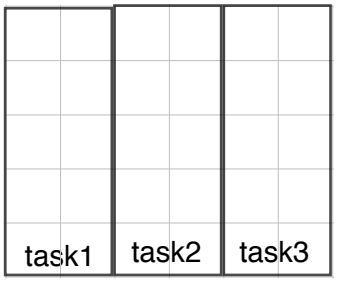
# Parallel Programming HW3

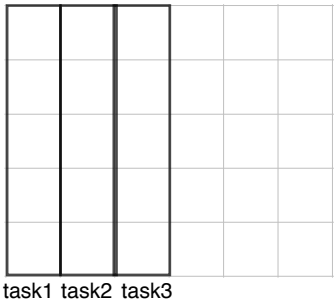
## Mandelbrot Set

### Implementation

1. MPI_static Version
   - Divide the columns into equal-size chunks and distribute a chunk to a process.
   - If `width` is not dividable by the number of process, add some redundant columns to make it dividable, but ignore those columns when computing.
   - Use `MPI_Gather()` to collect all computed results for the root process to draw.



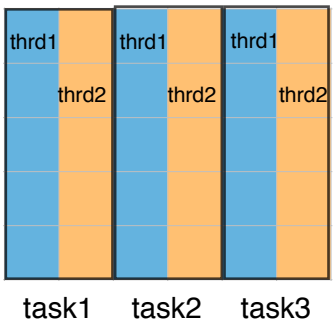

2. MPI_dynamic Version
   - An additional process is required to serve as master process, so when given T processes, only T-1 processes are computing.
   - Master process assigns tasks column by column to slave processes by sending the column ID to slave processes. The column ID indicates the column that the process should compute.
   - After a slave process finishes computing, it sends the column ID and the result back to the master process, and keep receiving new tasks until `MPI_TAG==0`. In other words, a slave process gets next column as the new task right after the previous one is done.
   - When the master process receives the result of certain column, it assigns another column to the slave process. If all columns are done, it sends a task with `MPI_TAG=0` to terminate the slave process.
   - When assigning task, `Isend()` is used so the master process can assign the next task immediately.



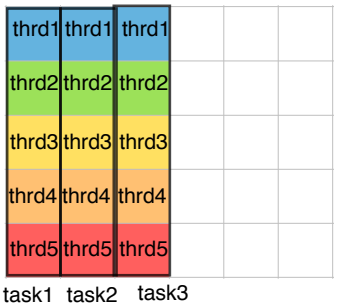
3. OpenMp_static Version
   - Specify `#pragma omp parallel for num_threads(threadN)` to divide the columns into even-size chunks and each thread gets a chunk to do computing. The task partitioning resembles that of MPI_static.

4. OpenMp_dynamic Version
   - Add the schedule clause `schedule(dynamic)` after the specification in OpenMp_static, then each thread receives one column and after computing, it gets another one.
   - The partitioning scheme resembles that of MPI_dynamic, but all T threads participate in computing.

5. Hybrid_static Version
   - Uses the same partitioning scheme in MPI_static.
   - Within each process, a chunk is distributed to each thread column by column, which is the same as OpenMp_static.




6. Hybrid_dynamic Version
   - One process performs as the master process that assigns tasks column by column to slave processes. This is the same as the partitioning scheme in MPI_dynamic.
   - Within a process, the same method of OpenMp_dynamic is specified to each thread to compute row by row for every column assigned.



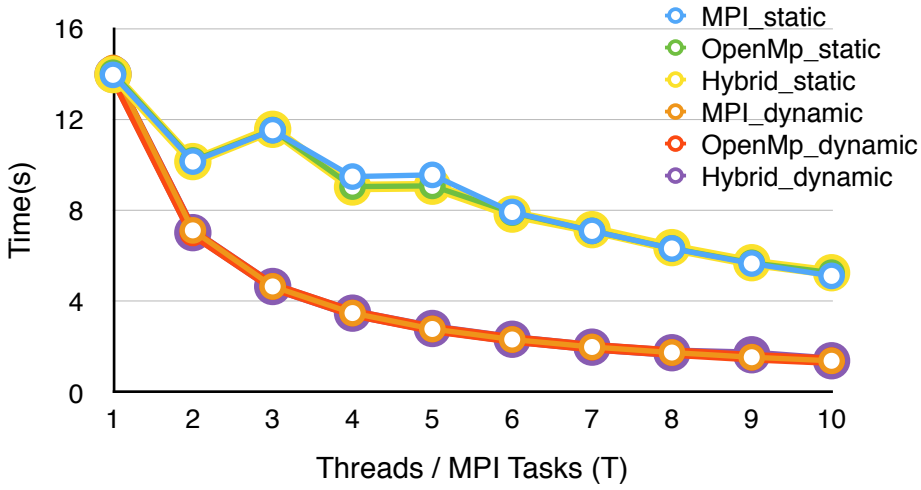
## Experiment & Analysis (Select Maximal Execution Time Among Threads/Tasks)

