Machine used to test: Oxford (Very busy on Thursday)

Since the machine was very busy by the time period, so I didn’t test size of 2048 for both version with a bash script but did later by hand.

C Section

Grids sizes used for test:

64*64 128*128 256*256 512*512 1024*1024 and 2048*2048

Note: The size of 2048 was test after along with Java version.

Command to run:

./JacobiC <N> <numThread>

Note: The option arguments are implemented into the program. For the convenience of testing, I just use the default value for the edges and epsilon which are: 10,10,800,800,0.1.

Bash script to run the test:

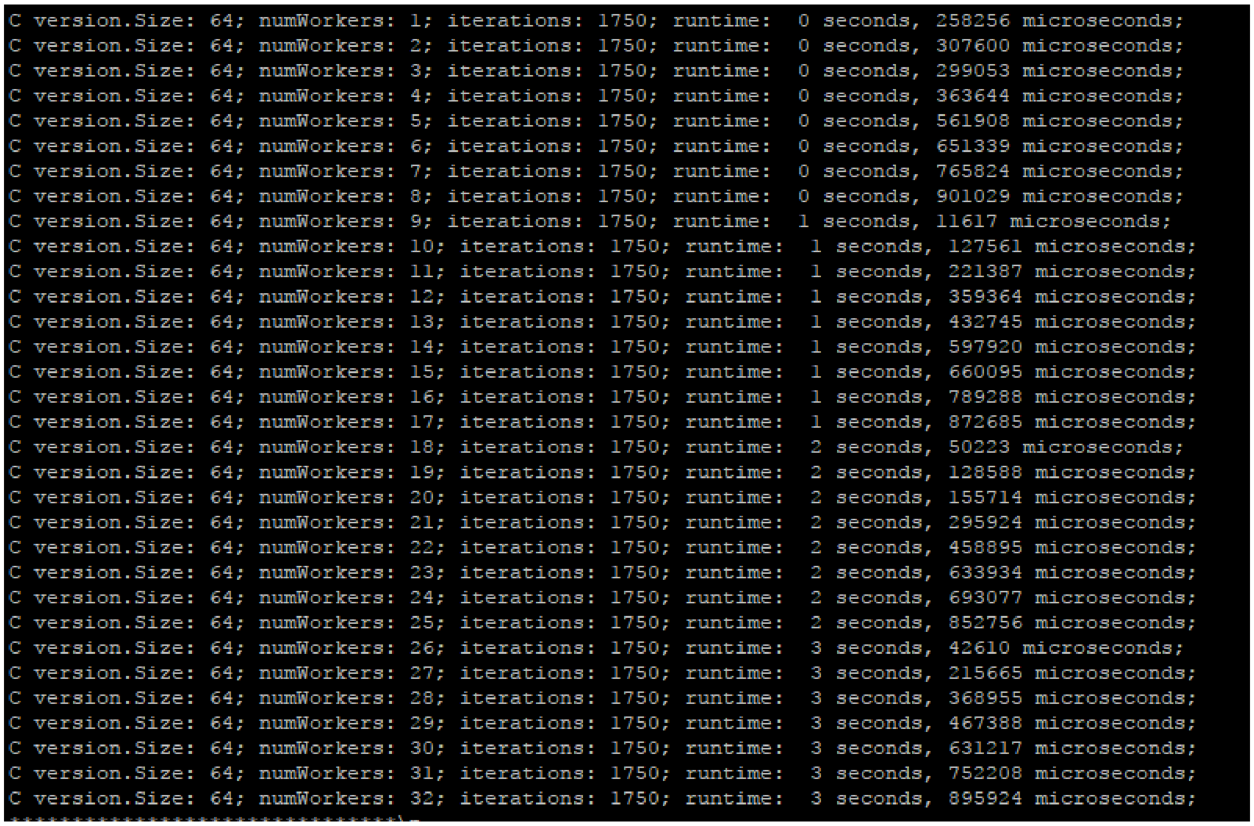
```
#! /bin/bash

if [[  ! -f JacobiC  ]]
then
        echo "cannot find the file JacobiC."
        exit 1
fi


for i in 64 128 256 512 1024
        do
        for j in {1..32}
        do ./JacobiC $i $j
        done
        echo "*******************************\n"
done

echo "---------------------------------"
```

Result:

The entire result for C version is included in the text file “Cto1024.txt”.

