gyulassy. Mach cones in heavy ion collisions. 2009.

[62] Christian R Trott, Lars Winterfeld, and Paul S Crozier. General-purpose molecular dynamics simulations on gpu-based clusters. *ArXiv*, 1009(4330):12, 2010.

[63] K. Urmossy and T.S. Biro. Cooper-Frye Formula and Non-extensive Coalescence at RHIC Energy. *Phys.Lett.*, B689:14–17, 2010.

[64] NVIDIA Website. `http://nvidia.com`.

[65] NVIDIA Website. `http://developer.nvidia.com/what-cuda`. (accessed 10 August 2013).

[66] Martin Weigel and Taras Yavors'kii. Gpu accelerated monte carlo simulations of lattice spin models. Technical Report arXiv:1107.5463, Jul 2011. Comments: 5 pages, 4 figures, 1 table; Physics Procedia 15, 92 (2011).

[67] Wikipedia. Comparison of nvidia graphics processing units. `http://en.wikipedia.org/wiki/Comparison_of_Nvidia_graphics_processing_units`. (accessed 10 August 2013).

[68] Wikipedia. Cuda. `http://en.wikipedia.org/wiki/CUDA`. (accessed 10 August 2013).

[69] Yulong Xing and Chi-Wang Shu. High order well-balanced finite volume weno schemes and discontinuous galerkin methods for a class of hyperbolic systems with source terms. *J. Comput. Phys.*, 214(2):567–598, May 2006.

[70] W. A. Zajc. The Fluid Nature of Quark-Gluon Plasma. *Nuclear Physics A*, 805:283–294, June 2008.