Proceedings of the 12th International Conference on Computational and Mathematical Methods in Science and Engineering, CMMSE 2012 La Manga, Spain, July, 2–5, 2012.

Evaluating the impact of reordering unstructured meshes on the performance of finite volume GPU solvers

Marc de la Asunción[1], José M. Mantas[1] and Manuel J. Castro[2]

[1] *Depto. Lenguajes y Sistemas Informáticos, Universidad de Granada*

[2] *Depto. Análisis Matemático, Universidad de Málaga*

emails: `marc@correo.ugr.es`, `jmmantas@ugr.es`, `castro@anamat.cie.uma.es`

Abstract

In this work, we study the impact of renumbering the cells of unstructured triangular finite volume meshes on the performance of CUDA implementations of several finite volume schemes to simulate two-layer shallow water systems. We have used several numerical schemes with different demands of computational power whose CUDA implementations exploit the texture and L1 cache units of the GPU multiprocessors. Two different reordering schemes based on reducing the bandwidth of the adjacency matrix for the volume mesh have been used. Several numerical experiments performed on a Fermi-class GPU show that enforcing an ordering which enhances the data locality can have a significant impact on the runtime, and this impact is higher when the numerical scheme is computationally expensive.

1 Introduction

Currently, Graphics Processing Units (GPUs) are being used extensively to accellerate considerably numerical simulations in science and engineering. These platforms make it possible to achieve speedups of an order of magnitude over a standard CPU in many applications and are growing in popularity [16]. In particular, GPUs have been used in many applications based on finite volume numerical schemes [1, 2, 3, 7]. Currently, most of the GPU implementations of numerical schemes are based on the CUDA framework [14] which includes an extension of the C/C++ language to facilitate the programming of NVIDIA GPUs for general purpose applications.

Although the performance of finite volume computations for unstructured meshes could be substantially improved by using GPU platforms, the irregularity of the memory access patterns hampers this goal.

Obtaining high performance on a CUDA-enabled GPU implementation of unstructured mesh computations is not easy because mesh data cannot be easily laid out so as to enable coalescing [15]. However, the renumbering of the data cells of a mesh has proved to be an important way to improve the performance in parallel numerical computations which work on unstructured meshes [4] because a suitable data ordering optimizes the cache usage. In GPU, this approach could be possible if the ordering enables enough locality to use textures and to improve the L1 cache usage.

In modern CUDA-enabled GPUs, reads from texture memory are cached in a manner that preserves spatial locality, meaning that data reads from nearby points in space will possibly be cache hits. On the other hand, in Fermi class GPUs, the same on-chip memory can be dedicated mostly as L1 cache for each kernel call to reduce bandwidth demand [15]. A better access to texture and global memory can be achieved by renumbering the elements in an unstructured mesh such that the elements nearby in the mesh remain nearby in texture and global memory, enabling a better exploitation of the texture and L1 cache. Thus, one can obtain substantial performance improvements without changing the code.

In this paper, we study the impact of renumbering the cells of unstructured triangular finite volume meshes on the GPU performance for CUDA implementations of several finite volume two-layer shallow water solvers. These CUDA solvers have been implemented to take advantage of the texture and L1 cache units of a Fermi-class GPU, and exhibit different numerical intensity profiles. In order to apply a cell reordering which enhances the data locality, two reordering schemes based on reducing the bandwidth of the adjacency matrix for the mesh are used. Our goal is to evaluate the effect of these reordering techniques on the runtime of the finite volume CUDA solvers.

The outline of the article is as follows: the next section describes the underlying mathematical model and presents three finite volume numerical schemes to solve it. In Section 3 the CUDA implementation of the schemes is briefly described. Next, two bandwidth reduction techniques which will be used as renumbering strategies are introduced in Section 4. Section 5 shows and analyzes the performance results obtained when the different CUDA solvers are applied to two test problems on a NVIDIA GTX 580 GPU using different ordering strategies. Finally, conclusions are drawn in Section 6.

2 Mathematical model and numerical schemes

The two-layer shallow water system [5] is a system of partial differential equations which governs the 2d flow of two superposed immiscible layers of shallow fluids in a subdomain $\Omega \subset \mathbb{R}^2$. This system has been used as the numerical model to simulate ocean currents, oil spills or tsunamis generated by underwater landslides and has the following form:

$$\frac{\partial W}{\partial t}+\frac{\partial F_1}{\partial x}(W)+\frac{\partial F_2}{\partial y}(W)=B_1(W)\frac{\partial W}{\partial x}+B_2(W)\frac{\partial W}{\partial y}+S_1(W)\frac{\partial H}{\partial x}+S_2(W)\frac{\partial H}{\partial y}, \quad (1)$$

where $W=\begin{pmatrix} h_1 & q_{1,x} & q_{1,y} & h_2 & q_{2,x} & q_{2,y}\end{pmatrix}^T$,

