Parallel Programming for Mandelbrot set

Hao-Hsuan Tseng Ching-Yi Chang
National Chiao Tung University

http://dpeecs.nctu.edu.tw/

ABSTRACT

This project describes to draw Mandelbrot set by using two types of parallel programming method, OpenMP and MPI. Further we also compare with non-parallel programming (serial) method about the speed-up performance. The mechanism of OpenMP is based on a shared memory programming; MPI is a distributed memory programming basis. However, the two methods have their own pros and cons, the OpenMP will lack in I/O access and getting lower speed-up; MPI will affect by software limitation. We will demonstrate the Mandelbrot set by different numbers of threads and find the best implementation method. Moreover, we will reveal clear and fine Mandelbrot images as in conclusion.

KEYWORDS

Fractal, Mandelbrot set

1. INTRODUCTION

We are curious about the animation at the end of the movie "Annihilation (from Netflix movie)"as in Figure 1. We wonder to know if the animation can be generated by mathematics, then we could parallel our program easily

*Figure 1: The computer generated images using **Mandelbrot set** in the move Annihilation*

Fractal derives from the Latin word "fractus", meaning irregular and to break. Shapes are made of smaller copies of itself, in which the copies are similar to the whole. Uses Iteration where the same process is applied over and over again in order to obtain a closer approximation

In 1904, Swedish Mathematician Von Koch also challenged Weierstrass from a geometrical viewpoint, he wanted to find a different way to prove that functions were non differentiable. i. He took an equilateral triangle and divided each of the three sides by three equal segments as shown in Figure 2(a). ii. Then he placed another equilateral triangle facing outward. iii. Then the middle part is removed. iv. The process continues infinitely as shown in Figure 2(b).

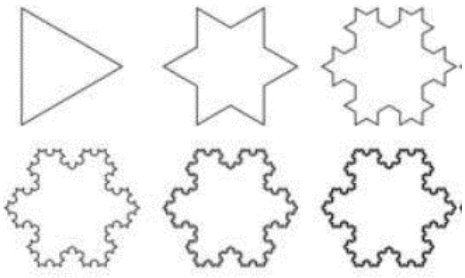


Figure 2(a): An equilateral triangle and divided each of the three sides by three equal segments

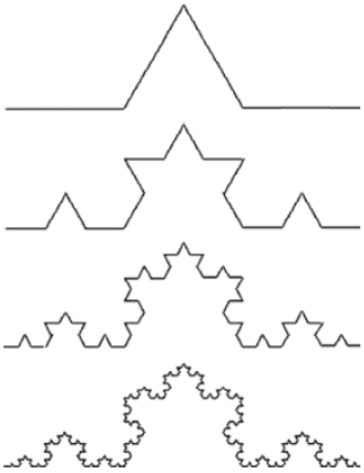


Figure 2(b): Keep on dividing the equilateral triangle to three sides, the segments will process to be infinite

The math is established by Mandelbrot, the Mandelbrot set implies the World War II ended he was left alone, isolated with fears and deprivation. Some think that his battles manifested itself into a phenomena that can't be measured, understood or interrelated. From Wikipedia, the fundamental of fractal standard for "The A formula can be very simple, and create a universe of bottomless complexity."

The background fractal meaning in Math

In 1980, while working for IBM in New York, Mandelbrot work on the Julia sets, He used a computer to run the equations millions of times. He combined the Julia sets and iterated the equations until it formed his own equation in Formula 1. Where c is a complex number and if c exist in the set. The Mandelbrot set became iconic for Fractal Geometry in Figure 4. From math, we know if complex number c>2, the following point will divergent, not belong Mandelbrot set as a converge point.

$$\lim_{n\to\infty} | Z_{n+1} | \leq 2$$

$$Z_{n+1} = Z_n^2 + c$$

Formula 1: By iteration, when n become to infinity Z_{n+1} will convergent if Z_{n+1} within the boundary 2.

