嵌入式系統軟體設計
Embedded System Software Design

PA1

指導教授: 陳雅淑 教授

課程學生: M10907305 陳俊億

● Part 1

[Global Scheduling. 10%]

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

```
System::System(char* input_file)
{
    loadInput(input_file); // Set up threadSet, singleResult, multiResult, and matrix

    for (int i = 0; i<numThread; i++) {
#if (PART == 1)
        // Set the singleResult, multResult, and matrix to thread.
        threadSet[i].initialThread(singleResult[0], multiResult[0], matrix[0]);
        /*~~~~~~~~~~~~Your code(PART1)~~~~~~~~~~~*/
        // Set up the calculate range of matrix.
        threadSet[i].setStartCalculatePoint(i*threadSet[i].matrixSize()/numThread);
        threadSet[i].setEndCalculatePoint((i+1)*threadSet[i].matrixSize()/numThread);
        /*~~~~~~~~~~~~~~~~~~END~~~~~~~~~~~~~~~~~~*/
#else
        // Set the singleResult, multResult, and matrix to thread.
        threadSet[i].initialThread(singleResult[i], multiResult[i], matrix[i]);
#endif

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

Fig1. Global Scheduling code-1

一開始會先使用 threadset 先定義每個，thread 的計算的範圍在哪。

```
/*~~~~~~~~~~~~Your code(PART1)~~~~~~~~~~~*/
// Create thread and join
pthread_create(&threadSet[0].pthreadThread,NULL, threadSet[0].matrixMultiplication,&threadSet[0]);
pthread_create(&threadSet[1].pthreadThread,NULL, threadSet[1].matrixMultiplication,&threadSet[1]);
pthread_create(&threadSet[2].pthreadThread,NULL, threadSet[2].matrixMultiplication,&threadSet[2]);
pthread_create(&threadSet[3].pthreadThread,NULL, threadSet[3].matrixMultiplication,&threadSet[3]);
pthread_join(threadSet[0].pthreadThread,NULL);
pthread_join(threadSet[1].pthreadThread,NULL);
pthread_join(threadSet[2].pthreadThread,NULL);
pthread_join(threadSet[3].pthreadThread,NULL);
/*~~~~~~~~~~~~~~~~~~END~~~~~~~~~~~~~~~~~~*/
```

Fig2. Global Scheduling code-1.

然後使用 pthread_create 將每個 thread 與指定的 function 連接去執行，執行完之後再透過 join 結束此 thread。

➢ Describe how to observe task migration. 5%

因為 global 沒有綁住 core，所以電腦會有自定義的排程將 thread 跳到電腦的排程規則裡，一開始先定義每一個 thread 一開始使用的 core，並把數值存入每個 thread 的 core 裡，再有不同 core 執行 thread 時就會顯示，原本的 core 跳到另一個 core 去執行。

```
/*~~~~~~~~~~~~Your code(PART1)~~~~~~~~~~~*/
// Observe the thread migration
if(PART !=3){
    // pthread_mutex_lock(&count_Mutex);
    if(obj->core != sched_getcpu() & PART ==1){
        printf("The thread %d PID %d is moved from CPU %d to %d\n",obj->_ID,obj->PID,obj->core,sched_getcpu());
        obj->core =sched_getcpu();
    }
    // pthread_mutex_unlock(&count_Mutex);
}
```

Fig3. Observe task migration code.

[Partition Scheduling. 5%]

- Describe how to implement partition scheduling by using pthread.

這邊要先自定義每個 thread 的 core 並需固定，然後再 pthread_create 和 join。

```
/*~~~~~~~~~~~~Your code(PART1)~~~~~~~~~~~*/
// Set thread execute core.
// Create thread and join.
if(PART == 1){
    threadSet[0].setThreadCore(0);
    threadSet[1].setThreadCore(1);
    threadSet[2].setThreadCore(2);
    threadSet[3].setThreadCore(3);
    pthread_create(&threadSet[0].pthreadThread,NULL, threadSet[0].matrixMultiplication,&threadSet[0]);
    pthread_create(&threadSet[1].pthreadThread,NULL, threadSet[1].matrixMultiplication,&threadSet[1]);
    pthread_create(&threadSet[2].pthreadThread,NULL, threadSet[2].matrixMultiplication,&threadSet[2]);
    pthread_create(&threadSet[3].pthreadThread,NULL, threadSet[3].matrixMultiplication,&threadSet[3]);
    pthread_join(threadSet[0].pthreadThread,NULL);
    pthread_join(threadSet[1].pthreadThread,NULL);
    pthread_join(threadSet[2].pthreadThread,NULL);
    pthread_join(threadSet[3].pthreadThread,NULL);
```

Fig4. Partition scheduling code-1.

上面是只有定義 core 的參數，thread 的執行 core 並未固定，在這邊透過判別是將定義的 core 透過 setUpCPUAffinityMask function，透過 cpu_set 將 thread 固定在指定的 core 上，最後 printf 所要的數據。

