嵌入式系統軟體設計

Embedded System Software Design

PA1

指導教授: 陳雅淑 教授

學生: M11007308 吳柏翰

Part 1

[Global Scheduling. 10%]

- **Describe how to implement Global scheduling by using pthread. 5%**

首先要定義每個執行緒(thread)計算的範圍，以本例而言，要計算的矩陣大小為 10000，平均分派給 4 個執行緒，所以執行緒各被分配到 2500。

```
    for (int i = 0; i < numThread; i++)
    {
#if (PART == 1)
        /*~~~~~~~~~~~~~Your code(PART1)~~~~~~~~~~~*/
        // For part1, we assign the matrix0 into all threads
        threadSet[i].init(multiResult[0], matrix[0], mask[0]);
        // Set up the calculation range of each thread matrix
        threadSet[i].setStartCalculatePoint(threadSet[i].matrixSize() / numThread * i);
        threadSet[i].setEndCalculatePoint(threadSet[i].matrixSize() / numThread * (i+1));
        /*~~~~~~~~~~~~~~~~~~~END~~~~~~~~~~~~~~~~~~*/
#else
```

pthread 的 pthread_create 函數可以用來建立新的執行緒，並指定子執行緒要執行的函數，子執行緒在建立之後，就會以平行的方式執行，在子執行緒的執行期間，主執行緒還是可以正常執行自己的工作，最後主執行緒再以 pthread_join 函數等待子執行緒執行結束，處理後續收尾的動作。

```
/*~~~~~~~~~~~~~Your code(PART1)~~~~~~~~~~~*/
// Create thread and join
for(int i = 0; i < numThread; i++){
    pthread_create(&threadSet[i]._thread, NULL, threadSet[i].convolution, &threadSet[i]);
}
for(int i = 0; i < numThread; i++){
    pthread_join(threadSet[i]._thread, NULL);
}
/*~~~~~~~~~~~~~~~~~~~END~~~~~~~~~~~~~~~~~~*/
```

若利用多執行緒進行運算，就需要設定 Affinity mask，決定哪個 CPU 可以執行程式，讓系統不要自動排程。如果指定好核心(core)的話就執行 setupCPU-AffinityMask，並更新目前的執行緒是在哪個 core 中及 thread ID，再顯示執行緒資訊。

```
  /*~~~~~~~~~~~~~Your code(PART1)~~~~~~~~~~~*/
  // Set up the affinity mask
  if(obj->core != -1)
    obj->setUpCPUAffinityMask(obj->core);
  obj->cur_core = sched_getcpu();
  obj->PID = syscall(SYS_gettid);

  if(PART != 3){
    pthread_mutex_lock(&count_Mutex);
    obj->printThreadInfo();
    pthread_mutex_unlock(&count_Mutex);
  }
  /*~~~~~~~~~~~~~~~~~~~END~~~~~~~~~~~~~~~~~~*/
```

以下是設定 Affinity mask 的副程式，將 thread 固定在指定的 Core 上。

```
void
Thread::setUpCPUAffinityMask (int core_num)
{
  /*~~~~~~~~~~~~Your code(PART1)~~~~~~~~~~*/
  // Pined the thread to core.
  cpu_set_t mask;
  CPU_ZERO(&mask);
  CPU_SET(core_num, &mask);
  if(sched_setaffinity(0, sizeof(mask), &mask) == -1)
    std::cerr << "Warning: could not set CPU affinity mask" << std::endl;
  /*~~~~~~~~~~~~~~~~~~END~~~~~~~~~~~~~~~~~~*/
}
```

- **Describe how to observe task migration. 5%**

當執行緒目前被某個 core 執行，若和上一次紀錄的 core 不同時，就表示發生 Task migration，就顯示該執行緒是從哪個 core 移動到其他哪個 core。

```
/*~~~~~~~~~~~~Your code(PART1)~~~~~~~~~~*/
// Observe the thread migration
int newCore = sched_getcpu();   // Get index of currently used CPU
if(obj->cur_core != newCore && PART == 1){
  std::cout << "The thread " << obj->_ID << " PID " << obj->PID << " is moved from CPU "
            << obj->cur_core << " to " << newCore << std::endl;
  obj->cur_core = newCore;
}
/*~~~~~~~~~~~~~~~~~~END~~~~~~~~~~~~~~~~~~*/
```

[Partition Scheduling. 5%]

