HPC for mathematicians
Assignment for week 3

Andrés Miniguano Trujillo
Andres.Miniguano-Trujillo@ed.ac.uk

Exercise 2

In this task, we worked on parallelising the product of two matrices A and B, namely $C = A \times B$, with all of these of size N. It is known that the former matrices are given by the formulae

$$A_{i,j} = (N - j + i + 1)i \qquad \text{and} \qquad B_{i,j} = (j + i)(N - j + 1).$$

The task was given as follows:

1. Employ `nproc` processes to divide the task of multiplication between them.

2. Only the root process generates matrix B and broadcasts it to all other processes.

3. Each process creates an assigned number of rows of matrix A.

4. Each process multiplies its assigned rows by the entire B and stores the corresponding rows of C.

5. Finally, the root process gathers all the information and presents the final matrix C.

The code for this task is presented in file `Matrix.cpp`. Here, we will comment some details of its implementation and results.

Implementation details

In the file, the number `nproc` is labelled as `size`. As we want to test parallel code, we require `size` ≥ 2 to at least have one master root process to assign some work to another process. This way, we split `size` into two categories `Master` and `workers`[1]. The former is the root process, that assembles matrix B, distributes work, collects the feedback from each process, and presents the final matrix. The latter counts the number of working processes which will compute the rows of matrix C.

Root process

To assign the work to each working process, the root computes the exact division `n_rows` $=$ `N/workers` $\in \mathbb{N}$ and its residual `n_extra` $=$ `N%workers` $\in \mathbb{N}$. The first quantity is to distribute an equal number of rows to each process, while the second quantity allows to add some extra work when `nproc` $\neq 0 \pmod N$.

Once we have these quantities, we initialise a counter `buffer` $= 0$ and do as follows for each worker:

1. If `nproc` $= 0 \pmod N$, we have that `n_extra` $= 0$, for we assign exactly `n_rows` rows to each process, which we call `assigned_rows`.

[1] Here we identify `Master` $= 0$ and `workers` $=$ `size` $- 1 =$ `nproc` $- 1$.

2. If nproc $\neq 0 \pmod N$, we have that n_extra > 0. Here the variable assigned_rows will not be equal for all processes. As a result, we contemplate two cases: (a) The label of the process is less or equal than n_extra, then the process will have an assigned number equal to n_rows $+ 1$. (b) The label of the process is greater than n_extra, for the process will have an assigned number equal to n_rows. Notice that this assignment will distribute homogeneously the amount of work: On the one hand, if $N <$ nproc, then the first N processes will have one row assigned to them. On the other hand, if $N >$ nproc, then n_extra processes will have an additional row to work with.

3. Finally, we do the update buffer $\leftarrow$ buffer $+$ assigned_rows to provide a marker to the next processor of how many rows of A have been assigned already.

Once the job has been distributed, we can receive each row of C from the processes. These can be identified as

$$(C_{\ell,j})_{1\leq j\leq N} \qquad \forall\, \texttt{buffer} \leq j+1 < \texttt{assigned\_rows}.$$

Worker processes

Each worker has three tasks: It has to build a part of A, then multiply it by B, and then send the results as an update to C.

Of these three, it is only important to mention how the rows of A are identified as the product with B follows a similar treatment and sending the corresponding components of C is already detailed for the root process.

With this clear, we use the same notation as above and recall that each worker has available the pair (buffer, assigned_rows). The first indicates the initial row of A that has to be computed, and the second gives a stopping criteria. As a result, we need to compute

$$(A_{\ell,j})_{1\leq j\leq N} \qquad \forall\, \texttt{buffer} \leq j+1 < \texttt{assigned\_rows}.$$

Notice that we can also add an if statement before computation, such that if assigned_rows $= 0$, then the worker process does not have to do anything at all.

COMPUTATIONAL RESULTS

The resulting code successfully computed and displayed C for various choices of nproc and N. In particular, Table 1 presents the computing time in seconds required for different values of each quantity. In this setting, the wall-time was limited to 10 minutes and the short job flag was used.

Table 1
Results of the task for different number of processes and matrix dimensions

Processes	**Computing time**								
nproc	$N=3$	$N=9$	$N=27$	$N=81$	$N=243$	$N=729$	$N=2\,187$	$N=5\,000$	$N=6\,561$
2	2.3×10^{-5}	3.6×10^{-5}	2.7×10^{-4}	4.9×10^{-3}	1.4×10^{-1}	3.997	1.1×10^{2}	–	–
4	3.3×10^{-5}	3.2×10^{-5}	1.2×10^{-4}	1.7×10^{-3}	4.7×10^{-2}	1.338	3.8×10^{1}	–	–
8	2.8×10^{-5}	3.8×10^{-5}	8.9×10^{-5}	8.6×10^{-4}	3.3×10^{-2}	5.9×10^{-1}	1.6×10^{1}	3×10^{2}	–
16	3.3×10^{-5}	4.7×10^{-5}	1.2×10^{-4}	8.8×10^{-4}	1.8×10^{-2}	2.8×10^{-1}	7.314	1.4×10^{2}	3.2×10^{2}
32	5.4×10^{-5}	1.1×10^{-4}	2.8×10^{-4}	9.9×10^{-4}	1.0×10^{-2}	2.6×10^{-1}	6.742	7.2×10^{1}	1.6×10^{2}

We can see that the computing time increases quickly as N increases. For the cases $N <$ `nproc`, we see that increasing the number of workers does not imply a faster runtime, which is obvious as the processes that do not have any workload are being called to do nothing, and the root processes has to wait for them to finish. Likewise, for the cases $N >$ `nproc`, we see a clear decrease in computing time. In several of these, we can see that using two additional processes can boost computations halving computing times with respect to the previous configuration.

Finally, let us notice the big gap between $N = 5000$ and $N = 6\,561$. In this case, we can see that for `nproc` $= 32$, it takes twice as much time to compute the matrix for $N = 6\,561$. The difference is just $2 \times 1\,561$ additional dimensions between the matrices, for we can expect that for $N = 3^9$, the required time for `nproc` $= 32$ will exceed 15 minutes.