Homework 3 / Mandelbrot set

李豪韋 (HW-Lee) ID 103061527

Overview

Assumed that there is an infinite sequence $Z(c) = \{z_n | n \in \mathbb{N}\}$ that can be generated by a complex number c with a recursive mapping $z_{n+1} = z_n^2 + c$ where $z_1 = c$, the Mandelbrot set is a set of complex numbers such that all elements in $Z(c)$ do not diverge to infinity. Mathematically, the Mandelbrot set can be defined as $\mathbb{M} = \{c | \lim_{n\to\infty} \|z_n(c)\| < \infty, c \in \mathbb{C}\}$, and it can be derived into the further form that we can obtain it more easily:

$$\mathbb{M} = \{c | \lim_{n\to\infty} \|z_n(c)\| < 2, c \in \mathbb{C}\}$$

To implement a program that computes the Mandelbrot set in an numerical way, the complex space must be bounded by a specific rectangular and the sampling resolution must be designated. Moreover, we cannot check if the sequence remains a small enough norm with infinity, the maximal number of iterations must be defined as well. In this way, the program will be able to obtain the Mandelbrot set by sampling points in the region and determining if the point belongs to the Mandelbrot set if $\|z(c)_{MAX_ITER}\| < 2$. For the convenience of discussing it in more detailed way, each point will be coloured with its iterations into gray-scaled. After all, the output image looks like below:

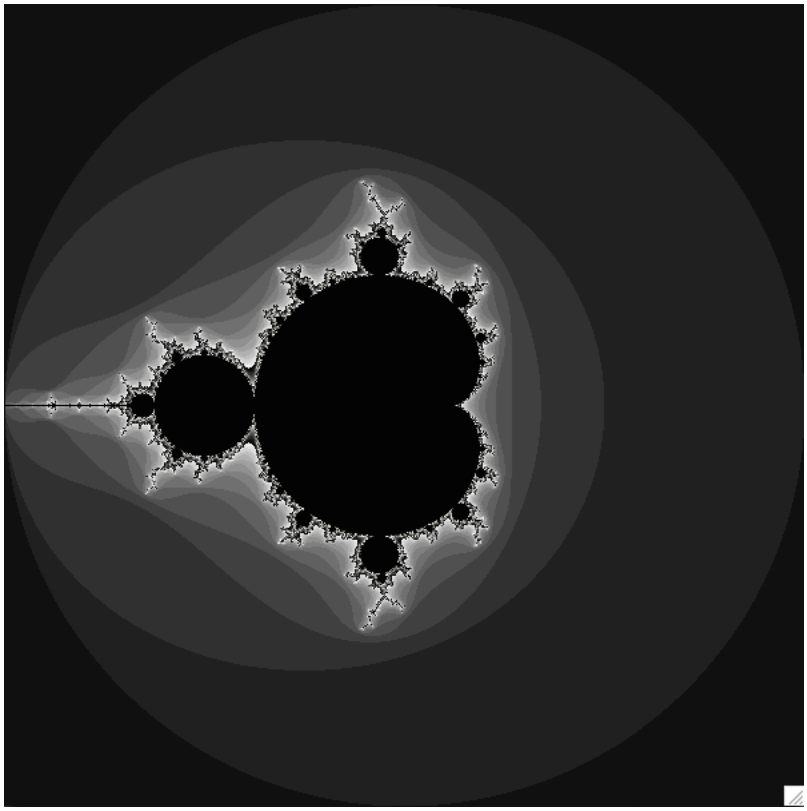


Figure 1: An example showing the Mandelbrot set where the points are ranged from $(-2, -2)$ to $(2, 2)$. Note that the brightness does not directly refer to the magnitude, the raw iterations are processed with modulo function to pronounce the difference between each pair of adjacent region. Therefore, we can observe the 'transition' between adjacent regions easily. However, I think this coloring method is not such good and the coloring methods I use will be discussed in the following section.

Obviously, it is important to decide the strategy separating jobs. Because the number of iterations of the point seems to be subjected to the distance from the origin rather than the position, simply deviding the data into a couple of equally large subsets might induce a bad speedup performance due to the unbalanced loads among workers. Therefore, the project is aimed at making us get familiar with APIs and analyze/fine tune a better scheduling strategy.

Implementation

In the project, the program is implemented with three different APIs, MPI/OpenMP/Hybrid, and two different scheduling strategies, static/dynamic.

- Static scheduling: each worker processes same number of columns.