1. Strong Scalability

- Objective : The relationship between execution time and number of threads/tasks.
- Input Setting :

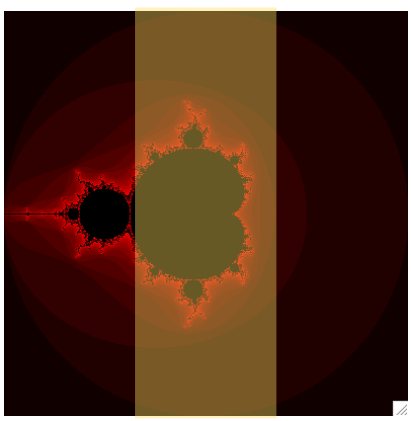
| | |
|---|---|
| Range: | x[-2 2], y[-2 2] |
| Points: | 400*400 |
| Hybrid_static: | change #thread |
| Hybrid_dynamic: | change #ppn |

- Result :

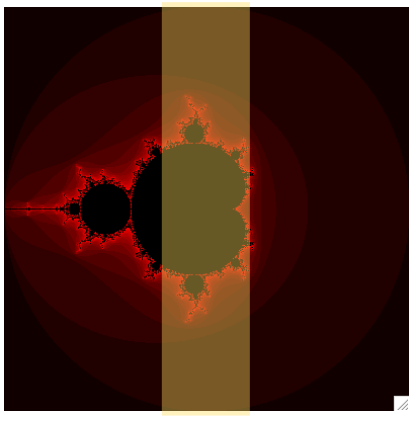



- Observation :

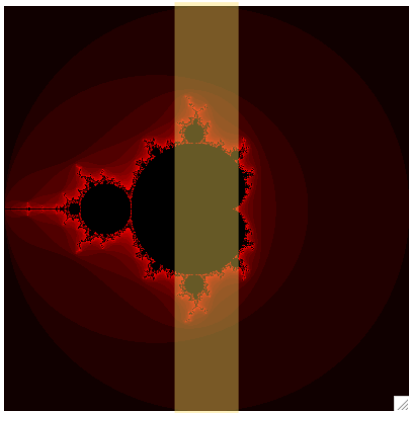
- The overall time decreases as the number of threads/tasks increases. It is worth noting that at T=3 and T=5, static versions spend more time. This happens because when dividing into 3 and 5 chunks, the middle threads get the time-consuming parts as highlighted below, making those threads to be the bottleneck. And as T grows to 7, the time-consuming part is getting smaller, so the load of computing is more distributed.



T=3



T=5



T=7

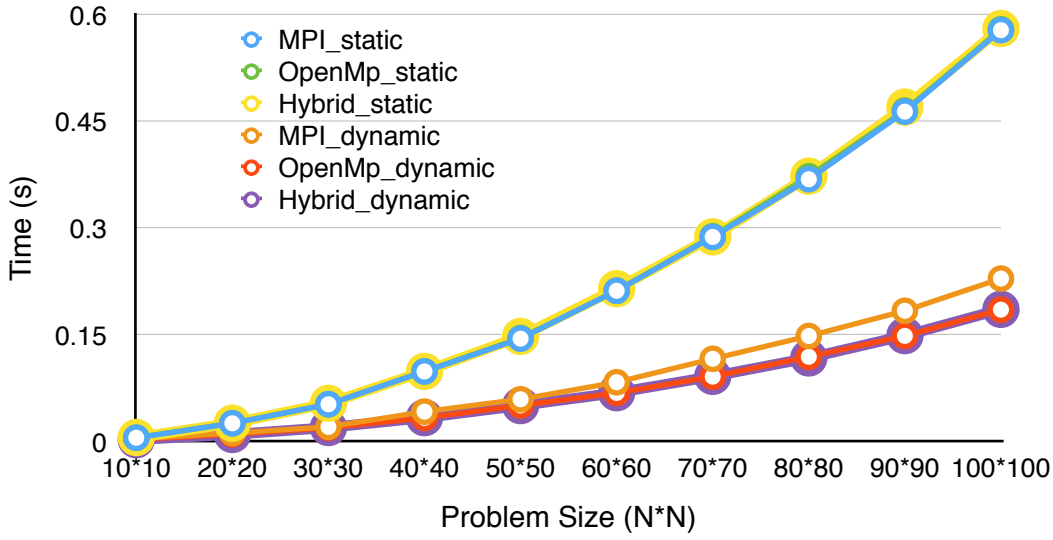
- For dynamic versions, since the threads/tasks receive another task after the previous one is done, the load of computing is distributed by computing time instead of number of points, making the execution more efficient thus spend less time than the static versions.
- Since the static versions utilize similar partitioning method—dividing into chunks, so their performance are also very similar, and the same reason for the results of dynamic versions.

