An efficient solution for hazardous geophysical flows simulation using GPUs

A. Lacasta[1], C. Juez[1], J. Murillo[1], P. García-Navarro[1],

Abstract

The movement of poorly sorted material over steep areas constitutes a hazardous environmental problem. Computational tools help in the understanding and predictions of such landslides. The main drawback is the high computational effort required for obtaining accurate numerical solutions due to the high number of cells involved in the calculus. In order to overcome this problem, this work proposes the use of GPUs for decreasing significantly the CPU simulation time. The numerical scheme implemented in GPU is based on a finite volume scheme and it was validated in previous work with exact solutions and experimental data. The computational cost time obtained with the Graphical Hardware technology, GPU, are compared against single-Core (sequential) and multi-Core (parallel) CPU implementations. The GPU implementation allows to reduce the computational cost time in two orders of magnitude.

Keywords: CUDA; GPU; Landslides; Numerical modeling; Shallow Flow; Coulomb forces

1. Introduction

Landslides play an important role on the evolution of landscape and constitute an important environmental risk. They can be responsible for dramatic civilian damages and that is the reason why the building of defenses and barriers is required. The computational tools are a suitable partner for

Email addresses: alacasta@unizar.es (A. Lacasta), carmelo@unizar.es (C. Juez), javier.murillo@unizar.es (J. Murillo), pigar@unizar.es (P. García-Navarro)
[1]LIFTEC-EINA, CSIC-Universidad de Zaragoza, Spain

developing a careful design of such elements. Over recent years, reliable predictions of the spreading of granular material have been obtained (Pirulli et al., 2007; Pirulli and Mangeney, 2008; Moretti et al., 2012; Bouchut et al., 2008) and numerical results have been validated against series of experiments based on granular dry flows (Savage and Hutter, 1989; Iverson and Denlinger, 2001; Pouliquen and Forterre, 2002; Lajeunesse et al., 2004; Mangeney et al., 2010). In particular, Murillo and García-Navarro (2012); Juez et al. (2013) have recently presented a robust finite volume upwind scheme which includes the presence of steep slopes leading to obtain promising results.

Once the forecasting capability of the computational tool has been achieved another important concern is the improvement of the efficiency in terms of the computational cost. This type of geophysical flow involves the study of huge domains, such as catchments, mountains or gorges, where an accurate Digital Terrain Model is required in order to mimic the complex topography of the terrain. For this reason, implementations based on the most recent GPU hardware (Graphics Processing Unit) emerges as a promising strategy for handling this environmental and up to date problem. This hardware consists of thousands of simple arithmetic processing units originally designed to pixel-based computations. In the recent years, this technology has been successfully extended and applied to more general problems in order to accelerate them. This concept is also known as General Purpose Computing on Graphics Processing Units (GPGPU) (Owens et al., 2007).

In terms of scientific computation, the last four decades have followed Moore's law (Moore, 2003), i.e. the number of transistors on a chip increases exponentially. This integration allowed to obtain faster and faster applications, by recompiling the code for these new processors. Unfortunately, power has become the primary design constraint for chip designers, where both energy and power dissipation create a technological barrier for the integration capacity (Dreslinski et al., 2010). Nevertheless, Multi-Core micro architecture together with an adequate programming model (OpenMP is one of the most extended) brings a chance to exploit the parallelism of some parts of the code (Sharma and Gupta, 2013). Moreover, the Multi-Core paradigm has a large power consumption rate when performing small tasks and, for these purposes, Many-Core systems appear to be a very interesting option (Borkar, 2007). Many-Core architectures are those composed of smaller and not so complex cores that usually have special purposes. Industrial implementation of this solution has been obtained in the field of Graphical Processing, where several efforts have been devoted to make more powerful devices. Indeed,

this technology has been historically oriented to the very particular task of performing shading operations when rendering graphics. Following Lacasta et al. (2014), the purpose of the present work is to apply this hardware to the simulation of hazardous and very high time consuming geophysical flows.

The outline of this work is as follows: Section 2 is devoted to explain the mathematical model and numerical scheme used for modeling and solving the landslides behavior. In Section 3, the implementation on GPU is described. Section 4 gathers several experimental and realistic cases considered for testing the GPU performance. The differences on the computational cost time between the sequential, parallel and GPU strategies is discussed in Section 5. Finally in Section 6 the conclusions are summarized.

2. Mathematical model and numerical scheme

2.1. Mathematical model

The mathematical model considered for reproducing the landslides phenomenon is based on the shallow flow equations, where the general three-dimensional conservation laws are depth averaged. The pressure distribution is considered hydrostatic and as frictional terms, only Coulomb type friction forces are assumed, Juez et al. (2013). Bearing in mind these hypothesis, the 2D equations are written in global coordinates as follows:

$$\frac{\partial \mathbf{U}}{\partial t} + \frac{\partial \mathbf{F}(\mathbf{U})}{\partial x} + \frac{\partial \mathbf{G}(\mathbf{U})}{\partial y} = \mathbf{S}_\tau + \mathbf{S}_b \tag{1}$$

where

$$\mathbf{U} = (h, hu, hv)^T \tag{2}$$

are the conserved variables with h representing granular material depth parallel to the z coordinate and (u, v) the depth averaged components of the velocity vector. The fluxes are given by

$$\begin{aligned} \mathbf{F} &= \left(hu, hu^2 + \frac{1}{2} g_\psi h^2, huv\right)^T \\ \mathbf{G} &= \left(hv, huv, hv^2 + \frac{1}{2} g_\psi h^2\right)^T \end{aligned} \tag{3}$$

with $g_\psi = g \cos^2 \psi$ and ψ the direction cosine of the bed normal with respect to the vertical axis. The physical basis of this gravity projection is explained

in detail in Juez et al. (2013) and it is of utmost importance for keeping accurate numerical predictions when the simulation involves the presence of steep slopes and consequently, the direction cosines become relevant.

The term $\mathbf{S}_\tau$ notes the frictional effects in the bed, and is defined as

$$\mathbf{S}_\tau = \left(0, -\frac{\tau_{b,x}}{\rho}, -\frac{\tau_{b,y}}{\rho}\right)^T \tag{4}$$

with $\tau_{b,x}, \tau_{b,y}$ the bed shear stress in the x and y directions respectively and ρ the density of the granular mass. In this work, only dense granular flows are considered, therefore, the main rheological properties are governed by the frictional forces. These interactions between the sand grains are computed by means of the Coulomb law. This formula is based on the internal friction angle of the material, θ_b.

On the other hand, the term $\mathbf{S}_b$ is defined for gathering the information relative to the pressure force exerted over the bottom.

