Multi-GPU parallelization and performance assessment of the **Sciara-fv2** 2D Cellular Automata lava-flows simulation model

(Project for up to 2 students)

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

November 20, 2025

1 The Project in Brief

The project consists in the multi-GPU parallelization and performance assessment of the Sciara-fv2 lava flows simulation model [4].

A serial/OpenMP reference implementation is provided to be used for correctness and performance assessment. A dataset, representing the initial configuration of a real case of study, namely the 2006 Mt Etna (Italy) lava flow, is also provided.

2 Sciara-fv2 Definition and Implementation Notes

The formal definition of the Sciara-fv2 model can be found in [4], which is provided together with this guide. The student is invited to read it carefully before proceeding further. In the following, we just report the differences between the implementation provided and the definition in [4].

In the reference paper, the cell's state variables are called *substates*, namely Q_z, Q_h, Q_T, and Q_f^8. They represent topographic elevation above sea level, lava thickness, lava temperature and lava outflows from the central cell to its 8 adjacent cells (expressed as thicknesses), respectively. In the serial/OpenMP implementation, the first three substates are stored into double-precision linear arrays, namely `Sz`, `Sh` and `ST`, representing two-dimensional matrices. Each element represents the state variable value of a cell for the domain R, also called cellular space, which is the discrete representation of a continuous domain (e.g., a portion of the Mt Etna volcano). The `Sz_next`, `Sh_next` and `ST_next` support arrays are also defined, which are used to write the new state values for the cell, as computed by the transition function's elementary processes (i.e., the kernels).

The outflows from the central cell toward the neighbors, which are defined as Q_f^8, are implemented as a single buffered structure made by 8 linear arrays, one for each adjacent cell, and is called `Mf`.

There are other buffers, namely `Mv`, `Mb`, and `Mhs`. Their meaning is not here provided as not relevant to the goal of the project. Note that the buffers that start with a `S` have a `_next` supplementary buffer, the ones starting with an `M` do not.

The access to the cell's state variables is guarantied by the `C` macros

```
SET(M, columns, i, j, value)
GET(M, columns, i, j)
BUF_SET(M, rows, columns, n, i, j, value)
BUF_GET(M, rows, columns, n, i, j)
```

`GET` and `SET` access the cell `(i,j)` of the array `M`. The number of elements in a row (i.e., the number of columns) is also required, represented by the parameter `columns` in the definitions. In addition, `BUF_GET` and `BUF_SET`

require further information, namely the number of rows, represented by the parameter `rows`, and the layer to be accessed, which is represented by the parameter `n`. The set macros update the state variable `M` at position `(i,j)` with the value of the `value` parameter.

The access to the neighboring cells is performed by adding the relative coordinates of the neighbors, which are stored in the `Xi` and `Xj` arrays, to the coordinates of the central cell. Therefore, it is sufficient to use the `(i+Xi[k],j+Xj[k])` coordinates to access the k^{th} neighboring cell of the cell `(i,j)`. The `Xi` and `Xj` arrays therefore correspond to the X geometrical pattern of the formal definition of Sciara-fv2. Nevertheless, the order of the neighbors slightly differ. It is specified in the following.

A 0-based label is used in the serial/OpenMP 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 adjacent ones. Cell labels therefore are in the {0, 1, ..., 8} index space, while the flow indices are in the {0, 1, ..., 7} index space. The flow 0 is distributed to the cell 1, the flow 1 to the cell 2, and so on. The diagram below represents the indices and labels of a cell and its neighborhood:

```
cell_label in {0,1,2,3,4,5,6,7,8}: label assigned to each cell
                                   in the neighborhood
flow_index in {_,0,1,2,3,4,5,6,7}: outgoing flow indices in Mf
                                   from cell 0 to the others

 cells               cells         outflows
 coordinates         labels        indices

-1,-1|-1,0|-1,1      |5|1|8|       |4|0|7|
 0,-1| 0,0| 0,1      |2|0|3|       |1|_|2|
 1,-1| 1,0| 1,1      |6|4|7|       |5|3|6|
```

Regarding the external function γ and the transition function τ, the first is implemented into the program by the `emitLava` kernel, while the last one by the following kernels:

- `computeOutflows`
- `massBalance`
- `computeNewTemperatureAndSolidification`

There is a further kernel for the boundary conditions, namely `boundaryConditions`, and a global reduction over `Sh` used to define a stopping criterion.