```
Thread::matrixMultiplication(void* args)
{
    Thread *obj = (Thread*)args;
#if (PART == 3)
    obj->setUpScheduler();
#endif
    /*~~~~~~~~~~~~Your code(PART1)~~~~~~~~~~~*/
    // Set up the affinity mask
    // pthread_mutex_lock(&count_Mutex);
    if(obj->setCore == -1 ){
        obj->core = sched_getcpu();
        obj->PID = syscall(SYS_gettid);
        if(PART != 3){
            obj->printInformation();
        }

    }
    else{
        obj->setUpCPUAffinityMask(obj->setCore);
        obj->core = sched_getcpu();
        obj->PID = syscall(SYS_gettid);
        if(PART != 3){
            obj->printInformation();
        }

    }
    // pthread_mutex_unlock(&count_Mutex);
    /*~~~~~~~~~~~~~~~~~~END~~~~~~~~~~~~~~~~~~*/
```

Fig5. Partition scheduling code-2.

```
void
Thread::setUpCPUAffinityMask(int cpu_num)
{
    /*~~~~~~~~~~~~Your code(PART1)~~~~~~~~~~~*/
    // Pined the thread to core.
    cpu_set_t set;
    CPU_ZERO(&set);
    CPU_SET(cpu_num,&set);
    sched_setaffinity(0,sizeof(set),&set);
    /*~~~~~~~~~~~~~~~~~~END~~~~~~~~~~~~~~~~~~*/
}
```

Fig6. Partition scheduling code-3.

[Result. 10%]

- Show the scheduling states of tasks. (You have to show the screenshot result of using the input part1_Input.txt)

(1) Screenshot result of using the input part1_Input.txt

```
chen@chen-VirtualBox:~/桌面/PA1$ ./part1.out ./input/part1_Input.txt
Input File Name : ./input/part1_Input.txt
numThread : 4

==========Start Single Thread Matrix Multiplication==========
Thread ID : 0   PID : 9302      Core : 0
Single Thread Spend time : 82.8834

==========Start Global Multi-Thread Matrix Multiplication==========
Thread ID : 0   PID : 9309      Core : 3
Thread ID : 1   PID : 9310      Core : 4
Thread ID : 2   PID : 9311      Core : 6
Thread ID : 3   PID : 9312      Core : 7
The thread 3 PID 9312 is moved from CPU 7 to 0
The thread 0 PID 9309 is moved from CPU 3 to 1
The thread 2 PID 9311 is moved from CPU 6 to 2
The thread 1 PID 9310 is moved from CPU 4 to 3
The thread 0 PID 9309 is moved from CPU 1 to 5
The thread 0 PID 9309 is moved from CPU 5 to 1
The thread 2 PID 9311 is moved from CPU 2 to 7
The thread 2 PID 9311 is moved from CPU 7 to 2
Part1 global matrix multiplication using global scheduling correct.
Part1 global matrix multiplication compute result correct
Global Multi Thread Spend time : 20.1476

==========Start Partition Multi-Thread Matrix Multiplication==========
Thread ID : 2   PID : 9316      Core : 2
Thread ID : 1   PID : 9315      Core : 1
Thread ID : 0   PID : 9314      Core : 0
Thread ID : 3   PID : 9317      Core : 3
Part1 partition matrix multiplication using parition scheduling correct.
Part1 partition matrix multiplication compute result correct
Partition Multi Thread Spend time : 20.4333
```

Fig7. Result of Global and Partition.

● 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，採取將 thread 優先給低 index 的 core 執行，在程式上先直白的，先將第一顆 core 的使用量塞滿，如果塞不下就給下一顆，直到四顆 core 都滿了，如果還有沒被排進去的 thread，就會被顯示出來以及每顆 core 被安排的 thread 是誰。

```
System::partitionFirstFit()
{
    std::cout << "\n==========Partition First-Fit Multi Thread Matrix Multiplication==========" << std::endl;
#if (PART == 2)
    check->setCheckState(PARTITION_FF);
#endif

    for (int i = 0; i < CORE_NUM; i++)
        cpuSet[i].emptyCPU(); // Reset the CPU set


    /*~~~~~~~~~~~~Your code(PART2)~~~~~~~~~~~*/
    // Implement parititon first-fit and print result.

    for(int cpu_number = 0; cpu_number<CORE_NUM;++cpu_number){
        for(int k = 0; k<numThread;++k){
            // printf("--%f--\n",cpuSet[cpu_number].utilization());
            if(cpuSet[cpu_number].utilization()+threadSet[k].utilization() <=1 & threadSet[k].setCore_()==-1){
                cpuSet[cpu_number].pushThreadToCPU(&threadSet[k]);
                threadSet[k].setThreadCore(cpu_number);
                // printf("utilization:%f,thread:ID:%d,cpuID:%d\n",cpuSet[cpu_number].utilization(),k,cpu_number);
            }
        }
    }
    for(int x = 0; x<numThread;++x){
        if(threadSet[x].setCore_()==-1){
            printf("Thread-%d not schedulable.\n",x);
        }
    }
    pthread_mutex_lock(&count_Mutex);
    for(int y = 0;y<CORE_NUM;++y){

        cpuSet[y].printCPUInformation();
    }
    pthread_mutex_unlock(&count_Mutex);

