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

Numerical simulation of pollutant transport in a shallow water system on the Cell heterogeneous processor

Carlos H. González · Basilio B. Fraguela · Diego Andrade · José A. García · Manuel J. Castro

the date of receipt and acceptance should be inserted later

Abstract This paper presents an implementation, optimized for the Cell processor, of a finite volume numerical scheme for 2D shallow water systems with pollutant transport. A description of the special architecture and programming required by the Cell processor motivates the methodology to develop optimized implementations for this platform. This process involves parallelization, data structure reorganization, explicit transfers of data and computation vectorization. Our implementation, tested using a realistic problem, achieves very good speedups with respect to the sequential execution on a standard CPU.

Keywords High Performance Computing, Finite Volume Method, Vectorization, Parallelism, Heterogeneous Architectures

1 Introduction

This paper addresses the optimized simulation on the Cell processor [12] of the transport of an inert pollutant on a one-layer homogeneous fluid governed by a shallow-water system. More precisely, the fluid is modeled by a system of shallow water equations in two dimensional domains and the pollutant transport is modeled by a transport equation. These coupled equations constitute a hyperbolic system of conservation laws with source terms, that can be discretized using finite volume schemes [15].

Carlos H. González · Basilio B. Fraguela · Diego Andrade
Computer Architecture Group, Electronics and Systems Dept., Univ. of Coruña, Spain
E-mail: {cgonzalezv, basilio.fraguela, diego.andrade}@udc.es

José A. García
Applied Mathematics Area, Mathematics Dept., Univ. of Coruña, Spain
E-mail: jagrodriguez@udc.es

Manuel J. Castro
Mathematical Analysis Dept., Univ. of Málaga, Spain
E-mail: castro@anamat.cie.uma.es

Finite volume schemes solve the integral form of the shallow-water equations in computational cells. Therefore, mass and momentum are conserved in each cell, even in the presence of flow discontinuities. Numerical finite volume schemes, for solving the shallow water equations have been developed in many works [4] [1] [13] [14] [17] [18]. The pollutant transport problem in the context of shallow water systems has been studied in [2][5][3][21].

These models have many applications to geophysical flows, being of interest for the study of rivers, channels, ocean currents, estuarine systems, or dambreak problems, for example. The simulations of these problems have very large computing requirements which grow with the size of the space and time dimensions of the domain. This has given place to the development of parallel implementations of numerical schemes to simulate shallow-water systems in reasonable times mainly in clusters of PCs [8] and GPUs [6][9][16][20]. There are few studies however of shallow-water system implementations [11][19] on the Cell heterogeneous processor [12], a distinctive architecture that requires special programming techniques to be fully exploited, different from those used in GPUs or homogeneous multicore architectures. Also, none of these works considers a pollutant transport model with wet-dry fronts. This allows to present an application using real topography data of a complex sea region. This way, this paper is the first one to study the optimized implementation of such model on the Cell architecture as far as we know. Other important differences are that [19] is written on a high-level library instead of at low level, which can miss important optimizations, and that [11] uses an standard double buffering approach that allows to eliminate most memory access delays, while we implemented a more ambitious multibuffering, discussed in Section 3.3 that completely avoided them. With the optimal use of the resources available in the Cell processors of a BladeCenter QS20, our implementation of the algorithm runs 46 times faster in this system than in our baseline general-purpose CPU.

The rest of this paper is organized as follows. Section 2 describes the physical model and the numerical scheme. Section 3 overviews the Cell architecture and describes the optimized implementation of our model on it, followed by a validation and a performance evaluation in Section 4. Finally, Section 5 presents our conclusions and ideas for future work.

2 Mathematical model and numerical scheme

2.1 Coupled model: 2d shallow water equations with pollutant transport

A pollutant transport model consists in the coupling of a fluid model and a transport equation. In this work, to model the fluid dynamics we consider the two dimensional one layer shallow water equations. The pollutant transport evolution is modelled by a transport equation. These four coupled equations form a system that can be written as a *system of conservation laws with source terms* in the following compact form:

$$\frac{\partial W}{\partial t}(\boldsymbol{x},t) + \operatorname{div} \boldsymbol{F}(W) = \boldsymbol{S}(W) \cdot \nabla H(\boldsymbol{x}) + \boldsymbol{S_f}, \tag{1}$$

where $W = \left[h,\, q_x,\, q_y,\, hC \right]^T$, is the vector of unknowns. The unknowns of the problem are: the vertically averaged height of the water column $h(\boldsymbol{x},t)$; the flux $\boldsymbol{q}(\boldsymbol{x},t) = (q_x(\boldsymbol{x},t), q_y(\boldsymbol{x},t)) = h(\boldsymbol{x},t)\cdot\boldsymbol{x}(\boldsymbol{x},t)$, where $\boldsymbol{u}(\boldsymbol{x},t) = (u_x(\boldsymbol{x},t), u_y(\boldsymbol{x},t))$ is the vertical averaged velocity of the fluid, at each point $\boldsymbol{x} = (x,y)$ of the computational domain and at time t; and the pollutant concentration $C(\boldsymbol{x},t)$, with $h(\boldsymbol{x},t)C(\boldsymbol{x},t)$ the amount of pollutant dissolved in the fluid. $H(\boldsymbol{x})$ is a data: the function that describes the bottom bathimetry, measured from a fixed reference level and g is the gravitational constant.