Thanks to the hyperbolic character of (1) it is possible to obtain a Jacobian matrix, $\mathbf{J_n}$, which is built by means of the flux normal to a direction given by the unit vector $\mathbf{n}$, $\mathbf{E_n} = \mathbf{F}n_x + \mathbf{G}n_y$,

$$\mathbf{J_n} = \frac{\partial \mathbf{E_n}}{\partial \mathbf{U}} = \frac{\partial \mathbf{F}}{\partial \mathbf{U}} n_x + \frac{\partial \mathbf{G}}{\partial \mathbf{U}} n_y \tag{5}$$

whose components are

$$\mathbf{J}_n = \begin{pmatrix} 0 & n_x & n_y \\ (g_\psi h - u^2)n_x - uvn_y & vn_y + 2un_x & un_y \\ (g_\psi h - v^2)n_y - uvn_x & vn_x & un_x + 2vn_y \end{pmatrix} \tag{6}$$

The eigenvalues of this Jacobian matrix constitute the basis of the upwind technique which is detailed in the next subsection.

2.2. Numerical scheme

System in (1) is integrated in a grid cell Ω_i and the Gauss theorem is applied, being the normal vector $\mathbf{n}$ outward to the cell Ω_i, as displayed in Figure 1

$$\frac{\partial}{\partial t} \int_{\Omega_i} \mathbf{U} d\Omega + \oint_{\partial\Omega_i} \mathbf{E_n} dl = \int_{\Omega_i} (\mathbf{S}_\tau + \mathbf{S}_b) d\Omega \tag{7}$$

The second integral in (7) can be explicitly expressed as a sum over the cell edges,

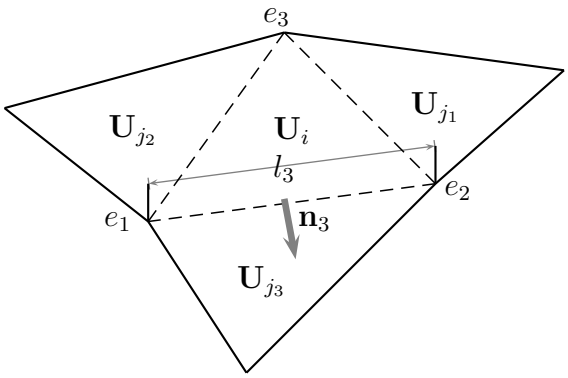

Figure 1: Cell parameters

$$\frac{\partial}{\partial t}\int_{\Omega_i}\mathbf{U}d\Omega + \sum_{k=1}^{NE}(\mathbf{E_n})_k l_k = \sum_{k=1}^{NE}\mathbf{S}_{\mathbf{n}\tau}l_k + \sum_{k=1}^{NE}\mathbf{S}_{\mathbf{n}b}l_k \tag{8}$$

where l_k is the corresponding edge length and $\mathbf{S}_{\mathbf{n}b}$ and $\mathbf{S}_{\mathbf{n}\tau}$ are suitable integrals of the bed slope and friction source terms (Juez et al., 2013),

$$\mathbf{S}_{\mathbf{n}b} = \left(0,\ -g_\psi h\frac{\partial z}{\partial x}n_x, -g_\psi h\frac{\partial z}{\partial y}n_y\right)^T \tag{9}$$

$$\mathbf{S}_{\mathbf{n}\tau} = \left(0,\ \rho g_\psi h\tan\theta_b n_x, \rho g_\psi h\tan\theta_b n_y\right)^T \tag{10}$$

where n_x and n_y are the components of the unit vector $\mathbf{n}$ of each edge of each computational cell.

The numerical scheme is constructed by defining a local linearization in terms of an approximate Jacobian matrix $\widetilde{\mathbf{J}}$ at each k edge between neighboring cells defined through the normal flux $\mathbf{E_n}$

$$(\delta\mathbf{E_n})_k = \widetilde{\mathbf{J}}_{\mathbf{n},k}\delta\mathbf{U}_k \tag{11}$$

with $\delta(\mathbf{E_n})_k = (\mathbf{E}_j - \mathbf{E}_i)_{\mathbf{n}_k}$, $\delta\mathbf{U}_k = \mathbf{U}_j - \mathbf{U}_i$, and $\mathbf{U}_i$ and $\mathbf{U}_j$ the initial values at cells i and j sharing edge k.

From this approximate Jacobian matrix a set of three real eigenvalues $\widetilde{\lambda}_k^m$ and eigenvectors $\widetilde{\mathbf{e}}_k^m$ are obtained. The vector of conserved variables, $\delta\mathbf{U}$, is projected onto the matrix eigenvectors basis, $\widetilde{\mathbf{P}}$, as

$$\delta\mathbf{U}_k = \widetilde{\mathbf{P}}_k\widetilde{\mathbf{A}}_k = \sum_{m=1}^{3}(\widetilde{\alpha}\widetilde{\mathbf{e}})_k^m \tag{12}$$

with

$$\widetilde{\mathbf{P}}_k = (\mathbf{e}^1, \mathbf{e}^2, \mathbf{e}^3)_k \qquad \widetilde{\mathbf{A}}_k = \begin{pmatrix} \alpha^1 & \alpha^2 & \alpha^3 \end{pmatrix}_k^T \tag{13}$$

The source terms are also projected onto the matrix eigenvectors basis, $\widetilde{\mathbf{P}}$, to guarantee the exact equilibrium between fluxes and source terms, i.e. well-balanced property, Murillo and García-Navarro (2010).

$$(\mathbf{S}_{\mathbf{n},b}, \mathbf{S}_{\mathbf{n},\tau})_k = \widetilde{\mathbf{P}}_k \widetilde{\mathbf{B}}_k = \sum_{m=1}^{3} \left(\widetilde{\beta}\widetilde{\mathbf{e}}\right)_k^m \tag{14}$$

with

$$\widetilde{\mathbf{B}}_k = \begin{pmatrix} \beta^1 & \beta^2 & \beta^3 \end{pmatrix}_k^T \tag{15}$$

Additionally, it is worth remarking that two numerical fixes are also needed for computing physical solutions (Murillo and García-Navarro, 2012): they are necessary for avoiding unphysical solutions by means of decreasing the wave source strengths, $\mathbf{B}_k$.

Gathering all the previous information the volume integral in the cell at time t^{n+1} is expressed as

$$\mathbf{U}_i^{n+1} = \mathbf{U}_i^n - \sum_{k=1}^{NE} \sum_{m=1}^{3} (\widetilde{\lambda}^{-}\widetilde{\alpha} - \widetilde{\beta}^{-})_k^m \mathbf{e}_k^m l_k \frac{\Delta t}{A_i} \tag{16}$$

The minus sign in (16) implies that only the incoming waves are considered for updating the values of each cell. The numerical scheme is only computed over the cells where there is a sand depth.

