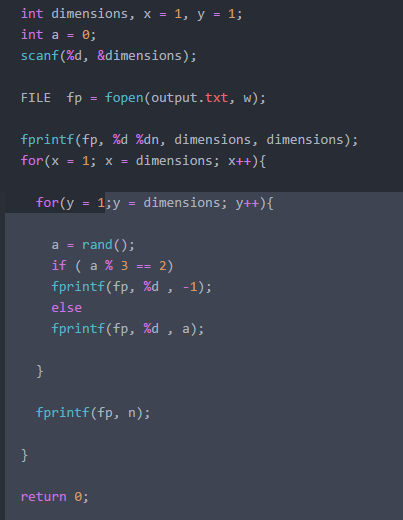


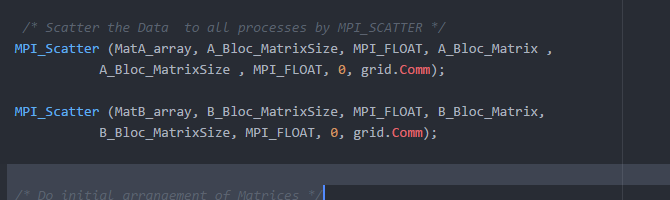
Solution:

1. A matrix with NxN dimensions with N= 210) was generated in c programming language. The code uses a random generation function inbuilt with c to generate a value. This value is normalized through dividing it by 3. The reminder is then asserted to obtain the value in the range [-1,1]. The values are written to the output file.

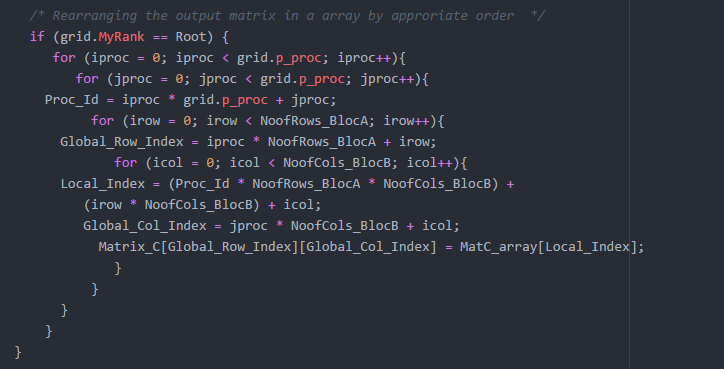
The code snippet of the matrix generator function is pasted below,



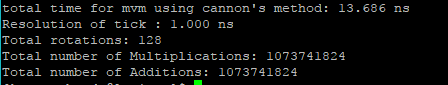
1. Cannon’s method was used to implement matrix multiplication. The multiplication in cannon’s method involves rearranging the matrices initially to get the proper fir-term partial sum. This rearrangement involves the rotation of the matrix about its row (or column) depending on their indices. After the rearrangement the matrices are multiplied and rotated by one index. This requires each processor to make the calculation of each element in the output matrix. Since in the problem statement the output is a matrix of size (1024\*1024) and we are limited to a maximum total of 256 processors, the cannon’s multiplication is performed by block matrix method. Initially before rearranging the matrix, both the matrices are divided into a Cartesian mesh in which a block of matrix will be processed by each processor. These blocks of data are first scattered to all the processors.



After this step each block is rotated depending on their row and column index to obtain the first partial sum. To perform the scatter and block matrix multiplication the 2-dimensional matrix is converted into a one-dimensional array which is later reconstructed back to a 2D array after the completion of the multiplication. The 1D array is referred by its local index where as the 2D array is referred by its global address.



The total number of addition, multiplications and rotations are noted in each processor. Later these values along with the time taken for the execution is sent to the root node which calculates the cumulative result. The method incurred the following results for a processor count of 16 when ran on seawulf.



The program was run many time to get the best results. The program evaluates and outputs the total time taken by the all the processors (in ns) as well as the number of arithmetic operations performed.

