Test the result of the correctness testing process:

Test if the answer is correct:

The input of the test:
The dimension of square array: 5
Square array: random
The number of processor: 2
The precision: 1

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

In order to check the correctness of my program, I set the initial array as the fix array.

The initial square is:

73.000000	6.000000	38.000000	44.000000	18.000000
4.000000	70.000000	25.000000	86.000000	92.000000
6.000000	32.000000	89.000000	54.000000	57.000000
8.000000	61.000000	46.000000	54.000000	6.000000
34.000000	39.000000	96.000000	11.000000	9.000000

The Final Answer is:

73.000000	6.000000	38.000000	44.000000	18.000000
4.000000	17.836684	36.853775	54.693778	92.000000
6.000000	22.621651	38.708984	44.050224	57.000000
8.000000	29.765206	47.568100	27.622300	6.000000
34.000000	39.000000	96.000000	11.000000	9.000000

After I checked by hand work, it shows the update array is right and every time the program always get the same final answer.

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):

73.000000	6.000000	38.000000	44.000000	18.000000
4.000000	18.316521	36.877075	55.173271	92.000000
6.000000	22.645081	39.667969	44.073669	57.000000
8.000000	30.244698	47.591675	28.101448	6.000000
34.000000	39.000000	96.000000	11.000000	9.000000

Array(n-1):

73.000000	6.000000	38.000000	44.000000	18.000000
4.000000	17.380539	37.789440	54.237686	92.000000
6.000000	23.557297	37.796875	44.985672	57.000000
8.000000	29.309189	48.503529	27.166336	6.000000
34.000000	39.000000	96.000000	11.000000	9.000000

The final answer Array(n):

73.000000	6.000000	38.000000	44.000000	18.000000
4.000000	17.836684	36.853775	54.693778	92.000000
6.000000	22.621651	38.708984	44.050224	57.000000
8.000000	29.765206	47.568100	27.622300	6.000000
34.000000	39.000000	96.000000	11.000000	9.000000

The array(n-2) have number 39.667969 and corresponding to array(n-1) 's number 37.796875. The different these two number 39.667969 -37.796875 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 processors are operating paralleled:

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

Rk: 2	Rk: 6	Rk: 0	Rk: 3	Rk: 7	Rk: 1	Rk: 5	Rk: 8	Rk: 9	Rk: 4
Rk: 7	Rk: 3	Rk: 6	Rk: 9	Rk: 4	Rk: 0	Rk: 5	Rk: 2	Rk: 8	Rk: 1

As you can see the processor 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 and precision to be fixed which are 100 and 0.01, 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:

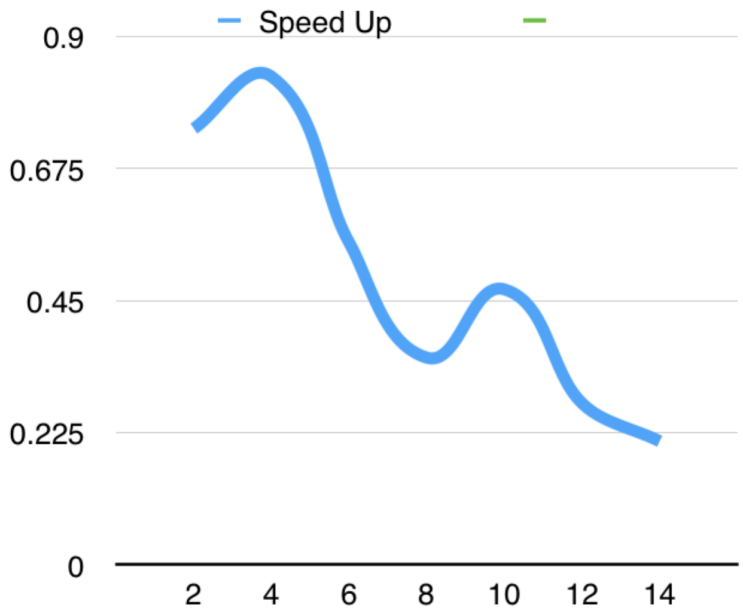
$$S_\infty = \frac{\text{time on a sequential processor}}{\text{time on parallel processors}}$$

