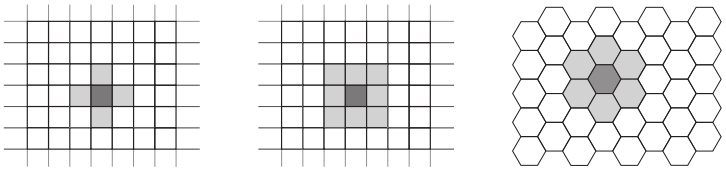
Multi-GPU parallelization and performance assessment of the **SciddicaT** 2D Cellular Automata landslide simulation model

(Single student project)

Course: Massively Parallel Programming on GPUs
Teacher: Donato D'Ambrosio

Master Degree in Computer Science
Department of Mathematics and Computer Science
University of Calabria, Italy

December 5, 2023



(a) (b) (c)

FIGURE 1: Example of two-dimensional domains (or cellular spaces) and neighborhoods (or stencils): (a) two-dimensional domain with von Neumann's neighborhood adopted by sciddicaT; two dimensional domain with Moor's neighborhood; two-dimensional hexagonal domain with Moor's neighborhood.

1 The Project in Brief

The project consists in the CUDA/MPI multi-GPU parallelization and performance assessment of a simple scientific application, namely the SciddicaT non-inertial fluid flows simulation model [1].

A serial/OpenMP reference implementation is provided to be used as the reference starting point for implementation, correctness and performance assessment. Two datasets, namely the 1982 Tessina (Italy) landslide (n. 1 flow source) and a larger *stress test case* (n. 100 flow sources), are also provided.

2 SciddicaT Definition and Implementation Notes

SciddicaT is a simple fluid-flow simulator based on the Cellular Automata computational paradigm [4], and the *Minimization Algorithm of the Differences* [2]. This latter is considered for computing flows among neighboring cells. Despite its simplicity, the model was able to simulate non-inertial landslides on real topographic surfaces, like the Tessina landslide, occurred in Italy in 1982 [1]. SciddicaT is defined as:

$$\text{SciddicaT} =< R, X, S, P, \sigma >$$

where R and X are the two-dimensional computational domain, subdivided in square cells of uniform size, and the von Neumann neighborhood, as shown in Figure 1a. Other examples are shown in Figures 1b and 1c.
S is the set of cell states. It is subdivided in the following substates:

- S_z is the set of values representing the topographic altitude (i.e. elevation a.s.l.);
- S_h is the set of values representing the fluid thickness;

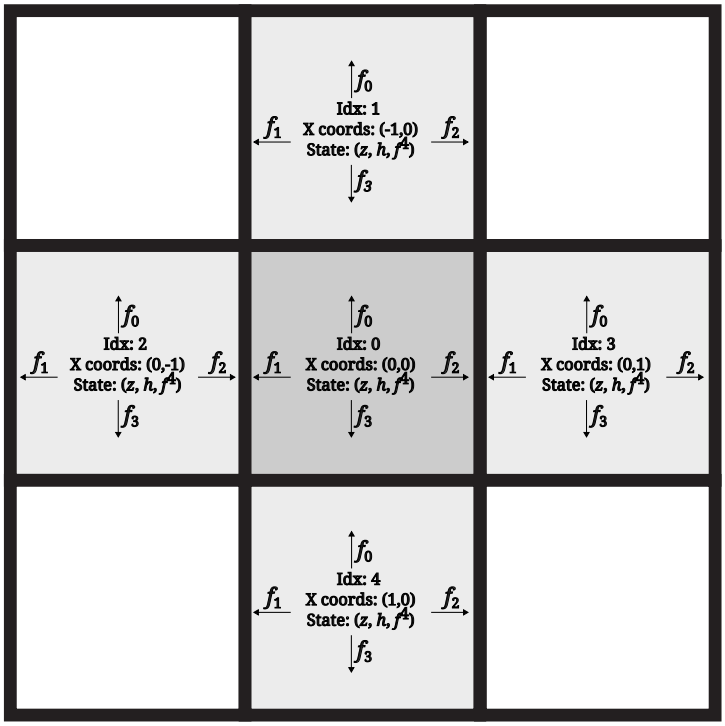


FIGURE 2: Graphical representation of the SciddicaT neighborhood. Cell's state (z, h, f^4), neighborhood's labels (Idx), relative coordinates (X coords), and outflows indices (subscripts of the four f state variables) are shown.

- S_f^4 are the sets of values representing the outflows from the central cell to the four adjacent neighbors.

