UCLouvain EPL-INMA 16 June 2026	**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.

---

1. Consider this C++ code, in which lines 14–17 have been removed:

```
1  #include <cmath>
2  #include <iostream>
3  using namespace std;
4
5  int operate (int a, int b)
6  {
7    return (a*b);
8  }
9
10 double operate (double a, double b)
11 {
12   return (a/b);
13 }
```

```
19 int main ()
20 {
21   int m=5,n=2;
22   double y=5.0,z=2.0;
23   cout << operate (y,z) << '\n';
24   cout << operate (m,n) << '\n';
25   cout << operate (y,n) << '\n';
26   return 0;
27 }
```

   (a) What output does line 23 produce? Briefly explain.

   (b) What output does line 24 produce? Briefly explain. What programming features does this illustrate?

(c) The prototype of function `pow`, that returns `base` raised to the power `exponent`, is
`double pow (double base, double exponent)`.
Fill out lines 14–17 in the box above such that line 25 prints 5.0 to the power 2, namely 25.0.

(d) Does the completed code perform narrowing cast or widening cast? Briefly explain.

2. Consider Poisson's equation

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2} = 1$$

in a cube $\Omega = \{(x, y, z) \in \mathbb{R}^3 : 0 \leq (x, y, z) \leq L\}$. The boundary conditions are zero Dirichlet conditions. Consider an axis-aligned grid with steps $\Delta_x = L/J_x$, $\Delta_y = L/J_y$, $\Delta_z = L/J_z$.

(a) Write a finite volume scheme for this problem based on the given grid.

(b) Explain how you obtained the scheme.

3. You are tasked to implement a matrix-matrix-transpose product in the context of transformer inference. One key component of this model is the attention head where, given a matrix of queries Q and a matrix of keys K and you need to compute the dot products of all pairs of rows. This amounts to computing the entries (the so-called *attention scores*) of the following R matrix:

$$R = QK^\top.$$

You launch 3 AI coding agents to solve the same task of computing this product and they produce 3 different solutions:

IJK solution:

```
void ijk(const float *Q, const float *K, float *R, int n) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            R[i * n + j] = 0.0f;
            for (int k = 0; k < n; k++)
                R[i * n + j] += Q[i * n + k] * K[j * n + k];
        }
    }
}
```

IKJ solution:

```
void ikj(const float *Q, const float *K, float *R, int n) {
    memset(R, 0, (size_t) (n * n * sizeof(float)));
    for (int i = 0; i < n; i++)
        for (int k = 0; k < n; k++) {
            for (int j = 0; j < n; j++)
                R[i * n + j] += Q[i * n + k] * K[j * n + k];
        }
}
```

Tiling solution[1]:

```
void block_tile(const float *Q, const float *K, float *R, int n, int b) {
    for (int bi = 0; bi < n; bi += b)
        for (int bj = 0; bj < n; bj += b) {
            for (int i = bi; i < bi + b; i++)
                for (int j = bj; j < bj + b; j++)
                    R[i * n + j] = 0.0f;
            for (int bk = 0; bk < n; bk += b)
                for (int i = bi; i < bi + b; i++)
                    for (int j = bj; j < bj + b; j++)
                        for (int k = bk; k < bk + b; k++)
                            R[i * n + j] += Q[i * n + k] * K[j * n + k];
        }
}
```

We will assume that

[1] To simplify, we assume that n is a multiple of b.

1. the three matrices are square $Q, K, R \in \mathbb{R}^{n \times n}$,
2. the `float` entries are represented as 32-bit floating point numbers,
3. the addresses of the first entries of Q, K and R are aligned with the start of a cache line,
4. the matrices are represented in a row-major format, which means that two entries of the same row and adjacent columns are adjacent in memory,
5. the dimension n is so large that even a single row of Q, K or R does not fit in the L1 cache.

(a) You first benchmark the IJK and IKJ solutions and notice that one is significantly faster than the other one. Which one do you think is faster[2] ? Explain why.