2. Weak Scalability

- Objective : The relationship between execution time and problem size.
- Input Setting :

| | |
|---|---|
| Range: | x[-2 2], y[-2 2] |
| #Threads/Tasks: | 5 |

- Result :




- Observation :
  - For static versions, when problem size grows from N*N to αN*αN, the execution time increases by approximately $\alpha^2$.
  - For dynamic versions, the execution time does not increase by such factor. When N=50, the increase rate is 15 in average, approximately $0.6\alpha^2$; when N=100, the increase rate is 58 in average, approximately $0.6\alpha^2$. This happens because the partitioning method is different from that of static versions. Load of computing is distributed by computing time rather than number of points, leading to a more efficient result.

- MPI_dynamic runs longer than OpenMp_dynamic because of communication time between processes . Although Hybrid_dynamic also needs communication between processes, every task within a process is parallelized by threads, so the communication time is balanced off by the parallelism.

3. Load Balance

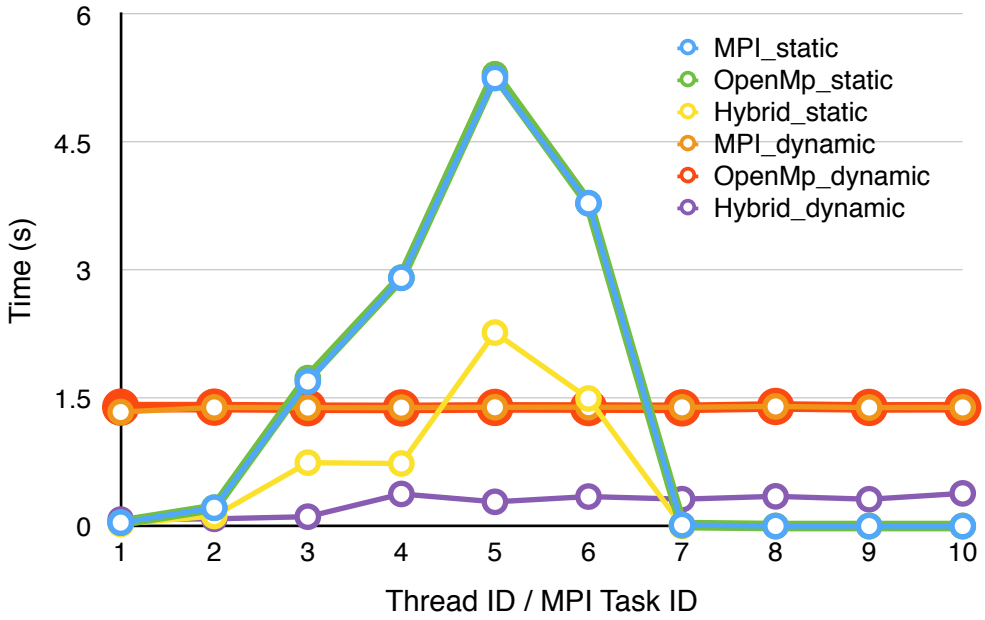
- Objective : The computing time of each thread / task.
- Input Setting :

| | |
|---|---|
| Range: | x[-2 2], y[-2 2] |
| Points: | 400*400 |
| #Thread / Task: | 10 |
| Hybrid_static: | change #thread |
| Hybrid_dynamic: | change #ppn |

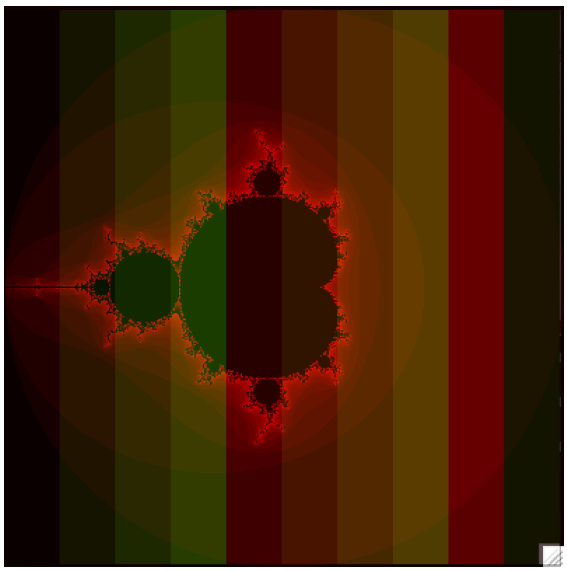
- Numeric Result :