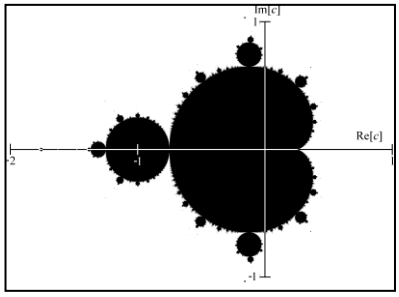


Figure 4: The standard type of Mandelbrot set

Further, the proof by simple math about the complex c condition and the result belong to Mandelbrot set or not as in Figure 5(a) and 5(b)

定理 1. 若 $|c| \leq \frac{1}{4}$，則 $c \in M$

【證明】若 $|z_0| = |c| \leq \frac{1}{4}$，則 $|z_1| = |z_0^2 + c| = |c^2 + c| \leq |c|^2 + |c| \leq \frac{1}{4} + \frac{1}{4} = \frac{1}{2}$

由數學歸納法我們可以證得 $|z_n| = |z_{n-1}^2 + c| \leq |z_{n-1}|^2 + |c| \leq (\frac{1}{2})^2 + \frac{1}{4} = \frac{1}{2}$

故 $c \in M$

Figure 5(a): If absolute c belong to an finite complex number with boundary limited.

定理 2. 若 $c \in M$，則 $|c| \leq 2$

【證明】若 $|z_0| = |c| > 2$，令 $r = |c| - 1 > 1$，則

$|z_1| = |z_0^2 + c| = |c^2 + c| = |c||c+1| \geq |c|(|c|-1) = |c|r$

同理可證 $|z_2| = |z_1^2 + c| \geq |c|r(|c|r - \frac{|c|}{|c|r}) \geq |c|r(|c|-1) = |c|r^2$

由數學歸納法我們可以證得 $|z_n| \geq |c|r^n$

因爲 $r > 1$，所以 $|z_n| \to \infty$，即 $\langle z_n \rangle$ 發散

Figure 5(b): If absolute c is bigger than 2, the following points will get divergent and not in Mandelbrot set.

2. PROGRAM REPRESENTATION

The one of Mandelbrot set is composed by points from #0 to #N. Each blocks has no data dependence. Using row major way to partition into N parts. Comparing performance between OpenMP(shared memory model) and MPI(distributed memory model) as in Figure 6, 7(a) and 7(b).

(1) Using conditional expressional check first

(2) Endless iteration by serial and parallel method comparison

(3) Utilization RGB color model and draw the Mandelbrot images implementation

Figure 6: Work flow of calculate Mandelbrot set

Figure 7(a): Concept of partitioning the image into pixels

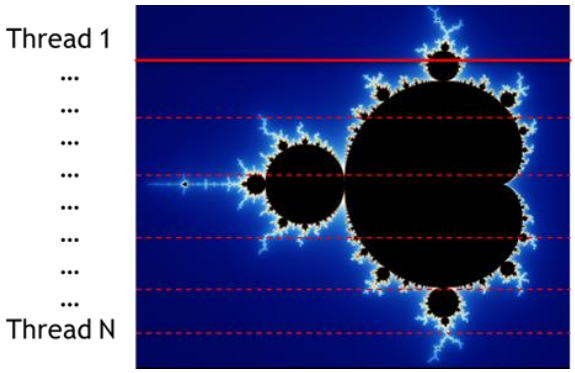


Figure 7(b): Divide the image into multithreads, threads from from Thread 1 to Thread N

2.1 OVERVIEW OF THE SYSTEM INFORMATION

System information: 4 core, 8 threads. Windows 10, visual Studio 2017, using C++ programing.

