Title: Mandelbrot Set

Name: 曹寓恆

Student ID: 110521167

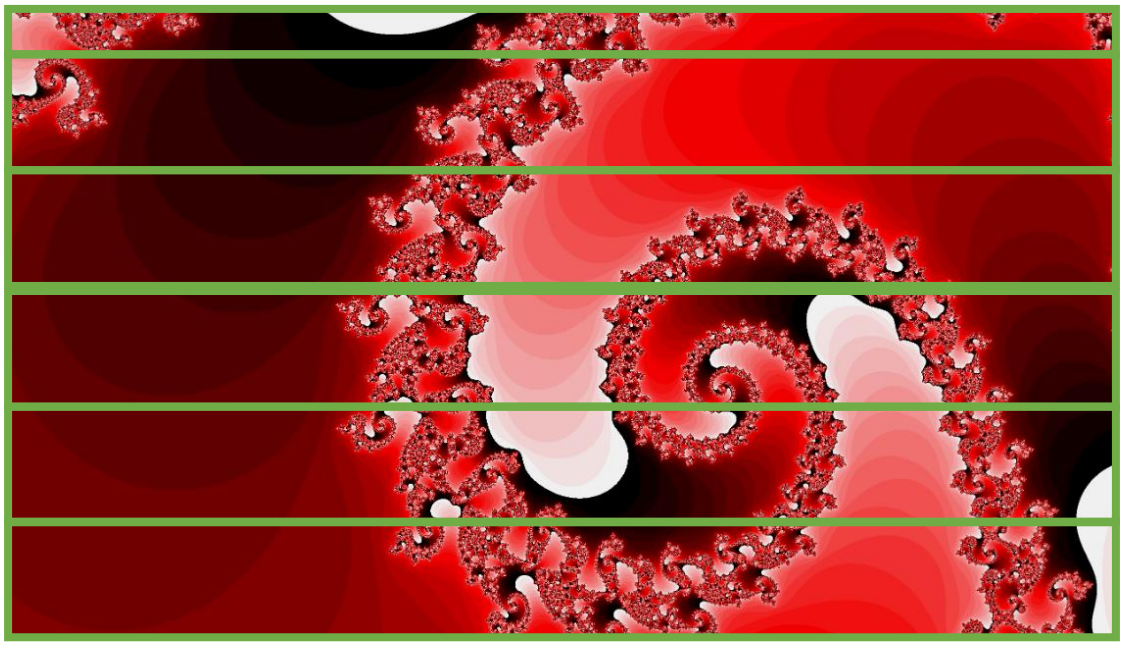
Implementation

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

I set process 0 to be the master and the other processes to be the servants. At initialization, the master process splits the image into strips and assign them to servant processes. When the servant process has finished the calculation, it will submit the result and take over the next strip until it receives the end flag.

2. How do you partition the task?

(1) Set strip_height to 32 times #process.
(2) If 32 times #process less than 400, then set strip_pixels to 400



Testcase slow1 is split into strips

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

According to my program, in the rendering section, dynamic scheduling policy in OpenMP for loop is the most effective. In my opinion, the reason is that the difference in the number of iterations per pixel is very large.

4. Other efforts you've made in your program.

I want to create a thread to handle the scheduling and have all processes join the calculation, so I tried calling MPI_Init_thread() with MPI_THREAD_MULTIPLE, however, I still can't solve the segmentation fault.

Experiment & Analysis

1. System spec

(1) 19 nodes for this course (apollo31 - 48, 50)
(2) Intel X5670 2x6 cores @ 2.93GHz (Hyper threading disabled)
(3) 96GB RAM (each node)
(4) 5.5TB shared RAID5 disk
(5) QDR Infiniband (40 Gb/s)

