Parallel Programming HW2 Mandelbrot Set Report

資工21 周聖諺 106033233

1. Implementation

(1) Pthread Version:

Implement with Thread Pool architecture.

Features

(a) SSE2 Vectorization:

Use SSE2 instruction(128 bits) to vectorize the computation of Mandelbrot Set.

(b) Dynamic chunk size initialization

Partition the task into the same chunk size. The chunk size is depend on the number of CPUs with the rule:

Chunk_Size = ((#CPU ^ 2) >> 1) << 1

if the Chunk Size is <= 1, Chunk Size = 2

(c) Integrate x0, y0 computation, Mandelbrot Set computation, and image computation in one function:

Integrate the computation into one function. It is convenient for programming and also enhances parallelism.

(2) Hybrid Version:

Implement with Master-Slave architecture.

(a) SSE2 Vectorization:

Same as the Pthread version.

(b) Dynamic chunk size initialization

Same as the Pthread version.

(c) Integrate x0, y0 computation, Mandelbrot Set computation, and image computation in one function:

Same as the Pthread version.

(d) Non-blocking MPI_Isend

Non-blocking message-passing for assigning and requesting tasks.

(e) Multi-tasking of master process

One thread for communication to the slaves and the remaining processes compute the Mandelbrot Set in parallel.

(f) Guided chunk size of processes(Fail)

Decrease the chunk-size through time. But always have bugs.

(g) Multithreaded master service(Fail)

It means that having multiple threads to assign tasks to slaves.

Although I've enable the multi-threading support for MPI with *MPI_Init_thread(&argc, &argv, MPI_THREAD_MULTIPLE, &provided)* , it always throw the system error that it cannot lock and unlock mutex lock correctly. As a result, I gave it up.

2. Experiment & Analysis

i、 Methodology

(a). **System Spec**

Appolo Cluster

(b). **Performance Metrics**

Count the execution time in seconds of each part of the program.

In Pthread Version, the I use the following parameters as testcase:

```
1000 -1 1 -1 1 1600 1600
```

In Hybrid Version, the I use the following parameters as testcase:

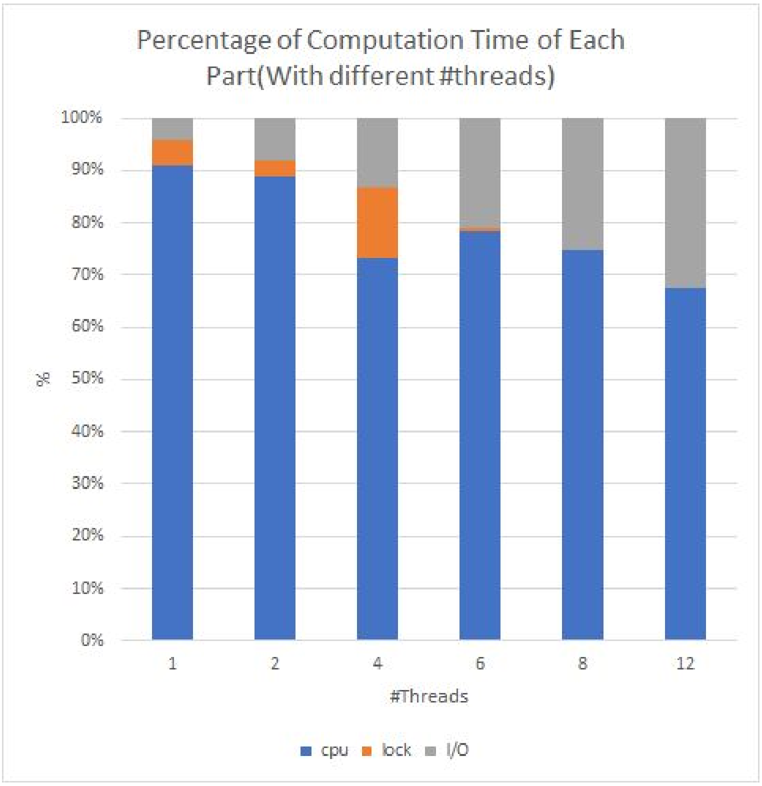
```
174170376 -0.7894722222222222 -0.7825277777777778
0.145046875 0.148953125 2549 1439
```

