EPJ manuscript No.
(will be inserted by the editor)

Accelerated GPU Simulation of Compressible Flow by the Discontinuous Evolution Galerkin Method

Block B. J.[1,a], Lukáčová-Medvid'ová M.[2,b], Virnau P.[1,c], and Yelash L.[2,d]

[1] Institute of Physics, University of Mainz, Staudingerweg 7, 55099 Mainz
[2] Institute of Mathematics, University of Mainz, Staudingerweg 9, 55099 Mainz

Abstract. The aim of the present paper is to report on our recent results for GPU accelerated simulations of compressible flows. For numerical simulation the adaptive discontinuous Galerkin method with the multidimensional bicharacteristic based evolution Galerkin operator has been used. For time discretization we have applied the explicit third order Runge-Kutta method. Evaluation of the genuinely multidimensional evolution operator has been accelerated using the GPU implementation. We have obtained a speedup up to 30 (in comparison to a single CPU core) for the calculation of the evolution Galerkin operator on a typical discretization mesh consisting of 16384 mesh cells.

1 Introduction

A characteristic feature of many geophysical flows is their multidimensional character with different localized structural phenomena such as cloud environment interface. In order to approximate these local structures efficiently, mesh adaptivity is a necessary tool that has to be used in computer simulations.

In [1] we have proposed new large time step finite volume evolution Galerkin (FVEG) methods for geophysical flows. They combine the simplicity of the finite volume methods with the theory of bicharacteristics yielding a genuinely multidimensional finite volume scheme. The evolution Galerkin operator used in order to evaluate fluxes over the cell interfaces can be interpreted as a multidimensional approximate numerical flux function [2,3]. Numerical simulations confirm high efficiency and good multidimensional resolution of the FVEG scheme. However, the FVEG schemes proposed in [1] have been considered only on regular rectangular meshes. The aim of this paper is to generalize genuinely multidimensional evolution Galerkin schemes for adaptive irregular meshes in order to allow more efficient simulations of various localized flow structures. Another important requirement for practical applications of computer simulations are low computational times. Hence, the next goal of this paper

[a] e-mail: blockb@uni-mainz.de
[b] e-mail: lukacova@mathematik.uni-mainz.de
[c] e-mail: virnau@uni-mainz.de
[d] e-mail: yelash@uni-mainz.de

is to report on the acceleration of the EG methods by means of implementation on a graphics processing unit (GPU).

Since the introduction of CUDA by NVidia in 2007, many scientific algorithms have been successfully ported to GPUs. Examples include the simple Ising model from statistical physics [4–6], molecular dynamics simulations [7–11] and the analysis of financial market data [12] to name just a few. Application of the GPUs for multidimensional flow problems using modern higher order numerical schemes has just been recently started, see, e.g., [13,14] for the GPU simulations of the shallow water equations, [15] for the Euler equations and [16,17] for the Navier-Stokes equations. In this paper we port the most time consuming part of our Computational Fluid Dynamics code, the EG operator, to the GPU while the rest of the code is still being executed on the CPU. By doing so, we are able to speed up the multidimensional evolution operator by a factor of 30 (GTX580 vs single core Nehalem i7 2.67Ghz), resulting in a roughly sixfold speedup of the overall code.

2 Mathematical model

We start with the description of mathematical model. Motion of compressible flows is governed by the Euler equations

$$\begin{aligned} \partial_t \rho + \nabla \cdot (\rho \mathbf{u}) &= 0 \\ \partial_t (\rho \mathbf{u}) + \nabla \cdot (\rho \mathbf{u} \otimes \mathbf{u} + p\,\mathrm{Id}) &= -\rho g \mathbf{k} \\ \partial_t (\rho\theta) + \nabla \cdot (\rho\theta \mathbf{u}) &= 0\,, \end{aligned} \tag{1}$$

where ρ denotes the density, $\mathbf{u} = (u, v)^{\mathrm{T}}$ is a two-component velocity vector, p pressure and θ the potential temperature. Further, g stays for the gravitational constant, Id is the identity matrix in $\mathbb{R}^2$ and $\mathbf{k}$ the unit vector in the vertical direction. In order to close the system we determine pressure from the equation of state

$$p = p_0 \left(\frac{R\rho\theta}{p_0} \right)^{\gamma},$$

where $\gamma = c_p/c_v$ is the adiabatic constant, c_p and c_v the isobaric and isochoric specific heat constants, and $p_0 = 10^5 Pa$ the reference pressure. Denoting by T temperature of air at pressure p, the potential temperature θ can be obtained from the equation of adiabatic process in an ideal gas:

$$\theta = T \left(\frac{p_0}{p} \right)^{R/c_p}, \; R = c_p - c_v.$$

In many geophysical applications, flows can be considered as a perturbation of some reference equilibrium state. For example, atmospheric flows are typically represented as a perturbation over the background hydrostatic state $(\bar{\rho}, \bar{\mathbf{u}}(=0), \bar{p}, \bar{\theta})$, cf., e.g., [18,19],

$$\frac{\partial \bar{p}}{\partial y} = -\bar{\rho} g.$$

Here we assume $\bar{\theta} = 300\mathrm{K}$ and $\bar{\rho} = \frac{p_0}{R\bar{\theta}} \bar{\pi}^{\frac{c_v}{R}}, \bar{p} \equiv p(\bar{\rho}, \bar{\theta}) = p_0 \left(\frac{R\bar{\rho}\bar{\theta}}{p_0} \right)^{\gamma}$ with the Exner pressure $\bar{\pi}(x, y) := 1 - gy/\left(c_p \bar{\theta}\right)$.

