Mesh multiplication of matrices using Open MPI

PRL of 2020/2021

Author: **Patrik Németh**
Login: **xnemet04**

1 The algorithm and its implementation

The mesh multiplication algorithm multiplies two matrices by utilizing multiple processes. The algorithm is initialized with n processors, where n is the product of the number of rows of the first matrix (r) and the number of columns of the second matrix (c) - i.e. $n = rc$. The processes are generally viewed as a mesh of r rows and c columns.

The following conventions are used in this text: the first input matrix is denoted as A, the second input matrix is denoted as B, the output matrix is denoted as C. For specific values of matrices their lowercase counterpart letters will be used with 0-based indices i (row) and j (column) counted from the top left corner of the matrix (i.e. the first value of matrix A would be $a_{0,0}$). Processes will be denoted and indexed similarly, only with the symbol P (the first process in the mesh is $P_{0,0}$)

The time complexity of this algorithm depends on the number of steps until the last values of A and B reach the last process, which is $r + c + m - 2$, where m is the shared dimension of A and B. This means, that the time complexity of the algorithm is $t(n) = O(n)$. The space complexity is given by the input as $p(n) = O(n^2)$. This results in a cost of $c(n) = t(n)p(n) = O(n^3)$.

Every process corresponds to a position of the output matrix C and is initialized by setting its output value $c_{i,j}$ to 0. The algorithm works by dividing the input matrices between the processes. This is done by feeding each row of A to the rows of the process mesh and each column of B to the columns of the process mesh. After receiving one value from each of the input matrices, an iteration of the $c_{i,j}$ value calculation in the form of $c_{i,j} = c_{i,j} + (a_{i,j-1}b_{i-1,j})$ is carried out. Each process, after its computation step, propagates the received values further by sending value a to the right and value b downward in the process mesh. Every process works until there are incoming values. If there are no more incoming values to $P_{i,j}$, it has finished computing value $c_{i,j}$. The processes can be divided into four groups based on behaviour:

- **root** - this group contains only the first process (if viewed as a mesh, mapped to the top left corner),
- **first row** - the first row of the mesh, excluding the root,
- **first column** - the first column of the mesh, excluding the root,
- **the rest** - this is the rest of the processes that do not fit into the other groups.

This division is not particularly important in the mathematical sense, but was used during the implementation of the algorithm.

1.1 The implementation

The program starts by parsing the first line of the input files. This is carried out by the root process. The parsed values, along with the file seek positions, are broadcast to every process. Next, processes of the **first row** and **first column** open the input files and seek to the position of the first values of the matrices. From this point on, no preemptive data loading is done, so all processes read data as needed, directly from the files. The next sections describe how each process group behaves. After every process finishes its computation, the root process gathers their respective c values and prints them in the correct format. A general sequence diagram can be seen in figure 1.

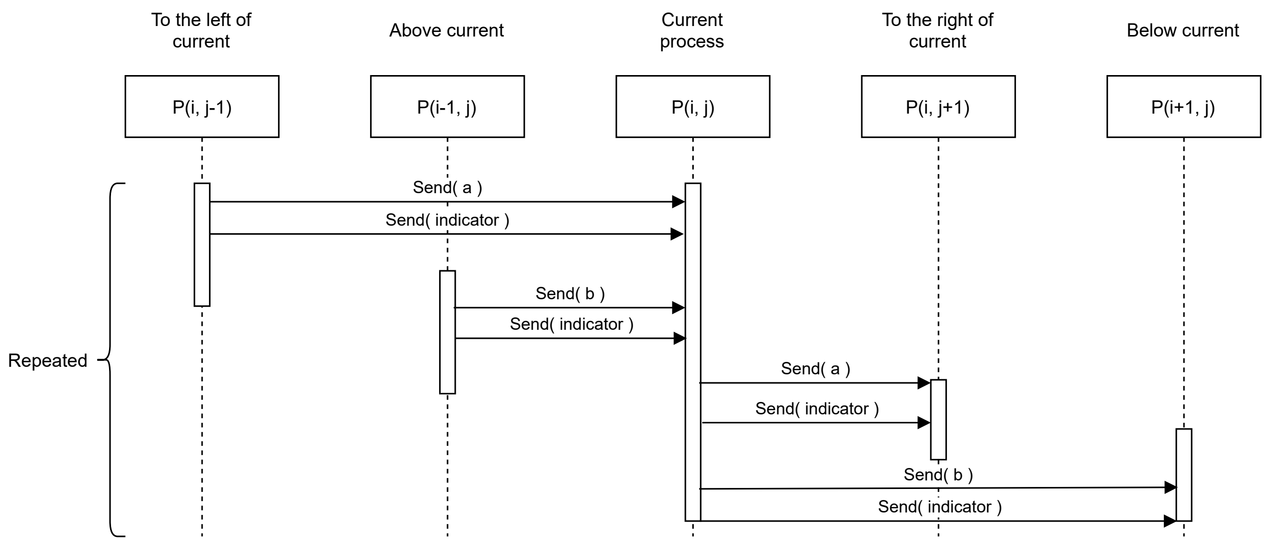


Figure 1: A general representation of the sequence of messages. Values a and b are values of matrices A and B respectively. The `indicator` indicates whether more values will be sent.