In equation (1) $\boldsymbol{F} = (F_1, F_2)$ is the flux function given by:

$$F_1(W) = \left[q_x,\, \frac{q_x^2}{h} + \frac{1}{2}gh^2,\, \frac{q_x q_y}{h},\, q_x C \right]^T,\ F_2(W) = \left[q_y,\, \frac{q_x q_y}{h},\, \frac{q_y^2}{h} + \frac{1}{2}gh^2,\, q_y C \right]^T.$$

The source term due to the bathimetry is $\boldsymbol{S}(W) = (S_1(W), S_2(W))$ with

$$S_1(W) = \left[0,\, gh,\, 0,\, 0 \right]^T, \quad S_2(W) = \left[0,\, 0,\, gh,\, 0 \right]^T \tag{2}$$

and S_f is the source term due to friction forces, given by a Manning Law

$$\boldsymbol{S_f} = \left[0,\, ghS_{f,x},\, ghS_{f,y},\, 0 \right]^T,\ S_{f,\alpha} = n^2 \frac{u_\alpha \|\boldsymbol{u}\|}{h^{1/3}},\ \alpha = x,y\ (n,\ \text{roughness coefficient}). \tag{3}$$

Given an unitary vector $\boldsymbol{\eta} = (\eta_x, \eta_y)$, we define the matrix

$$A(W,\boldsymbol{\eta}) = A_1(W)\eta_x + A_2(W)\eta_y,$$

where $A_i(W) = \frac{\partial F_i}{\partial W}(W)$ is the jacobian matrix of $F_i(W)$, $i = 1,2$. $A(W,\boldsymbol{\eta})$ is diagonalizable, with eigenvalues $\lambda_1 = \lambda_4 = \boldsymbol{u}\cdot\boldsymbol{\eta},\ \lambda_2 = \boldsymbol{u}\cdot\boldsymbol{\eta} - \sqrt{gh},\ \lambda_3 = \boldsymbol{u}\cdot\boldsymbol{\eta} + \sqrt{gh}$, and thus, if $h(\boldsymbol{x},t) > 0$, the system (1) is hyperbolic.

2.2 Finite volume numerical scheme

Therefore, to discretize system (1) we use a finite volume scheme. More details can be found in [7][10]. Let us remark that the term $\boldsymbol{S_f}$ is discretized in a semi-implicit way as detailed in [10], thus in what follows we focus in the discretization of system (1) where $\boldsymbol{S_f}$ is supposed to be zero.

To discretize the system (1), we split the computational domain in cells or control volumes, $V_i \subset \mathbb{R}^2,\ i = 1,\ldots,L$. In our case we will consider a structured mesh given by squares. We will use the following notation: given a finite volume V_i, $\boldsymbol{x}_i$ is its center, $|V_i|$ its area, $\mathcal{N}_i$ is the set of indexes j such that V_j is the neighbor of V_i, E_{ij} is the edge shared by two neighbor cells V_i and V_j and $|E_{ij}|$ its length and $\boldsymbol{\eta}_{ij} = (\eta_{ij,x}, \eta_{ij,y})$ is the unitary vectorial normal to edge E_{ij} and that points towards the cell V_j. Finally, we call V_{ij} the triangular subcell with one edge given by E_{ij} and the opposite vertex given by $\boldsymbol{x}_i$. In finite volume schemes, constant approximations of the solution at each of the previous cells are computed. More precisely, if $W(\boldsymbol{x},t)$ is the exact solution at point $\boldsymbol{x}$ and at time t, we will denote by ${W_i}^n$ an approximation of the average of the solution on the volume V_i at time t^n, ${W_i}^n \simeq 1/|V_i| \int_{V_i} W(\boldsymbol{x},t^n)d\boldsymbol{x}$.