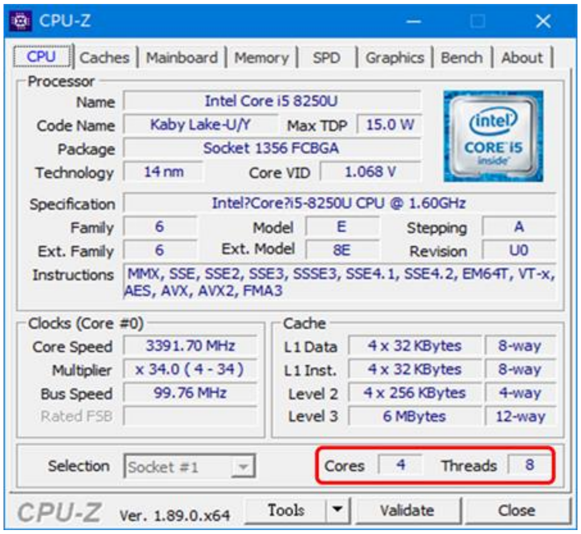


Figure 8: CPU by ineral core i5 and 4 cores and 8 threads setting

2.2 ORIGINAL SERIAL VERSION vs. PARALLEL VERSION COMPARSION

Serial version: 2048(W)*2048(H). Total Time:61.31 secs.
Parallel version:

(1) Producing Real and Imaginary by quadratic polynomial:

```
double mapToReal(int x, int imageWidth, double minR, double maxR)
{
    double range = maxR - minR;
    return x * (range / imageWidth) + minR;
}

double mapToImaginary(int y, int imageHeight, double minI, double maxI)
{
    double range = maxI - minI;
    return y * (range / imageHeight) + minI;
}
```

(2) Periodicity checking:

```
int findMandelbrot(double cr, double ci, int max_iterations)
{
    int i = 0;
    double zr = 0.0, zi = 0.0;
    while (i< max_iterations && zr*zr+zi*zi <4.0)
    {
        double temp = zr * zr - zi * zi + cr;
        zi = 2.0*zr*zi + ci;
        zr = temp;
        i++;
    }
    return i;
}
```

(3) Coloring:

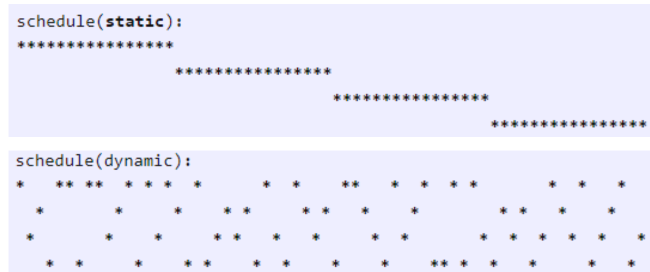
```
for (int y = 0; y < imageHeight; y++)
{
    for (int x = 0; x < imageWidth; x++)
    {
        double cr = mapToReal(x, imageWidth, minR, maxR);
        double ci = mapToImaginary(y, imageHeight, minI, maxI);
        int n = findMandelbrot(cr, ci, maxN);

        int r = (((int)(n*sinf(n))) % 256);
        int g = ((n*2) % 256);
        int b = (n % 256);
    }
}
```

(4) Method1: OpenMP(schedule without "chunk-size"):

```
#pragma omp parallel num_threads(i32ThreadNum) private(y,x)
{
    #pragma omp for schedule(THREAD_SCHEDULE)
    for (y = 0; y < imageHeight; y++)
    {
        for (x = 0; x < imageWidth; x++)
        {
            dfArrCr[y][x] =  x * (range1 / imageWidth) + minR;
            dfArrCi[y][x] =  y * (range2 / imageHeight) + minI;
        }
    }
}
```

(5) OpenMP(schedule without "chunk-size"):



