ENERGY AND PERFORMANCE-AWARE TASK SCHEDULING IN A MOBILE CLOUD COMPUTING ENVIRONMENT IN C++

BY
SATISH KUMAR ANBALAGAN
(NUID:001351994)
GANESHRAM KANAKASABAI
(NUID:001475047)

OVERVIEW OF THE PROJECT

INTRODUCTION TO DATA STRUCTURE AND RESCHEDULING PROCESS

TESTING SAMPLES

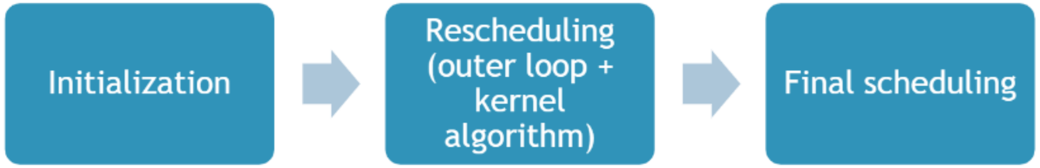
CODE

ABSTRACT OF THIS PROJECT

The high scientific applications which contain thousands of tasks are usually executed in virtualized cloud for many benefits. With the increment of the processing capability of the cloud system, the computation energy is significantly consumed along. Thus efficient energy consumption methods are quite necessary to save the energy cost. In this paper, the independent task scheduling problem in a cloud data center is considered. It is a big challenge to achieve the tradeoff between the minimization of computation energy and user-defined deadlines

DATA STRUCTURE AND PROCESS

- Initialization : Assigning each tasks to each core
- Rescheduling and final scheduling : Final assignment and to optimize schedule tasks.