Integrating the equation (1) over each finite volume V_i, dividing by $|V_i|$ and applying the Divergence Theorem:

$$\frac{\partial}{\partial t}\left(\frac{1}{|V_i|}\int_{V_i} W(\boldsymbol{x},t)\,dV\right) = -\frac{1}{|V_i|}\Big(\sum_{j\in\mathcal{N}_i}\int_{E_{ij}} \mathbf{F}(W)\cdot\boldsymbol{\eta}_{ij}\,d\gamma - \int_{V_i}\boldsymbol{S}(W)\cdot\nabla H(\boldsymbol{x})\,dV\Big). \tag{4}$$

To discretize equation (4) we will use the finite volume numerical scheme presented in [10]: once the approximation of W_i is known at time t^n, W_i^n, the approximation at time t^{n+1} is given by:

$$W_i^{n+1} = W_i^n - \frac{\Delta t}{|V_i|}\sum_{j\in\mathcal{N}_i}|E_{ij}|\mathscr{F}_{ij}^{-}(W_i^n, W_j^n, \boldsymbol{\eta}_{ij}), \tag{5}$$

with

$$\begin{aligned}\mathscr{F}_{ij}^{-}(W_i^n, W_j^n, \boldsymbol{\eta}_{ij}) = \mathscr{P}_{ij}^{n,-}\Big(\boldsymbol{F}(W_j^n)\cdot\boldsymbol{\eta}_{ij} - \boldsymbol{F}(W_i^n)\cdot\boldsymbol{\eta}_{ij} - \boldsymbol{S}_{ij}^n\Big)\\ - \frac{\boldsymbol{F}(W_j^n)\cdot\boldsymbol{\eta}_{ij} - \boldsymbol{F}(W_i^n)\cdot\boldsymbol{\eta}_{ij}}{2} + \boldsymbol{F}_\alpha(W_i^n, W_j^n, \boldsymbol{\eta}_{ij}) - \boldsymbol{S}_{\alpha,ij}^n,\end{aligned} \tag{6}$$

where the projection matrix, $\mathscr{P}_{ij}^{n,-}$, is given by:

$$\mathscr{P}_{ij}^{n,-} = \frac{1}{2}K_{ij}^n\Big(I - \operatorname{sgn}(D_{ij}^n)\Big)(K_{ij}^n)^{-1}, \tag{7}$$

being I is the identity matrix and K_{ij}^n the matrix whose columns are the eigenvectors related to the Roe matrix A_{ij}^n given by

$$A_{ij}^n = A(W_{ij}^n, \boldsymbol{\eta}_{ij}) = A_1(W_{ij}^n)\eta_{ij,x} + A_2(W_{ij}^n)\eta_{ij,y}, \tag{8}$$

where

$$W_{ij}^n = \big[\, h_{ij}^n,\, h_{ij}^n u_{ij,x}^n,\, h_{ij}^n u_{ij,y}^n,\, h_{ij}^n C_{ij}^n \,\big]^T, \tag{9}$$

is the intermediate Roe's state, satisfying $\boldsymbol{F}(W_j^n)\cdot\boldsymbol{\eta}_{ij} - \boldsymbol{F}(W_i^n)\cdot\boldsymbol{\eta}_{ij} = A_{ij}^n(W_j^n - W_i^n)$.

D_{ij}^n the diagonal matrix whose elements are the eigenvalues of A_{ij}^n, and sgn D_{ij}^n is the diagonal matrix that results from taking the sign of the elements of D_{ij}^n. The term $\boldsymbol{S}_{ij}^n$ is given by

$$\boldsymbol{S}_{ij}^n = \big[\, 0,\, gh_{ij}^n(H_j - H_i)\eta_{ij,x},\, gh_{ij}^n(H_j - H_i)\eta_{ij,y},\, 0 \,\big]^T. \tag{10}$$

$$\boldsymbol{F}_\alpha(W_i^n, W_j^n, \boldsymbol{\eta}_{ij}) = \frac{\boldsymbol{F}(W_{(1-\alpha)i+\alpha j})\cdot\boldsymbol{\eta}_{ij} + \boldsymbol{F}(W_{\alpha i+(1-\alpha)j})\cdot\boldsymbol{\eta}_{ij}}{2}, \tag{11}$$

where we denote $W_{(1-\alpha)i+\alpha_j} = (1-\alpha)W_i^n + \alpha W_j^n$, with $\alpha\in[0,1]$, a convex combination of W_i^n and W_j^n, and finally,

$$\boldsymbol{S}^n_{\alpha,ij} = \begin{bmatrix} 0 \\ \frac{g}{2}\left(\frac{h_{(1-\alpha)i+\alpha j}+h_i^n}{2}(H_{(1-\alpha)i+\alpha j}-H_i)+\frac{h_{\alpha i+(1-\alpha)j}+h_j^n}{2}(H_{\alpha i+(1-\alpha)j}-H_j)\right)\eta_{ij,x} \\ \frac{g}{2}\left(\frac{h_{(1-\alpha)i+\alpha j}+h_i^n}{2}(H_{(1-\alpha)i+\alpha j}-H_i)+\frac{h_{\alpha i+(1-\alpha)j}+h_j^n}{2}(H_{\alpha i+(1-\alpha)j}-H_j)\right)\eta_{ij,y} \\ 0 \end{bmatrix}, \quad (12)$$

where $H_{\alpha i+(1-\alpha)j} = \alpha H_i + (1-\alpha)H_j$, is again a convex combination of H_i and H_j.