    /*~~~~~~~~~~~~~~~~~~END~~~~~~~~~~~~~~~~~~*/

    partitionMultiCoreMatrixMulti(); // Create the multi-thread matrix multiplication
}
```

Fig7. First-Fit code

在 Best-Fit 中，優先把 thread 給使用量較高的 core，會每次去比對 core 的使用量來判斷。

```
void
System::partitionBestFit()
{
    std::cout << "\n==========Partition Best-Fit Multi Thread Matrix Multiplication==========" << std::endl;
    float totoal;
    float bestfit_valuse = 0;
    int thread_ID,cpu_ID;
#if (PART == 2)
    check->setCheckState(PARTITION_BF);
#endif

    for (int i = 0; i < CORE_NUM; i++){
        cpuSet[i].emptyCPU(); // Reset the CPU set
    }
    /*~~~~~~~~~~~~Your code(PART2)~~~~~~~~~~*/
    // Implement partition best-fit and print result.
    for(int j = 0 ; j<numThread;++j){
        for(int k = 0 ; k<CORE_NUM;++k){
            totoal = cpuSet[k].utilization() + threadSet[j].utilization();
            if(totoal <= 1 &  totoal> bestfit_valuse){
                bestfit_valuse = totoal;
                thread_ID = j;
                cpu_ID = k;
            }
        }
        if(thread_ID !=-1 & cpu_ID !=-1){
            cpuSet[cpu_ID].pushThreadToCPU(&threadSet[thread_ID]);
            threadSet[thread_ID].setThreadCore(cpu_ID);
            thread_ID =-1;
            cpu_ID = -1;
            bestfit_valuse = 0;
        }

    }
    for(int x = 0; x<numThread;++x){
        if(threadSet[x].setCore_()==-1){
            printf("Thread-%d not schedulable.\n",x);
        }
    }
    pthread_mutex_lock(&count_Mutex);
    for(int y = 0;y<CORE_NUM;++y){

        cpuSet[y].printCPUInformation();
    }
    pthread_mutex_unlock(&count_Mutex);

    /*~~~~~~~~~~~~~~~~~~END~~~~~~~~~~~~~~~~~~*/

    partitionMultiCoreMatrixMulti(); // Create the multi-thread matrix multiplication
}
```

Fig8. Best-Fit code

在 Worsr-Fit 中，優先把 thread 給使用量較低的 core，會每次去比對 core 的使用量來判斷。

```
void
System::partitionWorstFit()
{

    std::cout << "\n==========Partition Worst-Fit Multi Thread Matrix Multiplication==========" << std::endl;
    float totoal;
    float bestfit_valuse = 1;
    int thread_ID,cpu_ID;
#if (PART == 2)
    check->setCheckState(PARTITION_WF);
#endif

    for (int i = 0; i < CORE_NUM; i++)
        cpuSet[i].emptyCPU();

    /*~~~~~~~~~~~Your code(PART2)~~~~~~~~~~*/
    // Implement partition worst-fit and print result.
    for(int j = 0 ; j<numThread;++j){
        for(int k = 0 ; k<CORE_NUM;++k){
            totoal = cpuSet[k].utilization() + threadSet[j].utilization();
            if(totoal <= 1 &  totoal< bestfit_valuse){
                bestfit_valuse = totoal;
                thread_ID = j;
                cpu_ID = k;
            }
        }
        if(thread_ID !=-1 & cpu_ID !=-1){
            cpuSet[cpu_ID].pushThreadToCPU(&threadSet[thread_ID]);
            threadSet[thread_ID].setThreadCore(cpu_ID);
            thread_ID =-1;
            cpu_ID = -1;
            bestfit_valuse = 1;
        }

    }
    for(int x = 0; x<numThread;++x){
        if(threadSet[x].setCore_()==-1){
            printf("Thread-%d not schedulable.\n",x);
        }
    }
    pthread_mutex_lock(&count_Mutex);
    for(int y = 0;y<CORE_NUM;++y){

        cpuSet[y].printCPUInformation();
    }
    pthread_mutex_unlock(&count_Mutex);

    /*~~~~~~~~~~~~~~~~~END~~~~~~~~~~~~~~~~~*/

    partitionMultiCoreMatrixMulti(); // Create the multi-thread matrix multiplication
}
```

Fig9. Worst-Fit code

[Result. 30%]

- Show the scheduling states of tasks. (You have to show the screenshot result of using input part2_Input_10.txt and part2_Input_20.txt)
  (1) Result of using input part2_Input_10.txt.

