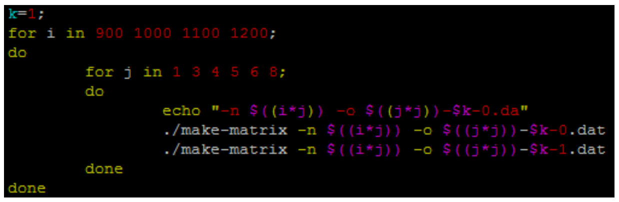
# Homework 06 Report

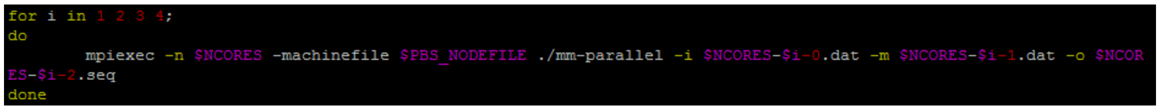
**Liuhua Chen**

I ran both serial and parallel versions of cannon's algorithms on varying numbers of processors. The number of processors varies from 1 to 64. I chose N=900, 1000, 1100 and 1200 as the sizes of the matrix for each processor. I developed a bash script *gen.sh* to generate the matrix files, *e.g. p-i-0.dat and p-i-1.dat*, where *p* denotes the number of processors, *i=1,2,3,4* denotes different workload sizes (*e.g. 900, 1000, 1100 and 1200*). The *gen.sh* looks like:



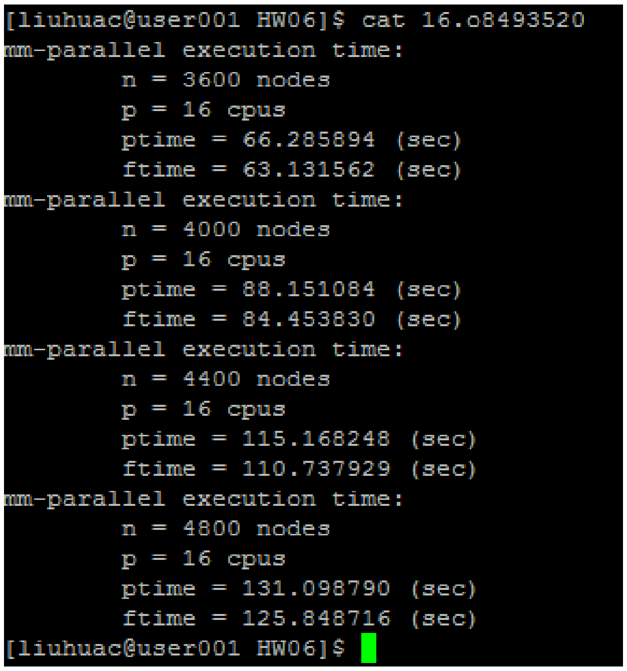
```
k=1;
for i in 900 1000 1100 1200;
do
        for j in 1 3 4 5 6 8;
        do
                echo "-n $((i*j)) -o $((j*j))-$k-0.da"
                ./make-matrix -n $((i*j)) -o $((j*j))-$k-0.dat
                ./make-matrix -n $((i*j)) -o $((j*j))-$k-1.dat
        done
done
```

I wrote a sets of pbs scripts *pbs-n.sh*, with *n=1, 9, 16, 25, 36* and *64*, regarding running the cannon's algorithm in n number of processors, respectively. Within each script, I designed a for-loop, making the algorithm run on the matrix files. The for-loop is shown as follows:



```
for i in 1 2 3 4;
do
        mpiexec -n $NCORES -machinefile $PBS_NODEFILE ./mm-parallel -i $NCORES-$i-0.dat -m $NCORES-$i-1.dat -o $NCOR
ES-$i-2.seq
done
```

I also made a sub-all.sh script to submit all the jobs to Palmetto at one time. Upon job completion, I had the output files from Palmetto as shown in follows:



```
[liuhuac@user001 HW06]$ cat 16.o8493520
mm-parallel execution time:
        n = 3600 nodes
        p = 16 cpus
        ptime = 66.285894 (sec)
        ftime = 63.131562 (sec)
mm-parallel execution time:
        n = 4000 nodes
        p = 16 cpus
        ptime = 88.151084 (sec)
        ftime = 84.453830 (sec)
mm-parallel execution time:
        n = 4400 nodes
        p = 16 cpus
        ptime = 115.168248 (sec)
        ftime = 110.737929 (sec)
mm-parallel execution time:
        n = 4800 nodes
        p = 16 cpus
        ptime = 131.098790 (sec)
        ftime = 125.848716 (sec)
[liuhuac@user001 HW06]$
```

When all the experiments were finished, I showed the results in the following figures:

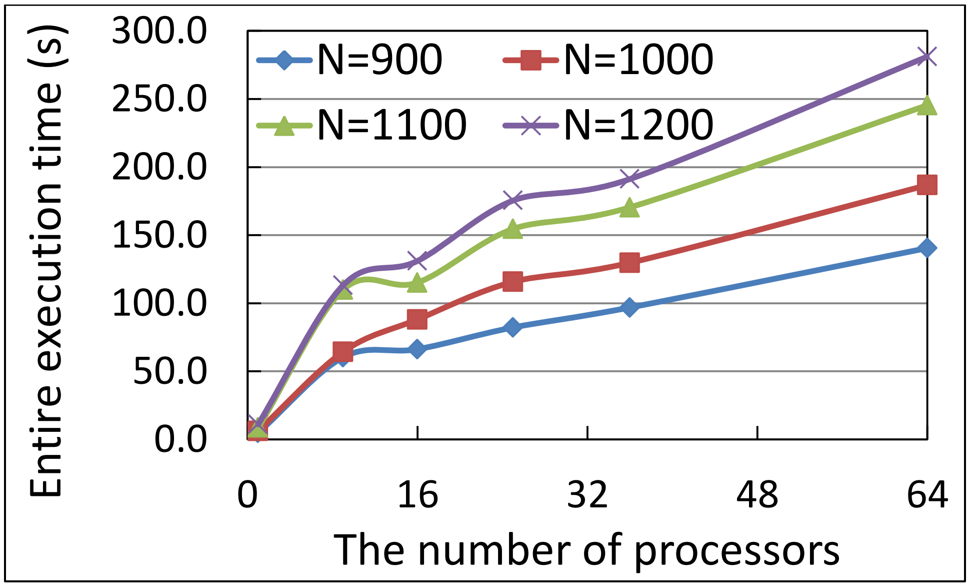


Figure 1

Figure 1 shows the entire execution time of cannon's algorithm on varying numbers of processors. We see that both serial and parallel versions have execution time that increases with the size of the matrix (N). For a specific matrix size per processor, the execution time increases with the number of processors. The reason is that communication overhead increases as the number of processors.

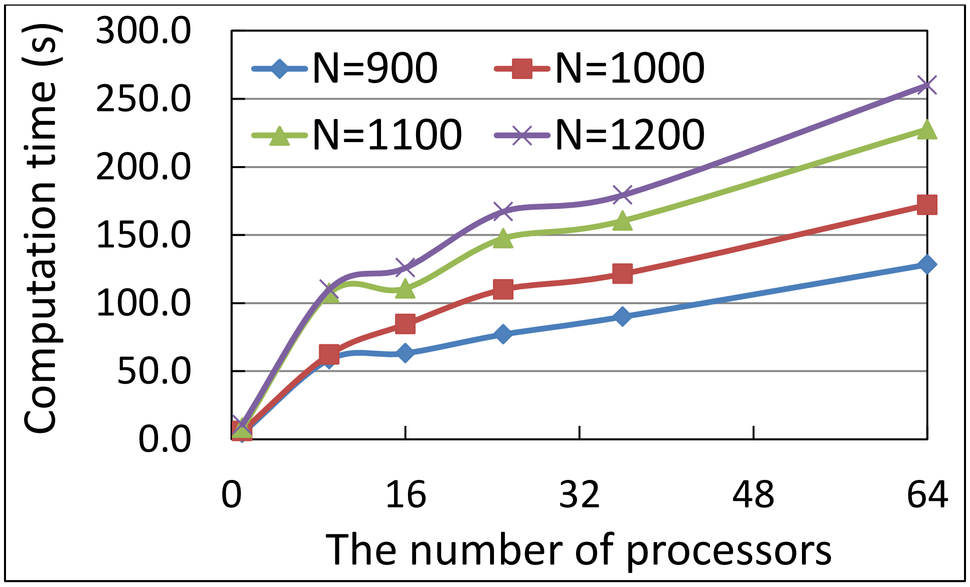


Figure 2

Figure 2 shows the computation time of cannon's algorit hm that does not include reading/writing files. We see that the computation time is approximately the same with the entire time as shown in Figure 1. This result indicates that commutation dominates in cannon's algorithm as compared to I/O processing.