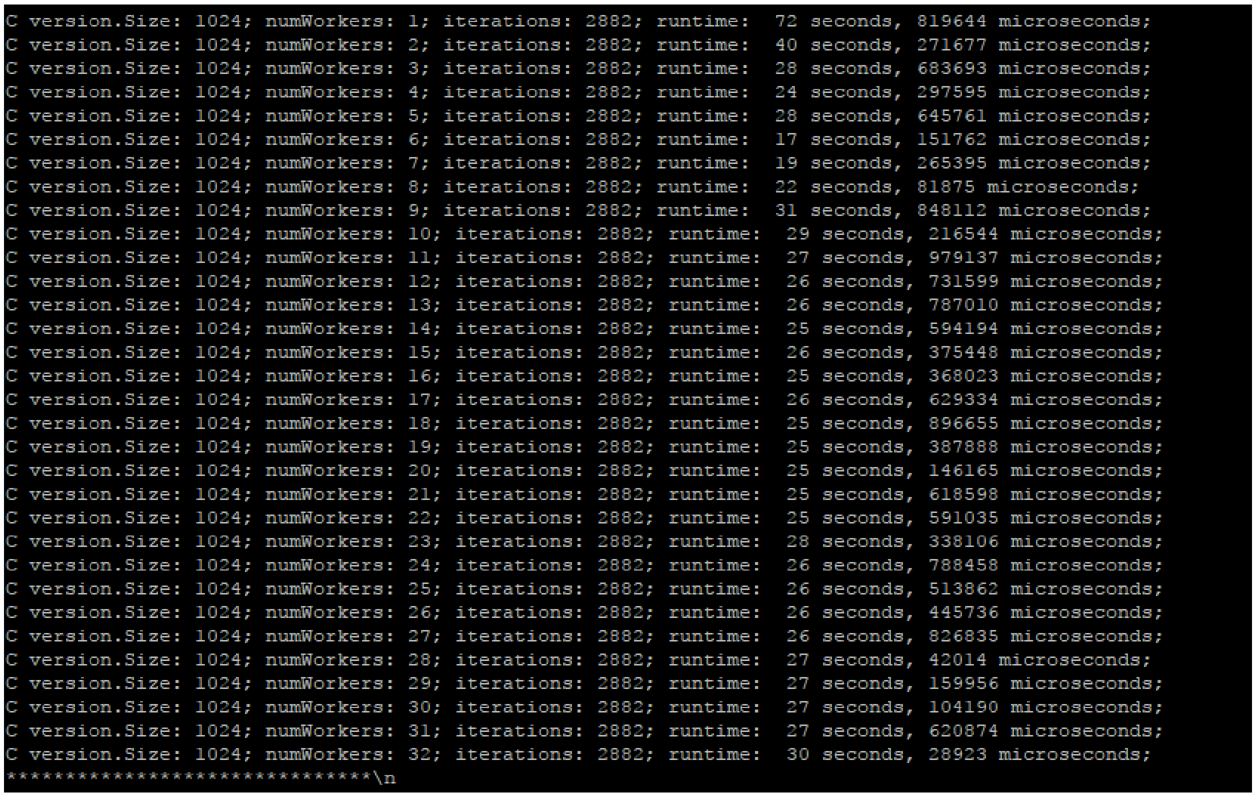
Based on the result, when the size of grid is small, the increase of number of workers does not speed up (cost less time). On the contrary, it cost more which means the performance decreases.