```
==========Partition First-Fit Multi Thread Matrix Multiplication==========
Thread-9 not schedulable.
Core Number : 0
[ 0, 1, 4, ]
Total Utilization : 0.9925

Core Number : 1
[ 2, 3, ]
Total Utilization : 0.729

Core Number : 2
[ 5, 6, ]
Total Utilization : 0.6505

Core Number : 3
[ 7, 8, ]
Total Utilization : 0.764

Thread ID : 0   PID : 4837     Core : 0       Utilization : 0.38     MatrixSize : 760
Thread ID : 1   PID : 4838     Core : 0       Utilization : 0.346    MatrixSize : 692
Thread ID : 3   PID : 4840     Core : 1       Utilization : 0.34     MatrixSize : 680
Thread ID : 2   PID : 4839     Core : 1       Utilization : 0.389    MatrixSize : 778
Thread ID : 4   PID : 4841     Core : 0       Utilization : 0.2665   MatrixSize : 533
Thread ID : 5   PID : 4842     Core : 2       Utilization : 0.344    MatrixSize : 688
Thread ID : 6   PID : 4843     Core : 2       Utilization : 0.3065   MatrixSize : 613
Thread ID : 7   PID : 4844     Core : 3       Utilization : 0.367    MatrixSize : 734
Thread ID : 8   PID : 4845     Core : 3       Utilization : 0.397    MatrixSize : 794
Thread ID : 9   PID : 4846     Core : 0       Utilization : 0.373    MatrixSize : 746
Part2 partiton result correct
Part2 compute result correct
Partition Multi Thread Spend time : 6.65529
```

Fig10. Result of using input part2_Input_10.txt by FF.

```
==========Partition Best-Fit Multi Thread Matrix Multiplication==========
Thread-9 not schedulable.
Core Number : 0
[ 0, 1, ]
Total Utilization : 0.726

Core Number : 1
[ 2, 3, 4, ]
Total Utilization : 0.9955

Core Number : 2
[ 5, 6, ]
Total Utilization : 0.6505

Core Number : 3
[ 7, 8, ]
Total Utilization : 0.764

Thread ID : 0   PID : 4847     Core : 0       Utilization : 0.38     MatrixSize : 760
Thread ID : 6   PID : 4853     Core : 2       Utilization : 0.3065   MatrixSize : 613
Thread ID : 9   PID : 4856     Core : 7       Utilization : 0.373    MatrixSize : 746
Thread ID : 3   PID : 4850     Core : 1       Utilization : 0.34     MatrixSize : 680
Thread ID : 1   PID : 4848     Core : 0       Utilization : 0.346    MatrixSize : 692
Thread ID : 5   PID : 4852     Core : 2       Utilization : 0.344    MatrixSize : 688
Thread ID : 8   PID : 4855     Core : 3       Utilization : 0.397    MatrixSize : 794
Thread ID : 4   PID : 4851     Core : 1       Utilization : 0.2665   MatrixSize : 533
Thread ID : 7   PID : 4854     Core : 3       Utilization : 0.367    MatrixSize : 734
Thread ID : 2   PID : 4849     Core : 1       Utilization : 0.389    MatrixSize : 778
Part2 partiton result correct
Part2 compute result correct
Partition Multi Thread Spend time : 6.96175
```

Fig11. Result of using input part2_Input_10.txt by BF.

```
==========Partition Worst-Fit Multi Thread Matrix Multiplication==========
Core Number : 0
[ 0, 6, ]
Total Utilization : 0.6865

Core Number : 1
[ 1, 5, ]
Total Utilization : 0.69

Core Number : 2
[ 2, 7, ]
Total Utilization : 0.756

Core Number : 3
[ 3, 4, 9, ]
Total Utilization : 0.9795

Thread ID : 0   PID : 4857     Core : 0       Utilization : 0.38      MatrixSize : 760
Thread ID : 4   PID : 4861     Core : 3       Utilization : 0.2665    MatrixSize : 533
Thread ID : 3   PID : 4860     Core : 3       Utilization : 0.34      MatrixSize : 680
Thread ID : 1   PID : 4858     Core : 1       Utilization : 0.346     MatrixSize : 692
Thread ID : 2   PID : 4859     Core : 2       Utilization : 0.389     MatrixSize : 778
Thread ID : 5   PID : 4862     Core : 1       Utilization : 0.344     MatrixSize : 688
Thread ID : 6   PID : 4863     Core : 0       Utilization : 0.3065    MatrixSize : 613
Thread ID : 7   PID : 4864     Core : 2       Utilization : 0.367     MatrixSize : 734
Thread ID : 9   PID : 4866     Core : 3       Utilization : 0.373     MatrixSize : 746
Thread ID : 8   PID : 4865     Core : 3       Utilization : 0.397     MatrixSize : 794
Part2 partiton result correct
Part2 compute result correct
Partition Multi Thread Spend time : 10.4968
```

Fig12. Result of using input part2_Input_10.txt by WF.

(2) Result of using input part2_Input_20.txt.