| | MPI_static | OpenMp_static | Hybrid_static | MPI_dynamic | OpenMp_dynamic | Hybrid_dynamic |
|---|---|---|---|---|---|---|
| 1 | 0.040649 | 0.040739 | 0.024236 | 1.339367 | 1.396290 | 0.076172 |
| 2 | 0.215411 | 0.215231 | 0.118449 | 1.393134 | 1.396598 | 0.085205 |
| 3 | 1.697191 | 1.737396 | 0.744760 | 1.391067 | 1.389962 | 0.110401 |
| 4 | 2.914166 | 2.916684 | 0.734612 | 1.391070 | 1.387293 | 0.377274 |
| 5 | 5.260104 | 5.304481 | 2.271731 | 1.395138 | 1.393291 | 0.284835 |
| 6 | 3.790769 | 3.790698 | 1.497923 | 1.396117 | 1.391804 | 0.346610 |
| 7 | 0.011833 | 0.011801 | 0.001841 | 1.394466 | 1.387989 | 0.315977 |
| 8 | 0.000564 | 0.000552 | 0.000361 | 1.404753 | 1.404612 | 0.349423 |
| 9 | 0.000511 | 0.000497 | 0.000420 | 1.391721 | 1.393026 | 0.315251 |
| 10 | 0.000474 | 0.000469 | 0.000291 | 1.393522 | 1.395290 | 0.384189 |

- Figurative Result :



- Observation :
  - For static versions, from threads/tasks 3 to 6 the computing time increases because they deal with the most time-consuming middle part of Mandelbrot set. The image on the right shows the part each thread/task is responsible for. Chunks 3 to 6 cover most Mandelbrot set, which takes a longer time to compute.



  - For Hybrid_static, task of a process is distributed to threads, so it requires less time than MPI_static and OpenMp_static.
  - It's obvious that the computing loads between threads/tasks are imbalanced, and those who need much time become the bottleneck.
  - For dynamic versions, the loads between threads/tasks are balanced, resulting in better efficiency.
  - For Hybrid_dynamic, since the task of a process is distributed to threads, so it runs much faster than MPI_dynamic and OpenMp_dynamic.

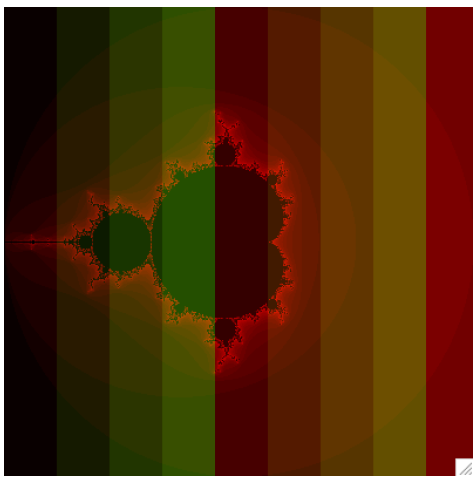
4. Load Balance

- Objective : The number of points every thread/task holds.
- Input Setting :

| | |
|---|---|
| Range: | x[-2 2], y[-2 2] |
| Points: | 400*400 |
| #Thread/Task: | 9 |
| | |
| Hybrid_static: | change #thread |
| Hybrid_dynamic: | change #ppn |

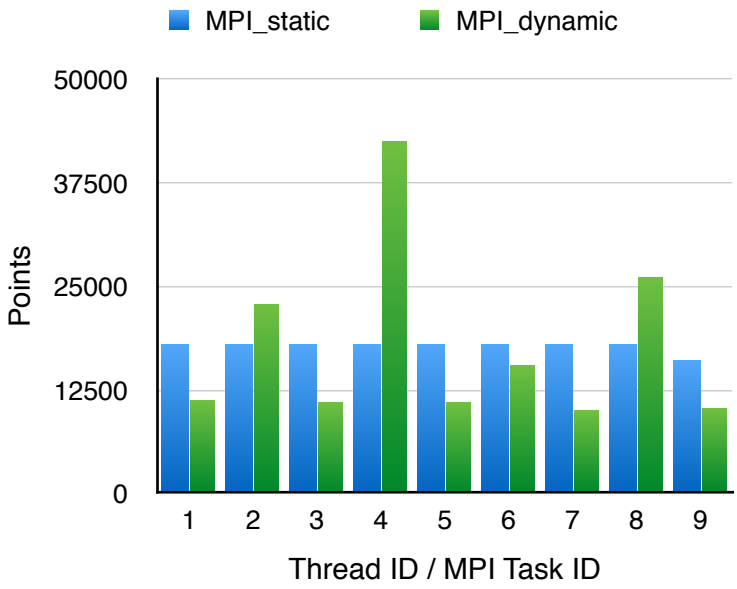
- Result Visualization :
  - Instead of drawing the image according to the repetition, I also draw the image according to the thread/task ID responsible for each chunk.
  - The following ID-color indicator image shows the color and chunk of each thread/task in static versions.