The expressions (11) and (12) are used to avoid entropy corrections needed by Roe scheme in critical points [10]. The authors propose different values of the parameter α. In practice, the value $\alpha = 1/8$ gives good results [10], so here we take $\alpha = 1/8$. Note that in the case $\alpha = 0$ we obtain the usual Roe Scheme [10].

The previous numerical scheme is exactly well-balanced for the stationary solution corresponding to water at rest (see [10]) and linearly L^∞ under the usual CFL condition:

$$\Delta t = \min_{i=1,\ldots,L}\left\{\frac{\sum_{j\in\mathcal{N}_i}|E_{ij}|\,\|D^n_{ij}\|_\infty}{2\gamma|V_i|}\right\}$$

where γ, $0<\gamma\leq 1$, is the CFL parameter and $\|D^n_{ij}\|_\infty$ the infinite norm of the matrix D^n_{ij}, that is, the maximum eigenvalue of the matrix A^n_{ij}.

The resulting time step can be tiny, what gives raise to a great number of time iterations for simulations that happen in big time scales, which is the case of geophysical flows. Thus from the computational point of view, the solution of the problem is reduced to a huge number of matrix operations and vectors of size 4×4.

Remark (Wet-dry fronts): It must be remarked that this numerical scheme corresponds to the case where the fluid occupies the whole domain. If this numerical scheme is applied without any modification to a case with dry-wet fronts (situations with emerging bottom), the obtained results have no physical meaning. In those cases it is necessary to modify the scheme, as is proposed in [10]. Let us remark that to avoid errors due to working with single precision, the velocities and concentration are desingularized following the technique suggested in [14], and taking $\varepsilon = 10^{-6}$ the simple precision limit. Also, if the thickness of the layer of fluid becomes tiny at both cells V_i and V_j, then the fourth component of the numerical flux $\mathscr{F}^-_{ij}(W^n_i, W^n_j, \boldsymbol{\eta}_{ij})$ is defined as follows:

$$\mathscr{F}^-_{ij\,[4]} = \begin{cases} \mathscr{F}^-_{ij\,[1]}\cdot C_j & \text{if } \boldsymbol{u}_{ij}\cdot\boldsymbol{\eta}_{ij} < 0, \\ \mathscr{F}^-_{ij\,[1]}\cdot C_i & \text{if } \boldsymbol{u}_{ij}\cdot\boldsymbol{\eta}_{ij} > 0, \end{cases}$$

where $\mathscr{F}^-_{ij\,[l]}$, denotes the l-th component of the vector $\mathscr{F}^-_{ij}$.

3 Optimized implementation on the Cell Broadband Engine

The Cell Broadband Engine (CBE) Architecture is an heterogeneous design characterized by the presence of two kinds of cores in the chip, the PowerPC Processing

Unit (PPU) and the Synergistic Processing Units (SPUs). The PPU is a a general-purpose PowerPC core who is in charge of the task management in the chip. Most computational tasks should be run in separate threads in the eight SPUs available in the processor, which are computing cores based on SIMD instructions attached to 256KB of software managed local memory. An SPU can only access directly its local memory. Thus programmers must explicitly code transfers between this memory and main memory by means of DMA operations, which results in high performance in exchange of a harder programming.

We programmed the system at very low level using the IBM CellSDK in order to obtain optimal performance. This way, process creation and assignment to SPUs was done via pthreads and libspe2, which implements basic task control routines. Compiler intrinsics were used for operation vectorization. Regarding communications and data transfers, they can be done in two ways: (1) DMA commands can be used to move data between main memory and the local stores, or between different local stores. Memory addresses must be 16 byte aligned, and preferably 128 byte aligned. (2) Using mailboxes, which are 32 bits registers exposed by each SPU and addressable by the PPU. They are mainly used for process synchronization and very small data communications between the PPU and the SPUs.

The Cell version of the algorithm was derived from an implementation [8] for x86 processors. The adaptation to the Cell architecture requires several stages which may also be applied to obtain Cell implementations in other areas of scientific computing:

- Parallelism identification: The algorithm is analyzed and the hotspots where the application consumes most of the time are identified. The execution time of these hotspots is reduced by extracting loop, task and/or data-parallelism. Thus, the code of these hotspots is modified to be executed by the SPUs. The PPU executes the rest of the code and coordinates the work of the SPUs.
- Work distribution: As a consequence of the previous stage, the computation is distributed among the PPU and SPUs. The data and code handled by each SPU must fit in the constrained space of its local storage (256 KB). Small data structures which are frequently used by the SPU reside permanently in the local storage while large or less frequently used data structures are cached or streamed in or out the local storage. In addition, the base addresses, the size and other characteristics of the data structures must be adapted to optimize DMA transfers.
- Code optimizations: The processing performed by the PPU and each SPU is optimized using the vector instructions available in these units and minimizing the overhead imposed by the data interchanges. Recursive functions must be rewritten as iterative functions, the number of branches must be minimized to the extent possible and basic-loop optimizations (e.g. loop unrolling) must be applied.

