Programming 3 SUMMA Matrix Multiplication

1. Environment Introduction

In this programming assignment, I will use GCC with the version of 8.2.0 and MPICH with the version of 3.1.4 as the compile environment in order to run the MPI program(shown in Figure 1.1). For the running environment, I have requested 64 nodes in Palmetto, and each node only has one CPU(shown in Figure 1.2). In this assignment, I will use the command of "***mpicc -o summa***(the name of my execution file) ***summa.c***(the name of my program)" to compile the SUMMA program I wrote and use the command of "***mpiexec -n ${number of processors} ./summa ${matrix size***}" to run my program. Specifically, when the number of CPU is less than 9, I will use qsub file to submit the job to palmetto. Otherwise, I will run the node I request directly. But, no matter which way I submit the job, I will make sure the running environment same as the environment I introduced above.

```
[wufangm@login001 ~]$ module load gcc/8.2.0
[wufangm@login001 ~]$ module load mpich/3.1.4
[wufangm@login001 ~]$ cd progs_3_MPI/
```

Figure 1.1 Compile environment

```
[wufangm@login001 progs_3_MPI]$ qsub -I -l select=64:ncpus=1:chip_type=e5-2665:mem=24gb
qsub (Warning): Interactive jobs will be treated as not rerunnable
qsub: waiting for job 4505313.pbs02 to start
qsub: job 4505313.pbs02 ready

[wufangm@node1643 ~]$
```

Figure 1.2 Running environment

2. Program Analysis

In this assignment, I need to implement the SUMMA algorithm by using Message Passing Interface(MPI). For the specific step, I need to set up the MPI processes, communicator in the single column and row, processor grid, matrix blocks and collect the computation time. In the step of setting up MPI, I use the function of ***MPI_Init()*** to initialize the execution environment of MPI and use the function of ***MPI_Comm_rank()*** and ***MPI_Comm_size()*** (shown in Figure2.1) to get the current processor and the total number of processors running in this program. Finally, using the function of ***MPI_Finalize()*** (shown in Figure 2.2)to terminate the MPI process.

```
/* insert MPI functions to 1) start process, 2) get total number of processors and 3) process rank*/


   MPI_Init (&argc, &argv);
   MPI_Comm grid_comm;
   MPI_Comm_rank(MPI_COMM_WORLD, &rank);
   MPI_Comm_size(MPI_COMM_WORLD, &num_proc);
```

Figure 2.1

```
// Destroy MPI processes

MPI_Finalize();
//printf("Program Compeleted....\n");
return 0;
```

Figure 2.2

Second, I need to define the matrix size, processor's grid size and the size of each block. The matrix size actually defined by the developer themselves. In my program, I will define the matrix size when I running my program by using MPI mentioned in section 1(code is shown in Figure 2.3). To get the number of processor's grid size and the size of each block, I need to do simple calculations, where the size of the processor's grid is the square root of the total number of processors and the size of each block is the matrix size divided by the size of the processor's grid(code shown in Figure 2.4).

```
int SZ = 0;

if (argc == 2) {
    SZ = atoi(argv[1]);
}else{
    SZ = 4000;
}
```

Figure 2.3 Define matrix size by using the parameter of SZ

```
/* assign values to 1) proc_grid_sz and 2) block_sz*/

    proc_grid_sz = (int)sqrt((double)num_proc);
    block_sz = SZ/proc_grid_sz;
    dimsizes[0] = dimsizes[1] = proc_grid_sz;
    wraparound[0] = wraparound[1] = 1;
```

Figure 2.4 Calculate the size of processor's grid and block size

After defined the size of matrix, processor's grid and each block, the next step is to set up the communicator between processors that just processes single column and row by using ***MPI_Cart***(shown in Figure 2.5 and Figure 2.6).

```
MPI_Cart_create(MPI_COMM_WORLD, 2, dimsizes, wraparound, reorder, &grid_comm);
MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
MPI_Cart_coords(grid_comm, my_rank, 2, coordinates);
if (SZ % proc_grid_sz != 0){
    printf("Matrix size cannot be evenly split amongst resources!\n");
    printf("Quitting....\n");
    exit(-1);
}
```

Figure 2.5 Create MPI_Cart and Cartesian coordinate

```
MPI_Comm row_comm;
MPI_Comm col_comm;
//MPI_Comm grid_comm;

// MPI_Cart_create(MPI_COMM_WORLD, 2, dimsizes, wraparound, reorder, &grid_comm);
// MPI_Comm_rank(grid_comm, &my_rank);
// MPI_Cart_coords(grid_comm, my_rank, 2, coordinates);

free_coords[0] = 0;
free_coords[1] = 1;
MPI_Cart_sub(grid_comm,free_coords, &row_comm);

free_coords[0] = 1;
free_coords[1] = 0;
MPI_Cart_sub(grid_comm, free_coords, &col_comm);
```

