Matrix

Verardo Thomas

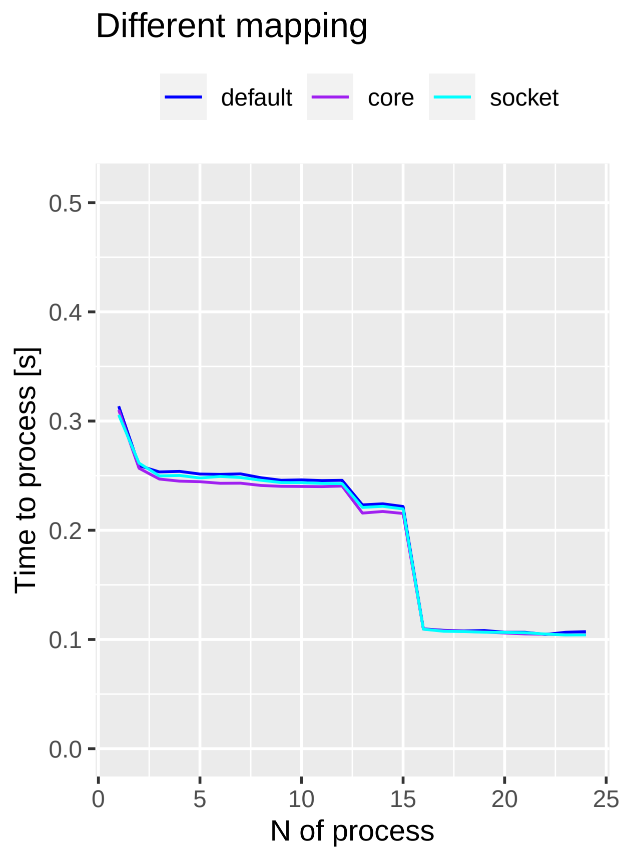
29/12/2021

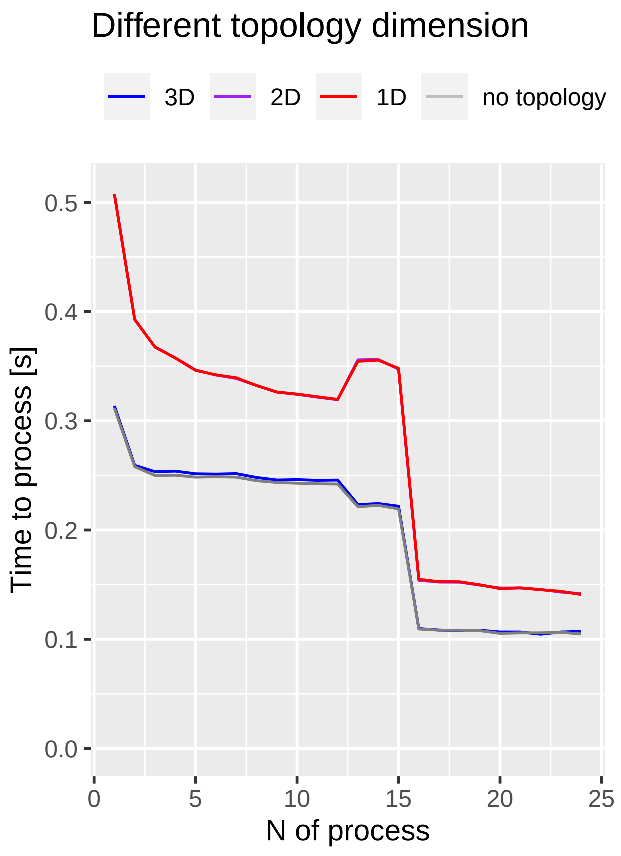
SECTION1 (Matrix)

The matrix is initialized with random number and, after building the topology, that is always (s, 1, 1) as s the numbers of processes, the matrix is first divided and then sent to all the processes. So, each process computed a part of the matrix, that is exactly the shape of the matrix $(x * y * z)$ divided by the number of processes.