The application of each stage to our application is now described in turn.

3.1 Parallelism identification

Figure 1 shows a flowchart of the algorithm. First in $\{1\}$, the domain of the simulated problem is decomposed into finite volumes. Then a loop iterates on time, from the

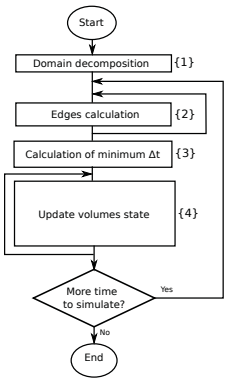

Fig. 1 Algorithm flowchart

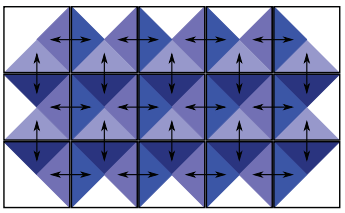

Fig. 2 Structured mesh

initial to the final time configured in the problem. Each iteration of this loop is a step of the simulation, and does three steps: in $\{2\}$, the flux between two volumes that passes through each edge is calculated. After these calculations each volume has a particular Δt given by the stability condition, and in $\{3\}$ the minimum of them is obtained. This value is used to advance the simulation to the next step in $\{4\}$, updating the values of the variables of the volumes.

In the Cell architecture, the domain is decomposed into disjoint subsets, and the processing of each subset is assigned to one different SPU. Thus, the algorithm contains a lot of data-parallelism in these steps that can be exploited using the SPUs. The reduction of the minimum time step is coordinated by the PPU, which provides the global minimum time step to each SPU to transit its volumes to the next state.

3.2 Work distribution

Before the work is distributed among the SPUs, the data structures used must be adapted to improve the performance in the Cell architecture. A general implementation of a finite volume method solver stores the data of volumes, edges and their connectivity in a graph in order to support unstructured grids. But most problems can be represented using a structured grid of rectangular volumes of the same size, which can be stored in a two dimensional matrix, and neighbors are obtained just by looking at the adjacent columns and rows. This structure has also lower memory requirements and, in the case of the Cell, eases the transfer of blocks of volumes that fit in the 256KB local memory of the SPUs. This can be done through DMA transfers of big consecutive data sets, which is more efficient than smaller transfers required to work with individual volumes in non-structured meshes. Data structures are forced to start at 128-byte boundaries and their size is rounded up to a multiple of 128 bytes in order to optimize DMA transfers.

Figure 2 shows an example of a structured mesh. Each cell of the matrix is a volume and the arrows represent the liquid flowing between the volumes. Step 2 of Figure 1 is done iterating in this volume array. For each volume, the flow between it and its left and bottom neighbours is computed.

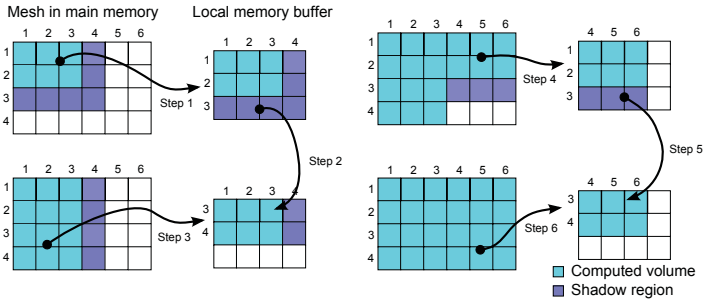

Fig. 3 Transfers using shadow regions (darker volumes)

Data-parallelism is exploited by executing the algorithm on a different set of volumes in each SPU. The matrix of volumes is distributed among the SPUs in blocks of columns. With this distribution, the volumes of the first and the last columns of the block assigned to a SPU miss the information about their left and right neighbors, respectively. This is solved using shadow regions in each SPU consisting of the rightmost column of the block to the left and the leftmost column of the block to the right. The usage of shadow regions implies that some calculations are duplicated, but its impact on the performance is only 0.21% of the execution time.

The code and the data size cannot exceed the 256 KB of local storage available in each SPU. This way, each SPU has to fragment its block of columns in smaller sub-blocks that fit in this space. The shadow region of each sub-block includes the volumes of the top and bottom neighboring rows, besides the volumes of the left and the right neighboring columns. The blocks are transferred to the SPU by rows, whose size is adjusted to fit the maximum limit of 16 KB transferrable per DMA request.