  - When putting two images together, it's easier to understand which thread/task computes each columns.

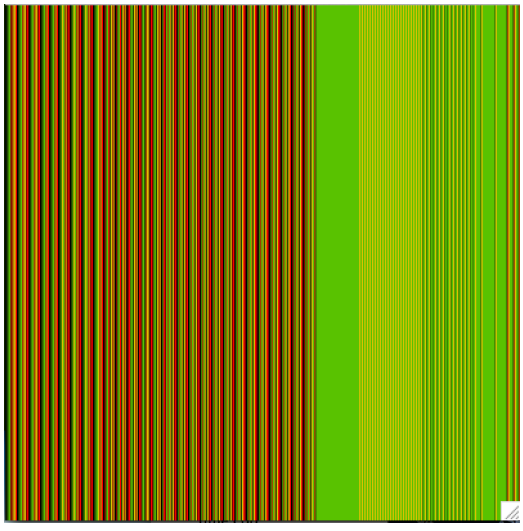


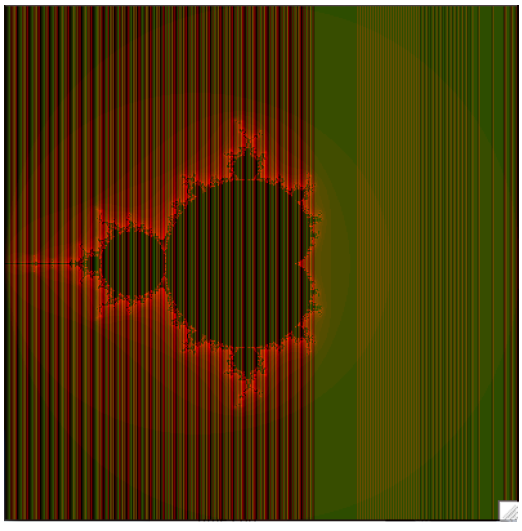
- MPI Result :

| | MPI_static | MPI_dynamic |
|---|---|---|
| 1 | 18000 | 11200 |
| 2 | 18000 | 22800 |
| 3 | 18000 | 10800 |
| 4 | 18000 | 42400 |
| 5 | 18000 | 10800 |
| 6 | 18000 | 15600 |
| 7 | 18000 | 10000 |
| 8 | 18000 | 26000 |
| 9 | 16000 | 10400 |



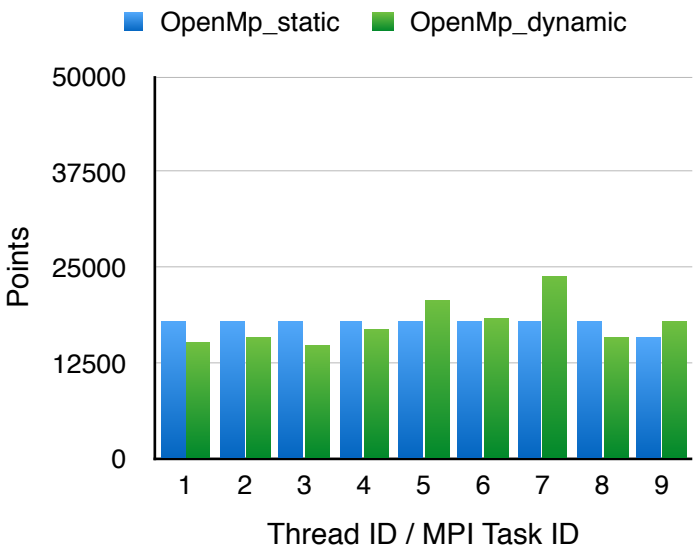

- From the data chart and bar graph we know that processes 2, 4, 8 compute most points.
- From the images on the left below, it can be pointed out that almost all points of the right half part of the set are handled by certain processes. And according to the ID-color indicator image on the previous page, we know that those points are handled by processes 2, 4, 8.
- Most processes are busy computing the time-consuming part of the set, and they cannot frequently receive new columns, so 2, 4, 8 can keep receiving new columns since they are luckily not stuck in the time-consuming part of the set.