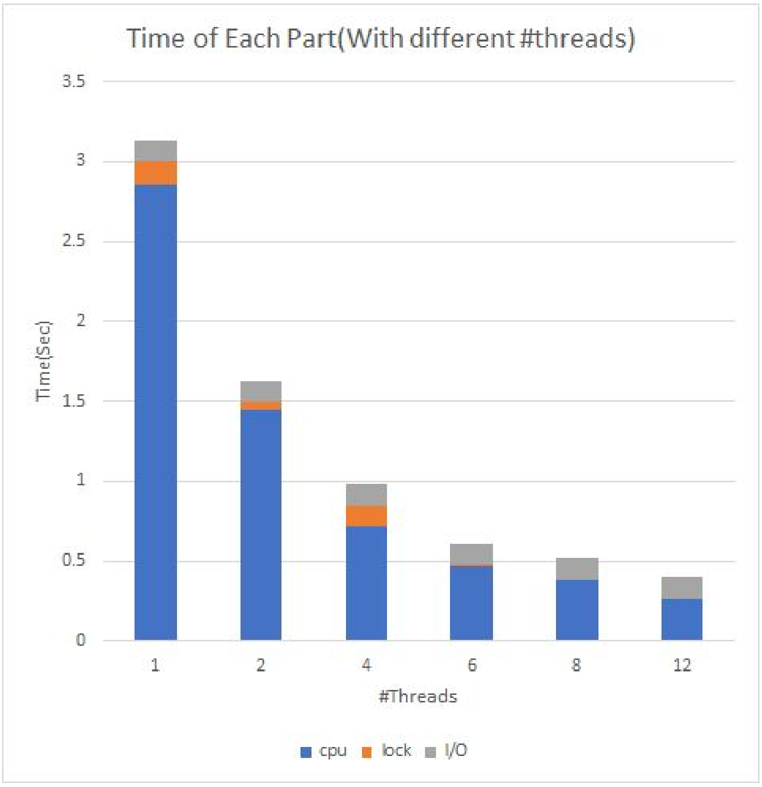
(The format of the parameters is the same as the given parameters of spec)

