MultiGPU development and performance assessment of a simple numerical simulator of non-inertial fluid flows in CUDA and MPI

Teacher: Donato D'Ambrosio

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

December 16, 2020

1 The Assignment in Brief

The assignment 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 a reference starting point for implementation, correctness and performance assessment. A dataset, representing the initial configuration of a real case of study, namely the 1982 Tessina (Italy) landslide, is also provided. The elapsed time of the serial version of SciddicaT on the JPDM2 reference workstation is 67.5 seconds. Actually, the JPDM2's Intel Xeon E5440 CPU is quite old. Nevertheless, we will refer to the corresponding sequential time for comparison.

Important Note: The assignment is mandatory. It, together with the previous four assignments, and a brief discussion about the work done, replaces the original *written exam plus project* formula. In other words, the student is dispensed by the written exam.

2 SciddicaT Definition and Implementation Notes

SciddicaT is a simple fluid-flow simulator based on the Cellular Automata computational paradigm [3], 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 is the two-dimensional computational domain, subdivided in square cells of uniform size, while X is the von Neumann neighborhood (a geometrical pattern identifying the central cell and a set of four cells located to the north, west, east, and south directions, adjacent to the central one). A 0-based label is used in the serial reference implementation to identify the cells belonging to the neighborhood, as well as a 0-based index is used to label the flows 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. The diagram below represents the indices and labels of a cell and its neighbourhood:

```
Cell format: |flow_index:cell_label:(row_coord,col_coord)|
   cell_label in {0,1,2,3,4}: cell labels
   flow_index in   {0,1,2,3}: outgoing flow indices
       (row_index,col_index): relative cells coordinates
```

```
... continue from the previous page. This is the diagram:

               |0:1:(-1, 0)|
 |1:2:( 0,-1)|_:0:( 0, 0)|2:3:( 0, 1)|
               |3:4:( 1, 0)|
```

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

Note that a linear buffer of type `double*` is used to model each substate. In particular, S_z and S_h have size equal to the product of the numbers of rows and columns of the two-dimensional domain, while S_o has size equal to 4 times the same product, since it models four outflows from the central cell towards the remaining neighbors. Row-major order is considered in the buffer linearization. Accordingly to this choice, the domain R is implicitly considered in the substates.
$P = \{p_\epsilon, p_r\}$ 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.
$\sigma : S^5 \to S$ is the deterministic cell transition function. It is composed by three elementary processes, listed below in the same order they are applied:

- $\sigma_0 : S_o^4 \to S_o^4$ sets the outflows from the central cell to the adjacent neighbors to zero.
- $\sigma_1 : (S_z \times S_h)^5 \times p_\epsilon \times p_r \to S_o^4$ 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 outflows are given by

  $$q_o(0, m) = f(0, m) \cdot p_r \ (m = 1, \ldots, 3)$$

  being $f(0, m)$ the outgoing flows towards the 4 adjacent cells, 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. The S_o^4 substates are updated accordingly with the values of the computed outflows.

- $\sigma_2 : S_h \times (S_o^4)^5 \to S_h$ determines the value of debris thickness inside the cell by considering mass exchange in the cell neighborhood: $h^{t+1}(0) = h^t(0) + \sum_{m=0}^{3}(q_o(0,m) - q_o(m,0))$. 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 $q_o(m,0)$ represents the inflow from the $n = (m+1)^{th}$ neighboring cell. The S_h substate is updated accordingly to account for the mass balance within the cell.

2.1 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). Actually, only the domain dimensions and the no-data value are taken into account in SciddicaT. The remaining files (grid maps) represent the information regarding the topographic altitude and the flow thickness for each domain cell. An example of header and topographic altitude grid is shown below and graphically represented in Figure 1:

```
# This is a comment. Below you can find the header...

ncols         10
nrows         10
xllcorner     2487289.5023187
yllcorner     4519797.0771414
cellsize      10.0
NODATA_value  -9999


# A 10x10 grid of topographic altutudes...

9.00 8.00 7.00 6.00 5.00 4.00 3.00 2.00 1.00 0.00
8.00 7.00 6.00 5.00 4.00 3.00 2.00 1.00 0.00 1.00
7.00 6.00 5.00 4.00 3.00 2.00 1.00 0.00 1.00 2.00
6.00 5.00 4.00 3.00 2.00 1.00 0.00 1.00 2.00 3.00
5.00 4.00 3.00 2.00 1.00 0.00 1.00 2.00 3.00 4.00
4.00 3.00 2.00 1.00 0.00 1.00 2.00 3.00 4.00 5.00
3.00 2.00 1.00 0.00 1.00 2.00 3.00 4.00 5.00 6.00
2.00 1.00 0.00 1.00 2.00 3.00 4.00 5.00 6.00 7.00
1.00 0.00 1.00 2.00 3.00 4.00 5.00 6.00 7.00 8.00
0.00 1.00 2.00 3.00 4.00 5.00 6.00 7.00 8.00 9.00
```