In order to guarantee stability the time step Δt has to be taken small enough so that there are no interactions of waves between neighboring cells.

$$\Delta t \leq CFL\ \Delta t^{\widetilde{\lambda}} \qquad \Delta t^{\widetilde{\lambda}} = \frac{\min(\chi_i, \chi_j)}{\max|\widetilde{\lambda}^m|} \tag{17}$$

with CFL=1/2 in the case of triangular unstructured grids. The term χ_i is the relevant distance, which in a 2D framework must consider the volume of the cell i and the length of the shared k edges (Murillo and García-Navarro, 2010),

$$\chi_i = \frac{A_i}{\max_{k=1,NE} l_k} \tag{18}$$

Due to the fact that the numerical scheme is controlled by a explicit global time step, the only way of reducing the computational cost is by means of using a high performance computing technique: the implementation of the numerical scheme on a GPU, as it is explained in the next section.

3. Implementation of the numerical scheme

The implementation of the model follows a time-stepping process where, until the simulation time is achieved, the numerical scheme (16) is applied over the whole domain. The way this process is performed is illustrated in Figure 2. The main operations of the numerical scheme are displayed in green and pink. The calculation of the fluxes is performed looping edgewise and the updating process as well as the boundary calculation are performed looping over the cells. Both processes go from 1 to n_{edges} and n_{cells} respectively.

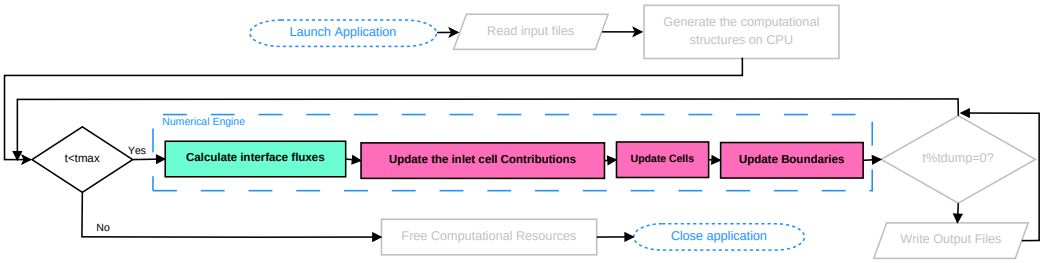

Figure 2: Flowchart of the simulation process. The green function is performed looping over cell edges, while the pink is a loop over the cells.

In order to make the process faster, some of the parts can be paralellized or even adapted to perform the operations in the GPU. For successing in this task, we know that most of the processors are designed with multi-threading capabilities. Despite these capabilities, it is not direct to perform operations using every single core, since the calculus operations have to be shared among all the cores. Figure 3 displays a typical iterative process using a single core of a Multi-Core processor. It represents how one core is activated and it has all the workload while the rest are not active and they do not participate in the calculations.

When possible, the iterative process may be split into different sub-processes allowing the processing of different elements at the same time by different processing units. This is the concept of parallelizetion. It can be directly applied when the loop contains straight-line code (a single basic block

```
void generateOperation(int *A){
 for(i=0;i<48;i++){
     A[i]= ...
 }
}
```

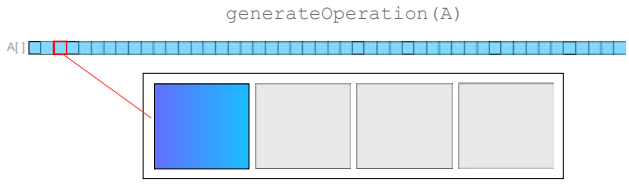

Figure 3: Sequential iteration over vector `A[]`

with no jumps) and when there is no data dependency between iterations (element $i-1$ is not required for the element i). If data flow does not satisfy the previous condition, the manner of processing the elements to make them parallelizable must be re-structured.

OpenMP is a very useful standard to perform shared-memory parallelization (Chapman et al., 2007). This standard implements parallel primitives in many programming languages such as C or Fortran making it very simple to implement a parallel solution in a reasonable time. The main disadvantage of this kind of parallelization is that the Multi-Core processors have a physical boundary, since the number of transistors which can be used within the space of a chip are limited. Contrary, with the Many-Core architectures devices, such as the GPUs, a lot of specialized processing units allow to make many more operations than in the multi-core unit. An example of a parallel loop using OpenMP is displayed in Figure 4.

```
void generateOperation(int *A){
 #pragma omp parallel for
 for(i=0;i<48;i++){
     A[i]= ...
 }
}
```

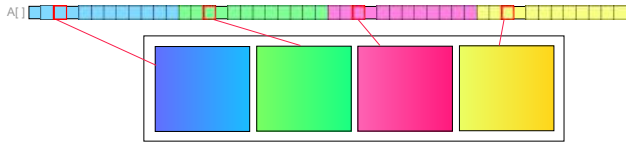

Figure 4: Parallel iteration over vector `A[]` using 4 cores

Many-Core processors provide a hardware architecture to perform a large number of independent operations in a parallel manner. The main difference between Multi-Core threads and Many-Core threads is the *lightweight* of the second solution as well as the hardware support of a larger number of the latter (> 1000s). Unlike the Multi-Core thread, where each thread processes a group of elements, in the Many-Core paradigm, each thread will process just one element. Moreover, the creation and management of the Many-Core threads as well as their capacity to perform operations are simpler than in the case of the common Multi-Core thread.

CUDA (Compute Unified Device Architecture) was released in order to perform this kind of parallelization on NVIDIA GPUs (NVIDIA, 2011). It provides both a software model and a set of compilation tools that support the NVIDIA Many-Core GPUs. An example of a CUDA instance for the previous loop is displayed in Figure 5.

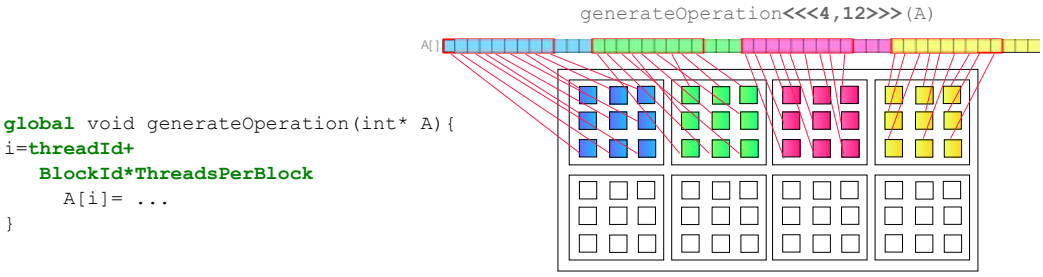

Figure 5: Massively parallel iteration over vector `A[]` using a Many-Core architecture.

The improvement on the implementation of the numerical solver will be achieved by means of the translation of the processes sketched in Figure 2 into both solutions: OpenMP for the Shared-Memory level parallelism in Multi-Core hardware and the CUDA based solution for the Many-Core GPUs architecture.