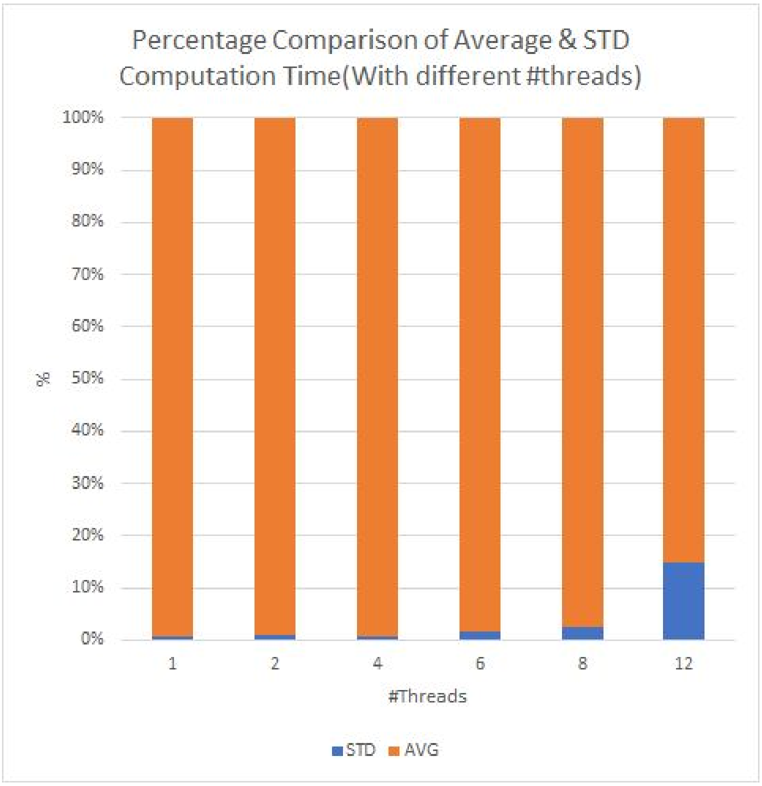
ii、 Plots: Scalability & Load Balancing

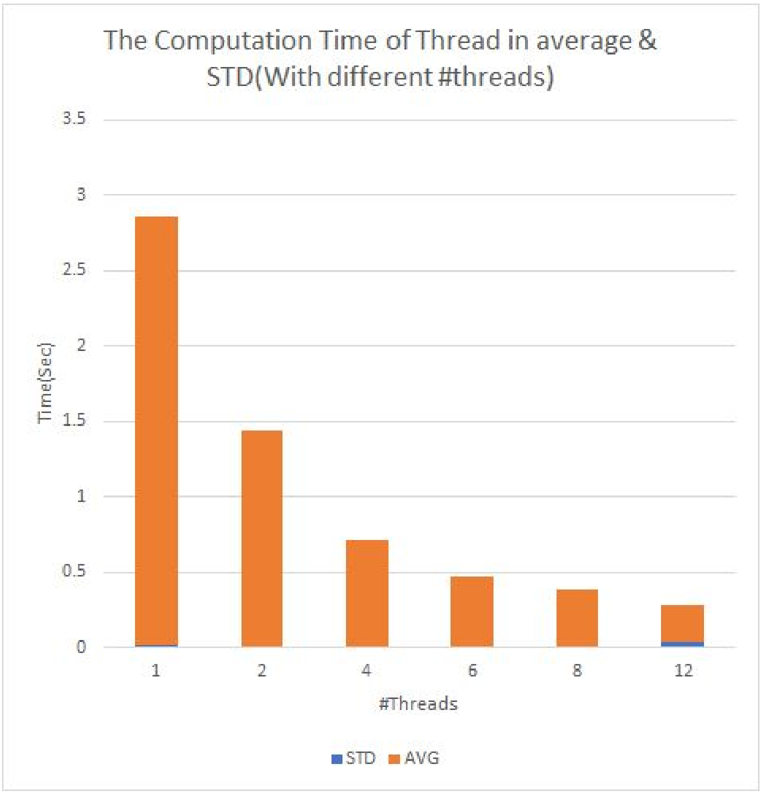
Note: #thread means the number of threads and #processes means the number of processes

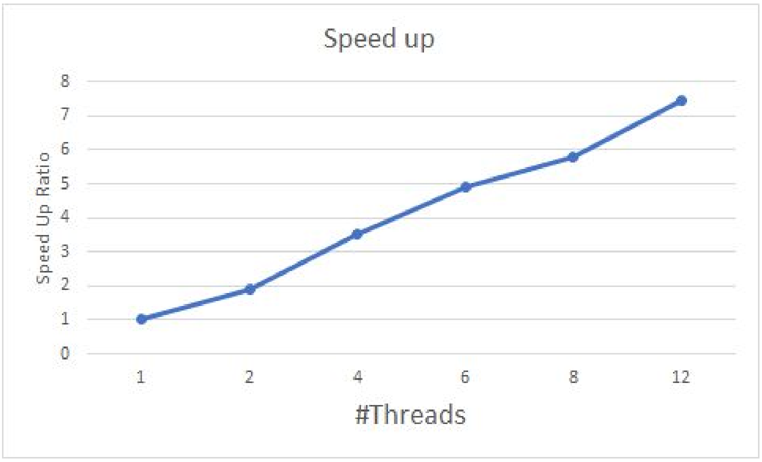
(a) Pthread Version


Percentage of Computation Time of Each Part(With different #threads)
%
100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%
1
2
4
6
8
12
#Threads
cpu
lock
I/O