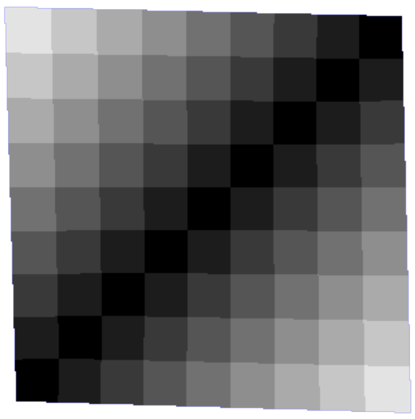


FIGURE 1: Graphical representation of the 10x10 Ascii Grid. It is based on a straightforward color mapping technique, which simply assigns the color black to the cell with the lowest value, white to the one with the highest value and grey tones to cells having intermediate values.

The initial configuration of the system is defined as follows:

- A topographic map is read from a file to initialize the Sz substate. An example is shown in Figure 1;
- A mass thickness map is read from a file to initialize the S_h substate;
- The outflow substate layers are initilized to zero everywhere.

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, three basic sub-steps (aka kernels, local processes, or elementary processes) are computed, corresponding to the transition function processes described above.

- The `sciddicaTResetFlows` kernel corresponds to the σ_0 elementary process.
- The `sciddicaTFlowsComputation` kernel corresponds to the σ_1 elementary process.
- The `sciddicaTWidthUpdate` kernel corresponds to the σ_2 elementary process.

2.2 Compile and Exec SciddicaT

A makefile comes with SciddicaT that can be used for both building and executing the program on the datasets described in the previous section. To build the executables, both the serial and the OpenMP ones, namely sciddicaTserial and sciddicaTomp, simply run the make utility in the Linux terminal:

```
make
```

and then use the following command to run the serial simulation (that must be used to asses the computational performance on the CPU) for a total of 4000 steps:

```
./sciddicaTserial ../data/tessina_header.txt \
../data/tessina_dem.txt ../data/tessina_source.txt \
./tessina_output_serial 4000 \
&&  md5sum ./tessina_output_serial \
&& cat ../data/tessina_header.txt ./tessina_output_serial  \
> ./tessina_output_serial.qgis && rm ./tessina_output_serial
```

or the following command to run the multi-core simulation:

```
OMP_NUM_THREADS=2 \
./sciddicaTomp ../data/tessina_header.txt \
../data/tessina_dem.txt ../data/tessina_source.txt \
./tessina_output_OpenMP 4000 \
&&  md5sum ./tessina_output_OpenMP && cat \
../data/tessina_header.txt ./tessina_output_OpenMP \
> ./tessina_output_OpenMP.qgis && rm ./tessina_output_OpenM
```

Alternatively, you can use the `run` and `run_omp` make targets. If you want to use more then 2 threads, simply change the `OMP_NUM_THREADS` value in the command line or in the Makefile.

As a result, depending on the run command used, one of the following two files is created, which represent the final configuration of the system:

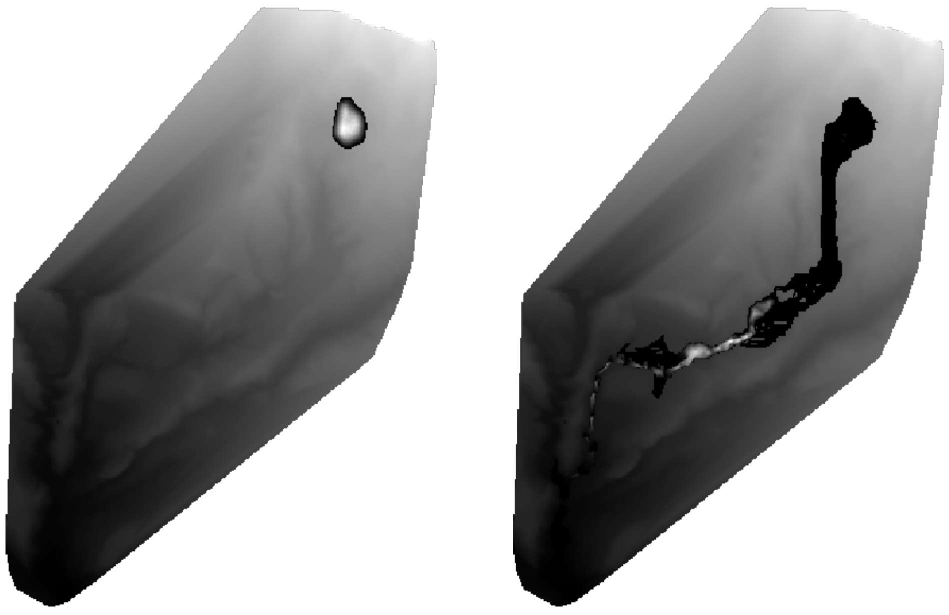
```
tessina_output_serial.qgis | tessina_output_OpenMP.qgis
```

