Exam

December 14, 2022 – Duration: 2 hours.
Authorized documents: *slides and personal notes from the lectures.*

- Cell phones should be turned off and stored in bags.
- This document contains 2 pages.
- Please avoid using a pencil.
- The grading points are indicative, not definitive.

Exercice 1 – Warm-up (4 points)

1. What is the difference between the synchronous mode and the eager mode when sending MPI messages?
   What are their respective advantages and inconvenients?
   Which mode corresponds to the standard send (MPI_Send())?
2. For best GPU performance, should we use one thread per GPU core? or multiple threads per GPU core?
   Why?

Exercice 2 – Heat diffusion on a grid (16 points)

We study the phenomenon of heat diffusion on a grid over several time steps. The temperature at each grid point is implemented by a floating-point number (in single precision). In this subject, we limit ourselves to a simplified version of the diffusion phenomenon. The computation of the temperature $u_{i,j}^{t+1}$ for the point of coordinates (i,j) at time $t+1$ is given by the following equation:

$$u_{i,j}^{t+1} = \frac{1}{5}\left(u_{i-1,j}^{t} + u_{i,j-1}^{t} + u_{i+1,j}^{t} + u_{i,j+1}^{t} + u_{i,j}^{t}\right) \quad (1)$$

Each point therefore retains a fifth of its former value, to which are added the contributions received from its four direct neighbors (North, South, East, West).

The algorithm ends when $\Delta < \epsilon$, where the error Δ is defined as:

$$\Delta = \max_{\forall(i,j)} \left| u_{i,j}^{t+1} - u_{i,j}^{t} \right|.$$

A possible implementation of this algorithm relies on two grids: the grid #0 containing the values at time t, we compute for the time $t+1$ the grid #1 using the grid #0. At each time step, the grids are switched. Each grid is stored as a 2-dimensional array of size $n \times n$. We will generally consider that $n >> p$, where p is the number of processes.

In order to simulate an infinite grid, the computations on the edges will be performed periodically. Namely, for the first line ($i = 0$), the values $u_{i-1,j}^{t}$ will correspond to the values $u_{n-1,j}^{t}$. And for the last line ($i = n-1$), the values $u_{i+1,j}^{t}$ will correspond to the values $u_{0,j}^{t}$. The same goes for the first and last columns.

In practice, we have a function init (G, h, l) which initializes a grid of size h × l starting at address G: all points of the grid are initialized at 25 degrees, except for some columns initialized at 100 degrees.

The sequential algorithm is given in Algorithm 1.

We first consider a multi-process parallelization, using p MPI processes, p divisor of n.

Algorithm 1 Sequential algorithm of heat diffusion on a grid

Require: n and ϵ

```
memory allocation for grids G[0] and G[1] of size n × n
init(G[0], n, n)
g ← 0
repeat
    computation of G[1 − g] from G[g], and of Δ between G[1 − g] and G[g]
    g ← 1 − g
until (Δ < ε)
save G[g] (in a file)
memory release of G[0] and G[1]
```

1. (1 point) How to update the termination condition in parallel?
   How can you implement this in MPI? (just give the principle)
2. (1 point) Which type of load balancing is relevant for this parallelization? Justify.
3. (6 points) Write the corresponding parallel algorithm in MPI using a 1D block distribution in the i dimension. You will use the function compute(G_target, G_source, h, l) which computes (according to equation (1)) lines 1 to $h-2$ of the grid of size $h \times l$ starting at address *G_target*, from lines 0 to $h-1$ of the grid of size $h \times l$ starting at address *G_source*. This function also returns the corresponding Δ value.
   As far as MPI communications are concerned, you can avoid specifying the details regarding pointers by precisely indicating (for example) which lines are concerned. For the MPI functions, you will specify the function used and its important parameters but you can omit the exact C programming syntax. All the local variables and their type must be specified.
   The complete grid will be collected by the process with rank ROOT in order to be written to disk.
4. (1 point) Is it possible to overlap communications by computations?
   If yes: present the principle of the corresponding algorithm; otherwise, justify your answer.

We now consider a CUDA deployment of this application on one GPU, considering a CPU sequential execution (i.e. without MPI).

5. (1 point) Is it possible to perform one single kernel call to run this application on GPU? Justify.
6. (4 points) Write the corresponding CUDA program (with both CPU and GPU code). The memory allocations on GPU must be specified, as well as the required CPU-GPU memory transfers, the number of dimensions of the thread blocks and of the block grid, as well as the kernel(s) code in pseudo-code or in CUDA (all details regarding parallelism (indices, ...) must be clearly specified), and the kernel call(s).
   The thread block size can be considered as constant here (with for example 256 threads per bloc).
7. (1 point) How can we improve the GPU performance? (no detailed algorithms required here; only the overall ideas)

---

8. (1 point) What is the compute intensity of this application? What can you (most likely) deduce about the hardware feature limiting the performance of this application on CPU and on GPU?