Parallel Programming Homework 2

Mandelbrot Set

CS 107062546 Chung-Kai Yang 楊仲愷

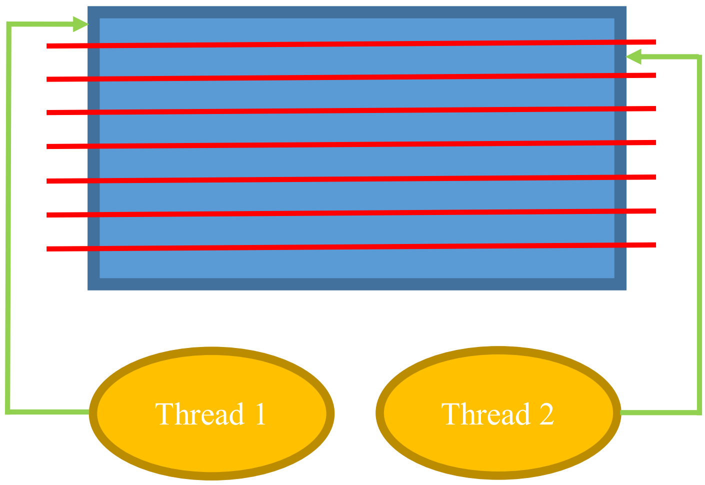
1. Implementation

(1) How you implement each of requested versions, especially for the hybrid parallelism.

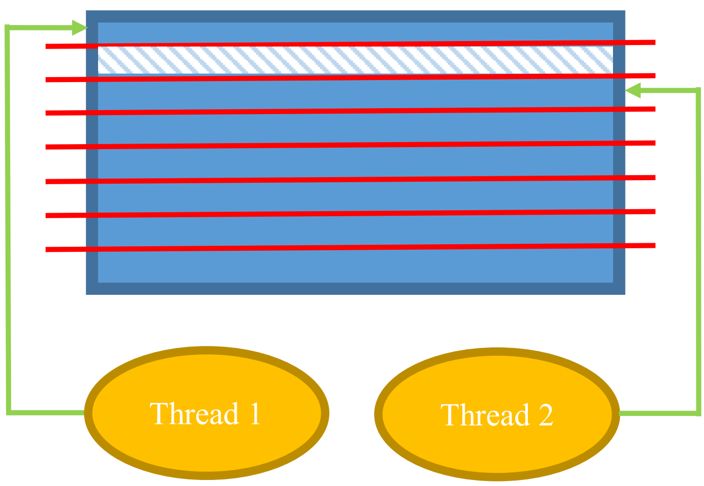
a. Pthread

As we can see in the implementation of Mandelbrot Set, I take advantage of dynamically calculate height (i.e., divide with row) with pthread.

Step 1:

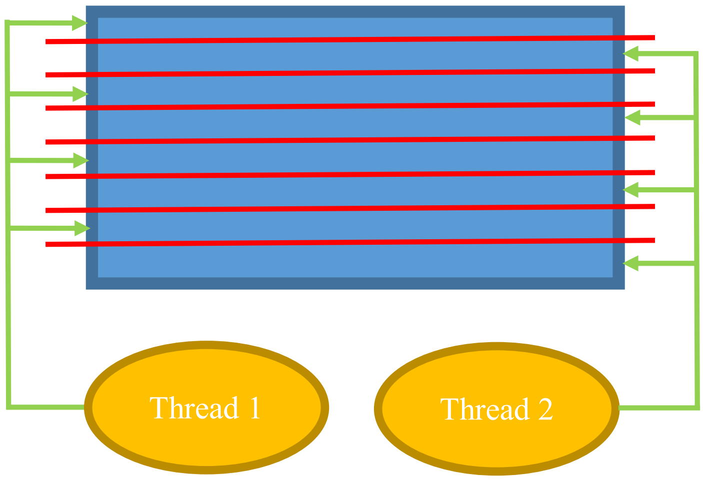


Step 2 (when thread 2 finish):



b. Hybrid (MPI + OpenMP)

In this section, I utilize the same concept to fetch the result of Mandelbrot Set. Here, I have tried static (i.e., distribute fixed chunk size before executing) and dynamic (i.e., when each thread finish, get the next one with round robin scheduling strategy) version.



