UCLouvain EPL-INMA 18 juin 2025	**Exam : LINMA2710** Scientific Computing	First name : Last name : NOMA :

Important instructions:

- The exam duration is 2 hours.
- Please start your answer just below each question (when possible) and continue on a new sheet of paper if necessary. Put your name on all sheets.
- As mentioned in the course outline, only personal handwritten notes and annotated printouts of the material posted on the course Moodle page are allowed. All the rest, including electronic devices (laptop, smartphone, etc.), is forbidden.
- Do not use red ink.
- The size of the answer box is chosen with a quite conservative margin so do not worry if your answer is shorter than the box.

UCLouvain EPL-INMA 18 juin 2025	**Exam : LINMA2710** Scientific Computing	First name : Last name : NOMA :

1. Consider this C++ code, in which lines 5, 7, 25, and 26 have been hidden:

```
1  #include <iostream>
2  using namespace std;
3
4  class Rectangle {
5
6  public:
7
8    int area(void) { return mWidth * mHeight; }
9  };
10
11 Rectangle::Rectangle(int w, int h)
12 {
13     mWidth = w;
14     mHeight = h;
15 }
16
17 int main() {
18   Rectangle * p_r;
19   Rectangle * a_r;
20   p_r = new Rectangle (1, 2);
21   a_r = new Rectangle[2] { {3,4}, {5,6} };
22   cout << "*p_r's area: " << p_r->area() << '\n';
23   cout << "a_r[0]'s area:" << a_r[0].area() << '\n';
24   cout << "a_r[1]'s area:" << a_r[1].area() << '\n';
25
26
27   return 0;
28 }
```

   (a) Write the four missing lines directly in the code.

   (b) Briefly justify your answers.

   (c) What output does line 24 produce? Briefly explain why.

2. Consider the FTCS (forward time, centered space) scheme for the following diffusion problem:

$$
\begin{aligned}
&u_t(t,x) = \kappa\, u_{xx}(t,x), \quad x \in [0,1], t \geq 0,\\
&u(t,0) = 0 = u(t,1) \quad \text{for all } t \geq 0,\\
&u(0,x) = u^0(x) \quad \text{for all } x \in [0,1].
\end{aligned}
$$

   (a) Write the FTCS scheme, including the boundary and initial conditions.

   (b) Briefly give the meaning of each symbol (excluding the basic arithmetic, of course) that you have introduced. You can use a mix of words and equations.

   (c) Explain the notion of *refinement path*.

   (d) Give (without proof) a refinement path along which the FTCS scheme is stable. Explain what this means.

   (e) Give (without proof) a refinement path along which the FTCS scheme is *not* stable. Explain what this means.

3. Consider the time integration of the following diffusion equation:

$$\frac{\partial u(\theta,t)}{\partial t} = \alpha \frac{\partial^2 u(\theta,t)}{\partial \theta^2}.$$

The solution is also assumed to be cyclic, i.e., $u(\theta + 2\pi, t) = u(\theta, t)\ \forall \theta \in \mathbb{R}$. The solution is discretized over n equally spaced points in θ: $\theta_1 = 0, \theta_2 = 2\pi/n, \theta_3 = 4\pi/n, \ldots$.

You have written the following OpenCL kernel to integrate the equation over `num_iter` time steps. The constant F is $\alpha\Delta t/(\Delta x)^2$ where $\Delta x = 2\pi/n$.

Simplifying assumption *This kernel assumes that the number of processing elements of a compute unit is at least n so all threads are in the same compute unit. You do not have to consider any situation in which this is not the case.*

```
1  __kernel void _diffuse(__local float* a, float F, int num_iter) {
2      int i = get_local_id(0);
3      int prev, next, iter;
4      if (i == 0)
5          prev = get_local_size(0) - 1;
6      else
7          prev = i - 1;
8      if (i == get_local_size(0) - 1)
9          next = 0;
10     else
11         next = i + 1;
12     for (iter = 0; iter < num_iter; iter++)
13         a[i] = a[i] + F * (a[prev] + a[next] - 2 * a[i]);
14 }
15
16 __kernel void diffuse(__global float* glob, __local float* shared, float F, int
       num_iter) {
17     int i = get_global_id(0);
18     shared[i] = glob[i];
19     _diffuse(shared, F, num_iter);
20     glob[i] = shared[i];
21 }
```

You observe that your kernel does not always provide the same answer, the answer even seems to be incorrect.