- **Describe how to implement partition scheduling by using pthread.**

Partition scheduling 與 Global scheduling 不同的地方是: Partition scheduling 需要預先將 thread 指定給哪個 core，這裡就將第 i 個 thread 指定給第 i 個 core。

```
/*~~~~~~~~~~~~Your code(PART1)~~~~~~~~~~~*/
// Create thread and join
for(int i = 0; i < numThread; i++){
    if(PART == 1) threadSet[i].setCore(i);
    pthread_create(&threadSet[i]._thread, NULL, threadSet[i].convolution, &threadSet[i]);
}
for(int i = 0; i < numThread; i++){
    pthread_join(threadSet[i]._thread, NULL);
}
/*~~~~~~~~~~~~~~~~~~END~~~~~~~~~~~~~~~~~~*/
```

[Result. 10%]

- **Show the scheduling states of tasks. (You have to show the screenshot result of using the input part1_input.txt)**

(1)Single thread 與 Global multi-thread scheduling 的結果

```
brian@brian-pc:~/Documents/PA1/ESSD_PA1 (2)$ ./part1.out ./input/part1_input.txt
Input File Name : ./input/part1_input.txt
numThread : 4
0.Matrix size : 10000
1.Matrix size : 10000
2.Matrix size : 10000
3.Matrix size : 10000
Workload Utilization : 4

==========Generate Matrix Data==========
Generate Date Spend time : 4.37499

==========Start Single Thread Convolution==========
Single Thread Spend time : 18.6699

==========Start Global Multi-Thread Convolution==========
Thread ID : 1   PID : 190863    Core : 10
Thread ID : 2   PID : 190864    Core : 2
Thread ID : 0   PID : 190862    Core : 1
Thread ID : 3   PID : 190865    Core : 9
The thread 3 PID 190865 is moved from CPU 9 to 3
The thread 0 PID 190862 is moved from CPU 1 to 7
The thread 3 PID 190865 is moved from CPU 3 to 9
The thread 0 PID 190862 is moved from CPU 7 to 1
The thread 1 PID 190863 is moved from CPU 10 to 4
The thread 3 PID 190865 is moved from CPU 9 to 3
The thread 1 PID 190863 is moved from CPU 4 to 10

==============checking==============
Part1 global matrix convolution using global scheduling correct.
Part1 global matrix convolution compute result correct
Global Multi Thread Spend time : 4.63042
```

(2)Partition multi-thread scheduling 的結果

```
==========Start Partition Multi-Thread Convolution==========
Thread ID : 0   PID : 190869    Core : 0
Thread ID : 2   PID : 190871    Core : 2
Thread ID : 3   PID : 190872    Core : 3
Thread ID : 1   PID : 190870    Core : 1

==============checking==============
Part1 partition matrix convolution using parition scheduling correct.
Part1 partition matrix convolution compute result correct
Partition Multi Thread Spend time : 4.46539
```

Part 2

[Partition method Implementation. 10%]

- **Describe how to implement the three different partition methods (First-Fit, Best-Fit, Worst-Fit) in partition scheduling.**

First-fit:優先將執行緒排入較低 index 的核心(core)中，如果第一個核心的使用量已被塞滿或是排入時使用量會大於 1，則接續排入下一個核心，最後顯示排程結果，若有執行緒未能排入，則顯示該執行緒不能被排程。

```
/*~~~~~~~~~~~~~Your code(PART2)~~~~~~~~~~~~*/
// Implement partition first-fit and print result.
for(int i = 0; i < CORE_NUM; i++)
    cpuSet[i].emptyCPU();

for(int i = 0; i < numThread; i++){
    bool isSchedulable = false;
    for(int j = 0; j < CORE_NUM; j++){
        if(cpuSet[j].utilization() + threadSet[i].utilization() <= 1){
            cpuSet[j].push_thread(threadSet[i].ID(), threadSet[i].utilization());
            threadSet[i].setCore(j);
            isSchedulable = true;
            break;
        }
    }
    if(!isSchedulable)
        std::cout << "Thread-" << i << " is not schedulable." << std::endl;
}

for(int i = 0; i < CORE_NUM; i++){
    cpuSet[i].printCPUInformation();
}
/*~~~~~~~~~~~~~~~~~~END~~~~~~~~~~~~~~~~~~*/
```