TEST SAMPLE 1

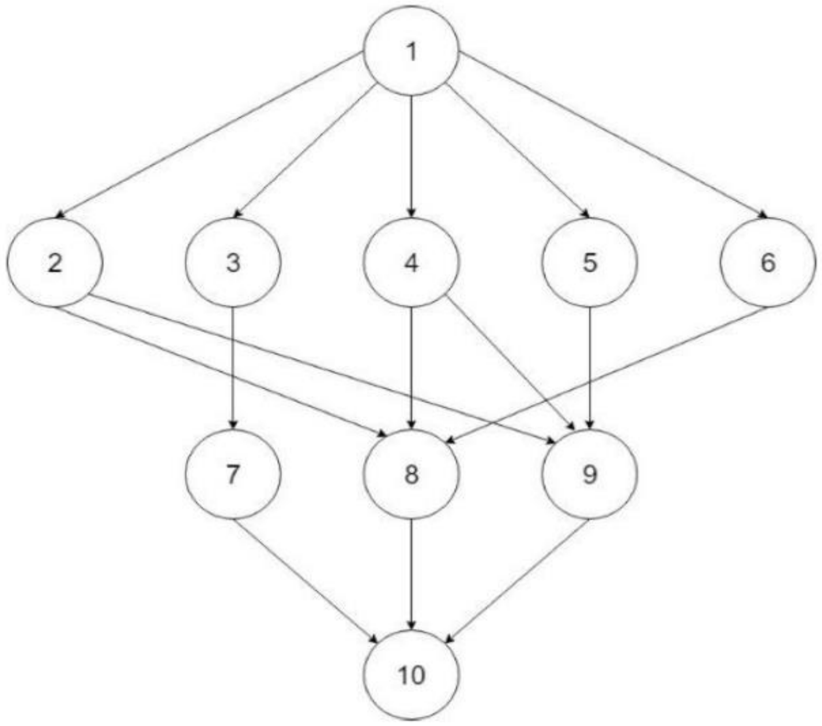


Figure 1. Task graph 1

Example 1 items

STEP 1 : INITIAL ASSIGNMENT

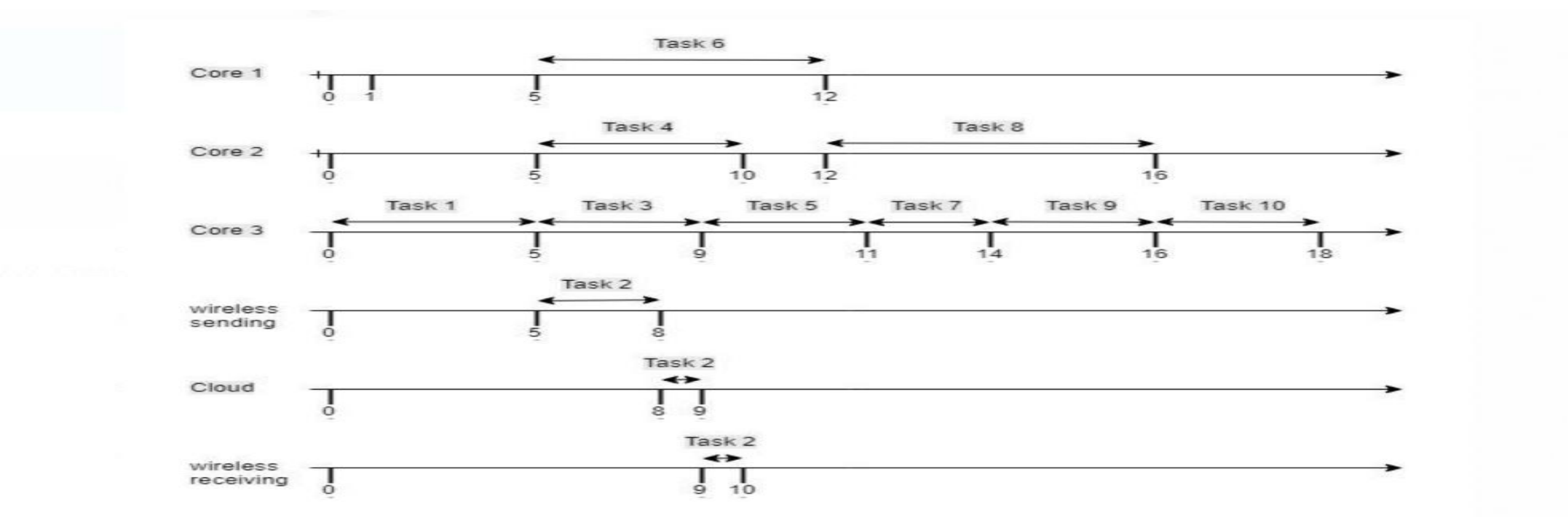


Figure 2. Initial scheduling diagram

Total Energy consumed (E_total)= 122 units
Total Execution time (T_total) = 18 units Thus, the maximum time for the next step should not exceed Tmax = 1.5·Ttotal = 27 units

