HPC

Description of algorithms:

MATMULT

Because the data of matrix A saved in each processor is a sub-matrix block, I choose the **'Submatrix factorization'** algorithm to compute the multiplication of a matrix and a vector. The main idea of this solution is that making all the processors share the **full** vector and doing multiplication in each local processor by locating the partial vector corresponding to the sub-matrix block saved in that processor.

The processes of the algorithm are:

1) Distribute vector: Read vector x in each processor and gather them to a full vector. Distribute the full vector to each process. (**MPI_Allgather()**)
2) Do local multiplication: Multiply every sub-matrix block and corresponding vector part (by finding the mathematic relation) and sum up the product in every processor.
3) Create a full vector(*ycopy) of size y->N for **each** processor. Put the outcome of step2 into the corresponding part of full vector.
4) Use **MPI_Allgather()** to sum the values of same row and gather values saved in different rows together, this is the full result of A * x.
5) Use **MPI_Scatter()** to scatter y->n length data of A*x to each processor.

SUMMA

The main idea of this algorithm is that each processor collects all the columns from the subblocks of matrix A in the same row processor and all the rows from the subblocks of matrix B in the same row processor, and then multiplies the rows and columns to form a block-matrix of matrix C. Finally, the processor whose rank=0 collects the data from other processors to form the final matrix C.

The processes of the algorithm are:

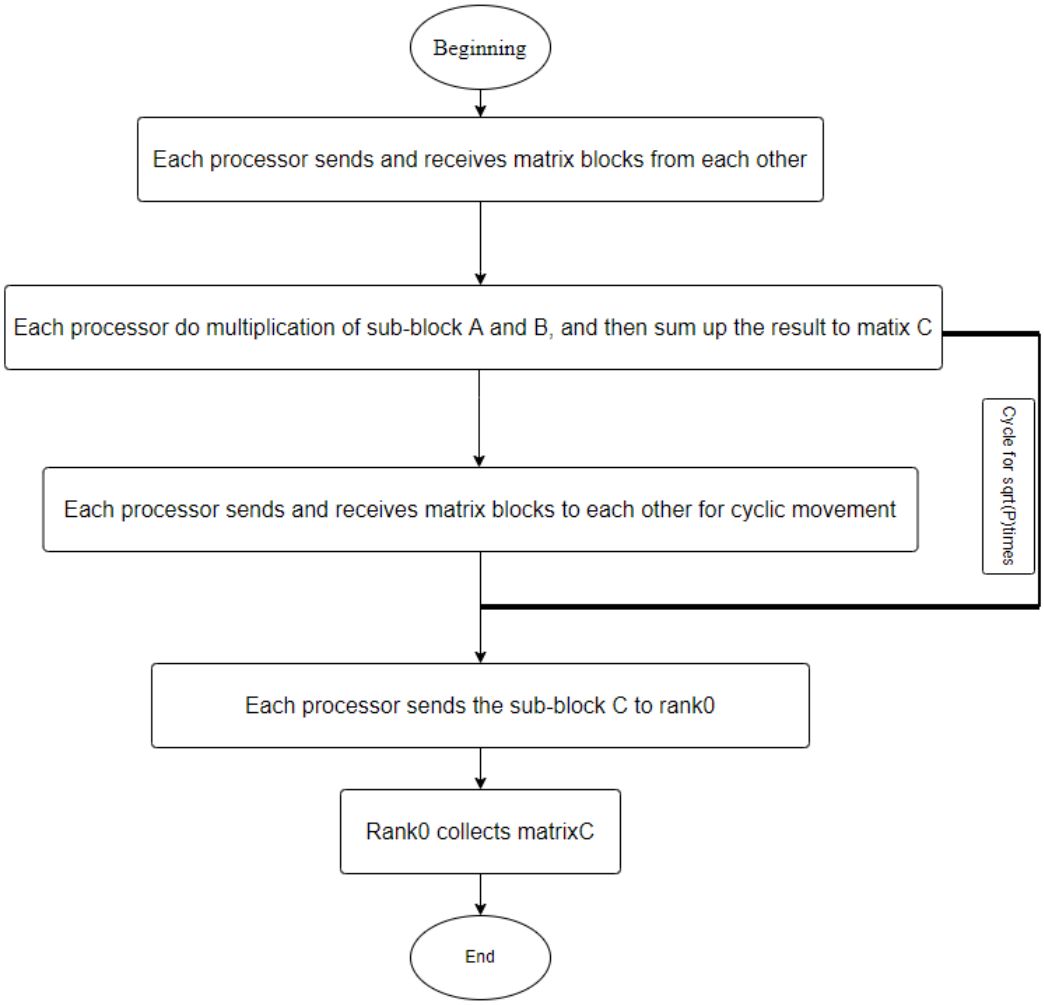
1) Each processor sends its own block columns of the A matrix to other processors in the same row as itself.
2) Each processor sends its own block of rows of the B matrix to other processors in the same column as itself.
3) Each processor receives columns from all blocks of A in the same row processor and rows from all blocks of B in the same column processor. Each processor already has all the information needed to compute the corresponding D subblock, which can be multiplied and accumulated to obtain the corresponding D subblock.
4) Sum up D subblock and C subblock in each processor to get the final result.

CANNON

At the start, processor $P_{i,j}$ saves block $A_{i,j}$ and block $B_{i,j}$, it is also responsible for computing block $C_{i,j}$.

1) Move block $A_{i,j}$ $(0 \le i,j < \sqrt{P})$ i steps to the left; move up block $B_{i,j}$ $(0 \le i,j < \sqrt{P})$ i steps.
2) $P_{i,j}$ do multiplication and adding operation. And then move block $A_{i,j}$ $(0 \le i,j < \sqrt{P})$ 1 steps to the left; move up block $B_{i,j}$ $(0 \le i,j < \sqrt{P})$ 1 steps.
3) Repeat STEP 2). $P_{i,j}$ does the multiplication-adding operation for $\sqrt{P}$ times, it also does the cyclic single step shift for $\sqrt{P}$ times.

Blow is the algorithm flow chart:



Data Analysis

Strong scaling

As I increase the number of processors and keep the whole size of matrix constant. I increase the number of processors up to 256 as required, so I fix the matrix size at N=9216. The numbers of processors I choose is P = [1,4,9,16,,36,64,144,256]. As the number of processors increases, the time consumed gradually decreases. So to reduce the running time, we can add more processors. However, parallel algorithm may not exhibit perfect speedup. That is, we may not get a performance

increase of P when we employ P processors. Because the maximum speedup is actually limited by the serial fraction of the code.

Weak scaling

In this test, I do not keep the size of matrix constant. I increases the total number of processors and the size of matrix simultaneously but I we fix the problem size per process, which means the ratio N/P is constant as we change P. For example, I use P = [1,4,9,16, ,36,64,144,256] and N = [10, 40, 160, 360, 640, 1440, 2560]. I find although the size of matrix also grows up, the speedup actually improves. A perfect weak scaling matches the number of processes to the problem size, giving constant runtime for increasing problem sizes by adding more processes.