ELSEVIER

Available online at www.sciencedirect.com

ScienceDirect

Journal of Computational and Applied Mathematics 221 (2008) 16–32

JOURNAL OF COMPUTATIONAL AND APPLIED MATHEMATICS

www.elsevier.com/locate/cam

Solving shallow-water systems in 2D domains using Finite Volume methods and multimedia SSE instructions

M.J. Castro[a], J.A. García-Rodríguez[a], J.M. González-Vida[b,*], C. Parés[a]

[a] *Dpto. de Análisis Matemático, Universidad de Málaga, Campus Teatinos s/n, 29071 Málaga, Spain*
[b] *Dpto. de Matemática Aplicada, Universidad de Málaga, Spain*

Received 23 June 2006; received in revised form 22 May 2007

Abstract

The goal of this paper is to construct efficient parallel solvers for 2D hyperbolic systems of conservation laws with source terms and nonconservative products. The method of lines is applied: at every intercell a projected Riemann problem along the normal direction is considered which is discretized by means of well-balanced Roe methods. The resulting 2D numerical scheme is explicit and first-order accurate. In [M.J. Castro, J.A. García, J.M. González, C. Pares, A parallel 2D Finite Volume scheme for solving systems of balance laws with nonconservative products: Application to shallow flows, Comput. Methods Appl. Mech. Engrg. 196 (2006) 2788–2815] a domain decomposition method was used to parallelize the resulting numerical scheme, which was implemented in a PC cluster by means of MPI techniques.

In this paper, in order to optimize the computations, a new parallelization of SIMD type is performed at each MPI thread, by means of SSE (“*Streaming SIMD Extensions*”), which are present in common processors. More specifically, as the most costly part of the calculations performed at each processor consists of a huge number of small matrix and vector computations, we use the *Intel© Integrated Performance Primitives small matrix* library. To make easy the use of this library, which is implemented using assembler and SSE instructions, we have developed a C++ wrapper of this library in an efficient way. Some numerical tests were carried out to validate the performance of the C++ small matrix wrapper. The specific application of the scheme to one-layer Shallow-Water systems has been implemented on a PC’s cluster. The correct behavior of the one-layer model is assessed using laboratory data.
© 2007 Elsevier B.V. All rights reserved.

Keywords: Conservation laws; Shallow-water equations; Geophysical flows; Nonconservative products; Finite Volume methods; Parallelization; SIMD parallelism; SSE instructions

1. Introduction

This article deals with the development of efficient parallel implementations of Finite Volume solvers on unstructured grids to solve 2D hyperbolic systems of conservation laws with source terms and nonconservative products.

* Corresponding author. Tel.: +34 952131898; fax: +34 952131894.
E-mail addresses: castro@anamat.cie.uma.es (M.J. Castro), vida@anamat.cie.uma.es (J.M. González-Vida).

0377-0427/$ - see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.cam.2007.10.034

Problems of this nature arise, for example, in Computational Fluid Dynamics. We are concerned in particular with the simulation of free-surface waves in shallow layers of homogeneous fluids or internal waves in stratified fluids composed by two shallow layers of immiscible liquids. The motion of a layer of homogeneous fluid is supposed here to be governed by the Shallow-Water system, formulated under the form of a conservation law with source terms or *balance law*. In the stratified case, the flow is supposed to be governed by a system composed by two coupled Shallow-Water systems. The global system can be formulated under the form of two coupled balance laws, the coupling terms having the form of nonconservative products.

We are mainly interested in the application of these systems to geophysical flows: models based on Shallow-Water systems are useful for the simulation of rivers, channels, dambreak problems, ocean currents, estuarine systems, etc. Simulating those phenomena gives place to very long lasting simulations in big computational domains, so extremely efficient implementations are needed to be able to analyze those problems in low CPU time.

It is well-known that standard methods that solve correctly systems of conservation laws can fail in solving systems of balance laws, specially when approaching equilibria or near to equilibria solutions. Moreover, they can produce unstable methods when they are applied to coupled systems of conservation or balance laws. In the context of the numerical analysis of systems and coupled systems of balance laws, many authors have studied the design of well-balanced schemes, that is, schemes that preserve some equilibria: [20,1,25,23,2,13,3,18], ...

In [5] an efficient implementation of a first-order well-balanced numerical scheme for general systems of balance laws with nonconservative products was presented. This numerical scheme was a generalization of the 1D solver presented in [3,18]. The scheme was parallelized using domain decomposition techniques and MPI in a PC cluster. Very good speed-up results were obtained and the scheme was assessed with numerical and experimental data. Even though, the speed-up results were very good, the nature of our targeted problems demands even more computing power: for example, 2D tidal simulations of months or years in the Strait of Gibraltar can take several days or even weeks of simulation time in a PC cluster.

In order to increase the performance of the implementation, two main difficulties appear: on the one hand, PC clusters present an important bottleneck due to network latency which is very difficult to overcome by using only low cost hardware components. On the other hand, the number of nodes has to be doubled in order to halve the CPU time, which becomes exponentially expensive.

In this work we follow a different approach to improve the CPU time. The most costly part of the computations performed at each processor consists of a huge number of small matrix and vector computations. This computations are similar to those carried out in multimedia programs, such as 3D games/CAD, graphics, data encryption, video/image compression, etc. Modern CPUs are provided with specific SIMD units specifically designed to carry out these kind of computations in a vectorial way. In this work we introduce a technique to develop a high level C++ small matrix wrapper that takes advantage of multimedia SIMD instructions, present in modern processors (like Intel© Pentium/Xeon, AMD© Athlon/Opteron, PowerPC©), hiding the difficulties related to the use of very low level coding (mostly assembler) needed to develop an efficient SIMD implementation. Several techniques must be used to develop a high level C++ small matrix wrapper without losing the efficiency of the low level SSE implementation: templates, overloading, function inlining, etc.

The organization of the article is as follows: in Section 2, we introduce the general formulation of systems of balance laws with nonconservative products and source terms in 2D domains and the particular cases corresponding to the one and two-layer Shallow-Water systems. Next, the numerical scheme is presented for the general case. In Section 4, after a brief description of the SSE instructions/registers, the high level C++ small matrix wrapper implementation is described. In Section 5 some numerical experiments are presented to show the performance of the developed matrix wrapper and to compare the efficiency of the SSE implementation of the algorithm with the MPI implementation: in the case of one-layer flow simulations, reduction of CPU time of factor 12 could be obtained.

Finally, a numerical experiment is presented in order to validate the solver and to assess the performance of the one-layer Shallow-Water model. The validation is done by comparison with experimental measurements of a river flooding performed in an scaled laboratory model developed at CITEEC (A Coruña).

2. Equations

We consider a general problem consisting of a system of conservation laws with nonconservative products and source terms given by:

$$\frac{\partial W}{\partial t}+\frac{\partial F_1}{\partial x_1}(W)+\frac{\partial F_2}{\partial x_2}(W)=B_1(W)\cdot\frac{\partial W}{\partial x_1}+B_2(W)\cdot\frac{\partial W}{\partial x_2}+S_1(W)\frac{\partial H}{\partial x_1}+S_2(W)\frac{\partial H}{\partial x_2}, \tag{1}$$

where $W(\boldsymbol{x},t)\colon D\times(0,T)\mapsto \Omega\subset\mathbb{R}^N$, being D a bounded domain of $\mathbb{R}^2$; $\boldsymbol{x}=(x_1,x_2)$ denotes an arbitrary point of D; Ω is an open convex subset of $\mathbb{R}^N$. Finally $F_i\colon\Omega\mapsto\mathbb{R}^N$, $B_i\colon\Omega\mapsto\mathcal{M}_N$, $S_i\colon\Omega\mapsto\mathbb{R}^N$, $i=1,2$, are regular functions, and $H\colon D\mapsto\mathbb{R}$ is a known function. Observe that if $B_1=B_2=0=S_1=S_2=0$, (1) is a system of conservation laws, and if $B_1=B_2=0$, (1) is a system of conservation laws with source term (or balance laws).