Figure 3 shows how these transfers are done step by step. Each sub-block is read adding a shadow region that includes the column to the right and the row to the bottom, as shown in steps 1 and 4. The exceptions are obviously the last subblocks of each group of columns (steps 3 and 6), where there are no bottom rows. The values computed for the volumes of the shadow regions are incomplete. Their final values will be calculated after the processing of the next sub-blocks. The shadow row of one sub-block is not sent back to the main memory when the processing of the sub-block finishes. Rather, it is relocated at the beginning of the buffer, as can be seen in steps 2 and 5 in the figure, so it can be reused during the processing of the next sub-block.

Communications between SPUs are avoided, which simplifies the program and minimizes the overheads. In order to compute the minimum Δt, each SPU computes it local minimum, and stores it in its mailbox. The PPU reads all the mailboxes, reduces their values, and puts the result in the input mailboxes of each SPU.

3.3 Code optimizations

Once the tasks have been distributed between the PPU and the SPUs, three aspects of the code must be improved: (1) different optimizations must be applied to the code executed in the SPU such as loop unrolling or the minimization of the number of

branches, (2) the SIMD instructions available in the Cell cores must be used, and (3) the overhead due to the data transfers between main memory and the SPUs local memory must be minimized. These aspects are now explained in turn.

Loop optimizations: Several optimizations have been applied to the code that will be executed on each SPU. First, the code size is important because data, code and stack must fit in the 256KB available on each SPU. Thus, the code has been modified looking for a balance between code size and performance. The reason is that there are optimization techniques that increase the code size. An example is loop unrolling, which is applied to facilitate vectorization. Besides, the operations that are the core of the algorithm are implemented using as few conditional instructions as possible. The reason is that mispredicted branches incur in a heavy performance penalty in SPUs, while their branch predictor is very simple, as it simply assumes that branches are never taken. The best approach to remove branches in this architecture is to replace them with vector comparisons and logical operations.

Vectorization: The fundamental vector and matrix operations are implemented using the vector instructions available in the PPU and SPUs. These instructions work with 128 bits wide vector registers, which can thus store 4 single precision floating point numbers. In the simulation, each volume is characterized with 4 variables –depth, pollutant concentration, and flow on x and y axes–, which can be represented using 4x4 matrices and 4 dimensional vectors. This enables their associated operations to be executed four at once using SIMD instructions.

Data transfers overhead minimization: Data transfers between the SPU local storage and the main memory are done through asynchronous DMA transfers. The latency of these transfers can be hidden by overlapping them with the computation. The multi-buffering technique has been used to define several buffers in the local storage of each SPU. This way, some buffers are used to store data that is being transferred from or to main memory, while the SPU computes on data of other buffers. The number of buffers has been adjusted experimentally to allow full overlapping between computation and communications in each SPU. Namely, 10 buffers are used, and each one stores 256 volumes with 54 bytes each. This uses 135KB of local memory, so a bit below one half of the memory is left for the heap, stack and program code. Our measurements indicate that this multibuffered implementation completely removed the memory access delays from our overall computation runtime.

4 Results

The solver has been tested with an academic problem and a realistic problem in two systems. The x86 version was tested in a server with a Xeon E5440 processor at 2.83GHz and a 6MB L2 cache. The code was compiled using GCC 4.1.2. For the Cell implementation, a BladeCenter QS20, which provides two 3.2GHZ Cell processors with shared memory and 8 SPUs each, with GCC 4.1.1 was used. Exploiting the two Cell processors in this system led to adding a new level of parallelization by splitting

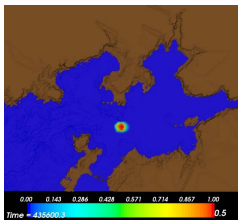

Fig. 4 Pollutant drop

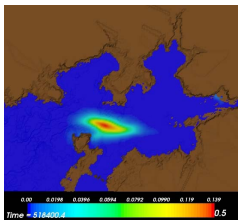

Fig. 5 Pollutant after 5 and 1/2 days

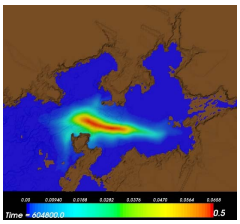

Fig. 6 Pollutant after the seventh day

the computations on them using pthreads. In both systems the compiler was invoked with the highest optimization level available.

The academic problem simulates the transport of a pollutant within a rectangular channel of 75×30 m. whose bottom has a bump. Free boundary conditions were imposed at the entrance and at the outlet of the channel and wall conditions were imposed on the sides. The problem was simulated with CFL=0.9 and a simulation time of 200 s. yielding the same results for five meshes with sizes from 9000 to 2304000 volumes. It was also used for checking the error of the usage of single precision computations in our implementation For this purpose we developed a double-precision version of our x86 code. The value of the L^1 norm for T = 100 seconds for the simulation using a mesh of 150×60 volumes was $2.7e-4$ for h, $3.72e-5$ for q_x and $4.01e-6$ for c. This way, the error measured for single-precision data is negligible.