$P = \{p_\epsilon, p_r\}$ is the set of parameters ruling the model dynamics. In particular, p_ϵ specifies the minimum thickness below which the fluid cannot outflow the cell due to the effect of adherence, while p_r is the relaxation rate parameter, which essentially is an outflow damping factor.
$\sigma : S^5 \rightarrow S$ is the deterministic cell transition function. It is composed by the elementary processes listed below in the same order they are applied:

- $\sigma_1 : (S_z \times S_h)^5 \times p_\epsilon \times p_r \rightarrow S_f^4$ computes outflows from the central cell to the four neighboring ones by applying the *minimization algorithm of the differences* [2]. As a simplification of the adherence effect, a fluid thickness h smaller than or equal to the p_ϵ threshold is considered unmovable and can not leave the cell. The outgoing flows are therefore computed based on the fluid part that exceeds the threshold (i.e., $h - p_\epsilon$). The resulting outflow towrds the i^{th} neighboring cells ($i =$

$1, 2, \ldots, 4$) has index $i-1$ and is given by:

$$f(0,i) = f_{i-1} = \begin{cases} f_{min}(0,i) \cdot p_r, & \text{if the } i^{th} \text{ cell is not eliminated} \\ 0, & \text{otherwise} \end{cases}$$

being $f_{min}(0,i)$ the outgoing flows towards the i^{th} adjacent cell ($i = 1, 2, \ldots, 4$), as computed by the minimization algorithm, and $p_r \in]0,1]$ a relaxation factor considered to damp outflows in order to obtain a smoother convergence to the global equilibrium of the system.

- $\sigma_2 : S_h \times (S_f^4)^5 \rightarrow S_h$ determines the value of debris thickness inside the cell by considering mass exchange in the cell neighborhood: $h^{t+1}(0) = h^t(0) + \sum_{i=1}^{4}(f(i,0) - f(0,i))$. Here, $h^t(0)$ and $h^{t+1}(0)$ are the mass thickness inside the cell at the t and $t+1$ computational steps, respectively, while $f(i,0)$ represents the inflow from the i^{th} neighboring cell ($i = 1, 2, \ldots, 4$).

2.1 Implementation Notes

A 0-based label is used in the reference implementation to identify the cells belonging to the neighborhood, as well as a 0-based index is used to label the outflows from the central cell toward the four adjacent ones. Cell labels therefore are in the {0, 1, 2, 3, 4} index space, while the flow indices are in the {0, 1, 2, 3} index spaces. See Figure 2 for a detailed representation of the cell's state, neighborhood's labels (Idx), relative coordinates (X coords), and outflows indices (subscripts of the four f state variables). Note that the index of the outflow towards the i^{th} neighboring cell is $i-1$.

A linear array of type `double*` is used to store the S_z and S_h state variables for each domain's cell, while S_f is a buffered linear array that stores the four per-cell outflows from the central cell towards the remaining neighbors. Row-major order is considered in the buffered linearization. Accordingly to this choice, the domain R is implicitly modeled in the substates' definition.

2.2 Data Input to the Model

Besides the values to be assigned to the parameters, which are defined into the source code (with $p_\epsilon = 0.001$ and $p_r = 0.5$), the input to the model is represented by three text files in Ascii grid format. The first one (the header) defines the domain dimensions (number of columns and rows), geographical coordinates (of the bottom left cell of the grid), cell size, and a no-data value (used to label the cells with a lack of information)[1]. The remaining files (grid maps) represent the information regarding the topographic altitude and the flow thickness for each domain cell.

[1] Actually, only the domain dimensions and the no-data value are taken into account in SciddicaT.

Make command	Result
make	Build the OpenMP version
make run	Run the Tessina OpenMP simulation
STRESS_TEST_R=1 make run	Run the stress test OpenMP simulation
make sciddicaTserial	Build the serial version
make run_serial	Run the Tessina serial simulation
STRESS_TEST_R=1 make run_serial	Run the stress test serial simulation

TABLE 1: Make commands to build SciddicaT and run the simulation.

The initial configuration of the system is defined as follows:

- A topographic map is read from file to initialize the z state variable for each domain's cell.
- A mass thickness map is read from file to initialize the h state variable for each domain's cell;
- The outflows $f_{i-1} = f(0, i)$ are set to zero everywhere ($i = 1, 2, \ldots, 4$).

Boundary conditions are very simplified and consist in ignoring (i.e., not evaluating the state transition for) the cells belonging to the grid boundaries.

At each iteration step, two basic sub-steps (aka kernels, local processes, or elementary processes) are computed, corresponding to the transition function processes described above.

- The `sciddicaTFlowsComputation` kernel corresponds to the σ_1 elementary process.
- The `sciddicaTWidthUpdate` kernel corresponds to the σ_2 elementary process.

