實驗五 Multithread-programming using OpenMP

9617145 許晏峻 資工 4C

1. 實驗目的

使用 OpenMP 加速程式效能，並熟悉其開發除錯流程。

2. 步驟過程

1. pi.c

在 for 迴圈外加上 #pragma omp parallel for reduction(+:pi)，並利用 lab2 中的 clock_gettime() 來計算時間。最後和未加上平行化的程式做比較和分析。

2. prime.c

在 work()中的 for 迴圈外加上 #pragma omp parallel for 來進行平行處理，並且在迴圈中的 “prime[total]=i” 前加上#pragma omp critical 來防止 critical section 有 race condition 的問題，並利用 lab2 中的 clock_gettime()來計算時間。最後和未加上平行化的程式做比較和分析。

3. NASA Parallel Benchmark : ep.c

先對環境做設定：

- 編輯 config/suite.def 檔案，將 SP 都改成 EP
- 編輯 config/make.def 將 gcc 參數多加一項 “-pg” 以利之後 profiler 分析程式
- 編輯 Makefile，依照測試的 level (A,B,C,W,S) 改變檔案中的 CLASS 值

首先我先利用 lab2 中使用的 gprof 來對 ep.c 做 sampling 分析，找到花最多時間的地方為 vranlc()，因此進入 EP/ep.c 尋找使用 vranlc()的地方，可以發現唯一不斷使用 vranlc()的地方在一個 for 迴圈中，所以仔細觀察迴圈中變數間的資料相依性。

可以發現幾乎所有變數都會先做初始化，因此在不同次的迴圈中並不會被上一次迴圈影響到，所以可以將這些變數設成 private。

而少數有相依性的是 sx 和 sy，這兩個變數會在迴圈中一直將運算結果做累加的動作，因此這邊就將這兩個變數設成 reduction(+:sx,sy) 來避免 race condition；另外，qq[] 這個陣列似乎也有相依性，也是做累加的動作，而經實驗結果發現並沒有影響到，因此就不多做處理。

最後，在觀察相依性時，因為有使用外部函式，vranlc() 和 ranlc()，這兩個函式是被放在 common/c_randdp.c 中，進入此檔案觀察這兩個函式發現，雖然傳入函式的指標會讓此位址值被改變，但是回到 ep.c 的 for 迴圈中，並沒有在直接使用這些值，因此在這個部分不需要考慮到 race condition 的問題。

綜合以上的觀察，必須在 for 迴圈前加上 #pragma omp parallel for privatie(kk,k,ik,i,t1,t2,t3,t4,x,x1,x2) reductions(+:sx.sy) 來做平行化處理。

最後將各個 level 執行時間和未修改版本的程式執行時間做比較和分析。

3. 數據結果

以下程式執行環境皆為：

OS 為 Linux Ubuntu 10.04.02 LTS x86 64bits

CPU 為 AMD Phenom(tm) II X6 1055T Processor (AMD 6 核心)

Memory 為 8G

1. pi.c

原始程式執行時間：約 11.25556542 秒

```
ychsu-workstation [/home/ychsu/Work/MP/lab5/pi] -ychsu- % ./pi.out
The approximate value of pi = 3.141593
run time in 11.25556542 sec
ychsu-workstation [/home/ychsu/Work/MP/lab5/pi] -ychsu- %
```

平行化後程式執行時間：約 2.184295606 秒

```
ychsu-workstation [/home/ychsu/Work/MP/lab5/pi] -ychsu- % ./pi.out
The approximate value of pi = 3.141593
run time in 2.184295606 sec
ychsu-workstation [/home/ychsu/Work/MP/lab5/pi] -ychsu- %
```

執行時間減少約 80.593639 % (效能增進 80.593639 %)

2. prime.c (以 23 為參數測試)

(為控制截圖大小沒有 printf 出所有質數)

原始程式執行時間：約 10.215261 秒

```
ychsu-workstation [/home/ychsu/Work/MP/lab5/prime] -ychsu- % ./prime.out 23
Number of prime numbers between 2 and 8388608: 564163

run in 10.215261 secs
ychsu-workstation [/home/ychsu/Work/MP/lab5/prime] -ychsu- %
```

平行化後程式執行時間：約 2.673104 秒

```
ychsu-workstation [/home/ychsu/Work/MP/lab5/prime] -ychsu- % ./prime.out 23
Number of prime numbers between 2 and 8388608: 564163

run in 2.673104 secs
ychsu-workstation [/home/ychsu/Work/MP/lab5/prime] -ychsu- %
```

執行時間減少約 73.832250 % (效能增進 73.832250 %)

3. NASA Parallel Benchmark : ep.c

(以下截圖因為 output 大小較大，只取重要部分 output)

- **Class (level) S**

原始程式執行時間：約 2.61 秒

```
EP Benchmark Completed
Class           =                        S
Size            =                       25
Iterations      =                        0
Threads         =                        1
Time in seconds =                     2.61
Mop/s total     =                    12.85
Operation type  = Random numbers generated
Verification    =                SUCCESSFUL
Version         =                      2.3
Compile date    =              31 Mar 2011
```