The simulation is obtained by calling the above function, in the same order they have been cited, within the following loop (pseudo-code):

```
while ( (max_steps > 0 && step < max_steps)
     || (elapsed_time <= effusion_duration)
     || (total_current_lava > thickness_threshold)
      )
{
  elapsed_time += Pclock;
  step++;

  for_each cell in R:
    emitLava(...);
  Sh <- Sh_next
  ST <- ST_next

  for_each cell in R:
    computeOutflows(...);

  for_each cell in R:
    massBalance(...);
  Sh <- Sh_next
  ST <- ST_next

  for_each cell in R:
    computeNewTemperatureAndSolidification(...);
  Sz <- Sz_next
  Sh <- Sh_next
  ST <- ST_next

  for_each cell in R:
    boundaryConditions(...);
  Sh <- Sh_next
  ST <- ST_next

  total_current_lava = reduceAdd(...);
}
```

2.1 Data Input to the Model

The input to the model is defined by a set of files, as follows:

- `2006_000000000000.cfg` contains the Sciara-fv2 parameters;
- `2006_000000000000_EmissionRate.txt` defines the lava feeding at vents;

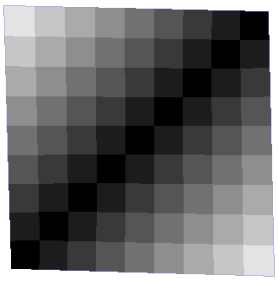

FIGURE 1: Graphical representation of the 10x10 Ascii Grid. It is based on a straightforward color mapping technique, which simply assigns the color black to the cell with the lowest value, white to the one with the highest value and grey tones to cells having intermediate values.

- `2006_000000000000_Morphology.asc` represents the topography over which the flow develops;
- `2006_000000000000_Vents.asc` is the map of vents (effusion points);
- `2006_000000000000_RealEvent.asc` file is the map of the real event.

Note that the morphology, vents, and real event files are in ascii format, with an header and a bi-dimensional grid of value. An example of topographic map is shown below and graphically represented in Figure 1:

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

The initial values of substates are defined as follows:

- The topographic map initializes the `Sz` array.
- The vent map initializes a vector called `vents` used for lava feeding during the simulation.
- The thickness map (if present) defines the initial values for the `Sh` array; if not present, `Sh` is set to zero everywhere;

- The temperature map (if present) defines the initial values for the `ST` array; if not present, `ST` is set to zero everywhere;
- The outflow buffered array `Mf` is initialized to zero everywhere.

Boundary conditions are very simplified and consist in setting to zero the substate values of cells belonging to the domain boundaries.

2.2 Compile and Exec

A makefile comes with Sciara-fv2 that can be used for both building and executing the program on the datasets described in the previous section. To build the executable, both the serial and the OpenMP ones, namely `sciara_serial` and `sciara_omp`, 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 16000 steps:

```
make run
```

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

```
make run_omp
```

The number of threads for the multi-core execution is set to 2. If you want to use more then 2 threads, simply edit the Makefile and change the value of the `THREADS` variable.

The (default) simulation produces the following output files:

```
output_2006_000000016000.cfg
output_2006_000000016000_EmissionRate.txt
output_2006_000000016000_Morphology.asc
output_2006_000000016000_SolidifiedLavaThickness.asc
output_2006_000000016000_Temperature.asc
output_2006_000000016000_Thickness.asc
output_2006_000000016000_Vents.asc
```

The Qgis application can be used to inspect most of them, namely those in the ascii grid format as raster layers. The final configuration of the system is shown in Figure 2. Note that the md5sum checksum is assessed based on the `output_2006_000000016000_Temperature.asc` output file. On the JPDM2 workstation, which runs `g++` version 10.2.1, the checksum gives the following result:

```
704a4a65d1890589e952b155d53b110d
```

FIGURE 2: Sciara reference simulation as visualized by Qgis. The surface topography and the final solidified lava flow are displayed.