In order to avoid numerical instabilities due to the multiscale behaviour of (1), numerical simulations are typically realized for perturbations $\rho' = \rho - \bar{\rho}$, $\theta' = \theta - \bar{\theta}$, $p' = p - \bar{p}$. The latter satisfy the following equation

$$\begin{aligned} \partial_t \rho' + \nabla \cdot (\rho \mathbf{u}) &= 0 \\ \partial_t (\rho \mathbf{u}) + \nabla \cdot (\rho \mathbf{u} \otimes \mathbf{u} + p' \,\mathrm{Id}) &= -\rho' g \mathbf{k} \\ \partial_t (\rho \theta') + \nabla \cdot (\rho \theta \mathbf{u}) &= 0. \end{aligned} \tag{2}$$

Our aim in what follows is to approximate (2) with the discontinuous Galerkin method. However, instead of the classical, one-dimensional numerical flux function (e.g., the Rusanov flux), we will apply a genuinely multidimensional evolution operator. To this extent let us rewrite (2) in the form of hyperbolic balance law for the vector variable $\mathbf{q} = (\rho, \rho u, \rho v, \rho\theta)^{\mathrm{T}}$

$$\frac{\partial \mathbf{q}}{\partial t} + \nabla \cdot \mathbf{F}(\mathbf{q}) = \mathbf{S}(\mathbf{q}), \tag{3}$$

where

$$\mathbf{F}(\mathbf{q}) = \begin{pmatrix} \rho \mathbf{u} \\ \rho \mathbf{u} \otimes \mathbf{u} + p' \,\mathrm{Id} \\ \rho \theta \mathbf{u} \end{pmatrix}, \qquad \mathbf{S}(\mathbf{q}) = \begin{pmatrix} 0 \\ -\rho' g \mathbf{k} \\ 0 \end{pmatrix}$$

is the nonlinear flux function and the source term, respectively. We should note that in our numerical experiments we will also use a stabilization through the artificial viscosity [19,20], which results in the following source term

$$\mathbf{S}(\mathbf{q}) = \begin{pmatrix} 0 \\ -\rho' g \mathbf{k} + \nabla \cdot (\mu \rho \nabla \mathbf{u}) \\ \nabla \cdot (\mu \rho \nabla \theta') \end{pmatrix}, \quad \mu > 0 \text{ is an artificial viscosity parameter.}$$

In the following, Eq. (3) will be approximated in space by the discontinuous Galerkin method and in time by the explicit Runge-Kutta scheme. The paper is organized as follows: The genuinely multidimensional evolution operator used for flux integration along cell interfaces is described in Section 3. In our numerical experiments discussed in Section 4, we benchmark the new GPU accelerated code on regular and adaptive grids, using quadratic polynomials for spatial approximation and explicit third order Runge-Kutta scheme for time integration. In Section 5 we will discuss implementation of the evolution operator on the GPU. In our forthcoming paper [21] we will compare different time integration schemes, in particular we will also use semi-implicit schemes in order to overcome the strong stability condition for time steps given by the Courant-Friedrichs-Lewy (CFL) number $\mathrm{CFL} = \frac{u \Delta t}{\Delta x}$.

3 Discontinuous Galerkin method and the multidimensional EG operator

In this section we follow [19,22,23] and derive the strong formulation of (3). Let us divide the computational domain Ω into a finite number of mesh cells Ω_e with a boundary $\partial\Omega_e$. In our numerical experiments we work with triangular mesh elements Ω_e and use the nodal basis functions $\{\psi_j, j = 1, \ldots, N\}$, N is a number of degrees

of freedom. Now, multiplying (3) with a nodal basis $\psi_i(\mathbf{x})$, integrating over Ω_e and applying twice integration by parts we obtain the strong formulation

$$\int_{\Omega_e} \left(\frac{\partial \mathbf{q}_h}{\partial t} + \nabla \cdot \mathbf{F}(\mathbf{q}_h) - \mathbf{S}(\mathbf{q}_h) \right) \psi_i(\mathbf{x}) \mathrm{d}\mathbf{x} \; = \\ \int_{\partial \Omega_e} \left(\mathbf{F}(\mathbf{q}_h) - \mathbf{F}(\mathbf{q}^*) \right) \psi_i(\mathbf{x}) \mathrm{d}S, \quad i = 1, \ldots, N.$$

Here $\mathbf{q}_h$ denotes a numerical solution $\mathbf{q}_h(\mathbf{x}) := \sum_j^N \mathbf{q}_j \psi_j(\mathbf{x})$ and $\mathbf{q}^* := EG\mathbf{q}_h$ a cell interface value predicted by the multidimensional evolution Galerkin EG operator. As in [24] Lagrange polynomials are used for the basis functions ψ_j with the Fekete points for the interpolation and Gauss points for the integrations. In the previous simulations using the discontinuous Galerkin method for atmospheric flows one-dimensional numerical flux functions, such as the Rusanov flux have been used, cf. e.g., [19, 23, 24] and the references therein. The novelty of our work relies on the application of a multidimensional evolution operator in order to compute $\mathbf{q}^*$.

3.1 Multidimensional EG operator

In this subsection we will give a brief description of the approximate evolution operator that is based on the theory of bicharacteristics for multidimensional hyperbolic conservation laws. More detailed derivation will be presented in our forthcoming paper [21]. First, let us rewrite (2) in a quasilinear form using the primitive variables $\mathbf{w} = (\rho', u, v, p')$

$$\partial_t \mathbf{w} + \underline{A}_1(\mathbf{w}) \, \partial_x \mathbf{w} + \underline{A}_2(\mathbf{w}) \, \partial_y \mathbf{w} = \mathbf{s}(\mathbf{w}) \tag{4}$$

with

$$\underline{A}_1 = \begin{pmatrix} u & \rho & 0 & 0 \\ 0 & u & 0 & \frac{1}{\rho} \\ 0 & 0 & u & 0 \\ 0 & \gamma p & 0 & u \end{pmatrix}, \; \underline{A}_2 = \begin{pmatrix} v & 0 & \rho & 0 \\ 0 & v & 0 & 0 \\ 0 & 0 & v & \frac{1}{\rho} \\ 0 & 0 & \gamma p & v \end{pmatrix}, \; \mathbf{s} = - \begin{pmatrix} \partial_y \bar{\rho} \, v \\ 0 \\ \frac{\rho'}{\rho} g \\ \partial_y \bar{p} \, v \end{pmatrix}. \tag{5}$$

Using the above thermodynamic relationship for $\bar{\rho}, \bar{p}$ we obtain

$$\partial_y \bar{\rho} = - \frac{p_0 \, c_v \, g}{(R\bar{\theta})^2 c_p} \left(1 - \frac{g\bar{y}}{c_p \, \bar{\theta}} \right)^{\frac{c_v}{R} - 1}, \quad \partial_y \bar{p} = - \frac{g \, p_0}{R \, \bar{\theta}} \left(1 - \frac{g\bar{y}}{c_p \, \bar{\theta}} \right)^{\frac{c_v}{R}}.$$