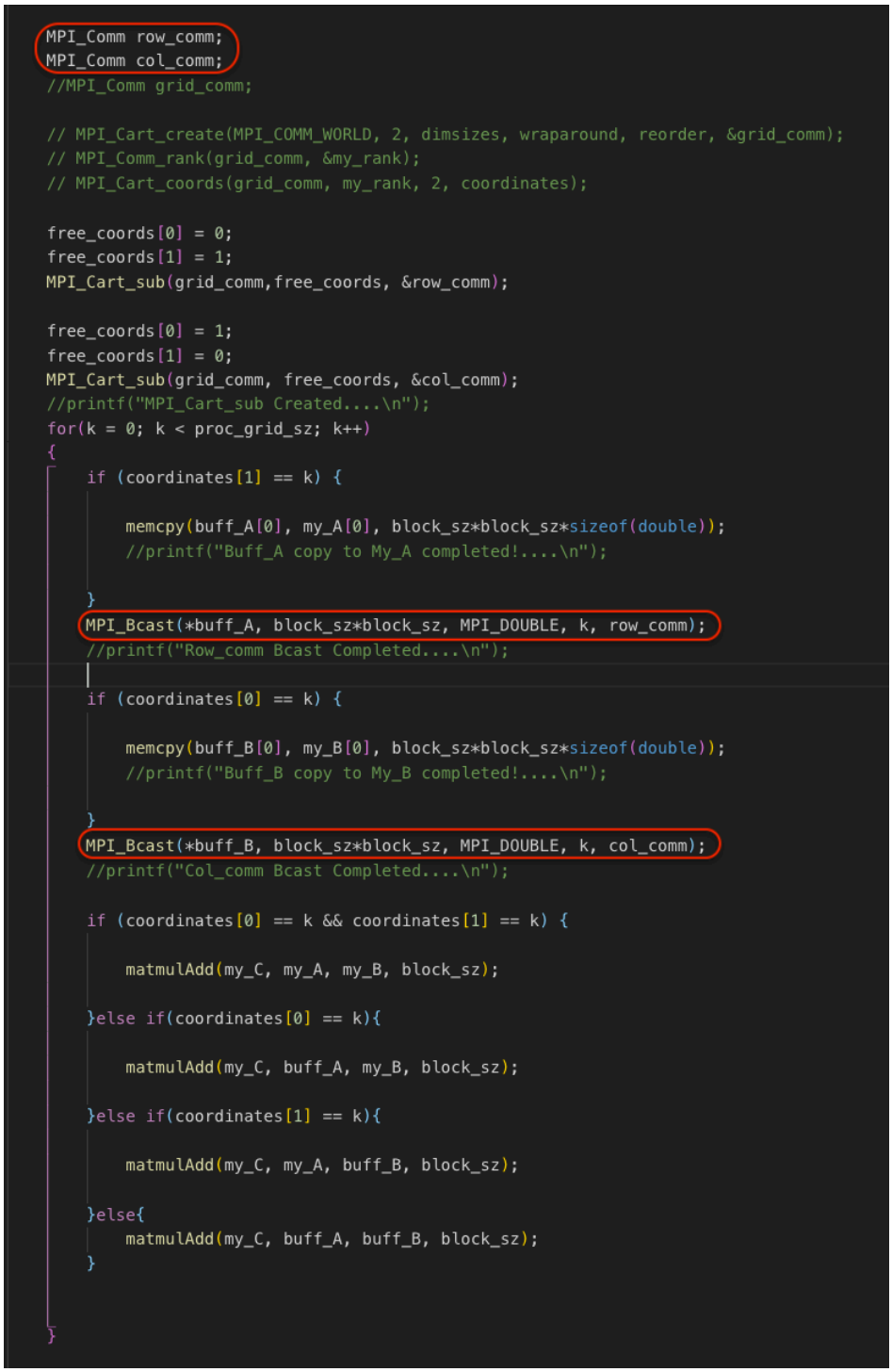
Figure 2.6 Create communicator

To collecting the computation time for the program, I need to use the function of ***MPI_Wtime()*** to collect the calculate time(shown in Figure 2.7).

```
// Use MPI_Wtime to get the starting time
start_time = MPI_Wtime();


// Use SUMMA algorithm to calculate product C
//printf("Summa Begin....\n");
matmul(rank, proc_grid_sz, block_sz, A, B, C, coordinates, grid_comm);
printf("Summa Finished....\n");

// Use MPI_Wtime to get the finishing time
end_time = MPI_Wtime();


// Obtain the elapsed time and assign it to total_time
total_time = end_time - start_time;
//printf("Computation Time: %f ms. \n", total_time);
```

Figure 2.7

In the next step, I need to implement the SUMMA algorithm. Two major part contained in SUMMA algorithm, one is Broadcast the matrix data from root processor to the other processors in the function of ***matmul()***(shown in Figure 2.8) and do the simple SUMMA matrix multiplication in the function of ***matmulAdd()*** (shown in Figure 2.9).



```
MPI_Comm row_comm;
MPI_Comm col_comm;
//MPI_Comm grid_comm;

// MPI_Cart_create(MPI_COMM_WORLD, 2, dimsizes, wraparound, reorder, &grid_comm);
// MPI_Comm_rank(grid_comm, &my_rank);
// MPI_Cart_coords(grid_comm, my_rank, 2, coordinates);

free_coords[0] = 0;
free_coords[1] = 1;
MPI_Cart_sub(grid_comm,free_coords, &row_comm);

free_coords[0] = 1;
free_coords[1] = 0;
MPI_Cart_sub(grid_comm, free_coords, &col_comm);
//printf("MPI_Cart_sub Created....\n");
for(k = 0; k < proc_grid_sz; k++)
{
    if (coordinates[1] == k) {

        memcpy(buff_A[0], my_A[0], block_sz*block_sz*sizeof(double));
        //printf("Buff_A copy to My_A completed!....\n");

    }
    MPI_Bcast(*buff_A, block_sz*block_sz, MPI_DOUBLE, k, row_comm);
    //printf("Row_comm Bcast Completed....\n");

    if (coordinates[0] == k) {

        memcpy(buff_B[0], my_B[0], block_sz*block_sz*sizeof(double));
        //printf("Buff_B copy to My_B completed!....\n");

    }
    MPI_Bcast(*buff_B, block_sz*block_sz, MPI_DOUBLE, k, col_comm);
    //printf("Col_comm Bcast Completed....\n");

    if (coordinates[0] == k && coordinates[1] == k) {

        matmulAdd(my_C, my_A, my_B, block_sz);

    }else if(coordinates[0] == k){

        matmulAdd(my_C, buff_A, my_B, block_sz);

    }else if(coordinates[1] == k){

        matmulAdd(my_C, my_A, buff_B, block_sz);

    }else{
        matmulAdd(my_C, buff_A, buff_B, block_sz);
    }


}
```