3.1. Implementation on GPU

The GPU contains a large number of processors working all together applying the same operation over different elements. In order to obtain high performance in the GPU implementation it is necessary to understand the way the model works. NVIDIA GPUs are formed by Streaming Multiprocessors (SMs). Each Multiprocessor is composed by Streaming Processors (SPs) that represent the minimum processing unit. In the case of the Tesla Series GPUs, there are between 14 and 16 SMs and each one is composed by 32 SPs (Glaskowsky, 2009). From the software point of view, there are groups of blocks that contain threads. More specifically, each block contains *blockDim* threads and the number of blocks *gridDim* must be larger than the number of elements (n_{elem}) to be processed ($n_{elem} \leq blockDim * gridDim$). Each block will be processed by a SM in groups of 32 elements which form a *warp* taking into account that each Streaming Processor will process each element of the warp. More details about CUDA and NVIDIA GPUs can be found in (NVIDIA, 2011).

Implementation for GPU has been developed using the approach of Lacasta et al. (2014) for unstructured meshes and adapting the required elements into this new model. A general overview of the CUDA implementation of the simulation process described in Figure 2 is highlighted in Listing 1.

Listing 1: Overview of the CUDA implementation.

```
...
// Configuration of the parameters
Threads=256;
edgeBlocks=nEdge/Threads;
cellBlocks=nCell/Threads;
while(t<tmax){
  // Calculate the fluxes
  calculateEdgeFluxes<<<edgeBlocks,Threads,0,executionStream>>>(...);
  // Establish the minimum dt obtaining the ID of the
  // minimum dt
  // (*) Explained at Listing 3
  cublasIdamin(...,nEdge,vDt,1,id);
  // And assign it
  newDt<<<1,1,0,executionStream>>>(dt,vDt,id);
  // Update the elapsed time (in GPU)
  updateT<<<1,1,0,executionStream>>>(cuda_t,dt);
  // Retrieves the value of t to CPU
  cudaMemcpy(t,cuda_t,sizeof(double),cudaMemcpyDeviceToHost);
  // Update the cell values
  assignFluxes<<<cellBlocks,Threads,0,executionStream>>>(...);
  // Verify whether it is neccessary to dump data and
  // if so, process it.
  // (*) Detailed in Listing 4
  if(t<t_dump){
    // Transfer data from GPU to CPU and write it
    // into an output file
  }
}
```

As it was justified in Juez et al. (2013) the topology of the mesh plays an important role on the quality of the numerical results. Only unstructured meshes avoid the presence of preferential directions. Because of this necessity, it is not straightforward to obtain an efficient solution for the GPU processing. GPUs are designed to work efficiently with ordered information. This means that, those structures without ordered pattern may reduce the overall performance of the implementation (Lacasta et al., 2014). Moreover, special attention on those parts that are not intrinsic parallel is required (such as the selection of the global time step).

To obtain an efficient solution three special considerations are highlighted.

- Using Structure of Arrays (SoA) instead of Arrays of Structures (AoS) to store data. Because of the manner of processing the elements, SoA provides improvements when accessing consecutive elements and this will be maximized if conserved variables (h, hu, hv) are stored consecutively as $h_0...h_{ncells}hu_0...hu_{ncells}hv_0...hv_{ncells}$ instead of $h_0hu_0hv_0...h_nhu_nhv_n$. When a thread within a warp reads the variable $h(i)$ it is likely that the following thread requires $h(i+1)$. In other words, this approach improves the data locality and allows to the GPU to obtain coalesced memory access.

- To maximize coalescence when accessing over cells it is required to reorder the mesh (Lacasta et al., 2014). It is applied by means of modifying the way the cells are connected among themselves. One strategy is to fit the connectivity matrix into a banded matrix. Figure 6 shows the effect of applying a reordering method in the connectivity matrix when using unstructured meshes. In order to obtain the previous ordering, bandwidth reduction methods are very useful. RCM algorithm is one of them. These methods have been traditionally used to obtain better performance in iterative methods by means of reducing the bandwidth of the matrix. Similarly, the idea can be applied to the connectivity matrix and this provides a reordered cells indexing that improves the spatial locality and provides coalescence when accessing by cells.

- The ordering when processing over edges. In the previous point, RCM achieves coalesced access when processing over cells, but it is not enough to process over edges. Actually, it is likely that when calculating by edges (when accessing the two neighboring cells of the edge) this reordered mesh may introduce penalty (Lacasta et al., 2014). To avoid this, reordering the edges (reordering the pair of cells) will strongly increase the performance of the GPU memory access when accessing over cells to the primitives values. The edge ordering is graphically detailed in Figure 7. This reordering will allow to process the pair of cells $(0, 1)$ for the thread which process edge 0 and the pair $(0, 2)$ for the thread which process edge 1 instead of the pair $(19, 23)$ that would be processed in the edge 1 without the edge ordering.

When applying all these previous improvements it is possible to achieve

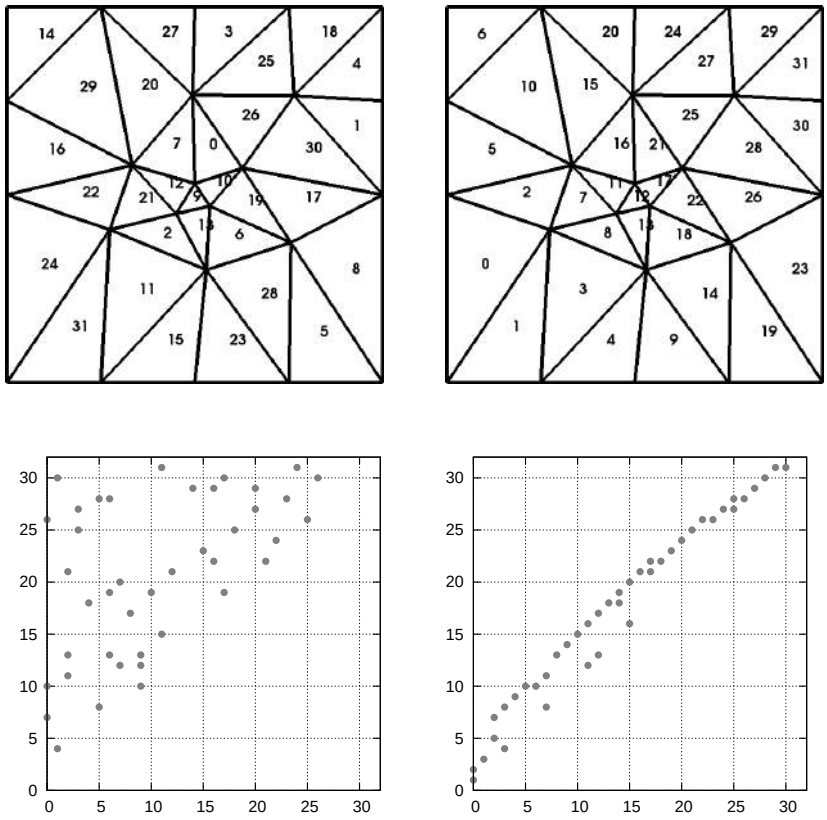

Figure 6: Upper: Unstructured mesh (left) and unstructured mesh postprocessed with RCM algorithm (right). Lower: Connectivity matrix for the original unstructured mesh (left) and for the reordered unstructured mesh (right). (Note that there exists a point if $m(i, j) = 1$).

great speed-ups comparing with a sequential or a multi-core implementation of the numerical scheme. This is discussed in Section 5.

