Test the result of the correctness testing process:

Test if the answer is correct:

<u>The input of the test:</u>
The dimension of square array: 6
Square array: random
The number of thread: 2
The precision: 1

1： whether the averaged number is right after each process : "replacing a value with the average of its four neighbors"

Here I only show the first turn for all the threads operated
The initial square is:

86.000000	96.000000	85.000000	72.000000	36.00000	36.000000
91.000000	12.000000	68.000000	67.000000	31.000000	34.000000
98.000000	87.000000	55.000000	25.000000	26.000000	98.000000
59.000000	96.000000	9.000000	67.000000	79.000000	23.000000
3.000000	23.000000	85.000000	42.000000	85.00000	97.000000
77.000000	72.000000	45.000000	63.000000	96.000000	33.000000

The updated array:

86.000000	96.000000	85.000000	72.000000	36.00000	36.000000
91.000000	85.500000	54.750000	49.000000	40.750000	34.000000
98.000000	65.250000	47.250000	53.750000	58.250000	98.000000
59.000000	44.500000	75.750000	38.750000	50.250000	23.000000
3.000000	64.000000	29.750000	75.000000	78.500000	97.000000
77.000000	72.000000	45.000000	63.000000	96.000000	33.000000

After I checked by hand work, it shows the update array is right.

2： whether the program finish its task: "iterate the averaging until the new values of all elements differ from the previous ones by less than the precision."

Here are the last three updated array

Array(n-2):

68.000000	17.000000	57.000000	44.000000	70.000000	59.000000
35.000000	40.468223	51.830467	50.587029	49.857957	11.000000
86.000000	59.762923	56.269102	59.447578	65.992481	94.000000
43.000000	53.309279	58.562491	60.080007	63.466318	50.000000
70.000000	54.722870	59.714731	63.398774	74.779127	96.000000
99.000000	34.000000	65.000000	56.000000	78.000000	35.000000

Array(n-1):

68.000000	17.000000	57.000000	44.000000	70.000000	59.000000
35.000000	40.898348	51.081089	51.284001	49.394878	11.000000
86.000000	59.011651	57.400865	58.232155	66.692963	94.000000
43.000000	54.012071	57.343280	61.218790	62.712904	50.000000
70.000000	54.256002	60.421034	62.643466	75.216273	96.000000
99.000000	34.000000	65.000000	56.000000	78.000000	35.000000

The final answer Array(n):

68.000000	17.000000	57.000000	44.000000	70.000000	59.000000
35.000000	40.523185	51.645803	50.677030	49.744241	11.00000
86.000000	59.577821	56.417044	59.149155	66.084984	94.000000
43.000000	53.402733	58.263190	60.232951	63.282007	50.000000
70.000000	54.608276	59.810687	63.214024	74.839092	96.000000
99.000000	34.000000	65.000000	56.000000	78.000000	35.000000

The array(n-2) have number 56.269102 and corresponding to array(n-1) 's number 57.400865. The different these two number 57.400865 - 56.269102 is greater than the precision 1. Hence the program is not stop until the final answer array(n). As you can see from those arrays shows above, the difference for each number between array(n) and array(n-1) are less than the precision. Hence, the program is stop here.

Therefore, my parallel program got the right answer.

Test if the threads are operating paralleled:

Here I will use 10 threads to operate the square array which the dimension is 50: I am going to print the thread number while they start to work at the same time. If all the threads are operating paralleled, the output thread number should not be sequential.

No: 0	No: 6	No: 2	No: 3	No: 5	No: 7	No: 1	No: 4	No: 8	No: 9
No: 9	No: 6	No: 3	No: 7	No: 4	No: 8	No: 0	No: 2	No: 5	No: 1

As you can see the thread are operated paralleled in my program.