Figure 2.8 Broadcast

```c
//Basic SUMMA Algorithm
void matmulAdd(double **my_C, double **my_A, double **my_B, int block_sz){

    int i, j, k;

    for(k = 0; k < block_sz; k++)
    {
        for(i = 0; i < block_sz; i++)
        {
            for( j = 0; j < block_sz; j++)
            {
                my_C[i][j] += my_A[i][k] * my_B[k][j];
            }
        }
    }
}
```

Figure 2.9 Basic SUMMA algorithm

In order to confirm the correctness of SUMMA algorithm part mentioned above, I need to Initialize Matrix A and Matrix B to the special matrix such as bi/tridiagonal and check its correctness after done the computation(shown in Figure 2.10 and Figure 2.11).

```c
/**
*   Initialize arrays A and B with random numbers, and array C with zeros.
*   Each array is setup as a square block of blck_sz.
*   ii,jj is the index of each block
*   And i,j is the index of the whole matrix
**/

void initialize_for_test(double **lA, double **lB, double **lC, int blck_sz, int coordinates[2]){
    int i, j, ii, jj;

    for (ii=0; ii<blck_sz; ii++){
        for (jj=0; jj<blck_sz; jj++){
            i = ii + blck_sz * coordinates[0];
            j = jj + blck_sz * coordinates[1];

            if (i == j) {
                lA[ii][jj] = 1.0;
                lB[ii][jj] = 1.0;
            }else if( (j-1) == i ){
                lB[ii][jj] = 1.0;
                lA[ii][jj] = 0.0;
            }else if( (i-1)== j ){
                lA[ii][jj] = 1.0;
                lB[ii][jj] = 0.0;
            }else{
                lA[ii][jj] =0.0;
                lB[ii][jj] =0.0;
            }

            lC[ii][jj] = 0.0;
        }
    }

}
```

Figure 2.10 Initialize Matrix A and B as special matrices

```
// Insert statements for testing
printf("Matrix Comparation Testing Begin....\n");
int ii,jj,i,j;

for(ii = 0; ii < block_sz; ii++)
{

    for(jj = 0; jj < block_sz; jj++)
    {
        i = ii + block_sz * coordinates[0];
        j = jj + block_sz * coordinates[1];


        if (i == 0 && j==0 ) {

            if (C[ii][jj]!=1) {
                printf("C[%d][%d] is incorrect\n", ii,jj);

            }

        }else if(i == j){
            if (C[ii][jj]!=2) {
                printf("C[%d][%d] is incorrect\n", ii,jj);

            }
        }else if( (i-1) == j){
            if (C[ii][jj]!=1) {
                printf("C[%d][%d] is incorrect\n", ii,jj);

            }
        }else if(i == (j-1) ){
            if (C[ii][jj]!=1) {
                printf("C[%d][%d] is incorrect\n", ii,jj);

            }
        }

    }

}
printf("Matrix Comparation Testing Finished....\n");
```

Figure 2.11 Check the correctness

3. Performance

In this section, I will initialize matrices as random number(code shown in Figure 3.1) rather the function mentioned in the previous section.

```
void initialize(double **lA, double **lB, double **lC, int blck_sz){
//  Set random values...technically it is already random and this is redundant
    int i, j;
    for (i=0; i<blck_sz; i++){
        for (j=0; j<blck_sz; j++){
            lA[i][j] = (double)rand() / (double)RAND_MAX;
            lB[i][j] = (double)rand() / (double)RAND_MAX;
            lC[i][j] = 0.0;
        }
    }
}
```

Figure 3.1 Initialize matrices as random number

To view the performance in the different number of processors from the same matrix size or different matrix sizes from the same number of processors, I just set the number of processors to 1, 4, 9, 16, 32, 64 and the matrix size to 4200, 5040, and 8400. After check the running result, I found that execution time increases as the matrix size increases when those data run in the same number of processors. Similarly, the execution time decreases with the number of processors increases(the execution time I collected shown in Table 1 and the trend of running time shown in Chart 1).