An example of size 64*64 is below:



```
C version.Size: 64; numWorkers: 1; iterations: 1750; runtime:  0 seconds, 258256 microseconds;
C version.Size: 64; numWorkers: 2; iterations: 1750; runtime:  0 seconds, 307600 microseconds;
C version.Size: 64; numWorkers: 3; iterations: 1750; runtime:  0 seconds, 299053 microseconds;
C version.Size: 64; numWorkers: 4; iterations: 1750; runtime:  0 seconds, 363644 microseconds;
C version.Size: 64; numWorkers: 5; iterations: 1750; runtime:  0 seconds, 561908 microseconds;
C version.Size: 64; numWorkers: 6; iterations: 1750; runtime:  0 seconds, 651339 microseconds;
C version.Size: 64; numWorkers: 7; iterations: 1750; runtime:  0 seconds, 765824 microseconds;
C version.Size: 64; numWorkers: 8; iterations: 1750; runtime:  0 seconds, 901029 microseconds;
C version.Size: 64; numWorkers: 9; iterations: 1750; runtime:  1 seconds, 11617 microseconds;
C version.Size: 64; numWorkers: 10; iterations: 1750; runtime:  1 seconds, 127561 microseconds;
C version.Size: 64; numWorkers: 11; iterations: 1750; runtime:  1 seconds, 221387 microseconds;
C version.Size: 64; numWorkers: 12; iterations: 1750; runtime:  1 seconds, 359364 microseconds;
C version.Size: 64; numWorkers: 13; iterations: 1750; runtime:  1 seconds, 432745 microseconds;
C version.Size: 64; numWorkers: 14; iterations: 1750; runtime:  1 seconds, 597920 microseconds;
C version.Size: 64; numWorkers: 15; iterations: 1750; runtime:  1 seconds, 660095 microseconds;
C version.Size: 64; numWorkers: 16; iterations: 1750; runtime:  1 seconds, 789288 microseconds;
C version.Size: 64; numWorkers: 17; iterations: 1750; runtime:  1 seconds, 872685 microseconds;
C version.Size: 64; numWorkers: 18; iterations: 1750; runtime:  2 seconds, 50223 microseconds;
C version.Size: 64; numWorkers: 19; iterations: 1750; runtime:  2 seconds, 128588 microseconds;
C version.Size: 64; numWorkers: 20; iterations: 1750; runtime:  2 seconds, 155714 microseconds;
C version.Size: 64; numWorkers: 21; iterations: 1750; runtime:  2 seconds, 295924 microseconds;
C version.Size: 64; numWorkers: 22; iterations: 1750; runtime:  2 seconds, 458895 microseconds;
C version.Size: 64; numWorkers: 23; iterations: 1750; runtime:  2 seconds, 633934 microseconds;
C version.Size: 64; numWorkers: 24; iterations: 1750; runtime:  2 seconds, 693077 microseconds;
C version.Size: 64; numWorkers: 25; iterations: 1750; runtime:  2 seconds, 852756 microseconds;
C version.Size: 64; numWorkers: 26; iterations: 1750; runtime:  3 seconds, 42610 microseconds;
C version.Size: 64; numWorkers: 27; iterations: 1750; runtime:  3 seconds, 215665 microseconds;
C version.Size: 64; numWorkers: 28; iterations: 1750; runtime:  3 seconds, 368955 microseconds;
C version.Size: 64; numWorkers: 29; iterations: 1750; runtime:  3 seconds, 467388 microseconds;
C version.Size: 64; numWorkers: 30; iterations: 1750; runtime:  3 seconds, 631217 microseconds;
C version.Size: 64; numWorkers: 31; iterations: 1750; runtime:  3 seconds, 752208 microseconds;
C version.Size: 64; numWorkers: 32; iterations: 1750; runtime:  3 seconds, 895924 microseconds;
```