Best-fit:優先將執行緒排入使用率最高的核心(core)中，如果該核心的使用量已被塞滿或是排入時使用量會大於 1，則接續排入使用量次高的核心，若還是不能排入，則接續和其他核心比較使用量，選出之中使用量最高的核心並排入，最後顯示排程結果，若有執行緒未能排入，則顯示該執行緒不能被排程。

```
/*~~~~~~~~~~~~~Your code(PART2)~~~~~~~~~~~*/
// Implement partition best-fit and print result.
for(int i = 0; i < CORE_NUM; i++)
    cpuSet[i].emptyCPU();

for(int i = 0; i < numThread; i++){
    float maxUtili = 0;
    int maxIdx = -1;
    for(int j = 0; j < CORE_NUM; j++){
        float totalUtili = cpuSet[j].utilization() + threadSet[i].utilization();
        if((totalUtili <= 1) && (totalUtili > maxUtili)){
            maxUtili = totalUtili;
            maxIdx = j;
        }
    }
    if( maxIdx != -1){
        cpuSet[maxIdx].push_thread(threadSet[i].ID(), threadSet[i].utilization());
        threadSet[i].setCore(maxIdx);
    }
    else{
        std::cout << "Thread-" << i << " not schedulable." << std::endl;
    }
}

for(int i = 0; i < CORE_NUM; i++){
    cpuSet[i].printCPUInformation();
}
/*~~~~~~~~~~~~~~~~~~END~~~~~~~~~~~~~~~~~~*/
```

Worst-fit:優先將執行緒排入使用率最低的核心(core)中，如果該核心的使用量已被塞滿或是排入時使用量會大於 1，則接續排入使用量次低的核心，若還是不能排入，則接續和其他核心比較使用量，選出之中使用量最低的核心並排入，最後顯示排程結果，若有執行緒未能排入，則顯示該執行緒不能被排程。

```
/*~~~~~~~~~~~~~Your code(PART2)~~~~~~~~~~~*/
// Implement partition worst-fit and print result.
for(int i = 0; i < CORE_NUM; i++)
    cpuSet[i].emptyCPU();

for(int i = 0; i < numThread; i++){
    float minUtili = 1;
    int minIdx = -1;
    for(int j = 0; j < CORE_NUM; j++){
        float totalUtili = cpuSet[j].utilization() + threadSet[i].utilization();
        if((totalUtili <= 1) && (totalUtili < minUtili)){
            minUtili = totalUtili;
            minIdx = j;
        }
    }
    if( minIdx != -1){
        cpuSet[minIdx].push_thread(threadSet[i].ID(), threadSet[i].utilization());
        threadSet[i].setCore(minIdx);
    }
    else{
        std::cout << "Thread-" << i << " not schedulable." << std::endl;
    }
}

for(int i = 0; i < CORE_NUM; i++){
    cpuSet[i].printCPUInformation();
}
/*~~~~~~~~~~~~~~~~~~END~~~~~~~~~~~~~~~~~~*/
```

[Result. 30%]

- **Show the scheduling states of tasks. (You have to show the screenshot result of using input part2_input_5.txt and part2_input_10.txt)**

(1) part2_input_5.txt 的結果

```
brian@brian-pc:~/Documents/PA1/ESSD_PA1 (2)$ ./part2.out ./input/part2_input_5.txt
Input File Name : ./input/part2_input_5.txt
numThread : 5
0.Matrix size : 5001
1.Matrix size : 5001
2.Matrix size : 5001
3.Matrix size : 5001
4.Matrix size : 5001
Workload Utilization : 2.5005

===========Generate Matrix Data===========
Generate Date Spend time : 1.38209

===========Start Single Thread Convolution===========
Single Thread Spend time : 26.1601
```

First-fit 排程結果

```
==========Partition First-Fit Multi Thread Matrix Multiplication==========
Thread-4 is not schedulable.
Core Number : 0
[ 0, ]
Total Utilization : 0.5001

Core Number : 1
[ 1, ]
Total Utilization : 0.5001

Core Number : 2
[ 2, ]
Total Utilization : 0.5001

Core Number : 3
[ 3, ]
Total Utilization : 0.5001

Thread ID : 2   PID : 225891    Core : 2        Utilization : 0.5001    MatrixSize : 5001
Thread ID : 0   PID : 225889    Core : 0        Utilization : 0.5001    MatrixSize : 5001
Thread ID : 4   PID : 225893    Core : 5        Utilization : 0.5001    MatrixSize : 5001
Thread ID : 3   PID : 225892    Core : 3        Utilization : 0.5001    MatrixSize : 5001
Thread ID : 1   PID : 225890    Core : 1        Utilization : 0.5001    MatrixSize : 5001

===============checking===============
Part2 partiton result correct
Part2 compute result correct
Partition Multi Thread Spend time : 4.47659
```

Best-fit 排程結果

```
==========Partition Best-Fit Multi Thread Matrix Multiplication==========
Thread-4 not schedulable.
Core Number : 0
[ 0, ]
Total Utilization : 0.5001

Core Number : 1
[ 1, ]
Total Utilization : 0.5001

Core Number : 2
[ 2, ]
Total Utilization : 0.5001

Core Number : 3
[ 3, ]
Total Utilization : 0.5001

Thread ID : 0   PID : 225904    Core : 0        Utilization : 0.5001    MatrixSize : 5001
Thread ID : 1   PID : 225905    Core : 1        Utilization : 0.5001    MatrixSize : 5001
Thread ID : 4   PID : 225908    Core : 8        Utilization : 0.5001    MatrixSize : 5001
Thread ID : 2   PID : 225906    Core : 2        Utilization : 0.5001    MatrixSize : 5001
Thread ID : 3   PID : 225907    Core : 3        Utilization : 0.5001    MatrixSize : 5001

===============checking===============
Part2 partiton result correct
Part2 compute result correct
Partition Multi Thread Spend time : 8.29602
```