$$F_1(W)=\begin{pmatrix} q_{1,x} & \dfrac{q_{1,x}^2}{h_1}+\dfrac{1}{2}gh_1^2 & \dfrac{q_{1,x}q_{1,y}}{h_1} & q_{2,x} & \dfrac{q_{2,x}^2}{h_2}+\dfrac{1}{2}gh_2^2 & \dfrac{q_{2,x}q_{2,y}}{h_2}\end{pmatrix}^T,$$

$$F_2(W)=\begin{pmatrix} q_{1,y} & \dfrac{q_{1,x}q_{1,y}}{h_1} & \dfrac{q_{1,y}^2}{h_1}+\dfrac{1}{2}g\,h_1^2 & q_{2,y} & \dfrac{q_{2,x}q_{2,y}}{h_2} & \dfrac{q_{2,y}^2}{h_2}+\dfrac{1}{2}gh_2^2\end{pmatrix}^T,$$

$$S_k(W)=\begin{pmatrix} 0 & gh_1(2-k) & gh_1(k-1) & 0 & gh_2(2-k) & gh_2(k-1)\end{pmatrix}^T, \quad k=1,2,$$

$$B_k(W)=\begin{pmatrix} \mathbf{0} & \mathcal{P}_{1,k}(W) \\ r\mathcal{P}_{2,k}(W) & \mathbf{0}\end{pmatrix}, \quad \mathcal{P}_{l,k}(W)=\begin{pmatrix} 0 & 0 & 0 \\ -gh_l(2-k) & 0 & 0 \\ -gh_l(k-1) & 0 & 0\end{pmatrix}, \quad l=1,2.$$

Index 1 in the unknowns makes reference to the upper fluid layer and index 2 to the lower one; g is the gravity and $H(\boldsymbol{x})$, the depth function measured from a fixed level of reference; $r=\rho_1/\rho_2$ is the ratio of the constant densities of the layers ($\rho_1<\rho_2$) which, in realistic oceanographical applications, is close to 1. Finally, $h_i(\boldsymbol{x},t)$ and $\boldsymbol{q}_i(\boldsymbol{x},t)$ are, respectively, the thickness and the mass-flow of the i-th layer at the point $\boldsymbol{x}$ at time t, and they are related to the velocities $\boldsymbol{u}_i(\boldsymbol{x},t)=(u_{i,x}(\boldsymbol{x},t),u_{i,y}(\boldsymbol{x},t))$, $i=1,2$ by the equalities: $\boldsymbol{q}_i(\boldsymbol{x},t)=\boldsymbol{u}_i(\boldsymbol{x},t)h_i(\boldsymbol{x},t), \quad i=1,2.$

To discretize System (1), the computational domain D is divided into L cells or finite volumes $V_i\subset\mathbb{R}^2$, which are assumed to be triangles. Given a finite volume V_i, $N_i\in\mathbb{R}^2$ is the barycenter of V_i, $\aleph_i$ is the set of indexes j such that V_j is a neighbour of V_i; Γ_{ij} is the common edge of two neighbouring cells V_i and V_j, and $|\Gamma_{ij}|$ is its length; $\boldsymbol{\eta}_{ij}=(\eta_{ij,x},\eta_{ij,y})$ is the unit vector which is normal to the edge Γ_{ij} and points towards V_j [5] (see Fig. 1).

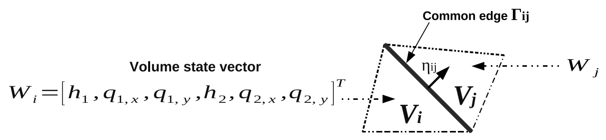

Common edge Γ_{ij}
Volume state vector
$W_i=[h_1, q_{1,x}, q_{1,y}, h_2, q_{2,x}, q_{2,y}]^T$
η_{ij}
V_i
V_j
W_j

Figure 1: Finite volumes

Assume that the approximations at time t^n, W_i^n ($i=1,\ldots,L$), have already been calculated. To advance in time, with Δt^n being the time step, all the numerical schemes which will be used have the following general form:

$$W_i^{n+1} = W_i^n - \frac{\Delta t^n}{|V_i|} \sum_{j\in\aleph_i} |\Gamma_{ij}|\, F_{ij}^-(W_i^n, W_j^n, H_i, H_j) \tag{2}$$

where $|V_i|$ is the area of V_i.

The computation of $F_{ij}^-(W_i^n, W_j^n, H_i, H_j) \in \mathbb{R}^6$ in (2) depends on the particular numerical scheme. Here, three different numerical schemes will be presented: the classical Roe scheme [5], the IR-Roe scheme [2] and the PVM-IFCP scheme [6].

To compute the n-th time step, the following condition can be used:

$$\Delta t^n = \min_{i=1,\dots,L} \left\{ \left[\frac{\sum_{j\in\aleph_i} \mid \Gamma_{ij} \mid\parallel \mathcal{D}_{ij} \parallel_\infty}{2\gamma \mid V_i \mid} \right]^{-1} \right\} \tag{3}$$

where γ, $0 < \gamma \leq 1$, is the CFL (Courant-Friedrichs-Lewy) parameter.