We first linearize (4) by freezing the Jacobian matrices $\underline{A}_1, \underline{A}_2$ at a suitable intermediate state $\tilde{\rho'}$, $\tilde{u}$, $\tilde{v}$, $\tilde{p'}$. Since our problem is hyperbolic we have real eigenvalues and a full set of linearly independent eigenvectors corresponding to the matrix pencil $\underline{P} := \underline{A}_1 n_x + \underline{A}_2 n_y$, where $\|(n_x, n_y)\| = 1$. Indeed, the eigenvalues of $\underline{P}$ are

$$\lambda_1 = \tilde{u} \, n_x + \tilde{v} \, n_y - a, \quad \lambda_2 = \lambda_3 = \tilde{u} \, n_x + \tilde{v} \, n_y, \quad \lambda_4 = \tilde{u} \, n_x + \tilde{v} \, n_y + a,$$

where $a := \sqrt{\gamma \frac{\tilde{p}}{\tilde{\rho}}} = \sqrt{\gamma R \theta \left(\frac{\rho R \theta}{p0} \right)^{\frac{R}{c_v}}}$ is a sonic speed. Now multiplying (4) by a matrix $\underline{R}^{-1}$, $\underline{R}$ consists of the right eigenvectors of $\underline{P}$, we can rewrite (4) using the so-called characteristic variables $\mathbf{v} = \underline{R}^{-1} \mathbf{w}$

$$\partial_t \mathbf{v} + \underline{B}_1 \partial_x \mathbf{v} + \underline{B}_2 \partial_y \mathbf{v} = \mathbf{r} \, ,$$

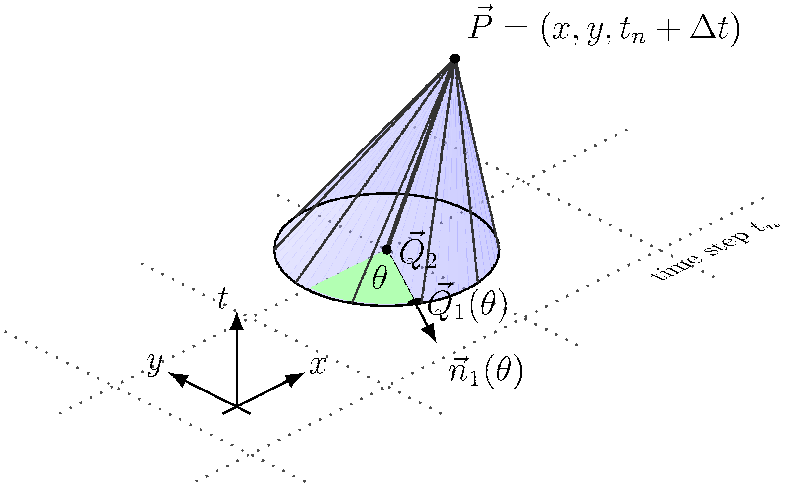

Fig. 1. Bicharacteristic cone used for the *EG* evolution operator

where $\mathbf{r} := \underline{R}^{-1}\mathbf{s}(\mathbf{w})$. Equivalently, we have

$$\partial_t \mathbf{v} + \mathrm{diag}(\underline{B}_1)\partial_x \mathbf{v} + \mathrm{diag}(\underline{B}_2)\partial_y \mathbf{v} = \mathbf{S} + \mathbf{r} \tag{6}$$

with

$$\mathbf{S}(\mathbf{x}, \theta) := -\left[(\underline{B}_1 - \mathrm{diag}(\underline{B}_1))\partial_x \mathbf{v} + (\underline{B}_2 - \mathrm{diag}(\underline{B}_2))\partial_y \mathbf{v}\right].$$

Integrating each equation of (6) along the corresponding bicharacteristic

$$\frac{d\mathbf{x}_j}{dt} := [\underline{B}_{1,jj}, \underline{B}_{2,jj}]^{\mathsf{T}},\ j = 1, \ldots, 4,$$

we obtain after some lengthy manipulations [21] the following exact integral representation

$$\begin{aligned}
\rho'(\mathbf{P}) = {} & \frac{\tilde{\rho}}{2\pi a}\int_0^{2\pi}\left[-\cos(\theta)\,u(\mathbf{Q}_1(\theta)) - \sin(\theta)\,v(\mathbf{Q}_1(\theta)) + \frac{1}{\tilde{\rho}\,a}p'(\mathbf{Q}_1(\theta))\right]d\theta \\
& + \rho'(\mathbf{Q}_2) - \frac{p'(\mathbf{Q}_2)}{a^2} \\
& - \frac{\tilde{\rho}}{2\pi a}\int_0^{2\pi}\int_{t_n}^{t_n+\Delta t}\beta(t,\theta)\,dt\,d\theta \\
& - \frac{\tilde{\rho}}{2\pi a}\int_0^{2\pi}\int_{t_n}^{t_n+\Delta t} -\sin(\theta) g\frac{\rho'}{\rho}(\mathbf{x}_1(t,\theta)) + \frac{v(\mathbf{x}_1(t,\theta))}{\tilde{\rho}\,a}\partial_y\bar{p}\,dt\,d\theta \\
& + \int_{t_n}^{t_n+\Delta t} v(\mathbf{x}_2(t))\left(-\partial_y\bar{\rho} + \frac{\partial_y\bar{p}}{a^2}\right)dt
\end{aligned} \tag{7}$$

$$\begin{aligned}
u(\mathbf{P}) = {} & \frac{1}{2\pi}\int_0^{2\pi}\left[-\frac{p'(\mathbf{Q}_1(\theta))}{\tilde{\rho}a}\cos(\theta) + u(\mathbf{Q}_1(\theta))\cos^2(\theta) + v(\mathbf{Q}_1(\theta))\sin(\theta)\cos(\theta)\right]d\theta \\
& + \frac{1}{2}u(\mathbf{Q}_2) \\
& + \frac{1}{2\pi}\int_0^{2\pi}\int_{t_n}^{t_n+\Delta t}\cos(\theta)\,\beta(t,\theta)\,dt\,d\theta
\end{aligned}$$