Worst-fit 排程結果

```
==========Partition Worst-Fit Multi Thread Matrix Multiplication==========
Thread-4 not schedulable.
Core Number : 0
[ 0, ]
Total Utilization : 0.5001

Core Number : 1
[ 1, ]
Total Utilization : 0.5001

Core Number : 2
[ 2, ]
Total Utilization : 0.5001

Core Number : 3
[ 3, ]
Total Utilization : 0.5001

Thread ID : 3   PID : 225915    Core : 3        Utilization : 0.5001    MatrixSize : 5001
Thread ID : 1   PID : 225913    Core : 1        Utilization : 0.5001    MatrixSize : 5001
Thread ID : 0   PID : 225912    Core : 0        Utilization : 0.5001    MatrixSize : 5001
Thread ID : 4   PID : 225916    Core : 8        Utilization : 0.5001    MatrixSize : 5001
Thread ID : 2   PID : 225914    Core : 2        Utilization : 0.5001    MatrixSize : 5001

===============checking===============
Part2 partiton result correct
Part2 compute result correct
Partition Multi Thread Spend time : 8.28422
```

(2) part2_input_10.txt 的結果

```
brian@brian-pc:~/Documents/PA1/ESSD_PA1 (2)$ ./part2.out ./input/part2_input_10.txt
Input File Name : ./input/part2_input_10.txt
numThread : 10
0.Matrix size : 5581
1.Matrix size : 6052
2.Matrix size : 2293
3.Matrix size : 3223
4.Matrix size : 4206
5.Matrix size : 1774
6.Matrix size : 4111
7.Matrix size : 2427
8.Matrix size : 4430
9.Matrix size : 3100
Workload Utilization : 3.7197

===========Generate Matrix Data===========
Generate Date Spend time : 1.71395

===========Start Single Thread Convolution===========
Single Thread Spend time : 32.7114
```

First-fit 排程結果

```
===========Partition First-Fit Multi Thread Matrix Multiplication===========
Core Number : 0
[ 0, 2, 5, ]
Total Utilization : 0.9648

Core Number : 1
[ 1, 3, ]
Total Utilization : 0.9275

Core Number : 2
[ 4, 6, ]
Total Utilization : 0.8317

Core Number : 3
[ 7, 8, 9, ]
Total Utilization : 0.9957

Thread ID : 0   PID : 225982    Core : 0        Utilization : 0.5581    MatrixSize : 5581
Thread ID : 4   PID : 225986    Core : 2        Utilization : 0.4206    MatrixSize : 4206
Thread ID : 1   PID : 225983    Core : 1        Utilization : 0.6052    MatrixSize : 6052
Thread ID : 7   PID : 225989    Core : 3        Utilization : 0.2427    MatrixSize : 2427
Thread ID : 8   PID : 225990    Core : 3        Utilization : 0.443     MatrixSize : 4430
Thread ID : 3   PID : 225985    Core : 1        Utilization : 0.3223    MatrixSize : 3223
Thread ID : 5   PID : 225987    Core : 0        Utilization : 0.1774    MatrixSize : 1774
Thread ID : 6   PID : 225988    Core : 2        Utilization : 0.4111    MatrixSize : 4111
Thread ID : 2   PID : 225984    Core : 0        Utilization : 0.2293    MatrixSize : 2293
Thread ID : 9   PID : 225991    Core : 3        Utilization : 0.31      MatrixSize : 3100

===============checking===============
Part2 partiton result correct
Part2 compute result correct
Partition Multi Thread Spend time : 8.10419
```