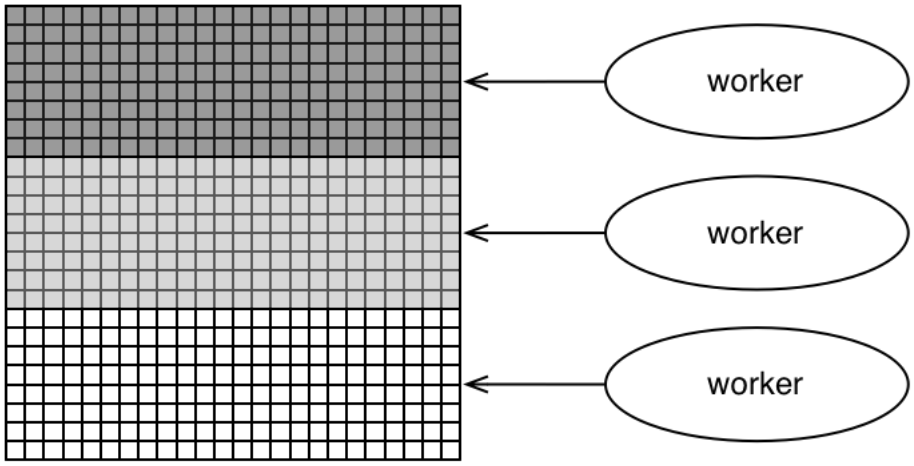


Figure 2: Static scheduling: the whole data will be equally separated into several pieces and each work processes its own piece.

- Dynamic scheduling: each column is assigned dynamically to the worker which completes its task.

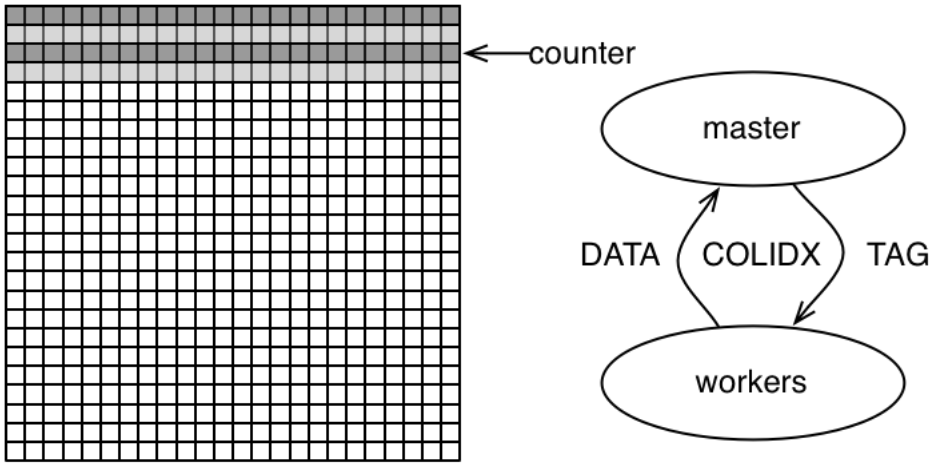


Figure 3: Dynamic scheduling: Only a small batch of columns are assigned to specific workers at the begining, and there is a master that monitors if each worker completes its job and then sends the next job and increments the counter until there is no job remains in the queue. This way provides a more flexible assigning strategy that keeps all workers work during the program.

In Hybrid implementations, they follow the same jobs partition strategies as MPI implementations do, and OpenMP is applied when processing a column by dynamically assigning each point in the column to threads.