Let $J_i(W)=\frac{\partial F_i}{\partial W}(W)$, $i=1,2$ denote the Jacobians of the fluxes F_i, $i=1,2$. Given a unit vector $\boldsymbol{\eta}=(\eta_1,\eta_2)\in\mathbb{R}^2$, we define the matrix

$$\mathcal{A}(W,\boldsymbol{\eta})=J_1(W)\eta_1+J_2(W)\eta_2-(B_1(W)\eta_1+B_2(W)\eta_2).$$

We assume here that (1) is strictly hyperbolic, i.e. for every W in Ω and every unit vector $\boldsymbol{\eta}\in\mathbb{R}^2$, $A(W,\boldsymbol{\eta})$ has N real distinct eigenvalues and thus it is diagonalizable:

$$\mathcal{A}(W,\boldsymbol{\eta})=\mathcal{K}(W,\boldsymbol{\eta})\mathcal{D}(W,\eta)\mathcal{K}^{-1}(W,\boldsymbol{\eta}), \tag{2}$$

where $\mathcal{D}(W,\boldsymbol{\eta})$ is the diagonal matrix whose coefficients are the eigenvalues of $\mathcal{A}(W,\boldsymbol{\eta})$ and $\mathcal{K}(W,\boldsymbol{\eta})$ is a matrix whose columns are associated eigenvectors.

Notice that the nonconservative products $B_1(W)\partial_{x_1}W$, $B_2(W)\partial_{x_2}W$, do not make sense in the sense of distributions for discontinuous solutions. The problem of giving a sense to the solution is difficult: we refer to [9,17].

2.1. Examples

Some examples of equations that fit into this abstract framework are the one and two-layer Shallow-Water systems.

(1) *One-layer Shallow-Water system*

The PDE system governing the flow of a shallow layer of fluid that occupies a subdomain $D\subset\mathbb{R}^2$ can be written in the form (1) with:

$$W=\begin{bmatrix}h\\ q_1\\ q_2\end{bmatrix}, \tag{3}$$

$$F_1(W)=\begin{bmatrix}q_1\\ \dfrac{q_1^2}{h}+\dfrac{1}{2}gh^2\\ \dfrac{q_1q_2}{h}\end{bmatrix},\qquad F_2(W)=\begin{bmatrix}q_2\\ \dfrac{q_1q_2}{h}\\ \dfrac{q_2^2}{h}+\dfrac{1}{2}gh^2\end{bmatrix}, \tag{4}$$

$$S_1(W)=\begin{bmatrix}0\\ gh\\ 0\end{bmatrix},\qquad S_2(W)=\begin{bmatrix}0\\ 0\\ gh\end{bmatrix}, \tag{5}$$

where $H(\boldsymbol{x})$ is the depth function measured from a fixed level of reference; g is the gravity constant; $h(\boldsymbol{x},t)$ and $\boldsymbol{q}(\boldsymbol{x},t)=(q_1(\boldsymbol{x},t),q_2(\boldsymbol{x},t))$ are, respectively, the thickness and the mass-flow of the water layer at the point $\boldsymbol{x}$ at time t, that are related to the velocity $\boldsymbol{u}(\boldsymbol{x},t)=(u_1(\boldsymbol{x},t),u_2(\boldsymbol{x},t))$ through the equality:

$$\boldsymbol{q}(\boldsymbol{x},t)=\boldsymbol{u}(\boldsymbol{x},t)h(\boldsymbol{x},t).$$

Here, $B_1=B_2=0$.

For the sake of simplicity friction terms are neglected.

(2) *Two-layer Shallow-Water system*

The system of equations governing the 2D flow of two superposed immiscible layers of shallow fluids in a subdomain $D\subset\mathbb{R}^2$, can be also written in the form (1) with:

$$W=\begin{bmatrix}h_1, & q_{1,1}, & q_{1,2}, & h_2, & q_{2,1}, & q_{2,2}\end{bmatrix}^{\mathrm{T}}, \tag{6}$$

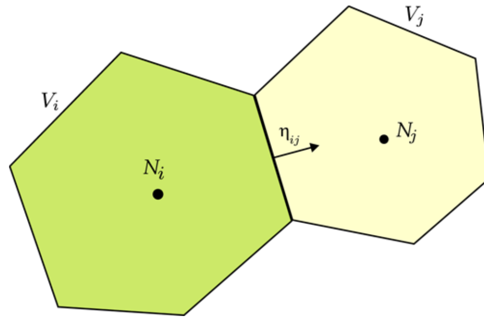


Fig. 1. Finite volumes.

$$F_1(W) = \begin{bmatrix} q_{1,1} \\ \dfrac{q_{1,1}^2}{h_1} + \dfrac{1}{2} g h_1^2 \\ \dfrac{q_{1,1} q_{1,2}}{h_1} \\ q_{2,1} \\ \dfrac{q_{2,1}^2}{h_2} + \dfrac{1}{2} g h_2^2 \\ \dfrac{q_{2,1} q_{2,2}}{h_2} \end{bmatrix}, \qquad F_2(W) = \begin{bmatrix} q_{1,2} \\ \dfrac{q_{1,1} q_{1,2}}{h_1} \\ \dfrac{q_{1,2}^2}{h_1} + \dfrac{1}{2} g\, h_1^2 \\ q_{2,2} \\ \dfrac{q_{2,1} q_{2,2}}{h_2} \\ \dfrac{q_{2,2}^2}{h_2} + \dfrac{1}{2} g h_2^2 \end{bmatrix}, \tag{7}$$

$$B_1(W) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -gh_1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ -rgh_2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \qquad B_2(W) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -gh_1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ -rgh_2 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \tag{8}$$

$$S_1(\boldsymbol{x}, W) = \begin{bmatrix} 0, & gh_1, & 0, & 0, & gh_2, & 0 \end{bmatrix}^{\mathrm{T}}, \tag{9}$$

$$S_2(\boldsymbol{x}, W) = \begin{bmatrix} 0, & 0, & gh_1, & 0, & 0, & gh_2 \end{bmatrix}^{\mathrm{T}}. \tag{10}$$

Index 1 refers to the upper layer and index 2 to the lower one. Again, g is the gravity and $H(\boldsymbol{x})$, the depth function measured from a fixed level of reference. $r = \frac{\rho_1}{\rho_2}$ 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 ith layer at the point $\boldsymbol{x}$ at time t, and again they are related to the velocities $\boldsymbol{u}_i(\boldsymbol{x}, t) = (u_{i,1}(\boldsymbol{x}, t), u_{i,2}(\boldsymbol{x}, t))$, $i = 1, 2$ by the equalities:

$$\boldsymbol{q}_i(\boldsymbol{x}, t) = \boldsymbol{u}_i(\boldsymbol{x}, t) h_i(\boldsymbol{x}, t), \quad i = 1, 2.$$

Again, friction terms are neglected.

3. Numerical scheme

In this section we present the discretization of system (1) by means of a Finite Volume scheme. First, the computational domain is decomposed into discretization cells or Finite Volumes, $V_i \subset \mathbb{R}^2$, which are supposed to be closed polygons. Let us denote by $\mathcal{T}$ the set of cells. Hereafter we will use the following notation: given V_i a Finite Volume, $N_i \in \mathbb{R}^2$ is the center of V_i, $\mathcal{N}_i$ is the set of indexes j such that V_j is a neighbor of V_i. Γ_{ij} is the common edge of two neighbor cells V_i and V_j, and $|\Gamma_{ij}|$ represents its length. $\boldsymbol{\eta}_{ij} = (\eta_{ij,1}, \eta_{ij,2})$ is the unit vector which is normal to the edge Γ_{ij} and points toward the cell V_j (see Fig. 1).

The approximations to the cell averages of the exact solution produced by the numerical scheme will be denoted as follows:

$$W_i^n \cong \frac{1}{|V_i|} \int W(x_1, x_2, t^n) \mathrm{d}x_1 \mathrm{d}x_2 \tag{11}$$

where $|V_i|$ is the area of the cell and $t^n = n\Delta t$, being Δt the time step which is supposed to be constant for simplicity.

