Master 1 Informatique and Master 1 MOSIG

Parallel Algorithms and Programming

Lab #5

April-2019

Done by
Salman Farhat
Nesrin Mansouri

How to Run

To run our code you should use:

1. mpicc -o try PROGX.c -lm
2. mpirun -n N try

N should have an integer square root (9 --> 3, 4-->2)
After you need to enter the matrix Dimension it will be NxN where N should be divisible by the dimension of the process grid n (n X n).

Fox Algorithm

For Part 1 in the project

To write the fox algorithm using MPI we did:

- Create a grid of processes (dim X dim)
- Create communicator on columns and rows
- Create a new datatype MatrixBlockType of (blockSize * blockSize) where blockSize is the dimension of the submatrix of the Matrix
- In each process, we initialize the submatrices
- Then we apply these steps:

```
MPI_Cart_shift(commcol, 0,1, &dest , &source);
```

Here in each process, we calculate from where to receive and where to send, receive from below send up.

```
for(int stage=0 ; stage < dim ; stage++){
```

Loop on number of columns in process grid which is dim (columns = rows dim*dim)

```
root=(coord_2d[0] + stage ) % dim;
```

We calculate the root for process 0 it will be first 0 then 1 etc ...

```
if(root==coord_2d[1]) {seconds
  MPI_Bcast(A , 1 , MatrixBlockType , root , commrow);
  multiply(A,B,C,blockSize);
}else {
  MPI_Bcast(tempa , 1 , MatrixBlockType , root , commrow);
  multiply(tempa,B,C, blockSize);
```

```
}
```
Here if the root is equal to the the column number it will broadcast it's block A and all others that are on the same row they will receive the A in TempA and proceed multiplication.
```
for(int l =0  ; l < blockSize ; l++){
  for(int o = 0 ; o <  blockSize ; o++){
    buff[ l*blockSize  + o]= B[ l*blockSize  + o] ;
  }
}
```
Here we are coping block B to a buffer to send using Isend
```
MPI_Isend(buff, 1, MatrixBlockType, dest, 1, commcol, &request2);
MPI_Irecv(buff2, 1, MatrixBlockType, source, 1, commcol, &request);
MPI_Wait(&request, &status);
MPI_Wait(&request2, &status);
```
After the process send it's block B to destination we need to receive new block from the source process which is right below and then we wait to have full completion for the send and receive operations.

```
for(int l =0  ; l < blockSize ; l++){
  for(int o = 0 ; o <  blockSize ; o++){
    B[l*blockSize  + o] = buff2[l*blockSize  + o];
  }
}
```
After receiving we are putting the received one to B
```
}
```
After this stage, all process will have the correct values of the C Matrix partition

1. Describe the optimizations that you included in your code if any.

We used non-blocking send and receive (MPI_Isend, MPI_Irecv).
We declared our data types

Cannon Algorithm

For Part 2 in the project

To write the fox algorithm using MPI we did:

- Create a grid of processes (dim X dim)
- Create a new datatype MatrixBlockType of (blockSize * blockSize) where blockSize is the dimension of the submatrix of the Matrix
- In each process, we initialize the submatrices
- Then we apply these steps:

```
int LS , RS , US , DS;

MPI_Cart_shift(cartcomm, 1,  coord_2d[0], &LS , &RS);
MPI_Cart_shift(cartcomm, 0, coord_2d[1], &US , &DS);
```
process at row 0 will not shift left-right
Process on the row 1 will shift by 1
Process on the row 2 will shift by 2
So we are using the row number `coord_2d[0]` to decide how much we need to shift in each process
Same for Up and down w.r.t columns

We copy A to buffA0 and B to BuffB0
After we send and receive in BuffB0 and BuffA0
```
MPI_Isend(buffA0, 1, MatrixBlockType, LS, 1, cartcomm, & requestA0);
MPI_Irecv(buffA1, 1, MatrixBlockType, RS, 1, cartcomm, & requestA1);

MPI_Isend(buffB0, 1, MatrixBlockType, US, 1, cartcomm, & requestB0);
MPI_Irecv(buffB1, 1, MatrixBlockType, DS, 1, cartcomm, & requestB1);

MPI_Wait( & requestA0, & status);
MPI_Wait( & requestA1, & status);
MPI_Wait( & requestB0, & status);
MPI_Wait( & requestB1, & status);
```
Then we put the received block in BuffB1 in B and BuffA1 in A.