Results

- Strong Scabilily

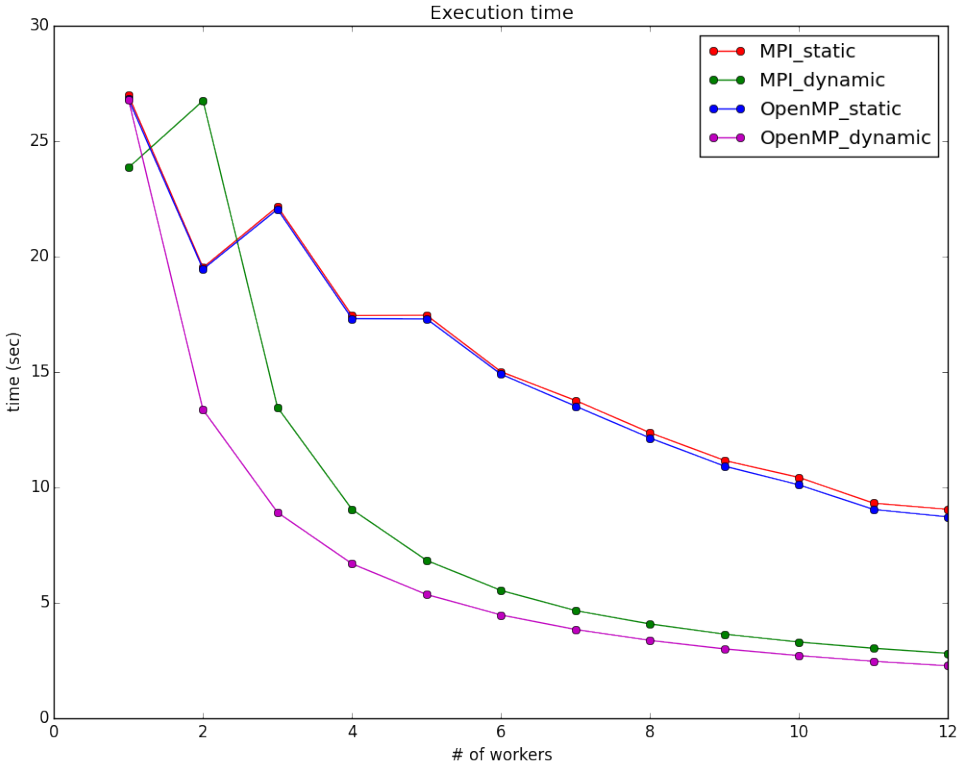


Figure 4: Strong scability: static implementations show roughly linear decay and dynamic ones show roughly reverse-propotional decay as the number of workers increases

- Weak Scability

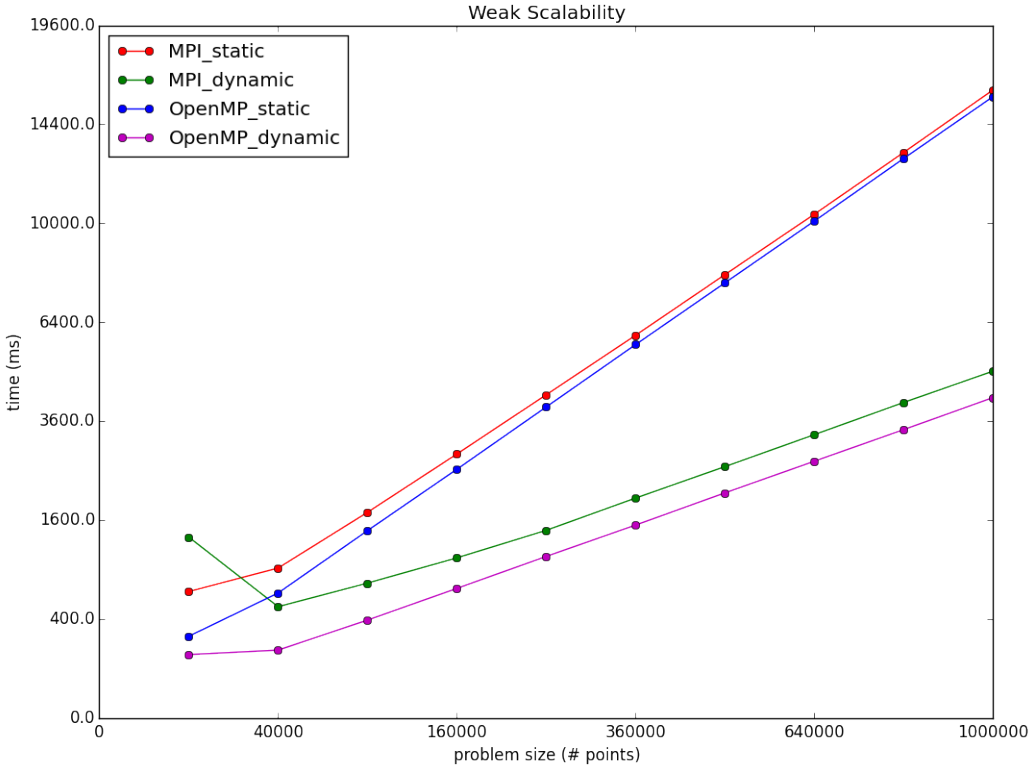
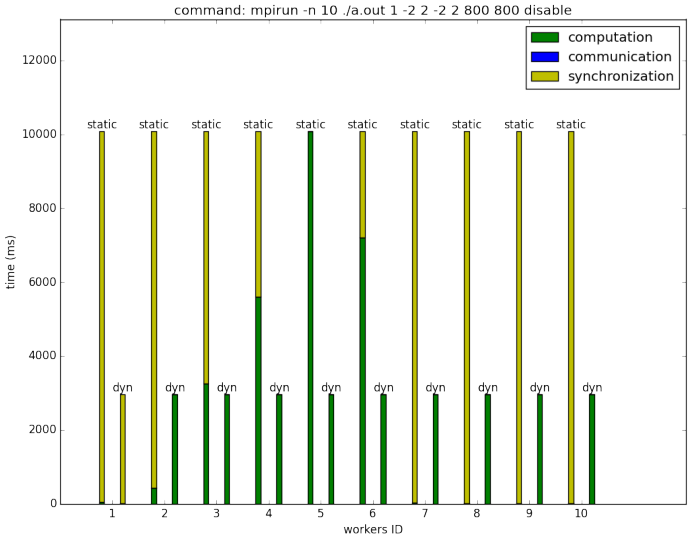
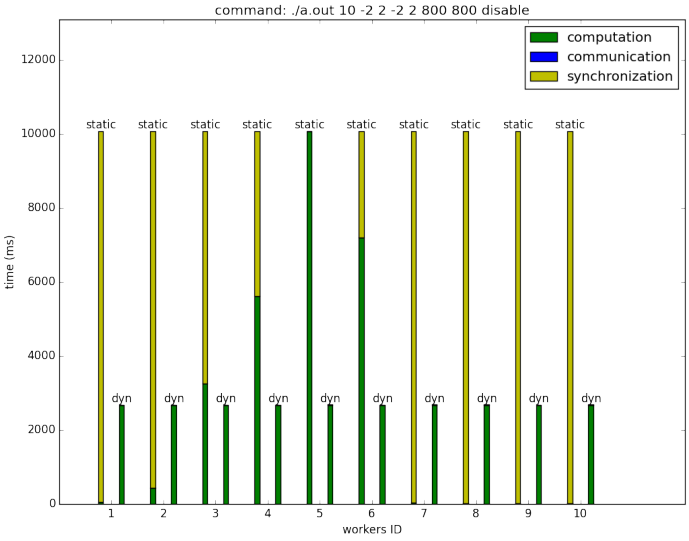