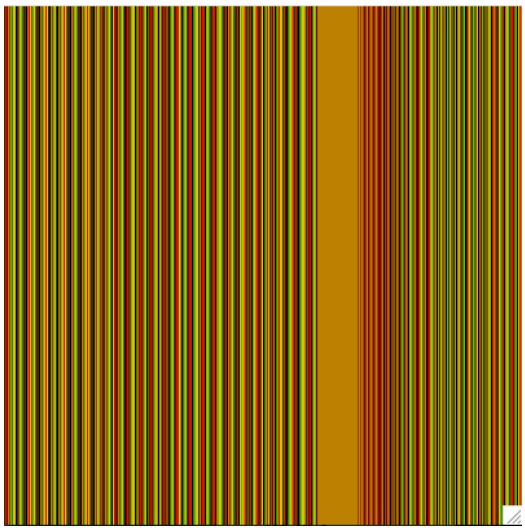


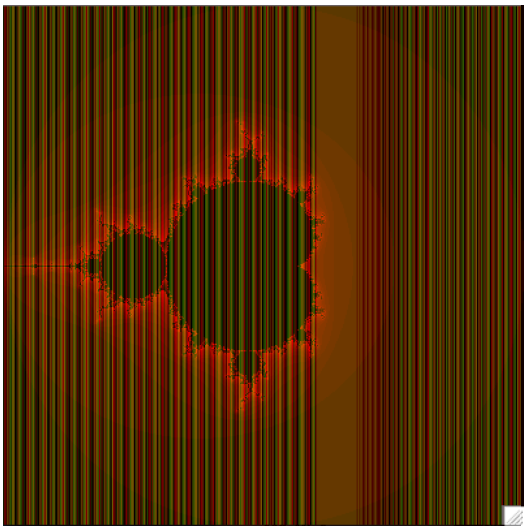
- OpenMp Result :

| | OpenMp_static | OpenMp_dynamic |
|---|---|---|
| 1 | 18000 | 15200 |
| 2 | 18000 | 16000 |
| 3 | 18000 | 14800 |
| 4 | 18000 | 16800 |
| 5 | 18000 | 20800 |
| 6 | 18000 | 18400 |
| 7 | 18000 | 24000 |
| 8 | 18000 | 16000 |
| 9 | 16000 | 18000 |



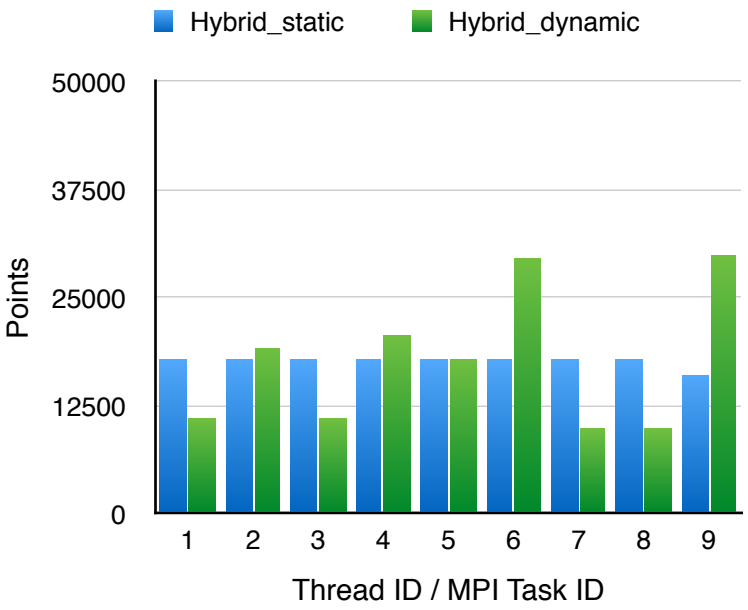

- From the data chart and the bar graph, it's apparent that the number of points is almost equally spread. Threads 5, 7 tend to deal with more points, but the difference is not as big as in MPI.
- In the images below, the light brown area is handled by only thread 7, but after a few columns of computation, other threads gradually get involved in computing the right half part. The dynamic scheduling in OpenMp makes the right half part more parallelized than MPI_dynamic.