Best-fit 排程結果

```
==========Partition Best-Fit Multi Thread Matrix Multiplication==========
Thread-9 not schedulable.
Core Number : 0
[ 0, 3, ]
Total Utilization : 0.8804

Core Number : 1
[ 1, 2, ]
Total Utilization : 0.8345

Core Number : 2
[ 4, 5, 7, ]
Total Utilization : 0.8407

Core Number : 3
[ 6, 8, ]
Total Utilization : 0.8541

Thread ID : 0   PID : 225993    Core : 0        Utilization : 0.5581    MatrixSize : 5581
Thread ID : 2   PID : 225995    Core : 1        Utilization : 0.2293    MatrixSize : 2293
Thread ID : 6   PID : 225999    Core : 3        Utilization : 0.4111    MatrixSize : 4111
Thread ID : 4   PID : 225997    Core : 2        Utilization : 0.4206    MatrixSize : 4206
Thread ID : 8   PID : 226001    Core : 3        Utilization : 0.443     MatrixSize : 4430
Thread ID : 5   PID : 225998    Core : 2        Utilization : 0.1774    MatrixSize : 1774
Thread ID : 1   PID : 225994    Core : 1        Utilization : 0.6052    MatrixSize : 6052
Thread ID : 3   PID : 225996    Core : 0        Utilization : 0.3223    MatrixSize : 3223
Thread ID : 7   PID : 226000    Core : 2        Utilization : 0.2427    MatrixSize : 2427
Thread ID : 9   PID : 226002    Core : 3        Utilization : 0.31      MatrixSize : 3100

===============checking===============
Part2 partiton result correct
Part2 compute result correct
Partition Multi Thread Spend time : 7.82327
```

Worst-fit 排程結果

```
==========Partition Worst-Fit Multi Thread Matrix Multiplication==========
Thread-8 not schedulable.
Core Number : 0
[ 0, 7, ]
Total Utilization : 0.8008

Core Number : 1
[ 1, 9, ]
Total Utilization : 0.9152

Core Number : 2
[ 2, 4, ]
Total Utilization : 0.6499

Core Number : 3
[ 3, 5, 6, ]
Total Utilization : 0.9108

Thread ID : 0   PID : 226004    Core : 0        Utilization : 0.5581    MatrixSize : 5581
Thread ID : 1   PID : 226005    Core : 1        Utilization : 0.6052    MatrixSize : 6052
Thread ID : 3   PID : 226007    Core : 3        Utilization : 0.3223    MatrixSize : 3223
Thread ID : 2   PID : 226006    Core : 2        Utilization : 0.2293    MatrixSize : 2293
Thread ID : 5   PID : 226009    Core : 3        Utilization : 0.1774    MatrixSize : 1774
Thread ID : 4   PID : 226008    Core : 2        Utilization : 0.4206    MatrixSize : 4206
Thread ID : 7   PID : 226011    Core : 0        Utilization : 0.2427    MatrixSize : 2427
Thread ID : 9   PID : 226013    Core : 1        Utilization : 0.31      MatrixSize : 3100
Thread ID : 6   PID : 226010    Core : 3        Utilization : 0.4111    MatrixSize : 4111
Thread ID : 8   PID : 226012    Core : 3        Utilization : 0.443     MatrixSize : 4430

===============checking===============
Part2 partiton result correct
Part2 compute result correct
Partition Multi Thread Spend time : 8.46305
```

Part 3

[Scheduler Implementation. 10%]

- **Describe how to implement the scheduler setting in partition scheduling.(FIFO with FF, RR with FF)**

  一開始初始化 thread 後，設定 thread 的排程方式是 FIFO 或 Round-Robin。

```
#if (PART == 3)
        /*~~~~~~~~~~~~~Your code(PART3)~~~~~~~~~~~~*/
        // Set the scheduling policy for thread.
        if(SCHEDULING == SCHED_FIFO)
            threadSet[i].setSchedulingPolicy(SCHED_FIFO);
        else if(SCHEDULING == SCHED_RR)
            threadSet[i].setSchedulingPolicy(SCHED_RR);
        /*~~~~~~~~~~~~~~~~~~~~END~~~~~~~~~~~~~~~~~~~~*/
#endif
```

利用 Linux 支援的排程器，像是 FIFO 和 RR，再使用 sched_setscheduler 函數設定排程規則。

```
void
Thread::setUpScheduler()
{
  /*~~~~~~~~~~~~~Your code(PART3)~~~~~~~~~~~~*/
  // Set up the scheduler for current thread
  if(schedulingPolicy() == SCHED_FIFO){
    struct sched_param sp;
    sp.sched_priority = sched_get_priority_max(SCHED_FIFO);
    int ret = sched_setscheduler(0, SCHED_FIFO, &sp);
    if(ret == -1)
      std::cerr << "An error occurs when setting scheduler." << std::endl;
  }
  else if(schedulingPolicy() == SCHED_RR){
    struct sched_param sp;
    sp.sched_priority = sched_get_priority_max(SCHED_RR);
    int ret = sched_setscheduler(0, SCHED_RR, &sp);
    if(ret == -1)
      std::cerr << "An error occurs when setting scheduler." << std::endl;
  }
  /*~~~~~~~~~~~~~~~~~~~~END~~~~~~~~~~~~~~~~~~~~*/
}
```

