Compulsory Assignment no.3

Dense matrix multiplication

Parallel and Distributed Programming
Division of Scientific Computing, Department of Information Technology
Uppsala University

Spring 2019

1 Problem setting

Let two dense matrices be given, $A = \{a_{i,j}\}$ and $B = \{a_{i,j}\}$, $A, B \in \mathbf{R^{n\times n}}$, i.e., $i, j = 1, 2, \cdots, n$. Let $C = AB$. As is well known, $c_{i,j} = \sum_{k=1}^{n} a_{i,k}b_{k,j}, \quad i, j = 1, 2, \cdots, n$.
The task is to:

1. design a matrix-matrix multiplication procedure for performing the multiplication in a distributed memory parallel computer environment
2. implement the algorithm using C and MPI, and
3. evaluate performance

You are encouraged to work in pairs. However, all topics in the assignment should be covered by each student.

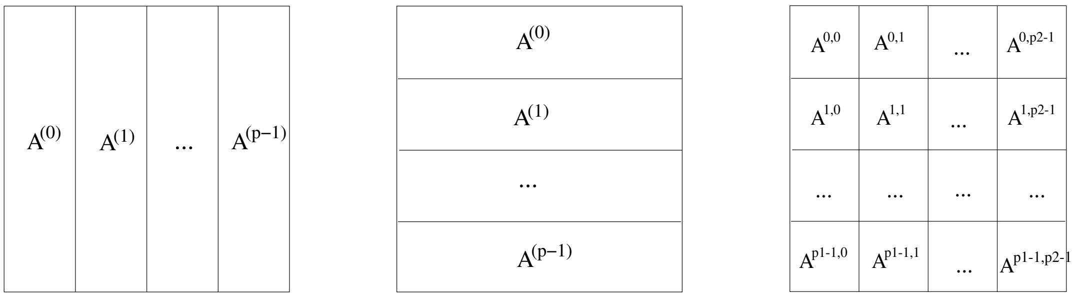
2 Partitioning strategies

Since your program is targeting a distributed memory architecture, you need to partition the matrices A, B and C. Choose the partitioning strategy that you find most suitable. It is up to you whether all matrices are partitioned the same way or not. Some possible partitioning strategies are shown in Figure 1 and presented below.

2.1 Colum-wise (1D-type) partitioning

The matrices are partitioned in blocks, containing $r = n/p$ columns (Figure 1(a)). (We assume that p is a divisor of n, thus, $n = mp$.) If all matrices are partitioned the same way, the blocks $A^{(s)}, B^{(s)}, C^{(s)}$, each of size $n \times m$, are local for processor P_s, $s = 0, 1, \cdots, p-1$. The data distribution imposes additional structure in each block-column $A_i^{(s)}, B_i^{(s)}, C_i^{(s)}$, $i = 0, 1, \cdots, p-1$, each of size $m \times m$. A block $C_i^{(s)}$ is computed as follows

$$C_i^{(s)} = \sum_{j=0}^{p-1} A_i^{(j)} B_j^{(s)},\ i = 0, 1, \cdots, p-1.$$



(a) Colum-wise partitioning (b) Row-wise partitioning (c) Checkerboard partitioning

Figure 1:

The blocks $B_j^{(s)}$ are local to process P_s. From the blocks $A_i^{(j)}$ only $A_i^{(s)}$ is local, the others have to be sent to P_s.

2.2 Row-wise (1D-type) partitioning

It is very similar to the column-wise partitioning (Figure 1(b)).

2.3 Checkerboard (2D-type) partitioning

This partitioning strategy is depicted in Figure 1(c). The processing elements are assumed to form a 2D Cartesian grid where $p = p1 \times p2$. It is also assumed that n is divisible by both $p1$ and $p2$, namely, $r1 = n/p1$ and $r2 = n/p2$, where the size of the blocks is $r1 \times r2$ and $r2 \times r1$ respectively. You are free to use existing algorithms, provided that you give the reference to the source you have used.

