# 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 predictions of such landslides. The main drawback is the high computational effort required for obtaining accurate numerical solutions. In order to overcome this problem, this work proposes the use of GPUs for decreasing significantly the 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 times obtained with the Graphical Hardware technology are compared against Single-Core (sequential) and Multi-Core (parallel) CPU implementations.

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

## 1. Introduction

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

*Email address:* alacasta@unizar.es (A. Lacasta)
[1]LIFTEC, CSIC-Universidad de Zaragoza, Spain

Manuscript accepted in *Computers and Geosciences* *July 7, 2014*

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 capacity of the computational tool has been reached 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 where the accuracy of the results is tied to the resolution of the Digital Terrain Model considered. Hence, a high number of cells is usually needed in the simulation. For this reason, the hardware GPU emerges as a promising strategy for handling this environmental and up to date problem.

In terms of scientific computation, the last four decades have followed Moore's law (Moore, 2003) where the number of transistors on a chip increased exponentially. This integration allowed to obtain faster and faster applications just recompiling the code for this 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 microarchitecture 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, Multi-Core paradigm has a large power consumption rate when performing small tasks and, for these purposes the Many-Core systems appears 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 a very particular task of performing shading operations when rendering graphics. The purpose of the present work is to apply this hardware to the simulation of hazardous and 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, pararell 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. 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 in 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 $\psi$ the direction cosine of the bed normal with respect to the vertical. The physical basis of this gravity projection is explained in Juez et al. (2013) and it is of utmost importance for not ruining the numerical predictions when the simulation involves the presence of steep slopes.

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 $\rho$ the density of the granular mass. Since the geophysical flows considered in this work are dense, 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, $\theta_b$.

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 $\Omega_i$ and the Gauss theorem is applied, being the vector $\mathbf{n}$ outward to the cell $\Omega_i$, 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,

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

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),

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

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

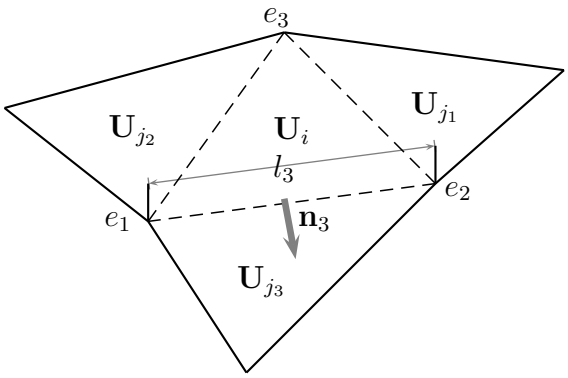


Figure 1: Cell parameters

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

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

with

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

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 sign minus in (16) implies that only the incoming waves are considered for updating the values of each cell. It is worth remarking that the feature of steep slopes has to be retained in the numerical scheme (Juez et al., 2013). Additionally, two numerical fixes regarding the friction term 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, $\mathbf{B}_k$.

In order to avoid the presence of instabilities the time step $\Delta t$ has to be taken small enough so that there is 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 $\chi_i$ 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. This latter is explained in the following section.

## 3. Implementation of the numerical scheme

The implementation of the model follows a time-stepping process where, until the simulated time is reached, 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 following the edges and the updating process as well as the boundary calculation are performed looping by cells. Both processes go from 1 to $n_{edges}$ and $n_{cells}$ respectively.

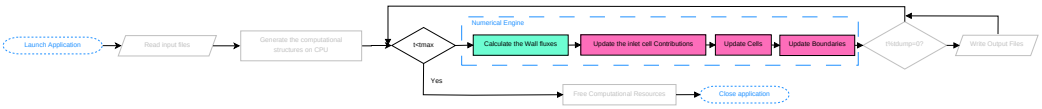


Figure 2: Flowchart of the simulation process. Green function is performed looping by cell edges, while the pink is a loop following the cells.

In order to make the process faster, some of the parts can be paralellized or even adapted to perform the operation in the GPU. Nowadays most of the processors are designed with multi-threading capabilities. Despite these capabilities, it is not direct to perform operations using every single core. Figure 3 displays a typical iterative process using a 1 core of a Multi-Core processor.