$$
\begin{aligned}
&+\frac{1}{2\pi}\int_0^{2\pi}\int_{t_n}^{t_n+\Delta t} -\sin(\theta)\cos(\theta)g\frac{\rho'}{\rho}(\mathbf{x}_1(t,\theta))+\cos(\theta)\frac{v(\mathbf{x}_1(t,\theta))}{\tilde{\rho}\,a}\partial_y\bar{p}\,dt\,d\theta\\
&-\frac{1}{2\tilde{\rho}}\int_{t_n}^{t_n+\Delta t}\partial_x p'(\mathbf{x}_1(t))\,dt\,, \qquad (8)\\
v(\mathbf{P}) &= \frac{1}{2\pi}\int_0^{2\pi}\left[-\frac{p'(\mathbf{Q}_1)}{\tilde{\rho}a}\sin(\theta)+u(\mathbf{Q}_1)\cos(\theta)\sin(\theta)+v(\mathbf{Q}_1)\sin^2(\theta)\right]d\theta\\
&+\frac{1}{2}v(\mathbf{Q}_2)\\
&+\frac{1}{2\pi}\int_0^{2\pi}\int_{t_n}^{t_n+\Delta t}\sin(\theta)\,\beta(t,\theta)\,dt\,d\theta\\
&+\frac{1}{2\pi}\int_0^{2\pi}\int_{t_n}^{t_n+\Delta t} -\sin^2(\theta)g\frac{\rho'}{\rho}(\mathbf{x}_1(t,\theta))+\sin(\theta)\frac{v(\mathbf{x}_1(t,\theta))}{\tilde{\rho}\,a}\partial_y\bar{p}\,dt\,d\theta\\
&-\frac{1}{2\tilde{\rho}}\int_{t_n}^{t_n+\Delta t_n}\partial_y p'(\mathbf{x}_2(t))\,dt-\frac{1}{2}g\int_{t_n}^{t_n+\Delta t}\frac{\rho'}{\rho}(\mathbf{x}_2(t))\,dt\,, \qquad (9)\\
p'(\mathbf{P}) &= \frac{1}{2\pi}\int_0^{2\pi}\left[p'(\mathbf{Q}_1(\theta))-\tilde{\rho}au(\mathbf{Q}_1(\theta))\cos(\theta)-\tilde{\rho}av(\mathbf{Q}_1(\theta))\sin(\theta)\right]d\theta\\
&-\tilde{\rho}a\frac{1}{2\pi}\int_0^{2\pi}\int_{t_n}^{t_n+\Delta t}\beta(t,\theta)\,dt\,d\theta\\
&-\frac{1}{2\pi}\int_0^{2\pi}\int_{t_n}^{t_n+\Delta t} -\sin(\theta)\tilde{\rho}a\,g\frac{\rho'}{\rho}(\mathbf{x}_1(t,\theta))+v(\mathbf{x}_1(t,\theta))\partial_y\bar{p}\,dt\,d\theta. \qquad (10)
\end{aligned}
$$

Here $\beta(t,\theta)=a\,[\partial_x u\,\sin^2(\theta)-(\partial_y u+\partial_x v)\sin(\theta)\cos(\theta)+\partial_y v\,\cos^2(\theta)]$ and $\mathbf{P}=(x,y,t+\Delta t)$, $\mathbf{Q}_1(\theta)=(x-(\tilde{u}-a\cos(\theta))\Delta t,\ y-(\tilde{v}-a\sin(\theta))\Delta t,\ t_n)$, $\mathbf{Q}_2=(x-\tilde{u}\Delta t,\ y-\tilde{v}\Delta t,\ t_n)$ are respectively the pick and footpoints of the bicharacteristics that generate the mantle of the bicharacteristic cone, cf. Figure 1.

To obtain a time explicit approximate evolution operator the above exact integral representation needs to be approximated. First, time integrals along the mantle of the bicharacteristic cone are approximated using the rectangle rule. Integrals along the base perimeter, that obtain $\beta(t_n,\theta)$ terms, are replaced by means of the integration by parts, cf. Lemma 2.1 [25]. Further we approximate $\frac{\rho'}{\rho}$ with $\frac{\rho'}{\bar{\rho}}$ and substitute the condition for hydrostatic balance $\partial_y\bar{p}=-\bar{\rho}g$. This yields $\partial_y\bar{\rho}=-\pi^{-1}\frac{c_v}{c_p\,R\bar{\theta}}\bar{\rho}g=-\frac{\bar{\rho}g}{\bar{a}^2}$ used in the approximation for $\rho'(\mathbf{P})$. For more details on the derivation of the approximate evolution operator see [21]. This procedure yields finally the desired cell interface values $\mathbf{q}^*\equiv(\rho'(\mathbf{P}),u(\mathbf{P}),v(\mathbf{P}),p'(\mathbf{P}))=EG\,\mathbf{q}_h$ in (4). We should point out that all integrals along the base perimeter (sonic circle), i.e. integrals with respect to θ, are evaluated exactly. We make a transformation of the actual triangle to the reference triangle, where the corresponding integrals along the arcs of sonic circle were precomputed with the help of the computer algebra package Mathematica.

4 Numerical experiments

To verify the accuracy and computational performance of the GPU accelerated code, we carry out two test case simulations. For our tests we have chosen free convection of a smooth warm air bubble as introduced by Giraldo and Restelli [26] as well as free convection of a large warm bubble with a small cold bubble placed on top of the warm one as introduced by Robert [27].

In the first experiment shown in Figure 2, the warm bubble is placed at $x_c = 500$m, $y_c = 350$m with the initial temperature perturbation:

$$\theta' = \begin{cases} 0 & \text{for } r > r_c \\ (\theta'_c/2)\left[1 + \cos\left(\pi r/r_c\right)\right] & \text{for } r \le r_c \end{cases}$$

where $\theta'_c = 0.5^{\circ}$C is the maximal initial amplitude, the bubble radius $r_c = 250$m, and r the distance to the center of the bubble (x_c, y_c).

In the Robert experiment, two bubbles are placed at $(x_c, y_c) = (500\text{m}, 300\text{m})$ and $(x_c, y_c) = (560\text{m}, 640\text{m})$, for the warm and the cold bubbles, respectively (Figure 3). The maximal initial temperature amplitudes are $\theta'_c = 0.5^{\circ}$C and $\theta'_c = -0.15^{\circ}$C, respectively. The profiles of the initial perturbation for the excess potential temperature are given by a Gaussian distribution

$$\theta' = \begin{cases} \theta'_c & \text{for } r \le r_c \\ \theta'_c \exp\left[-\left(r - r_c\right)^2/50^2\right] & \text{for } r > r_c \end{cases}$$

with a flat core of radius $r_c = 150$ for the warm bubble and $r_c = 0$ for the cold bubble.