(2) How do you partition the task?

a. Phtread

In this version, as I describe before, I divide the image by rows. And each thread will get one to compute; when each one finish, it will get the next one until all the rows are done.

b. Hybrid (MPI + OpenMP)

The partition in this section is a little different. I distribute the image into rows, and each rank get a part of them (e.g., the height is 7, and the rank size is 3; thus, rank 0 will get height 0, 3, 6; rank 1 will get 1, 4, 7; rank 2 will get 2, 5.) because of the probability for the amount of computation.

(3) What technique do you use to reduce execution time and increase scalability?

a. Pthread

In this part, I have tried some other versions, like partition image into pixels, or into columns. However, it seems partition into rows is the best choice. Furthermore, I also have tried partition statically, but it doesn’t perform well. Therefore, I partition image into rows and dynamically obtain the next one to compute to increase scalability.

b. Hybrid (MPI + OpenMP)

In this part, I have tried some other versions as well, like partition image into pixels or into columns. However, it seems partition into rows is the best choice. Furthermore, I have tried some improvements to accelerate. First, I change the for loop step into rank size. Second, I replace and decrease the number of branches. And this performs well.

(4) Other efforts you made in your program.

As mentioned before, I have tried some other versions as well, like partition image into pixels or into columns. Also, I change the for loop step into rank size. And I replace and decrease the number of branches, etc.

2. Experiment & Analysis

i. Methodology

(a) System Spec

Run on cluster.

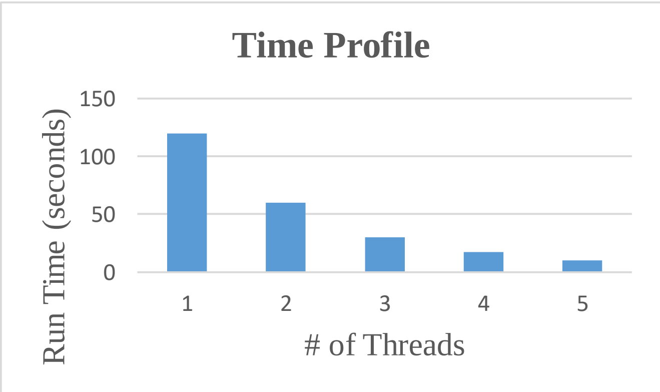
(b) Performance Metrics

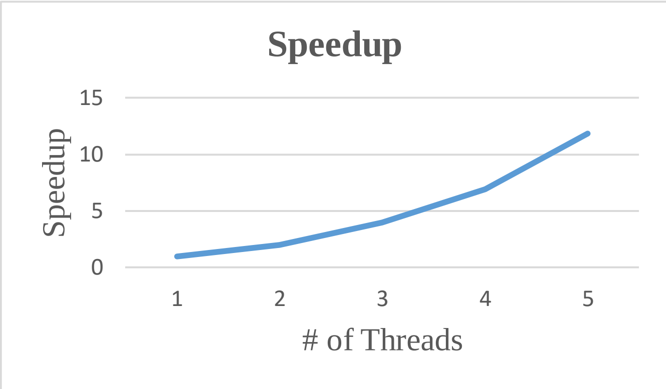
(1) Computing time: Time for accessing CPU. (e.g., calculate Mandelbrot Set)

ii. Plots: Scalability & Load Balancing

(a) Pthread

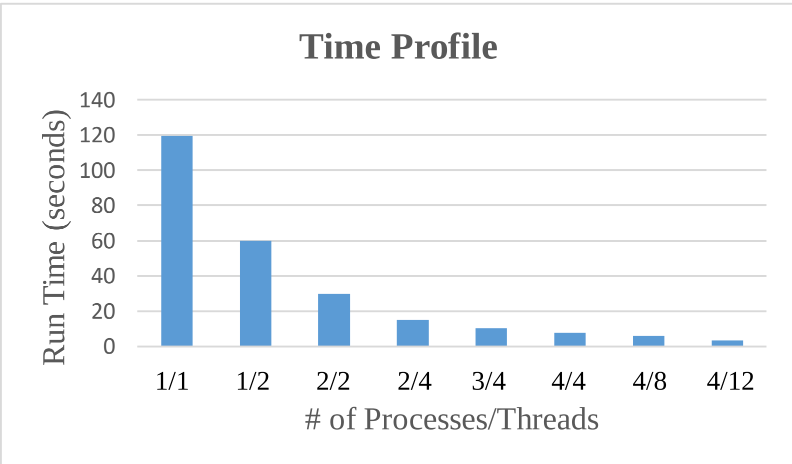
	CPU	Total Time	Speedup
1	119.682365	**119.6824**	**1**
2	59.865865	**59.86587**	**1.999175**
4	30	**29.98975**	**3.990775**
8	17.245502	**17.2455**	**6.939918**
12	10.122554	**10.12255**	**11.82334**