Then process transfer blocks A on left right and B on up and down
Then each process proceed multiplication
```
multiply(A , B , C , blockSize);
```

Now we need to loop from 1 to dim which is (dim X dim) dimension for the process matrix, and do shift by 1 and transfer blocks Between process and proceed multiplication in each process.

```
for (i = 1; i < dim; i++) {
MPI_Cart_shift(cartcomm, 1, 1, & LS, & RS);
MPI_Cart_shift(cartcomm, 0, 1, & US, & DS);
for (int I = 0; I < blockSize; I++) {
  for (int o = 0; o < blockSize; o++) {
    buffA0[I * blockSize + o] = A[I * blockSize + o];
  }
}
for (int I = 0; I < blockSize; I++) {
  for (int o = 0; o < blockSize; o++) {
    buffB0[I * blockSize + o] = B[I * blockSize + o];
  }
}
MPI_Isend(buffA0, 1, MatrixBlockType, LS, 1, cartcomm, & requestA0);
MPI_Irecv(buffA1, 1, MatrixBlockType, RS, 1, cartcomm, & requestA1);
MPI_Isend(buffB0, 1, MatrixBlockType, US, 1, cartcomm, & requestB0);
MPI_Irecv(buffB1, 1, MatrixBlockType, DS, 1, cartcomm, & requestB1);
```

```
MPI_Wait( & requestA0, & status);
MPI_Wait( & requestA1, & status);
MPI_Wait( & requestB0, & status);
MPI_Wait( & requestB1, & status);

for (int l = 0; l < blockSize; l++) {
  for (int o = 0; o < blockSize; o++) {
    A[l * blockSize + o] = buffA1[l * blockSize + o];
  }
}
for (int l = 0; l < blockSize; l++) {
  for (int o = 0; o < blockSize; o++) {
    B[l * blockSize + o] = buffB1[l * blockSize + o];
  }
}

multiply(A, B, C, blockSize);

}
```

After this stage, all process will have the correct values of the C Matrix partition.

1. Describe the optimizations that you included in your code if any.

We used non-blocking send and receive (MPI_Isend, MPI_Irecv)
We declared our data types

Data management

For the data management part, we create big matrices in all process and initialize them in process rank 0

```
int *Main_A = NULL,*Main_B = NULL,*Main_C = NULL;
if(rank == 0){
  Main_A = (int *) malloc(matrixSize * matrixSize * sizeof(int ));
  Main_B = (int *) malloc(matrixSize * matrixSize * sizeof(int ));
  Main_C = (int *) malloc(matrixSize * matrixSize * sizeof(int ));
  fillData(Main_A , Main_B , Main_C , matrixSize);
}
```

After w initialize sendcount array that indicates how much elements from matrix each process should take, and displs indicates from where each process should starts.

```
int c = 0;
for(int i = 0 ; i < dim; i++){
  for(int j  =0 ; j  <dim; j++){
```

```
    sendcounts[c] = (matrixSize*matrixSize) / numtasks;
    displs[c] = ((i * blockSize)*matrixSize) + j*blockSize;
    c++;
  }
}
```

After we need to rearrange the matrix to make the scatter, so this function arranges the array in a way each process take the correct chunk of the matrix.

```
void PrepareForScater(int * Main_A , int *arrangedBuf ,int startFrom ,int strid, int blockSize , int
matrixSize , int numtasks , int *displs){
 int counter = 0;
 for(int i  =0  ; i < numtasks ; i++ ){
   startFrom = displs[i];
   //printf("start from is %d  " , startFrom);
  for(int j =0  ; j < blockSize ; j++ ){
    for(int k = startFrom ; k < startFrom + blockSize ; k++){
      arrangedBuf[counter] = Main_A[k];
      counter++;
    }
    startFrom = startFrom  + strid;
  }
 }
}
```

```
PrepareForScater(Main_A  ,arrangedBufA , startFrom , strid , blockSize ,  matrixSize , numtasks ,
displs);
PrepareForScater(Main_B  ,arrangedBufB , startFrom , strid , blockSize ,  matrixSize , numtasks ,
displs);
```

arrangedBufA is the matrix that is arranged in a right way

Then we call the scatter

```
MPI_Scatter(arrangedBufA ,blockSize *  blockSize, MPI_INT , A , 1 , MatrixBlockType , 0,
cartcomm);
MPI_Scatter(arrangedBufB , blockSize *  blockSize, MPI_INT , B , 1 , MatrixBlockType , 0,
cartcomm);
```