As already mentioned above, in order to simulate efficiently localized structures arising in geophysical flows, such as cloud boundaries, adaptive mesh refinement is a very suitable tool. In our work, both numerical experiments were performed on the domain of 1km×1km with no-flux boundary conditions, using the h-adaptive mesh refinement method, where the spatial resolution is adapted by refining or coarsening the mesh cells. The maximal resolution degree of the mesh is $n = 12$, which yields the finest resolution of about $1000/\sqrt{2}^{n+1} \approx 11$m per shortest edge in the simulation domain. We work with the function library AMATOS of Behrens et al. [28], where h-adaptive mesh refinement is based on the space filling curve approach. Analogously as in [19], in the numerical experiments presented below we use a slightly modified, simple refinement criterion

$$\max_{\mathbf{x}\in\Omega_e} \left[\text{sgn}(\theta'_c)\theta'(\mathbf{x}, t)\right] \ge \sigma|\theta'_c| \tag{11}$$

for the deviation of the potential temperature from the background state $\theta' = \theta - \bar{\theta}$; $\sigma \ll 1$ is a test dependent parameter (for the numerical experiments in this work we use $\sigma = 0.1$), and θ'_c is the maximal initial amplitude for the perturbation of the potential temperature (discussed below).

If condition (11) holds on some element Ω_e, the element will be recursively refined up to a specified finest mesh resolution. In the rest of the computational domain the mesh is adaptively coarsened, see also [19] for further details. Furthermore, we have used the software package CLOUD-FLASH of Müller, Giraldo et al. [19, 23] where the discontinuous Galerkin method (4) is implemented. In our work we have generalized the toolbox CLOUD-FLASH by including the GPU implementation of the genuinely multidimensional EG operator.

In both experiments, due to the differences in the air density of the bubbles and the isothermal environment, the initially resting bubbles develop a vertical motion. In the Giraldo–Restelli test shown in Figure 2 for both the GPU and CPU simulations, the warm bubble rises and deforms symmetrically due to the shear friction with the surrounding air at the warm/cold air interface, adapting a mushroom-like shape gradually. The results for the GPU simulations in the Robert experiment are shown in Figure 3. The shape of the rising warm bubble is affected in addition by the small cold bubble, which slides downwards along the right-hand side of the interface, destroying the symmetry of the warm bubble.

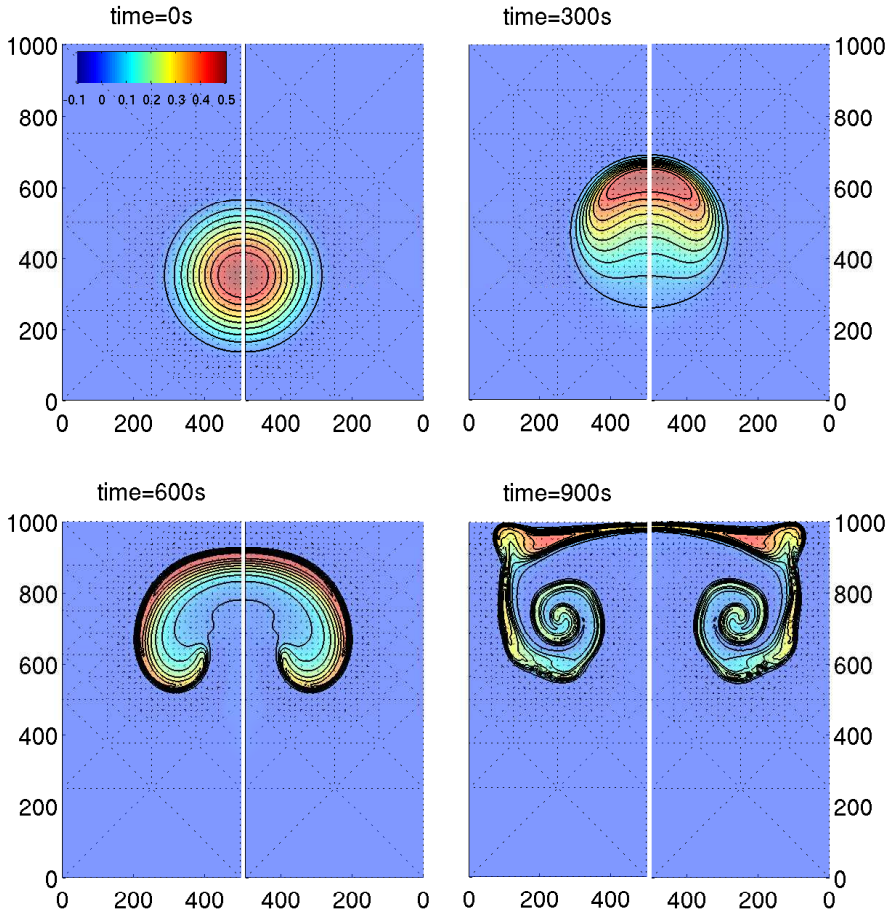

Fig. 2. Excess potential temperature θ' for the rising thermal bubble experiment as introduced by Giraldo and Restelli [26], on an adaptive resolution grid with the coarse/fine grid resolution levels $n = 1 - 12$, respectively. The left-hand side: CPU simulations, on the right-hand side: accelerated GPU implementation. The real-world domain is 1km×1km (only a half of the squared computational domain is shown in the x-direction); the shortest edge of the adaptive mesh elements corresponds to $\approx$ 11m. The simulation times are as indicated. Contour levels correspond to $\theta' = 0.025$, 0.075, 0.125, 0.175, 0.225, 0.275, 0.325, 0.375, and 0.425°C.