Number of Professor	Matrix size:4200	Matrix size:5040	Matrix size:8400
1	471.227121 ms	816.586155 ms	3503.125226 ms
4	109.298071 ms	188.652501 ms	1587.568707 ms
9	50.96891 ms	88.124351 ms	387.614225 ms
16	29.933145 ms	46.245886 ms	221.479777 ms
25	19.080212 ms	29.787911 ms	151.87137 ms
36	13.229789 ms	20.705407 ms	106.184737 ms
49	9.566638 ms	15.336245 ms	77.306214 ms
64	7.294031 ms	11.973806 ms	58.183483 ms

Table 1 Running time at different matrix size and running in different number of processors

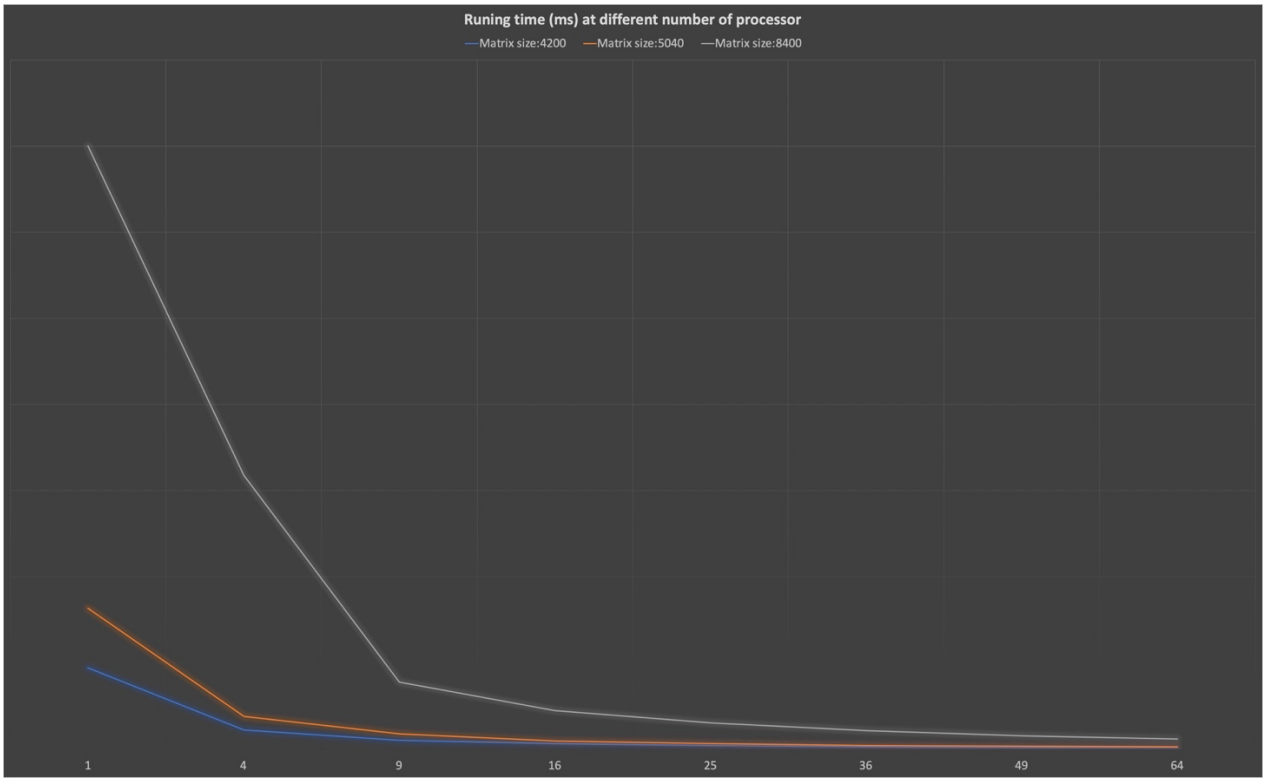


Chart 1 The trend of execution time

In Chart 1, the X-axis represents the number of processors that involves computation and the Y-axis represents execution time. The gray line represents the time trend of different processor operations when the matrix size is 8400. The orange line represents the time trend of different processor operations when the matrix size is 5040. The blue line represents the time trend of different processor operations when the matrix size is 4200.

I will show screenshots of the execution time of different processor sizes for different processor sizes below:

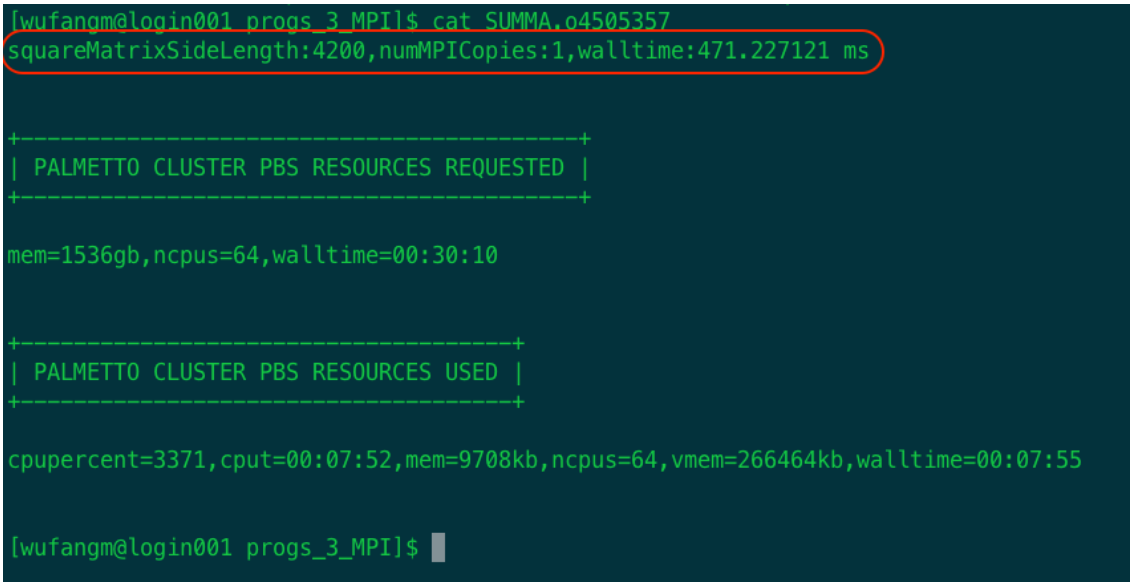


Figure 3.2 The execution time when processor's number is 1 and matrix size is 4200