(6) Method2: MPI

```
while( i32InitCnt<i32TotalRankCnt )
{
    i32Prev = i32RankId-i32InitCnt;
    i32Next = i32RankId+i32InitCnt;
    while( i32Next>=i32TotalRankCnt ) i32Next -= i32TotalRankCnt;
    while( i32Prev<0 ) i32Prev += i32TotalRankCnt;

    i32OffsetBack = i32Prev*i32BlockCnt;
    i32StFor = i32RankId*i32BlockCnt;

    MPI_Irecv(&r[i32OffsetBack][0] , i32ElePerBlock , MPI_INT , i32Prev , 0 ,MPI_COMM_WORLD, &Request[0] );
    MPI_Isend(&r[i32StFor][0] , i32ElePerBlock , MPI_INT , i32Next , 0 ,MPI_COMM_WORLD, &Request[1] );

    MPI_Irecv(&g[i32OffsetBack][0] , i32ElePerBlock , MPI_INT , i32Prev , 0 ,MPI_COMM_WORLD, &Request[2] );
    MPI_Isend(&g[i32StFor][0] , i32ElePerBlock , MPI_INT , i32Next , 0 ,MPI_COMM_WORLD, &Request[3] );

    MPI_Irecv(&b[i32OffsetBack][0] , i32ElePerBlock , MPI_INT , i32Prev , 0 ,MPI_COMM_WORLD, &Request[4] );
    MPI_Isend(&b[i32StFor][0] , i32ElePerBlock , MPI_INT , i32Next , 0 ,MPI_COMM_WORLD, &Request[5] );

    MPI_Waitall(SEND_RECV_CNT, Request, Status);
    i32InitCnt++;
}
```

2.3 ANALYZE PERFORMING ISSUE BY OPENMP

Using OpenMP: to evaluate our program, however, the parallel program doesn't have linear speedup. Because of I/O access dominated.

$$E = \frac{S}{p} = \frac{\left(\frac{T_{\text{serial}}}{T_{\text{parallel}}}\right)}{p} = \frac{T_{\text{serial}}}{p \cdot T_{\text{parallel}}}$$

Formula 2: Calculation speedup by serial and parallel method.

Threads	Serial (sec)	OpenMP (Static) (sec/times)	
1	61.32	59.46	1.03
2	N/A	53.55	0.57
3	N/A	52.97	0.29
4	N/A	50.63	0.15
5	N/A	50.18	0.10
6	N/A	49.8	0.08

Table 1: Serial calculated timing vs. different numbers of thread by OpenMP static method

2.4 OPTIMIZATION OPENMP BY REMOVE I/O ACCESS

The parallel program has linear speedup after removing I/O access. We would notify the speedup with obvious OpenMP with better performance than serial especial when arriving 8 threads, which matches the system setting 4 cores and 8 threads.

Threads	Serial (sec)	OpenMP(Static) (sec/times)		OpenMP (Dynamic) (sec/times)	
1	12.21	11.92	1.02	12.31	0.99
2	N/A	6.10	1.00	6.16	0.99
4	N/A	5.36	0.57	3.03	1.01
8	N/A	3.49	0.44	2.15	0.71
12	N/A	2.86	0.36	2.27	0.45
16	N/A	2.65	0.29	2.18	0.35

Table 2: Serial calculated timing vs. different numbers of threads by OpenMP static and optimization dynamic method

As the PP-s19 class teaching, the dynamic is used in unpredictable and highly variable work per iteration.

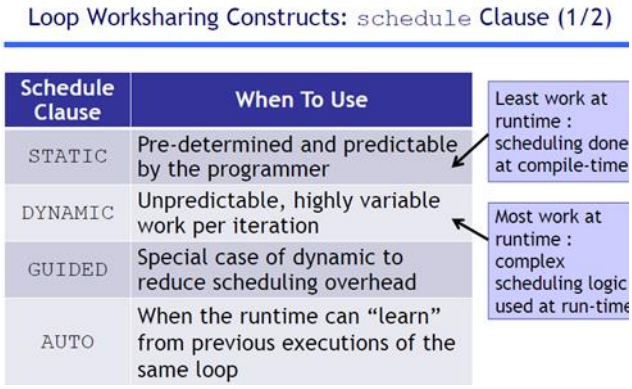


Figure 9: Schedule Clause "static vs. dynamic"

2.5 ANALYZE PERFORMING ISSUE BY MPI

Join the MPI in the parallel program and more processes will produce more overhead in MPI program. We observed that more processes will produce more overhead in MPI program.