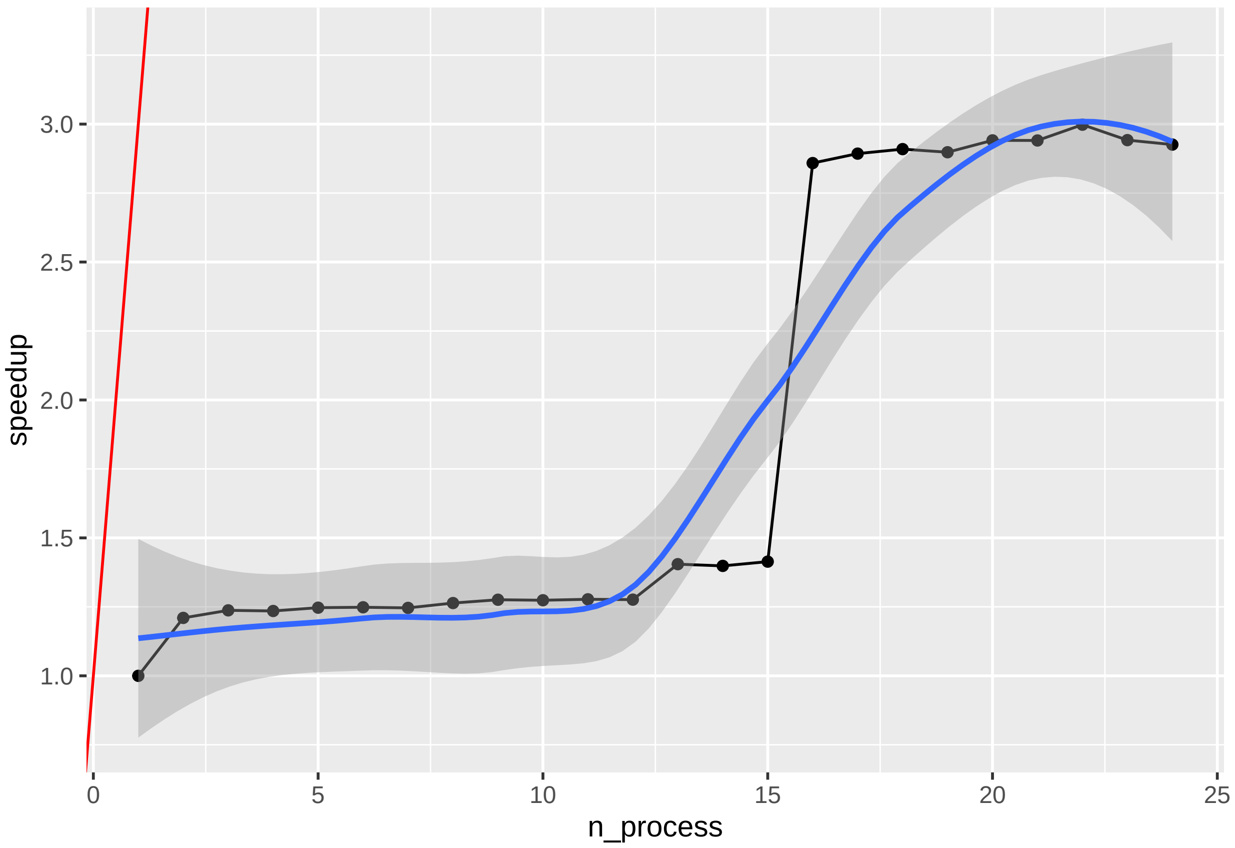
I produce different plots: the first one is not very useful and it represent the different shapes of the matrix in relation to the number of processes (the plot is in the LINK). In that graph, we can see that, if we use different shapes, that are 2400x100x100 (red), 1200x200x100 (orange) and 800x300x100 (yellow), the time to sum the matrix is always the same, even using multiple processes.

Here, there are two plots. The first is the plot with always the same matrix (1200x200x100), with the same topology (3D) but with different mapping and we can see the result doesn't change. In fact, the time to process is always similar. We note that for the first 15 processes executed in parallel, the time is quiet constant, but after 16 processes in parallel, the time to compute decrease drastically and remains constant until the end.





After this, I computed the time to do the matrix-matrix sum with the same mapping and the same shape (1200x200x100) but with different topology, expressed in the second plot. The 2D and 1D topology are the same, even because the matrix is a 3D array and the two topology are built with Nx, that is the number of processes ($[size]$ for 1D and $[size, 1]$ for 2D, as size the number of processes). We can see also that built a topology is useless because it has the same performance.



Furthermore, I calculate the speedup for the matrix with the shape of 1200x200x100 with the default configuration and with the 3D topology. To calculated the speedup, I used the formula:

$$S_p = \frac{T_1}{T_p}$$

Where T_1 is the best time with one single process and T_p is the time to calculated the matrix with p processes. Then I calculated the efficiency, that is $speedup/n_process$ and it's a value that varies between zero and one, and is a measure of how well our parallel algorithm uses processors. As seen from the red line that represent the best theoretical parallel efficiency, the efficiency is not really good.