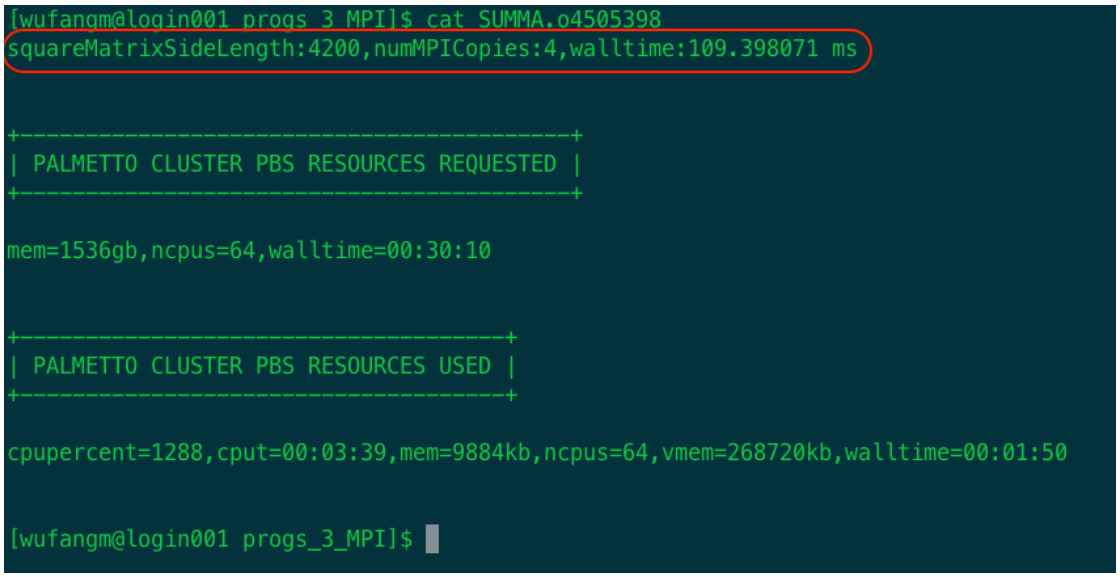


Figure 3.3 The execution time when processor's number is 4 and matrix size is 4200

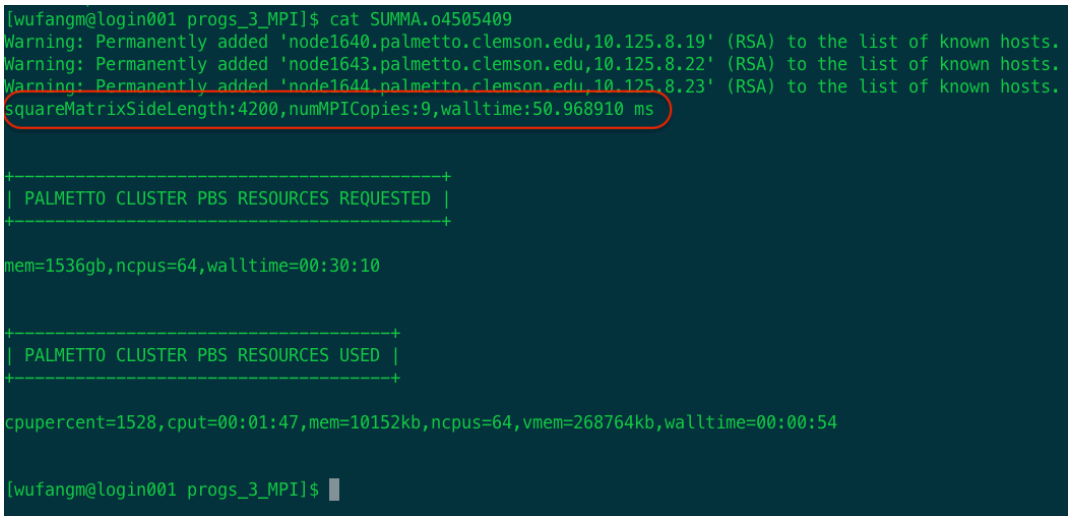


Figure 3.4 The execution time when processor's number is 9 and matrix size is 4200

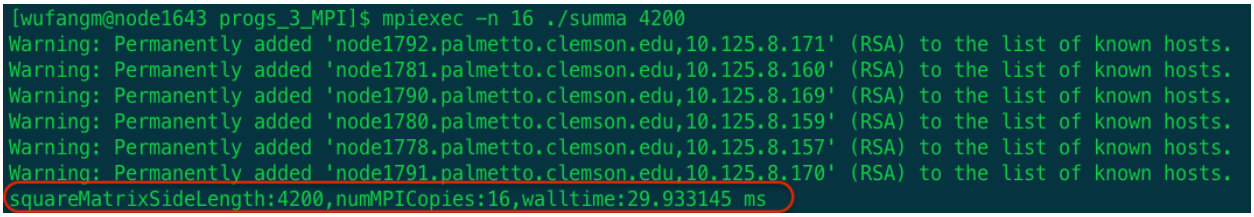


Figure 3.5 The execution time when processor's number is 16 and matrix size is 4200