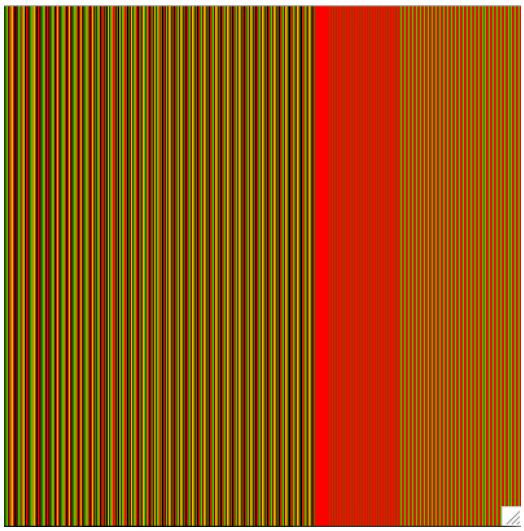


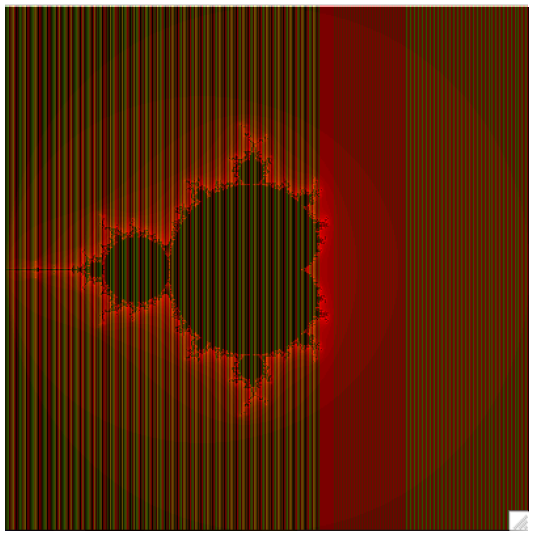
- Hybrid Result :

| | Hybrid_static | Hybrid_dynamic |
|---|---|---|
| 1 | 18000 | 11200 |
| 2 | 18000 | 19200 |
| 3 | 18000 | 11200 |
| 4 | 18000 | 20800 |
| 5 | 18000 | 18000 |
| 6 | 18000 | 29600 |
| 7 | 18000 | 10000 |
| 8 | 18000 | 10000 |
| 9 | 16000 | 30000 |




- From the data chart and the bar graph we know there are three threads, 4, 6, 9, that compute more than 20000 points.
- As the images below show,, Hybrid_dynamic behaves similarly to MPI_dynamic. The red part is obvious the most time-consuming part, because only 2 threads, 6 and 9, are computing points from the red area.
- On the right-most striped area, there are only 3 threads, 4, 6 and 9, are handling those points, since other threads may be still dealing with the time-consuming points.



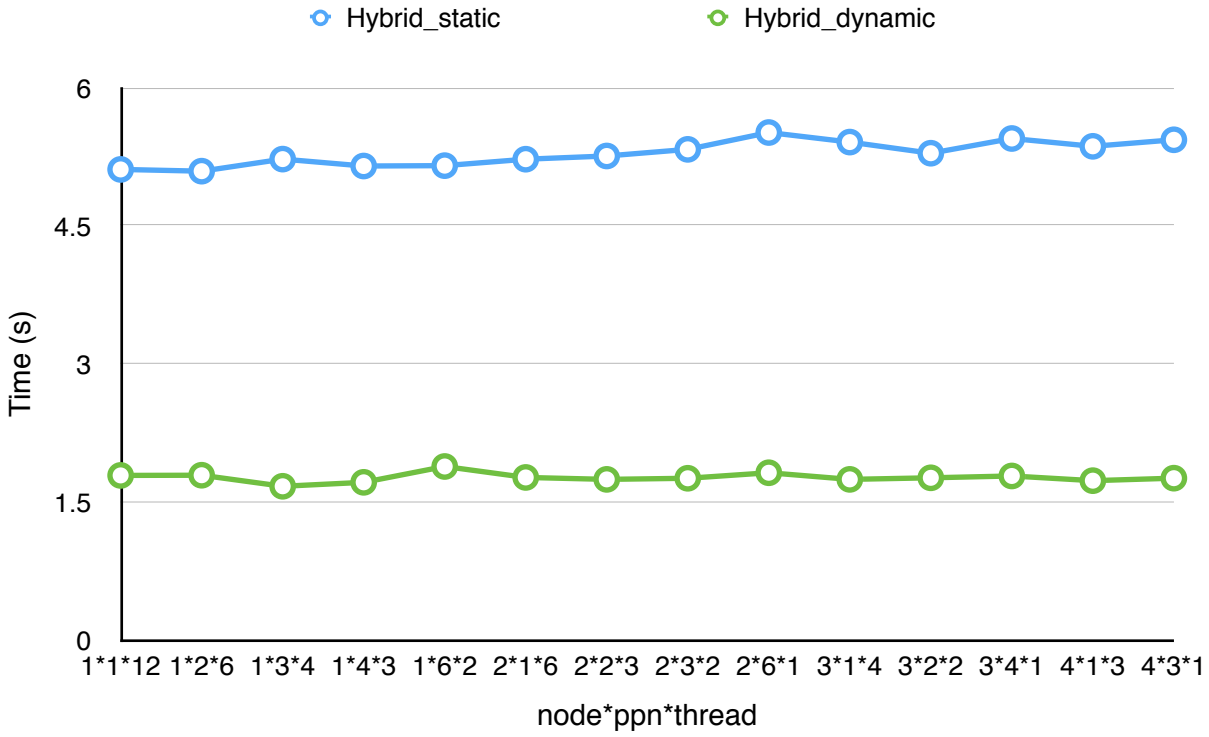


5. Distribution Between nodes & ppn & threads

- Objective : The execution time of different combination of nodes & ppn & threads of a fixed total thread number in hybrid versions.
- Input Setting :

| | |
|---|---|
| Range: | x[-2 2], y[-2 2] |
| Points: | 400*400 |
| #Thread/Task: | 12 |

- Result :




- Observation :
  - When the total thread number is fixed, all combinations seem to perform equally.