2. Performance metrics (experiment with slow01)

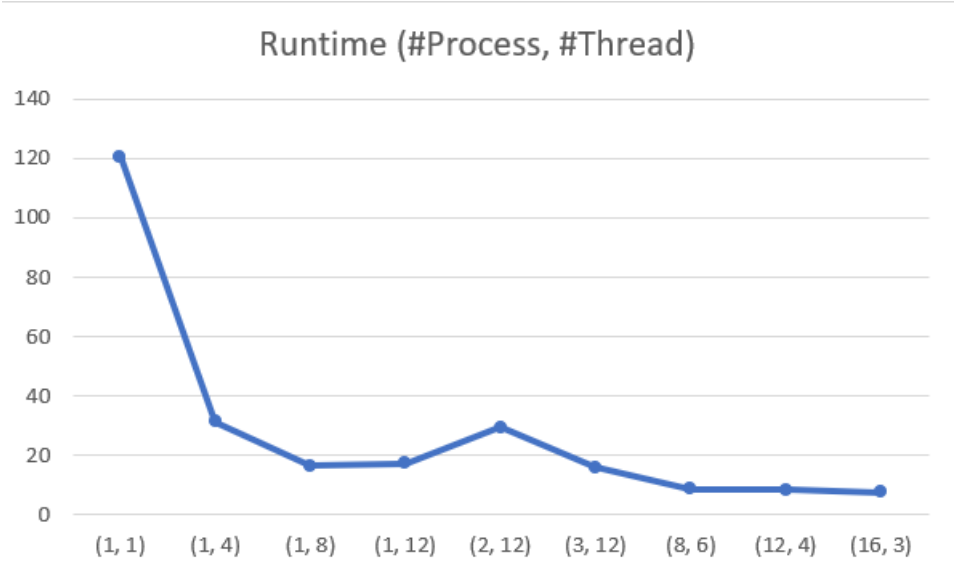


Figure-1

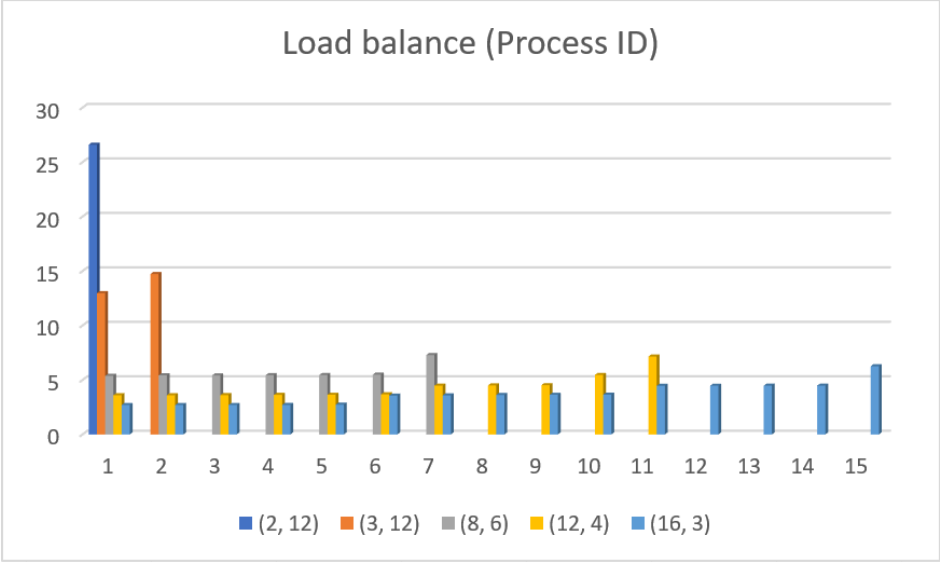


Figure-2

3. Measurement method

```
void cpu_begin()
{
    if (enable)
        cpu_temp = MPI_Wtime();
}
void cpu_end()
{
    if (enable)
        cpu += MPI_Wtime() - cpu_temp;
}
```

```
  timer.cpu_begin();
  proc->render(blk);
  timer.cpu_end();
```

Call MPI_Wtime at the beginning and end of the function

```
void Timer::print_cpu_time()
{
    if (!enable)
        return;

    /* int pid = -1, nproc; …
    printf("CPU time:\t%f\n", cpu); // self cpu time
}
```

Print how much time to spend on the rendering

4. Discussion

(1) Scalibility

Because the master process does not join the computation, more processes can dilute such disadvantages. Look at Figure-1, in the single-process case, multithreading brings ideal parallelism, but in the two-process case, one process needs to act as the master and the other as the servant, effectively only the servant is computing, and increasing the time spent on communication. However, as the number of processes increases, the master-servant architecture can bring further acceleration.

(2) Load balance

Look at Figure-2, a fine-grained policy results in more load balancing in large testcases, but it increases communication time. After a trade-off, the implementation of my program is setting a lower bound and split the image into strips as much as possible.

Experiences / Conclusion

In this task, I learned about hybrid parallel architectures, the use of the pthread library. And I would like to ask the TA to introduce more tools like address-sanitizier if possible, thanks.