After we need to gather the C chunks from all process to the main process (rank 0) we do Gather

```
MPI_Gather(C, 1  , MatrixBlockType , Main_C  , 1  , MatrixBlockType , 0 , cartcomm);
```

Exercise 4: Performance evaluation

Fox :

Experiment 1:

Hardware description: on 4 cores
2 physical cores each having hyper thread (2 threads) so in total we have 4 logical cores.
Number of processes = 4
Matrix size = 4
Result:
Total time it took to do the multiplication is :0.001016
Number of cycles: 2017206 cycles

Experiment 2:
Hardware description: on 4 cores
2 physical cores each having hyper thread (2 threads) so in total we have 4 logical cores.
Number of processes = 4
Matrix size = 1000
Result:
Total time it took to do the multiplication is :5.022948
Number of cycles: 3938555105 cycles

Experiment 3:

Hardware description: on 64 cores
32 physical cores each having hyper thread (2 threads) so in total we have 64 logical cores
Number of processes = 16
Matrix size = 5000

```
akhorguani@dahu-5:~/parallel_mpi/nisreen$ perf stat -d mpirun -n 16 ./fox

number of tasks is : 16 , dim 4 4
Matrix is NxN it should be N mod dim = 0   enter N:
5000

matrix columns size is : 5000 , process columns size is 4

 Fox algo                       3653207778 cycles


Totall time it took to do the multiplication is :77.594628

 Performance counter stats for 'mpirun -n 16 ./fox':

    1293893.238896      task-clock (msec)         #   15.654 CPUs utilized
            85,783      context-switches          #    0.066 K/sec
               293      cpu-migrations            #    0.000 K/sec
           284,988      page-faults               #    0.220 K/sec
 3,618,176,093,330      cycles                    #    2.796 GHz                      (50.01%)
 5,834,022,225,383      instructions              #    1.61  insn per cycle           (62.51%)
   142,411,469,347      branches                  #  110.064 M/sec                    (62.51%)
       120,448,197      branch-misses             #    0.08% of all branches          (62.50%)
 2,781,408,931,276      L1-dcache-loads           # 2149.643 M/sec                    (62.50%)
   133,847,613,209      L1-dcache-load-misses     #    4.81% of all L1-dcache hits    (62.51%)
     7,913,029,600      LLC-loads                 #    6.116 M/sec                    (50.01%)
     4,215,503,958      LLC-load-misses           #   53.27% of all LL-cache hits     (50.01%)

      82.653197252 seconds time elapsed
```

Experiment 4:

Hardware description: on 64 cores

32 physical cores each having hyper thread (2 threads) so in total we have 64 logical cores.

Number of processes = 25

Matrix size = 5000

```
number of tasks is : 25 , dim 5 5
Matrix is NxN it should be N mod dim = 0   enter N:
5000

matrix columns size is : 5000 , process columns size is 5

 Fox algo                       3304559258 cycles


Totall time it took to do the multiplication is :52.827924

 Performance counter stats for 'mpirun -n 25 ./fox':

    1506839.895981      task-clock (msec)         #   23.930 CPUs utilized
           310,733      context-switches          #    0.206 K/sec
             2,967      cpu-migrations            #    0.002 K/sec
           321,054      page-faults               #    0.213 K/sec
 4,200,176,213,617      cycles                    #    2.787 GHz                      (50.03%)
 6,001,661,329,988      instructions              #    1.43  insn per cycle           (62.52%)
   177,823,221,413      branches                  #  118.011 M/sec                    (62.50%)
       164,480,999      branch-misses             #    0.09% of all branches          (62.50%)
 2,839,830,165,420      L1-dcache-loads           # 1884.626 M/sec                    (62.51%)
   133,499,542,668      L1-dcache-load-misses     #    4.70% of all L1-dcache hits    (62.50%)
     7,876,326,830      LLC-loads                 #    5.227 M/sec                    (50.00%)
     2,868,954,430      LLC-load-misses           #   36.43% of all LL-cache hits     (50.02%)

      62.969235645 seconds time elapsed

akhorguani@dahu-5:~/parallel_mpi/nisreen$
```