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

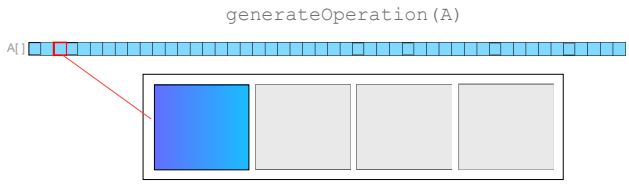


Figure 3: Sequential iteration over vector `A[]`

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 feasible when the loop contains straight-line code (a single basic block with no jumps) and when there is no data dependency between iterations (element $i-1$ is not required for the element $i$). If dataflow 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 paralelization. This standard implements parallel primitives in many programming

languages such as C or Fortran making really easy to implement a parallel solution in a reasonable time. The main disadvantage of this kind of paralellization is that the Multi-Core processors are not growing as much as the most recent Many-Core architectures, where 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]= ...
 }
}
```

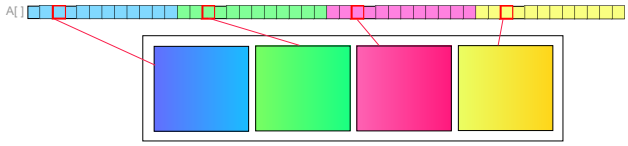


Figure 4: Parallel iteration over vector `A[]` using 4 cores

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.

CUDA (Compute Unified Device Architecture) was released in order to perform this kind of parallelization on the NVIDIA's GPUs. 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.

```
global void generateOperation(int* A){
i=threadId+
   BlockId*ThreadsPerBlock
     A[i]= ...
}
```

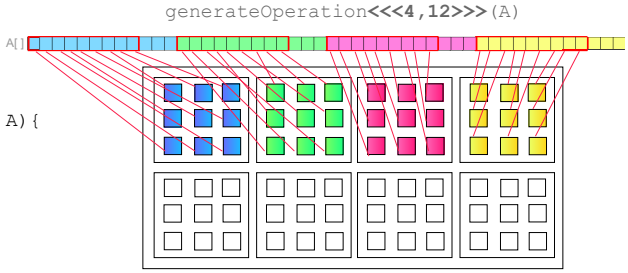


Figure 5: Massively parallel iteration over vector `A[]` using a Many-Core architecture.

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 paralelism 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 14-16 SMs and they are 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 on GPU has been developed using the approach of Lacasta et al. (2014) for unstructured meshes and adapting the required elements into these 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)
```

```
16   updateT<<<1,1,0,executionStream>>>(cuda_t,dt);
17   // Retrieves the value of t to CPU
18   cudaMemcpy(t,cuda_t,sizeof(double),cudaMemcpyDeviceToHost);
19   // Update the cell values
20   assignFluxes<<<cellBlocks,Threads,0,executionStream>>>(...);
21   // Verify whether it is neccessary to dump data and
22   // if so, process it.
23   // (*) Detailed in Listing 4
24   if(t<t_dump){
25     // Transfer data from GPU to CPU and write it
26     // into an output file
27   }
28 }
```

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 avoids the presence of preferential directions. Because of this necessity, it is not straightforward to obtain an efficient solution for the GPU processing. Moreover, special attention on those parts that are not intrinsic parallel is required (such as the selction of the global time step).
To obtain an efficient solution three special considerations are highlighted.

Using Structure of Arrays (SoA) in spite of Arrays of Structures (AoS) to store data. Because of the manner of processing the elements, SoA provides improvements when accessing to 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 by cells it is required to reorder the mesh, (Lacasta et al., 2014). 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 perfomance 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.