These files are in a format that can be read by the Qgis application as a raster layer. Figures 2a and 2b show the initial and final configuration of the system, respectively. Note that the md5sum checksum is assessed based on the output files without the header. The check gives the following result:

```
8ed78fa13180c12b4d8aeec7ce6a362a
```

3 The Assignment

The assignment is defined in the following sections.



(A) Initial configuration (B) Final configuration

FIGURE 2: Initial and final configuration (after 4000 computational steps) of the system.

3.1 Straightforward parallelization

Implement a straightforward CUDA parallel version of the SciddicaT model using the Global Memory only. Start with monolithic kernels and then update each kernel using grid-stride loops. Assess the correctness of the parallel version with respect the md5 checksum of the serial version, namely: `8ed78fa13180c12b4d8aeec7ce6a362a`. For this purpose, remember that the checksum is computed with the md5sum application on the grid file (without header) storing the final configuration of the S_h substate. Please, make sure to use the same output functions of the serial version provided to avoid possible mismatches.

3.2 Tiled parallelization with Halo Cells

Based on the straightforward parallel implementation, develop a new CUDA parallel implementation using a tiled algorithm with halo cells. Shared Memory is required in this implementation. Assess the correctness of the developed parallel implementation with respect the md5 checksum of the serial version.

3.3 Tiled parallelization without Halo Cells

Based on the tiled parallel implementation with halo cells, develop a new CUDA parallel implementation using a tiled algorithm without halo cells (cf. Chapter 7 of the textbook). Shared Memory is also required in this implementation. Assess the correctness of the developed parallel implementation with respect the md5 checksum of the serial version.

3.4 MPI Parallelization

Based on the most performing SciddicaT version developed, implement a multi-GPU version using MPI. Then run it on JPDM2 using two processes, one process per GPU ans assess the correctness of the developed multi-GPU implementation with respect the md5 checksum of the serial version.

3.5 Performance Assessment

Assess the performance of all the developed parallel versions in terms of speed-up on JPDM2. For this purpose, refer the JPDM2 serial time of 67.5 seconds. Try different configuration for the CUDA grid and use the CUDA occupancy calculator to better calibrate blocks and threads for each computing kernel. For each implementation and for each kernel, declare the tested grid configurations in a table and plot the best result you have obtained in a readable way, paying attention to label the axes and to write a meaningful caption. In addition, report the elapsed times of the serial and of the best executions of the developed parallel versions in a table.

3.6 Applying the Roofline Model

Define the Roofline plot for the GTX 980 GPU, for both the Global and the Shared Memory. This latter must be based on the throughput measured by means of the *gpumembench* micro-benchmark[1]. Then, for each of the three implemented single-GPU versions, evaluate the Arithmetic Intensity of each kernel, and assess their performance by means of `nvprof`. Then, put a point for each kernel in the Roofline plot and briefly comment the kernels' nature, if compute and/or memory bound.

3.7 Write a Report

Eventually, summarize the outcomes in a (well written) report. A possible structure could be the following:

[1]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.

- Abstract - Very short, just to let the reader know what you have done and the results you have obtained.
- Introduction - Short description of the SciddicaT model, its serial implementation and what you are going to with it.
- Parallel Implementations - Here a subsection for each of the four parallel versions you are invited to develop. Briefly describe what you have done, the potential advantage of each parallelization approach used, and state the correctness with respect to the serial implementation.
- Computational Performance - For each parallel version, here you 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).
- Roofline Assessment - Here report your evaluations regarding the Arithmetic Intensity of each kernel, and plot their performance in a single Roofline plot together (this is possible because the roofline depends on the hw only) and comment the kernels nature, if memory and/or compute bound.
- Conclusion - Here recap the work done and comment the results achieved. In addition, outline possible development of your work that may help further improve the performance of SciddicaT.

Feel free to develop other ideas or change (read improve!) the structure of the report.

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