Figure 5: Weak scability: all implementations show roughly linear rise as the number of processed points increases

- Cost of each worker





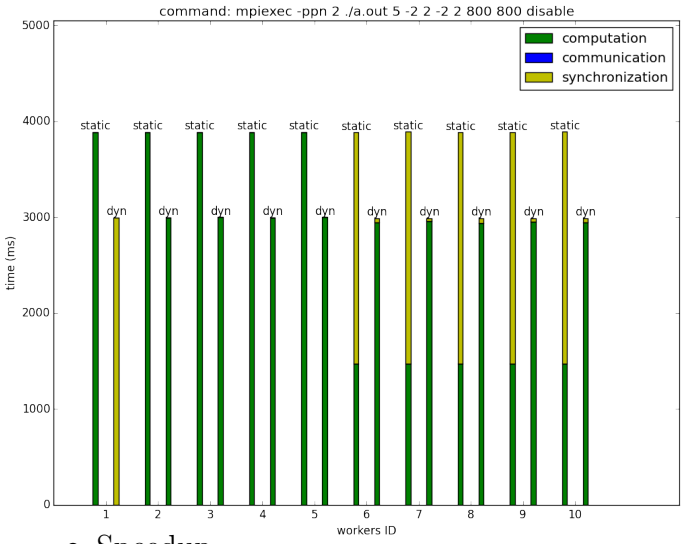


Figure 6: Loads of each worker: MPI(left-upper), OpenMP(right-upper), and Hybrid(left-lower). Execution time is highly dominated by the worker who processes the toughest part of data in static implementations, while it is equally distributed to workers such that the synchronization is very low in dynamic implementations. Note that worker0 always waits for others in both MPI and Hybrid implementation because the worker is assigned to be the master that manages scheduling and handles the parallel jobs.