Time of Each Part(With different #threads)
Time(Sec)
3.5
3
2.5
2
1.5
1
0.5
0
1
2
4
6
8
12
#Threads
cpu
lock
I/O

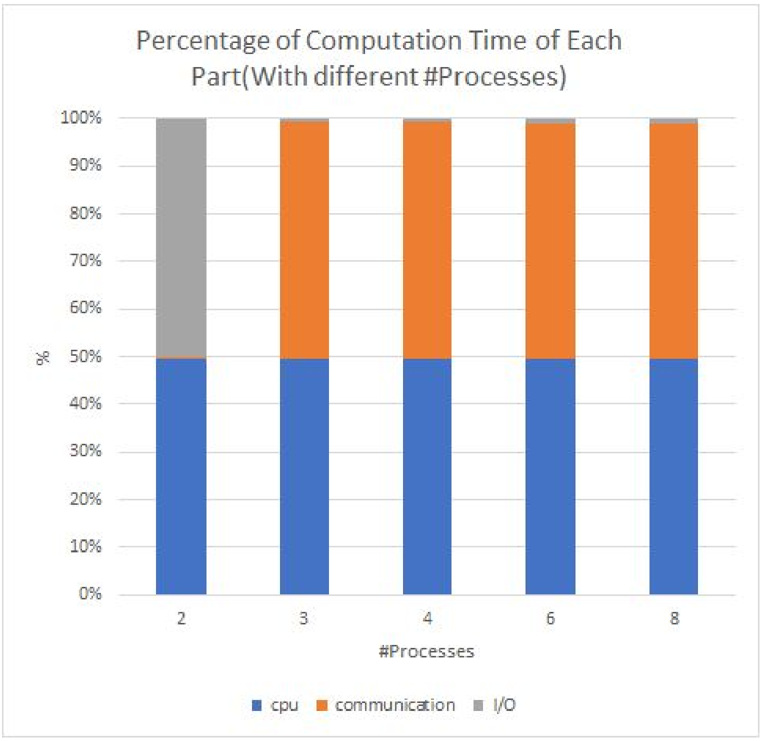

Percentage Comparison of Average & STD Computation Time(With different #threads)
100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%
%
1
2
4
6
8
12
#Threads
STD
AVG