2.1 The classical Roe scheme

In the classical Roe scheme, F_{ij}^- (herein called F_{ij}^{ROE-}) is computed as follows:

$$\begin{aligned}
\mathcal{F}_{ij}^{ROE^-}(W_i^n, W_j^n, H_i, H_j) &= P_{ij}^- \left(A_{ij}\left(W_j^n - W_i^n\right) - S_{ij}\left(H_j - H_i\right)\right) + F_{\boldsymbol{\eta}_{ij}}(W_i^n)\,, \\
\mathcal{F}_{ij}^{ROE^+}(W_i^n, W_j^n, H_i, H_j) &= P_{ij}^+ \left(A_{ij}\left(W_j^n - W_i^n\right) - S_{ij}\left(H_j - H_i\right)\right) - F_{\boldsymbol{\eta}_{ij}}(W_j^n)\,.
\end{aligned}$$

$F_{ij}^{ROE^-}$ and $F_{ij}^{ROE^+}$ are the contributions of the edge Γ_{ij} to the state of the volumes V_i and V_j, respectively, where $F_{\boldsymbol{\eta}_{ij}}(W) = F_1(W)\,\eta_{ij,x} + F_2(W)\,\eta_{ij,y}$ and $H_\alpha = H(N_\alpha)$ with $\alpha = i, j$. $A_{ij} \in \mathbb{R}^{6\times 6}$ and $S_{ij} \in \mathbb{R}^6$ depends on W_i^n and W_j^n (see [5] for more details). The matrix $P_{ij}^\pm$ is calculated as:

$$P_{ij}^\pm = \frac{1}{2}\mathcal{K}_{ij} \cdot (I \pm \operatorname{sgn}(\mathcal{D}_{ij})) \cdot \mathcal{K}_{ij}^{-1}$$

where I is the identity matrix, $\operatorname{sgn}(\mathcal{D}_{ij})$ is a diagonal matrix whose coefficients are the sign of the eigenvalues of A_{ij}, and the columns of $\mathcal{K}_{ij} \in \mathbb{R}^{6\times 6}$ are the associated eigenvectors (see [5] for more details).

2.2 The IR-Roe scheme

The IR-Roe scheme exploits that system (1) verifies the property of rotational invariance (see [2] for more details) to reduce the computational costs without losing excessive accuracy. The resultant formula for $F_{ij}^\pm$ (herein called $F_{ij}^{IR-ROE\pm}$) reads as follows:

$$F_{ij}^{IR-ROE\pm} = T_{\eta_{ij}}^{-1} \left[\left(\Phi_{\boldsymbol{\eta}_{ij}}^\pm\right)_{[1]} \quad \left(\Phi_{\boldsymbol{\eta}_{ij}}^\pm\right)_{[2]} \quad \left(\Phi_{\boldsymbol{\eta}_{ij}^\perp}^\pm\right)_{[1]} \quad \left(\Phi_{\boldsymbol{\eta}_{ij}}^\pm\right)_{[3]} \quad \left(\Phi_{\boldsymbol{\eta}_{ij}}^\pm\right)_{[4]} \quad \left(\Phi_{\boldsymbol{\eta}_{ij}^\perp}^\pm\right)_{[2]} \right]^T.$$

where:

- $T_{\eta_{ij}} = \begin{pmatrix} R_{\eta_{ij}} & \mathbf{0} \\ \mathbf{0} & R_{\eta_{ij}} \end{pmatrix}, \quad R_{\eta_{ij}} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \eta_{ij,x} & \eta_{ij,y} \\ 0 & -\eta_{ij,y} & \eta_{ij,x} \end{pmatrix}.$

- $\left(\Phi^{\pm}_{\boldsymbol{\eta}_{ij}}\right)_{[l]}$ is the l-th component of the vector $\Phi^{\pm}_{\boldsymbol{\eta}_{ij}} \in I\!R^4$ which is defined as follows:

  $$\begin{aligned} \Phi^{-}_{\boldsymbol{\eta}_{ij}} &= P^{-}_{ij}\left(\mathcal{F}_1(W_{\boldsymbol{\eta}_{ij},j}) - \mathcal{F}_1(W_{\boldsymbol{\eta}_{ij},i}) - \mathcal{B}_{ij}\left(W_{\boldsymbol{\eta}_{ij},j} - W_{\boldsymbol{\eta}_{ij},i}\right) - \mathcal{S}_{ij}\left(H_j - H_i\right)\right) + \mathcal{F}_1(W_{\boldsymbol{\eta}_{ij},i}) \, , \\ \Phi^{+}_{\boldsymbol{\eta}_{ij}} &= P^{+}_{ij}\left(\mathcal{F}_1(W_{\boldsymbol{\eta}_{ij},j}) - \mathcal{F}_1(W_{\boldsymbol{\eta}_{ij},i}) - \mathcal{B}_{ij}\left(W_{\boldsymbol{\eta}_{ij},j} - W_{\boldsymbol{\eta}_{ij},i}\right) - \mathcal{S}_{ij}\left(H_j - H_i\right)\right) - \mathcal{F}_1(W_{\boldsymbol{\eta}_{ij},j}) \, . \end{aligned}$$