3 The Assignment

The assignment consists in the implementations of several CUDA version of Sciara-fv2, comparative performance assessment and Roofline analysis.

3.1 CUDA Implementations

The CUDA parallel implementations to be developed, each using CUDA unified memory (i.e., allocating memory by `cudaMallocManaged()`) are listed below[1]:

1. `Global`, namely a straightforward parallelization using the global memory only.

2. `Tiled`, namely a CUDA shared memory based tiled parallelization without halo cells.

3. `Tiled_wH`, 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.

4. `CfAMe`, namely the memory-equivalent conflict-free tiled algorithm. This algorithm require a revision of the orignal simulation model. Please, refer to [1] for further details.

[1] Assess the correctness of each parallel implementation with respect the md5 checksum of the serial/OpenMP versions.

5. `CfAMo`, namely the memory-optimized conflict-free tiled algorithm [1]. This algorithm require a revision of the orignal simulation model. Please, refer to [1] for further details.

Please, note that even if the reference serial implementation is made by several files, you only have to update `Sciara.cpp` and `sciara_fv2.cpp`. In the first file, substates memory is allocated while in the second the elementary processes, the reduction and the simulation loop are defined.

Note also that, in most applications, not all the kernels need to be updated to a tiled implementation. It is a your responsibility to chose (by profiling) which Sciara-fv2 kernel would benefit of a tiled implementation.

You are also invited to inspect the kernels' code for optimizing it and for assessing possible flow-divergence issues (if any), and to adopt suitable countermeasures in the case. Moreover, pay attention to have coalesced access when you implement the Sciara-fv2 reduction kernel (just use the correct access pattern, as described in the Chapter 10 of the reference CUDA textbook [2]).

3.2 Performance Assessment

Assess the performance of the developed implementations in terms of speed-up with respect to the `Straightforward_Unified` implementation. Try different configurations 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 time of the best execution of each developed parallel version in a table, together with the block configuration used.

3.3 Applying the Roofline Model

Define the Roofline plot for the GTX 980 GPU, for both the global and the shared memory. You can either use the ERT (Empirical Roofline Toolkit) [5] or the *gpumembench* [3] micro-benchmark[2]. 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 or memory bound.

[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 Report and Presentation

Eventually, summarize the outcomes in a very brief report (3/4 pages) by using the LeTeX LNCS class. For each parallel version implemented, report the best achieved results by means of meaningful plots and tables. In the tables reporting the numerical data, also insert details about the CUDA grid used (how many block and how many threads per block), as done in [1].

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.

Eventually, arrange a brief presentation (no more than 10 slide) to present the work done and the achieved results.

References

[1] D. D'Ambrosio, A. De Rango, S. Fiorentino, F. Barbara, A. Rizzo, G. Mendicino, and P. Sabatino. Efficient Execution of Flow-Based Low-Order Stencil Algorithms on GPUs. Available at SSRN: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=5056757 (also aivailable at: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=5056757.

[2] Wen mei W. Hwu, David B. Kirk, and Izzat El Hajj. Chapter 10 - Reduction. In *Programming Massively Parallel Processors: A Hands-on Approach*, pages 211–233. Morgan Kaufmann, fourth edition, 2022.

[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] W. Spataro, M.V. Avolio, V. Lupiano, G.A. Trunfio, R. Rongo, and D. D'Ambrosio. The latest release of the lava flows simulation model SCIARA: first application to Mt Etna (Italy) and solution of the anisotropic flow direction problem on an ideal surface. In *Procedia Computer Science*, volume 1, pages 17–26, 2010. Available at: https://www.sciencedirect.com/science/article/pii/S1877050910000050.

[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.