2.3 Compile and Exec SciddicaT

A makefile is provided with SciddicaT. Refer to Table 1 for the `make` commands to be used to build and run SciddicaT.

Depending on the run command used, one of the following files is created, which represent the final configuration of the system: `tessina_output_OpenMP.qgis`, `strss_test_R_output_OpenMP.qgis`, `tessina_output_serial.qgis`, `strss_test_R_output_OpenMP.qgis`. These files are in a format that can be read by the Qgis application as raster layers. An example of Qgis visualization of the Tessina and stress test simulations are are provided in Figures 3 and 4, repsectively. Note that the md5sum checksum of eachg simulation is assessed based on the output files without the header. Checksums are reported in Table 2.

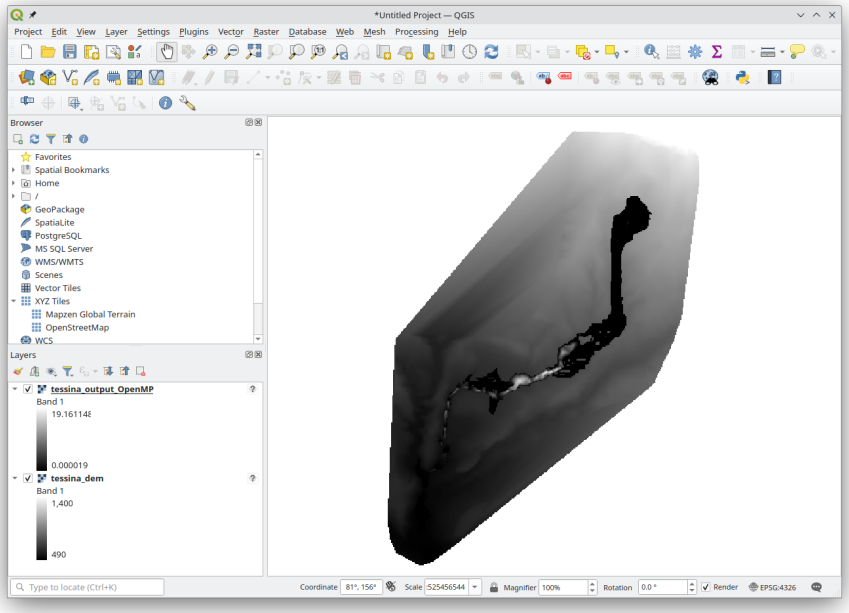


FIGURE 3: Example of Qgis visualization of the Tessina simulation.

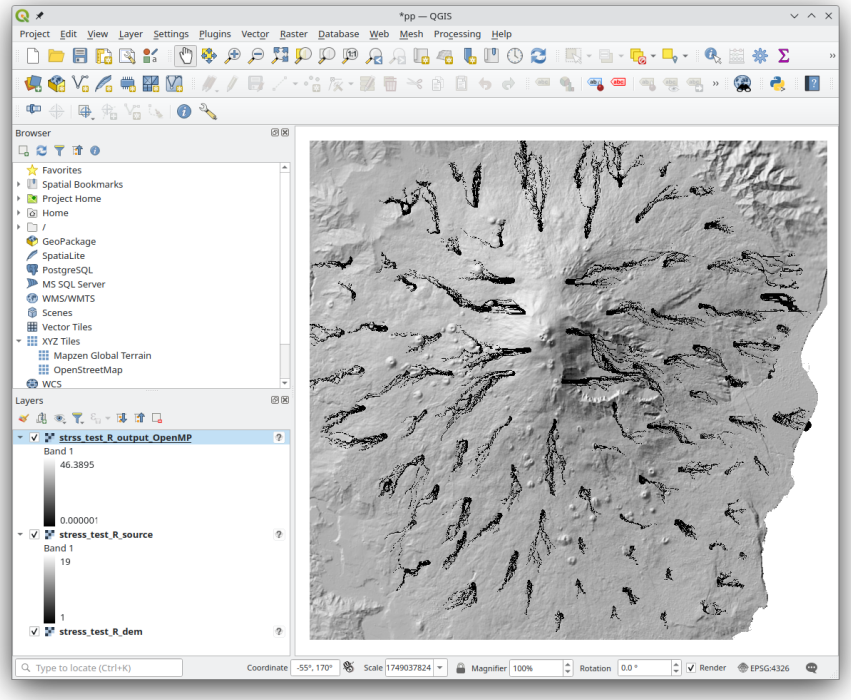


FIGURE 4: Example of Qgis visualization of the stress test simulation.