```
==========Partition First-Fit Multi Thread Matrix Multiplication==========
Thread-19 not schedulable.
Core Number : 0
[ 0, 1, 2, 3, 4, 5, 7, 9, ]
Total Utilization : 0.937

Core Number : 1
[ 6, 8, 10, ]
Total Utilization : 0.9125

Core Number : 2
[ 11, 12, 13, 14, 16, ]
Total Utilization : 0.857

Core Number : 3
[ 15, 17, 18, ]
Total Utilization : 0.9695

Thread ID : 0   PID : 8200     Core : 0       Utilization : 0.02      MatrixSize : 40
Thread ID : 13  PID : 8213     Core : 2       Utilization : 0.128     MatrixSize : 256
Thread ID : 4   PID : 8204     Core : 0       Utilization : 0.12      MatrixSize : 240
Thread ID : 5   PID : 8205     Core : 0       Utilization : 0.296     MatrixSize : 592
Thread ID : 2   PID : 8202     Core : 0       Utilization : 0.08      MatrixSize : 160
Thread ID : 3   PID : 8203     Core : 0       Utilization : 0.08      MatrixSize : 160
Thread ID : 6   PID : 8206     Core : 1       Utilization : 0.3465    MatrixSize : 693
Thread ID : 7   PID : 8207     Core : 0       Utilization : 0.05      MatrixSize : 100
Thread ID : 8   PID : 8208     Core : 1       Utilization : 0.233     MatrixSize : 466
Thread ID : 9   PID : 8209     Core : 0       Utilization : 0.131     MatrixSize : 262
Thread ID : 10  PID : 8210     Core : 1       Utilization : 0.333     MatrixSize : 666
Thread ID : 11  PID : 8211     Core : 2       Utilization : 0.272     MatrixSize : 544
Thread ID : 1   PID : 8201     Core : 0       Utilization : 0.16      MatrixSize : 320
Thread ID : 12  PID : 8212     Core : 2       Utilization : 0.241     MatrixSize : 482
Thread ID : 15  PID : 8215     Core : 3       Utilization : 0.29      MatrixSize : 580
Thread ID : 16  PID : 8216     Core : 2       Utilization : 0.1       MatrixSize : 200
Thread ID : 18  PID : 8218     Core : 3       Utilization : 0.333     MatrixSize : 666
Thread ID : 14  PID : 8214     Core : 2       Utilization : 0.116     MatrixSize : 232
Thread ID : 17  PID : 8217     Core : 3       Utilization : 0.3465    MatrixSize : 693
Thread ID : 19  PID : 8219     Core : 1       Utilization : 0.1825    MatrixSize : 365
Part2 partiton result correct
Part2 compute result correct
Partition Multi Thread Spend time : 4.36675
```

Fig13. Result of using input part2_Input_20.txt by FF.

```
==========Partition Best-Fit Multi Thread Matrix Multiplication==========
Thread-19 not schedulable.
Core Number : 0
[ 0, 1, 2, 3, 4, 5, 7, 9, ]
Total Utilization : 0.937