Threads	Serial (sec)	OpenMP (Static) (sec/times)		OpenMP (Dynamic) (sec/times)		MPI	
1	12.21	11.92	1.02	12.31	0.99	11.94	1.02
2	N/A	6.10	1.00	6.16	0.99	5.96	1.02
4	N/A	5.36	0.57	3.03	1.01	5.08	0.60
8	N/A	3.49	0.44	2.15	0.71	3.45	0.44
12	N/A	2.86	0.36	2.27	0.45	2.93	0.35
16	N/A	2.65	0.29	2.18	0.35	30.3	0.25

Table 3: OpenMP dynamic vs. MPI also have significant speedup when increasing more multithreads.

The result coincides with the PP-s19 learning, the effort of message passing at upfront will consume a lot but easily to debug and less time to solution as shown in Figure 10[9].

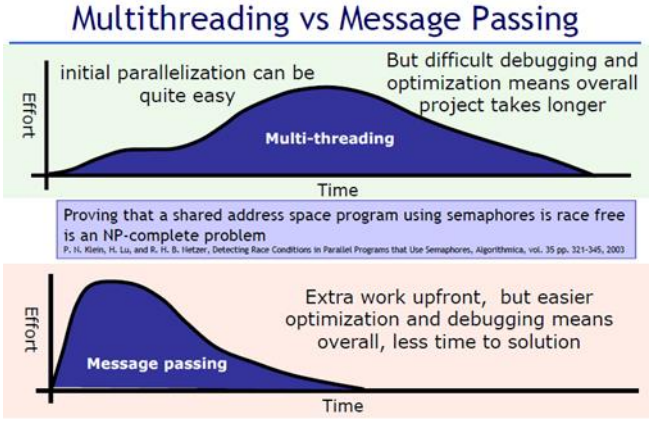


Figure 10: MPI message passing

3. COMPARISON

The best way to parallel data is OpenMP with dynamic thread scheduling in this case.