顯示 core-0 的資訊，以及發生 context switch 的情況。

```
#if (PART == 3)
    /*~~~~~~~~~~~~~Your code(PART3)~~~~~~~~~~~~*/
    /* Observe the execute thread on core-0 */
    if(obj->core == 0){
      if(current_PID == -1){
        std::cout << "core-0 start from " << obj->PID << std::endl;
        current_PID = obj->PID;
      }
      else if(current_PID != obj->PID){
        std::cout << "Core-0 context switch from " << current_PID << " to " << obj->PID << std::endl;
        current_PID = obj->PID;
      }
    }
    /*~~~~~~~~~~~~~~~~~~~~END~~~~~~~~~~~~~~~~~~~~*/
#endif
  }
```

[Result. 10%]

- **Show the process execution states of tasks. (You have to show the screen-shot result of using input part3_input.txt)**

(1)part3_input.txt (Round-Robin)的結果

```
brian@brian-pc:~/Documents/PA1/ESSD_PA1 (2)$ sudo ./part3_rr.out ./input/part3_input.txt
[sudo] password for brian:
Input File Name : ./input/part3_input.txt
numThread : 10
0.Matrix size : 5581
1.Matrix size : 6052
2.Matrix size : 2293
3.Matrix size : 3223
4.Matrix size : 4206
5.Matrix size : 1774
6.Matrix size : 4111
7.Matrix size : 2427
8.Matrix size : 4430
9.Matrix size : 3100
Workload Utilization : 3.7197

==========Generate Matrix Data==========
Generate Date Spend time : 1.63141

==========Start Single Thread Convolution==========
Single Thread Spend time : 30.8152
```

```
===========Partition First-Fit Multi Thread Matrix Multiplication===========
Core Number : 0
[ 0, 2, 5, ]
Total Utilization : 0.9648

Core Number : 1
[ 1, 3, ]
Total Utilization : 0.9275

Core Number : 2
[ 4, 6, ]
Total Utilization : 0.8317

Core Number : 3
[ 7, 8, 9, ]
Total Utilization : 0.9957

core-0 start from 191346
Core-0 context switch from 191346 to 191348
Core-0 context switch from 191348 to 191351
Core-0 context switch from 191351 to 191346
Core-0 context switch from 191346 to 191348
Core-0 context switch from 191348 to 191351
Core-0 context switch from 191351 to 191346
Core-0 context switch from 191346 to 191348
Core-0 context switch from 191348 to 191351
Core-0 context switch from 191351 to 191346
Core-0 context switch from 191346 to 191348
Core-0 context switch from 191348 to 191351
Core-0 context switch from 191351 to 191346
Core-0 context switch from 191346 to 191348
Core-0 context switch from 191348 to 191351
Core-0 context switch from 191351 to 191346
Core-0 context switch from 191346 to 191348
Core-0 context switch from 191348 to 191351
Core-0 context switch from 191351 to 191346
Core-0 context switch from 191346 to 191348
Core-0 context switch from 191348 to 191351
Core-0 context switch from 191351 to 191346
Core-0 context switch from 191346 to 191348
Core-0 context switch from 191348 to 191346
Core-0 context switch from 191346 to 191348
Core-0 context switch from 191348 to 191346
Core-0 context switch from 191346 to 191348
Core-0 context switch from 191348 to 191346
Core-0 context switch from 191346 to 191348
Core-0 context switch from 191348 to 191346
Core-0 context switch from 191346 to 191348
Core-0 context switch from 191348 to 191346

===============checking===============
Part3 change scheduler correct
Part3 compute result correct
Partition Multi Thread Spend time : 9.94043
```

```
===========Partition Best-Fit Multi Thread Matrix Multiplication===========
Thread-9 not schedulable.
Core Number : 0
[ 0, 3, ]
Total Utilization : 0.8804

Core Number : 1
[ 1, 2, ]
Total Utilization : 0.8345

Core Number : 2
[ 4, 5, 7, ]
Total Utilization : 0.8407

Core Number : 3
[ 6, 8, ]
Total Utilization : 0.8541

Core-0 context switch from 191346 to 191357
Core-0 context switch from 191357 to 191360
Core-0 context switch from 191360 to 191357
Core-0 context switch from 191357 to 191360
Core-0 context switch from 191360 to 191357
Core-0 context switch from 191357 to 191360
Core-0 context switch from 191360 to 191357
Core-0 context switch from 191357 to 191360
Core-0 context switch from 191360 to 191357
Core-0 context switch from 191357 to 191360
Core-0 context switch from 191360 to 191357
Core-0 context switch from 191357 to 191360
Core-0 context switch from 191360 to 191357
Core-0 context switch from 191357 to 191360
Core-0 context switch from 191360 to 191357
Core-0 context switch from 191357 to 191360
Core-0 context switch from 191360 to 191357
Core-0 context switch from 191357 to 191360
Core-0 context switch from 191360 to 191357
Core-0 context switch from 191357 to 191360
Core-0 context switch from 191360 to 191357
Core-0 context switch from 191357 to 191360
Core-0 context switch from 191360 to 191357
Core-0 context switch from 191357 to 191360
Core-0 context switch from 191360 to 191357
Core-0 context switch from 191357 to 191360
Core-0 context switch from 191360 to 191357
Core-0 context switch from 191357 to 191360
Core-0 context switch from 191360 to 191357
Core-0 context switch from 191357 to 191360
Core-0 context switch from 191360 to 191357
Core-0 context switch from 191357 to 191360
Core-0 context switch from 191360 to 191357
Core-0 context switch from 191357 to 191360
Core-0 context switch from 191360 to 191357
Core-0 context switch from 191357 to 191360
Core-0 context switch from 191360 to 191357
Core-0 context switch from 191357 to 191360
Core-0 context switch from 191360 to 191357
Core-0 context switch from 191357 to 191360
Core-0 context switch from 191360 to 191357
Core-0 context switch from 191357 to 191360
Core-0 context switch from 191360 to 191357
Core-0 context switch from 191357 to 191360
Core-0 context switch from 191360 to 191357

===============checking===============
Part3 change scheduler correct
Part3 compute result correct
Partition Multi Thread Spend time : 9.48499
```

```
===========Partition Worst-Fit Multi Thread Matrix Multiplication===========
Thread-8 not schedulable.
Core Number : 0
[ 0, 7, ]
Total Utilization : 0.8008

Core Number : 1
[ 1, 9, ]
Total Utilization : 0.9152

Core Number : 2
[ 2, 4, ]
Total Utilization : 0.6499

Core Number : 3
[ 3, 5, 6, ]
Total Utilization : 0.9108

Core-0 context switch from 191357 to 191368
Core-0 context switch from 191368 to 191375
Core-0 context switch from 191375 to 191368
Core-0 context switch from 191368 to 191375
Core-0 context switch from 191375 to 191368
Core-0 context switch from 191368 to 191375
Core-0 context switch from 191375 to 191368
Core-0 context switch from 191368 to 191375
Core-0 context switch from 191375 to 191368
Core-0 context switch from 191368 to 191375
Core-0 context switch from 191375 to 191368
Core-0 context switch from 191368 to 191375
Core-0 context switch from 191375 to 191368
Core-0 context switch from 191368 to 191375
Core-0 context switch from 191375 to 191368
Core-0 context switch from 191368 to 191375
Core-0 context switch from 191375 to 191368
Core-0 context switch from 191368 to 191375
Core-0 context switch from 191375 to 191368
Core-0 context switch from 191368 to 191375
Core-0 context switch from 191375 to 191368
Core-0 context switch from 191368 to 191375
Core-0 context switch from 191375 to 191368
Core-0 context switch from 191368 to 191375
Core-0 context switch from 191375 to 191368

==============checking==============
Part3 change scheduler correct
Part3 compute result correct
Partition Multi Thread Spend time : 10.2608
```