Answer the following questions:

(a) What is the issue causing the kernel to produce incorrect results ?

Solution: We are missing synchronization. So a thread could be using the wrong value of `a[prev]` or `a[next]` in case they are not synchronized.

Note: In view of the simplifying assumption, the local and global sizes are the same so `get_local_id(0)` and `get_global_id(0)` are the same. This is not the source of issues.

(b) You identify that depending on your device, if n is sufficienty small you always get the correct answer. More precisely, you get the correct answer in the following situations:

- $n \leq 8$ and you use your CPU as device for your kernel,
- $n \leq 32$ and you use your GPU as device for your kernel.

Explain why.

Solution: If n is that small then the execution can be run using SIMT in which case the threads are guaranteed to be executed each instruction at the same time.

(c) How can you fix this issue you identified in a. ?

Solution: We should add `volatile` and also add barriers to synchronize the threads.

4. After the success of the kernel you developed in "Question 3", you now want to target applications with much larger n. For this reason, you now want to solve it with several compute nodes. You have access to m compute nodes and enough material to connect m pairs of compute nodes together. Using these connections, the communication of k bytes takes a time equal to $\alpha + \beta k$.

   Answer the following questions:

   (a) How would you interconnect the m compute nodes ?

   Solution: We could use n/m adjacent values for each node and then connect each node with the two nodes holding adjacent values as a ring. A tree would also be possible, it would only require connecting $m - 1$ pairs but some neighbors won't be connected directly in a tree and will need $\log_2(m)$ communications.

   (b) How much time would the communication take at each time step of the integration in terms of α, β, m, n ?

   Solution: We can first communicate the values of `prev` and then the values of `next`. That's two communications, each of 4 bytes:

   $$2\alpha + 8\beta.$$

   (c) You would like to compute $\sum_{i=1}^{n} |u(\theta_i, t)|$ to verify whether the solution is diverging. Which MPI collectives would you use ? How much time would this take in terms of α, β, m, n and also γ, the time taken by the compute node to execute one arithmetic operation.

Solution: We can use `MPI_reduce` with `MPI_sum`. We cannot use the classical complexity of `MPI_reduce` with $\log_2(m)$ since we are using a ring. As the diameter is $m/2$, we need at least $m/2$ communications. Each communication only needs to communicate the value of the sum so it is only 4 bytes. The complexity of doing the sum in each node is $2n/m\gamma$ (we multiply by 2 since we both need to do `+` and the absolute value). One could also consider that each node had access to at least n/m threads in which case the complexity drops down to $\log_2(n/m)\gamma$, this was also accepted. We also need to merge the partial sums. This case the complexity depends on the approach, both were accepted (since this part of the complexity would hardly be the bottleneck): Let $x_1, \ldots, x_m$ be the partial sums.

- We could
  1. first sum x_1, x_2 into x_2 on node 2 and sum x_{m-1}, x_m into x_{m_1} on node $m-1$.
  2. Next we merge x_2, x_3 into x_3 on node 3 and x_{m-1}, x_{m-2} on node $m-2$,
  3. etc...

  The complexity will then be $m/2\gamma$.
- We could
  1. sum x_{2i}, x_{2i-1} into x_{2i} on node $2i$ for $i = 1, \ldots, m/2$.
  2. Then we communicate x_{4i-2} to x_{4i-1} (no arithmetic operation needed) on node $4i-1$ for $i = 1, \ldots, m/4$.
  3. Then we sum x_{4i-1}, x_{4i} into x_{4i} on node $4i$ for $i = 1, \ldots, m/4$.
  4. Then we communicate x_{8i-4} to x_{8i-3} (no arithmetic operation needed) on node $8i-3$ for $i = 1, \ldots, m/8$.
  5. Then we communicate x_{8i-3} to x_{8i-2} (no arithmetic operation needed) on node $8i-2$ for $i = 1, \ldots, m/8$.
  6. Then we communicate x_{8i-2} to x_{8i-1} (no arithmetic operation needed) on node $8i-1$ for $i = 1, \ldots, m/8$.
  7. etc...

  This would require more communication but since they are in parallel it gives the same amount of communication time (but this might consume more energy though). The complexity in terms of γ is decreased to $\log_2(m)\gamma$.

In total, we have

$$m/2(\alpha + 4\beta + \gamma) + 2n/m\gamma.$$

As it is a complexity, constants didn't matter so, the following was also accepted for instance:

$$m(\alpha + \beta + \gamma) + n/m\gamma.$$