STEP 2 : AFTER RUNNING OUTER LOOP +KERNEL ALGORITHM MANY TIMES

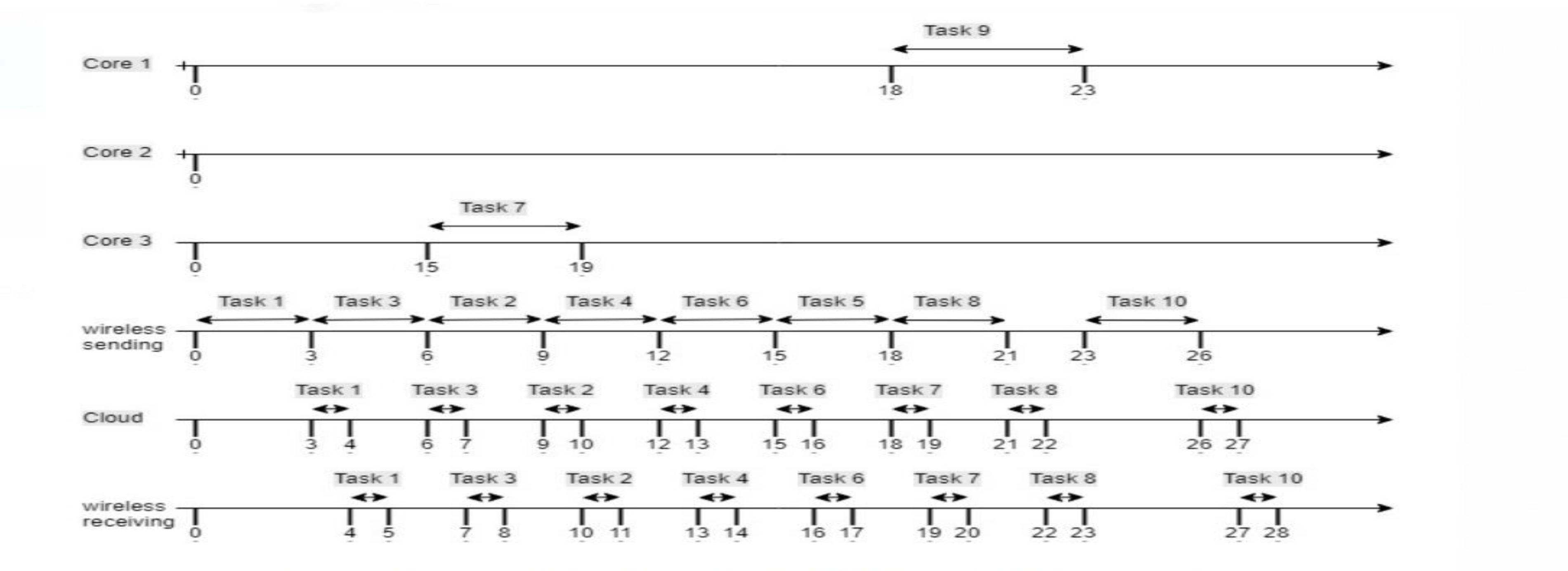


Figure 3. Task scheduling diagram by the MMC task scheduling algorithm

Total Energy Consumption (E_total) = 29 units
Total Execution Time (T_total) = 28 units
Total running time = 1.259 ms