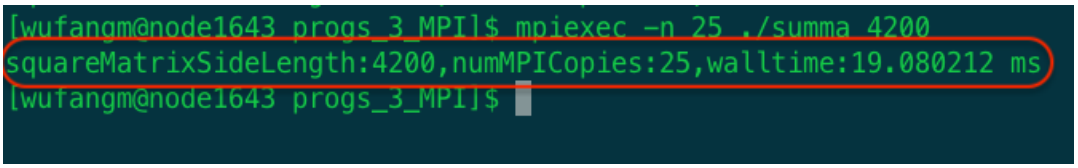


Figure 3.6 The execution time when processor's number is 25 and matrix size is 4200

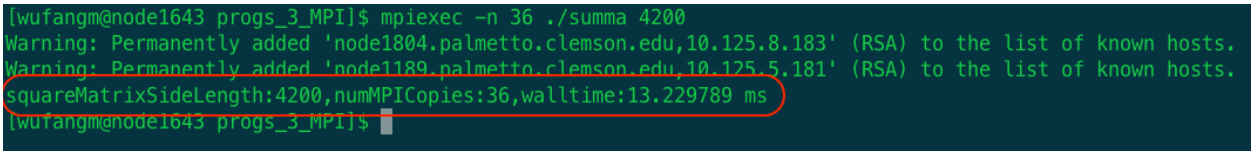


Figure 3.7 The execution time when processor's number is 36 and matrix size is 4200

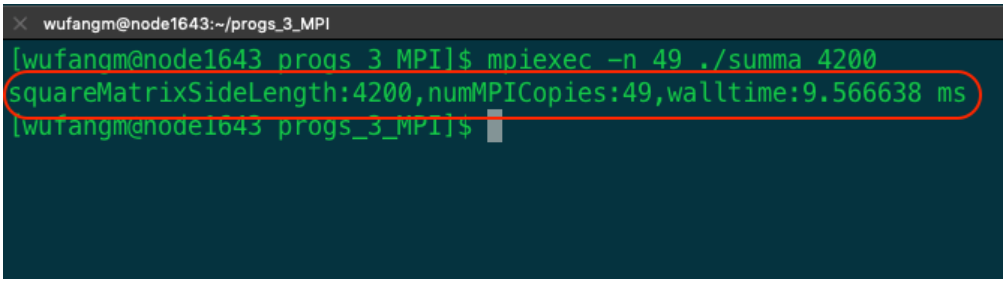


Figure 3.8 The execution time when processor's number is 49 and matrix size is 4200

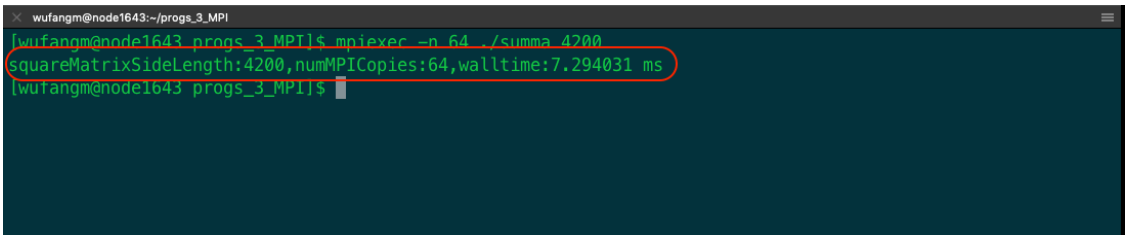


Figure 3.9 The execution time when processor's number is 64 and matrix size is 4200

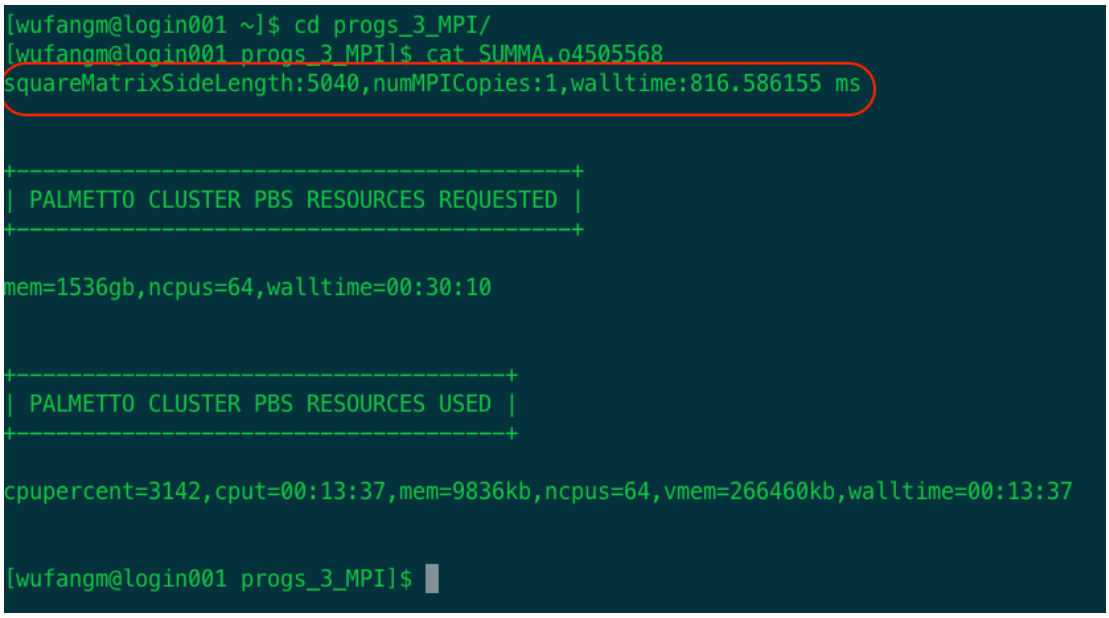


Figure 3.10 The execution time when processor's number is 1 and matrix size is 5040