Core Number : 1
[ 6, 8, 10, ]
Total Utilization : 0.9125

Core Number : 2
[ 11, 12, 13, 14, 16, ]
Total Utilization : 0.857

Core Number : 3
[ 15, 17, 18, ]
Total Utilization : 0.9695

Thread ID : 0   PID : 8221      Core : 0        Utilization : 0.02      MatrixSize : 40
Thread ID : 3   PID : 8224      Core : 0        Utilization : 0.08      MatrixSize : 160
Thread ID : 8   PID : 8229      Core : 1        Utilization : 0.233     MatrixSize : 466
Thread ID : 13  PID : 8234      Core : 2        Utilization : 0.128     MatrixSize : 256
Thread ID : 11  PID : 8232      Core : 2        Utilization : 0.272     MatrixSize : 544
Thread ID : 14  PID : 8235      Core : 2        Utilization : 0.116     MatrixSize : 232
Thread ID : 16  PID : 8237      Core : 2        Utilization : 0.1       MatrixSize : 200
Thread ID : 1   PID : 8222      Core : 0        Utilization : 0.16      MatrixSize : 320
Thread ID : 7   PID : 8228      Core : 0        Utilization : 0.05      MatrixSize : 100
Thread ID : 5   PID : 8226      Core : 0        Utilization : 0.296     MatrixSize : 592
Thread ID : 4   PID : 8225      Core : 0        Utilization : 0.12      MatrixSize : 240
Thread ID : 9   PID : 8230      Core : 0        Utilization : 0.131     MatrixSize : 262
Thread ID : 10  PID : 8231      Core : 1        Utilization : 0.333     MatrixSize : 666
Thread ID : 12  PID : 8233      Core : 2        Utilization : 0.241     MatrixSize : 482
Thread ID : 6   PID : 8227      Core : 1        Utilization : 0.3465    MatrixSize : 693
Thread ID : 17  PID : 8238      Core : 3        Utilization : 0.3465    MatrixSize : 693
Thread ID : 18  PID : 8239      Core : 3        Utilization : 0.333     MatrixSize : 666
Thread ID : 15  PID : 8236      Core : 3        Utilization : 0.29      MatrixSize : 580
Thread ID : 2   PID : 8223      Core : 0        Utilization : 0.08      MatrixSize : 160
Thread ID : 19  PID : 8240      Core : 2        Utilization : 0.1825    MatrixSize : 365
Part2 partiton result correct
Part2 compute result correct
Partition Multi Thread Spend time : 4.42628
```

Fig14. Result of using input part2_Input_20.txt by BF.

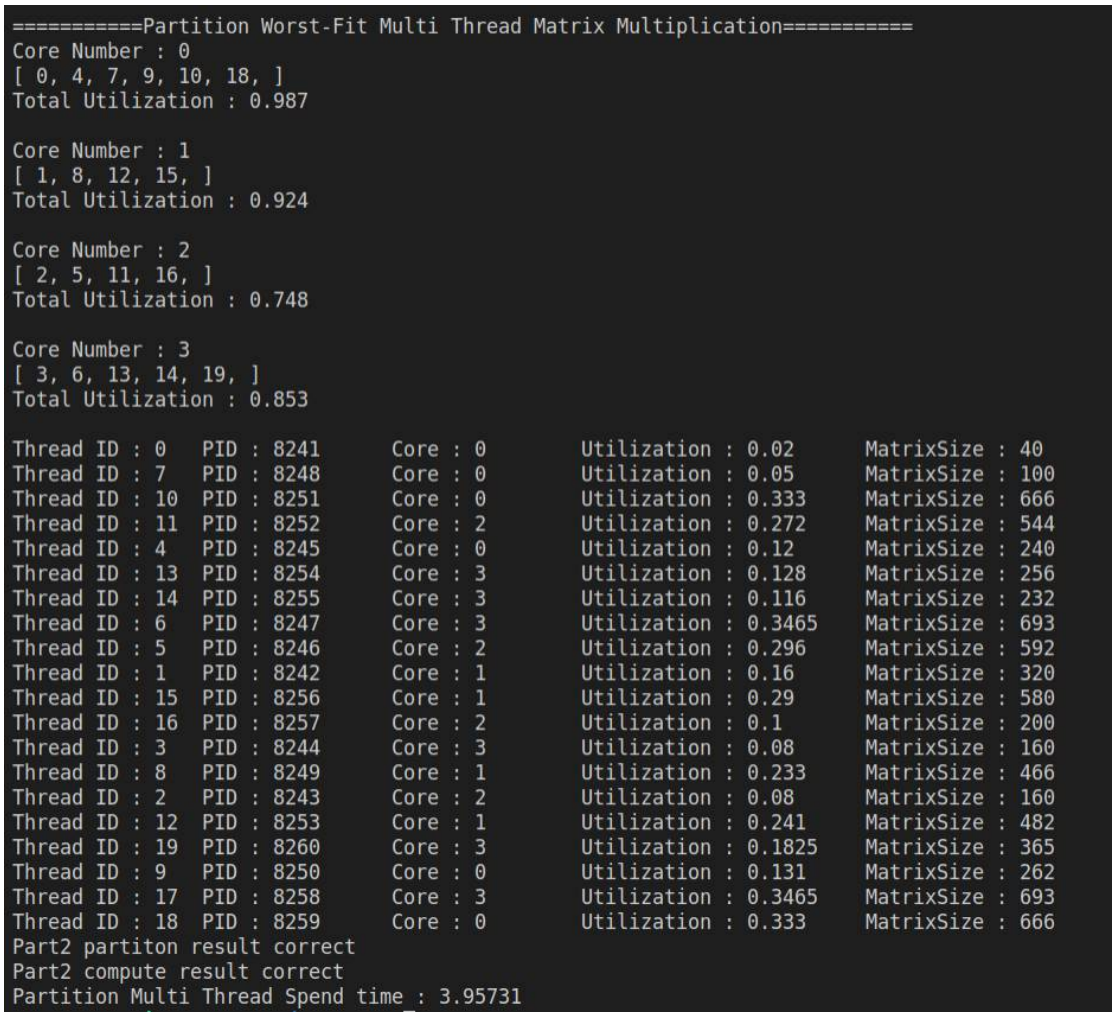

Fig15. Result of using input part2_Input_20.txt by WF.

● Part 3

[Scheduler Implementation. 10%]

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

一開始除了初始化 thread，在 part3，一開始會定義出要使用哪個排程(FIFO or RR)，但還沒有成功設定排程規則。

```
System::System(char* input_file)
{
    loadInput(input_file); // Set up threadSet, singleResult, multiResult, and matrix

    for (int i = 0; i<numThread; i++) {
#if (PART == 1)
        // Set the singleResult, multResult, and matrix to thread.
        threadSet[i].initialThread(singleResult[0], multiResult[0], matrix[0]);
        /*~~~~~~~~~~~~Your code(PART1)~~~~~~~~~~~~*/
        // Set up the calculate range of matrix.
        threadSet[i].setStartCalculatePoint(i*threadSet[i].matrixSize()/numThread);
        threadSet[i].setEndCalculatePoint((i+1)*threadSet[i].matrixSize()/numThread);
        /*~~~~~~~~~~~~~~~~~~END~~~~~~~~~~~~~~~~~~*/
#else
        // Set the singleResult, multResult, and matrix to thread.
        threadSet[i].initialThread(singleResult[i], multiResult[i], matrix[i]);
#endif

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

Fig16. implement the scheduler setting code-1.