3 Implementation

Your program is supposed to take 2 arguments. The first argument is the name of a file containing the matrices A and B (the input file). The second argument is the name of the file to which the program will write the result, that is matrix C (the output file).
The input file will contain $2n^2 + 1$ numbers separated by white spaces. The first number is n (the matrix size). This is supposed to be an integer. The subsequent n^2 numbers are the elements of A, stored in row-major order. The last n^2 elements are the elements of B, also stored in row-major order. The output file shall contain the elements of C stored in ASCII format, in row-major order, and nothing else. Just as in the input file, the elements shall be separated by white spaces. All matrix elements are floating-point numbers and should be stored with six (6) decimal places.
Your implementation must work for matrices of different sizes, but you may make the assumption that n is divisible by the number of processes p if you use row-wise or column-wise partitioning. Likewise, you may assume that $\sqrt{p}$ is an integer, by which n is divisible, if you use checkerboard partitioning.

Measure the time for multiplication, including computations and communications, but not I/O, in seconds. The number of seconds (and nothing else, not even 'seconds' or an abbreviation thereof) shall be written to `stdout` when the multiplication is done.
Note that your code is supposed to be reasonably well structured and documented.

4 Numerical experiments

Your numerical experiments shall be run on UPPMAX. Use the batch system for every execution of your program! You are free to use as many processors as you decide. Pay attention whether you are using the resource you ask for up to its full capacity.
Perform experiments to demonstrate the strong and weak scalability of your implementation.
Estimate (measure) the memory usage for your algorithm.
If you don't want do write your own input files, you may use the files stored in the directory `/proj/g2019005/nobackup/matmul_indata` on UPPMAX. Larger files are intended for performance experiments, while while smaller files can be used for functionality testing.
Files named `inputN.txt` contain input matrices (A and B) of size $N \times N$ on the format described in Section 3. Files named `outputN_ref.txt` contain the expected output when using `inputN.txt` as input and are intended to be used as a reference solution on development testing.
Note that you have a limited disc quota at UPPMAX! Therefore, it is recommended that you don't copy the larger input files to your home directory, but read them from the project directory (where they are stored now). Moreover, it is a good idea to remove large output files as soon as possible. You may also skip the printing to files during the numerical experiments. However, note that the code that you hand in must follow the specifications in Section 3.

5 Report

You are supposed to write a report on your results. The report can be written in Swedish or English, and should contain (at least) the following parts:

1. A brief description of the theoretical problem.
2. A description of your implementation. Don't forget to describe and motivate the partitioning strategy. Why has this strategy been chosen?
3. A description of your numerical experiments.
4. The results from the numerical experiments. Execution times should be presented in tables. The same holds for speedup values. Also plot the ideal speedup in the speedup plot. The plots can be drawn using MATLAB, or any other tool that you are familiar with.
5. A discussion of the results. Do they follow your expectations? Why/why not?

6 File to be submitted

The file that you upload in Studentportalen must be a compressed tar file named A3.tar.gz. It shall contain a directory named A3, with the following files inside:

- A PDF file named A3_Report.pdf containing you report.
- Your code, following the specifications listed in Section 3.
- A Makefile that does the following:
    1. Builds a binary named `matmul` from your code when the command `make` is invoked. This must work on the Linux system as well as on UPPMAX!
    2. Removes all binary files and object files (if any) when the command `make clean` is invoked.

Since your code will be tested automatically, it is very important that you follow the instructions regarding the implementation and the structure of the tar file! Failure to do so will result in immediate rejection of the assignment.

7 Deadline

The full report must be submitted no later than May 25. Assignments that do not meet the minimum requirements are returned. A new version must be submitted no later than June 1. Assignments submitted after this date will not be approved.

8 Precaution - reminder

Use the UPPMAX resources with care! We are given 4000 core/hours for the whole course.

Success!

Maya & Malin

Any comments on the assignment will be highly appreciated and will be considered for further improvements. Thank you!