Parallel Programming Analysis:

I am going to test the relationship between the processor, efficiency and speedup. I will make the dimension to be fixed which are 200 and 400, after that I will gradually increase the number of processor to test the time consumed. Finally I plan to use the time consumed to calculate the speedup and the efficiency.
The method to use the time consumed to calculate the speedup is:

$$S_\infty = \frac{\text{time on a sequential processor}}{\text{time on parallel processors}}$$

The efficiency is calculate by using the speedup:

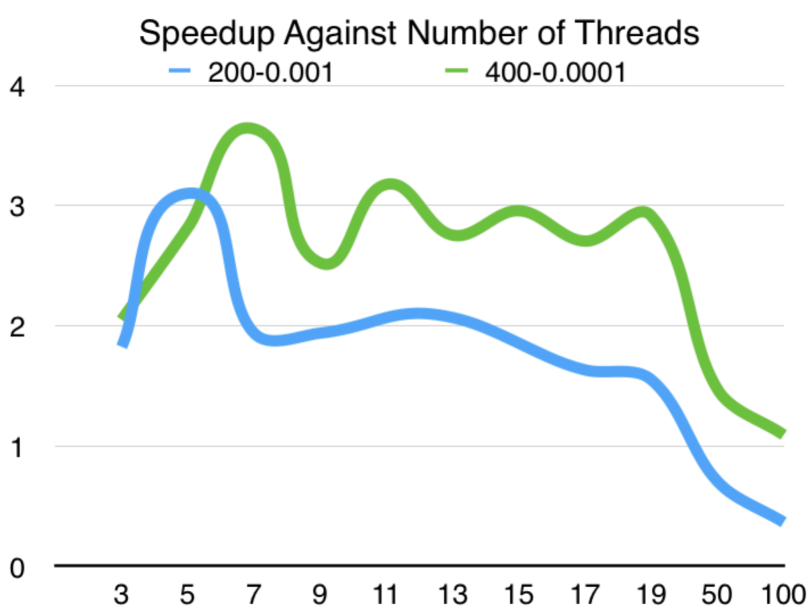
$$E_p = \frac{S_p}{p}$$

<u>The input of test:</u>
Precision:0.001
Dimension:200

Number of Thread	Time Consumed	Number of Thread	Time Consumed
1	00:00:31	13	00:00:15
3	00:00:17	15	00:00:17
5	00:00:10	17	00:00:19
7	00:00:16	19	00:00:20
9	00:00:16	50	00:00:44
11	00:00:15	100	00:01:25

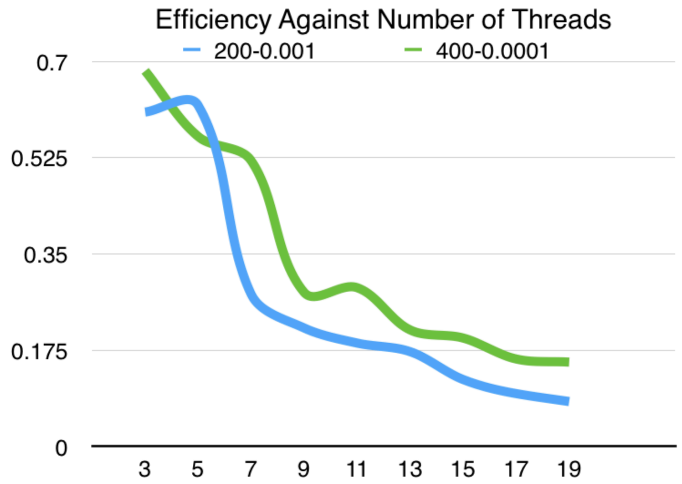
Precision: 0.0001
Dimension: 400

Number of thread	Time Consumed	Number of thread	Time Consumed
1	>00:10:00	13	00:03:38
3	00:04:53	15	00:03:23
5	00:03:33	17	00:03:42
7	00:02:45	19	00:03:26
9	00:03:58	50	00:06:48
11	00:03:09	100	00:09:10



For the second situation, its time on a sequential processor is greater than 10:00, hence, we cannot get the exactly speedup for that situation. Therefore I set the time for a sequential processor is 10:00. Hence, we can just check how the trends going for that situation.

As the graph shows, at the beginning, the speed up is dramatic increased, after then, the speedup is going down as the number of thread is increasing. These indicate that according to the different size of the array, the speedup will be increased if the number of thread is increased appropriately. But if the number of thread is too many for a small size array, this will cause decreasing of the speedup. For example, in the second test, which its dimension is 200, the biggest speedup is when the number of thread is 7, and after then, the speedup is decreasing as the number of thread, increased.



According the efficiency against number of threads graph, we can know that the efficiency is decreasing as the number of threads is increasing.