平行化後程式執行時間：約 1.21 秒

```
EP Benchmark Completed
Class           =                        S
Size            =                       25
Iterations      =                        0
Threads         =                        1
Time in seconds =                     1.21
Mop/s total     =                    27.75
Operation type  = Random numbers generated
Verification    =                SUCCESSFUL
Version         =                      2.3
Compile date    =              01 Apr 2011
```

執行時間減少約 53.639847 % (效能增進 53.639847 %)

- **Class (level) W**

原始程式執行時間：約 5.16 秒

```
EP Benchmark Completed
Class           =                        W
Size            =                       26
Iterations      =                        0
Threads         =                        1
Time in seconds =                     5.16
Mop/s total     =                    12.99
Operation type  = Random numbers generated
Verification    =                SUCCESSFUL
Version         =                      2.3
Compile date    =              01 Apr 2011
```

平行化後程式執行時間：約 2.39 秒

```
EP Benchmark Completed
Class           =                        W
Size            =                       26
Iterations      =                        0
Threads         =                        1
Time in seconds =                     2.39
Mop/s total     =                    28.12
Operation type  = Random numbers generated
Verification    =                SUCCESSFUL
Version         =                      2.3
Compile date    =              01 Apr 2011
```

執行時間減少約 53.682171 % (效能增進 53.682171 %)

- **Class (level) A**

原始程式執行時間：約 41.14 秒

```
EP Benchmark Completed
Class           =                        A
Size            =                       29
Iterations      =                        0
Threads         =                        1
Time in seconds =                    41.24
Mop/s total     =                    13.02
Operation type  = Random numbers generated
Verification    =                SUCCESSFUL
Version         =                      2.3
Compile date    =              01 Apr 2011
```

平行化後程式執行時間：約 22.52 秒

```
EP Benchmark Completed
Class           =                        A
Size            =                       29
Iterations      =                        0
Threads         =                        1
Time in seconds =                    22.52
Mop/s total     =                    23.84
Operation type  = Random numbers generated
Verification    =                SUCCESSFUL
Version         =                      2.3
Compile date    =              01 Apr 2011
```

執行時間減少約 45.392823 % (效能增進 45.392823 %)

- **Class (level) B**

原始程式執行時間：約 164.91 秒

```
EP Benchmark Completed
Class           =                        B
Size            =                       31
Iterations      =                        0
Threads         =                        1
Time in seconds =                   164.91
Mop/s total     =                    13.02
Operation type  = Random numbers generated
Verification    =                SUCCESSFUL
Version         =                      2.3
Compile date    =              01 Apr 2011
```

平行化後程式執行時間：約 76.58 秒

```
EP Benchmark Completed
Class           =                        B
Size            =                       31
Iterations      =                        0
Threads         =                        1
Time in seconds =                    76.58
Mop/s total     =                    28.04
Operation type  = Random numbers generated
Verification    =                SUCCESSFUL
Version         =                      2.3
Compile date    =              01 Apr 2011
```

執行時間減少約 53.562549 % (效能增進 53.562549 %)

- **Class (level) C**

原始程式執行時間：約 659.86 秒

```
EP Benchmark Completed
Class           =                        C
Size            =                       33
Iterations      =                        0
Threads         =                        1
Time in seconds =                   659.86
Mop/s total     =                    13.02
Operation type  = Random numbers generated
Verification    =                SUCCESSFUL
Version         =                      2.3
Compile date    =              01 Apr 2011
```

平行化後程式執行時間：約 338.25 秒

```
EP Benchmark Completed
Class           =                        C
Size            =                       33
Iterations      =                        0
Threads         =                        1
Time in seconds =                   338.25
Mop/s total     =                    25.40
Operation type  = Random numbers generated
Verification    =                SUCCESSFUL
Version         =                      2.3
Compile date    =              01 Apr 2011
```

執行時間減少約 48.739126 % (效能增進 48.739126 %)

4. 結論心得

這次作業用到了之前 lab2 的 gprof 和 clock_gettime()。重點是使用 OpenMP 可以很簡單方便的使用 multi-thread 來做程式平行化執行，由以上幾個小實驗可看出效能都有提升約 50 % 以上，感覺很棒。

而最難的部分莫過於判斷每個變數之間的相依性，是否會有 race condition 的情況發生，並且該怎麼修改才能順利地提升效率，就像上課講的 Andahl's Law 一樣，不是胡亂加上 multi-thread 就可以提升效率，而是要使用在對的地方，否則，還有可能增加 overhead，反而拖慢整個程式的速度，甚至出錯。

在這次實驗中，我也有嘗試對 omp critical 和 omp reductions 做比較，效能可以說是天差地別，很明顯地 reductions 有較好的表現，但相反的 reductions 也有使用時機的限制，所以判斷何時可以用 reductions 來減少 lock 的 overhead 也是寫 OpenMP 應該注意的地方之一。

我想有機會也可以試試 OpenMP 和 pthread 的表現是否效能有所差異。