Parallel and Distributed algorithms - PRL 2020/2021

Mesh Multiplication

Lukáš Dobiš
xdobis01@stud.fit.vutbr.cz

1 Algorithm description

Mesh Multiplication (MM) is parallel matrix multiplication algorithm which for matrix **A** (*m*x*n*) and matrix **B** (*n*x*k*) uses processors arranged into *m*x*k* mesh configuration. Mesh rows are numbered $1, ..., m$ and mesh columns $1, ..., k$. Matrices **A** and **B** are fed into boundary processors in first column and first row of mesh processor grid. This is visualized on example mesh grid, for matrix **A** of size $m = 4$, $n = 5$ and matrix **B** of size $n = 5$, $m = 3$ on figure 1. Matrix **A** i-th row lags behind row $i - 1$ for $2 \leq i \leq m$, similarly matrix **B** j-th column lags behind column $j - 1$ for $2 \leq j \leq k$. This way value $a_{i,s}$ from matrix **A**, and value $b_{s,j}$ **B** arrive in $P(i, j)$ processor at the same time and where they will be multiplied. Product of such multiplications $1 \leq s \leq n$ is accumulated in each processor in $c_{i,j}$ element. After *n* multiplications in $P(i, j)$, element $c_{i,j}$ holds value of computed product matrix **C**. Overall processing of arrived value pair a and b, in process or $P(i, j)$ is following. Multiply a and b and add computed product to $c_{i,j}$, then send a to the right process $P(i, j + 1)$ unless $j = k$, and similarly send b to the bottom process $P(i - 1, j)$ unless $i = m$.

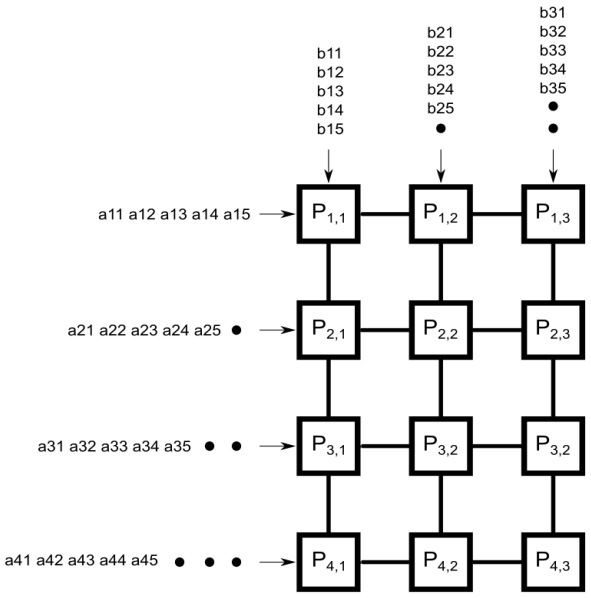

Figure 1: Example of processors arranged in mesh configuration

2 Complexity analysis

Time complexity depends on time it takes for elements $a_{m,1}$ and $b_{1,k}$ to arrive at $P(m, k)$ processor, since this processor is last one to get all his required values. Therefore equation for time complexity is following 1, which means algorithm has linear time complexity. And for processor cost complexity $p(n) = \mathcal{O}(n^2)$, overall cost equation 2 leads to cubic cost $\mathcal{O}(n^3)$, which is not optimal for matrix multiplication.

$$t(n) = m + k + n - 2 = \mathcal{O}(n) \qquad (1) \qquad\qquad c(n) = \mathcal{O}(n) \cdot \mathcal{O}(n^2) = \mathcal{O}(n^3) \qquad (2)$$

3 Implementation

Algorithm is implemented in programming language C++, with openMPI library for message passing between processors. Code of algorithm is in single file `mm.cpp`. For sending and receiving messages containing matrix value are used openMPI functions `MPI_Send` and `MPI_Recv`. Program starts with openMPI initialization, then each processor gets his process rank, which is used for message passing and to distinguish each processor behavior. Processors are numbered from 0 to $(m \cdot k - 1)$, where m is row dimension of first matrix and k is columns dimension of second matrix. After this part all processors diverge in behavior depending on their rank.