- Speedup

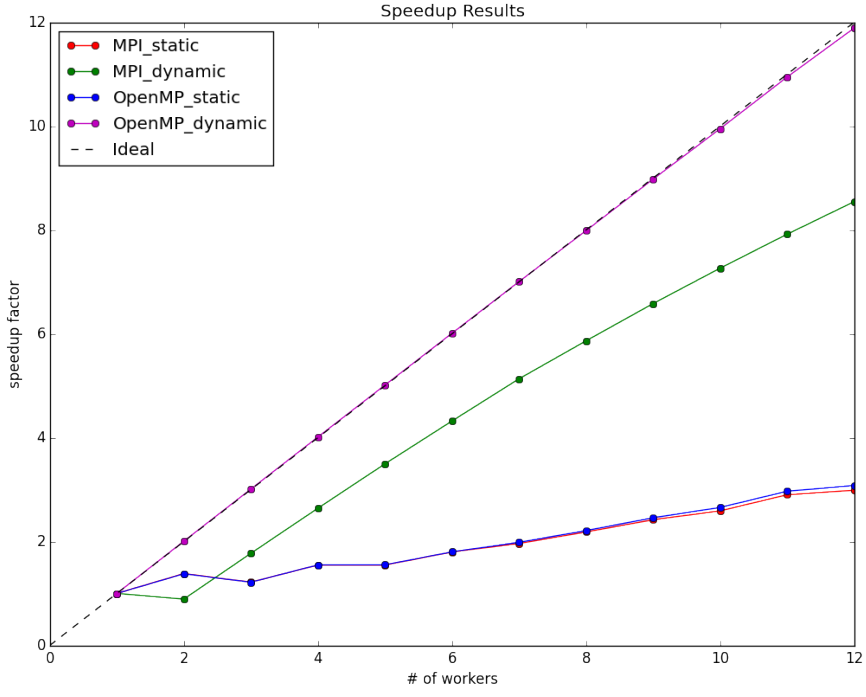


Figure 7: Speedup factors

Analysis & Discussion

- Computation complexity

Assumed that there are N_{pt} points need to be computed by N_{wks} workers with a maximal number of iterations N_{mxit}, and then the complexity can be obtained as below:

Type / Implementation	Comm.	Comp.	Overall	Prop. to
MPI	$O\left(N_{pt}\right)$	$O\left(\frac{N_{pt}N_{mxit}}{N_{wks}-1}\right)$	$O\left(\frac{N_{pt}N_{mxit}}{N_{wks}-1}\right)$	$\frac{1}{N_{wks}-1}$
OpenMP	N/A	$O\left(\frac{N_{pt}N_{mxit}}{N_{wks}}\right)$	$O\left(\frac{N_{pt}N_{mxit}}{N_{wks}}\right)$	$\frac{1}{N_{wks}}$
Hybrid	$O\left(N_{pt}\right)$	$O\left(\frac{N_{pt}N_{mxit}}{N_{wks}-1}\right)$	$O\left(\frac{N_{pt}N_{mxit}}{N_{wks}-1}\right)$	$\frac{1}{N_{wks}-1}$

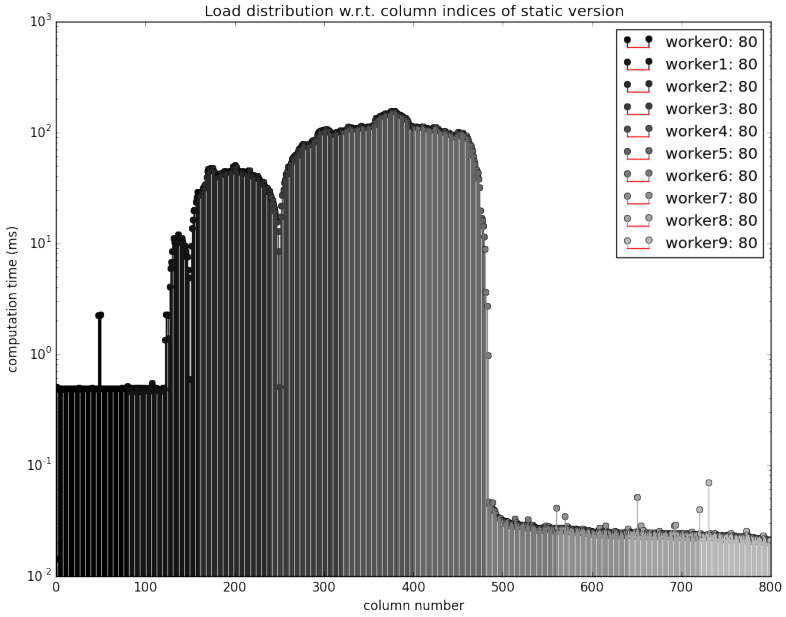
In MPI/Hybrid implementations, communication only used for gathering computed results and the complexity, therefore, can be estimated as $O(N_{pt})$, which means it is propotional to the problem size. In fact, it should be $O(N_{pt} + \sqrt{N_{pt}})$ in dynamic implemenations because the master must send $\sqrt{N_{pt}}$ indices of columns to slaves during the operations. However, it is much smaller than computation loadings so communication consumptions can be ignored. In computation stage, the complexity should be propotional to N_{pt}, N_{mxit}, and $\frac{1}{N_{wks}-1}$ (because one worker must be the master managing working process dynamically) then it is noted as $O(\frac{N_{pt}N_{mxit}}{N_{wks}-1})$. After all, the observed results fit the expectation shown above then I can conclude that the implemenations work well and reasonably.