  Here $P^{\pm}_{ij} = \frac{1}{2}\mathcal{K}_{ij} \cdot (I \pm \text{sgn}(\mathcal{D}_{ij})) \cdot \mathcal{K}^{-1}_{ij}$, where I is the identity matrix, $\mathcal{K}_{ij}$ is the matrix whose columns are the eigenvectors of a matrix $\mathcal{A}_{ij} \in \mathbb{R}^{4\times 4}$ which depends on W^n_i and W^n_j (see [2] for more details), and $\text{sgn}(\mathcal{D}_{ij})$ is the diagonal matrix whose coefficients are the signs of the eigenvalues of $\mathcal{A}_{ij}$. $\mathcal{F}_1(W_{\boldsymbol{\eta}_{ij},\alpha}) = F_1(T_{\boldsymbol{\eta}_{ij}} W^n_\alpha)_{[1,2,4,5]}$ $(\alpha = i, j)$, $\mathcal{S}_{ij}(H_j - H_i) = \left(S_1(W^n_{ij})(H_j - H_i)\right)_{[1,2,4,5]}$, and $\mathcal{B}_{ij}(W_{\boldsymbol{\eta}_{ij},j} - W_{\boldsymbol{\eta}_{ij},i}) = \left(B_1(W^n_{ij})(T_{\boldsymbol{\eta}_{ij}}(W^n_j - W^n_i))\right)_{[1,2,4,5]}$ where W^n_{ij} is the 'Roe intermediate state' corresponding to W^n_i and W^n_j (see [5] for more details), and $W_{[i_1,\cdots,i_s]}$ is the vector defined from vector W, using its i_1-th, ..., i_s-th components.

- $\Phi^{\pm}_{\boldsymbol{\eta}^{\perp}_{ij}} = \mp \left[\left(\Phi^{-}_{\boldsymbol{\eta}_{ij}}\right)_{[1]} u^*_{1,\boldsymbol{\eta}^{\perp}_{ij}} \quad \left(\Phi^{-}_{\boldsymbol{\eta}_{ij}}\right)_{[3]} u^*_{2,\boldsymbol{\eta}^{\perp}_{ij}} \right]^{\mathrm{T}}$,where $u^*_{k,\boldsymbol{\eta}^{\perp}_{ij}}$ is defined as follows:

  $$u^*_{k,\boldsymbol{\eta}^{\perp}_{ij}} = \begin{cases} \dfrac{q_{k,i,\boldsymbol{\eta}^{\perp}_{ij}}}{h_{k,i}} & \text{If } \left(\Phi^{-}_{\boldsymbol{\eta}_{ij}}\right)_{[2k-1]} > 0 \\ \dfrac{q_{k,j,\boldsymbol{\eta}^{\perp}_{ij}}}{h_{k,j}} & \text{Otherwise} \end{cases} \quad , \quad k = 1, 2.$$

2.3 The PVM-IFCP-scheme

The PVM (Polynomial Viscosity Matrix) schemes are a family of numerical schemes for non conservative hyperbolic systems [6] which are defined in terms of viscosity matrices obtained from the polynomial evaluation of a Roe matrix. The main advantage of these methods is that they only need some information about the eigenvalues of the system and the spectral decomposition of the Roe matrix is not needed unlike the previous schemes.

For a PVM scheme, $F^{-}_{ij}(W^n_i, W^n_j, H_i, H_j)$ is obtained by applying a similar process to that described in 2.2 to derive $\mathcal{F}^{IR-ROE^{\pm}}_{ij}$. However, the flux $\Phi^{\pm}_{\boldsymbol{\eta}_{ij}}$ is obtained by:

$$\begin{aligned}
\Phi^-_{\boldsymbol{\eta}_{ij}} &= \frac{1}{2}\Big(\mathcal{F}_1(W_{\boldsymbol{\eta}_{ij},j}) - \mathcal{F}_1(W_{\boldsymbol{\eta}_{ij},i}) - \mathcal{B}_{ij}\left(W_{\boldsymbol{\eta}_{ij},j} - W_{\boldsymbol{\eta}_{ij},i}\right) - \mathcal{S}_{ij}\left(H_j - H_i\right) \\
&\quad - Q_{ij}\left(W_{\boldsymbol{\eta}_{ij},j} - W_{\boldsymbol{\eta}_{ij},i} - A_{ij}^{-1}\mathcal{S}_{ij}\left(H_j - H_i\right)\right)\Big) + \mathcal{F}_1(W_{\boldsymbol{\eta}_{ij},i}), \\
\Phi^+_{\boldsymbol{\eta}_{ij}} &= \frac{1}{2}\Big(\mathcal{F}_1(W_{\boldsymbol{\eta}_{ij},j}) - \mathcal{F}_1(W_{\boldsymbol{\eta}_{ij},i}) - \mathcal{B}_{ij}\left(W_{\boldsymbol{\eta}_{ij},j} - W_{\boldsymbol{\eta}_{ij},i}\right) - \mathcal{S}_{ij}\left(H_j - H_i\right) \\
&\quad + Q_{ij}\left(W_{\boldsymbol{\eta}_{ij},j} - W_{\boldsymbol{\eta}_{ij},i} - A_{ij}^{-1}\mathcal{S}_{ij}\left(H_j - H_i\right)\right)\Big) - \mathcal{F}_1(W_{\boldsymbol{\eta}_{ij},j}),
\end{aligned} \tag{4}$$