Full Fox :

Experiment 1:

Hardware description: on 4 cores

2 physical cores each having hyper thread (2 threads) so in total we have 4 logical cores

Number of processes = 4

Matrix size = 4

Result:

Number of cycles: 9386317 cycles

Total time it took to do the multiplication is: 0.003781

Experiment 2:

Hardware description: on 4 cores

2 physical cores each having hyper thread (2 threads) so in total we have 4 logical cores

Number of processes = 4

Matrix size = 1000

Result:

Number of cycles: 487236063 cycles

Total time it took to do the multiplication is: 7.083110

Experiment 3:

Hardware description: on 64 cores

32 physical cores each having hyper thread (2 threads) so in total we have 64 logical cores

Number of processes = 16

Matrix size = 5000

```
akhorguani@dahu-5:~/parallel_mpi/nisreen$ perf stat -d mpirun -n 16 ./full_fox

number of tasks is : 16 , dim 4 4
Matrix is NxN it should be N mod dim = 0   enter N:
5000

matrix columns size is : 5000 , process columns size is 4

 Fox algo                        2532994040 cycles


Totall time it took to do the multiplication is :81.159986

 Performance counter stats for 'mpirun -n 16 ./full_fox':

    1327089.895331      task-clock (msec)         #   15.751 CPUs utilized
           40,522       context-switches          #    0.031 K/sec
            1,125       cpu-migrations            #    0.001 K/sec
          358,247       page-faults               #    0.270 K/sec
 3,709,375,276,520      cycles                    #    2.795 GHz                      (50.00%)
 5,849,484,042,055      instructions              #    1.58  insn per cycle           (62.50%)
   145,245,580,522      branches                  #  109.447 M/sec                    (62.50%)
       120,251,759      branch-misses             #    0.08% of all branches          (62.50%)
 2,786,531,717,403      L1-dcache-loads           # 2099.731 M/sec                    (62.50%)
   134,720,958,085      L1-dcache-load-misses     #    4.83% of all L1-dcache hits    (62.50%)
     7,900,834,970      LLC-loads                 #    5.954 M/sec                    (50.00%)
     4,093,366,543      LLC-load-misses           #   51.81% of all LL-cache hits     (50.00%)

      84.253385774 seconds time elapsed
```

Experiment 4:

Hardware description: on 64 cores

32 physical cores each having hyper thread (2 threads) so in total, we have 64 logical cores

Number of processes = 25

Matrix size = 5000

```
akhorguani@dahu-5:~/parallel_mpi/nisreen$ perf stat -d mpirun -n 25 ./full_fox

number of tasks is : 25 , dim 5 5
Matrix is NxN it should be N mod dim = 0   enter N:
5000

matrix columns size is : 5000 , process columns size is 5

 Fox algo                         695244998 cycles


Totall time it took to do the multiplication is :63.882631

 Performance counter stats for 'mpirun -n 25 ./full_fox':

    1640441.540649      task-clock (msec)         #   24.512 CPUs utilized
           68,787       context-switches          #    0.042 K/sec
            5,000       cpu-migrations            #    0.003 K/sec
          393,988       page-faults               #    0.240 K/sec
 4,575,747,210,917      cycles                    #    2.789 GHz                      (50.00%)
 6,038,709,384,106      instructions              #    1.32  insn per cycle           (62.51%)
   185,395,120,102      branches                  #  113.015 M/sec                    (62.50%)
       161,173,010      branch-misses             #    0.09% of all branches          (62.50%)
 2,853,256,505,421      L1-dcache-loads           # 1739.322 M/sec                    (62.50%)
   133,627,639,621      L1-dcache-load-misses     #    4.68% of all L1-dcache hits    (62.50%)
     7,886,059,571      LLC-loads                 #    4.807 M/sec                    (50.00%)
     2,865,854,006      LLC-load-misses           #   36.34% of all LL-cache hits     (50.01%)

      66.925155705 seconds time elapsed
```