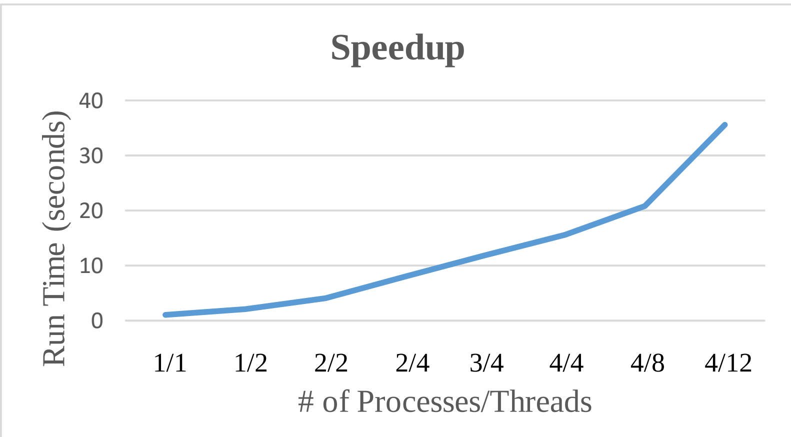




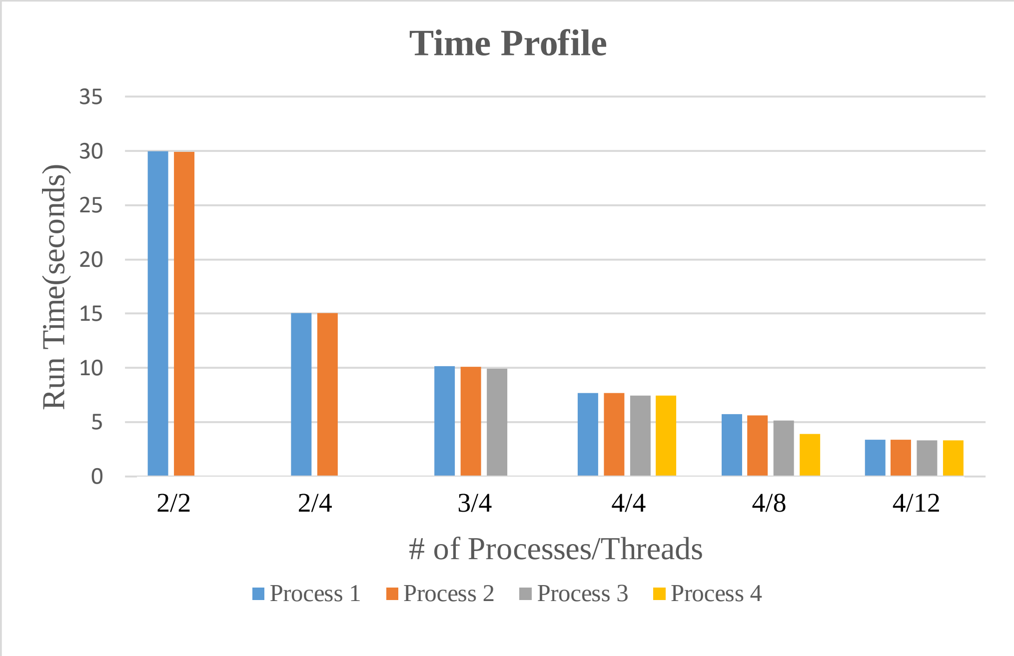
(b) Hybrid (MPI + OpenMP)