where $\mathcal{F}_1(W_{\boldsymbol{\eta}_{ij}})$, $\mathcal{S}_{ij}(H_j - H_i)$ and $\mathcal{B}_{ij}(W_{\boldsymbol{\eta}_{ij},j} - W_{\boldsymbol{\eta}_{ij},i})$ are defined in 2.2. Q_{ij} is the viscosity matrix defined as $Q_{ij} = \alpha_0^{ij} I + \alpha_1^{ij} A_{ij} + \alpha_2^{ij} A_{ij}^2 + \ldots + \alpha_l^{ij} A_{ij}^l$, where I is an identity matrix and α_k^{ij}, $k = 0, \ldots, l$, are particular coefficients of the PVM scheme.

The PVM-IFCP (Intermediate Field Capturing Parabola) scheme [9] is defined by the coefficients α_k, $k = 0, 1, 2$, obtained by solving the following system:

$$\begin{pmatrix} 1 & \lambda_1 & (\lambda_1)^2 \\ 1 & \lambda_n & (\lambda_4)^2 \\ 1 & \chi_{int} & (\chi_{int})^2 \end{pmatrix} \begin{pmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \end{pmatrix} = \begin{pmatrix} |\lambda_1| \\ |\lambda_4| \\ |\chi_{int}| \end{pmatrix} \tag{5}$$

being $\lambda_1 < \lambda_2 < \lambda_3 < \lambda_4$ the eigenvalues of the matrix A_{ij} and

$$\chi_{int} = \mathcal{S}_{ext} \cdot \max_{2 \le i \le 3}(|\lambda_i|) \text{ with } \mathcal{S}_{ext} = \begin{cases} \text{sgn}(\lambda_1 + \lambda_4) & \text{si } (\lambda_1 + \lambda_4) \neq 0 \\ 1 & \text{otherwise} \end{cases}$$

The choice of these coefficients provokes that this scheme is linearly stable L^∞ with the aforementioned CFL condition (see Equation (3)).

3 CUDA implementation of the schemes

The general structure of the CUDA implementation of the three numerical schemes exposed in Section 2 is the same for all the schemes. This implementation is a variant of the implementation described in [7], Section 7.3 and [2], Section 5. The general steps of the implementation are depicted in Figure 2. Each step executed on the GPU is assigned to a CUDA kernel. Next, we briefly describe each step:

- Build finite volume mesh: Volume data is stored in two arrays of L `float4` elements as 1D textures, where each element contains the data (state, depth and area) of a cell. We have used textures because each edge (thread) only needs the data of adjacent cells and texture memory is especially suited for each thread to access its closer environment in texture memory by exploiting the texture cache. Edge data is stored in two arrays in global memory with a size equal to the number of edges: an array of `float2` elements for storing

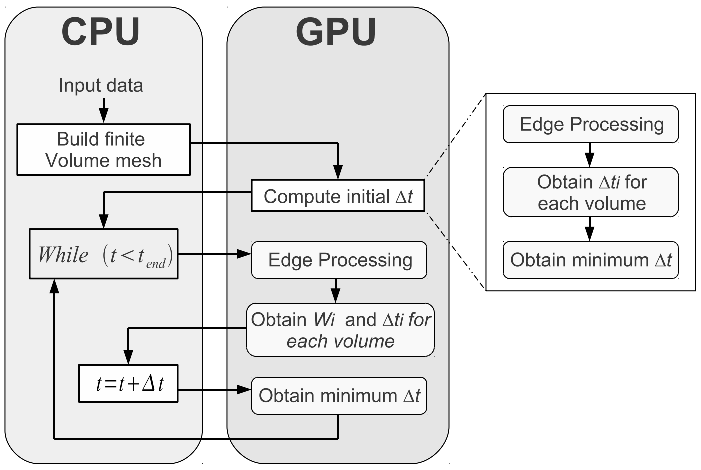


Figure 2: Structure of the CUDA Implementation for all the schemes

the normals, and another array of `int4` elements for storing, for each edge, the positions of the neighboring volumes in the volume textures and the positions of the two accumulators where the edge must write its contributions to the state of the neighboring volumes.