The numerical scheme considered is:

$$W_i^{n+1} = W_i^n - \frac{\Delta t}{|V_i|} \sum_{j \in \mathcal{N}_i} |\Gamma_{ij}| F_{ij}^-, \tag{12}$$

where

$$F_{ij}^- = \mathcal{P}_{ij}^- (\mathcal{A}_{ij}(W_j^n - W_i^n) - \mathcal{S}_{ij}(H_j - H_i)) \tag{13}$$

with $\mathcal{A}_{ij} = \mathcal{A}(W_{ij}^n, \boldsymbol{\eta}_{ij})$; where W_{ij}^n is the Roe "intermediate state" corresponding to W_i^n and W_j^n and

$$P_{ij}^- = \frac{1}{2} \mathcal{K}_{ij} \cdot (I - \mathrm{sgn}(\mathcal{D}_{ij})) \cdot \mathcal{K}_{ij}^{-1}, \tag{14}$$

$$\mathcal{S}_{ij} = \eta_{ij,1} S_1(W_{ij}^n) + \eta_{ij,2} S_2(W_{ij}^n), \tag{15}$$

I being the identity matrix, $\mathcal{D}_{ij}$ the diagonal matrix whose coefficients are the eigenvalues of A_{ij}, and $\mathcal{K}_{ij}$ a matrix whose columns are associated eigenvectors. Finally $\mathrm{sgn}(\mathcal{D}_{ij})$ is the diagonal matrix whose coefficients are the signs of the eigenvalues of matrix $\mathcal{A}_{ij}$.

3.1. Some remarks

(1) Due to the explicit character of the numerical scheme, a CFL condition has to be imposed. In practice, the following condition can be used:

$$\frac{1}{2} \frac{\Delta t}{|V_i|} \sum_{j \in \mathcal{N}_i} |\Gamma_{ij}| \|D_{ij}\|_\infty \leq \gamma, \quad 1 \leq i \leq M,$$

where $0 < \gamma \leq 1$ and M is the number of cells in the spatial domain.

The time step computed from this criterion can be small during the calculations, which means that a large number of time iterations can be necessary for large time simulations. It is known that implicit numerical schemes allow for large time steps. Nevertheless, we consider explicit numerical schemes for the following reasons:

(a) We are interested not only in reaching steady states but also in the simulation of fast waves as moving shocks or dry/wet fronts appearing in fluvial or coastal hydraulics.

(b) Explicit schemes involve much reduced memory overhead, as complex iterative matrix solvers are not required.

(c) Explicit schemes are much easier to implement in massively parallel computers.

(2) As in the case of systems of conservation laws, when sonic rarefaction waves appear, it is necessary to modify the approximate Roe solver in order to obtain entropy-satisfying solutions. The Harten–Hyman Entropy Fix technique (see [15]) can be easily adapted to this numerical scheme.

(3) In [6], a 1D high-order well-balanced Finite Volume scheme was developed in a general nonconservative framework. This scheme uses high-order reconstruction of states and it is based on the generalized Roe schemes introduced in [24], whose well-balanced properties have been studied in [18]. In particular, it was successfully applied to the Shallow-Water system with bottom topography, using fifth-order WENO reconstructions in space and a third-order TVD Runge–Kutta scheme to advance in time. Finally, in [11] and [8], the extension of this scheme to 2D high-order well-balanced Finite Volume schemes for nonconservative hyperbolic systems has been carried out. The use of the C++ small matrix wrapper described in this paper in the implementation of these high-order schemes is straightforward.

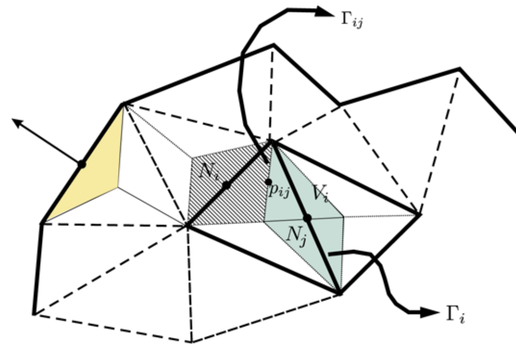


Fig. 2. Edge finite volume.

(4) In the numerical experiments shown in Section 5 we have considered an edge-based Finite Volume scheme, constructed on the basis of a Finite Element unstructured meshes. This kind of meshes makes easier the discretization of domains of complex geometry and the numerical treatment of boundary conditions. In order to build the Finite Volume mesh we proceed as follows: first, a Delaunay Finite Element mesh is constructed. Next, given an edge Γ_i, each triangle containing Γ_i is decomposed into three subtriangles having the barycenter as a common vertex. The Finite Volume V_i corresponding to Γ_i is then constructed by joining the subtriangles containing this edge. Nodes are defined as the middle points of the edges (see Fig. 2).

(5) An important difficulty arising in the numerical experiments shown in Section 5 is the appearance of dry areas. If no modifications are made, the numerical scheme used in this work loses its well-balanced property in the presence of wet/dry transitions. Moreover, it may produce negative values of the thickness of the water layer in the proximities of the wet/dry front. In [4,7], a modification was proposed that overcomes this difficulty. It consists in replacing, at the intercells where a wet/dry transition is detected, the corresponding linear Riemann problem by an adequate nonlinear one. Finally, in [12] a numerical scheme combining the numerical scheme developed in [6] with the treatment of wet/dry fronts introduced in [7] has been proposed. Again, the use of the C++ small matrix wrapper described here is straightforward in this case.

4. Parallel SIMD implementation

Explicit numerical methods can be parallelized in a natural way: the computational domain is split into subdomains and then each subdomain is sent to a node of a cluster. At each time step, information related to the boundaries between two neighbor subdomains have to be efficiently exchanged between the corresponding nodes. Following this approach, it is possible to nearly "halve" the calculus time when the number of computing nodes is doubled. In [5] a parallelization of (12) based on domain decomposition techniques was carried out, but this approach presents two main drawbacks:

(1) in PC clusters, there is an important bottleneck due to network latency, which is almost impossible to overcome if only low cost components are used (and even when expensive high class network hardware is used),

(2) the cost of building a cluster to halve the computing time grows exponentially. Thus, if the cluster becomes big enough, halving the CPU time can become really prohibitive.

Therefore, in order to obtain big performance improvements with a medium cluster, it is necessary to make a better use of the computational power at each node. It is well-known that most of the circuitry of modern common processors is devoted to `L2` cache, and just a little part is really devoted to carry out operations: the FPU ("floating point units") and the SSE registers, which are used to carry out the very demanding multimedia operations. The idea is to use also these registers to perform the computations.

An analysis of the numerical scheme described in the previous section shows that the most expensive operations are:

- The computation of the inverse matrix $\mathcal{K}_{ij}^{-1}$ in expression (14).
- `Matrix· Diagonal Matrix· Matrix` computations in expression (14).

- The computation of the matrices $\mathcal{D}_{ij}$ and $\mathcal{K}_{ij}$ in (14), when their explicit expressions are not available, as in the case of the two-layer shallow-water system. In this case, we approximate the eigenvalues and eigenvectors of $\mathcal{A}_{ij}$ by means of a C algorithm due to Burton S. Garbow et al. [14]: it is based on reducing the matrix to upper Hessemberg form and then using the QR method to find its eigenvalues.

Our main objective is thus to optimize the matrix–matrix, matrix–vector, and vector–vector computations. To do this, the way in which small matrix computations are carried out at each CPU must be revised and then improved by using the SIMD parallelism available in modern processors. More precisely, we will focus on efficient small matrix operations in Pentium© IV and Xeon© processors. The way to achieve a better performance is to use a vectorized matrix library. This can be done, for example, in Pentium© or AMD© processors using SSE instructions and, in PowerPC© processors, using Altivec.

SSE instructions, introduced in 1999 by Intel© in its Pentium© III processors, are an approach to SIMD parallelism in common processors. They are used to perform highly time consuming calculations like for example image and video compression, signal processing, 3D operations for CAD and games, etc. They were designed with the intention of using the CPU for multimedia purposes when powerful DSPs ("Digital Signal Processors") and 3D cards were not generalized among PC users.

SSE provides a set of eight registers each having 128 bits. These registers can store data in 128 bits, 64 bits, 32 bits, 16 bits, etc. A complete set of instructions are provided to carry out operations directly, once the data are stored in them. With the introduction of EM64T these registers have been doubled and now there are 16 registers available in 64 bit processors, such as Xeon© EM64T and Opteron©.

PowerPC© processors provide a similar approach called *Altivec* which consists of 32 registers each having 128 bits. An important difference between these two implementations is that Altivec does not allow the use of 64 bit numbers directly.

