

**EECE 7205 – Project 2:**  
**Energy and Performance - Aware Task Scheduling in**  
**a Mobile Cloud Computing**  
**Environment**

Tejas Krishna Reddy: 001423166

Iuliia Klykova: 001448298

December, 2018

## 1 Problem description

The purpose of this project was to develop a code that would resolve task scheduling problem with minimizing the total energy consumption of an application in a mobile device with access to the computing resources on the cloud under a hard application completion time constraint.

## 2 Results

All tasks will be executed on one of the three local cores or in the cloud. The execution time in a cloud consist o time f sending time  $T_i^s = 3$  , cloud computation time  $T_i^c = 1$  , and receiving time  $T_i^r = 1$  . Execution for every task on each of k cores  $T_{i,k}^l = 3$  , is presented in the table:

Task	Core 1	Core 2	Core 3
1	9	7	5
2	8	6	5
3	6	5	4
4	7	5	3
5	5	4	2
6	7	6	4
7	8	5	3
8	6	4	2
9	5	3	2
10	7	4	2

The power consumption of local cores 1-3 is  $P_1 = 1$  ,  $P_2 = 2$  , and  $P_3 = 4$  . The power consumption for sending a task to the cloud is  $P^s = 0.5$  .  
The final execution time shall not exceed 150% of an initial scheduling time.

## 2.1 Case\_1

### 2.1.1 Input

The tasks for the Case 1 are represented by a directed acyclic graph in Figure 1.

- The task 1 is an entry task.
- The tasks 2, 3, 4, 5, 6 are successors of the task 1 and can be executed only after the task 1 had been executed.
- The task 7 is a successor of the task 3.
- The task 8 is a successor of the tasks 2, 4, and 6.
- The task 9 is a successor of the tasks 2, 4, and 5.
- The task 10 is a successor of the tasks 7, 8, 9 and is an exit task.

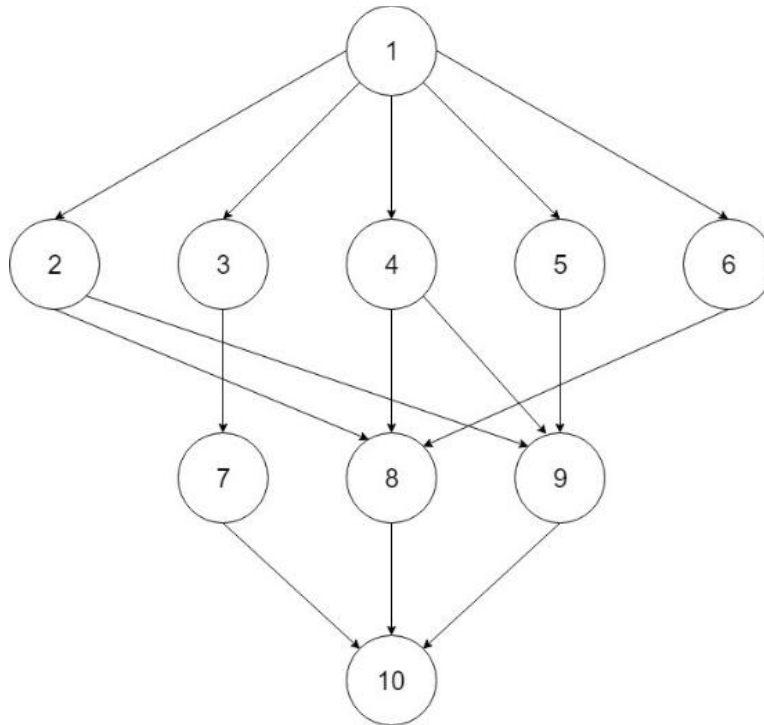


Figure 1. Task graph 1

### 2.1.2 Output

#### Step 1. Initial Scheduling

The initial scheduling gave us the following result:

Total energy consumption  $E_{total} = 102 \text{ units}$

Total execution time  $T_{total} = 18 \text{ units}$

Thus, the maximum time for the next step should not exceed

$$T_{max} = 1.5 \cdot T_{total} = 27 \text{ units}$$

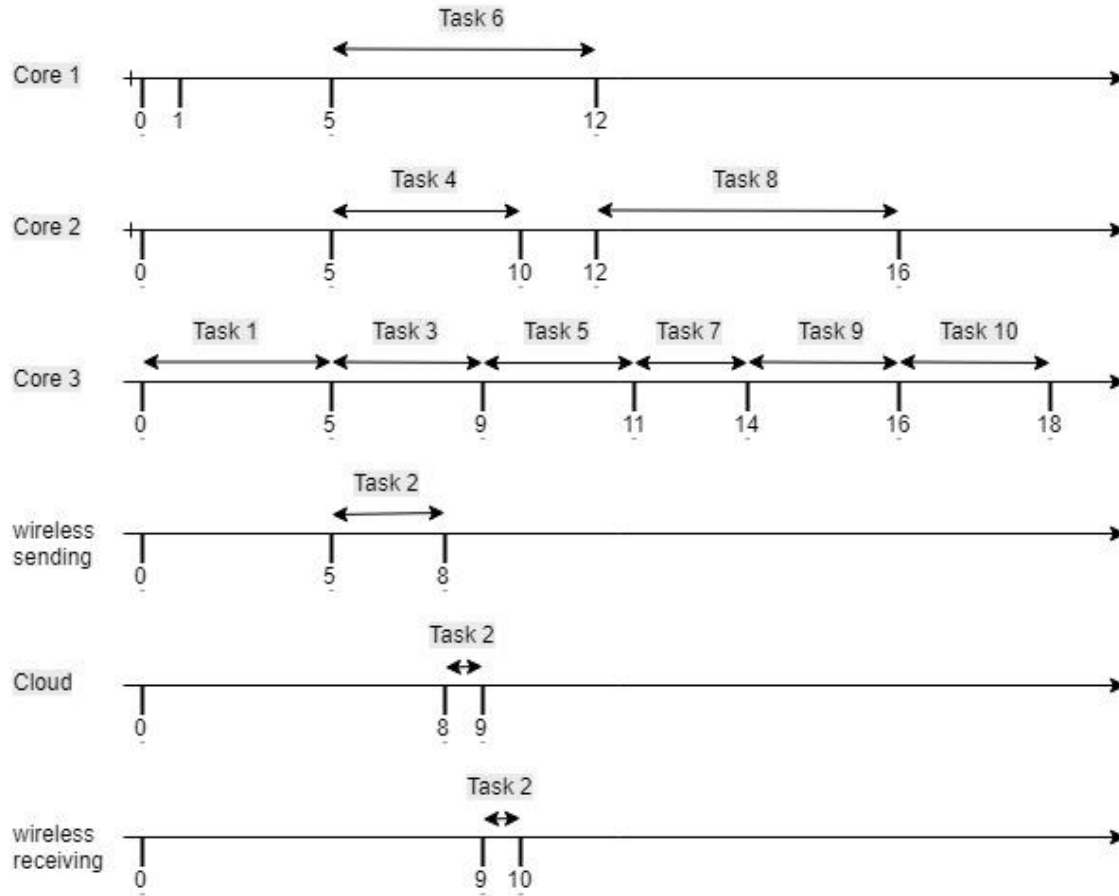


Figure 2. Initial scheduling diagram

## Step 2. Task Migration Algorithm

In this step we trying to minimize the energy consumption of the tasks execution while not exceeding the maximum time we obtained in the previous step. The Task Migration Algorithm gave us the following result:

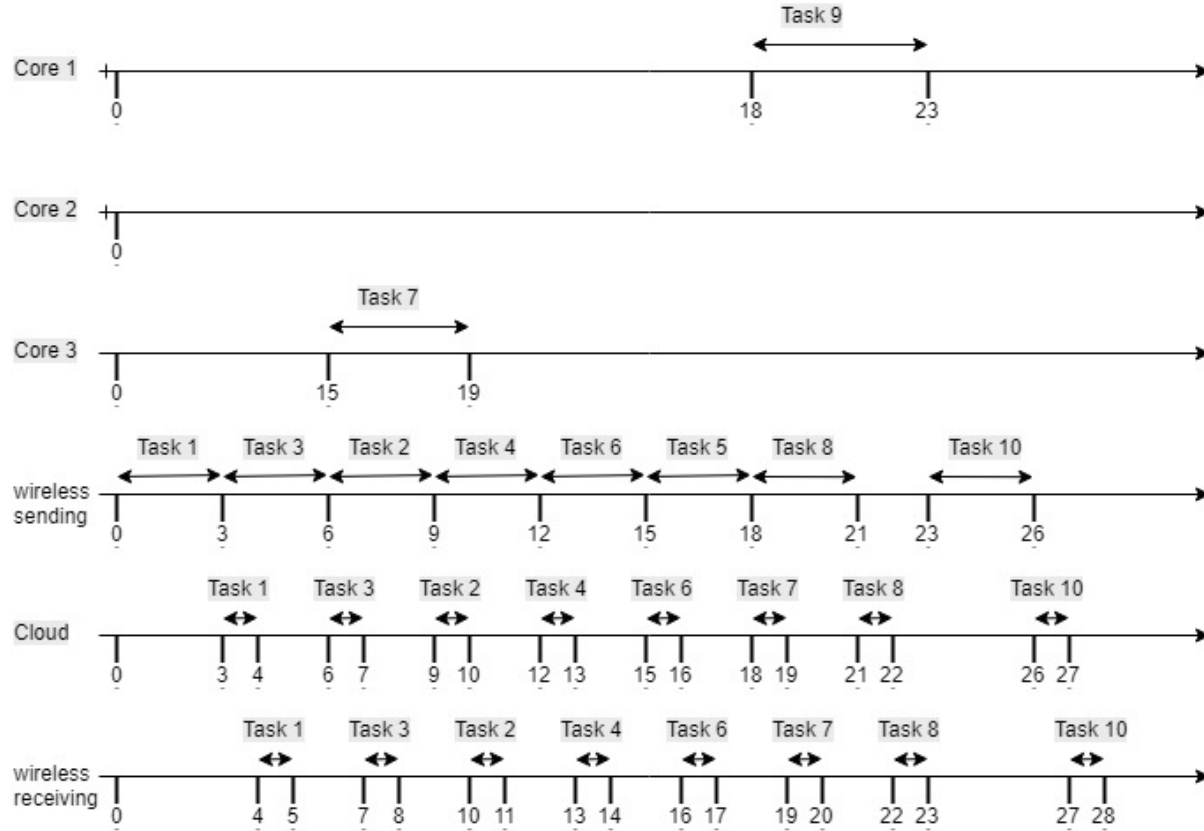


Figure 3. Task scheduling diagram by the MMC task scheduling algorithm

Total energy consumption  $E_{total} = 30 \text{ units}$

Total execution time  $T_{total} = 28 \text{ units}$

## 2.2 Case\_2

### 2.1.1 Input

The tasks for the Case 1 are represented by a directed acyclic graph in Figure 1.

- The task 1 is an entry task.
- The tasks 2, 3, 4, 5 are successors of the task 1 and can be executed only after the task 1 had been executed.
- The task 6 is a successor of the task 2.
- The task 7 is a successor of the tasks 2 and 3.
- The task 8 is a successor of the tasks 3 and 4.
- The task 9 is a successor of the tasks 4 and 5.
- The task 10 is a successor of the tasks 6, 7, 8, and 9 and is an exit task.

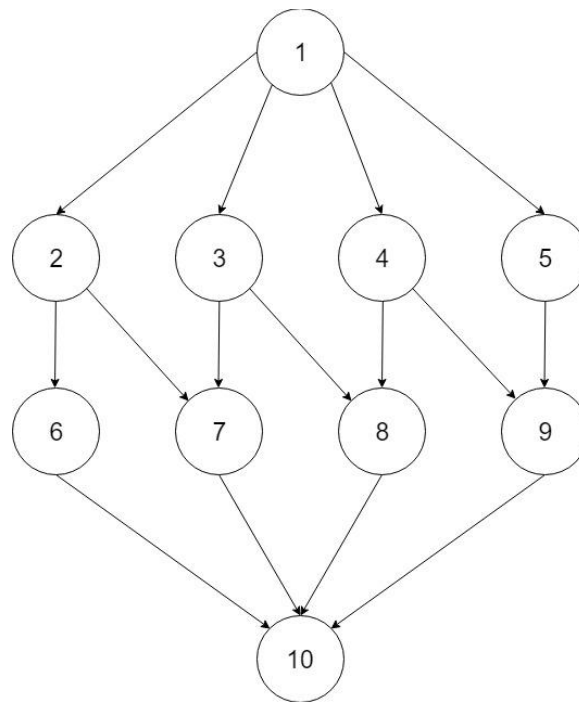


Figure 4. Task graph 2

## 2.2.2 Output

### Step 1. Initial Scheduling

The initial scheduling gave us the following result:

Total energy consumption  $E_{total} = 103 \text{ units}$

Total execution time  $T_{total} = 18 \text{ units}$

Thus, the maximum time for the next step should not exceed

$$T_{max} = 1.5 \cdot T_{total} = 27 \text{ units}$$

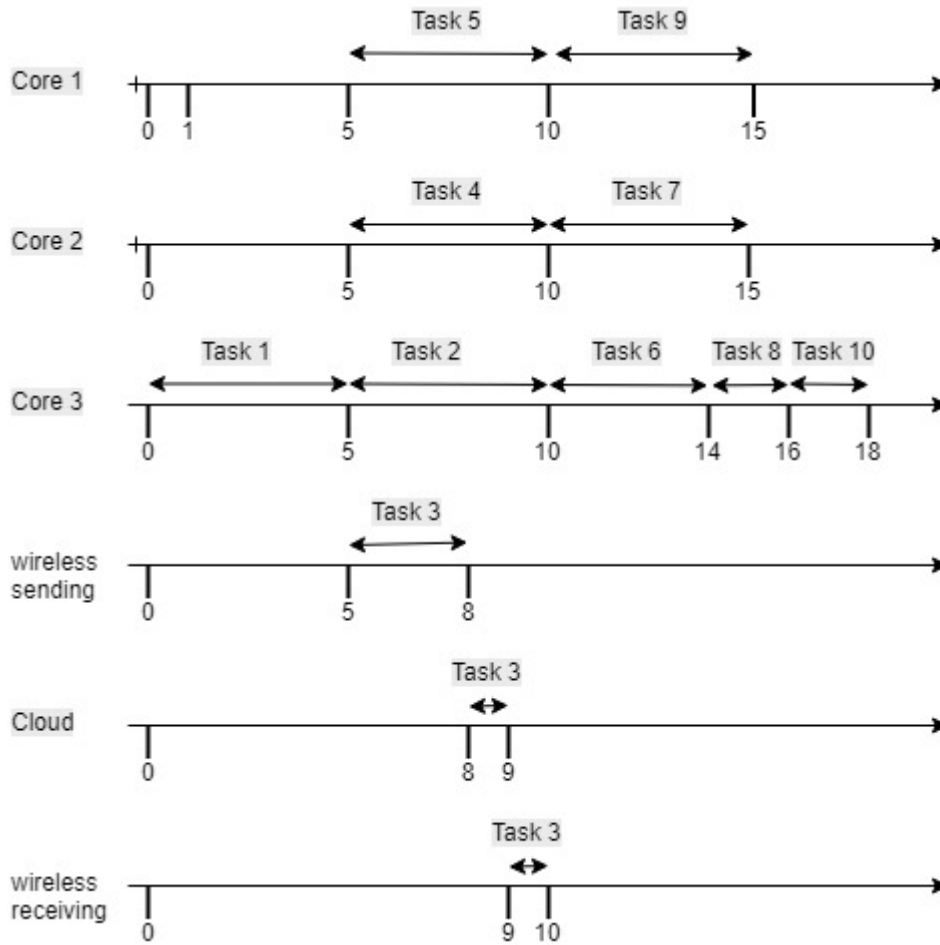


Figure 5. Initial scheduling diagram

## Step 2. Task Migration Algorithm

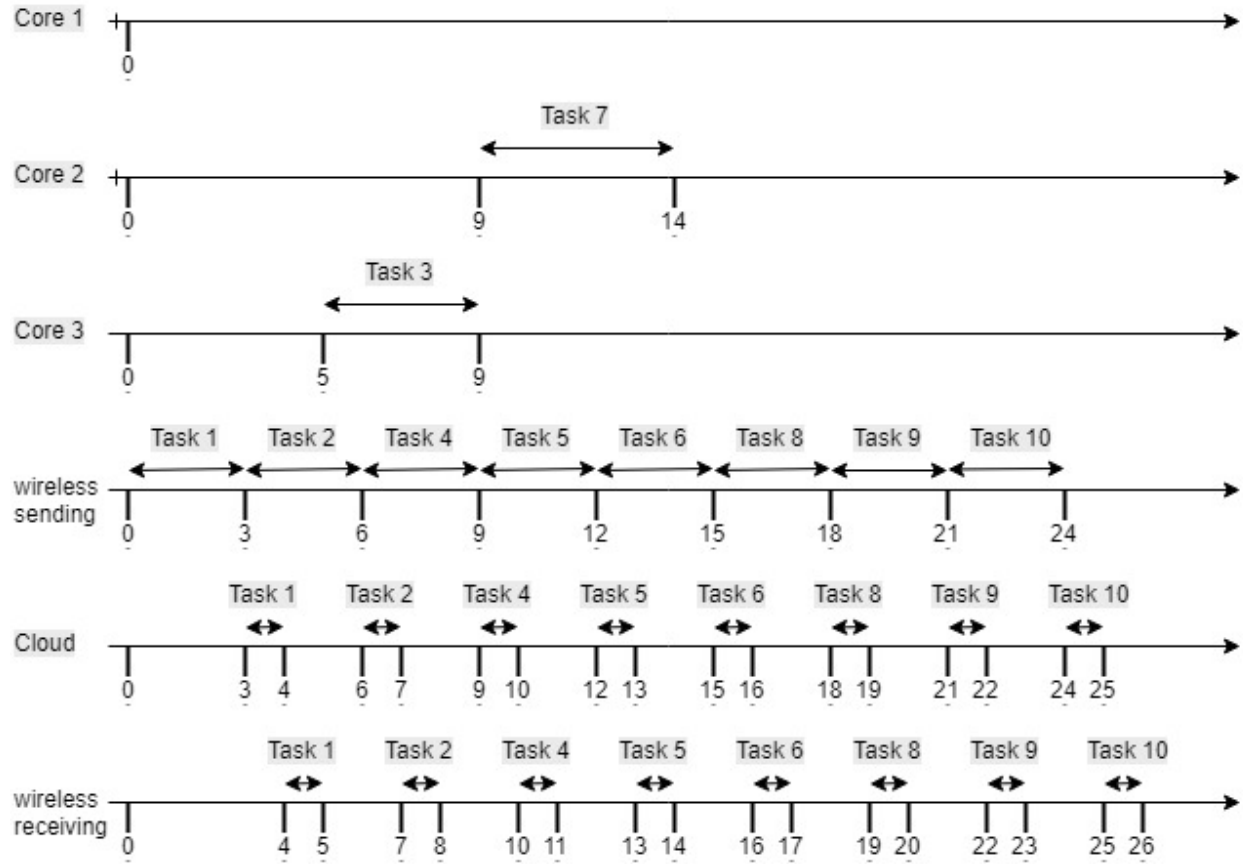


Figure 6. Task scheduling diagram by the MMC task scheduling algorithm

Total energy consumption  $E_{total} = 34 \text{ units}$

Total execution time  $T_{total} = 26 \text{ units}$



### 3. Code Analysis

The input graph in the code is represented by a 10 x 10 adjacency matrix. All execution units are represented as arrays.

#### a. Primary assignments

In *Task\_Scheduling* function for each task we define how much time it takes to execute it on each local core and a cloud and then select the one that gives the smallest execution time.

#### b. Task prioritizing

Find the initial priority level for each task by calculating the computation cost and the priority level of its successors. We calculated the exit task priority first. Save all results in the array *priority[V]*. The priority for the task 1 is *priority[0]* and so on.