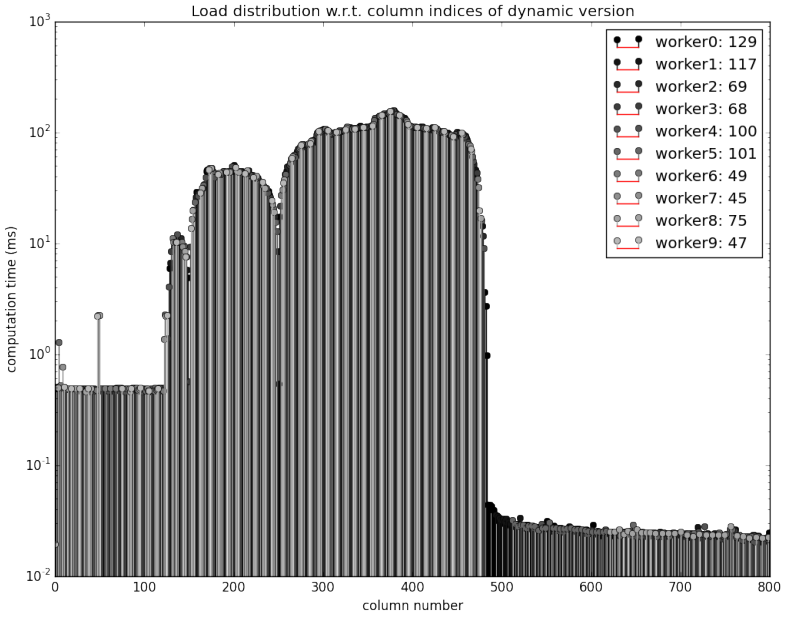
- Speedup results

According to Figure 7, it shows dynamic implementations are superior to static ones. Both dynamic implementations yield good results: which grow linearly as the number of workers increases, and the performance of dynamic MPI saturates gradually at high number of workers, the fact might be caused by communication.
In fact, all implementations show the results decaying linearly as the number of workers increases. However, execution time of static implementations is always dominated by the toughest regions which require lots of iterations to check if each point belongs to the Mandelbrot set. Therefore, the efficiency of parallelizing is not significant.

- Work scheduling

The figures shown below illustrate how the works have been partitioned, and the numbers at the end of labels refer to the number of columns that are assigned to the corresponded worker. In such visualization, the fact that equally separating tasks into several parts is not a good strategy because the loadings of workers are not balanced. To pronounce the correctness of this conclusion, the computation time is visualized with log-scale so that how unbalanced the works are can be easily observed.





- Performance with various resource configurations

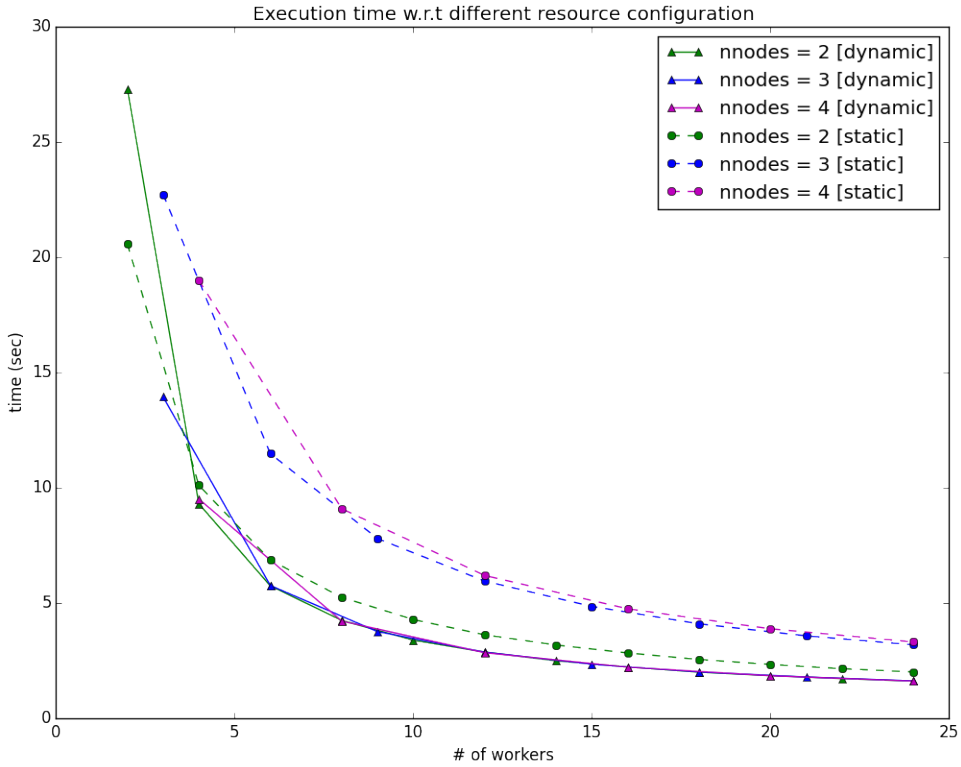


Figure 8: Execution time with different number of nodes/ppn: the results show that there is little difference between configurations when applying dynamic scheduling, while applying static scheduling with two nodes seems to be a better choice.