The easiest way to make use of the SIMD registers is to turn on the compiler optimizers to autovectorize the codes. The main drawback is that this procedure is only efficient in some easy cases. In general, it is necessary to include in codes specific directives to use the SIMD registers in an optimal way. There are three programming techniques to do this (see [10]), which are in the order of increasing level, the following:

- Assembler.
- Intrinsics.
- Libraries.

While at the lower levels the programmer has to control the memory positions of data, the way in which the SIMD registers are filled, the availability of the registers, etc., the higher interfaces allow the programmer to hand over the control of some of these basic aspects to the compiler. Nevertheless, even when using libraries, which is the interface chosen here, the calls to routines that carry out operations in the SIMD registers are still far more complicated than the usual sentences in high level languages. This is obviously an important drawback to developers of numerical methods.

Therefore, our objective is to develop a high level C++ small matrix wrapper allowing us to use the SIMD registers in an optimal way from standard C++ codes. The three main requirements for such a wrapper in order to be useful are the following:

- Easiness of use: the sentences to carry out the matrix–vector operations must be easy and natural.
- Portability: the wrapper must be a layer on top of the specific small matrix libraries corresponding to a particular architecture.
- The performance of the small matrix libraries must be preserved.

Taking these requirements in mind, we have developed a wrapper able to use the small matrix libraries corresponding to the SIMD registers for Altivec or SSE registers. This development has been based on the use of the advanced characteristics of C++ like overloading, templates and function inlining (see [22]).

4.1. Intel© IPP small matrix library

The *Integrated Performance Primitives* (IPP) are a set of numerical libraries developed by Intel© using assembler and intrinsics to help software developers to make use of SSE registers using a higher programming level than rough assembler (see [16] for details). These libraries are grouped in several categories: video and audio compression,

Table 1
Efficiency of the C++ wrapper: 3×3 matrices and/or 3×1 vectors

Operation	IPP (s)	Optimized wrapper (s)	Wrapper with temporary variables (s)
$V + V$	7.263	7.266	10.772
$M + M$	18.660	18.667	29.312
$M \cdot V$	13.130	13.145	24.533
$M \cdot V + V$	16.387	16.391	29.852
M^{-1}	44.735	44.745	49.970
$M \cdot D \cdot M^{-1} \cdot V$	No primitive	108.260	226.663

speech recognition, cryptography, mathematical operations like signal analysis and Fourier transform, small matrix operations for physics modeling, etc.

The small matrix library contains all the usual matrix–vector and matrix–matrix operations such as: matrix–matrix addition, vector–vector addition, matrix–vector product, matrix inversion, vector–vector cross-product, solving linear systems using LU or Cholesky decomposition, etc. It is interesting to point out that it also provides a special set of optimized primitives related to very small matrix–vector computations (up to size 6×6–6×1 for vectors) which are very well-suited for the kind of problems we are interested in, as the size of the matrices appearing in the implementation of the numerical scheme described in Section 3 is 3×3 or 6×6 when it is applied to the one-layer or the two-layer Shallow-Water system respectively.

As it has been said above, even if this library allows us to use SSE instructions from a higher level, the syntax specifications are by no means simple. For example, if we want to add two matrices of float double precision numbers, A, B of size 6×6 and store the result in C, the resulting line of code should read as follows:

```
ippmAdd_mm_64f_6x6(A,LR,B,LR,C,LR);
```

where LR denotes the distance in bits between the columns of the matrix (in this example $LR = 48 = 6 \times 8$ bits). So, the name of each function depends on the data type, on the matrix size and on the distance in bits between columns.

A code generated with such a syntax will be difficult to debug, which is not desirable when implementing complex numerical methods. Besides, the resulting codes will be difficult to reuse and to modify.

4.2. *Small matrix wrapper*

In this section, we describe the different techniques used to develop a portable and easy-to-use small matrix wrapper preserving the original performance of the Intel© IPP small matrix library. One of the main advantages of using such a wrapper is that it can be combined with the parallel domain decomposition implementation presented in [5] in a very easy manner: the numerical expression involving matrix–matrix or matrix–vector computations is implemented using the small matrix wrapper. Thus, the resulting code makes use of all the resources available in a PC cluster in a very efficient way.

4.2.1. *Optimized operator overloading*

The main task when developing a matrix wrapper is the implementation of the most common matrix–vector, vector–vector and matrix–matrix operations. Afterwards these operations will be used in the design of complex numerical codes.

The fundamental tool to develop the matrix wrapper is the *operator overloading*, available in object programming languages like C++. Operator overloading allows us to program using a notation which is close to the mathematical language. Moreover, it allows us to build user types like matrix, vector, complex number, etc. which look like the built-in types like integer, simple or double precision number (see [22] for details).

Nevertheless, if the operator overloading is carried out in the usual way, a great part of the efficiency of the original library may be lost, as it will be shown in Section 4.3.1, Tables 1 and 2.

Usually, when a binary operation is carried in a processor, a temporary variable is needed to store the result, before it is assigned. For example, when executing $C = A + B$, A, B and C being three matrices, a temporary matrix variable

Table 2
Efficiency of the C++ wrapper: 6×6 matrices and/or 6×1 vectors

Operation	IPP (s)	Optimized wrapper (s)	Wrapper with temporary variables (s)
$V + V$	1.206	1.207	2.373
$M + M$	4.766	4.772	7.377
$M \cdot V$	5.118	5.201	7.986
$M \cdot V + V$	5.112	5.117	9.251
M^{-1}	6.596 Wrong calculus	152.170	156.722
$M \cdot D \cdot M^{-1} \cdot V$	No primitive	173.351	216.361

is created, $\widetilde{C}$ storing the result of the addition and then it is assigned to the variable C. The creation of this temporary variable can nearly double the CPU time required to compute the operation.

To avoid the creation of temporary matrix or vector variables, we used a technique described in [22] based on *composition closure data types*. The main idea is the use of compile-time analysis and closure data types to transfer the evaluation of subexpressions into a data type representing a composite operation. Using this technique it is possible to achieve that an operation of n-binary operators behaves like a $(n+1)$-ary operator, creating for this purpose as many closure data types as needed. Let us see how it works with a simple example.

Let us consider again the following computation $C = A + B$, where A, B and C are matrices. To avoid temporary variable creation, the computation of $A + B$ should be stored directly in C. This can be done by considering a new data type noted by ''MMsum'' (composition closure data type), that does not perform any operation: it only saves the memory references to the operands as follows:

```
#ifndef _matriz6_h
#define _matriz6_h
#include <ipp_w7.h>
#include<ippm.h>

class Matrix;
class MMsum {
 public:
 const Matrix& m0; % memory reference to matrix A
 const Matrix& m1; % memory reference to matrix B
 MMsum(const Matrix&mm0,const Matrix& mm1): m0(mm0),m1(mm1) {};
 operator Matrix();
 friend inline MMsum operator+(const Matrix& mm0,const Matrix& mm1)
{
  return MMsum(mm0,mm1);
}
};

MMsum::operator Matrix() {
  Matrix m;
  ippmAdd_mm_{6}4f_{6}x6(m0.v,LR,m1.v,LR,m.v,LR);
  return m;
}
Matrix& operator=(const MMsum& m){
    ippmAdd_mm_{6}4f_{6}x6(m.m0.v,LR,m.m1.v,LR,v,LR); % m.m0.v is A, m.m1.v is B and v is C
    return (*this);
}
```

Let us analyze how the compiler works when it finds an expression like $C = A + B$. It reads the expression from right to left and, when it finds two matrices separated by the sign "+", the compiler identifies $A + B$ as an object of the data type MMsum. Thus it only stores the memory references to the operands A and B, but no computation is carried out at this stage. Then, the sign "=" as well as the matrix C is found at the left side. At this stage, the function `Matrix& operator=(const MMsum& m)` is used and the computation $A + B$ is performed and stored directly in C using the proper IPP function ippmAdd_mm_64f_6×6. Note that, through all this process, no temporary matrix has been created and the operation is stored directly in C.

The use of this technique allows the definition of more complex operations like $V = A \cdot V - W$, A being a matrix and V, W two vectors, or in general, any set of operations that are commonly used in a numerical algorithm, without using temporary matrix o vector variables.