TEST SAMPLE 2

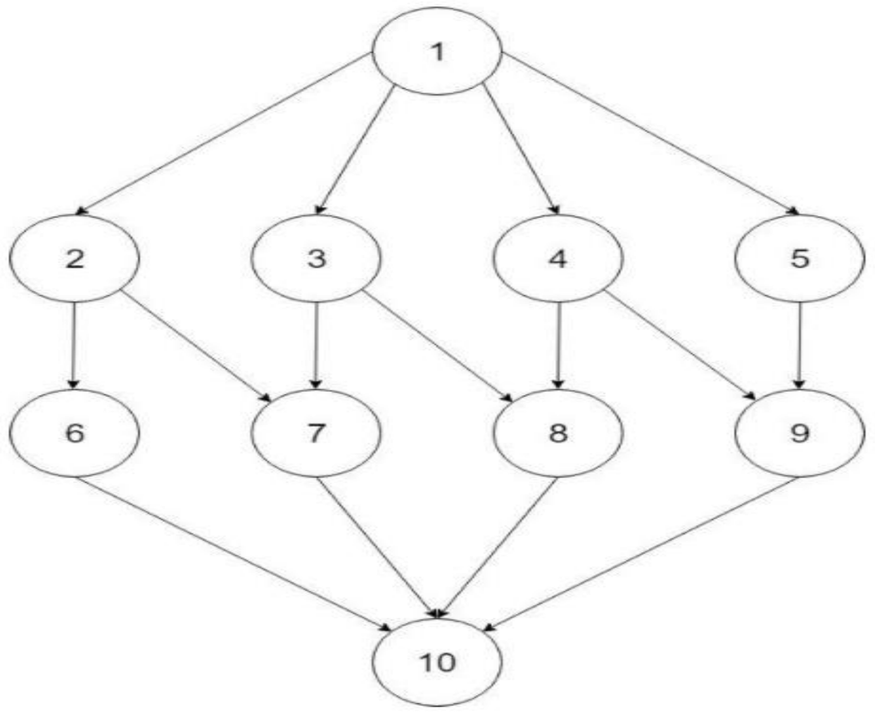


Figure 4. Task graph 2

example 2 -

STEP 1 : INITIAL ASSIGNMENT

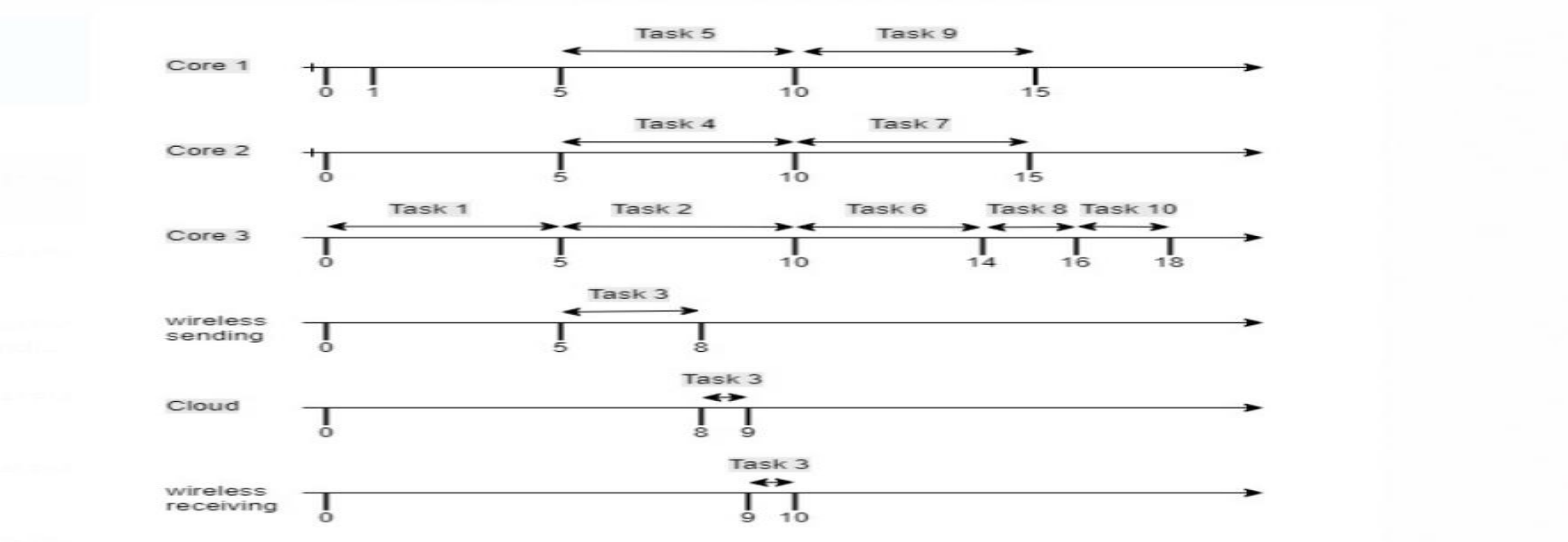


Figure 5. Initial scheduling diagram

Total energy consumption (E_total) = 123 units
Total execution time (T_total) = 18 units Thus, the maximum time for the next step should not exceed Tmax = 1.5·Ttotal = 27 units