By comparing the GPU and CPU results, one can recognize no difference between the solutions obtained by CPU and GPU program codes (shown in Figure 2). This is an important issue since our simulations were performed in single precision on the GPU and in double precision on the CPU. When solving differential equations, slight deviations in initial data or higher inaccuracy of intermediate solution can develop to a different final solution in long time simulations. In order to quantify expected deviations between the GPU and CPU solutions we calculate the L_2 norm

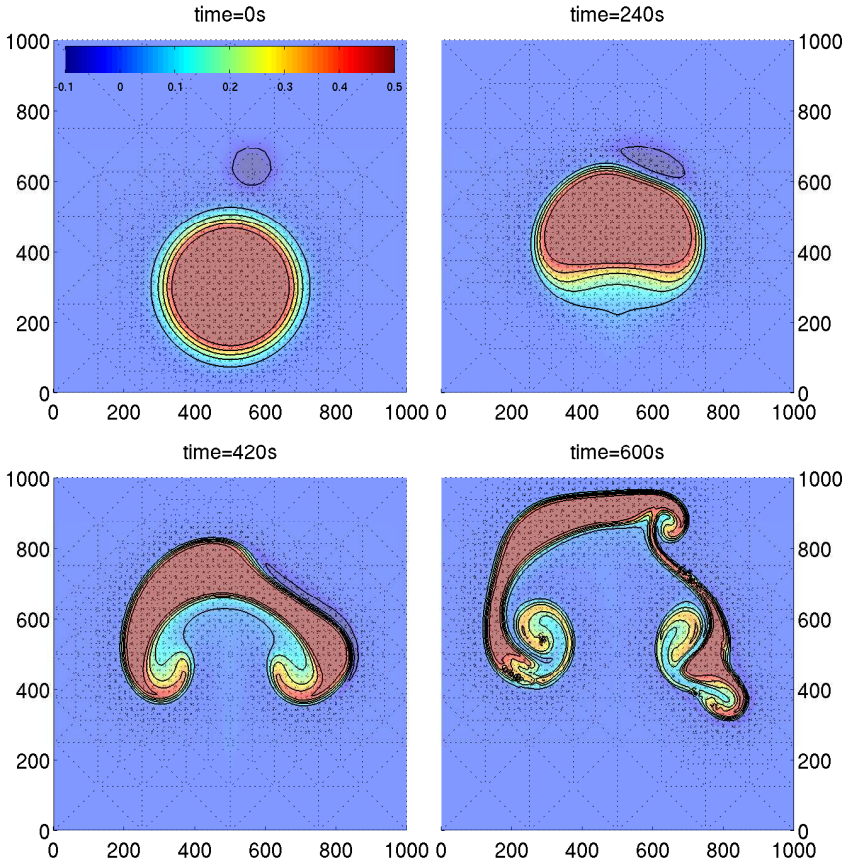

Fig. 3. Excess potential temperature θ' for a large warm air bubble with a small cold bubble on top, as introduced by Robert [27], obtained by the accelerated GPU simulations on an adaptive resolution grid with the coarse/fine grid resolution levels $n = 2 - 11$, respectively. The real-world domain is 1km×1km; the shortest edge the adaptive grid element corresponds to ≈ 15.6m. The simulation times are as indicated. Contour levels correspond to $\theta' = -0.05$, 0.05, 0.15, 0.25, 0.35, and 0.45°C.

$$L_2(\mathbf{q}) = N^{-1} \left[\sum_{j=1}^{N} \left(\mathbf{q}_{i,\mathrm{CPU}} - \mathbf{q}_{i,\mathrm{GPU}} \right)^2 \right]^{1/2} \tag{12}$$

where N is the number of degrees of freedom. Of course, the grids in both simulations (performed on GPU and CPU) must have the same structure for this comparison, that cannot be expected in the simulations using adaptive grids. For this reason we have additionally performed simulations on a regular grid with the resolution level $n = 10$, which yields 4096 mesh cells (triangles). This corresponds approximately to the number of mesh elements in simulations on the adaptively refined grid with fine resolution level $n = 12$. In Figure 4 one can see that discrepances are indeed present in both the experiments, Giraldo–Restelli and Robert, however, they are of the order of magnitude of rounding errors in the single precision arithmetics and they remain bounded during the simulations. The largest error has been found for the energy

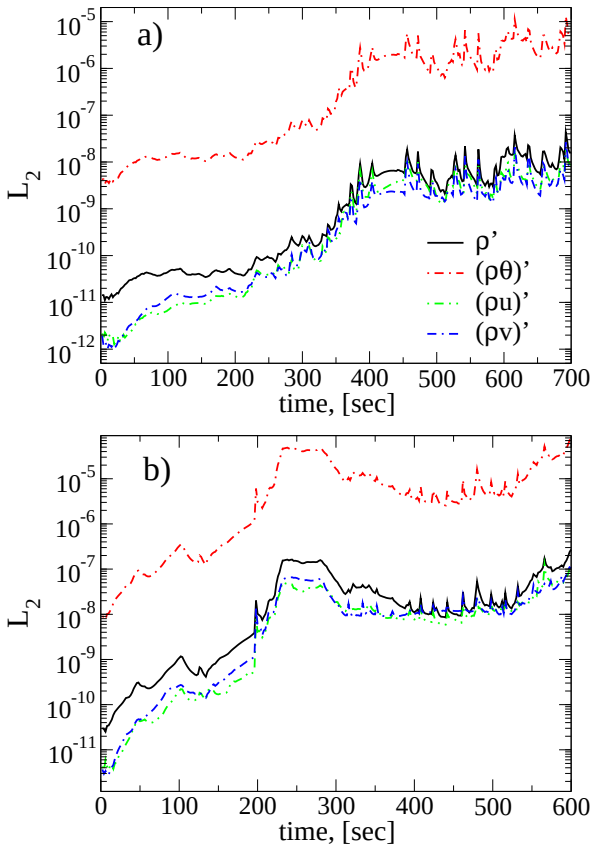

Fig. 4. The L_2 norm calculated from (12) for the solutions obtained by GPU and CPU codes on regular mesh with $n = 10$ in a) Giraldo–Restelli and b) Robert experiments.

variable, $\rho\theta$, which is due to the fact that the potential temperature, θ, is by 2-3 orders of magnitude larger than the other variables in our tests.