Table 3
CPU time: Matrix library performances: 3 × 3 matrices and/or 3 × 1 vectors

Operation	Newmat v. 10.0 (s)	Gmm++ (s)	C++ small matrix wrapper (s)
$V + V$	353.810	16.434	7.266
$M + M$	602.675	364.012	18.657
$M \cdot V$	666.830	84.316	13.145
$M \cdot V + V$	891.380	190.810	16.381
M^{-1}	2122.920	440.120	44.745
$M \cdot D \cdot M^{-1} \cdot V$	5141.870	1012.053	108.260

4.2.2. *Function inlining*

When some small functions are called several times in a program, it is possible to declare them as *inline*: in this case the compiler copies the function in the corresponding part of the binary executable file as many times as necessary. This procedure avoids searching for the function in execution time.

4.2.3. *Templates*

As we have seen, the names of the IPP small matrix routines depends on data type and on matrix sizes. This fact has to be taken in account when developing the wrapper. If the choice of the adequate routine is done in execution time by means of conditionals, the performance of the wrapper can be drastically reduced. Instead, the use of *templates* makes it possible to do this choice in precompile time, i.e. when the executable is being generated. In effect, templates allow us to introduce parameters in the definition of data types or functions (see [22] for details). Together with the wrapper a template matrix has been created where the matrix size or the data type of the elements is given as arguments when creating a matrix. Then, within the template the definitions of the matrix operations depend on these arguments.

4.3. *Performance tests*

4.3.1. *Comparison between the C++ small matrix wrapper and the original Intel© IPP SIMD small matrix library*

In this section we present comparisons of the performance of the implemented C++ small matrix wrapper and the direct use of IPP small matrix functions.

CPU times corresponding to an implementation of the overloading operator using temporary variables (the usual implementation) are also presented.

In the tables we will use the following notation: V will mean vector and M matrix. The operations are carried out 1.000.000.000 times (respectively 100.000.000 times) for 3 × 3 matrices and/or 3 × 1 vectors (respectively 6 × 6 matrices and/or 6 × 1 vectors) in order to be able to measure a significant CPU time.

Note that there is no significant difference between the C++ small matrix wrapper and the original IPP functions.

Observe that the CPU time is nearly doubled for n-ary, $n \geq 2$ operators when temporary variables are used (see Tables 1 and 2).

4.3.2. *Comparison between the C++ small matrix wrapper and other C++ matrix libraries*

In this section we present comparisons between the developed C++ small matrix wrapper and some usual C++ matrix libraries. To carry out these comparisons we have considered Newmat v10.0, which is a C++ matrix library having the usual matrix–matrix and matrix–vector operations and Gmm++, which is based on C++/Blas.

Table 3 shows the CPU time corresponding again to 10^9 operations. Note that for the operation $M \cdot D \cdot M^{-1} \cdot V$ (which is required in (14)), the CPU time is reduced 48 times with respect to Newmat and 10 times with respect to Gmm++.

5. Numerical results

In this section we present some numerical results to show the performance of the C++ small matrix wrapper when it is used with the parallel implementation developed in [5].

Table 4
Number of finite volumes for each mesh

Mesh	Mesh 1	Mesh 2	Mesh 3	Mesh 4	Mesh 5
Number of volumes	620	1207	2590	5162	10 832

Table 5
Calculus time: Meshes 1, 2 and 3

CPUs	Mesh 1		Mesh 2		Mesh 3	
	C++ wrapper (s)	NON-SSE (s)	C++ wrapper (s)	NON-SSE (s)	SSE (s)	NON-SSE (s)
1	2.241	32.230	6.065	94.823	18.507	292.201
2	1.861	17.644	3.936	49.001	10.685	152.606
4	1.965	9.900	3.366	26.224	6.876	77.556
8	2.124	5.612	3.526	14.310	4.340	40.120

Table 6
CPU time: Meshes 4 and 5

CPUs	Mesh 4		Mesh 5	
	C++ wrapper (s)	NON-SSE (s)	C++ wrapper (s)	NON-SSE (s)
1	51.764	856.735	185.985	3021.319
2	29.066	426.800	98.830	1525.037
4	17.078	218.655	53.459	763.717
8	10.032	111.360	29.315	386.135

5.1. Numerical performance of the parallel SIMD implementation

The numerical tests in this section have been designed to test the efficiency of using together the MPI and SIMD parallelism.

5.1.1. One-layer flows

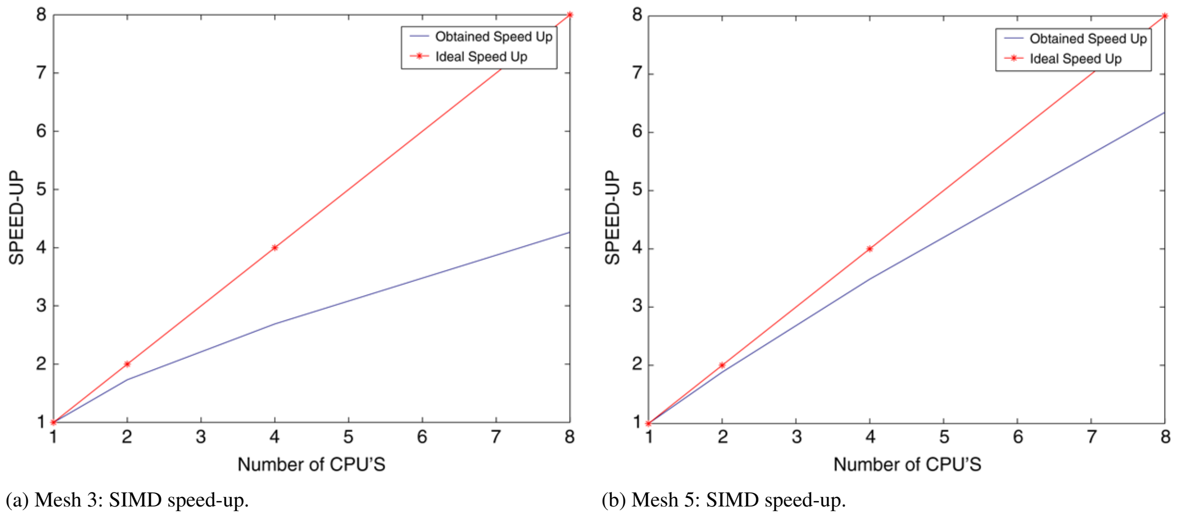
In this example, the one-layer Shallow-Water model is considered over a rectangular channel of 1 m width and 10 m long with a bump placed at the middle of the domain given by the depth function $H(x_1, x_2) = 1 - 0.2\mathrm{e}^{-(x_1-5)^2}$. Five meshes of the domain are constructed, each of them having a number of elements which are approximately double than the previous one (see Table 4).

The initial condition is given by

$$\boldsymbol{q}(x_1, x_2) = \mathbf{0},$$
$$h(x, y) = \begin{cases} H(x_1, x_2) + 0.7 & \text{if } 4 \le x_1 \le 6, \\ H(x_1, x_2) + 0.5 & \text{other case.} \end{cases} \tag{16}$$

The numerical scheme is run in the time interval [0, 10] with $CFL = 0.9$. Wall-boundary conditions $\boldsymbol{q} \cdot \boldsymbol{\eta} = 0$ are considered.

Tables 5 and 6 show the CPU time for each run. Observe that when only a single CPU is used, the CPU time is reduced approximately 16 times when the C++ small matrix wrapper is used. This efficiency is gradually lost when using a number of increasing CPUs in the parallel implementation. Nevertheless, this reduction is not critical for meshes bigger enough as is shown in Tables 5 and 6 where a reduction factor of order 12 is obtained when 8 CPUs are used (see the numbers for mesh 3 and mesh 5). This effect could also be observed in Fig. 3(a) and (b) where the speed-up for meshes 3 and 5 are shown. Finally, if a fine mesh composed by 244,163 volumes is used, the speed-up is nearly optimal as it is shown in Fig. 4.



(a) Mesh 3: SIMD speed-up.

(b) Mesh 5: SIMD speed-up.

Fig. 3. Speed-up for meshes 3 and 5: SIMD one-layer model.