透過 matrixMultiplication 呼叫每個 thread 的 setUpScheduler()。

```
Thread::matrixMultiplication(void* args)
{
    Thread *obj = (Thread*)args;
#if (PART == 3)
    obj->setUpScheduler();
#endif
    /*~~~~~~~~~~~~Your code(PART1)~~~~~~~~~~~~*/
    // Set up the affinity mask
    // pthread_mutex_lock(&count_Mutex);
    if(obj->setCore == -1 ){
        obj->core = sched_getcpu();
        obj->PID = syscall(SYS_gettid);
        if(PART != 3){
            obj->printInformation();
        }

    }
    else{
        obj->setUpCPUAffinityMask(obj->setCore);
        obj->core = sched_getcpu();
        obj->PID = syscall(SYS_gettid);
        if(PART != 3){
            obj->printInformation();
        }

    }
    // pthread_mutex_unlock(&count_Mutex);
    /*~~~~~~~~~~~~~~~~~~END~~~~~~~~~~~~~~~~~~*/
```

Fig17. implement the scheduler setting code-2.

透過此 function 可以直接對每一個 thread 進行排程規則調。

```
void
Thread::setUpScheduler()
{
    /*~~~~~~~~~~~~~~Your code(PART3)~~~~~~~~~~~~*/
    // Set up the scheduler for current thread
    struct sched_param sp;
    // printf("ID:%d,_schedulingPolicy:%d\n",PID,_schedulingPolicy);
    sp.sched_priority = sched_get_priority_max(_schedulingPolicy);
    sched_setscheduler(0,_schedulingPolicy,&sp);
    /*~~~~~~~~~~~~~~~~~~~~END~~~~~~~~~~~~~~~~~~~~*/
}
```

Fig18. implement the scheduler setting code-3.

```
    }
#if (PART == 3)
    /*~~~~~~~~~~~~Your code(PART3)~~~~~~~~~~*/
    // Obaserve the execute thread on core-0
    // pthread_mutex_lock(&count_Mutex);

    if(obj->core ==0 && current_PID == -1){
        printf("Core0 start PID-%d\n",obj->PID);
        current_PID =syscall(SYS_gettid);
    }
    // else if (current_PID != obj->PID && obj->core ==0 ){
    else if (current_PID != syscall(SYS_gettid) && obj->core ==0 ){
        printf("Core:%d context switch from PID-%d to PID-%d\n",obj->core,current_PID,obj->PID);
        current_PID = syscall(SYS_gettid);
    }
    // pthread_mutex_unlock(&count_Mutex);
    /*~~~~~~~~~~~~~~~~~~END~~~~~~~~~~~~~~~~~~*/
```

Fig19. implement the scheduler setting code-4.

[Result. 10%]

- Show the process execution states of tasks. (You have to show the screenshot result of using input part3_Input.txt)

(1) Result of FIFO

```
==========Partition First-Fit Multi Thread Matrix Multiplication==========
Thread-9 not schedulable.
Core Number : 0
[ 0, 1, 4, ]
Total Utilization : 0.9925

Core Number : 1
[ 2, 3, ]
Total Utilization : 0.729

Core Number : 2
[ 5, 6, ]
Total Utilization : 0.6505

Core Number : 3
[ 7, 8, ]
Total Utilization : 0.764

Core0 start PID-9027
Core:0 context switch from PID-9027 to PID-9028
Core:0 context switch from PID-9028 to PID-9031
Part3 change scheduler correct
Part3 compute result correct
Partition Multi Thread Spend time : 4.88486
```

Fig20. Result of FIFO

(1) Result of RR

```
===========Partition First-Fit Multi Thread Matrix Multiplication===========
Thread-9 not schedulable.
Core Number : 0
[ 0, 1, 4, ]
Total Utilization : 0.9925

Core Number : 1
[ 2, 3, ]
Total Utilization : 0.729

Core Number : 2
[ 5, 6, ]
Total Utilization : 0.6505

Core Number : 3
[ 7, 8, ]
Total Utilization : 0.764

Core0 start PID-9156
Core:0 context switch from PID-9156 to PID-9157
Core:0 context switch from PID-9157 to PID-9160
Core:0 context switch from PID-9160 to PID-9156
Core:0 context switch from PID-9156 to PID-9157
Core:0 context switch from PID-9157 to PID-9160
Core:0 context switch from PID-9160 to PID-9156
Core:0 context switch from PID-9156 to PID-9157
Core:0 context switch from PID-9157 to PID-9160
Core:0 context switch from PID-9160 to PID-9156
Core:0 context switch from PID-9156 to PID-9157
Core:0 context switch from PID-9157 to PID-9160
Core:0 context switch from PID-9160 to PID-9156
Core:0 context switch from PID-9156 to PID-9157
Core:0 context switch from PID-9157 to PID-9160
Core:0 context switch from PID-9160 to PID-9156
Core:0 context switch from PID-9156 to PID-9157
Core:0 context switch from PID-9157 to PID-9160
Core:0 context switch from PID-9160 to PID-9156
Core:0 context switch from PID-9156 to PID-9157
Core:0 context switch from PID-9157 to PID-9160
Core:0 context switch from PID-9160 to PID-9156
```