(1)part3_input.txt (FIFO)的結果

```
brian@brian-pc:~/Documents/PA1/ESSD_PA1 (2)$ sudo ./part3_fifo.out ./input/part3_input.txt
Input File Name : ./input/part3_input.txt
numThread : 10
0.Matrix size : 5581
1.Matrix size : 6052
2.Matrix size : 2293
3.Matrix size : 3223
4.Matrix size : 4206
5.Matrix size : 1774
6.Matrix size : 4111
7.Matrix size : 2427
8.Matrix size : 4430
9.Matrix size : 3100
Workload Utilization : 3.7197

===========Generate Matrix Data===========
Generate Date Spend time : 1.63363

===========Start Single Thread Convolution===========
Single Thread Spend time : 30.8406
```

```
==========Partition First-Fit Multi Thread Matrix Multiplication==========
Core Number : 0
[ 0, 2, 5, ]
Total Utilization : 0.9648

Core Number : 1
[ 1, 3, ]
Total Utilization : 0.9275

Core Number : 2
[ 4, 6, ]
Total Utilization : 0.8317

Core Number : 3
[ 7, 8, 9, ]
Total Utilization : 0.9957

core-0 start from 191513
Core-0 context switch from 191513 to 191515
Core-0 context switch from 191515 to 191518

===============checking===============
Part3 change scheduler correct
Part3 compute result correct
Partition Multi Thread Spend time : 9.86781
```

```
==========Partition Best-Fit Multi Thread Matrix Multiplication==========
Thread-9 not schedulable.
Core Number : 0
[ 0, 3, ]
Total Utilization : 0.8804

Core Number : 1
[ 1, 2, ]
Total Utilization : 0.8345

Core Number : 2
[ 4, 5, 7, ]
Total Utilization : 0.8407

Core Number : 3
[ 6, 8, ]
Total Utilization : 0.8541

Core-0 context switch from 191518 to 191523
Core-0 context switch from 191523 to 191526

===============checking===============
Part3 change scheduler correct
Part3 compute result correct
Partition Multi Thread Spend time : 9.61631
```

```
==========Partition Worst-Fit Multi Thread Matrix Multiplication==========
Thread-8 not schedulable.
Core Number : 0
[ 0, 7, ]
Total Utilization : 0.8008

Core Number : 1
[ 1, 9, ]
Total Utilization : 0.9152

Core Number : 2
[ 2, 4, ]
Total Utilization : 0.6499

Core Number : 3
[ 3, 5, 6, ]
Total Utilization : 0.9108

Core-0 context switch from 191526 to 191536
Core-0 context switch from 191536 to 191543

===============checking===============
Part3 change scheduler correct
Part3 compute result correct
Partition Multi Thread Spend time : 10.2362
```