STEP 2 : AFTER RUNNING OUTER LOOP +KERNEL ALGORITHM MANY TIMES

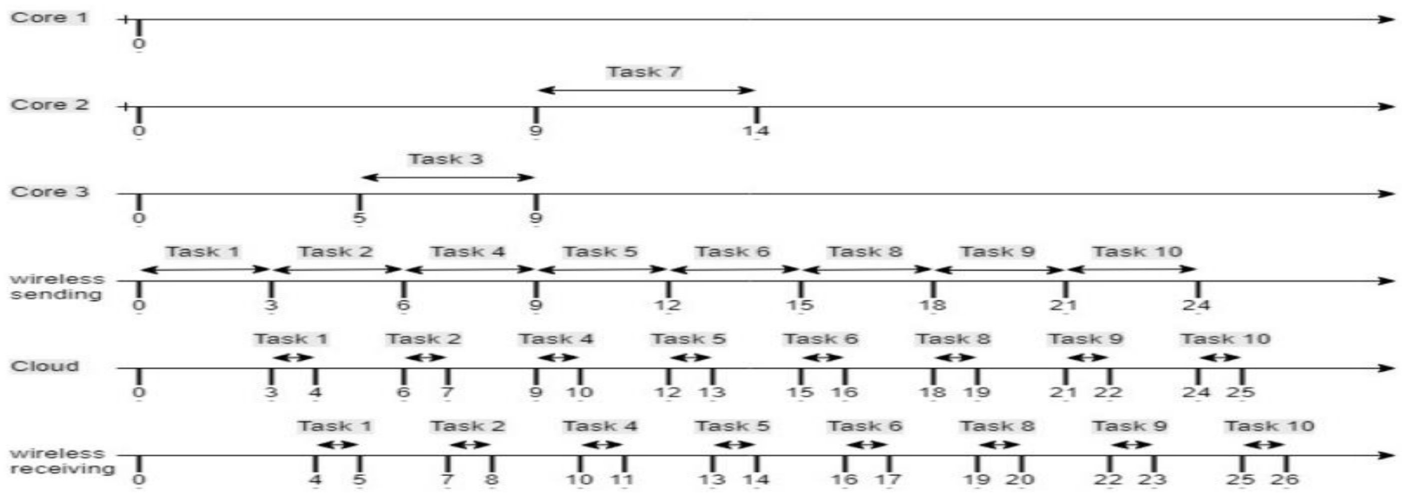


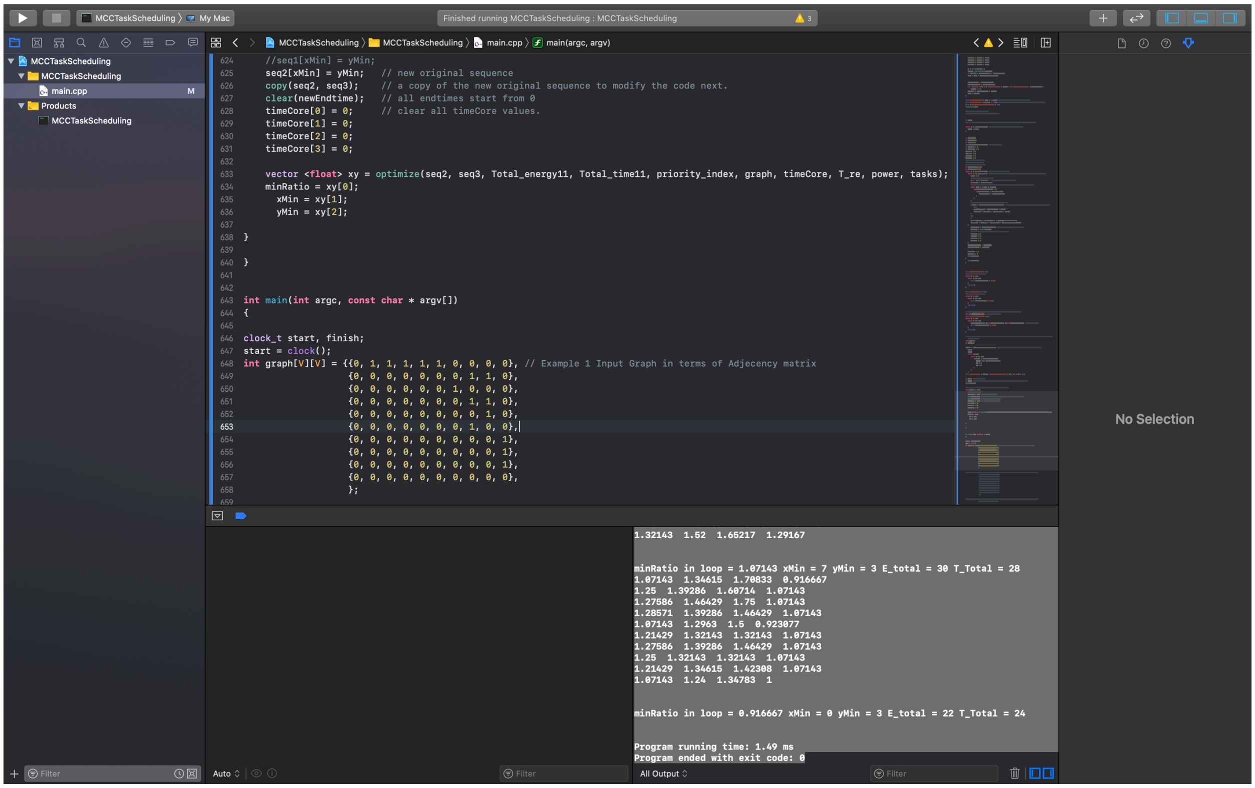
Figure 6. Task scheduling diagram by the MMC task scheduling algorithm

Total energy consumption (E_total) = 31 units
Total execution time (T_total) = 26 units
Total running time = 1.504 ms

CONTRIBUTIONS

- SAMPLE 1 TESTING AND ANALYSIS : GANESHRAM KANAKASABAI
- SAMPLE 2 TESTING AND ANALYSIS : SATISH KUMAR ANBALAGAN
- REPORT : GANESHRAM KANAKASABAI AND SATISH KUMAR ANBALAGAN
- CODE REPORT ANALYSIS : SATISH KUMAR ANBALAGAN AND GANESHRAM KANAKASABAI

ScreenShots:



Code :

```
//
//  main.cpp
//  MCCTaskScheduling
//
//  Created by Satish Kumar Anbalagan on 12/1/19.
//  Copyright © 2019 Satish Kumar Anbalagan. All rights reserved.
//

#include<iostream>
#include<cstdlib>
#include<vector>
#include<algorithm>
#include<time.h>
using namespace std;

# define V 10        // The number of tasks
# define C 3         // Number of cores
```

```
void copy(int arr[V], int arr1[V]){          // To duplicate the arrays for further manipulations
   for(int i=0; i<V; i++){
      arr1[i] = arr[i];
   }
}

int maximum(int arr[C]){             // To find maximum in an array of C elements
   int maxi = arr[0];
   for(int i=1; i<C+1; i++){
      if(arr[i] > maxi){
         maxi = arr[i];
      }
   }
   return maxi;
   }


void reset(int seq1[V], int seq[V]){   // reset the elements of the intial scheduling sequence
during task migration step.
   for(int i=0; i<V; i++){
      seq1[i] = seq[i];
   }
}


int minimum(int arr[], int n){      //  Maximum of an array with 4 elements
   int mini = arr[0];
   for(int i=0; i<=n; i++){
      if(arr[i] < mini){
         mini = arr[i];
      }
   }
   return mini;
}

int minimum_index(int arr[], int n){  // Find the index of the minimum element present in the
array.
   int min_index = 0;
   int mini = arr[0];
   for(int i=0; i<=n; i++){
      if(arr[i] < mini){
         mini = arr[i];
         min_index = i;
```

```
        }
    }
    return min_index;
}

int clear1(int arr[V]){           // makes all ele = -1
    for (int i=0; i<V; i++){
        arr[i] = -1;
    }
    return arr[V];
}

int checkifpresent(int a, int arr[V]){   // if ele a present in array arr, 1 is returned
    int ans=0;
    for(int i=0;i<V;i++){
        if(arr[i] == a){
            ans = 1;
        }
    }
return ans;
}



void unlock(int graph[V][V], int Exe_current, int unlocked_tasks[V]){        // to unlock all the
successors
for(int j=0; j<V; j++){
    if(graph[Exe_current][j] == 1){
        unlocked_tasks[j] = j;    // So, Now all positive values in Unlocked_tasks are ready to be
scheduled.
    }
}
}

void reverseArray(int arr[], int start, int end)   // to reverse an array
{
    while (start < end)
    {
        int temp = arr[start];
        arr[start] = arr[end];
        arr[end] = temp;
        start++;
        end--;
    }
```

```
}

int clear(int arr[V]){              // to make all elements in that array = 0;
   for (int i=0; i<V; i++){
      arr[i] = 0;
   }
   return arr[V];
}

void printarray1D(int arr[],int n){    //  A Function to print 1D arrays
   for(int i=0;i<V;i++){

      cout <<arr[i]<< "\t";
   }

}

void printvector1D(vector<int>arr){    // Print 1 Dimensional arrays
   for(int i=0; i<arr.size(); i++){
      cout << arr[i] << "\t";
   }
}

// Function to optimize the sequence untill we get desired results.
vector <float> optimize(int seq2[V], int seq3[V], int Total_energy11[V][C+1], int
Total_time11[V][C+1], int priority_index[V], int graph[V][V],  int timeCore[C], int T_re[V], int
power[C],int tasks[V][V]){
   vector<float> returnval;
   int presentTask;
   int newEndtime[V];
   clear(newEndtime);
   int energyTotal = 0;
   int finishTime = 0;
   float energyTimeRatio[V][C+1];
   returnval.clear();

   for(int i=0; i<V; i++){             // each element in the sequence
   for(int j=0; j<C+1; j++){         // change each element for each core and iterate to calculate
total power and energy
      seq3[i] = j;
      //presentTask = priority_index[i];
      for(int k=0; k<V; k++){       // to iterate through all tasks in seq1
      presentTask = priority_index[k];
```

```
    // Here check if parents have executed the tasks, if not then change the newendtimes
values.
    for(int parent = 0; parent <V; parent++){
      if(graph[parent][presentTask] == 1){
        if(newEndtime[parent] > timeCore[seq3[k]]){
          timeCore[seq3[k]] = newEndtime[parent];
        }
      }
    }
    // Calculate the finish time on the corresponding core
    if(seq3[k] == 3){                          // if it has to be executed on cloud, then tasks[task][3]
wouldn't exist
      timeCore[seq3[k]] = timeCore[seq3[k]] + 3;
      energyTotal = energyTotal + 0.5*3;
    }
    else
    {
    timeCore[seq3[k]] = timeCore[seq3[k]] + tasks[presentTask][seq3[k]];
    energyTotal = energyTotal + power[seq3[k]] * tasks[presentTask][seq3[k]];
  }

    if(seq3[k] == 3){
    newEndtime[k] = timeCore[seq3[k]] + 2;     // Update the new endtime over that task