Simulation	md5sum
Tessina simulation	8ed78fa13180c12b4d8aeec7ce6a362a
Stress test simulation	ab43bc91a375c638cd6c518a7accafdb

TABLE 2: Checksums of the SciddicaT and stress test simulations.

Block configuration
8x8
16x32
32x16
32x32

TABLE 3: Block configurations to be used for performance assessment.

3 The project

The project is defined in the following sections.

The CUDA/MPI parallel implementations to be developed are listed below:

1. `Straightforward_Unified`, namely a direct parallelization using the CUDA Unified Memory.
2. `Straightforward_Standard`, namely a direct parallelization using the CUDA standard host/device memory.
3. `Tiled_no_halos`, namely a CUDA shared memory based tiled parallelization without halo cells.
4. `Tiled_with_halos`, namely a CUDA tiled parallelization in which block's boundary threads perform extra work to copy halo cells from adjacent tiles in the shared memory.
5. `Tiled_with_halos_larger_block`, namely a CUDA tiled parallelization in which the block size is larger than the tile size so that each thread just copies a single cell to the shared memory and only a number of threads equal to the tile size is used to update the tile.
6. `Tiled_with_halos_larger_block_MPI`, namely an MPI multi-GPU implementation based on the `Tiled_with_halos_larger_block` version. This implementation will be executed on the JPDM2 workstation by using two processes, one process per GPU.

Note that, in most applications, not all the kernels need to be updated to a tiled implementation. It is a your responsibility to chose which kernels would benefit of a tiled implementation.

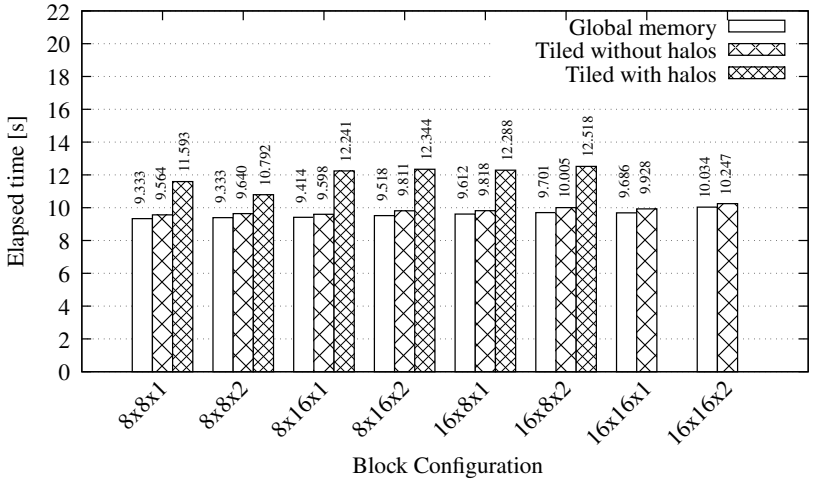


FIGURE 5: Examples of bar plot chart.

Implementation	Block	Time	Speedup	Occupancy[%]
`Straightforward_Unified`	8x8	1.00	1.00	75.02
`Straightforward_Standard`	32x16	1.02	0.83	64.07
`Tiled_no_halos`	32x32	0.97	1.03	87.12
`Tiled_with_halos`	16x32	0.99	1.01	90.23
`Tiled_with_halos_larger_block`	8x8	0.91	1.10	78.96
`Tiled_with_halos_larger_block_MPI`	8x8	0.91	1.10	78.96

TABLE 4: Example of table for the performance assessment.

3.1 Performance Assessment

For each CUDA implementation, run four experiments and plot the best elapsed time registered. Use a bar plot chart for this purpose, by paying attention to label the axes and to write a meaningful caption. An example is shown in Figure 5. Reporting the elapsed times on top of each bars is optional but recommended.

In addition, report the elapsed times registered by the best (implementation, block configuration) couples in a table. Add a column to report the speed-up with respect to the `Straightforward_Unified` implementation (defined as elapsed time of `Straightforward_Unified`/elapsed time other implementation). Furthermore, perform a profiling analysis of the best performing (implementation, block configuration) executions to assess the warp occupancy of each of them. Report the results in a new column of the same table used for the elapsed times. The table has to resemble to Table 4.

Note that the experiments must be executed on the GTX 980 GPU for comparison.