5 GPU implementation

To evaluate the runtime spent in different parts of the program, we used the Giraldo–Restelli test case with 16384 mesh cells. The total procedure for the calculation of the EG operator takes up to about 85% of the computation time of the whole program in this special case. To date, all of the GPU related work has been done to speed up this procedure, which is structured into a few subprocedures taking the following CPU times in the above described test case:

compute EG procedure:

- change basis
  - −compute quadrature points [0.11 %]
  - −compute basis transformation [3.90 %]
- find intersections of wave fronts with elements' edges
  - −compute linearized state [1.10 %]
  - −compute arcs [4.80 %]
- calculate approx. fields, cf. (7-10)
  - −compute $\rho'(\mathbf{P})$, $u(\mathbf{P})$, $v(\mathbf{P})$, $p'(\mathbf{P})$ [90.01 %]

The first step was to port all the code related to the computation of this operator from FORTRAN to C which consisted of several thousand lines of code. Since the data of the main program is stored in the main computer memory, for the calculation of the EG operator on the GPU, the input and output fields have to be transferred to/from the GPU before/after execution. Furthermore, the host program calculates all the fields and quantities in double precision. Therefore, before being transferred to the GPU, the data must be converted into a single precision floating point representation.

We noticed that the data transfer time as well as the conversion time are very low compared to the calculations that are running on the GPU. The last part that takes up over 90% of the execution time was ported to GPU first, to process all the mesh cells in parallel as a heavy weight kernel that needs 63 registers and 1000 bytes of stack memory. This limits GPU occupancy to 33%, but this version of the kernel still performs so fast that the rest of the subprocedures become the new bottleneck of the program. In future work we plan to optimize the kernel to increase the GPU occupancy and, hence, the overall performance of the code.

The next step was to bring the computation of the linearized state, arcs, and basis transformation to the GPU. The computation of basis transformation proved to be the most problematic and least efficient on the GPU, but it was necessary to process it on the GPU since copying the data back and forth between CPU and GPU memory in the middle of the computation is unacceptable.

We define the speedup of a GPU implementation as the ratio

$$s = \frac{t_{\mathrm{CPU}}}{t_{\mathrm{GPU}}} \tag{13}$$

between the time t_{CPU} that is spent on computation in a nonaccelerated implementation and the time t_{GPU} that is spent on computations in a GPU accelerated implementation.

Since not all of the program code can be accelerated using a GPU, a fraction of the program code will always be executed on the CPU. Hence, we split the time t_{GPU} into two parts

$$t_{\mathrm{GPU}} = t_{\mathrm{GPU,\ accel}} + t_{\mathrm{unaccel}}, \tag{14}$$

where $t_{\mathrm{GPU,\ accel}}$ is the execution time for the part of the program that is actually executed in parallel on a GPU and t_{unaccel} is for the rest of the code that stays in the CPU and is not executed in parallel.

For the CPU time, we can do the same split

$$t_{\text{CPU}} = t_{\text{CPU, accel}} + t_{\text{unaccel}}, \tag{15}$$

which yields the following formula for the speedup of the GPU accelerated program

$$s = \frac{t_{\text{CPU, accel}} + t_{\text{unaccel}}}{t_{\text{GPU, accel}} + t_{\text{unaccel}}}. \tag{16}$$

Here $t_{\text{CPU, accel}}$ refers to the execution time on CPU of that part of the program which we have also implemented in parallel in GPU accelerated version.

From (16), the maximum possible speedup corresponding to the case $t_{\text{GPU,accel}} = 0$ is

$$s_{\max} = \frac{t_{\text{CPU, accel}}}{t_{\text{unaccel}}} + 1. \tag{17}$$

In our case the maximum speedup to be expected, if the whole procedure 'compute EG' is brought to GPU is $s_{\max} = 6.7$ for grid size 12. This clearly justifies the effort of a GPU implementation of the procedure in our case.

To benchmark our implementation, we use a NVidia Geforce GTX 580 with a Intel Core i7 Nehalem processor at 2.67 GHz. The measured execution times are reported in Table 1 and compared in Figure 5 for the CPU and GPU implementations with the grid size level up to $n = 12$, which corresponds to up to 16384 finite volume elements on our computation domain.

Grid size level, n	8	9	10	11	12
Number of finite elements	1024	2048	4096	8192	16384
$t_{\text{CPU, accel}}$/sec	0.077	0.152	0.307	0.650	1.650
$t_{\text{GPU, accel}}$/sec	0.0045	0.0065	0.0126	0.0026	0.0569
$s_{\text{EG}} = t_{\text{CPU,accel}}/t_{\text{GPU, accel}}$	17.11	23.38	24.36	25.00	29.70
s_{tot}	5.0	5.4	5.5	5.5	6.6

Table 1. Execution times in the Giraldo–Restelli experiment for regular grid of different grid sizes. The net speedup of the GPU implemented parts of the code, s_{EG}, is by a factor up to 30 faster if compared to the CPU execution times. However, due to the non-parallelized parts of the program still running on the CPU, the performance speedup of the whole program, s_{tot}, is much lower (cf. (16)).

Since the kernels are very complex and large, it is hard to predict execution times for finer resolved grids. In our simulations we were restricted to $n = 12$ due to the CFL condition, which relates the sizes of finite elements to the time step used in the explicit time integration scheme for our problems. However, one can see in Figure 5 that we achieved the most efficient speedup for 'compute EG' using our current implementation for $n = 12$. The raw acceleration factor for the GPU implemented 'compute EG' procedure is nearly 30 for grid size 12, which means we are nearly reaching the theoretically maximum possible speedup of about 6.7 for this grid size.

Because of the massive speedups gained in the parts of the program executed on the GPU, the execution time of the whole program is largely determined by the parts of the program that remain on the CPU. This means in practice that the differences in speedup for the different grid sizes are barely noticeable in reality and we can expect a relatively stable speedup factor of about 5.0-6.6 for the overall simulation in a real-world example. As the computation of the EG operator is no longer the bottleneck of our program, future work will focus on parallelizing the rest of the code base.