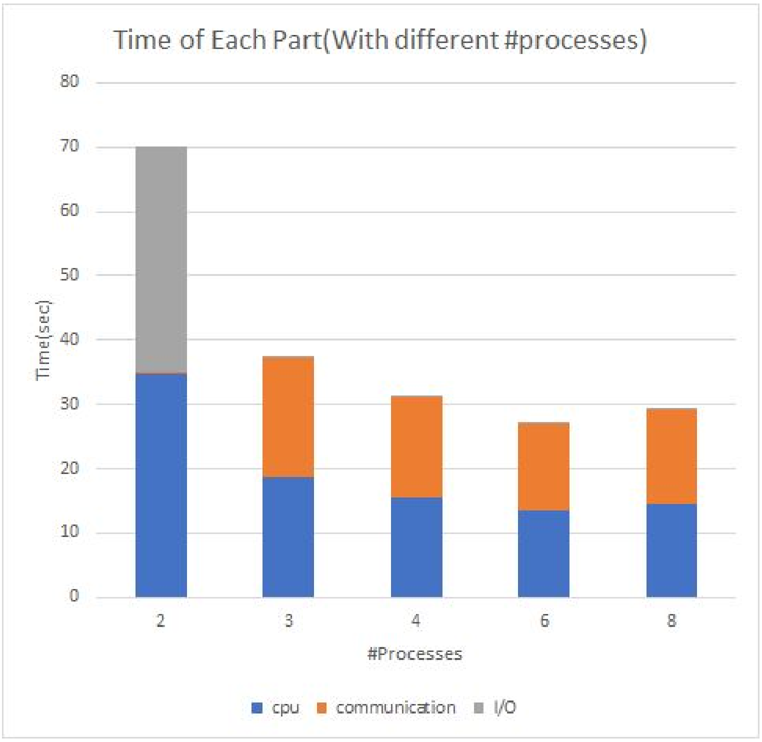

The Computation Time of Thread in average & STD(With different #threads)
3.5
3
2.5
2
1.5
1
0.5
0
Time(Sec)
1
2
4
6
8
12
#Threads
STD
AVG