Main processor is indexed with rank 0 finds out how many processes are running. Then he reads matrices using `readMatrix` function into custom `Mat` structure which holds their dimensions and values. So to start, first he reads their m and k dimensions and values, from files "$mat1$" and "$mat2$". He checks if shared n dimension matches between both matrices, and if number of active processes matches product of $m \cdot k$, to have correct number of processes for algorithm. When every test passes, all other processes are initialized with m, k, n values to know how to compute rank of processes they are supposed to communicate with, during computation and how many multiplications should they compute. After initialization ends, main process starts loading values pairs from matrices according to 1 figure, and multiplies them, product is added to his (mesh grid) cell value. Following computation processor sends value from matrix **A** to processor right to his side, and value from matrix **B** to his bottom processor. Unlike other processors main processor after sending two values from his computation, also manages loading and passing inputs to boundary processors in first row and first column of mesh grid. After main processor accumulates his n products in his cell value, he waits for all processes to send him their computed cell value. When all cell values arrive main processor prints product matrix **C** onto standard output using `printMatrix` function.

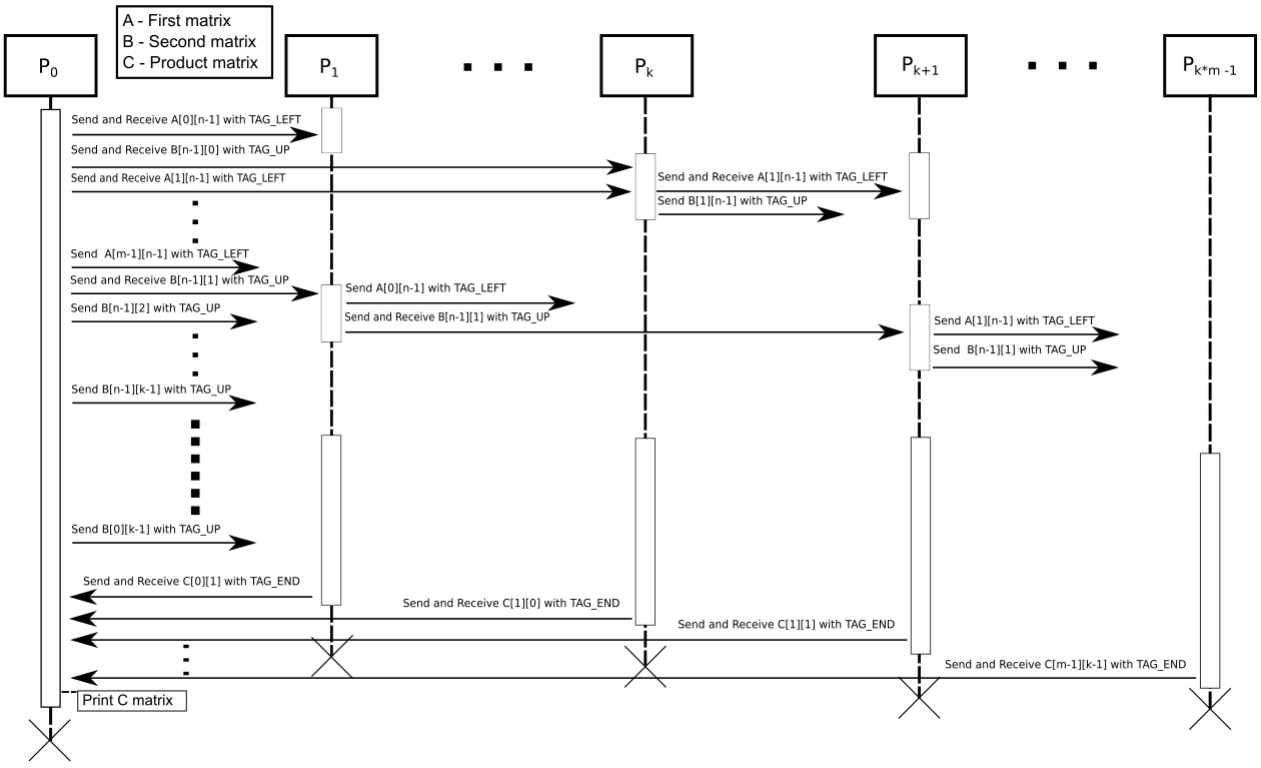

Figure 2: Communication protocol for message passing between processes

Other processors are initialized with m, k, n values. Then they make n products in while loop which are accumulated in their cell value, this value represents value of computed product matrix and is then send to main processor. Pair of values for computing product arrives either one value from main processor if they are in first row and second value arrives from neighbooring left processor, or if they in first column of mesh grid, one value is also from main processor and other value is from upper neigbouring processor. If processor is not in first row or column, then one value from pair arrives from neighbooring upper processor and other from left processor. After computing