When the size of the grid goes to large, the increasing of number of workers does optimize the performance and that means less time cost. But the performance will decrease a little after a point. An example of size = 1024*1024 is below:



```
C version.Size: 1024; numWorkers: 1; iterations: 2882; runtime:  72 seconds, 819644 microseconds;
C version.Size: 1024; numWorkers: 2; iterations: 2882; runtime:  40 seconds, 271677 microseconds;
C version.Size: 1024; numWorkers: 3; iterations: 2882; runtime:  28 seconds, 683693 microseconds;
C version.Size: 1024; numWorkers: 4; iterations: 2882; runtime:  24 seconds, 297595 microseconds;
C version.Size: 1024; numWorkers: 5; iterations: 2882; runtime:  28 seconds, 645761 microseconds;
C version.Size: 1024; numWorkers: 6; iterations: 2882; runtime:  17 seconds, 151762 microseconds;
C version.Size: 1024; numWorkers: 7; iterations: 2882; runtime:  19 seconds, 265395 microseconds;
C version.Size: 1024; numWorkers: 8; iterations: 2882; runtime:  22 seconds, 81875 microseconds;
C version.Size: 1024; numWorkers: 9; iterations: 2882; runtime:  31 seconds, 848112 microseconds;
C version.Size: 1024; numWorkers: 10; iterations: 2882; runtime:  29 seconds, 216544 microseconds;
C version.Size: 1024; numWorkers: 11; iterations: 2882; runtime:  27 seconds, 979137 microseconds;
C version.Size: 1024; numWorkers: 12; iterations: 2882; runtime:  26 seconds, 731599 microseconds;
C version.Size: 1024; numWorkers: 13; iterations: 2882; runtime:  26 seconds, 787010 microseconds;
C version.Size: 1024; numWorkers: 14; iterations: 2882; runtime:  25 seconds, 594194 microseconds;
C version.Size: 1024; numWorkers: 15; iterations: 2882; runtime:  26 seconds, 375448 microseconds;
C version.Size: 1024; numWorkers: 16; iterations: 2882; runtime:  25 seconds, 368023 microseconds;
C version.Size: 1024; numWorkers: 17; iterations: 2882; runtime:  26 seconds, 629334 microseconds;
C version.Size: 1024; numWorkers: 18; iterations: 2882; runtime:  25 seconds, 896655 microseconds;
C version.Size: 1024; numWorkers: 19; iterations: 2882; runtime:  25 seconds, 387888 microseconds;
C version.Size: 1024; numWorkers: 20; iterations: 2882; runtime:  25 seconds, 146165 microseconds;
C version.Size: 1024; numWorkers: 21; iterations: 2882; runtime:  25 seconds, 618598 microseconds;
C version.Size: 1024; numWorkers: 22; iterations: 2882; runtime:  25 seconds, 591035 microseconds;
C version.Size: 1024; numWorkers: 23; iterations: 2882; runtime:  28 seconds, 338106 microseconds;
C version.Size: 1024; numWorkers: 24; iterations: 2882; runtime:  26 seconds, 788458 microseconds;
C version.Size: 1024; numWorkers: 25; iterations: 2882; runtime:  26 seconds, 513862 microseconds;
C version.Size: 1024; numWorkers: 26; iterations: 2882; runtime:  26 seconds, 445736 microseconds;
C version.Size: 1024; numWorkers: 27; iterations: 2882; runtime:  26 seconds, 826835 microseconds;
C version.Size: 1024; numWorkers: 28; iterations: 2882; runtime:  27 seconds, 42014 microseconds;
C version.Size: 1024; numWorkers: 29; iterations: 2882; runtime:  27 seconds, 159956 microseconds;
C version.Size: 1024; numWorkers: 30; iterations: 2882; runtime:  27 seconds, 104190 microseconds;
C version.Size: 1024; numWorkers: 31; iterations: 2882; runtime:  27 seconds, 620874 microseconds;
C version.Size: 1024; numWorkers: 32; iterations: 2882; runtime:  30 seconds, 28923 microseconds;
*******************************\n
```

Java Section:

Grids sizes used for test:

64*64 128*128 256*256 512*512 1024*1024 and 2048*2048

Note: The size of 2048 was test after along with C version.