4. Test cases

This section is devoted to describe the test cases considered as benchmark for evaluating the performance of GPU implementation with respect to Single-Core and Multi-Core CPU implementations. The first three tests are based on experimental works previously used for validating the numerical scheme (Juez et al., 2013). The last test case considered constitutes a

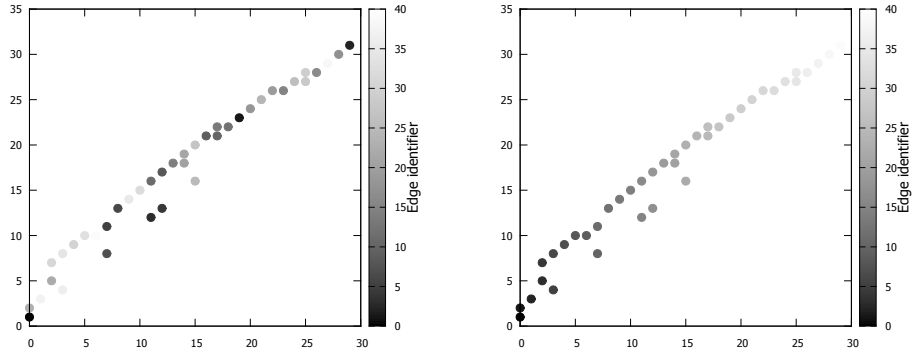

Figure 7: Unstructured mesh RCM processed edges ordering. Original ordering (left), `qsort` postprocesed ordering (right)

practical application, based on a real topography which is located in Spain (García-Ruiz et al., 2005).

All the numerical results have been computed using unstructured meshes to avoid numerical effects associated to the presence of preferential flow propagation directions as it was justified in Juez et al. (2013). Additionally, the CFL imposed in the numerical scheme has been equal to 0.5 in all the cases.

4.1. Spreading of cylindrical granular mass

The numerical simulation described in this test case is related with the experimental work developed by Lajeunesse et al. (2004). It has been chosen due to the simplicity of the initial configuration. During this experiment the spreading of a granular mass, originally retained within a cylinder, was recorded over a flat plane. The material was characterized with an internal friction angle θ_b=32^o. The test case considered is Test A, where the initial height of the sand was 39.48 mm and the radius was 70.5 mm. The number of cells involved in the simulation has been 320000.

Figure 8 displays a 3D view of the temporal evolution of the sand depth and velocity, from the beginning, when the mass is enclosed within the cylinder up to the final stage when all the granular mass has achieved an equilibrium condition and it was stopped.

On the other hand, if we try to achieve a CPU simulation with a computational cost time similar to the one provided by the GPU, the mesh has to

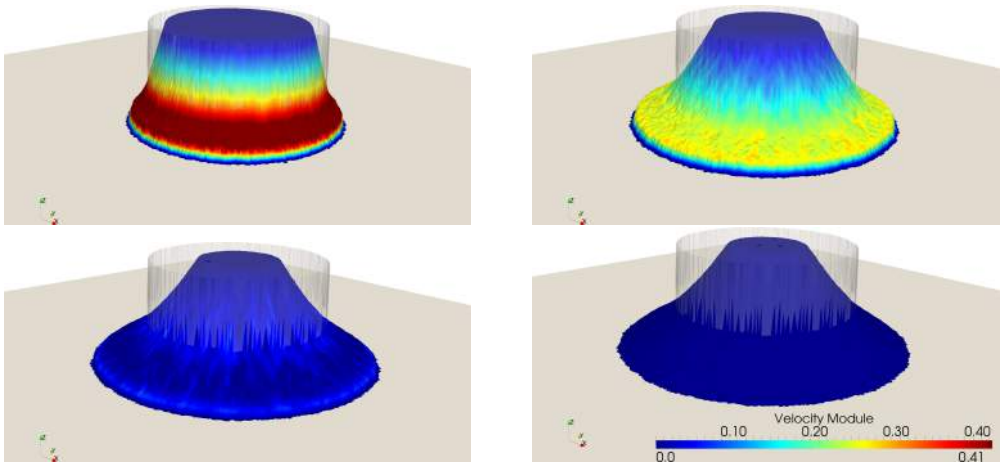

Figure 8: Evolution of the Sand Depth with the velocity (m/s) at time $t = 0.05s$. (top-left), $t = 0.15s$. (top-right), $t = 0.25s$. (bottom-left) and $t = 0.4s$. (bottom-right). The transparent element corresponds to the initial condition.

be consequently coarser and the results suffer from a lack of accuracy. This idea is showed in Figure 9, where the coarse mesh is composed by 16000 cells.

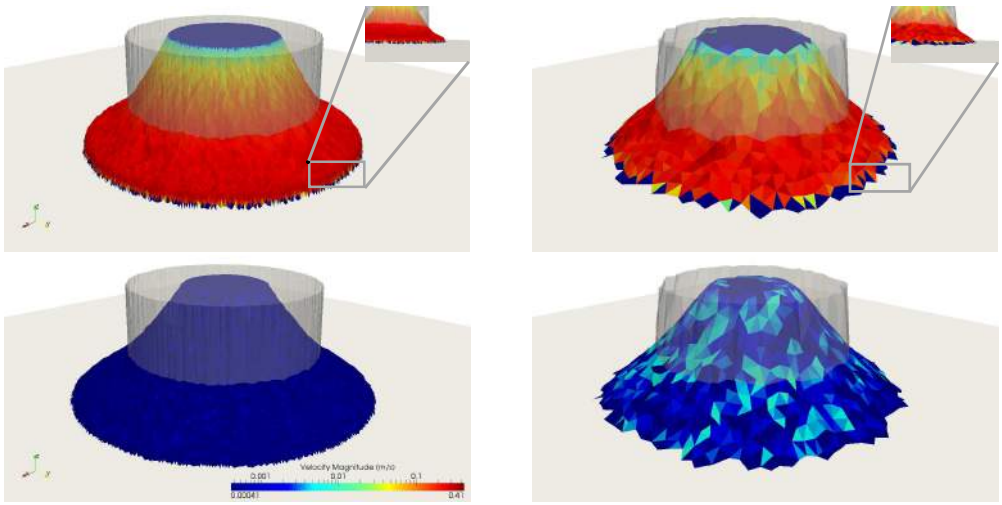

Figure 9: Comparison of fine mesh (left) and coarse mesh (right) of velocity magnitude at $t = 0.1s$. (upper) and $t = 0.4s$. (lower)

306 *4.2. Spreading of a granular mass including the presence of an obstacle*

307 The development of this experiment was motivated by the presence of
308 obstacles in real events, i.e. avalanches. The presence of these obstacles
309 causes the birth and propagation of quick moving shocks. The experimental
310 work was carried out by Juez et al. (2014). The experimental setup consists
311 of a rough inclined plane with a changing slope and with an obstacle in the
312 middle of the flow path. The initial condition is given by a semi spherical
313 cap full of sand. When this cap is pulled out, the granular material is free to
314 evolve downwards from its original position until surrounding the obstacle.
315 The internal friction angle of the material was characterized as $\theta_b = 26^o$ and
316 the number of cells included in the simulation was 460000.