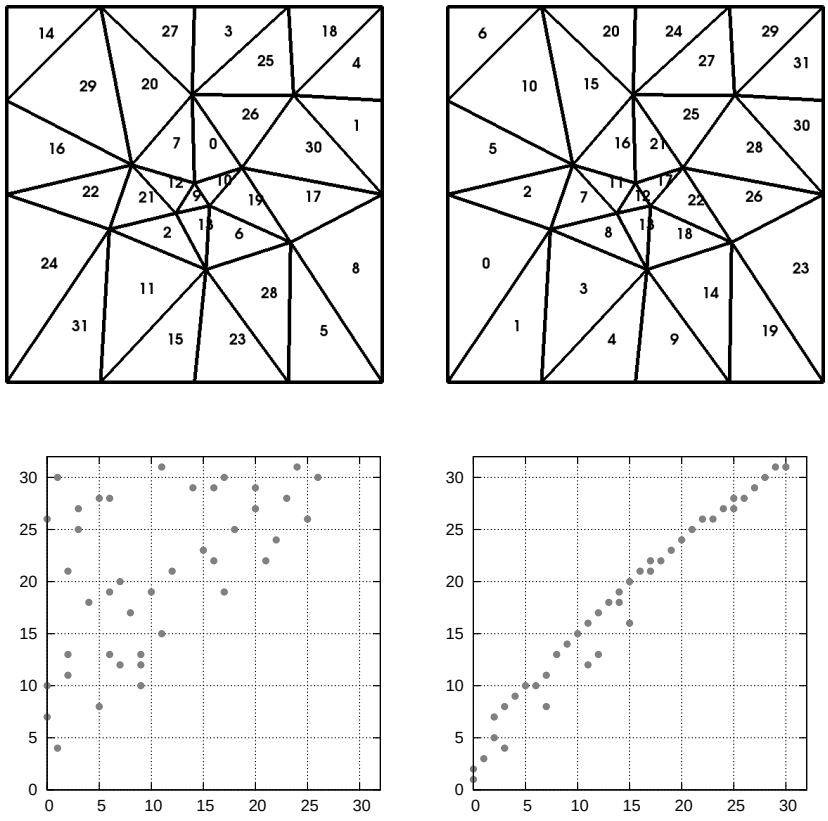


Figure 6: Upper: Unstructured mesh (left) and unstructured mesh postprocessed with RCM algorithm (right). Lower: Connectivity matrix for the original unstructured mesh (left) and for the reordered unstructured mesh (right). (Note that there exists a point if $m(i, j) = 1$).

The ordering when processing by edges. In the previous point, RCM achieves coalesced accesses when processing by cells, but it is not enough to process by edges. Actually, it is likely that when calculating by edges (when accessing to the two neighbouring 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 accesses when accessing by 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 despite the pair $(19, 23)$ that would be processed in the edge 1 without the edge ordering.

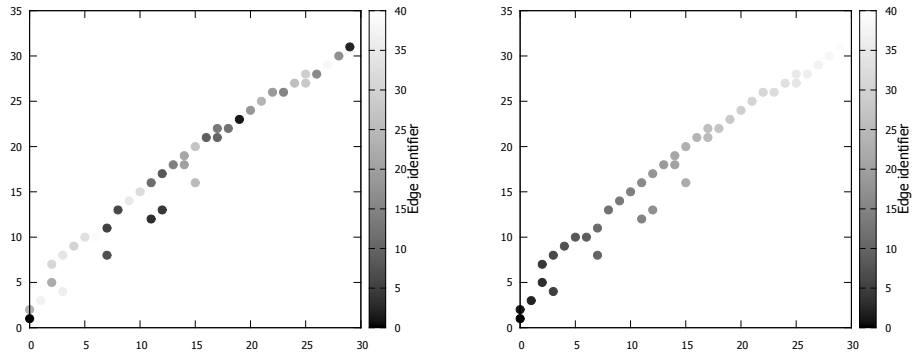


Figure 7: Unstructured mesh RCM processed edges ordering. Original ordering (left), `qsort` postprocesed ordering (right)

Applying all these previous improvements it is possible to achieve great speed-ups comparing with a sequential or a multi-core implementation of the numerical scheme. This is discussed in Section 5.

## 4. Results

This section is devoted to describe the test cases considered as benchmarking 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 by the authors for validating the numerical scheme (Juez et al., 2013). The computational cost was huge in comparison with the simulation 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 $mm$.

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.

The last test case considered constitutes a 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 $\theta_b$=32. 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 views 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 is stopped.

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

The development of this experiment was motivated by the presence of obstacles in real events. The presence of these obstacles causes the birth and propagation of quick moving shocks. The experimental work was carried out by Juez et al. (2013) and it is still under review. The setup of the experiment consists of a rough inclined plane with a changing slope and with an obstacle in the middle of the flow way. The initial condition is given by a semispherical cap full of sand. When this cap is pulled out, the granular material is free for evolving downwards its original position until surrounding the obstacle. The internal friction angle of the material was characterized as $\theta_b$ = 26 and the number of cells included in the simulation was 460000.

The temporal evolution of the granular flow is displayed in Figure 8. The front of the avalanches moves quickly until reaching the obstacle. Then, 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 semispherical cap full of sand which was suddenly removed over a parabolic chute. The material was characterized with a internal friction angle $\theta_b$=30. The number of cells considered in the calculus has been 670000.

Numerical results are plotted in Figure **??**. The material flows quickly downstream up to reach the horizontal area. In that position, the velocity of the front is reduced by the smaller 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 considered for the validation of the GPU implementation 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 part of the bare terrain 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. In Figure **??** is plotted the fixed bed rock and onto it the moving granular material. Due to the large dimensions of the catchment, the number of cells involved in the calculus is over 869000, making this type of phenomena a suitable candidate for GPU strategy.