The realistic problem simulates the transport of a pollutant with a radius of 400 m in a 94 km^2 area of an actual estuary called ría de Arousa with 7 days of simulated time. The bottom and left boundary conditions are fixed using the equation of the tide in that area, while wall conditions are used in the top and right borders. Figures 4 to 6 show the evolution of the simulation from the pollutant drop to the final snapshot.

Figure 7 reflects the execution times of the x86 version and the Cell implementation using a varying number of SPUs for several mesh sizes. For example, the runtime for the largest mesh decreases from 5.8×10^6 s. for the x86 version to 0.24×10^6 s. using the 8 SPUs in a Cell or 0.13×10^6 using all the resources in the QS20. The Cell achieves with 16 SPUs between 496.7 units of real time simulated per unit of simulation time for the smallest mesh tested, and 4.7 for the largest mesh tried. This way it can perform the simulations in real time with a high level of detail, even allowing to evaluate different scenarios in the event of a contaminant spill. The speedups of the Cell with respect to the x86 are shown in Figure 8. As can be seen, the Cell implementation using a single SPU is between 2 and 4 times faster than the x86, and its performance improves linearly with the number of SPUs, achieving a maximum speedup of 46x with respect to the x86 version. These results are possible thanks to the minimization of the communications, the high level of exploitation of the SPU SIMD units of our code, and the efficient application of cell-specific optimizations such as branch elimination and the overlapping of the computations with the data transfers between main memory and the local storage. Another important reason is that the Cell processor clock rate is 13.1% higher than that of the x86 system. Figure 9 shows the speedups for 2, 4, 8 and 16 SPUs with respect to using one SPU. These speedups are almost perfectly linear but for the smallest mesh on 16 SPUs, in

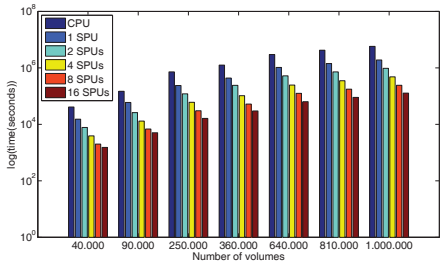

Fig. 7 Running times for the realistic problem in the Xeon and Cell processors

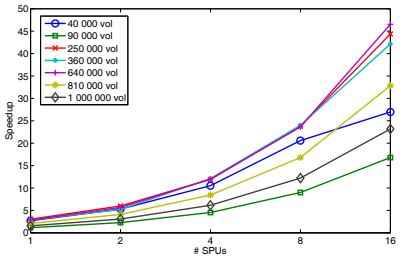

Fig. 8 Cell speedups using 1, 2, 4, 8, and 16 SPUs against Xeon for the realistic problem

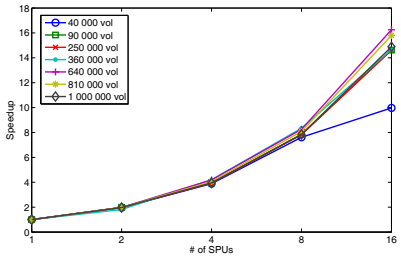

Fig. 9 Cell speedups using 1, 2, 4, 8, and 16 SPUs against 1 SPU for the realistic problem

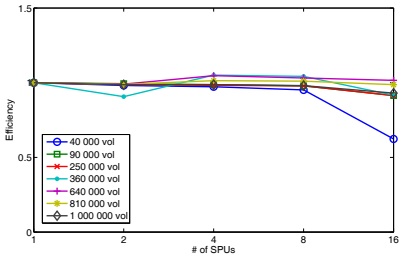

Fig. 10 Cell efficiencies using 1, 2, 4, 8, and 16 SPUs for the realistic problem

which the overhead of synchronizing the two Cell processors is large in relation to the reduced computing time in each time step for this problem size. This can be also seen in the efficiency graph for the runs with different numbers of SPUs with respect to the usage of a single one in Figure 10. Except for this point, the efficiency is very high thanks to the minimal parallelization overhead, with an average of 97.6%.

5 Conclusions

This work presents the implementation for the Cell architecture of a shallow water system model with transport of pollutants, which uses the finite volume method. The development followed well-defined steps that constitute a method which could be used to create applications for the Cell architecture in other areas of scientific computing. The implementation was tested with an academic and a realistic problem.

The evaluation shows a considerable performance gain of the Cell implementation with respect to an x86-based one. This was accomplished by exploiting data parallelism and using many low-level optimizations targeted to the Cell processor, achieving up to a 46x speedup with respect to the x86 version. The Cell implementation takes advantage efficiently of an increasing number of SPUs thanks to the minimization of the communications and the lack of data dependences.

Regarding future work, a relevant contribution would be to incorporate support of non-structured meshes. Furthermore, a high order scheme could be implemented, which would allow more precision using coarser meshes.