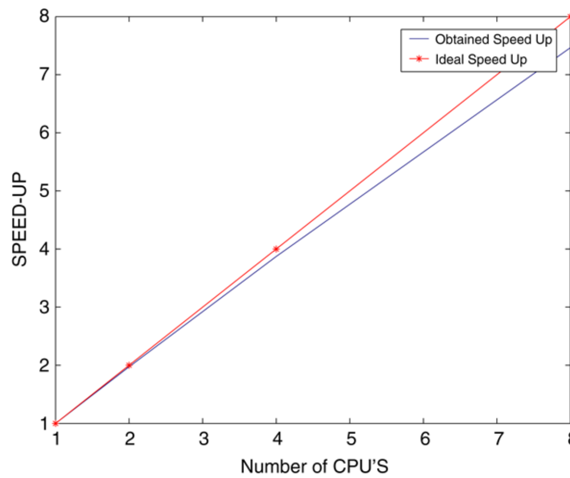


Fig. 4. Speed-up for a mesh of 244.163 volumes: SIMD one-layer model.

5.1.2. Two-layer flows

The initial condition is now given by

$$\begin{aligned} &\boldsymbol{q}_1(x_1, x_2) = \boldsymbol{q}_2(x_1, x_2) = \mathbf{0}, \\ &h_1(x_1, x_2) = \begin{cases} 0.7 & \text{if } 4 \le x_1 \le 6 \\ 0.5 & \text{other case} \end{cases}; \qquad h_2(x_1, x_2) = H(x_1, x_2). \end{aligned} \tag{17}$$

Wall-boundary conditions $\boldsymbol{q}_j \cdot \boldsymbol{\eta} = 0$, $j = 1, 2$ are again considered.

The *CFL* parameter is set to 0.9 and $r = 0.9985$. The model is integrated again in the time interval [0, 10] using the same meshes as in the previous numerical test and 1, 2, 4 or 8 nodes of the cluster.

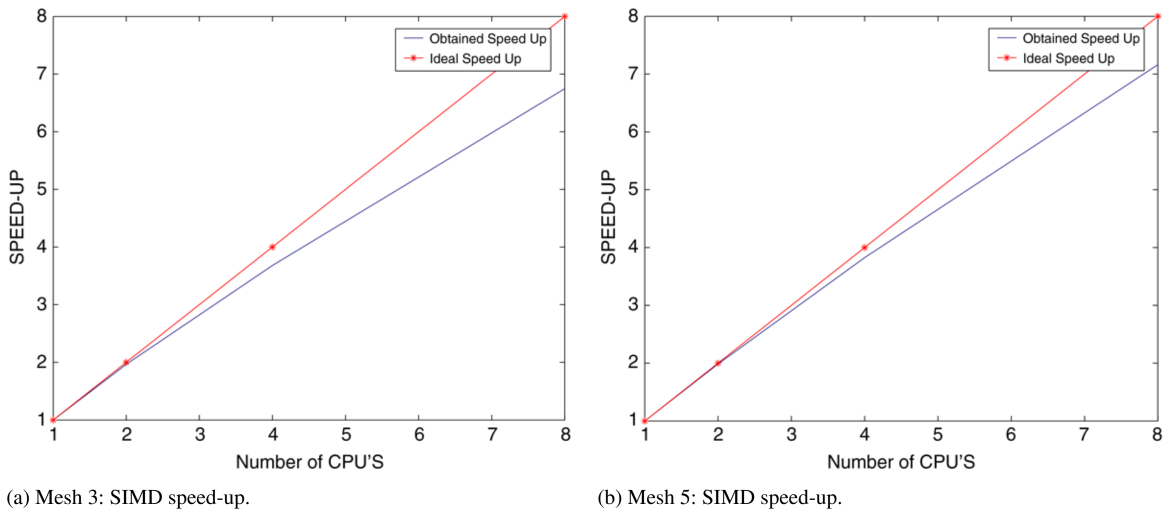
Tables 7 and 8 show the CPU time for each run. Observe that the CPU time is reduced approximately 2.5 times when the C++ small matrix wrapper is used and it is preserved when using a increasing number of CPUs in the parallel implementation. It could be also observed that the CPU time is approximately halved when the number of processors is doubled. Moreover, the reduction factor is closer to 2 as the mesh size increases.

Table 7
Elapsed calculus time: Meshes 1, 2 and 3

CPUs	Mesh 1		Mesh 2		Mesh 3	
	C++ wrapper (s)	NON-SSE (s)	C++ wrapper (s)	NON-SSE (s)	SSE (s)	NON-SSE (s)
1	30.829	76.07	89.308	120.30	272.403	648.53
2	16.528	39.57	45.611	113.57	138.585	338.16
4	8.268	21.74	25.671	58.85	74.014	186.69
8	5.120	12.86	15.134	32.57	40.378	100.13

Table 8
Elapsed calculus time: Meshes 4 and 5

CPUs	Mesh 4		Mesh 5	
	C++ wrapper (s)	NON-SSE (s)	C++ wrapper (s)	NON-SSE (s)
1	781.660	1837.51	2748.807	6478.15
2	397.231	977.24	1385.169	3426.47
4	208.077	501.53	718.672	1836.63
8	111.320	258.42	383.781	980.13



(a) Mesh 3: SIMD speed-up.

(b) Mesh 5: SIMD speed-up.

Fig. 5. Speed-up for meshes 3 and 5: SIMD two-layer model.

The CPU times and speed-up corresponding to meshes 3 and 5 are compared in Fig. 5 with an ideal linear efficiency. As expected, the behavior is nearly linear.

5.2. Comparison with experimental data: Numerical simulation of the flooding caused by River Mero near the Cecebre dam

This experiment is based on a project developed in the CITEEC laboratory of University of A Coruña. A scaled physical model of River Mero was developed in this laboratory to measure the velocities and water depth obtained when this river overflows near Cecebre dam. The objective was to design contention walls in the River banks to mitigate the effects of the floods. More precisely a scaled model of the meanders where flooding are more frequent was developed. In the design of the model sediment transport and sedimentation processes were not taken into account.

To build the scaled model, scientists from CITEEC used a highly detailed topography of the mentioned meanders. This topographic data contains a high number of points in the main river bed, obtained from digital topography and some measurements taken in fieldwork using a crane.

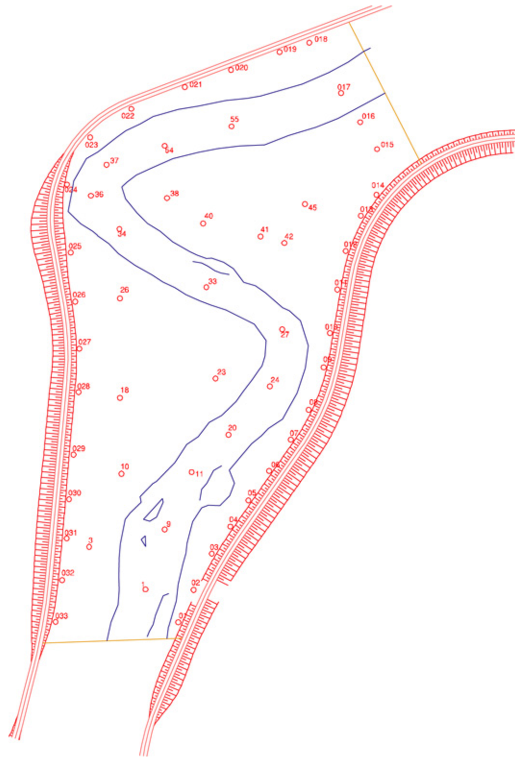


Fig. 6. Points of measurement in the physical model.

The topographic data were stored in Autocad© format. The plans of the project were used to locate the position of the walls to be built.

Once the scaled model was built, several experiments were performed and the corresponding data were registered in several points (see Fig. 6), which becomes a valuable source to compare free-surface elevations and velocities with the outcomes of the numerical models.

A detailed description of the construction of the model and the process of taking the measurements is available in [19].