- Edge Processing: In this step each thread represents an edge $\Gamma_{ij}, \quad i,j \in 1,\dots,L$, and computes the contribution $F_{ij}^{\pm}$ of the edge to their adjacent volumes V_i and V_j. This is the most costly computation phase and the particular calculations performed in this step depends on the particular numerical scheme.

The threads contribute to a particular cell by means of six accumulators, each one being an array of L `float4` elements stored in global memory (see [7] for more details). The ordering of the elements at each accumulator matches with the ordering of the elements in the volume data array.

- Compute W_i and Δt_i for each volume: In this step, each thread represents a volume and computes the local Δt_i of the volume V_i in accordance with (3) and also updates the state W_i of that volume in accordance with (2) (see [7] for more details).

Since the 1D textures containing the volume data are stored in linear memory, we update the textures by writing directly into them.

- Obtain minimum Δt: This phase finds the minimum of the local Δt_i of the volumes by applying the most optimized kernel of the reduction sample included in the CUDA Software Development Kit [14]. The global step size Δt obtained will be used in the next iteration.

4 Reordering techniques based on bandwidth reduction

Many algorithms to reduce the bandwidth of a sparse symmetric matrix have been published in the literature. The Cuthill-McKee [8] and the Gibbs, Poole and Stockmeyer (GPS) [11] algorithms are two of the most popular. The Reverse Cuthill-McKee algorithm (RCM) [10] is a modification of the original algorithm where the resulting index numbers are reversed. RCM algorithm usually generates a more reduced profile than the original algorithm and consequently is most widely used.

Note that the application of a bandwitdh reduction algorithm to reorder the elements of the volume arrays enables adjacent volumes in the mesh to be in closer positions in the arrays which store the volume data. As a consequence, these algorithms can improve the data locality and enable an optimization of the cache usage. In this work, we will analyze the impact of ordering the mesh cells according to RCM and GPS algorithms.

5 Numerical Experiments

In this section we will study how the ordering of the volumes in the arrays which store the volume data affects the GPU execution times obtained with different solvers. We will consider two test problems:

Test 1 This test consists in an internal circular dambreak problem in the $[-5,5]\times[-5,5]$ domain. Depth is given by $H(x,y)=6$ and the initial condition is:

$$W_i^0(x,y)=\big(h_1(x,y),\quad 0,\quad 0,\quad h_2(x,y),\quad 0,\quad 0\big)^{\mathrm{T}},\qquad \text{where}$$

$$h_1(x,y)=\begin{cases}4.0 & \text{if } \sqrt{x^2+y^2}>1.5\\ 0.5 & \text{otherwise}\end{cases},\qquad h_2(x,y)=H-h_1(x,y)$$

The ratio of densities is $r=0.5$ and CFL paramenter is $\gamma=0.9$.

Test 2 This test represents two unstable water layers in the $[-5,5]\times[-5,5]$ domain. Depth function is $H(x,y)=1-1.5\cdot e^{-x^2-y^2}$ and the initial state is:

$$W_i^0(x,y)=\big(h_1(x,y),\quad 0,\quad 0,\quad h_2(x,y),\quad 0,\quad 0\big)^{\mathrm{T}},\qquad \text{where}$$

$$h_1(x,y)=\begin{cases}4.0 & \text{if } x\geq 0\\ 0.5 & \text{otherwise}\end{cases},\quad h_2(x,y)=\begin{cases}0.5 & \text{if } x\geq 0\\ 4.0 & \text{otherwise}\end{cases}$$

The ratio of densities is $r=0.98$ and CFL parameter is $\gamma=0.9$.

For both test problems, the simulation time is 0.1 seconds and wall boundary conditions ($q_1 \cdot \boldsymbol{\eta} = 0,\ q_2 \cdot \boldsymbol{\eta} = 0$) are considered. All the CUDA programs have been executed on a Intel Xeon server with 8 GB RAM containing a GeForce GTX 580 card and the GNU compiler has been used to derive the executables. We assign a size of 48 KB to L1 cache and 16 KB to shared memory in all the kernels excepting the kernel used to obtain the minimum Δt. The edge processing kernel has been executed using a one dimensional grid of blocks with a blocksize equals to 64 threads.

	Classical Roe			IR-Roe			PVM-IFCP		
Volumes	MATLAB	RCM	GPS	MATLAB	RCM	GPS	MATLAB	RCM	GPS
4000	0.069	0.072	0.071	0.020	0.022	0.021	0.0048	0.0050	0.0050
16000	0.44	0.38	0.43	0.092	0.081	0.084	0.020	0.024	0.023
64000	3.13	2.36	2.71	0.65	0.51	0.57	0.14	0.15	0.15
256000	22.91	15.12	17.04	4.68	3.26	3.70	1.01	1.00	1.04
1024000	167.2	99.98	108.9	33.97	21.34	23.70	7.81	7.47	7.69
2080560	625.6	384.6	480.0	127.1	78.58	94.61	29.37	27.98	28.74

Table 1: Execution times in seconds for test 1 before and after applying the RCM and GPS algorithms using a GeForce GTX 580.