- Coloring

In addition to all works mentioned above which are related to performance concerns, the way checking if the program works accurately is also important. Therefore, it might be a good solution to display a 2-D diagram where each pixel refers to iterations at that position because the results can be checked visually/intuitively. The mapping function mapped from integers to gray scales suggested originally is $f(x) = (x \mod 256)/255.0$, this function, however, cannot acctually illustrate the real magnitude, i.e. $f(x) \not> f(y), \forall x > y$. As a result, I decide to use sigmoid function $\phi(x) = \frac{1}{1+\exp(-x)}$ and propose a monotonic function $g(x) = 2 \times \phi(\log(x)) - 1$, $g(x) \in [0, 1]$. The visual results have been shown below:

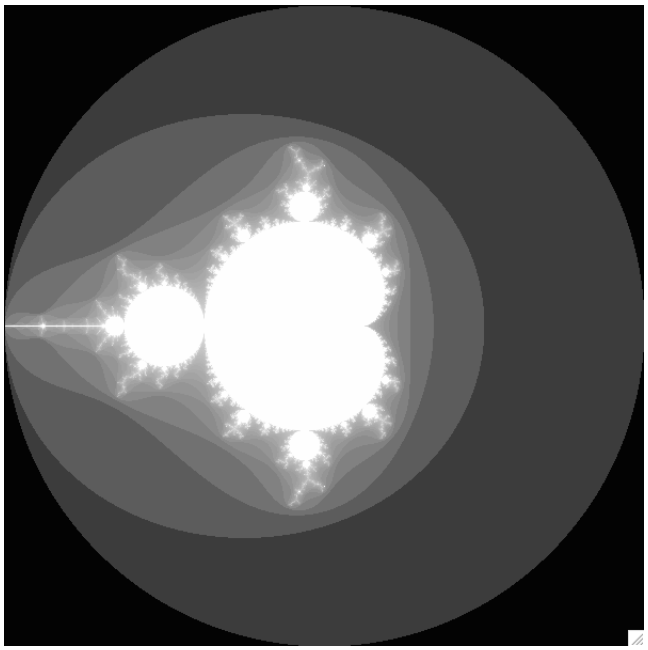


Figure 9: Visual result (8-bit)

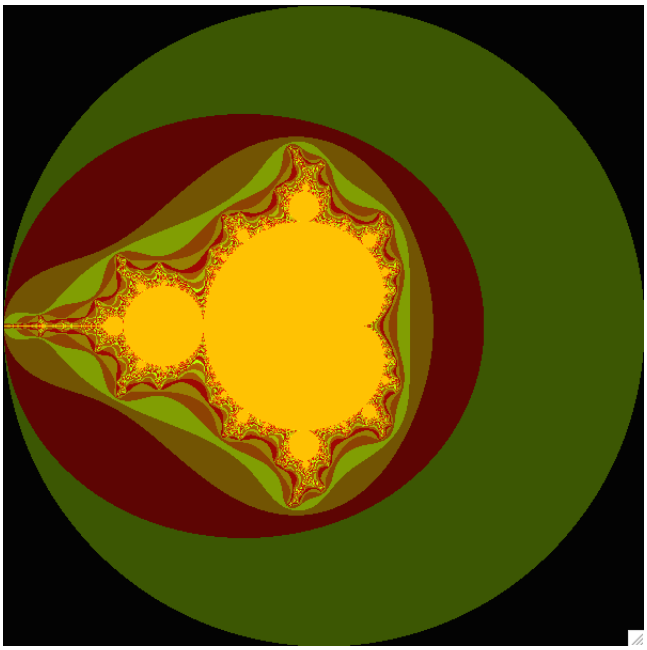


Figure 10: Visual result (16-bit)

This project provides an practice to experience advantages and disadvantages of MPI/OpenMP relatively, and to make good use of both models and optimize the algorithm we want to implement.

- MPI (Distributed Memory/Message Passing Model)

The main feature of MPI is providing communication interface between processes, and it is limited by communications time consumption as the number of processes increases.

- OpenMP (Shared Memory/Threads Model)

The main feature of OpenMP is providing memory management interface between threads, and it is limited by scability, in other word, it is limited by the number of physical cores built in a computer.

Hybrid is used for compensating disadvantages and combining advantages of MPI and OpenMP: 1) reducing memory duplication with shared memory, 2) reducing communications, 3) increasing portability, 4) increasing scability for advanced systems.