(b) Given that the size of the L1 cache is 64 KiB (so 2^{16} bytes), what value of b should you take for the tiling solution ? Explain why.

[2]Note that this is not the classical matrix-matrix product, since the left-hand side is multiplying by the **transpose** of the right-hand side.

(c) Given a cache line size of 64 bytes, how many cache lines will the L1 cache need to fetch[3] for each of the 3 solutions ?

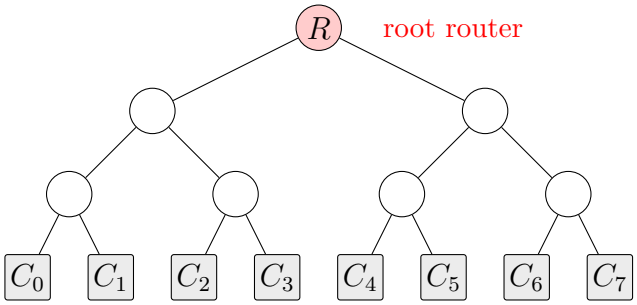
[3]from the L2/L3 or main memory. To simplify, you can ignore the L2 and L3 caches for this question.

(d) You now port the tiling solution to a GPU using OpenCL. Instead of tiling for the L1 cache, you tile for the on-chip *shared memory*: each work-group computes one $b \times b$ tile of R with $b \times b$ work-items that cooperatively stage one tile of Q and one tile of K into a buffer shared by the whole group, synchronize, and then accumulate the product out of that buffer. Fill in the missing lines in the following kernel.

```
#define b 32

__kernel void tiled_qkt(__global const float *Q,
                        __global const float *K,
                        __global float *R,
                        const int n) {
    __local float BQ[b][b];
    __local float BK[b][b];

    const int li = get_local_id(1);
    const int lj = get_local_id(0);
    const int i  = get_global_id(1);
    const int j  = get_global_id(0);
    const int bj = get_group_id(0) * b;

    float acc = 0.0f;

    for (int bk = 0; bk < n; bk += b) {
        BQ[li][lj] = Q[i * n + (bk + lj)];
        BK[li][lj] = K[(bj + li) * n + (bk + lj)];
```

```
    }

    R[i * n + j] = acc;
}
```

4. You now train your transformer on a cluster. The cluster's interconnect is wired as a *binary tree*: the 8 **leaves** are the compute nodes $C_0, \ldots, C_7$ and every **internal node** is a router. Each router has at 4 available ports. Each non-root router uses one port towards its parent and one towards each of its two children, leaving one unused. We assume that all links and ports have the same bandwidth.



   (a) Computing the attention scores $R = QK^\top$ in a distributed way requires, at one point, a *transpose* of a matrix that is spread across the 8 compute nodes: each node initially holds a block of rows and, after the transpose, must hold a block of columns, so every node sends a piece of its data to every other node. You time this exchange and it is far slower than the individual link speeds would lead you to expect. Analyse how this collective behaves on the tree interconnect: what limits its running time, and how does that limit change if the cluster keeps growing as a binary tree? Justify your answer.

(b) The company buys 5 **more routers** (same 4-port routers), so you now have $7 + 5 = 12$ routers in total. Explain how to use these 5 new routers to improve the running time of the transpose operation.

5. While inspecting the cluster, you discover that one of your colleagues Alice has been mining bitcoin on the compute nodes whenever no user job is running on them, i.e. he keeps the otherwise-idle nodes busy at 100% utilization.

   Bob complains that this is inflating the electricity bill. Alice replies:

   > *"The nodes have to stay powered on anyway as they must be ready the instant a user asks for compute, so we are not allowed to switch them off. A node that is on consumes its power regardless of what it does, so it draws the same whether it sits idle or mines bitcoin. I might as well put those idle cycles to use."*

   We grant Alice her premise: the nodes **must stay on and ready**, so the question is *not* whether they could be switched off. Taking that as fixed, is the rest of Alice reasoning "that an **on-but-idle** node draws the same power as an **on-and-fully-loaded** one" correct? Justify quantitatively.