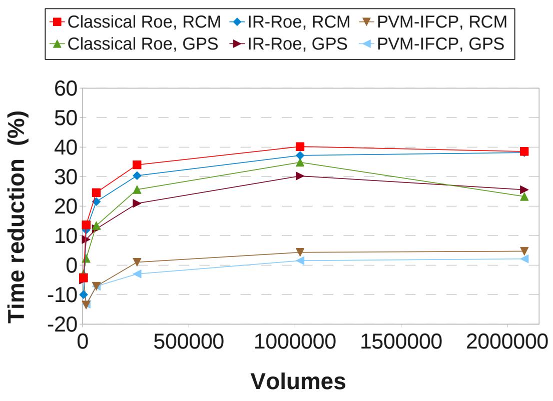
	Classical Roe			IR-Roe			PVM-IFCP		
Volumes	MATLAB	RCM	GPS	MATLAB	RCM	GPS	MATLAB	RCM	GPS
4000	0.059	0.061	0.061	0.022	0.021	0.021	0.0042	0.0045	0.0044
16000	0.37	0.34	0.37	0.11	0.089	0.098	0.017	0.020	0.020
64000	2.71	2.19	2.44	0.79	0.53	0.59	0.11	0.12	0.12
256000	20.97	14.80	16.46	6.06	3.31	3.82	0.85	0.86	0.88
1024000	166.9	103.5	113.4	46.07	21.35	24.48	6.73	6.37	6.56
2080560	623.8	395.7	502.2	169.2	75.40	100.3	25.15	23.40	24.93

Table 2: Execution times in seconds for test 2 before and after applying the RCM and GPS algorithms using a GeForce GTX 580.

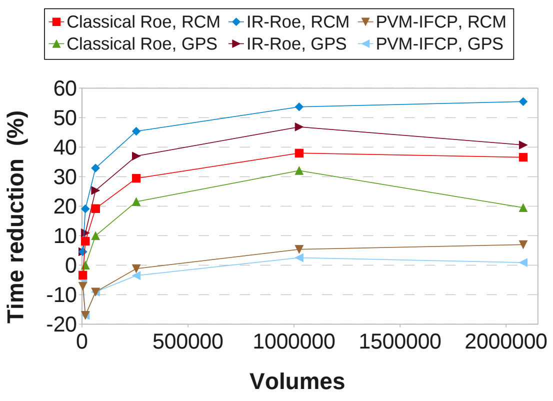
We have generated several triangular meshes using the Partial Differential Equation Toolbox for MATLAB [13]. In order to compare the different orderings of the volume arrays, we will execute the single precision CUDA programs of the classical Roe, IR-Roe and PVM-IFCP schemes using the two former test problems and three different volume orderings: the original provided by MATLAB and the resulting of applying the RCM and the GPS algorithms to the original meshes.

The `symrcm` MATLAB function and the Fortran code given by the 582 TOMS algorithm [12] have been used to apply the RCM and GPS algoritms, respectively.

For all the volume orderings, Tables 1 and 2 show the GPU runtimes in seconds for tests 1 and 2, respectively, Figure 3 shows the time reduction obtained with all the numerical schemes and test problems, Figure 4 depicts graphically the adjacency matrices of the 4000



(a)



(b)

Figure 3: GPU time reduction obtained with respect to the original meshes provided with MATLAB: (a) Test 1; (b) Test 2.

Number of volumes	MATLAB	RCM	GPS
4000	3786	73	62
16000	12000	147	126
64000	48000	294	249
256000	192000	598	500
1024000	768000	1206	1004
2080560	1996159	1728	1378

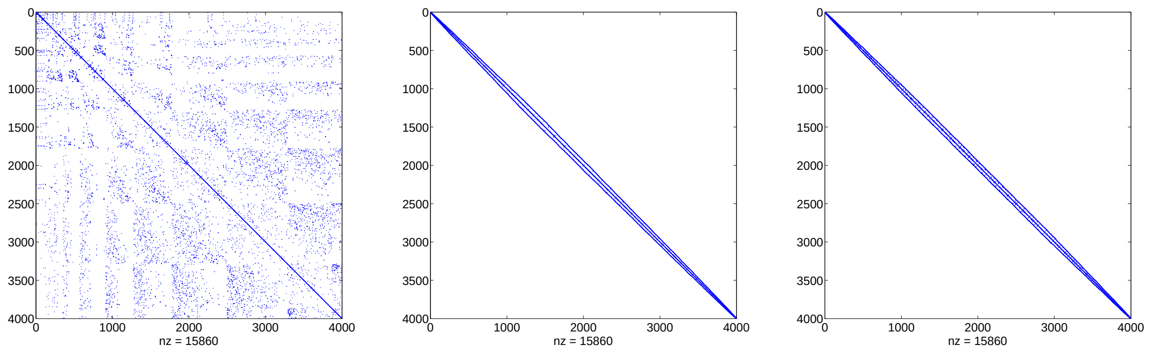
Table 3: Bandwidth of the meshes before and after applying RCM and GPS algorithms.

volumes mesh, and Table 3 shows the bandwidth of the adjacency matrices of all meshes.