  }
  else
  {
  newEndtime[k] = timeCore[seq3[k]];
  }

    finishTime = maximum(timeCore);
    // Now clear the timeCore values for the next iteration
    timeCore[0] = 0;
    timeCore[1] = 0;
    timeCore[2] = 0;
    timeCore[3] = 0;
}
  Total_energy11[i][j] = energyTotal;
  Total_time11[i][j] = finishTime;
 //   finishTime2.push_back(finishTime);
  //Total_energy1.push_back(energyTotal);
  //Total_time1.push_back(finishTime);
  energyTotal = 0;
  finishTime = 0;
  reset(seq3, seq2);
```

```
}
  reset(seq3, seq2);
}

for(int i=0; i<V; i++){
  for(int j=0; j<=C; j++){
    energyTimeRatio[i][j] = (float)Total_energy11[i][j] / (float)Total_time11[i][j];    // int float
casting.
    cout << energyTimeRatio[i][j] << "  ";
  }
  cout << endl;
}
float minRatio = energyTimeRatio[0][0];             // finding minRatio and its indices for first
iteration manually.
  int xMin=0;
  int yMin=0;
  for(int i=0;i<V;i++){
    for(int j=0; j<=C; j++){
      if(minRatio > energyTimeRatio[i][j]){
        minRatio =     energyTimeRatio[i][j];
        xMin = i;
        yMin = j;
      }
    }
  }
  cout << "\n\nminRatio in loop = " << minRatio << " xMin = " << xMin << " yMin = " << yMin <<
" E_total = " << Total_energy11[xMin][yMin] << " T_Total = " << Total_time11[xMin][yMin] <<
endl;
returnval.push_back(minRatio);
returnval.push_back(xMin);
returnval.push_back(yMin);


return returnval;
}

void Task_Scheduling(int graph[V][V], int tasks[V][V], int Ts, int Tc, int Tr){  // Main
Task_scheduling Algorithm
  int T_L_min[10];

  for(int i=0; i<10;i++){                // Calculating T_L_min
      int min_temp = tasks[i][0];
      for (int j = 0; j <2; j++) {
      min_temp > tasks[i][j+1];
```

```
            min_temp =  tasks[i][j+1];
        }
        T_L_min[i] = min_temp;
    }

int T_re[V];                        // Calculating T_Re
    for(int i=0; i<V;i++){
        T_re[i] = Ts+Tc+Tr;
    }

int cloud_task[V];                   // Assigning boolean to the cloud task
for(int i=0;i<V;i++){

    if(T_re[i]< T_L_min[i]){           // Add <= if cloud tasks need to be considered.
        cloud_task[i]=1;
    }
    else{
        cloud_task[i]=0;
    }
}

// Primary assignment ends
int w[10]; // Task Prioritizing
int sum=0;
int Avg=0;
for(int i=0; i<V;i++){              // Calculating the W value
    if(cloud_task[i] == 1){
        w[i]= T_re[i];
    }
    else{
        sum=0;
        Avg=0;
        for(int j=0; j<C; j++){
            sum = sum + tasks[i][j];
             Avg = sum/3;
        }

        w[i] = Avg;
    }
}


int sum1=0;
int priority[V];
```

```
for(int i=0; i<V; i++){        // priority
   int temp[V];
   sum1=0;
   for(int k=0; k<V; k++){
      temp[k] = graph[i][k];
      sum1 = sum1+graph[i][k];    // temp has 1 row after one iteration
   }
   if(sum1 == 0){
      priority[i]= w[i];         // if it is an end task, i.e no 1's in all 10 elements

   }

}

// Finding priority without recursion

int prior[V];
clear(prior);
int max_prior;
for(int i=V-1; i>=0; i--){      // bottoms-up approch. For each vertex
   for(int k=0; k<V; k++){      // checking for successors at each vertex
      if(graph[i][k] == 1){
         prior[k] = (priority[k]);
      }
   }

   for(int h=0; h<V; h++){
      if(max_prior < prior[h]){
         max_prior = prior[h];
      }
   }
   clear(prior);
   priority[i] = w[i] + max_prior;
   max_prior=0;
}

int priority_sorted[V]; // Just duplicating priority as priority1, priority2 for further
manipulations
int priority2[V];
for(int i=0;i<V;i++){
   priority_sorted[i] = priority[i];
   priority2[i] = priority[i];
}
```

```
cout << "Priority" << endl;
   printarray1D(priority, V);

// Sorting the priorities in Descending order
// Also, printing the sorted array
sort(priority_sorted, priority_sorted+V);
 cout << "\nSorted Priority Array looks like this:" << endl;
   for (size_t i = 0; i !=V; ++i){
      cout << priority_sorted[i] << " ";
}
// Find the index_priorities to know which node has most priority.
int priority_index[V];
for(int i=0; i<V; i++){
   for(int j=0; j<V; j++){
      if(priority_sorted[i] == priority2[j]){
         priority_index[i] = j;
      }
   }
}

//Convert the priority_index from ascending to descending order
reverseArray(priority_index, 0, V-1);
priority_index[2] = 1;          // For the given graph.
priority_index[3] = 3;


   //cout<<"\nunlocked elements" << endl;
   //printarray1D(unlocked_tasks, V);

   cout<<"\npriority_index" << endl;