Command to run:

```
java Jacobi <N>  <numThread>
```

Note: The option arguments are implemented into the program. For the convenience of testing, I just use the default value for the edges and epsilon which are: 10,10,800,800,0.1.

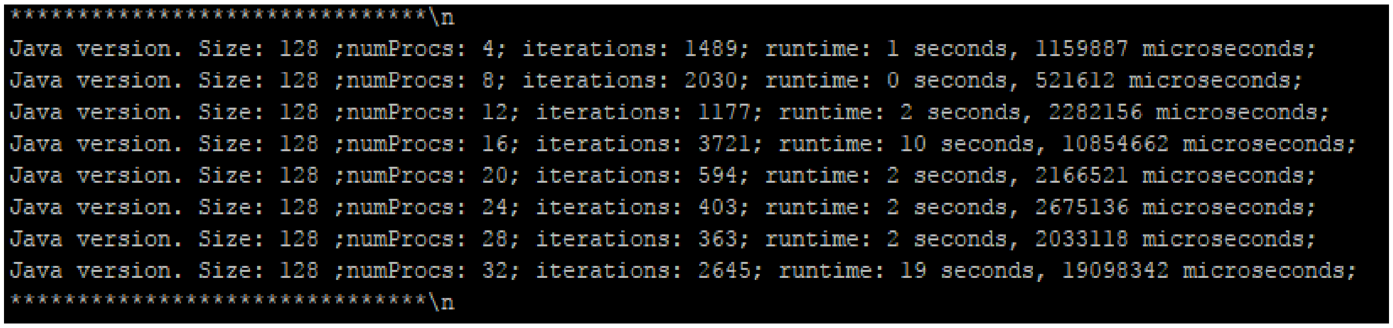
Bash script to run the test:

```
#! /bin/bash

if [[  ! -f Jacobi.class  ]]
then
        echo "cannot find the file Jacobi.class"
        exit 1
fi


for i in 64 128 256 512 1024
        do
        for j in 4 8 12 16 20 24 28 32
        do
                java Jacobi $i $j
        done
        echo "*******************************\n"
done

echo "---------------------------------"
```

Result:

The entire result for Java section is included in the file “Javato1024.txt”.

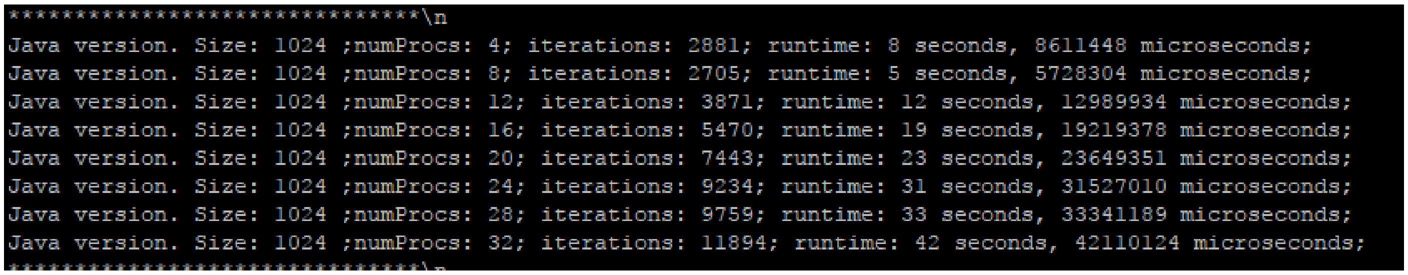
Based on the result, when the grids size is small, the number of workers increases does not mean the better performance. An example of grid size 128*128 is below:



```
*******************************\n
Java version. Size: 128 ;numProcs: 4; iterations: 1489; runtime: 1 seconds, 1159887 microseconds;
Java version. Size: 128 ;numProcs: 8; iterations: 2030; runtime: 0 seconds, 521612 microseconds;
Java version. Size: 128 ;numProcs: 12; iterations: 1177; runtime: 2 seconds, 2282156 microseconds;
Java version. Size: 128 ;numProcs: 16; iterations: 3721; runtime: 10 seconds, 10854662 microseconds;
Java version. Size: 128 ;numProcs: 20; iterations: 594; runtime: 2 seconds, 2166521 microseconds;
Java version. Size: 128 ;numProcs: 24; iterations: 403; runtime: 2 seconds, 2675136 microseconds;
Java version. Size: 128 ;numProcs: 28; iterations: 363; runtime: 2 seconds, 2033118 microseconds;
Java version. Size: 128 ;numProcs: 32; iterations: 2645; runtime: 19 seconds, 19098342 microseconds;
*******************************\n
```