Canon :

Experiment 1:

Hardware description: on 4 cores
2 physical cores each having hyper thread (2 threads) so in total we have 4 logical cores.
Number of processes = 4
Matrix size = 4
Number of cycles: 1168420 cycles
Total time it took to do the multiplication is :0.000697

Experiment 2:

Hardware description: on 4 cores
2 physical cores each having hyper thread (2 threads) so in total we have 4 logical cores
Number of processes = 4
Matrix size = 1000
Number of cycles: 33340488 cycles
Total time it took to do the multiplication is :3.457254
Experiment 3:

Hardware description: on 64 cores
32 physical cores each having hyper thread (2 threads) so in total we have 64 logical cores
Number of processes = 16
Matrix size = 5000

```
akhorguani@dahu-9:~/parallel_mpi/nisreen$ perf stat -d mpirun -n 16 ./cannon

number of tasks is : 16 , dim 4 4
Matrix is NxN it should be N mod dim = 0   enter N:
5000

matrix columns size is : 5000 , process columns size is 4

 Fox algo                       2104725784 cycles


Total time it took to do the multiplication is :87.105842

 Performance counter stats for 'mpirun -n 16 ./cannon':

    1390011.850311      task-clock (msec)         #   15.274 CPUs utilized
           320,244      context-switches          #    0.230 K/sec
            51,786      cpu-migrations            #    0.037 K/sec
           236,147      page-faults               #    0.170 K/sec
 3,883,897,378,215      cycles                    #    2.794 GHz                      (50.01%)
 5,823,717,307,759      instructions              #    1.50  insn per cycle           (62.51%)
   139,624,498,052      branches                  #  100.448 M/sec                    (62.50%)
       133,388,661      branch-misses             #    0.10% of all branches          (62.50%)
 2,778,760,347,342      L1-dcache-loads           # 1999.091 M/sec                    (62.49%)
   137,019,291,436      L1-dcache-load-misses     #    4.93% of all L1-dcache hits    (62.50%)
     7,906,514,956      LLC-loads                 #    5.688 M/sec                    (50.00%)
     4,760,496,780      LLC-load-misses           #   60.21% of all LL-cache hits     (50.01%)

      91.006071295 seconds time elapsed
```

Experiment 4:

Hardware description: on 64 cores

32 physical cores each having hyper thread (2 threads) so in total, we have 64 logical cores.

Number of processes = 25

Matrix size = 5000

```
akhorguani@dahu-9:~/parallel_mpi/nisreen$ perf stat -d mpirun -n 25 ./cannon

number of tasks is : 25 , dim 5 5
Matrix is NxN it should be N mod dim = 0   enter N:
5000

matrix columns size is : 5000 , process columns size is 5

 Fox algo                       589297308 cycles


Total time it took to do the multiplication is :74.082357

 Performance counter stats for 'mpirun -n 25 ./cannon':

    1779244.205761      task-clock (msec)         #   22.897 CPUs utilized
         1,462,392      context-switches          #    0.822 K/sec
            41,394      cpu-migrations            #    0.023 K/sec
           271,874      page-faults               #    0.153 K/sec
 4,964,270,087,826      cycles                    #    2.790 GHz                      (49.98%)
 5,902,665,109,669      instructions              #    1.19  insn per cycle           (62.46%)
   155,434,977,098      branches                  #   87.360 M/sec                    (62.48%)
       237,096,629      branch-misses             #    0.15% of all branches          (62.50%)
 2,803,577,917,407      L1-dcache-loads           # 1575.713 M/sec                    (62.52%)
   134,640,407,486      L1-dcache-load-misses     #    4.80% of all L1-dcache hits    (62.54%)
     7,977,253,617      LLC-loads                 #    4.484 M/sec                    (50.02%)
     4,889,006,402      LLC-load-misses           #   61.29% of all LL-cache hits     (49.99%)

      77.707492466 seconds time elapsed
```

Full Canon :

Experiment 1:

Hardware description: on 4 cores
2 physical cores each having hyper thread (2 threads) so in total we have 4 logical cores.
Number of processes = 4
Matrix size = 4
Number of cycles: 1227127 cycles
Total time it took to do the multiplication is :0.000626

Experiment 2:

Hardware description: on 4 cores
2 physical cores each having hyper thread (2 threads) so in total we have 4 logical cores.
Number of processes = 4
Matrix size = 1000
Number of cycles: 401592955 cycles
Total time it took to do the multiplication is :3.609431

Experiment 3:

Hardware description: on 64 cores
32 physical cores each having hyper thread (2 threads) so in total we have 64 logical cores
Number of processes = 16
Matrix size = 5000

```
akhorguani@dahu-9:~/parallel_mpi/nisreen$ perf stat -d mpirun -n 16 ./full_cannon

number of tasks is : 16 , dim 4 4
Matrix is NxN it should be N mod dim = 0   enter N:
5000

matrix columns size is : 5000 , process columns size is 4

 Full Canon algo                        3480526122 cycles


Total time it took to do the multiplication is :81.718370

 Performance counter stats for 'mpirun -n 16 ./full_cannon':

    1339869.855618      task-clock (msec)         #   15.728 CPUs utilized
            49,314      context-switches          #    0.037 K/sec
               634      cpu-migrations            #    0.000 K/sec
           382,445      page-faults               #    0.285 K/sec
 3,745,235,587,930      cycles                    #    2.795 GHz                      (50.01%)
 5,861,465,198,671      instructions              #    1.57  insn per cycle           (62.51%)
   147,010,607,738      branches                  #  109.720 M/sec                    (62.50%)
       120,188,855      branch-misses             #    0.08% of all branches          (62.50%)
 2,791,530,362,231      L1-dcache-loads           # 2083.434 M/sec                    (62.50%)
   134,926,125,030      L1-dcache-load-misses     #    4.83% of all L1-dcache hits    (62.50%)
     7,900,662,696      LLC-loads                 #    5.897 M/sec                    (50.00%)
     4,009,410,468      LLC-load-misses           #   50.75% of all LL-cache hits     (50.01%)

      85.189507000 seconds time elapsed
```

Experiment 4:

Hardware description: on 64 cores

32 physical cores each having hyper thread (2 threads) so in total, we have 64 logical cores

Number of processes = 25

Matrix size = 5000

```
akhorguani@dahu-9:~/parallel_mpi/nisreen$ perf stat -d mpirun -n 25 ./full_cannon

number of tasks is : 25 , dim 5 5
Matrix is NxN it should be N mod dim = 0   enter N:
5000

matrix columns size is : 5000 , process columns size is 5

 Full Canon algo                        3509716226 cycles


Total time it took to do the multiplication is :69.433487

 Performance counter stats for 'mpirun -n 25 ./full_cannon':

    1755242.768422      task-clock (msec)         #   24.133 CPUs utilized
           249,803      context-switches          #    0.142 K/sec
            12,141      cpu-migrations            #    0.007 K/sec
           418,583      page-faults               #    0.238 K/sec
 4,905,958,779,867      cycles                    #    2.795 GHz                      (49.99%)
 6,075,268,625,701      instructions              #    1.24  insn per cycle           (62.49%)
   192,214,327,203      branches                  #  109.509 M/sec                    (62.50%)
       171,560,790      branch-misses             #    0.09% of all branches          (62.51%)
 2,867,238,403,128      L1-dcache-loads           # 1633.528 M/sec                    (62.52%)
   133,958,905,824      L1-dcache-load-misses     #    4.67% of all L1-dcache hits    (62.51%)
     7,886,264,776      LLC-loads                 #    4.493 M/sec                    (50.00%)
     2,524,868,009      LLC-load-misses           #   32.02% of all LL-cache hits     (50.00%)

      72.733331309 seconds time elapsed
```

References

“Efficient Parallel Implementation of the Fox Algorithm.” *CiteSeerX*, citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.84.7509.

An Introduction to Parallel Computing, parallelcomp.uw.hu/ch06lev1sec7.html.

Special Thanks to Ana Khorguani. She helped me in testing phase where she run my algorithm on the 64 cores she has access to.