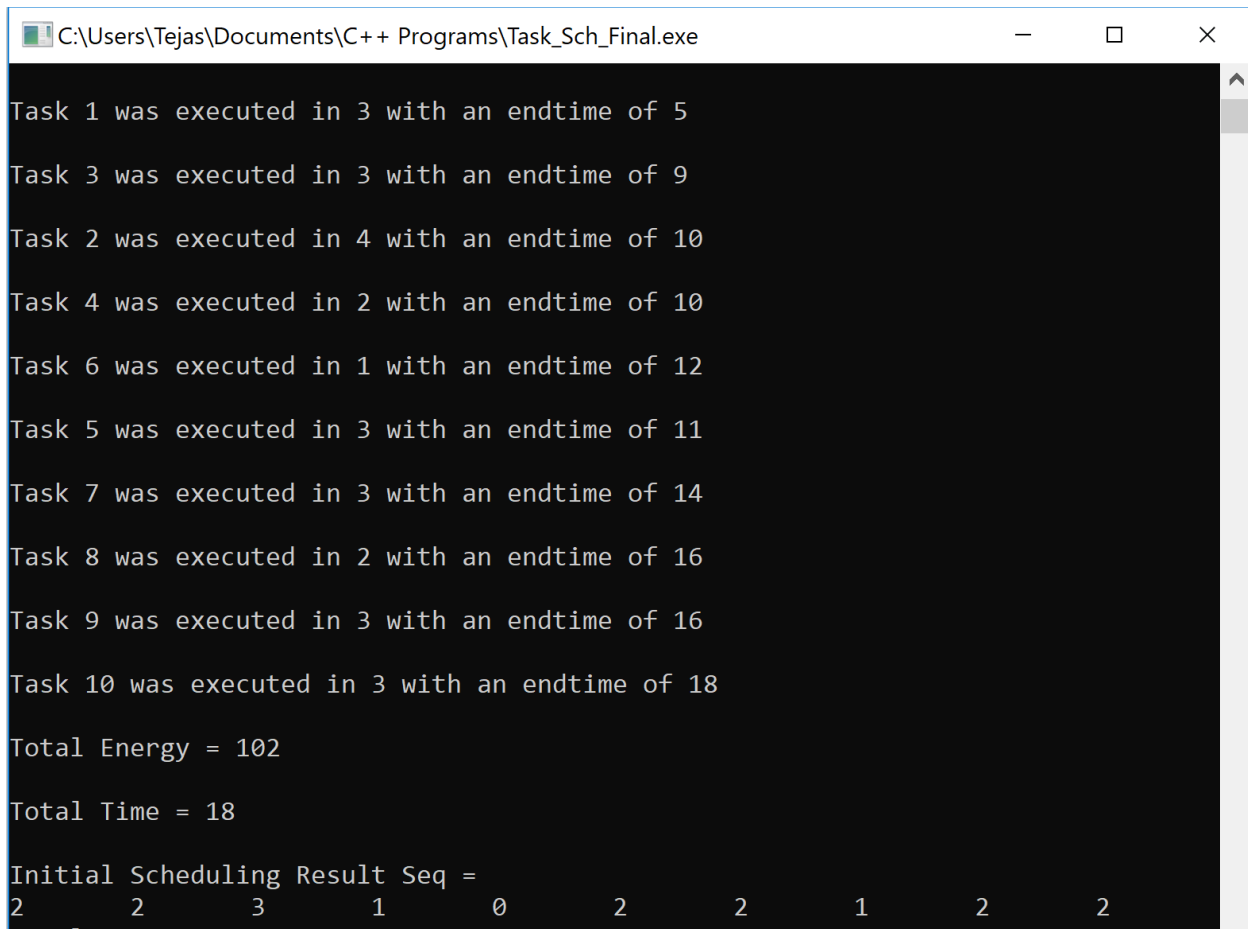
```
C:\Users\Tejas\Documents\C++ Programs\Task_Sch_Final.exe
Priority
21    14    14    13    10    13    9    8    7    4
Sorted Priority Array looks like this:
4 7 8 9 10 13 13 14 14 21
priority_index
0     2     1     3     5     4     6     7     8     9
cloud_task
0     0     0     0     0     0     0     0     0     0
T_re
5     5     5     5     5     5     5     5     5     5
T_L_min
5     5     4     3     2     4     3     2     2     2
w
7     6     5     5     3     5     5     4     3     4
```

The “*Priority*” variable in the Result window indicates the priority calculated for each of the corresponding tasks [from task 0 to task 9]. The calculated priority is then sorted in ascending order which is displayed as “*Sorted Priority Array*”. The task that corresponds to the sorted priority are stored in an array and finally reversed to display them in descending order as instructed in the paper. These tasks in the order of their priority are displayed under the variable “*Priority\_index*” in the above result window. Each of these tasks are then processed to find out if its a cloud\_task or a local task. If the task is determined as a cloud task then, we store ‘1’ in the corresponding location, else we store ‘0’ indicating that is a local task. This results can be observed under the variable “*cloud\_task*” in the above result window. “*T\_re*” determines the sum of sending\_time [Ts] + cloud\_computing\_time [Tc] + Wireless Receiving time [Tr]. “*T\_L\_min*”

represents the minimum local execution time (on the fastest core). “w” is a variable used to calculate the priority of corresponding tasks.