Numerical experiment

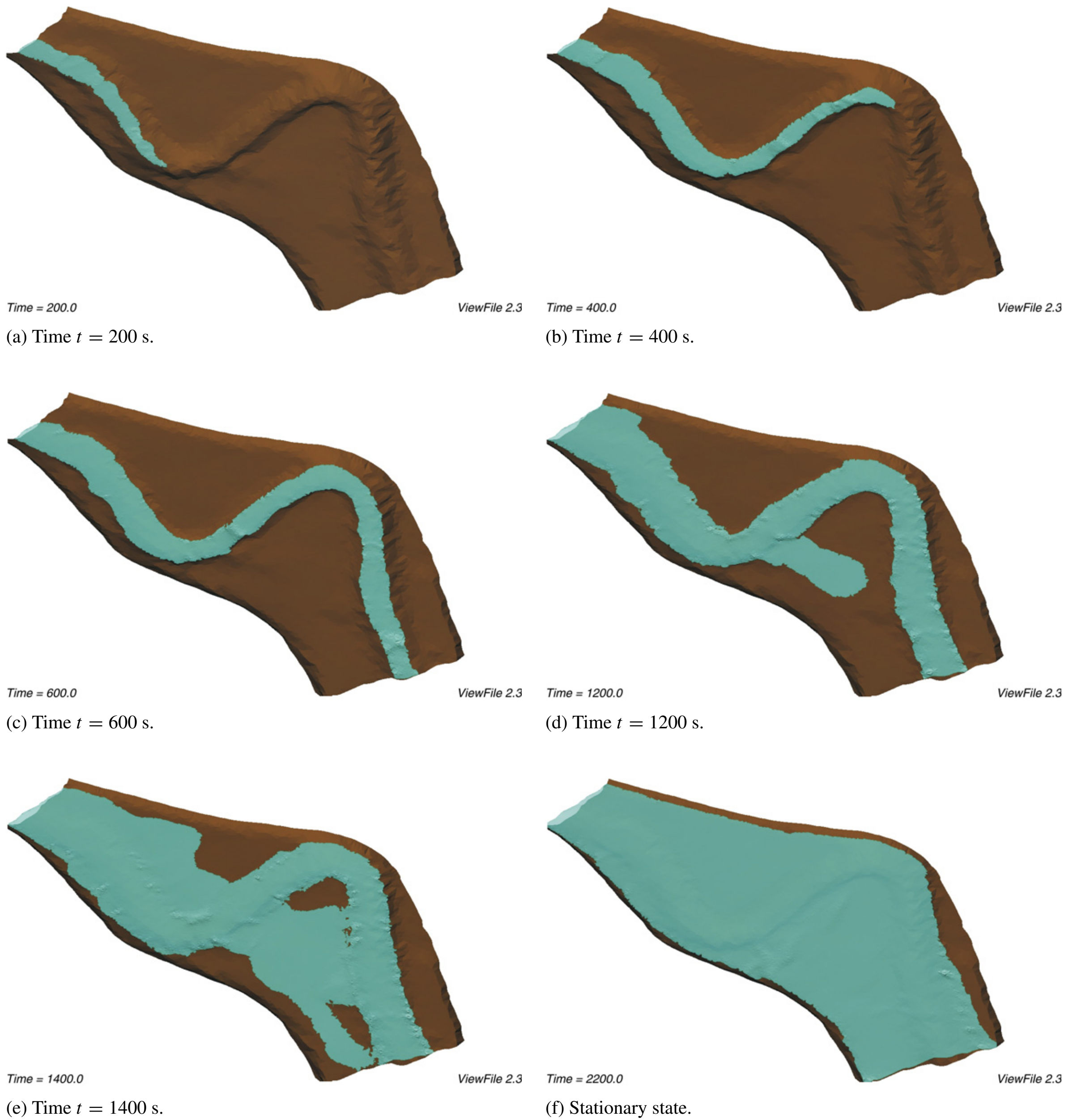
First, a mesh of 17 081 of the targeted zone using the given topographic data has been built. The mesh has been divided into three different subdomains, the first one corresponding to the river bed and the others to the margins. The friction in the river bed is parameterized by a Manning law with a coefficient of 0.023, while the coefficient corresponding to the margins is 0.035. These values have been proposed by scientists from CITEEC in agreement with data proposed in HECRAS documentation.

The experimental data correspond to an stationary state where the flux is 100 m^3/s. In order to simulate such a state, the model is run in the time interval [0, 10] hours, until a steady state is reached. As initial condition, the river bed is supposed to be dry, that is, $h(\boldsymbol{x}, 0) = 0$ m. In the upstream open boundary, Γ_0, the flux $Q(\Gamma_0, t)$ is prescribed:

$$Q(\Gamma_0, t) = \begin{cases} \dfrac{1}{1.8} t \ \mathrm{m}^3/\mathrm{s} & \text{if } 0 \le t \le 1800 \text{ s} \\ 100 \ \mathrm{m}^3/\mathrm{s} & \text{if } t > 1800 \text{ s}. \end{cases}$$

As water never exceeds the contention walls during the simulation, no boundary condition is needed in the lateral boundaries. Finally, free-boundary conditions are prescribed downstream.

The model has been run using a four node cluster. The *CFL* parameter has been fixed to 0.8. The CPU time to perform this simulation is about 40 min using the parallel SIMD implementation, while the CPU time is about 7 hours when the parallel non-SIMD implementation is used.



(a) Time $t = 200$ s.

(b) Time $t = 400$ s.

(c) Time $t = 600$ s.

(d) Time $t = 1200$ s.

(e) Time $t = 1400$ s.

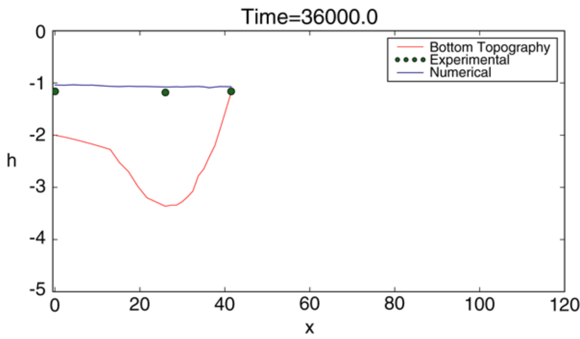
(f) Stationary state.

Fig. 7. Simulation of a flooding in River Mero.

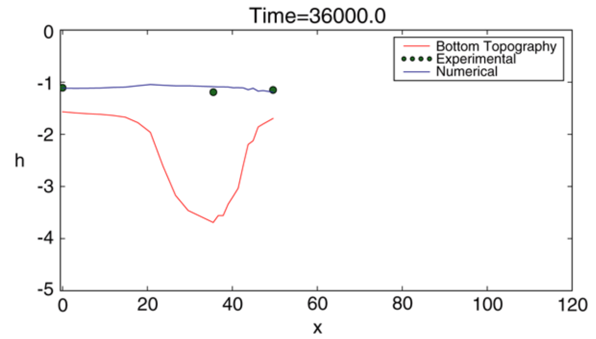
In Fig. 7, some intermediate times corresponding to the flooding are depicted. In Fig. 8(a)–(e) the free surface calculated by the numerical model (continuous line) and the experimental measurements (dotted line) for several transversal sections located along the river bed are compared at the stationary state.

In Fig. 8(f) the computed free surface (continuous line) is compared with the experimental measurements (dotted line) for a section of the domain corresponding to the axis of the river bed. As it can be seen, the position of the free surface is well-captured by the numerical model, having a maximum error of about 10 cm. The margin of error of the experimental measurements is $\pm$ 2 cm.

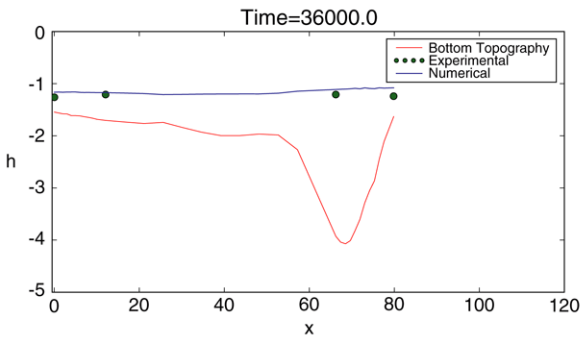
The computed velocities also agree with the measured ones, being about 1.1 m/s upstream and 0.86 m/s downstream. Animations corresponding to this experiment can be found at http://www.damflow.org.