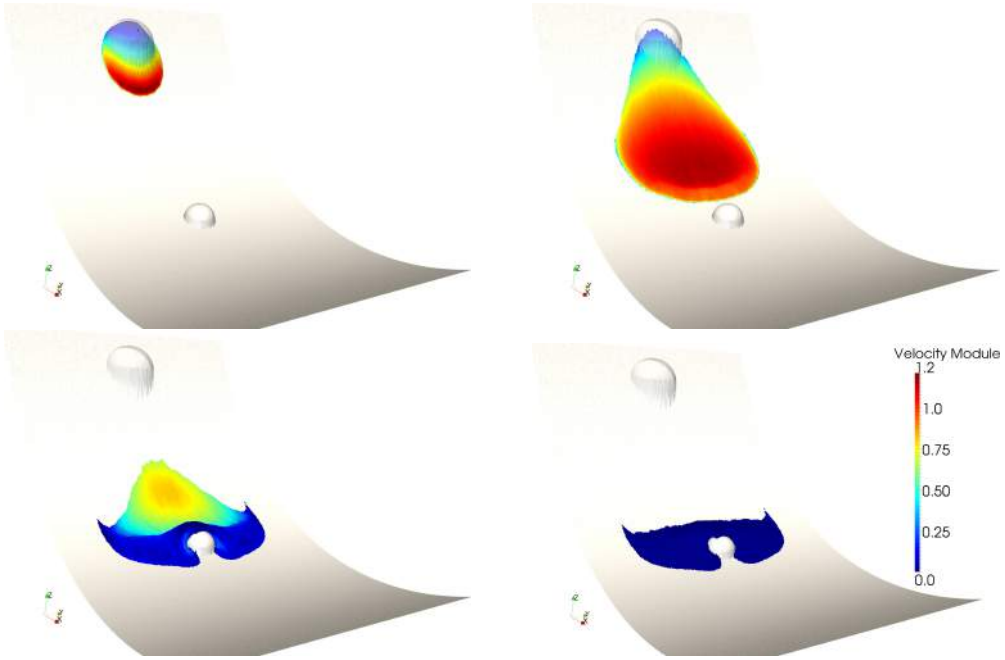

Figure 10: Evolution of the Sand Depth with the velocity (m/s) at time $t = 0.05s$. (top-left), $t = 0.4s$. (top-right) $t = 0.8s$. (bottom-left) and $t = 1.2s$. (bottom-right). The transparent element corresponds to the initial condition.

317 The temporal evolution of the granular flow is displayed in Figure 10.
318 The front of the avalanches moves quickly until reaching the obstacle. Then,
319 the flow is divided in two directions up to achieve a static equilibrium stage.

4.3. Spreading of a granular mass over a parabolic chute

This test case is another step in complexity, since the bed slope changes in the longitudinal and transversal direction. The original experiment was carried out by (Gray et al., 1999) and it was based on a semi spherical cap full of sand which was suddenly removed over a parabolic chute. The material was characterized with a internal friction angle $\theta_b = 30^o$. The number of cells considered in the calculation has been 670000.

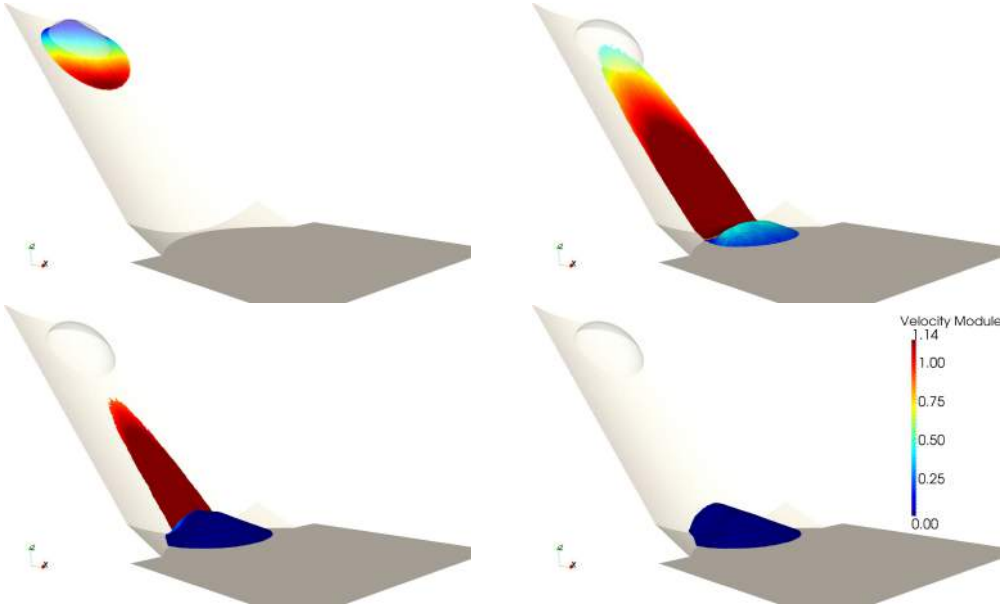

Figure 11: Evolution of the depth with the velocity (m/s) at time $t = 0.2s$. (top-left), $t = 1.4s$. (top-right), $t = 2s$. (bottom-left) and $t = 2.6s$. (bottom-right). The transparent element corresponds to the initial condition.

Numerical results are plotted in Figure 11. The material flows quickly downstream until it reaches the horizontal area. At that position, the velocity of the front is reduced by the lower bed slope. Hence, the granular mass starts to spread transversally and finally, an uneven surface level is obtained at the equilibrium stage.

4.4. Landslide in a practical application

The final test is based on a real topography of a catchment (García-Ruiz et al., 2005). The Arnas catchment is located in the northern Spain

Pyrenees, in the Borau valley, and has a surface of 2.84 km^2, ranging in altitude from around 900 to 1340 meters. Geologically, the catchment lies over Eocene flysch formations and has suffered land use and coverage changes in recent decades, generating a mixed vegetation cover which ranges from forest patches, dense and open shrubs, grassland cover and bare land. The assumption made by the authors is that the soil is composed by poorly sorted material and the idea is to verify the maximum run out and potential consequences of a massive mobilization of that material. Figure 12 displays the fixed bed rock and the soil mass which can be potentially mobilized. Due to the large dimensions of the catchment, the number of cells involved in the calculation is over 869000, making this type of phenomena a suitable candidate for GPU strategy.