product, value from upper direction is send to the bottom processor, if processor is not in last row of mesh grid. And similarly value from left direction is send to the right processor, if processor is not in last column of mesh grid. Values passed from left to right, (even sended from main processor at beginning) are tagged with `TAG_LEFT` tag, and values passed from up to bottom, are tagged with `TAG_UP` tag. Each non main processor final message with their cell value for main processor is tagged with `TAG_END` tag. Communication protocol of all processors in mesh multiplication algorithm is on figure 3. Protocol on figure considers only matrices with following constraints on their dimensions $m > 2$ and $k > 2$.

4 Experiments

To test algorithm theoretical time complexity, experiments were conducted by running implemented algorithm on matrices of different sizes, to use up 1 to 15 processors. Result values are averages of 5 measurements for each number of processors. Values for following numbers of processors were interpolated: $7, 11, 13$. Time measurement was taken by `std::chrono::high_resolution_clock::now` function, on school server Merlin. Time was measured as duration from main processor start to his end. Plotted graph on figure 4 is linear which confirms theoretical time complexity, but there is deviation between 6 to 8 processors. This deviation could be explained by increased overhead, caused by greater management cost of message distribution to first row and column of mesh grid. Another hidden factor could be size of shared dimension n because $m = 2$ $n = 12$ and $n = 12$ $k = 2$ matrices have 48 multiplications on 4 processors, that is more multiplications per processor then for $m = 3$ $n = 3$ and $n = 3$ $k = 3$ matrices, which have 27 multiplications on 9 processors.

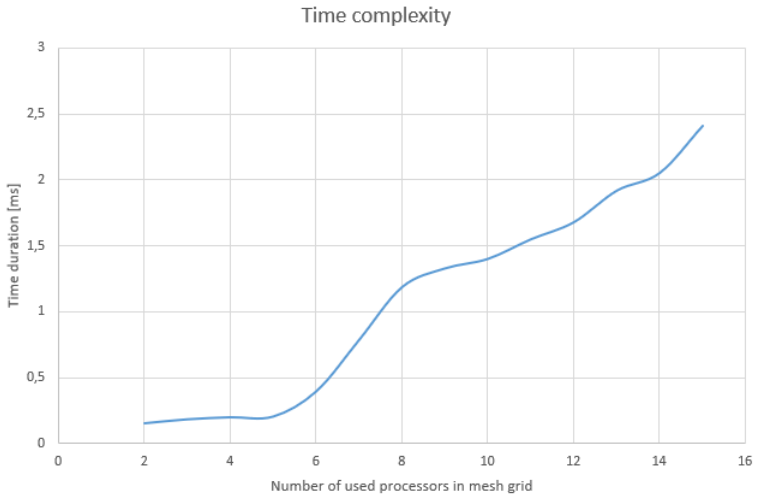

Figure 3: Graph mapping algorithm time complexity for different number of processors

5 Conclusion

This work explains and successfully implements Mesh Multiplication matrix multiplication algorithm. Starts by describing algorithm steps, his complexity analysis and implementation based on openMPI library. Experiments conducted on small number of processors offer weak proof of algorithm linear complexity.