The following table shows the number of rotations made and the total time taken for each processor count.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Total Time of execution(ns) and rotations per processor | | Processor count, P | | |
| 16 | 64 | 256 |
| Cannon's algorithm | Time taken | 9.342 | 7.625 | 6.328 |
| Rotations | 10 | 18 | 42 |

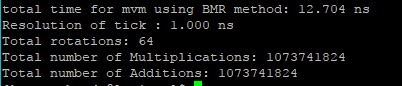
Inferences:

1. We can see from the results above that the total time taken for the multiplication of matrices decreases as the processor count increase. This is due to the fact that, as the processor count increase the number of arithmetic operations performed by each processor decrease hence increasing the efficiency of the program.
2. Another observation to make from the above table is that the number of rotations (per processor) increase with the processor count. When the processor budget is high the block matrix received by each processor will be less. Because of this the maximum row and column indices will be significantly less. Hence the rotations become limited as they are dependent on the row and column index. But since the number of blocks are significantly increased the rotations more rotations occur to get the first local partial product.
3. The additions and multiplications also follow the trend by rotations. As the processor count increase the number of arithmetic operations performed by each processor deceases.

Although the number of operations per processor decreases with increasing processor count, it should be noted that the total number of rotations and arithmetic operations remain the same.

1. The next method of implementation is Broadcast-multiply and roll (BMR). This method is also referred to as fox’s method. In BMR the diagonal elements are first broadcasted to all the processors in the respective rows. This is multiplied with the elements of the second matrix. In the second step the elements of the first matrix are rotated by a step and the new matrix is broadcasted to the group. In this step the second matrix is also rotated by one step. By doing this we obtain the corresponding partial sum. Since the resulting matrix is to large in the given case the matrix can be subdivided in to blocks of matrices. To convert the huge matrix to smaller blocks the method used is similar to that of the previous method. We use the inbuilt features in MPI that allow us to setup a Cartesian mesh. In the program this can be found under the function setup\_mesh(). And later the block of data is scattered using the MPI scatter command.

The fox’s method was implemented on c programming language and was implemented on the seawulf cluster. The following are the results when the processor budget was 16.



The program was run many time to get the best results. The results from running the algorithm in multiple configuration and processor budget was recorded and is presented in the table below.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Time of excecution in ns | | Processor count, P | | |
| 16 | 64 | 256 |
| BMR method | Time taken | 9.181 | 7.676 | 6.163 |
| Rotations | 4 | 8 | 16 |

Inferences:

1. From above results it can be noted that the total number of rotations for BMR method is less that that of cannons method. This is the inherent feature that BMR method provides to reduce the over all communication overhead.
2. Also, we can observe that the total number of arithmetic operations performed by both methods is same. This is due to the fact that for a given size of matrix the total number of multiplications and additions always remain the same. It is also a result of making equal blocks of data.
3. The total number of arithmetic operations in both cases were observed to be same. Since in both method the blocks are made up of equal size it can be safely assumed that arithmetic operation count will remain the same. It can be observed from the results also. In my code all the results processes by each processor are sent to the root node, which cumulates the total time taken as well as the total number of arithmetic operations. The following were the results in both the cases. Since matrix multiplication works like a multiply and accumulate unit for every increase in addition the multiplication also increases. Hence the total number of additions and multiplications are same.

|  |  |
| --- | --- |
| Additions | 1,073,741,824 |
| Multiplications | 1,073,741,824 |
| Total arithmetic operations | 2,147,483,648 |

The time take to complete the operation is mentioned in the tables on page 3 and 4 along with the rotation per processor. The BMR method seems to show slightly better output that cannon’s method. BMR method performing less number of rotations suffers from less communication overhead. This also helps BMR method to manage the buffered memory efficiently.

References :

<https://www3.nd.edu/~zxu2/acms60212-40212-S12/Lec-07-3.pdf>

http://www.cse.iitd.ernet.in/~dheerajb/MPI/Document/hos\_cont.html

<https://github.com/cstroe/PP-MM-A03>

<http://orca.st.usm.edu/~seyfarth/sc730/cannon.html>

<https://www.eecis.udel.edu/~saunders/courses/879-03s/fox.c>