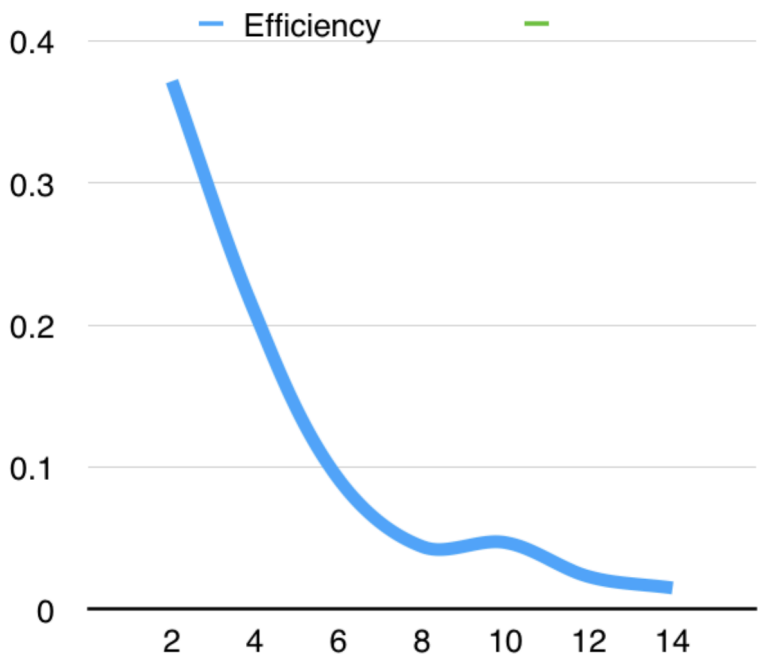
The efficiency is calculate by using the speedup:

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

The input of test:
Precision:0.01
Dimension:100

Number of Processors	Time Consumed	Number of Processors	Time Consumed
1	00:01:24	8	00:03:58
2	00:01:53	10	00:02:59
4	00:01:41	12	00:05:05
6	00:02:33	14	00:06:39



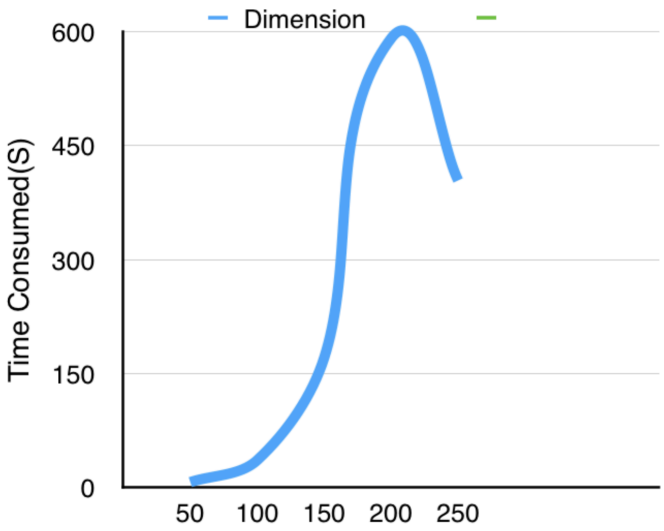


As the graph shows, at the beginning, the speed up is dramatic increased, after then, the speedup is going down as the number of processors are increasing. This indicates that according to the different size of the array, the speedup will be increased if the numbers of processors are increased appropriately. But if the numbers of processors are too many for a small size array, this will cause decreasing of the speedup. For example, in this test, the biggest speedup is when the number of thread is 4, and after then, the speedup is decreasing as the number of thread, increased.

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

Precision: 0.01
Number of Processor: 6

Dimension	Time Consumed
50	00:00:07
100	00:00:36
150	00:02:51
200	00:09:51
250	00:06:44



As we can see that the time consumed is increasing as the number of processor increased from 50 to 200 and decreased as the number of processor from 200 to 250.