### c. The initial scheduling

Schedule the cores and the cloud for the task:



```
C:\Users\Tejas\Documents\C++ Programs\Task_Sch_Final.exe

Task 1 was executed in 3 with an endtime of 5
Task 3 was executed in 3 with an endtime of 9
Task 2 was executed in 4 with an endtime of 10
Task 4 was executed in 2 with an endtime of 10
Task 6 was executed in 1 with an endtime of 12
Task 5 was executed in 3 with an endtime of 11
Task 7 was executed in 3 with an endtime of 14
Task 8 was executed in 2 with an endtime of 16
Task 9 was executed in 3 with an endtime of 16
Task 10 was executed in 3 with an endtime of 18

Total Energy = 102

Total Time = 18

Initial Scheduling Result Seq =
2      2      3      1      0      2      2      1      2      2
```

Based on the priority of the tasks, the tasks were scheduled in a way that the tasks are executed if all its predecessors have been scheduled and no tasks overlap their time in the same core or cloud. The results of Initial\_Task\_Scheduling process have been displayed in the above output window. Each of the tasks from 1 to 10 in the input graph, were scheduled in either core 1 “1” or core 2, “2”, or core 3 “3” or on the cloud “4”. The task ending time was recorded in order to verify if tasks overlapped or have been scheduled before their predecessors.

For example, the first line,

“Task 1 was executed in 3 with an endtime of 5” indicates that Task 1 was executed in core 3 and ended in 5 time units.

Similarly, “Task 2 was executed in 4 with an endtime of 10” represents that Task 2 was executed in cloud [“4”] and ended its execution after 10 time units.

The final sequence obtained shows the cores utilized to execute the corresponding tasks in “priority\_index” array. where,

“0” represents core 1

“1” represents core 2

“2” represents core 3

“3” represents that task is executed in the cloud.

The maximum endtime represents the total time taken for executing all tasks in allocated cores. the Total\_time was found to be 18 time units.

$Energy = Power \times Time$ .

Power to execute on  $P_{core 1}$  was considered to be 1 units,  $P_{core 2} = 2$  units,  $P_{core 3} = 4$  and power required to send information to cloud was considered to be  $P_s = 0.5$  units. From these assumptions the total\_energy was found to be 102 units.

#### d. Task Migration Algorithm

- **Outer loop.** Repeats the *Kernal algorithm* until the power consumption is reduced to the minimum possible value while also maintaining the application time constraint

$T_{max} \geq T_{total}$  without violation.

- **Kernel Algorithm** (*optimize()* function). Here, we generate a new scheduling of each task graph by executing on the new location (local core/cloud) while the remaining tasks are executed on the same locations as the original scheduling. Try to

minimize the application completion time  $T_{total}$ . The energy consumption is fixed.

In order to optimize the Sequence to get proportional Energy and Time consumption, we calculate the ratio,

$$EnergyTimeRatio[i][j] = Improvement\ in\ Energy[i][j] / Improvement\ in\ Time\ [i][j];$$

From the resultant values, we obtain the minimum ratio [*minRatio*] from EnergyTimeRatio matrix and choose the corresponding sequence that resulted the ratio, to optimize next. This process is repeatedly executed until we get the desired results.

In the output figures below, we can visually see the the ratio matrix and minimum ratio obtained at each iteration. Also, it can be observed that the *minRatio* tends to reduce to 1 in a step by step fashion indicating the success of the optimization.

In the final iteration displayed, we get the most optimized results with *Total\_energy* = 30 units and *Total\_time* = 28 time units.

“*MinRatio*” is the least ratio in the “*EnergyTimeRatio*” matrix which indicates the most optimized Sequence. “*xMin*” and “*yMin*” represent the location of the most optimized Sequence, through which the corresponding sequence can be obtained.

Iterations 1, 2 and 3 are shown in the below figure.

```
Min Ratio = 4.55 was present at i,j = 0 0
4.35 5.11111 6.125 4.9375
3.66667 4.05 4.35 3.6
4.27273 4.9 5.3 4.35
3.81818 4.35 4.45 3.9
4.35 4.6 4.8 4.05
4.2 4.35 4.35 4
3.45833 4.04762 4.35 3.61905
3.86364 4.35 4.35 3.80952
3.65217 4.04762 4.35 3.47826
3.44 3.95455 4.35 3.80952

minRatio in loop = 3.44 xMin = 9 yMin= 0 E_total= 86 T_Total= 25
3.44 3.95652 4.61905 3.71429
2.92308 3.2 3.44 2.84
3.44444 3.88 4.2 3.44
3.07407 3.44 3.52 3.08
3.44 3.64 3.8 3.2
3.32 3.44 3.44 3.16
2.82759 3.23077 3.44 2.88462
3.11111 3.44 3.44 3.03846
2.96429 3.23077 3.44 2.82143
3.44 3.95455 4.35 3.80952

minRatio in loop = 2.82143 xMin = 8 yMin= 3 E_total= 79 T_Total= 28
2.82143 3.23077 3.75 2.95833
2.46429 2.60714 2.82143 2.28571
3.07143 3.21429 3.5 2.82143
2.71429 2.82143 2.89286 2.5
2.82143 3.11111 3.38462 2.80769
2.71429 2.82143 2.82143 2.57143
2.58621 2.75 2.82143 2.42857
2.75 2.82143 2.82143 2.57143
2.96429 3.23077 3.44 2.82143
2.82143 3.2 3.47826 3.04167
```