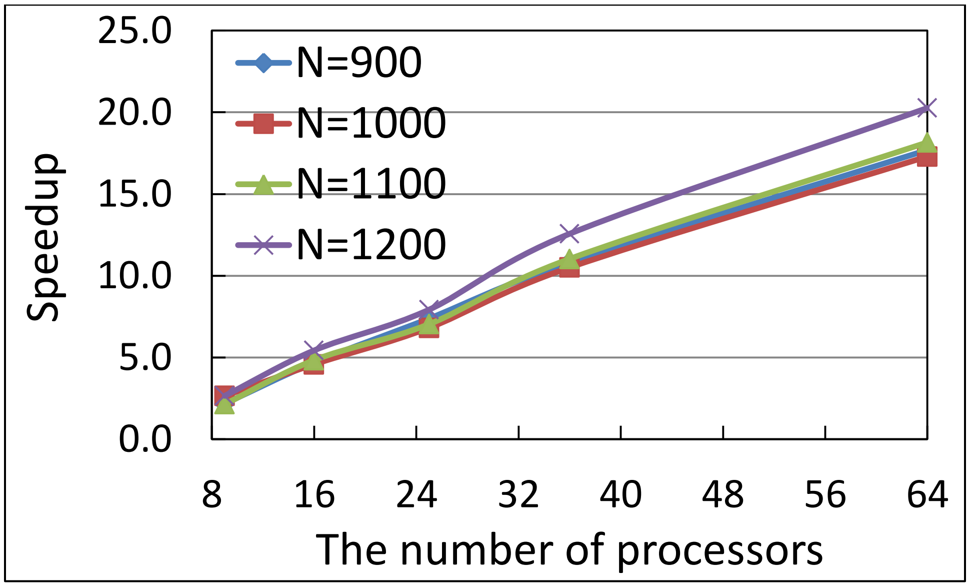


Figure 3

We ran experiments to record the execution time with respect to the size of matrix per processor. That is 900*900, 1000*1000, 1100*1100 and 1200*1200 for each processor. In order to calculate speedup, we need to compare the time of both serial and parallel program on the same size of workload. In this homework, I scaled the time of the serial program to estimate the time it requires to compute the workload of the same size of the parallel version. For example, I scaled the time of serial program on 900*900 by multiplying n^(3/2) to get the time that on a (900*900)*n matrix. Figure 3 shows the speedup of the parallel program as a function of the number of processors. We see that the speedup increases with the number of processors. The speedup ratio is not sensitive to the size of matrix N. We also see that the algorithm scales well under the matrix sizes in the experiment.

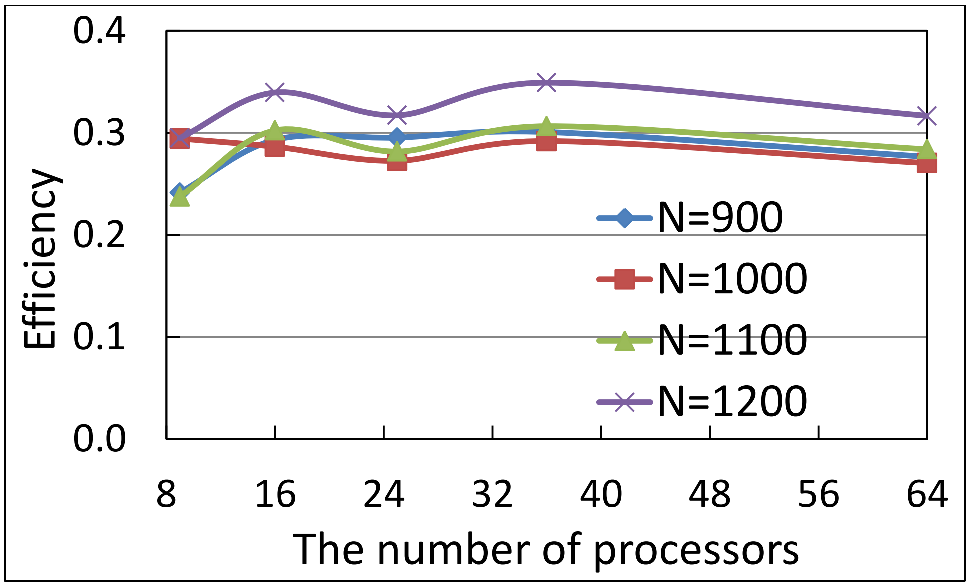


Figure 4

Figure 4 shows the efficiency of the parallel program as a function of the number of processors. We see that the efficiencies remain at a stable level (0.3). The efficiencies do not decrease as the size of the matrix increase, which means that the algorithm scales well under the workload sized used in the experiment.

**File lists:**

1. Report file: MPI-HW06.pdf
2. Figure source file: MPI-HW06.xlsx
3. Experiment output files in output subfolder:
```
-rwx------ 1 liuhuac cuuser   436 Nov 25 17:45 64.o8493523
-rwx------ 1 liuhuac cuuser   434 Nov 25 17:31 36.o8493522
-rwx------ 1 liuhuac cuuser   434 Nov 25 17:30 25.o8493521
-rwx------ 1 liuhuac cuuser   432 Nov 25 17:28 16.o8493520
-rwx------ 1 liuhuac cuuser   428 Nov 25 17:27 9.o8493519
-rwx------ 1 liuhuac cuuser   829 Nov 25 17:24 1.o8493525
```
4. pbs script files:
```
-rwx------ 1 liuhuac cuuser 383 Nov 25 17:23 pbs-1.sh
-rwx------ 1 liuhuac cuuser 439 Nov 25 17:19 pbs-16.sh
-rwx------ 1 liuhuac cuuser 439 Nov 25 17:19 pbs-25.sh
-rwx------ 1 liuhuac cuuser 439 Nov 25 17:19 pbs-36.sh
-rwx------ 1 liuhuac cuuser 439 Nov 25 17:15 pbs-64.sh
-rwx------ 1 liuhuac cuuser 438 Nov 25 16:32 pbs-9.sh
```
5. bash script for submitting all the jobs:
```
-rwx------ 1 liuhuac cuuser  63 Nov 25 17:21 sub-all.sh
```
6. bash script for matrix generation:
```
-rwx------ 1 liuhuac cuuser 471 Nov 25 17:39 gen.sh
```