The “10 seconds” and “19 seconds” are odd data. They might have been affected by the machine’s performance. (It was very busy that time)

When the size of grids going to be large, the increase of workers still does **not** optimize the performance, neither. On the contrary, it cost more time as the number of workers increases. An example of size 1024*1024:



```
*********************************\n
Java version. Size: 1024 ;numProcs: 4; iterations: 2881; runtime: 8 seconds, 8611448 microseconds;
Java version. Size: 1024 ;numProcs: 8; iterations: 2705; runtime: 5 seconds, 5728304 microseconds;
Java version. Size: 1024 ;numProcs: 12; iterations: 3871; runtime: 12 seconds, 12989934 microseconds;
Java version. Size: 1024 ;numProcs: 16; iterations: 5470; runtime: 19 seconds, 19219378 microseconds;
Java version. Size: 1024 ;numProcs: 20; iterations: 7443; runtime: 23 seconds, 23649351 microseconds;
Java version. Size: 1024 ;numProcs: 24; iterations: 9234; runtime: 31 seconds, 31527010 microseconds;
Java version. Size: 1024 ;numProcs: 28; iterations: 9759; runtime: 33 seconds, 33341189 microseconds;
Java version. Size: 1024 ;numProcs: 32; iterations: 11894; runtime: 42 seconds, 42110124 microseconds;
*********************************\n
```