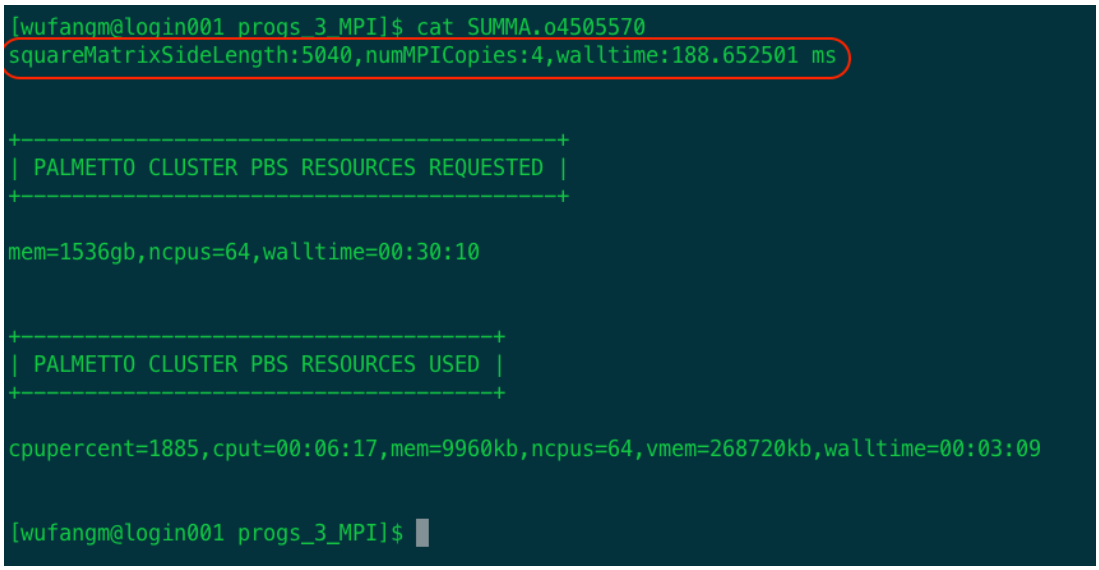


Figure 3.11 The execution time when processor's number is 4 and matrix size is 5040

```
[wufangm@login001 progs_3_MPI]$ cat SUMMA.o4505571
Warning: Permanently added 'node1904.palmetto.clemson.edu,10.125.10.27' (RSA) to the list of known hosts.
Warning: Permanently added 'node1903.palmetto.clemson.edu,10.125.10.26' (RSA) to the list of known hosts.
squareMatrixSideLength:5040,numMPICopies:9,walltime:88.124351 ms


+-----------------------------------------+
| PALMETTO CLUSTER PBS RESOURCES REQUESTED |
+-----------------------------------------+

mem=1536gb,ncpus=64,walltime=00:30:10


+-------------------------------------+
| PALMETTO CLUSTER PBS RESOURCES USED |
+-------------------------------------+

cpupercent=3700,cput=00:07:24,mem=9912kb,ncpus=64,vmem=268728kb,walltime=00:01:29


[wufangm@login001 progs_3_MPI]$
```

Figure 3.12 The execution time when processor's number is 9 and matrix size is 5040

```
[wufangm@node1793 ~]$ module load gcc/8.2.0
[wufangm@node1793 ~]$ module load mpich/3.1.4
[wufangm@node1793 ~]$ cd progs_3_MPI/
[wufangm@node1793 progs_3_MPI]$ mpiexec -n 16 ./summa 5040
Warning: Permanently added 'node1795.palmetto.clemson.edu,10.125.8.174' (RSA) to the list of known hosts.
Warning: Permanently added 'node1803.palmetto.clemson.edu,10.125.8.182' (RSA) to the list of known hosts.
squareMatrixSideLength:5040,numMPICopies:16,walltime:46.245886 ms
[wufangm@node1793 progs_3_MPI]$ mpiexec -n 25 ./summa 5040
squareMatrixSideLength:5040,numMPICopies:25,walltime:29.787911 ms
[wufangm@node1793 progs_3_MPI]$ mpiexec -n 36 ./summa 5040
squareMatrixSideLength:5040,numMPICopies:36,walltime:20.706507 ms
[wufangm@node1793 progs_3_MPI]$ mpiexec -n 49 ./summa 5040
squareMatrixSideLength:5040,numMPICopies:49,walltime:15.336245 ms
[wufangm@node1793 progs_3_MPI]$ mpiexec -n 64 ./summa 5040
squareMatrixSideLength:5040,numMPICopies:64,walltime:11.973806 ms
[wufangm@node1793 progs_3_MPI]$
```