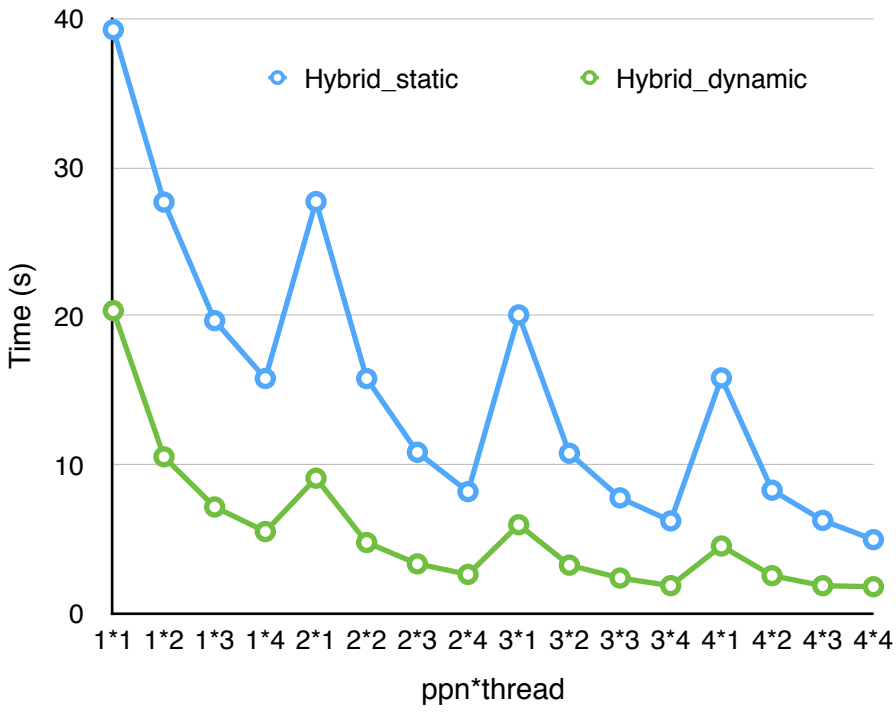
6. Distribution Between ppn & threads

- Objective : The execution time between different combination of nodes & ppn & threads in hybrid versions.
- Input Setting :

| | |
|---|---|
| Range: | x[-2 2], y[-2 2] |
| Points: | 800*800 |
| #Node: | 4 |

- Result :

| ppn*thread | Hybrid_static | Hybrid_dynamic |
|---|---|---|
| 1*1 | 39.229 | 20.351 |
| 1*2 | 27.645 | 10.529 |
| 1*3 | 19.680 | 7.161 |
| 1*4 | 15.795 | 5.510 |
| 2*1 | 27.681 | 9.096 |
| 2*2 | 15.789 | 4.776 |
| 2*3 | 10.837 | 3.340 |
| 2*4 | 8.191 | 2.631 |
| 3*1 | 20.066 | 5.969 |
| 3*2 | 10.771 | 3.252 |
| 3*3 | 7.772 | 2.387 |
| 3*4 | 6.224 | 1.875 |
| 4*1 | 15.784 | 4.538 |
| 4*2 | 8.290 | 2.549 |
| 4*3 | 6.261 | 1.871 |
| 4*4 | 4.967 | 1.804 |




- Observation :
  - When the number of threads increases, the execution time decreases.
  - For Hybrid_static, when thread number is fixed, the execution time remains the same. For example, 1*4 and 2*2 and 4*1 all result in same amount of execution time of 15.7.
  - For Hybrid_dynamic, when thread number is fixed, the execution time dwindles as ppn increases. For example, 1*4:5.510, 2*2:4.776, 4*1:4.538. This happens because there are more processes can receive tasks from the master process, so the execution time can be slightly reduced.

**Experience**

From the Mandelbrot set assignment, I learned how to combine MPI and OpenMp with extension of known knowledge and I became more familiar with the relationship between nodes, processes, and thread, and able to manipulate them into required scales.

Since this assignment does not need complicated coding, interpretation of the experiment results are more challenging than previous assignments.

Another difficulty I encountered is that when implementing MPI_dynamic and Hybrid_dynamic, I choose process 0 to serve as the master process and other 1~T-1 are slave processes. A problem emerges when using only 1 ppn, there is only master process and no slave processes. So in all my experiments with MPI_dynamic and Hybrid_dynamic, I had to assign an additional process to run as master.