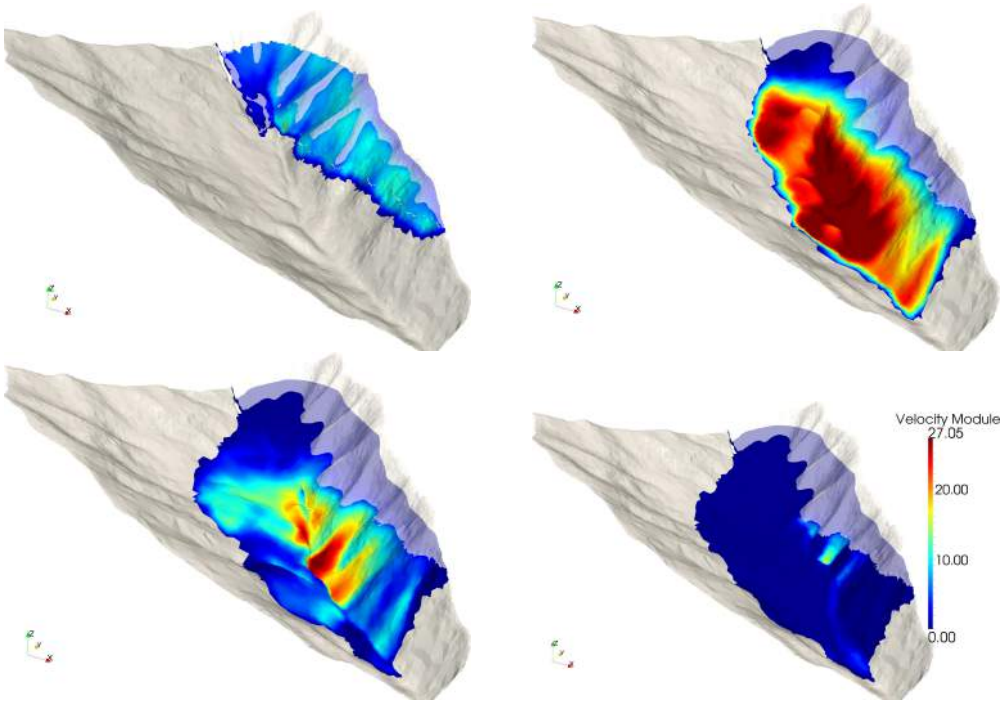

Figure 12: Evolution of the depth with the velocity (m/s) at time $t = 1s$. (top-left), $t = 2s$. (top-right), $t = 4s$. (bottom-left) and $t = 54s$. (bottom-right). The transparent element corresponds to the initial condition.

5. Results and computational load

As it has been mentioned before, Tests 1, 2 and 3 were previously used by the authors for CPU computations. In such situations, the computational cost was huge in comparison with the simulation time, ≈ 100 times the physical time. The cause of this problem has a twofold nature: first, the number of cells involved in the computation is large, since the scale of interest in these tests was in the order of millimeters. Additionally, due to the explicit numerical scheme considered, there is a restriction in the time step used for updating the solution. In these test cases, the time step is highly penalized due to the small area of the cells and the fast flows, as it has been explained in 17. On the other hand, the last test case does not require such high refinement because of the uncertainty of the data acquisition. Indeed, the mesh has been designed using the most accurate available LIDAR data for the topography, with the resolution of 5mx5m.

In order to analyze the computational load of the numerical engine as well as the gain obtained using both Single-Core and Multi-Core approaches, a sequential version using an *Intel Core i7 3770k@3.5 GHz* is compared against an OpenMP (4 Threads) parallel version and a GPU version without taking into account the improvements proposed on the paper (i.e. without using Structure of Arrays, the RCM technique for cells and edges) and against an optimized version running on GPU. Both GPU versions, non-optimized and optimized, have been executed with a *NVIDIA Tesla c2075* GPU. All the implementations have been tested in the previous four cases and the computational cost time as well as the speed-ups of the parallel versions are highlighted in Table 1 and in Figure 13.

Taking into account both factors, small time-step size and high number of cells, the computational cost time for the sequential version of the numerical engine for tests cases 1, 2, and 3 is three orders of magnitude larger than the physical time. On the other hand, the parallel Multi-Core version, with 4 cores of the same CPU, can accelerate the computation between 2.25 and 2.65 times, depending on the test chosen. The GPU improves the simulation cost between 34.88 and 49.40 times compared with the sequential version. Moreover, with the optimized GPU version, the computational cost time is smaller and a speed-up between 49.96 and 59.85 is obtained.

The main reason for differences on the speed-up of cases 1, 2 and 3 against test case 4 is that the number of calculations in the latter is higher compared with the number of calculations for the other cases, i.e. there are more cells

384 which have a soil volume able of being mobilized. Consequently, the more
385 cells are involved in the calculus, the capabilities of the GPU are exploited in
386 a more more efficient way, leading to larger speed-ups (Lacasta et al., 2014).

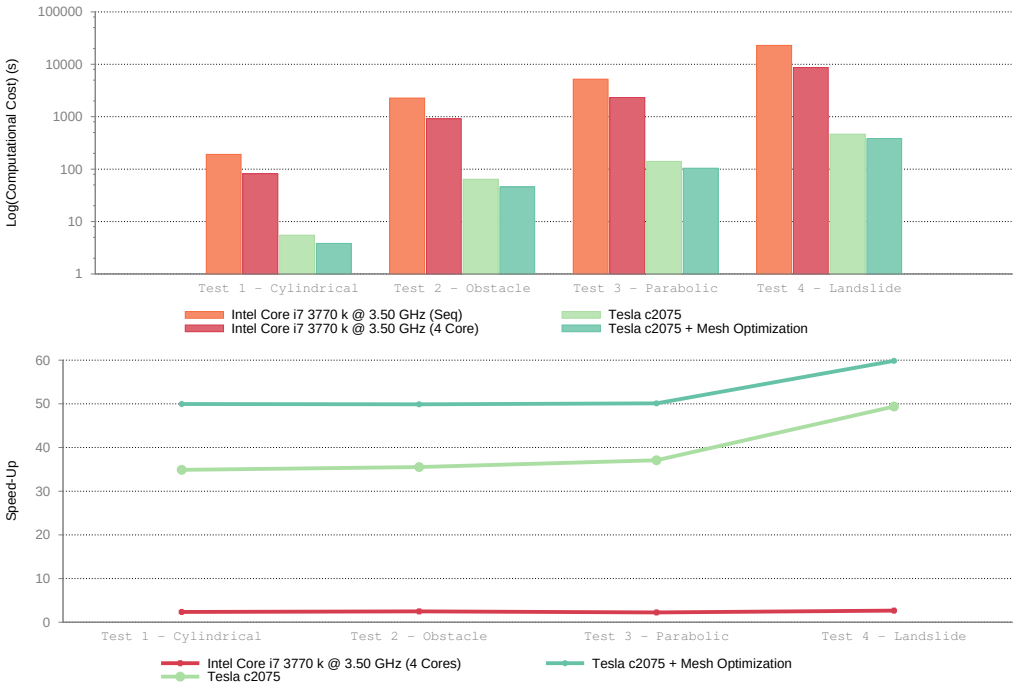

Figure 13: Computational time (upper) using logarithmic scale and speed-up (lower) for the compared implementations

		Seq.	4 Cores		GPU Std.		GPU Opt	
Case	n_{cells}	$t(s)$	$t(s)$	s_{up}	$t(s)$	s_{up}	$t(s)$	s_{up}
Test 1	319354	191.15	81.63	**2.34**	5.48	**34.88**	3.83	**49.96**
Test 2	458684	2274.13	912.94	**2.49**	64.03	**35.52**	45.59	**49.89**
Test 3	670940	5217.70	2319.48	**2.25**	140.68	**37.09**	104.12	**50.11**
Test 4	869149	22929.15	8657.59	**2.65**	464.16	**49.40**	383.13	**59.85**

Table 1: Detail of computational cost and speed-up for the compared implementations using the four cases.

