清華大學計算機系
並行程式設計

Mandelbrot Set

姓名	吳育昕
學號	2011011271
郵箱	ppwwyyxxc@gmail.com
時間	2012 年 8 月

目錄

1 Introduction

1. 編譯: 程式共有四個版本,分別為序列,MPI,OpenMP,pthread.需通過改變環境變數DEFINES分別編譯.源碼中提供了一個指令碼make_all_version可以一次性編譯出四個可執行檔案. 編譯並行版本的環境變數分別為DEFINES=-DUSE_OMP, -DUSE_MPI, -DUSE_PTHREAD,單獨編譯出的可執行檔案為main

```
$ ./make_all_version
make seq verson ...
done
make omp version ...
done
make pthread version ...
done
make mpi version ...
done
$ DEFINES=-DUSE_OMP make
mkdir: created directory `' obj
[dep] ./calculate.cc ...
[dep] ./main.cc ...
[dep] ./Xoutput.cc ...
[dep] ./png_writer.cc ...
[cc] calculate.cc ...
[cc] Xoutput.cc ...
[cc] main.cc ...
[cc] png_writer.cc ...
Linking ...
$
```

2. 命令列參數:

```
$ ./main
Options:
  --nproc=NUM, -n      number of threads(pthread only). number of CPUs by default
  --size=SIZE, -s      size of picture. SIZE is of format `<width>x<height>'
                       eg. `800x800' (default),
  --domain=DOMAIN, -d  plotting domain. DOMAIN is of format
                       `<xmin>,<ymin>,<width>,<height>', where x and y are
```

```
                      the coordinates of left-bottom corner.
                      eg. `-2,-2,4,4` (default).
  --iter=NUM, -i      set max number of iteration of function. 1000 by default.
  --png=FILENAME, -p  save image to png file.
  --X, -x                           use X to show image. can move and zoom image.
  --help, -h          show this help and quit
```

程式支援如下的命令列參數:

`--nproc=NUM`指定 pthread 使用的執行緒數量,對其他多執行緒模式無效.

`--size=SIZE`指定生成圖片的大小.

`--domain=DOMAIN`指定繪圖區域的座標.

`--iter=NUM`指定計算時的迭代次數.

`--png=FILE`將結果輸出到 png 檔案中.

`--X`將圖片用 Xlib 顯示.

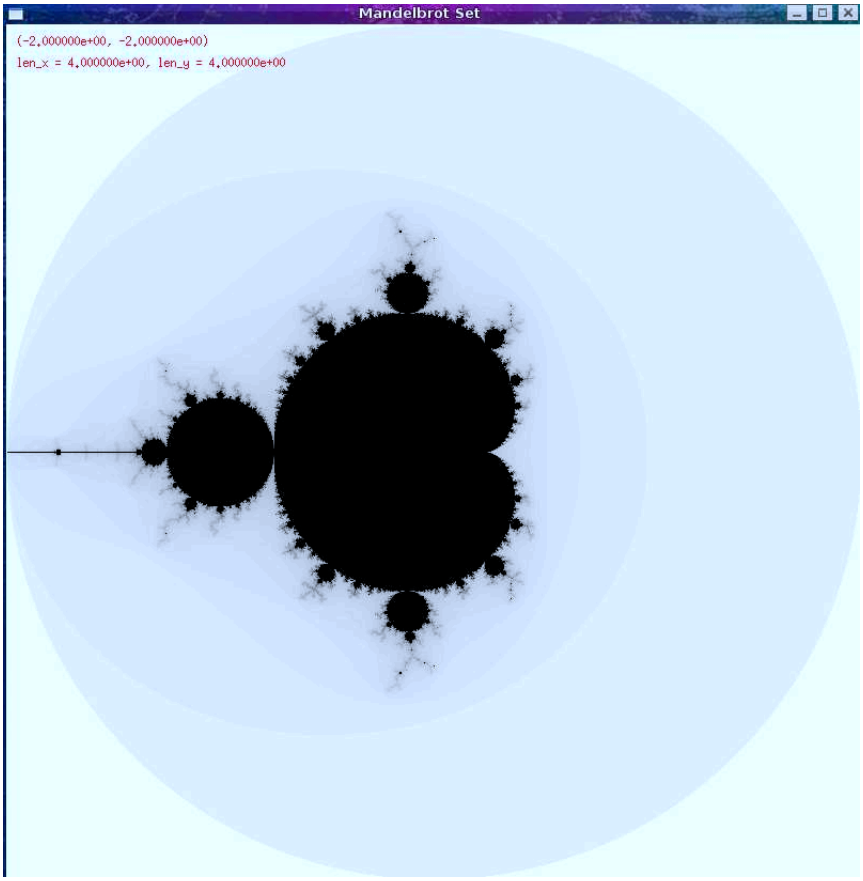
`--help`輸出幫助資訊.