Process	Thread	CPU	Total Time	Speedup
1	1	119.6905	**119.6905**	**1**
1	2	59.97039	**59.97039**	**1.995826**
2	2	30	**29.98946**	**3.991086**
2	4	15.04867	**15.04867**	**7.95356**
3	4	10.13889	**10.13889**	**11.80509**
4	4	7.690905	**7.690905**	**15.5626**
4	8	5.750069	**5.750069**	**20.81549**
4	12	3.360959	**3.360959**	**35.61201**

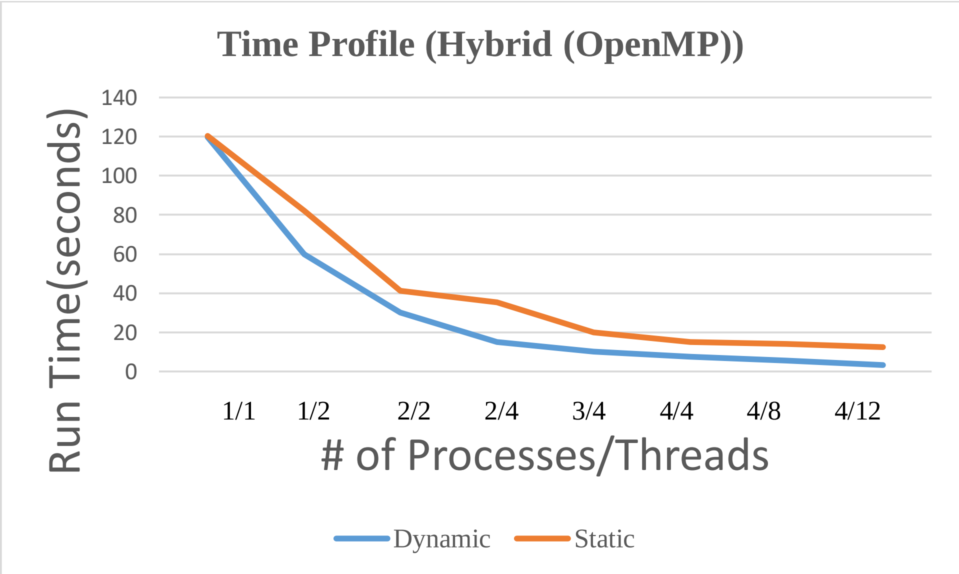


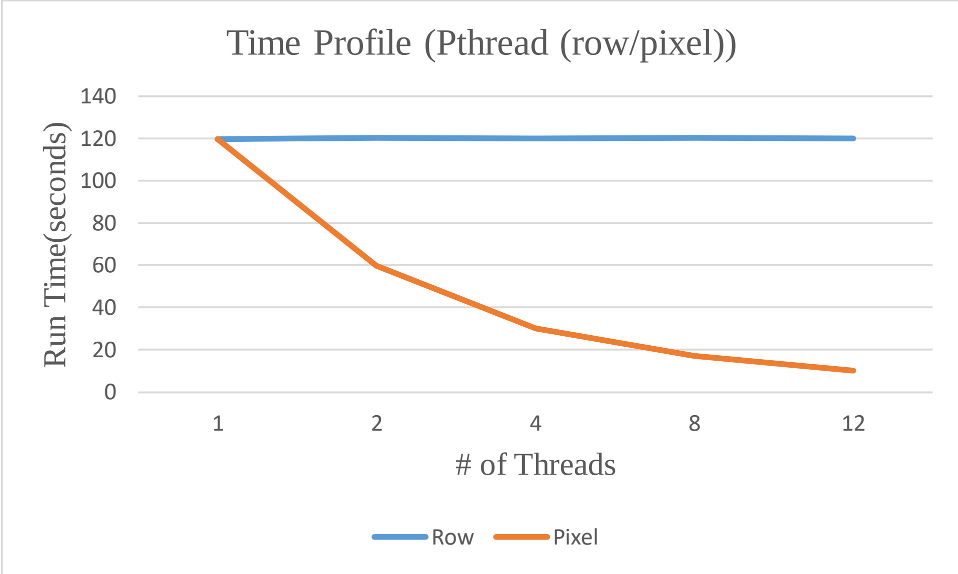


(c) Load Balance



(d) Other Experiment





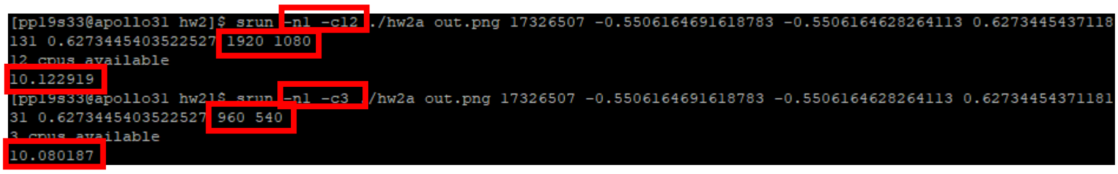
3. Discussion

(a) Scalability:

(1) Strong Scaling: No matter Pthread or Hybrid version, according to figures mentioned before, as the number of processes increasing, the total amount of computation holds; therefore, the calculation quantity each processor has to process will decrease, and the computing time will also lower with parallel computing.

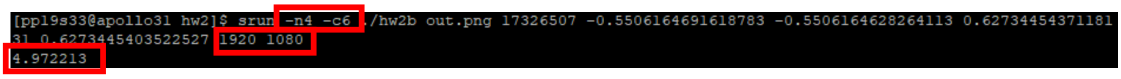
(2) Weak Scaling: I take the example below (i.e., case *slow04*), and I test different image size (i.e., 1920×1080, 960×540). First, I lower 2 times than the height and width before; then, the total computing quantity 4 times, and the time it takes is almost the same.

Pthread:

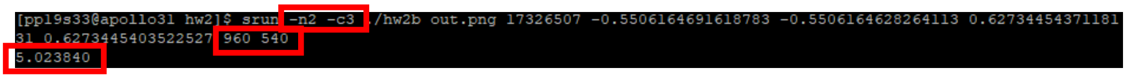


```
[pp19s33@apollo31 hw2]$ srun -n1 -c12 ./hw2a out.png 17326507 -0.5506164691618783 -0.5506164628264113 0.6273445437118
131 0.6273445403522527 1920 1080
12 cpus available
10.122919
[pp19s33@apollo31 hw2]$ srun -n1 -c3 ./hw2a out.png 17326507 -0.5506164691618783 -0.5506164628264113 0.62734454371181
31 0.6273445403522527 960 540
3 cpus available
10.080187
```

Hybrid:



```