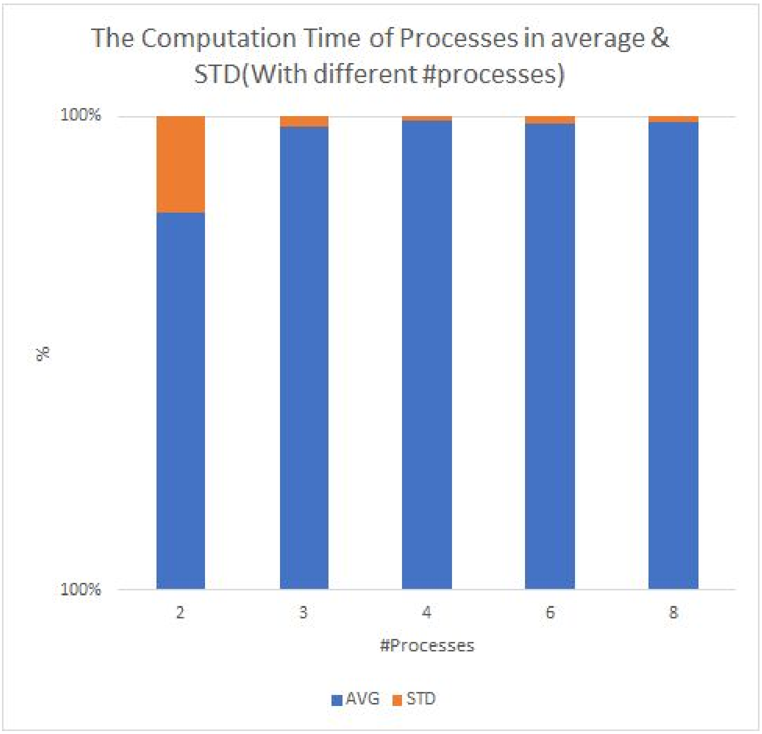


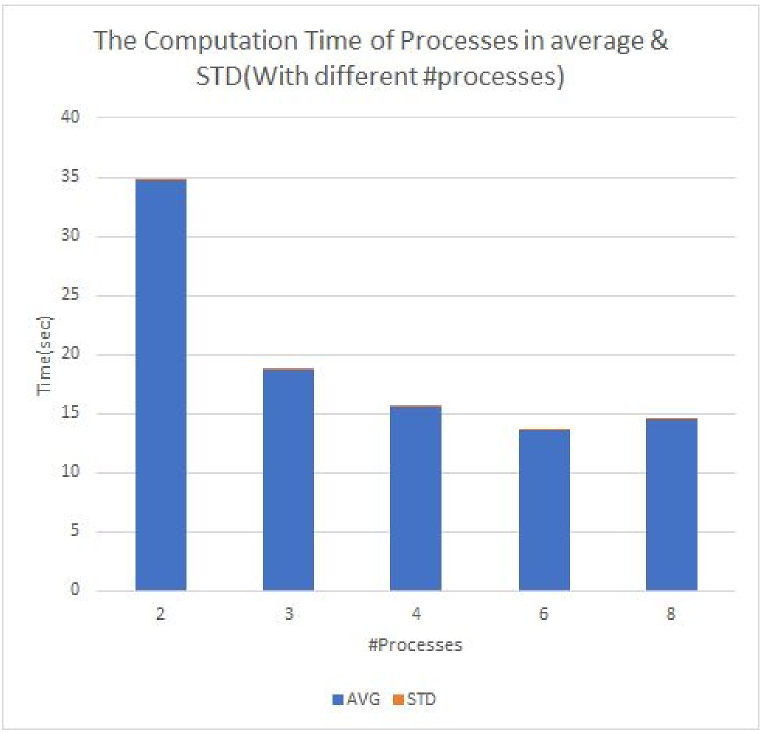
(b) Hybrid Version

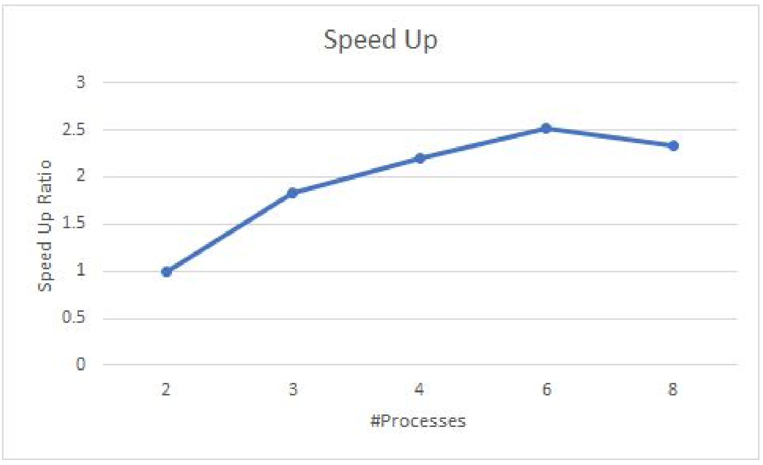
Note: The times of communication and CPU overlap. That is, the master process sends and receives the message and computes the **Mandelbrot Set at the same time.**




Time of Each Part(With different #processes)
Time(sec)
80
70
60
50
40
30
20
10
0
2
3
4
6
8
#Processes
cpu
communication
I/O


The Computation Time of Processes in average & STD(With different #processes)
%
100%
100%
2
3
4
6
8
#Processes
AVG
STD





iii、Discussion

(a) Pthread Version

For **Scalability**, It seems to scale well. The cost of waiting for Mutex lock is often under 10% of the total execution time. The speed up ratio is close to the number of threads.

As for **Load Balancing, according to the comparison between average(AVG) and standard deviation(STD),** the deviation is quite small(only 8 threads version > 10%, the others are < 5% of the average execution time). It should balance the task properly.

(b) Hybrid Version

For **Scalability**, It seems weird that it doesn't scale well after the number of the processes is larger than 6.

However, as for **Load Balancing,** it performs quite well. The standard deviation often < 5%. The standard deviation is > 10% only under 2 processes.

I guess the main reason for the bad scalability is high communication cost due to the increasing processes. Although I've increased the initial task number and use non-blocking calls. I still get stuck while passing messages. I've thought that I should cache the next task before the slave requests a new task to avoid the waiting. But due to lack of time, i didn't do that. Another way that I've tried is to increase the number of the service(that is having multiple threads to assign tasks to slaves) thread of the master process. But it seems to have a bug.

iv、 Others

- You are strongly encouraged to conduct more experiments and analysis of your implementations.

3. Experience & Conclusion

The most difficult part of the assignment is optimization. I am used to optimizing a program with only guessing, not profiling. As a result, it takes me lots of time to fix the wrong optimization. It is obvious that the hybrid version has some issues to fix, although it passed the judge and got a not bad rank. I think a better way to reduce the time of try-and-error for optimization is profiling before coding.