3. 測試運行:

```
$ ./omp -s 2000x2000
0.206523 seconds elapsed
$ ./pthread -s 2000x2000
0.199212 seconds elapsed
$ mpirun -n 4 ./mpi  -s 2000x2000
0.254203 seconds elapsed
$ ./seq -s 2000x2000
0.653799 seconds elapsed
$
```

4. 介面展示:

```
$ ./pthread -x
0.035687 seconds elapsed
$
```



在 X 視窗中,可使用方向鍵移動螢幕,用“=”/“-”進行放大/縮小(由於精度限制,只能放大約 10^{15} 倍),用滑鼠左右鍵改變區域中心再放大/縮小,s 讓程式從`stdin`接收檔名並截圖儲存, c 鍵從`stdin`接受新的迭代次數取值,以觀察同一位置在不同迭代次數下的不同圖形.ESC 鍵退出.

視窗左上角顯示的是視窗左下角座標以及視窗在 x, y 方向上跨越的座標繫上的長度.

2 Algorithms

定義:複平面上使得數列 $\{z_n\} : z_0 = 0; z_{n+1} = z_n^2 + c$ 收斂的全體複數 c 的集合稱為 Mandelbrot Set.

定理:$\{z_n\} : z_0 = 0; z_{n+1} = z_n^2 + c$ 中有 z_k 滿足 $|z_k| > 2$,則 $\{z_n\}$ 必發散.

證明:

首先可以看出,必存在 z_k 滿足 $|z_k| \geq |c|$ 且 $|z_k| > 2$. 當 $|c| \leq 2$ 時顯然存在,因為存在 $|z_k| > 2$. 當 $|c| > 2$ 時,$|z_1| = |c| \geq |c|$ 滿足條件.

於是,選取一個 z_k,滿足 $|z_k| \geq |c|, |z_k| > 2$,那麼一定 $\exists t > 0, |z_k| > 2 + t$.

我們可以得到如下的不等式:

$$|z_{k+1}| = |z_k^2 + c| \geq |z_k^2| - |c| \geq |z_k^2| - |z_k| = (|z_k| - 1)|z_k| > (1 + t)|z_k|$$

迭代下去,有

$$|z_{k+2}| = |z_{k+1}^2 + c| \geq |z_{k+1}^2| - |c| \geq |z_{k+1}^2| - |z_{k+1}| > (1+t)|z_{k+1}| > (1+t)^2|z_k|$$

.....

於是,$|z_{k+n}| > (1+t)^n|z_k|$. 由於 $1+t>1$, 顯然數列發散. 得證.

由上述定理,計算一點 $c = c_r + c_i i$ 是否在 Mandelbrot Set 中,在迭代時一旦發現超出複平面上半徑為 2 的圓,就可直接退出迴圈,大幅減少迭代時間.演算法程式碼如下:

res/src/calculate.cc

```
40      double zr = cr, zi = ci,
41             tr = zr * zr, ti = zi * zi;
42      int k = maxiter;
43      /*
44       *if (tr + ti < 1/16) return 255;
45       */
46      for (; (tr + ti < 4) && (k > 0); k --){
47          zi = 2 * zr * zi + ci;
48          zr = tr - ti + cr;
49          tr = zr * zr;
50          ti = zi * zi;
51      }
```

利用類似手段,還可以得到一個結論:若 $|c| < \frac{1}{4}$, 則數列必發散. 經實驗,這個結論由於只對小範圍資料有效,因此添加到程式中後(註釋掉的程式碼)大多數情形下會影響程式效率.

3 Design

程式通過編譯選項`-DUSE_MPI, -DUSE_OMP, -DUSE_PTHREAD`指定使用的多執行緒庫,通過命令列參數指定演算法迭代深度,繪圖範圍,圖片大小等資訊. 初始化完畢後,程式呼叫`MPI_Wtime()`計時,對指定區域計算,計算結束後將影象資料存放在`short* img`中,輸出所耗時間,如有必要則進行圖形渲染,隨後退出.

當定義了`USE_OMP`宏時,程式呼叫`cal_rectangle_omp()`函數進行計算,函數中迴圈體前有`#pragma omp parallel for schedule(dynamic)`一行, 對下方的 for 迴圈自動進行了動態多執行緒任務排程.