6. Conclusions

A finite volume scheme for modeling landslides has been implemented on GPU. Unstructured meshes have been considered, since this grid topology is the only one which avoid misleading preferential flow directions. The use of unstructured meshes generates difficulties when moving to the GPU architecture since an optimization process is required. In this work, the RCM technique is described for overcoming this difficulty.

The computational cost times have been compared with those obtained when considering Single-Core and Multi-Core processors. The GPU implementation and specially, the application of cell and wall ordering algorithms, have driven to obtain noticeable improvements in the speed-up of the test cases. The GPU use allows to obtain speed-ups ranging from 49-59, depending on the test case.

Future work may include the research of the implementation of these methods using distributed computing. These ideas can also be extended to develop a multi-GPU solution where the limitation on the problem size can be relaxed.

References

Borkar, S., 2007. Thousand core chips: A technology perspective, in: Proceedings of the 44th Annual Design Automation Conference, ACM, New York, NY, USA. pp. 746–749.

Bouchut, F., Fernández-Nieto, E.D., Mangeney, A., Lagrée, P.Y., 2008. On new erosion models of Savage-Hutter type for avalanches. Acta Mechanica 199, 181–208.

Chapman, B., Jost, G., Pas, R.v.d., 2007. Using OpenMP: Portable Shared Memory Parallel Programming (Scientific and Engineering Computation). The MIT Press.

Dreslinski, R., Wieckowski, M., Blaauw, D., Sylvester, D., Mudge, T., 2010. Near-threshold computing: Reclaiming moore's law through energy efficient integrated circuits. Proceedings of the IEEE 98, 253–266.

García-Ruiz, J., Arnaez, J., Beguería, S., Seeger, M., Marti-Bono, C., Regues, D., Lana-Renault, N., White, S., 2005. Runoff generation in an inten-

sively disturbed, abandoned farmland catchment, Central Spanish Pyrenees. Catena 59, 79–92.

Glaskowsky, P.N., 2009. Nvidia's fermi: The first complete GPU computing architecture. A white paper prepared under contract with NVIDIA Corporation , 126.

Gray, J., Wieland, K., Hutter, K., 1999. Gravity-driven free surface flow of granular avalanches over complex basal topography. Proc. Royal Soc. London Ser. A, 455, 1841.

Iverson, R., Denlinger, R., 2001. Flow of variably fluidized granular masses across three-dimensional terrain. A Coulomb mixture theory. Journal of Geophysical Research 106, 537–552.

Juez, C., Caviedes, D., Murillo, J., García-Navarro, P., 2014. Experimental study of 2D granular flow over a complex topography with obstacles using a consumer-grade RGB-D sensor. Under review in Journal of Fluid Mechanics 73, 177–197.

Juez, C., Murillo, J., García-Navarro, P., 2013. 2D simulation of granular flow over irregular steep slopes using global and local coordinates. Journal of Computational Physics 255, 166–204.

Lacasta, A., Morales-Hernández, M., Murillo, J., García-Navarro, P., 2014. An optimized GPU implementation of a 2d free surface simulation model on unstructured meshes. Advances in Engineering Software 78, 1 – 15.

Lajeunesse, E., Mangeney-Castelnau, A., Villote, J.P., 2004. Spreading of a granular mass on a horizontal plane. Physics of Fluids 16, 2371–2381.

Mangeney, A., Roche, O., Hungr, O., Mangold, N., Faccanoni, G., Lucas, A., 2010. Erosion and mobility in granular collapse over sloping beds. Journal of Geophysical Research 115, F03040.

Moore, G., 2003. No exponential is forever: but "forever" can be delayed! [semiconductor industry], in: Solid-State Circuits Conference, 2003. Digest of Technical Papers. ISSCC. 2003 IEEE International, pp. 20–23 vol.1.

Moretti, L., Mangeney, A., Capdeville, Y., Stutzmann, E., Huggel, C., Schneider, D., Bouchut, F., 2012. Numerical modeling of the Mount Steller

450 landslide flow history and of the generated long period seismic waves. Geo-
451 physical Research Letters 39, L16402.

452 Murillo, J., García-Navarro, P., 2010. Weak solutions for partial differential
453 equations with source terms: Application to the shallow water equations.
454 Journal of Computational Physics 229, 4327–4368.

455 Murillo, J., García-Navarro, P., 2012. Wave Riemann description of friction
456 terms in unsteady shallow flows: Application to water and mud/debris
457 floods. Journal of Computational Physics 231, 1963–2001.

458 NVIDIA, 2011. NVIDIA CUDA C Programming Guide.

459 Owens, J.D., Luebke, D., Govindaraju, N., Harris, M., Krger, J., Lefohn, A.,
460 Purcell, T.J., 2007. A survey of general-purpose computation on graphics
461 hardware. Computer Graphics Forum 26, 80–113.

462 Pirulli, M., Bristeau, M., Mangeney-Castelnau, A., Scavia, C., 2007. The
463 effect of the earth pressure coefficients on the runout of granular material.
464 Environmental Modelling and Software 22, 1437–1454.

465 Pirulli, M., Mangeney, A., 2008. Results of Back-Analysis of the Propaga-
466 tion of Rock Avalanches as a Function of the Assumed Rheology. Rock
467 Mechanics and Rock Engineering 41, 59–84.

468 Pouliquen, O., Forterre, Y., 2002. Friction law for dense granular flows:
469 application to the motion of a mass down a rough inclined plane. Journal
470 of Fluid Mechanics 453, 133–151.

471 Savage, S., Hutter, K., 1989. The motion of a finite mass of granular material
472 down a rough incline. Journal of Fluid Mechanics 199, 177–215.

473 Sharma, S., Gupta, K., 2013. Parallel performance of numerical algorithms
474 on multi-core system using openmp, in: Meghanathan, N., Nagamalai, D.,
475 Chaki, N. (Eds.), Advances in Computing and Information Technology.
476 Springer Berlin Heidelberg. volume 177 of *Advances in Intelligent Systems*
477 *and Computing*, pp. 279–288.