Additional Work: Dynamic MPI Implementation

In dynamic implementations, the most important thing is how to implement a master, and in the dynamic MPI implementation I directly assign a worker to be the master. However, it sacrifices a computing unit and then leads the speedup factor to be reversely propotional to only $N_{wks} - 1$ instead of N_{wks}. In practice, the master is always idle for waiting and checking if any message has been sent and the computation power is wasted. To improve the situation, the way assigning a worker to be not only a master, but a worker is a good choice. Therefore, Pthread has been used. In this way, the refactored results have been shown below:

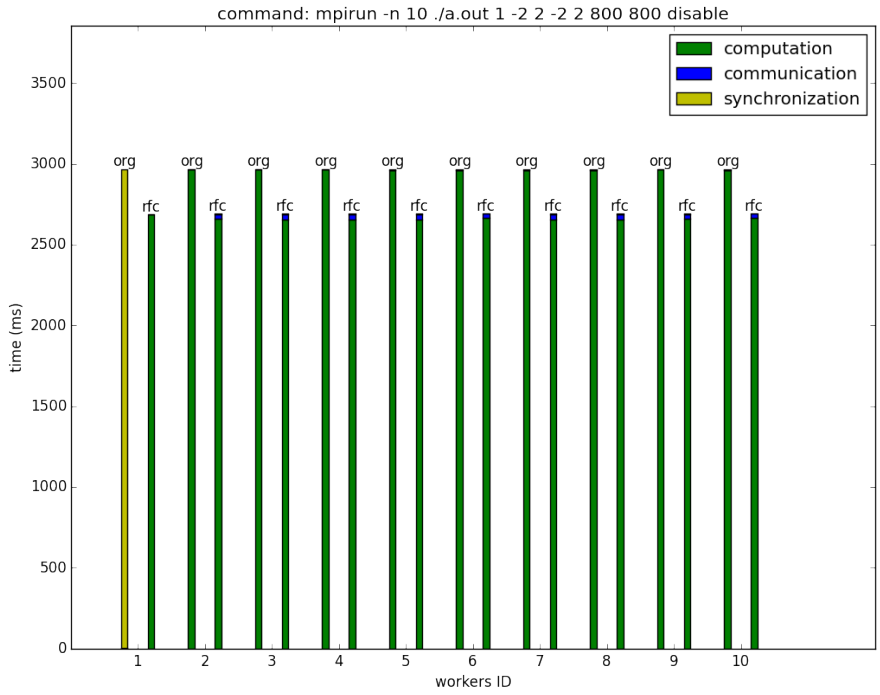


Figure 11: Loads figure of different dynamic MPI implementations: Choosing one worker to be both worker and master, the worker will launch an extra thread for managing jobs. This strategy will decrease the efficiency of communication but make the worker not always idle anymore, so the total results can be improved.

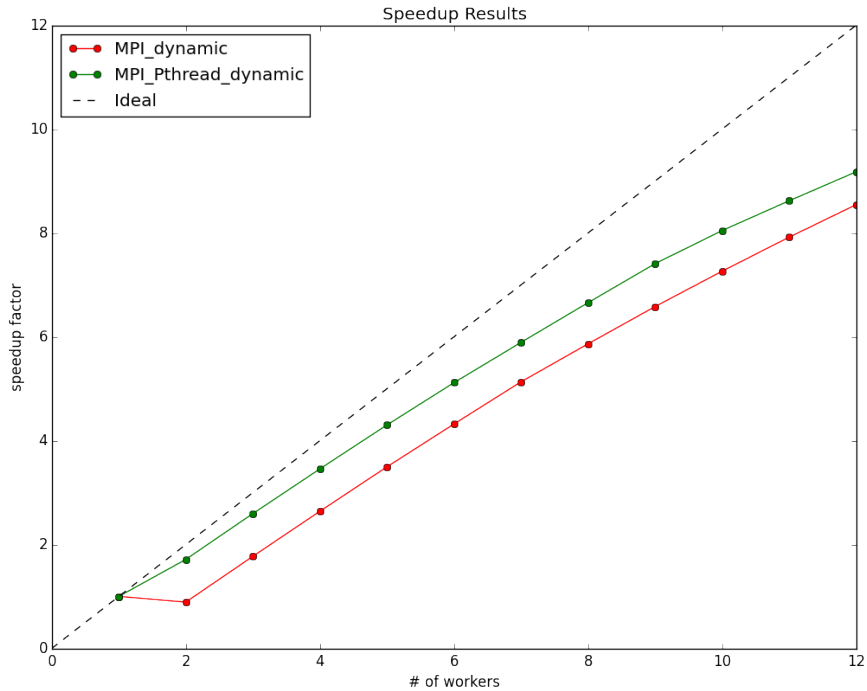


Figure 12: Speedup factors between different dynamic MPI implementations: In this strategy, the results are closer to ideal results. However, it suffers from message passing as the number of workers increases.