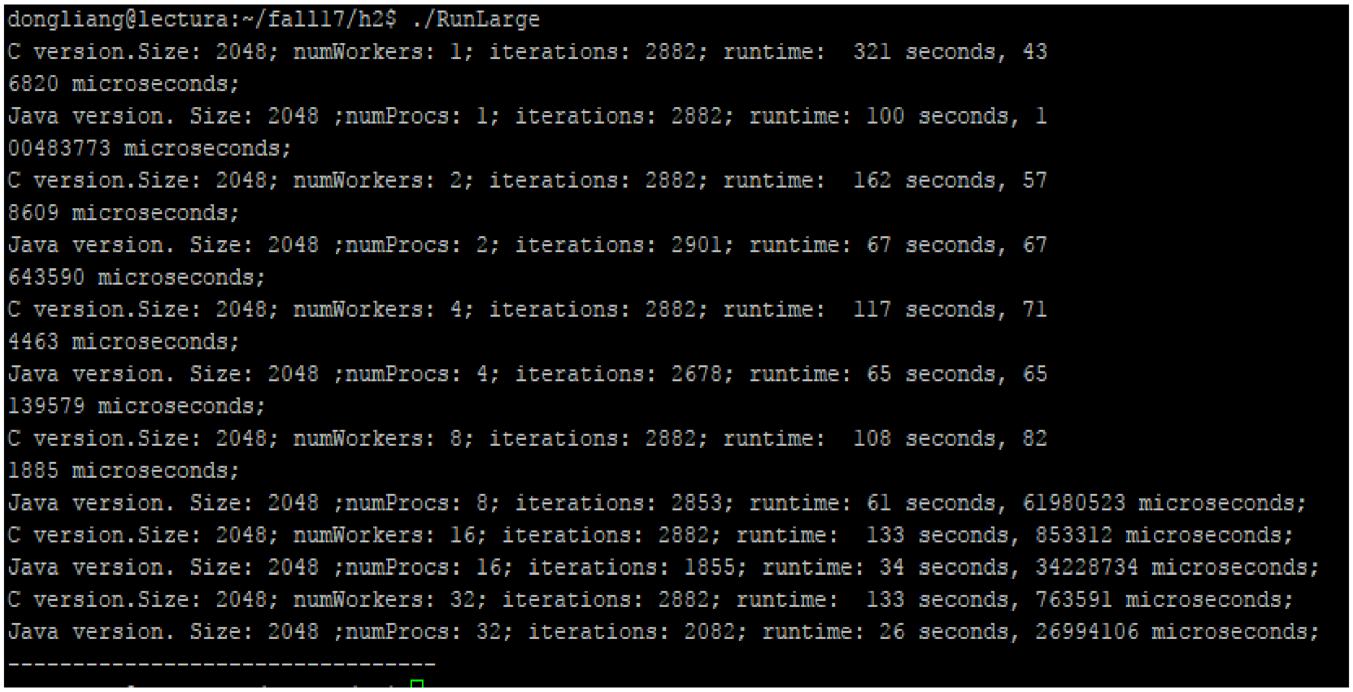
Compare Java and C:

As mentioned before, I use size of 2048*2048 to test both C and Java versions at the same time. This can reduce the error caused by the different machine performance at different time period.

The bash script I used for this is:

```
#! /bin/bash

if [[  ! -f JacobiC  ]]
then
        echo "cannot find the file JacobiC."
        exit 1
fi

        for i in 1 2 4 8 16 32
        do
        ./JacobiC 2048 $i
        java Jacobi 2048 $i
        done

echo "----------------------------------"
```

Result and Conclusion:



```
dongliang@lectura:~/fall17/h2$ ./RunLarge
C version.Size: 2048; numWorkers: 1; iterations: 2882; runtime:  321 seconds, 43
6820 microseconds;
Java version. Size: 2048 ;numProcs: 1; iterations: 2882; runtime: 100 seconds, 1
00483773 microseconds;
C version.Size: 2048; numWorkers: 2; iterations: 2882; runtime:  162 seconds, 57
8609 microseconds;
Java version. Size: 2048 ;numProcs: 2; iterations: 2901; runtime: 67 seconds, 67
643590 microseconds;
C version.Size: 2048; numWorkers: 4; iterations: 2882; runtime:  117 seconds, 71
4463 microseconds;
Java version. Size: 2048 ;numProcs: 4; iterations: 2678; runtime: 65 seconds, 65
139579 microseconds;
C version.Size: 2048; numWorkers: 8; iterations: 2882; runtime:  108 seconds, 82
1885 microseconds;
Java version. Size: 2048 ;numProcs: 8; iterations: 2853; runtime: 61 seconds, 61980523 microseconds;
C version.Size: 2048; numWorkers: 16; iterations: 2882; runtime:  133 seconds, 853312 microseconds;
Java version. Size: 2048 ;numProcs: 16; iterations: 1855; runtime: 34 seconds, 34228734 microseconds;
C version.Size: 2048; numWorkers: 32; iterations: 2882; runtime:  133 seconds, 763591 microseconds;
Java version. Size: 2048 ;numProcs: 32; iterations: 2082; runtime: 26 seconds, 26994106 microseconds;
---------------------------------
```

So, Java beats C when size is 2048*2048. To be honest, I didn’t expect java version could be this fast.