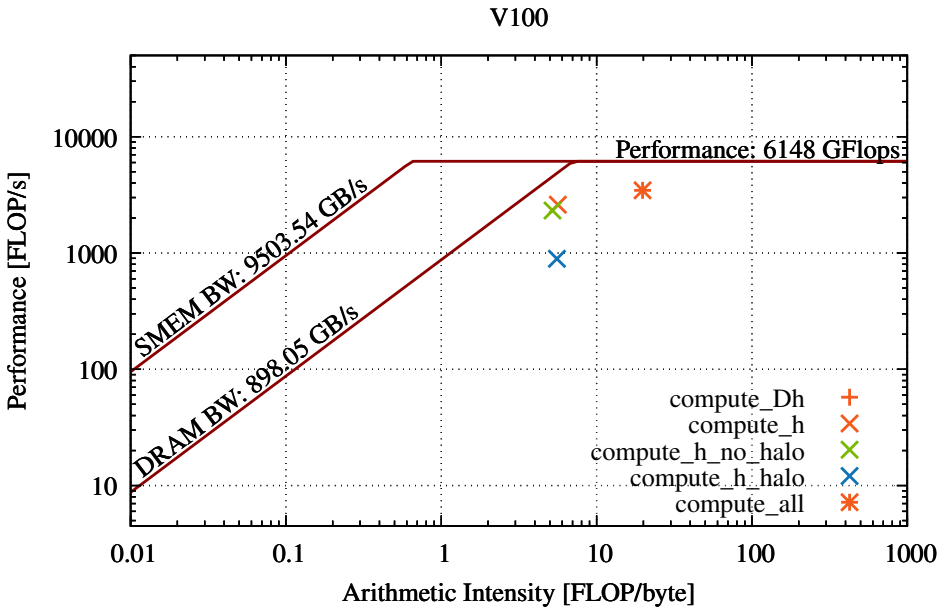


FIGURE 6: Examples of Roofline chart.

3.2 Optional Research Activity

As you will probably notice, the performance of the tiled implementations is unsatisfying. This is mainly due to the small stencil (i.e., to the von Neumann neighbourhood), which is made by only five cells. The issue is known in the literature, and no alternative algorithms have been proposed to speed-up small stencil-based applications like SciddicaT. You are invited to think about the problem and propose a possible solution. This activity is optional. If not done, it will not impact to the final mark. If done, it will positively impact on the final mark.

3.3 Applying the Roofline Model

Define the Roofline plot for the GTX 980 GPU, for each best (implementation, block configuration) couple. You can either use the ERT (Empirical Roofline Toolkit) [5] or the *gpumembench* [3] micro-benchmark[2]. Then, for each of couple, evaluate the Operational Intensity of the transition function kernel, and assess its performance by means of `nvprof`. Then, put a point for the kernel in the Roofline plot and briefly comment the kernel's nature, if compute or memory bound. An example of Roofline chart is shown in Figure 6.

[2]Note that you need to change the NVCODE macro in the CUDA makefiles in: `NVCODE = -gencode=arch=compute_52,code="compute_52" -ftz=true` in order to generate the binaries for the compute architecture 5.2 only.

3.4 Write a Report

Eventually, summarize the outcomes in a very brief report by using the LeTeX LNCS class. You are free to chose the structure of the report.

For each parallel version implemented, report the best achieved results both in numerical, by means of tables, and visual terms, by means of meaningful plots. In the tables reporting the numerical data, also insert details about the CUDA grid used (how many block and how many threads per block).

Regarding the Roofline assessment, report your evaluations regarding the Arithmetic Intensity of each kernel, and plot their performance in a single plot and comment the kernels nature, if memory or compute bound.

4 Team and Mark

The project has to be carried out individually. The maximum score is 30/30. In case of delivery by the first and second exam call of the winter session, additional 2 and 1 points will be awarded respectively. The *summa cum laude* price may be awarded in any case.

References

[1] MV Avolio, Salvatore Di Gregorio, Franco Mantovani, Alessandro Pasuto, Rocco Rongo, Sandro Silvano, and William Spataro. Simulation of the 1992 Tessina landslide by a cellular automata model and future hazard scenarios. *International Journal of Applied Earth Observation and Geoinformation*, 2(1):41–50, 2000.

[2] S. Di Gregorio and R. Serra. An empirical method for modelling and simulating some complex macroscopic phenomena by cellular automata. *Future Generation Computer Systems*, 16:259–271, 1999.

[3] Elias Konstantinidis and Yiannis Cotronis. A quantitative performance evaluation of fast on-chip memories of gpus. In *2016 24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP)*, pages 448–455, 2016.

[4] John von Neumann. *Theory of Self-Reproducing Automata*. University of Illinois Press, Champaign, IL, USA, 1966.

[5] Charlene Yang, Thorsten Kurth, and Samuel Williams. Hierarchical roofline analysis for gpus: Accelerating performance optimization for the nersc-9 perlmutter system. *Concurrency and Computation: Practice and Experience*, 32(20):e5547, 2020.