Discussion

- **Analyze and compare the response time of the program, with single thread and multi-thread using in part 1 and part 2. (Including Single, Global, First-Fit, Best-Fit, Worst-Fit) 5%**

(1) part1_input.txt 的測試結果

	Single	Global	Partition
Time (sec)	18.6699	4.6304	4.4654

(2) part2_input_5.txt 的測試結果

	Single	First-fit	Best-fit	Worst-fit
Time (sec)	26.1601	4.4766	8.2960	8.2842

(3) part2_input_10.txt 的測試結果

	Single	First-fit	Best-fit	Worst-fit
Time (sec)	32.7114	8.1042	7.8233	8.4631

從 Part1 的結果可得知在速度上 Partition> Global> Single，Multi-threading 的運行速度遠快於 Single-threading，因為多個執行緒同時運算一定比單個執行緒來的更有效率，另外在運算時間上 Partition scheduling 比 Global scheduling 快一些，因為 Global scheduling 會將所有 task 先存放在 queue，再依序分配給 CPU 執行; 而 Partition scheduling 會直接將 task 分配給 CPU 執行。

從 Part2 的第一項結果可看出 First-fit 的速度是最快，但第二項結果顯示 Best-fit 是最快，理論上 First-fit 應該是最快，因為在演算法上較為單純，每次在排入 task 時都是從較小 index 的 CPU 開始排入，Best-fit 和 Worst-fit 就需要比較各個 CPU 的利用率再排入 task，與 First-fit 相比會慢一些。但是在經過反覆測試後，我發現這三種排法所花費的時間都不固定，有時是 First-fit 最快，有時是 Best-fit 或 Worst-fit，推測應該是有某個 thread 沒辦法排入，所以少了一點排入所花費的時間。

- **Analyze and compare the characteristic of the three different partition methods (First-Fit, Best-Fit, Worst-Fit) 5%**

First-fit: 優先將執行緒排入較低 index 的核心(core)中，如果第一個核心的使用量已被塞滿或是排入時使用量會大於 1，則接續排入下一個核心。優點是執行速度最快，且演算法簡單容易實現。

Best-fit:優先將執行緒排入使用率最高的核心(core)中，如果該核心的使用量已被塞滿或是排入時使用量會大於 1，則接續排入使用量次高的核心，若還是不能排入，則接續和其他核心比較使用量，選出之中使用量最高的核心並排入。特點是可集中使用某些 GPU，讓剩餘沒有被排入工作的 CPU 降低工作量。

Worst-fit:優先將執行緒排入使用率最低的核心(core)中，如果該核心的使用量已被塞滿或是排入時使用量會大於 1，則接續排入使用量次低的核心，若還是不能排入，則接續和其他核心比較使用量，選出之中使用量最低的核心並排入。

特點是平均使用每個 CPU，其利用率最平均，對 CPU 而言功耗最少。

- **Analyze and compare the response time of the program, with two different schedulers. (FIFO with FF, RR with FF) 5%**

	RR with FF	FIFO with FF
Time (sec)	9.9404	9.8678

由測試結果可發現 FIFO 比 RR 快一些，RR 需要一直做 context switch，要花費較長的時間處理 context switch;而 FIFO 是要等待上一個 thread 執行完才會執行下一個 thread，若前一個 thread 長時間佔住 CPU，就會導致下一個 thread 遲遲無法執行，可能花更多時間在等待上。