當定義了`USE_PTHREAD`宏時,程式呼叫`cal_rectangle_pth()`函數. 函數創建`nproc`個執行緒,每個執行緒每次領取區域中十列資料的計算任務,並使用一個 mutex 用於記錄當前未領的任務.

當定義了`USE_MPI`宏時,程式呼叫`cal_rectangle_mpi()`函數. root 程序只負責進行任務排程,將任務以 100 列資料為一個單位發送.各程序接收 root 發送的 BEGIN 資訊以及任務的起始列數,便開始計算,計算完成後向 root 程序發送 FINISH 資訊及計算資料. 由於單個資料規模小,因此使用 short 進行儲存以節省通訊成本.

資料計算完畢後,若需要輸出 png 圖片,則呼叫 libpng 庫進行圖片渲染. 若需要實時顯示,則初始化 Xwindow, 將圖片逐點繪製並開始接收鍵盤事件. 接收到縮放,移動按鍵後對繪圖參數進行處理,再呼叫計算函數重新繪製但不再計時,直至收到 ESC 按鍵,程式退出.

在染色方面,經過多次嘗試,設計出了一個關於迭代次數與最大迭代次數的函數代表各點的灰度,這個函數使得集合內的點為純黑色,集合周圍迭代次數較高的點為灰色,但與黑色無法充分接近,迭代次數少的點較白. 這個配色函數可保證即使放大多倍,集合邊界也十分明顯,同時在最大迭代次數改變時常常能找到美麗的圖案.

4 Results and Analysis

各參數對程式運行時間的影響:

1. iter:
   程式在判斷一個點是否屬於集閤中時,對一點迭代 iter 次,iter 越高,所得影象越精確,集合邊界越清晰,所花時間也越長. 同時,由於程式配色函數比較特殊,iter 取值的改變可能使得集合周圍顯示一些美妙圖案.

   迭代次數極少時,集合邊界非常圓滑,隨著迭代次數增加,邊界向內收縮.如圖1

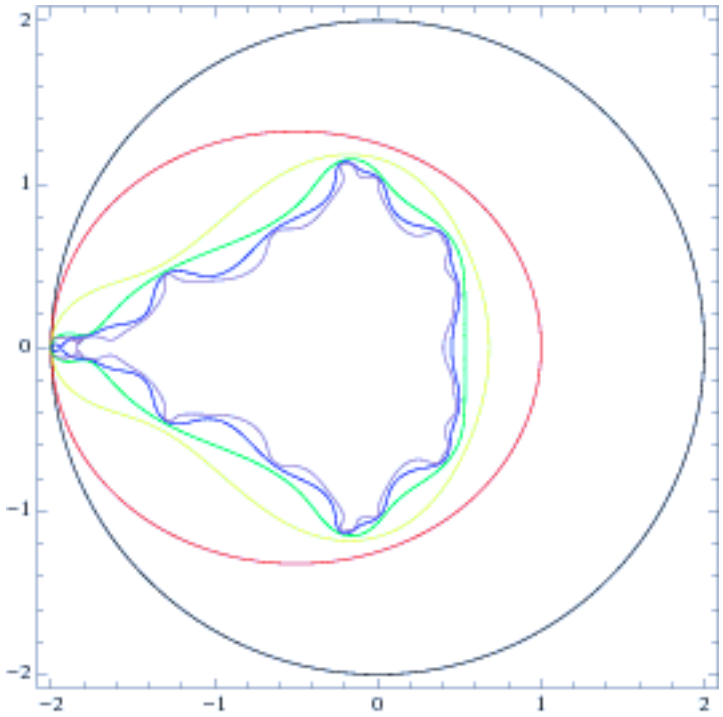


圖 1

```
$ ./pthread -s 2000x2000 -i 1000
0.622379 seconds elapsed
$ ./pthread -s 2000x2000 -i 5000
```

```
2.880889 seconds elapsed
$ ./pthread -s 2000x2000 -i 8000
4.572143 seconds elapsed
$
```

從以上速度測試可以看出,iter 增加到 k 倍時,耗時略小於 k 倍.這是因為對區域中的許多點,經過少量迭代就已經越界.