Fig21. Result of RR-1

```
Core:0 context switch from PID-9156 to PID-9157
Core:0 context switch from PID-9157 to PID-9160
Core:0 context switch from PID-9160 to PID-9156
Core:0 context switch from PID-9156 to PID-9157
Core:0 context switch from PID-9157 to PID-9160
Core:0 context switch from PID-9160 to PID-9156
Core:0 context switch from PID-9156 to PID-9157
Core:0 context switch from PID-9157 to PID-9160
Core:0 context switch from PID-9160 to PID-9156
Core:0 context switch from PID-9156 to PID-9157
Core:0 context switch from PID-9157 to PID-9156
Core:0 context switch from PID-9156 to PID-9157
Core:0 context switch from PID-9157 to PID-9156
Core:0 context switch from PID-9156 to PID-9157
Core:0 context switch from PID-9157 to PID-9156
Core:0 context switch from PID-9156 to PID-9157
Core:0 context switch from PID-9157 to PID-9156
Core:0 context switch from PID-9156 to PID-9157
Core:0 context switch from PID-9157 to PID-9156
Core:0 context switch from PID-9156 to PID-9157
Core:0 context switch from PID-9157 to PID-9156
Core:0 context switch from PID-9156 to PID-9157
Core:0 context switch from PID-9157 to PID-9156
Core:0 context switch from PID-9156 to PID-9157
Core:0 context switch from PID-9157 to PID-9156
Core:0 context switch from PID-9156 to PID-9157
Core:0 context switch from PID-9157 to PID-9156
Core:0 context switch from PID-9156 to PID-9157
Core:0 context switch from PID-9157 to PID-9156
Part3 change scheduler correct
Part3 compute result correct
Partition Multi Thread Spend time : 5.02761
```

Fig22. Result of RR-2

Discussion

- Analyze and compare the response time of the program, with single thread and multi-thread using in part1 and part2. (Including Single, Global, First-Fit, Best-Fit, Worst-Fit) 10%

  Part 1: Result of using input part1_Input.txt.

Single	Global
93.4943	**25.1637**
93.5339	**23.7803**
93.6785	**21.425**
93.6103	**24.6045**
93.3217	**24.9995**

  Part 2: Result of using input part2_Input_10.txt.

Single	First-Fit	Best-Fit	Worst-Fit
18.6205	**6.91561**	**6.58652**	**9.40535**
18.4112	**6.0931**	**6.73887**	**9.28765**
18.1321	**6.14959**	**6.54708**	**9.35252**
18.7423	**6.108**	**6.48719**	**9.30566**
18.6457	**6.41064**	**7.1208**	**9.28416**

  Part 2: Result of using input part2_Input_20.txt.

Single	First-Fit	Best-Fit	Worst-Fit
10.5764	**4.44263**	**4.37661**	**3.9445**
10.559	**4.39945**	**4.22268**	**3.9663**
10.5462	**4.47182**	**4.41158**	**3.93752**
10.5662	**4.41889**	**4.27109**	**3.90958**
10.5418	**4.24276**	**4.20603**	**3.89103**

  從上述個數據表得知，在 part 1 時，單一 core 跑出來的時間很明顯與 global 差異蠻大的，代表在切範圍分散給各 core 去平行運算，是有達到加速的功用。而在 part 2，很明顯可以看到 First-Fit 與 Best-Fit 其實差距不大，甚至在每個 core 的排程上快幾乎一模一樣，所以無法上述數據中比較出他們之間的誰比較快，可能需要更多的測試數據去觀察，在 Worst-Fit，輸入

10 筆與輸入 20 筆差距蠻大的,甚至在速度上比其他兩個演算法來的優秀,但為啥差距之大,有可能是在輸入 10 筆的排程中,再等一個 loading 比較大的拖累時間,而在輸入 20 筆中,剛好分配的很平均,才導致比其他兩個演算法還要好,但還是可能要測試不同種數據才能在更深入的比對。

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

FIFO with FF	RR with FF
4.8846	5.0271
4.9	5.10912
4.82911	5.10921

觀察 part3 的結果後，我們得知 FIFO 與 RR 的排程方法不一樣，在 FIFO 中採取先進先出，當 core 0 第一個 thread 進去執行，他會把 core 0 鎖住不讓其他想要用 core 0 的 thread 得進去執行，等到第一個進去 thread 的做完之後，才會讓下一個 thread 使用，以此類推。而在 RR 中採取循環的方式,每一輪每個 thread 都會進去做一點事,然後馬上切換到下一個 thread 執行，且每次進去的排程都是固定的，如 1、2、4 接下來也是 1、2、4，除非有其中幾項 thread 做完，不然會一直輪流下去。Response time 目前實行過幾次發現，FIFO 速度有優於 RR，可能是在 context swtich 的延遲，才會有一點點差距。