Parallel algorithms and programs
TP 5: Shared memory LU factorisation

Jean-Marie Madiot, Julien Herrmann
jeanmarie.madiot@ens-lyon.fr, julien.herrmann@ens-lyon.fr

Before, we used a **distributed-memory** paradigm with MPI. In this session, we will used a **shared-memory** paradigm, with OpenMP (OpenMulti- Processor). This is an extension of the C, C++ and Fortran programming languages, with a set of `pragmas` that allows the programmer to simply express parallelism, given a sequential program:

```
#pragma omp parallel
{
  for(i = 0; i < n; i++)
    a[i] = a[i] << 10 + b[i];
}
```

MPI deals with processes, OpenMP deals with threads. The model of parallelism is similar to the theoretical PRAM model and CRCW machine such that all threads can access the same memory locations in concurrent reads and writes (with unpredictable results on concurrent writes) but where the execution time of instructions is not ensured to be the same (there is no synchronism of the instructions). You can find a detailed specification of OpenMP for the C programming language at:
`http://openmp.org/mp-documents/OpenMP3.1-CCard.pdf`

Question 1 Fill `tp5.c` to get a simple implementation for LU factorisation. Use the following algorithm:

```
L = identity matrix
U = zero matrix
for i = 0 to n-1:
  for j = 0 to n-1:
    if j <= i:  U[j][i] = A[j][i] - sum(k=0, j-1, L[j][k] * U[k][i]);
    if j >= i:  L[j][i] = (A[j][i] - sum(k=0, i-1, L[j][k] * U[k][i])) / U[i][i];
```

Question 2 Add OpenMP primitives to `tp5.c`, run it. What is the speed-up?

Question 4 In fact, adding OpenMP primitives is not enough: one must first change the sequentiality of the basic operations.

Use the 2D bloc cyclic scheme, still in OpenMP (using the routines `trsml`, `trsmu` and `gemm`) to make your program faster. Applying OpenMP primitives on the `for` loops inside which `gemm` is called.

Question 4 Use the sequential algorithm to implement `getrf` in TP 4, and finish TP 4.