## 5. Computational load

The necessity of refined mesh for tests 1, 2 and 3 is justified by the important effect of the bed slope in the phenomena and then, an accurate representation of the topography is required. This means that the time-step

length restricted by the CFL condition, is very low and then many time-steps are required to complete the simulation. Moreover, each time-step has a very high computational cost because of the number of cells. On the other hand, the last test case does not require such 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, being 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 (not optimized) and against an optimized version running on a *NVIDIA Tesla c2075* GPU. All the implementations have been tested in the previous four cases and the computational cost as well as the speed-ups of the parallel versions are highlighted in Table 1 and in Figure 8.

Taking into account both factors, small time-step size and high number of cells, the cost of the simulations for the sequential version of the numerical engine for tests cases 1 2 and 3 is three orders of magnitude larger than the real 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. The GPU improves the simulation cost between 34.88 and 49.40 times compared with the sequential version. Moreover, if the mesh is reordered, the computational cost is smaller and a speed-up between 49.96 and 59.85 is obtained.

The main reason for discrepancies 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 that satisfy $h > 0$ in test 4. Therefore, the more cells are involved in the calculus, the larger speed-ups can be obtained with the GPU.

## 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 computational times have been compared with those obtained when considering Single-Core and Multi-Core processors. The GPU implementa-

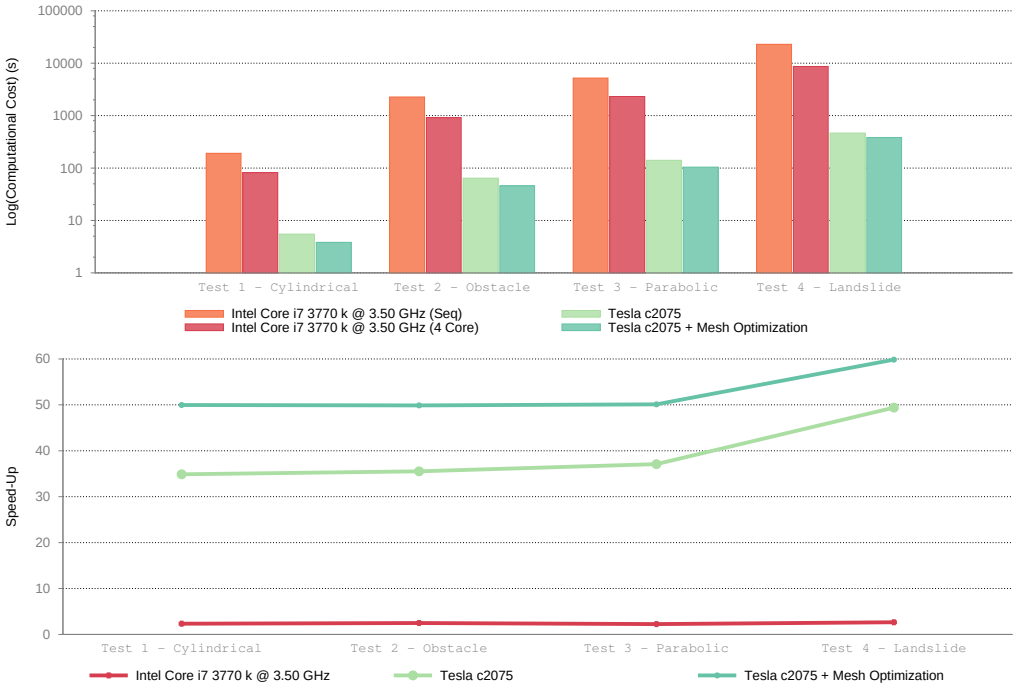


Figure 8: Computational time (upper) using logarithmic scale and speed-up (lower) for the compared implementations

| | | 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.

tion and specially, the application of cell and wall ordering algorithms, have driven to obtain noticeable improvements in the speed-up of the test cases.

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.

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 intensively 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., 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 -, (Under Review).

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 landslide flow history and of the generated long period seismic waves. Geophysical Research Letters 39, L16402.

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

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

NVIDIA, 2011. NVIDIA CUDA C Programming Guide.

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

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

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

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

Sharma, S., Gupta, K., 2013. Parallel performance of numerical algorithms on multi-core system using openmp, in: Meghanathan, N., Nagamalai, D., Chaki, N. (Eds.), Advances in Computing and Information Technology.

Springer Berlin Heidelberg. volume 177 of *Advances in Intelligent Systems and Computing*, pp. 279–288.