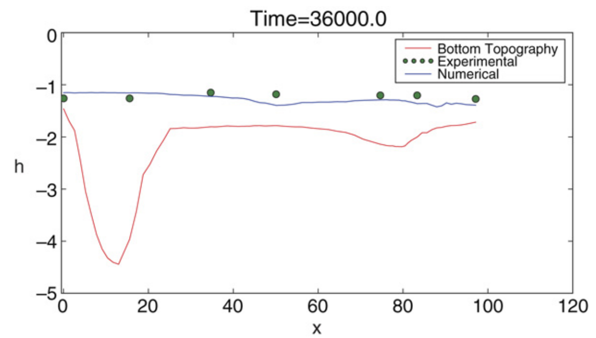
(a) Transversal Section 1.



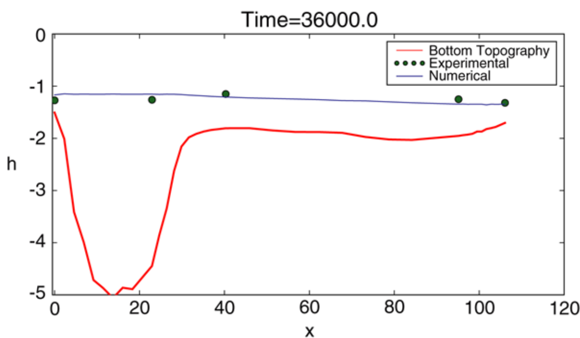
(b) Transversal Section 2.



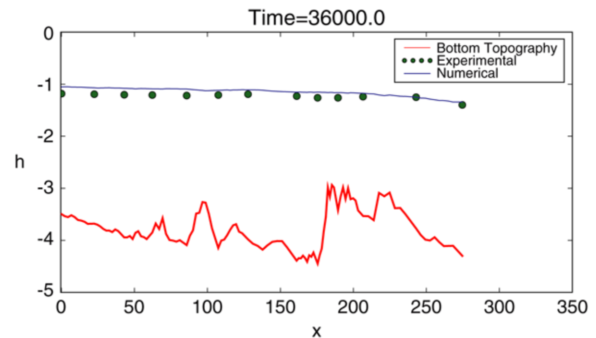
(c) Transversal Section 5.



(d) Transversal Section 6.



(e) Transversal Section 9.



(f) Longitudinal central section.

Fig. 8. Comparison between numerical and experimental elevations (stationary state).

6. Further developments

There are nowadays new SIMD units available that are much more powerful than the ones used in this work. Since they are used widely in multimedia (games mostly), GPUs ("Graphical Processor Units") have become very efficient. These units have been used in [21] to obtain some efficient implementations of numerical schemes similar to those considered here. An interesting development is thus to combine parallelization based on domain decomposition methods and SIMD parallelization based on GPUs.

Recently IBM, Toshiba and SONY have developed the "*Cell*" processor to empower a third generation console. This processor is a massive SIMD machine. On the other hand, Silicon Graphics and Cray are offering machines that include programmable hardware (FPGAs) to perform intensive computations. The development of libraries for both, Cell processor and FPGAs is also an interesting incident which helps in deploying real time geophysical flow simulators.

Acknowledgments

This research has been partially supported by the Spanish Government Research projects BFM2003-07530-C02-02, MTM2006-08075. The authors thank Enrique Peña from CITEEC for providing them with the experimental data from the laboratory experiments on River Mero. The numerical computations have been performed at the Laboratory of Numerical Methods of the University of Malaga.

References

[1] A. Bermúdez, M.E. Vázquez-Cendón, Upwind methods for hyperbolic conservation laws with source terms, Comput. & Fluids 23 (8) (1994) 1049–1071.
[2] F. Bouchut, Nonlinear stability of Finite Volume Methods for Hyperbolic Conservation Laws and Well-Balanced Schemes for Sources, Birkhäuser, 2004.
[3] M.J. Castro, J. Macías, C. Parés, A Q-Scheme for a class of systems of coupled conservation laws with source term. Application to a two-layer 1-D Shallow Water system, ESAIM-Math. Model. Num. 35 (1) (2001) 107–127.
[4] M.J. Castro, A. Ferreiro, J.A. García, J.M. González, J. Macías, C. Parés, M.E. Vázquez-Cendón, On the numerical treatment of wet/dry fronts in shallow flows: Application to one-layer and two-layers systems, Math. Comput. Modelling 42 (2005) 419–439.
[5] M.J. Castro, J.A. García, J.M. González, C. Pares, A parallel 2D Finite Volume scheme for solving systems of balance laws with nonconservative products: Application to shallow flows, Comput. Methods Appl. Mech. Engrg. 196 (2006) 2788–2815.
[6] M. Castro, J.M. Gallardo, C. Parés, High order Finite Volume schemes based on reconstruction of states for solving hyperbolic systems with nonconservative products. Applications to Shallow Water systems, Math. Comput. 75 (2006) 1103–1134.
[7] M.J. Castro, J.M. González, C. Parés, Numerical treatment of wet/dry fronts in shallow flows with a modified Roe scheme, Math. Models Methods Appl. Sci. 16 (2006) 897–931.
[8] M.J. Castro, E.D. Fernández-Nieto, A. Ferreiro, J.A. García-Rodríguez, C. Parés, High order extensions of Roe schemes for two dimensional nonconservative hyperbolic systems (2007) (submitted for publication).
[9] G. Dal Maso, P.G. LeFloch, F. Murat, Definition and weak stability of nonconservative products, J. Math. Pures Appl. 74 (1995) 483–548.
[10] M. Douze, Accéleration SIMD d'un calcul matriciel: Comparison de trois plateformes, in: IRIT/ENSEEIHT, 2005.
[11] A. Ferreiro Ferreiro, Desarrollo de técnicas de post-proceso de flujos hidrodinámicos, modelización de problemas de transporte de sedimentos y simulación numérica mediante técnicas de volúmenes finitos, Ph.D. Thesis, Universiad de Sevilla, 2006.
[12] J.M. Gallardo, C. Parés, M. Castro, On a well-balanced high-order finite volume scheme for shallow water equations with topography and dry areas, J. Comput. Phys. 227 (2007) 574–601.
[13] T. Gallouet, J.M. Herard, N. Seguin, Some approximate Godunov schemes to compute shallow-water equations with topography, Comput. & Fluids 32 (4) (2003) 479–513.
[14] B.S. Garbow, J.M. Boyle, J. Dongarra, C.B. Moler, Matrix Eigensystem Routines — EISPACK Guide Extension, in: Lecture Notes in Computer Science, vol. 51, Springer, 1977.
[15] A. Harten, J.M. Hyman, Self-adjusting grid methods for one-dimensional hyperbolic conservation laws, J. Comput. Phys. 50 (1983) 235–269.
[16] Intel© Corporation. Intel© Integrated Performance Primitives for Intel© Architecture, Reference Manual, vol. 3: Small Matrices, Document Number: A68761-005, 2004.
[17] P.G. LeFloch, Hyperbolic systems of conservation laws, the theory of classical and nonclassical shock waves, in: ETH Lecture Note Series, Birkhäuser, 2002.
[18] C. Parés, M.J. Castro, On the well-balance property of Roe's method for nonconservative hyperbolic systems. Applications to shallow-water systems, ESAIM-Math. Model. Num. 38 (5) (2004) 821–852.
[19] E. Peña, et al. A physical laboratory model and numerical model for the study of River Mero floodings (2007) (submitted for publication).
[20] P.L. Roe, Upwinding difference schemes for hyperbolic conservation laws with source terms., in: C. Carasso, P.A. Raviart, D. Serre (Eds.), Proceedings of the Conference on Hyperbolic Problems, Springer, 1986, pp. 41–51.
[21] T.R. Hagen, M.O. Henriksen, J. Hjelmervik, K.-A. Lie, How to solve Systems of Conservation Laws Numerically Using the Graphics Processor as a High-Performance Computational Engine, in: Geometric Modelling, Numerical Simulation, and Optimization: Applied Mathematics at SINTEF, vol. 207, 2007.
[22] B. Stroustrup, The C++ Programming Language, Addison Wesley, 1997.
[23] E.F. Toro, Shock-Capturing Methods for Free-Surface Shallow Flows, Wiley, 2001.
[24] I. Toumi, A weak formulation of Roe's approximate Riemann Solver, J. Comput. Phys. 102 (2) (1992) 360–373.
[25] M.E. Vázquez-Cendón, Improved treatment of source terms in upwind schemes for the shallow water equations in channels with irregular geometry, J. Comput. Phys. 148 (1999) 497–526.