Iterations 4, 5, 6 and 7 are shown in the below figure.

```
minRatio in loop = 2.28571 xMin = 1 yMin= 3 E_total= 64 T_Total= 28
2.28571 2.65385 3.125 2.33333
2.46429 2.60714 2.82143 2.28571
2.53571 2.67857 2.96429 2.28571
2.17857 2.28571 2.35714 1.96429
2.28571 2.55556 2.80769 2.23077
2.17857 2.28571 2.28571 2.03571
2.06897 2.21429 2.28571 1.89286
2.21429 2.28571 2.28571 2.03571
2.42857 2.65385 2.84 2.28571
2.28571 2.6 2.82609 2.41667

minRatio in loop = 1.89286 xMin = 6 yMin= 3 E_total= 53 T_Total= 28
1.89286 2.23077 2.66667 1.875
2.07143 2.21429 2.42857 1.89286
2.06897 2.28571 2.57143 1.89286
1.78571 1.89286 1.96429 1.57143
1.89286 2.14815 2.38462 1.80769
1.78571 1.89286 1.89286 1.64286
2.06897 2.21429 2.28571 1.89286
1.82143 1.89286 1.89286 1.64286
2.03571 2.23077 2.30769 1.89286
1.89286 2.16 2.34783 1.95833

minRatio in loop = 1.57143 xMin = 3 yMin= 3 E_total= 44 T_Total= 28
1.57143 1.88462 2.29167 1.5
1.75 1.89286 2.10714 1.57143
1.75862 1.96429 2.25 1.57143
1.78571 1.89286 1.96429 1.57143
1.57143 1.81481 2.03846 1.46154
1.46429 1.57143 1.57143 1.32143
1.75862 1.89286 1.96429 1.57143
1.5 1.57143 1.57143 1.32143
1.71429 1.88462 1.96154 1.57143
1.57143 1.8 1.95652 1.58333

minRatio in loop = 1.32143 xMin = 5 yMin= 3 E_total= 37 T_Total= 28
1.32143 1.61538 2 1.20833
1.5 1.64286 1.85714 1.32143
1.51724 1.71429 2 1.32143
1.53571 1.64286 1.71429 1.32143
1.32143 1.55556 1.76923 1.19231
1.46429 1.57143 1.57143 1.32143
```

Iterations 8 and 9 are displayed below:

```
minRatio in loop = 1.32143 xMin = 5 yMin= 3 E_total= 37 T_Total= 28
1.32143 1.61538 2 1.20833
1.5 1.64286 1.85714 1.32143
1.51724 1.71429 2 1.32143
1.53571 1.64286 1.71429 1.32143
1.32143 1.55556 1.76923 1.19231
1.46429 1.57143 1.57143 1.32143
1.51724 1.64286 1.71429 1.32143
1.25 1.32143 1.32143 1.07143
1.46429 1.61538 1.69231 1.32143
1.32143 1.52 1.65217 1.29167

minRatio in loop = 1.07143 xMin = 7 yMin= 3 E_total= 30 T_Total= 28
1.07143 1.34615 1.70833 0.916667
1.25 1.39286 1.60714 1.07143
1.27586 1.46429 1.75 1.07143
1.28571 1.39286 1.46429 1.07143
1.07143 1.2963 1.5 0.923077
1.21429 1.32143 1.32143 1.07143
1.27586 1.39286 1.46429 1.07143
1.25 1.32143 1.32143 1.07143
1.21429 1.34615 1.42308 1.07143
1.07143 1.24 1.34783 1
```

After 9 iterations, we obtain the most optimized results where the “Total\_Energy” was calculated to be 30 units and “Total\_time” was calculated to be 28 time units.

### **Program Running Time:**

The running time of the program was calculated using the clock\_t function and was found to be approximately around 1810 ms as displayed in the figure below.

```
Program running time: 1810 ms
```