[pp19s33@apollo31 hw2]$ srun -n4 -c6 ./hw2b out.png 17326507 -0.5506164691618783 -0.5506164628264113 0.62734454371181
31 0.6273445403522527 1920 1080
4.972213
```



```
[pp19s33@apollo31 hw2]$ srun -n2 -c3 ./hw2b out.png 17326507 -0.5506164691618783 -0.5506164628264113 0.62734454371181
31 0.6273445403522527 960 540
5.023840
```

(b) Load Balance

(1) Pthread: As I mentioned before, I distribute every thread to calculate each row. Then, when each thread finish, it will fetch the next, and it is like a dynamic version.

(2) **Hybrid:** Because the computation quantity of each row is not uniform, I take the same concept as pthread version, I divide the image by rows. And each rank will only get a part of them (e.g., the height is 7, and the rank size is 3; thus, rank 0 will get height 0, 3, 6; rank 1 will get 1, 4, 7; rank 2 will get 2, 5.).

(c) Other Experiment

From the first figure, it shows the dynamic version is more efficient than static version, and I think maybe static version doesn't distribute uniformly enough to cause the result

From the second figure, it shows if I divide by pixels it may not lower the time because when a thread finish, it fetches the next pixel to calculate; however, it happens too frequently to lock the pixel and other threads are always waiting for release the mutex.

4. Conclusion

From this homework, I am getting more familiar with pthread and hybrid (MPI + OpenMP) version. Therefore, according to every result, I try different ways to improve the latency. And I also try dynamic and static or chunk size. Also, I learn the importance of load balance. In spite of the fact that I take advantage of parallel computing, I still can't lower the time.