Root

This group contains only the root process. It is the only process that reads from both files. After reading the inputs, the next iteration of c is computed and the input values are propagated into the mesh. It finishes its computations after encountering a newline or EOF in the first file.

First row

During the first iteration, all processes in this group move to their respective initial positions in the second file. This means to the top of the `rank`-th column of matrix B. After this these processes read the next value from B and wait for the next value from A sent from the process to their left. The next iteration of c is computed and the incoming values are propagated into the mesh. If there are no more values to be sent, then a message indicating this is propagated as well.

First column

Similar to the previous group. During the first iteration all processes in this group move to their respective positions in the first file, so to the $\frac{\texttt{rank}}{\texttt{cols}}$-th row of matrix A. After this these processes read the next value from A and wait for the next value from B sent from theprocess to their top. The next iteration of c is computed and the incoming values are propagated into the mesh. If there are no more values to be sent, then a message indicating this is propagated as well.

The rest

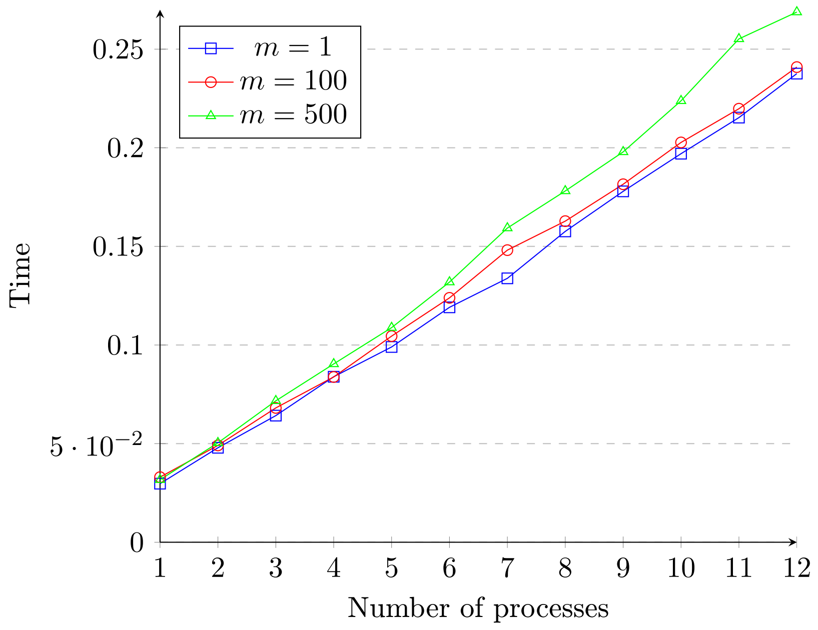
These processes listen to their neighbours for values, as well as to the message indicating no more incoming values. The next iteration of c is computed and all received messages are propagated further into the mesh. If a message indicating the exhaustion of input values is received, the process may finish the computation.

2 Experiments

Experiments were done on the *Merlin* server and the maximum number of processes used was 12. The experiments were conducted as follows.

For every combination of output matrix dimensions (that required at most 12 processes), 6 timed runs of the program were completed. The 6 runs were done for the purpose of averaging run times for a given matrix pair input, therefore during these 6 runs the input matrices did not change.

As only the number of rows of matrix A and the number of columns of matrix B dictate the dimensions of the output matrix C, three variations of the experiment were conducted. These variations change the size of the shared dimension of A and B (dimenion m when $A = r \times m$ and $B = m \times c$). The sizes of this dimension were 1, 100 and 500 for the experiments. The results of the experiments are shown in the plot below.



3 Conclusion

As can be seen from the plot of experimentation results, the required computation time rises linearly with the size of the output matrices. The impact of the shared dimension of the input matrices can also be seen. It is apparent that the linearity of the function has not changed, however its slope has. This supports the theoretical complexity of the algorithm shown above.

The implemeted program has been tested on multiple inputs and works well with correct inputs. Malformed inputs may break the program or make it behave unexpectedly.