Acknowledgements

This work was partially supported by the Science and Innovation Ministry of Spain (Projects TIN2010-16735, MTM2010-21135-C02-01 and MTM2009-11923), Xunta de Galicia CN2012/211 (partially supported by FEDER funds), and the FPU program of the Spanish Government (ref AP2009-4752). We thank Prof. Xavier Martorell and BSC for providing access to the MariCel system, and the PRACE prototype access program.

References

1. Audusse, E., Bouchut, F., Bristeau, M., Klein, R., Perthame, B.: A fast and stable well-balanced scheme with hydrostatic reconstruction for shallow water flows. SIAM J. Sci. Comp. **25 (6)**, 2050–2065 (2004)
2. Audusse, E., Bristeau, M.O.: Transport of pollutant in shallow water. a two time steps kinetic method. M2AN **37**, 389–416 (2003)
3. Benkhaldoun, F.I., Elmahi, I., Seaid, M.: Well-balanced finite volume schemes for pollutant transport on unstructured meshes. Journal of Computational Physics **226**, 180–203 (2007)
4. Bermúdez, A., Vázquez, M.E.: Upwind methods for hyperbolic conservation laws with source terms. Comput. and Fluids **23**, 1049–1071 (1994)
5. Bristeau, M.O., Perthame, B.: Transport of pollutant in shallow water using kinetic schemes. ESAIM Proceedings **10**, 9–21 (2001)
6. Brodtkorb, A.R., Sætra, M.L., Altinakar, M.: Efficient shallow water simulations on gpus: implementation, visualization, verification and validation (2010)
7. Castro, M., García-Rodríguez, J., González-Vida, J., Parés, C.: A parallel 2D finite volume scheme for solving systems of balance laws with nonconservative products: Application to shallow flows. Computer methods in applied mechanics and engineering **195**, 2788–2815 (2006)
8. Castro, M., García-Rodríguez, J., González-Vida, J., Parés, C.: Solving shallow-water systems in 2D domains using Finite Volume methods and multimedia SSE instructions. Journal of Computational and Applied Mathematics **221**, 16–32 (2008)
9. Castro, M.J., Ortega, S., de la Asunción, M., Mantas, J.M., Gallardo, J.M.: Gpu computing for shallow water flows simulation based on finite volume schemes. C.R. Mecanique (2010)
10. Castro-Díaz, M.J., Chacón, T., Fernández Nieto, E.D., González-Vida, J.M., Parés, C.: Well-balanced finite volume schemes for 2d non-homogeneous hyperbolic systems. application to the dam break of Aznalcóllar. Computer Methods in Applied Mechanics and Engineering **197**, 3932–3950 (2008)
11. Geveler, M., Ribbrock, D., Göddeke, D., Turek, S.: Lattice-boltzmann simulation of the shallow-water equations with fluid-structure interaction on multi- and manycore processors. In: Lecture Notes in Computer Science, vol. 6310, chap. Facing the multicore-challenge, pp. 92–104. Springer-Verlag (2010)
12. IBM, Sony, Toshiba: Cell Broadband Engine Architecture. IBM (2006)
13. Kurganov, A., Noelle, S., Petrova, G.: Semi-discrete central-upwind schemes for hyperbolic conservation laws and Hamilton-Jacobi equations. SIAM J. Sci. Comput. **23**, 707–740 (2001)
14. Kurganov, A., Petrova, G.: A second-order well-balanced positivity preserving central-upwind scheme for the Saint-Venant system. Commun. Math. Sci. **5(1)**, 133–160 (2007)
15. LeVeque, R.J.: Finite volume methods for hyperbolic problems. Cambridge University Press (2002)
16. Lobeiras, J., Viñas, M., Amor, M., Fraguela, B.B., Arenaz, M., García, J., Castro, M.: Parallelization of shallow water simulations on current multi-threaded systems. International Journal of High Performance Computing Applications (2012)

17. Noelle, S., Xing, Y., Shu, C.: High order well-balanced finite volume WENO schemes for shallow water equations with moving water. J. Comp. Phys. **226**, 29–58 (2007)
18. Perthame, B., Simeoni, C.: A kinetic scheme for the Saint-Venant system with a source term. Calcolo **38**, 201–231 (2001)
19. Rostrup, S., Sterck, H.D.: Parallel hyperbolic PDE simulation on clusters: Cell versus GPU. Computer Physics Communications **181**(12), 2164–2179 (2010)
20. Viñas, M., Lobeiras, J., Fraguela, B.B., Arenaz, M., Amor, M., García, J., Castro, M., Doallo, R.: A multi-gpu shallow-water simulation with transport of contaminants. Concurrency and Computation: Practice and Experience (2012)
21. Xu, Z., Shu, C.W.: Anti-diffusive finite difference weno methods for shallow water with transport of pollutant. SIAM J Numer. Analysis **46**, 1012–1039 (2006)