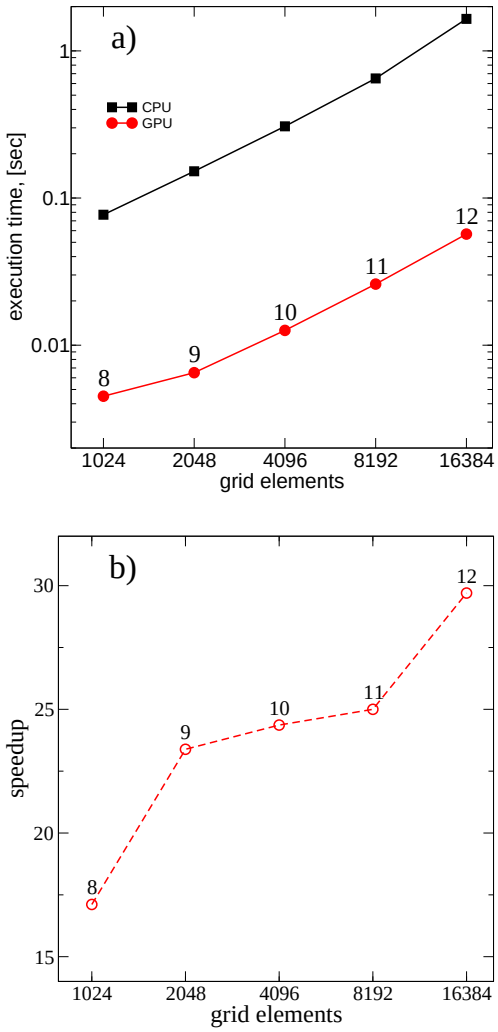

Fig. 5. a) Execution times in the Giraldo–Restelli experiment for regular grids of different resolutions. b) Numerical speedup of the GPU implemented code for different number of mesh cells in the computation domain. The numbers annotating the symbols are for the grid resolution level, n, (cf. Table 1).

6 Concluding remarks

In the present paper we reported on our recent results for a GPU accelerated implementation of the Euler equations for fully compressible flows. The numerical simulations were performed using a discontinuous Galerkin method coupled with the genuinely multidimensional, bicharacteristic based evolution operator used for the flux integration along cell interfaces. For spatial approximation we use second order polynomials with the h-adaptive mesh refinement method and the explicit third order Runge-Kutta method for time integration. Evaluation of the multidimensional evolution operator has been accelerated using NVidia's CUDA Framework. We benchmarked the GPU accelerated code on regular and adaptive grids and compared to the results obtained from CPU simulations showing good agreement. For the GPU

accelerated parts of the code we have obtained a significant speedup of a factor up to 30 in comparison to a single CPU core, with a potential for further improvement of the performance of the code.

Acknowledgements

This work has been supported by the German Science Foundation under the DFG grant LU 1470/2-2 as well as by the research grant of the Center of Computational Sciences in Mainz. M.L. and L.Y. would like to thank A. Müller, V. Wirth (Mainz) and F.X. Giraldo (Monterey) for fruitful discussions on the topic and for providing their software package CLOUD-FLASH. B.B. would like to thank the MPGC for funding.

References

1. Hundertmark, A., Lukáčová-Medvid'ová, M., Prill, F., J. Sci. Comp. **48**, (2011), 227-240.
2. Lukáčová-Medvid'ová, M., Morton, K. W., Warnecke, G., SIAM J. Sci. Comp. **26(1)**, (2004), 1-30.
3. Lukáčová-Medvid'ová, M., Noelle, S., Kraft, M., J. Comput. Phys. **221**, (2007), 122-147.
4. Preis, T., Virnau, P., Wolfgang, P., Schneider, J.J., J. Comput. Phys., **228** (2009) 4468-4477.
5. Block, B.J., Virnau, P., Preis, T., Comp. Phys. Communic., **181** (2010) 1549-1556.
6. Weigel, M., Phys. Rev. E, **84** (2011) 036709.
7. Anderson, J.A., Lorenz, Ch.D., Travesset, A., J. Comput. Phys., **227** (2008) 5342-5359.
8. van Meel J.A., Arnold, A., Frenkel, D., Portegies Zwart, S.F., Belleman, R.G., Mol. Sim., **34** (2008) 259-266.
9. Colberg, P.H., Höfling, F:, Comp. Phys. Comm., **182**, (2011) 1120.
10. Reith, D., Mirny, L., Virnau, P., Prog. Theor. Phys. Suppl., **191**, (2011) 135-145.
11. Reith, D., Milchev, A., Virnau, P., Binder, K., EPL, **95** (2011) 28003.
12. Preis T., Virnau, P., Wolfgang, P., Schneider, J.J., New J. Physics, **11** (2009) 093024.
13. Brodtkorb, A.R., Hagen, T.R., Lie, K.-A., Natvig, J.R. Comput. Visual. Sci., **13** (2010) 341-353.
14. Castro, M.J., Ortega, S., Asunción, M. de la, Mantas, J.M., Bol. Soc. Esp. Mat. Apl. **50** (2010), 27-44.
15. Hagen, T.R.; Henriksen, M.O., Hjelmervik, J.M., Lie, K.-A., *How to solve systems of conservation laws numerically using the graphics processor as a high-performance computational engine. Geometric modelling, numerical simulation, and optimization: applied mathematics at SINTEF*, (Springer, Berlin, 2007), 211-264.
16. Xian, W., Takayuki, A., Parallel Computing, **37**, (2011), 521-535.
17. Appleyard, J., Drikakis, D., Computers & Fluids, **46**, (2011) 101-105.
18. Giraldo, F.X, Restelli, M., Lauter, M., SIAM J. Sci. Comp. **32**, (2010), 3394-3425.
19. Müller, A., Behrens, J., Giraldo, F.X., Wirth, V., J. Comput. Phys., submitted (2011).
20. Straka, J.M., Wilhelmson, R.B., Wicker, L.J., Anderson, J.R., Droegemeier, Int. J. Numer. Fl. **17**, (1993), 1-22.
21. Lukáčová-Medvid'ová, M., Müller, A., Giraldo, F.X., Wirth, V., Yelash, L., in preparation (2012).
22. Giraldo, F.X, Restelli, M., SIAM J. Sci. Comput. **31**, (2009), 2231-2257.
23. Müller, A., Behrens, J., Giraldo, F.X., Wirth, V., Proceedings of the V European Conference on Computational Fluid Dynamics ECCOMAS CFD 2010, (2010), 1-20.
24. Giraldo, F.X, Wartburton, T., Int. J.Numer. Meth. Fl. **56**, (2008), 899-925.
25. Lukáčová-Medvid'ová, M., Morton, K. W., Warnecke, G., MathComp. **69**, (2000), 1355-1384.
26. Giraldo, F.X, Restelli, M., J. Comput. Phys. **227**, (2008), 3849-3877.
27. Robert, A., J. Atmos. Sci. **50**, (1993), 1865-1873.
28. Behrens, J., Rakowsky, N., Hiller, W., Handorf, D., Lauter, M., Papke, J., Dethloff, K., Ocean Model **10**, (2005), 171-183.