2. size:

```
$ ./pthread -s 1000x1000
0.157242 seconds elapsed
$ ./pthread -s 2000x2000
0.622366 seconds elapsed
$ ./pthread -s 3000x3000
1.396862 seconds elapsed
$ ./pthread -s 4000x4000
2.478775 seconds elapsed
```

從資料可以看出,耗時的變化大約與點的個數成正比,耗時略少於 k 倍,猜想是多執行緒的啟動耗時,openmp,MPI 也有類似情況.於是增大資料規模:

```
$ ./pthread -s 10000x10000 -i 300
5.636299 seconds elapsed
$ ./pthread -s 10000x20000 -i 300
11.265757 seconds elapsed
$ ./pthread -s 20000x20000 -i 300
22.500245 seconds elapsed
$ ./pthread -s 20000x40000 -i 300
46.199222 seconds elapsed
$ ./pthread -s 30000x40000 -i 300
71.325897 seconds elapsed
```

資料規模變大後,執行緒啟動耗時的影響漸漸可以忽略不計,此時可以看出,資料規模變為 k 倍時,程式耗時略多於 k 倍,這應當是執行緒間鎖定/程序間通訊耗時的結果.

3. domain:
由於區域各點所執行的迭代次數不同,改變區域可能大幅改變程式效率.

```
$ ./pthread -d -5,-5,2,2
0.008872 seconds elapsed
$ ./pthread -d -0.2,-0.2,0.1,0.1
0.974035 seconds elapsed
```

4. nproc:
使用 pthread 加速時,通過命令列參數改變 nproc:

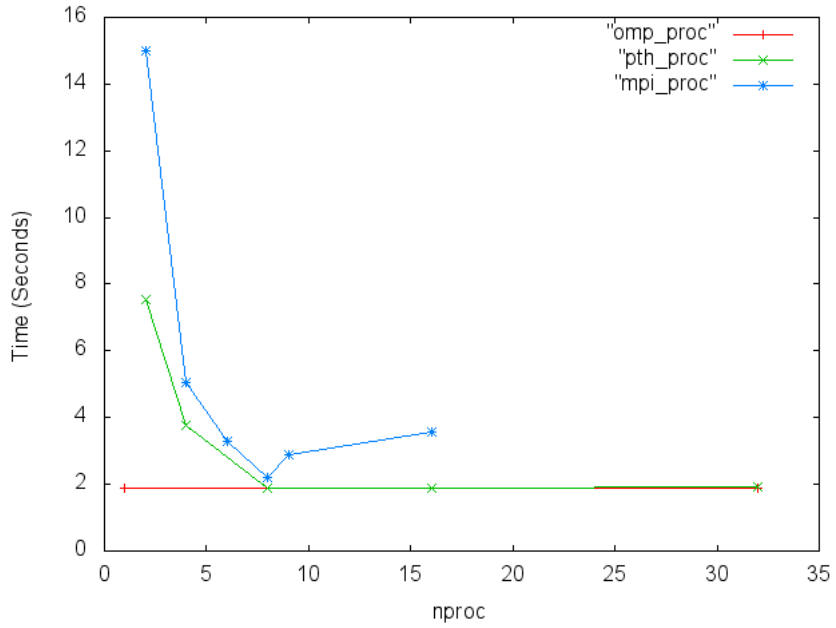
```
$ ./pthread -n 2 -s 5000x5000
7.524309 seconds elapsed
$ ./pthread -n 4 -s 5000x5000
```

```
3.761441 seconds elapsed
$ ./pthread -n 8 -s 5000x5000
1.882019 seconds elapsed
$ ./pthread -n 16 -s 5000x5000
1.894062 seconds elapsed
$ ./pthread -n 32 -s 5000x5000
1.935740 seconds elapsed
```

使用 MPI 加速時,利用 mpirun 改變 nproc:

```
$ mpirun -n 2 ./mpi -s 5000x5000
15.010539 seconds elapsed
$ mpirun -n 4 ./mpi -s 5000x5000
5.037870 seconds elapsed
$ mpirun -n 6 ./mpi -s 5000x5000
3.273679 seconds elapsed
$ mpirun -n 8 ./mpi -s 5000x5000
2.186701 seconds elapsed
$ mpirun -n 9 ./mpi -s 5000x5000
2.895381 seconds elapsed
$ mpirun -n 16 ./mpi -s 5000x5000
3.586846 seconds elapsed
```

```
$ ./omp -s 5000x5000
1.887277 seconds elapsed
```