We can see that, for the biggest meshes, the RCM ordering has given the best runtimes in all cases. Specifically, with the classical Roe and IR-Roe schemes, execution times have reduced approximately between 37 and 55 % for the meshes with more than one million volumes in both test problems, whereas using the PVM-IFCP scheme the runtimes have reduced the 5 %. The GPS ordering, although providing a lower bandwidth than RCM, has given worse execution times than RCM for the biggest meshes in all cases.

6 Conclusions

The optimization of finite volume shallow water solver for unstructured meshes running on GPUs has been dealt. The reordering of the volumes in the arrays which store the volume



(a) (b) (c)

Figure 4: Adjacency matrices for the mesh with 4000 volumes before and after applying RCM and GPS algorithms. `nz` denotes the number of non-zero elements: (a) original mesh obtained with MATLAB; (b) after applying RCM; (c) after applying GPS.

data in GPU by using bandwidth reduction techniques makes it possible to reduce substantially the execution times obtained because the L1 and texture cache usage is optimized. The highest reduction has been achieved achieved when the reverse Cuthill-McKee ordering is applied to these arrays and we obtain an improvement of 5 to 55% approximately in the GPU simulation times (depending on the numerical scheme) with respect to the ordering provided when the volume mesh is generated, by the MATLAB PDE toolkit. The impact seems to be higher when the numerical intensity of the solver and the problem size grow.

Acknowledgements

This research has been partially supported by the Spanish Government Research projects MTM09-11923, MTM2009-07719, and MTM2011-27739-C04-02.

References

[1] M. de la Asunción, J. M. Mantas and M. J. Castro, *Programming CUDA-based GPUs to simulate two-layer shallow water flows*, Euro-Par 2010 Ischia (Italy) (2010).

[2] M. de la Asunción, José M. Mantas, M. J. Castro, E.D. Fernández-Nieto, *An MPI-CUDA implementation of an improved Roe method for two-layer shallow water systems*, Journal of Parallel and Distributed Computing **in press** DOI: http://dx.doi.org/10.1016/j.bbr.2011.03.031 (2011).

[3] A. R. Brodtkorb, M. L. Sætra, and M. Altinakar, *Efficient Shallow Water Simulations on GPUs: Implementation, Visualization, Verification, and Validation*, Computers & Fluids **55** (2011) 1–12.

[4] D.A. Burgess and M.B. Giles, *Renumbering unstructured grids to improve the performance of codes on hierarchical memory machines*, Adv. in Eng. Software **28 (3)** (1997) 189–201.

[5] M. J. Castro, J. A. García-Rodríguez, J. M. González-Vida and C. Parés, *A parallel 2D finite volume scheme for solving systems of balance laws with nonconservative products: Application to shallow flows*, Comput. Meth. Appl. Mech. Eng. **195** (2006) 2788–2815.

[6] M. J. Castro and E. D. Fernández-Nieto, *A class of computationally fast first order finite volume solvers: PVM methods*, Submitted to SIAM J. of Sci. Computing.

[7] M. J. Castro, S. Ortega, M. de la Asunción, J. M. Mantas and J. M. Gallardo, *GPU computing for shallow water flow simulation based on finite volume schemes*, Comptes Rendus Mécanique **339** (2011) 165–184.

[8] E. Cuthill and J. McKee, *Reducing the bandwidth of sparse symmetric matrices*, Proc. of the 24th Nat. Conf. ACM (1969), 157–172.

[9] E. D. Fernández-Nieto, M. J. Castro and C. Parés, *On an intermediate field capturing Riemann solver based on a parabolic viscosity matrix for the two-layer shallow water system*, Journal of Scientific Computing **48** (2011) 117–140.

[10] J. A. George and J. W-H. Liu, *Computer Solution of Large Sparse Positive Definite Systems*, Prentice-Hall, 1981.

[11] N. E. Gibbs, W. G. Poole and P. K. Stockmeyer, *An algorithm for reducing the bandwidth and profile of a sparse matrix*, SIAM J. on Num. Anal. **13** (1976) 236–250.

[12] J. G. Lewis, *Algorithm 582: The Gibbs-Poole-Stockmeyer and Gibbs-King Algorithms for Reordering Sparse Matrices*, ACM Trans. Math. Softw. **8 (2)** (1982) 190–194.

[13] MathWorks, MATLAB R2011b, http://www.mathworks.com/products/matlab.

[14] NVIDIA Corporation, *CUDA Zone*, http://www.nvidia.com/object/cuda_home_new.html.

[15] NVIDIA Corporation, *CUDA C Best Practices Guide 4.1*, (2012).

[16] J.D. Owens, M. Houston, D. Luebke, S. Green, J.E. Stone and J.C. Phillips, *GPU Computing*, Proceedings of the IEEE **96** (2008) 879–899.