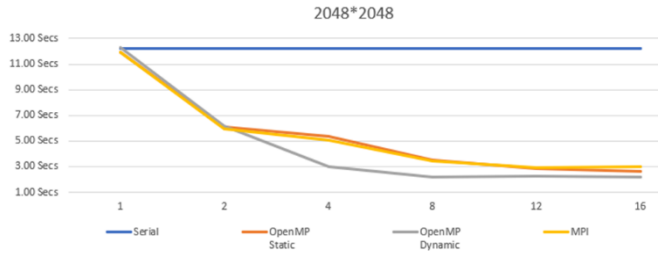


*Figure 11(a): The calculation time of 2048*2048 pixels comparison*

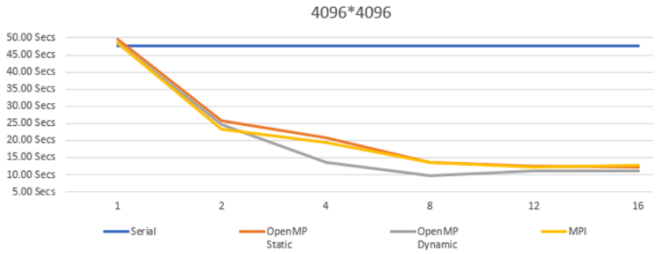


*Figure 11(b): The calculation time of 4096*4096 pixels comparison*

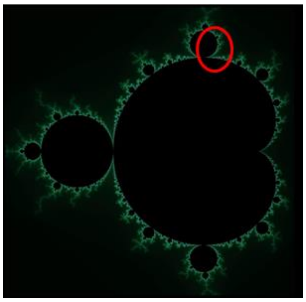


Figure 12: Calculation of partial Mandelbrot set image

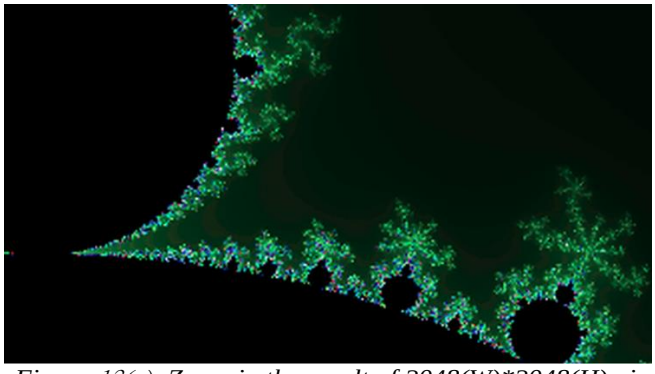


*Figure 13(a): Zoom in the result of 2048(W)*2048(H) pixels*

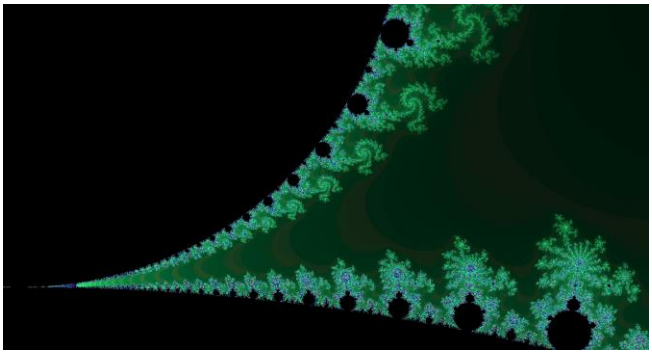
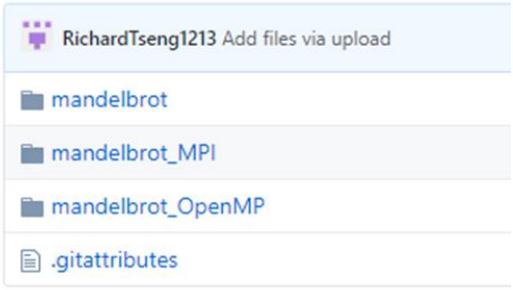


*Figure 13(b): Zoom in the result of 16384(W)*16384(H) pixels*

4. RELATED WORK

This project programing source codes are put on **Github.** Cloud link https://github.com/RichardTseng1213/NCTU-PP.



5. CONCLUSION

The I/O access may waste a lot of time. Hardware limitation (4C8T) is a very importance information to shared memory model. Reducing traffic of data transactions is a very importance part to distributed memory model.

6. REFERENCES

[1] http://dx.doi.org/10.1000/0-000-00000-0.

[2] https://en.wikipedia.org/wiki/Mandelbrot_set

[3] https://en.wikipedia.org/wiki/Fractal

[4] https://en.wikipedia.org/wiki/Benoit_Mandelbrot

[5] https://people.cs.nctu.edu.tw/~ypyou/courses/PP-s19/

[6] https://ent.fanpiece.com/kcmoviedrama/%E6%BB%85-%E5%A2%83-%E2%94%80-%E7%95%B6%E5%B0%8F%E8%AA%AA%E6%B9%AE%E6%BB%85%E5%9C%A8%E9%9B%BB%E5%BD%B1%E4%B8%AD-c1322078.html

[7] http://jakascorner.com/blog/2016/06/omp-for-scheduling.html

[8] http://lib.ck.tp.edu.tw/mathcenter/custom/resource/article9611_7.pdf

[9] Tim Mattson, *"Recent developments in parallel programming: the good, the bad, and the ugly"*, Euro-Par 2013