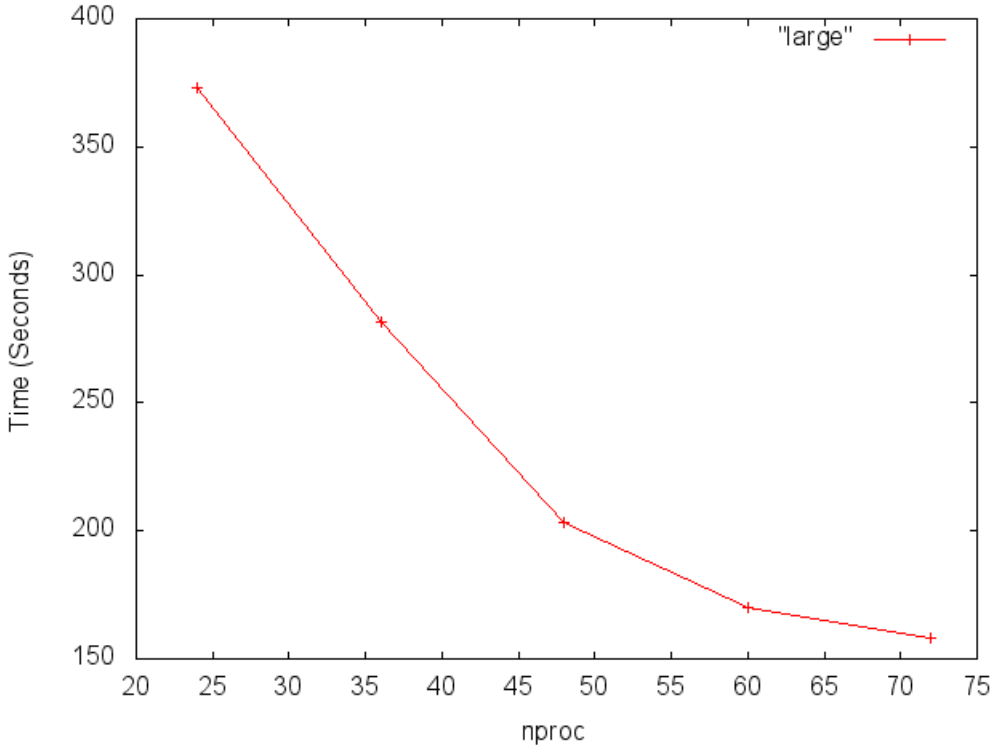


由以上資料(由清華大學 FIT 樓集群計算機生成)可以看出,對於 pthread, 執行緒數等於處理器個數時最佳.對於 MPI,由於此程式有一程序只負責管理任務,因此效率與

程序數關係可能略有不同,無法與 pthread 進行直接比較.但從資料來看仍是 8 個程序時效率最高. 而且由於 MPI 的程序通訊及初始化更為複雜,因此程序數更高時效率會明顯下降. OpenMP 由於設定成自動分配,無法改變執行緒數,但其效率很高.

對於更大規模的資料,只能使用 MPI 進行多節點測試,測試資料如下.

nproc	time
24	373.262603
36	281.326281
48	203.073978
60	169.850154
72	157.953276



測試參數: -s 30000x30000 -i 20000

5 Experience

這次使用三種並行庫分別進行多執行緒程式設計,讓我有很大進步. 首先是對於三種並行庫都有了更深入的瞭解.OpenMP 程式實現上最容易,但對演算法有較大要求,對多執行緒需要共享資訊的情形會難以實現. Pthread 是一個方便的多執行緒庫,程式設計上難度不大,同時也有`mutex_lock`機制使執行緒間可以進行資訊交流. MPI 由於是多程序

的,因此無法共享內存,需要另外使用訊息機制來管理通訊,因此在實現上較為複雜,需要在原始版本上增添不少程式碼,效率也不高. 但這也使得 MPI 程式可以在多個機器上共同運行,適合計算更加巨型的任務.

上次作業我使用 gtkmm 創建 GUI,這次便改用了 Xlib. 相比之下 Xlib 操作簡單,但功能較少. 尤其是在 event 處理上只能採用時刻不停接收的方法.如果有多個事件短時間內發生,會一個接一個處理,對效能十分不利. 例如對於視窗的拉伸操作,就會一次性觸發大量`ConfigureNotify`事件,造成程式多次重新整理. 同時,由於多程序/多執行緒操作內部實現複雜,因此與較複雜的圖形庫共用時也許會有不可預見的後果,而 Xlib 的簡單結構較易於掌控.

另外,這次作業生成的 Mandelbrot Set,是一個經典的分形圖形,將其放大可以找到大量的美麗圖片,使我們在完成作業的同時深刻感受到了數學之美.