Figure 3.13 The execution time when processor's number is 16,25,36,49,64 and matrix size is 5040

```
[wufangm@login001 progs_3_MPI]$ cat SUMMA.o4506921
squareMatrixSideLength:8400,numMPICopies:1,walltime:3503.125226 ms


+-----------------------------------------+
| PALMETTO CLUSTER PBS RESOURCES REQUESTED |
+-----------------------------------------+

mem=1536gb,ncpus=64,walltime=01:45:10


+-------------------------------------+
| PALMETTO CLUSTER PBS RESOURCES USED |
+-------------------------------------+

cpupercent=9025,cput=00:58:25,mem=9844kb,ncpus=64,vmem=266464kb,walltime=00:58:26


[wufangm@login001 progs_3_MPI]$
```

Figure 3.14 The execution time when processor's number is 1 and matrix size is 8400

```
[wufangm@login001 progs_3_MPI]$ cat SUMMA.o4506918
Warning: Permanently added 'node1809.palmetto.clemson.edu,10.125.9.5' (RSA) to the list of known hosts.
squareMatrixSideLength:8400,numMPICopies:4,walltime:1587.568707 ms


+-----------------------------------------+
| PALMETTO CLUSTER PBS RESOURCES REQUESTED |
+-----------------------------------------+

mem=1536gb,ncpus=64,walltime=00:45:10


+--------------------------------------+
| PALMETTO CLUSTER PBS RESOURCES USED |
+--------------------------------------+

cpupercent=7322,cput=00:26:28,mem=10264kb,ncpus=64,vmem=268732kb,walltime=00:39:17


[wufangm@login001 progs_3_MPI]$
```

Figure 3.15 The execution time when processor's number is 4 and matrix size is 8400

```
[wufangm@login001 progs_3_MPI]$ cat SUMMA.o4506919
Warning: Permanently added 'node1920.palmetto.clemson.edu,10.125.10.43' (RSA) to the list of known hosts.
Warning: Permanently added 'node1921.palmetto.clemson.edu,10.125.10.44' (RSA) to the list of known hosts.
Warning: Permanently added 'node1919.palmetto.clemson.edu,10.125.10.42' (RSA) to the list of known hosts.
squareMatrixSideLength:8400,numMPICopies:9,walltime:397.614225 ms


+-----------------------------------------+
| PALMETTO CLUSTER PBS RESOURCES REQUESTED |
+-----------------------------------------+

mem=1536gb,ncpus=64,walltime=00:45:10


+--------------------------------------+
| PALMETTO CLUSTER PBS RESOURCES USED |
+--------------------------------------+

cpupercent=2846,cput=00:13:17,mem=9908kb,ncpus=64,vmem=268740kb,walltime=00:06:39


[wufangm@login001 progs_3_MPI]$
```

Figure 3.16 The execution time when processor's number is 9 and matrix size is 8400

```
[wufangm@login001 progs_3_MPI]$ cat SUMMA.o4506920
Warning: Permanently added 'node1627.palmetto.clemson.edu,10.125.8.5' (RSA) to the list of known hosts.
Warning: Permanently added 'node1628.palmetto.clemson.edu,10.125.8.6' (RSA) to the list of known hosts.
Warning: Permanently added 'node1918.palmetto.clemson.edu,10.125.10.41' (RSA) to the list of known hosts.
squareMatrixSideLength:8400,numMPICopies:16,walltime:221.479777 ms


+----------------------------------------+
| PALMETTO CLUSTER PBS RESOURCES REQUESTED |
+----------------------------------------+

mem=1536gb,ncpus=64,walltime=00:45:10


+--------------------------------------+
| PALMETTO CLUSTER PBS RESOURCES USED |
+--------------------------------------+

cpupercent=3185,cput=00:03:43,mem=10128kb,ncpus=64,vmem=268872kb,walltime=00:03:42


[wufangm@login001 progs_3_MPI]$
```

Figure 3.16 The execution time when processor's number is 16 and matrix size is 8400

```
[wufangm@node1673 ~]$ module load gcc/8.2.0
[wufangm@node1673 ~]$ module load mpich/3.1.4
[wufangm@node1673 ~]$ cd progs_3_MPI/
[wufangm@node1673 progs_3_MPI]$ mpiexec -n 25 ./summa 8400
Warning: Permanently added 'node0653.palmetto.clemson.edu,10.125.3.142' (RSA) to the list of known hosts.
Warning: Permanently added 'node1113.palmetto.clemson.edu,10.125.5.105' (RSA) to the list of known hosts.
Warning: Permanently added 'node0046.palmetto.clemson.edu,10.125.1.46' (RSA) to the list of known hosts.
squareMatrixSideLength:8400,numMPICopies:25,walltime:151.871370 ms
[wufangm@node1673 progs_3_MPI]$ mpiexec -n 36 ./summa 8400
squareMatrixSideLength:8400,numMPICopies:36,walltime:106.184737 ms
[wufangm@node1673 progs_3_MPI]$ mpiexec -n 49 ./summa 8400
squareMatrixSideLength:8400,numMPICopies:49,walltime:77.306214 ms
[wufangm@node1673 progs_3_MPI]$ mpiexec -n 64 ./summa 8400
Warning: Permanently added 'node1711.palmetto.clemson.edu,10.125.8.90' (RSA) to the list of known hosts.
squareMatrixSideLength:8400,numMPICopies:64,walltime:58.183483 ms
[wufangm@node1673 progs_3_MPI]$
```

Figure 3.16 The execution time when processor's number is 25,36,49,64 and matrix size is 8400