   printarray1D(priority_index,V);

   cout<<"\ncloud_task"<<endl;
   printarray1D(cloud_task, V);

   cout<<"\nT_re"<<endl;
   printarray1D(T_re, V);

   cout<<"\nT_L_min" << endl;
   printarray1D(T_L_min, V);

   cout<<"\nw" << endl;
   printarray1D(w, V);
```

```
/*
 Execution Unit Selection Algorithm
*/

//int secs = 0;          // To count the execution time
int executed_tasks[V];
//int unexecuted_tasks[V];
int unlocked_tasks[V];
//int exe_time[C+1];      // 3 cores and cloud. // Simultaneously to allot time
//int count_unlocked=0;
int pred[V];
int count5=0;

for(int i=0; i<V;i++){     // Clearning junk values
   unlocked_tasks[i] = -1;
   executed_tasks[i] = -1;
   pred[i] = -1;
}

int Exe_current = priority_index[0];  // Starting Execution on the first task.
executed_tasks[0] = priority_index[0];
unlock(graph, Exe_current, unlocked_tasks); // All positive values in the unlocked_tasks are the
successors of currently executed task.

for(int hp=0; hp<V; hp++){  // hp = high priority
if(checkifpresent(priority_index[hp], unlocked_tasks) == 1){   // checking if priority index[hp] is
present in unlocked_tasks array.
// Now check if all the predecessors of hp element are already executed.
for (int p = 0; p<V; p++){      // p= pred
   if(graph[p][hp] == 1){
      pred[count5] = p;
      count5 +=1;
   }
}
// bool flag=1;
int ans[V];
   for(int cnt = 0; cnt<=count5; cnt++){
      if(checkifpresent(pred[cnt], executed_tasks) == 1){
         ans[cnt] == 1;
      }
}
```

```
bool flag = 0;
for(int cnt1 = 0; cnt1 <count5; cnt1++){
   if(ans[cnt1] != 1){
      flag = 1;
      goto statement;
   }
}              /*    // If flag = 0 that means all its pred are executed, else some of the pred are not
executed.    */

// Schedule the cores and the cloud for the task
unlock(graph, priority_index[hp], unlocked_tasks);
executed_tasks[hp] = hp;
priority_index[hp] = -1;
count5=0;
cout << "hey";
}
statement:
   cout<< " ";      // if there is a situation where all its pred ele are not executed still, then print
that ele.
}

//int core1 = 0, core2=0, core3=0, cloud=0;
int tcore1 =0, tcore2=0, tcore3=0, tcloud = 0;

// Time schedule for task 1:
tcloud = T_re[0];
tcore1 = tasks[0][0];
tcore2 = tasks[0][1];
tcore3 = tasks[0][2];
int endtime[C+1];

endtime[0] = tcore1;
endtime[1] = tcore2;
endtime[2] = tcore3;
endtime[3] = tcloud;

int min = minimum(endtime, C);
int first_minIndex = minimum_index(endtime, C);

for(int i=0; i<C+1;i++){
   endtime[i] = min;
}

int minIndex;
```

```
int tendtime[C+1];  // temp endtime
int task_endtime[V];
task_endtime[0] = endtime[0];   // task 1 end time

// now, program working till 1st iteration, now iterate to all other nodes of the graph to
determine the final endtime of execution
int power[C+1];
power[0] = 1;
power[1] = 2;
power[2] = 4;
power[3] = Ts * 0.5;     // sending time x 0.5;
int energy = 0;
int Total_time = 0;
energy = endtime[0] * power[first_minIndex];  // assigning energy

//cout << "first_minIndex is " << first_minIndex << endl;
//cout << "\n\nTask " << 1 << " was executed in " << first_minIndex + 1 << " with an endtime of
" << endtime[0] << endl;
int initTime = endtime[0];
int seq[V];  // initial scheduling

seq[0] = first_minIndex;

for(int p=0; p<V; p++){
   int pTask = priority_index[p];     // present task starts from the second element in priority
matrix.
   tcloud = T_re[pTask];
   tcore1 = tasks[pTask][0];
   tcore2 = tasks[pTask][1];
   tcore3 = tasks[pTask][2];
   // check if the parents 'task_endtimes' are greater than the 'endtime'. If they are greater then
update the 'endtime'
   for(int h=0; h<V; h++){             // h = each of the parent
      if(graph[h][pTask] == 1){         // find parents
         for(int temp=0; temp<C+1; temp++){   // check if c1,c2,c3 or cloud times are less than
taskend time of parent, then update it
         if(task_endtime[h] > endtime[temp]){ // because all parents need to be executed before
their children
            endtime[temp] = task_endtime[h];
         }
      }
   }
}
```

```
   tendtime[0] = endtime[0] + tcore1;
   tendtime[1] = endtime[1] + tcore2;
   tendtime[2] = endtime[2] + tcore3;
   tendtime[3] = endtime[3] + tcloud;

   min = minimum(tendtime, C);
   minIndex = minimum_index(tendtime,C);
   int time_taken = tendtime[minIndex] - endtime[minIndex];
   energy = energy + time_taken*power[minIndex];


   endtime[minIndex] = tendtime[minIndex];
   task_endtime[pTask] = tendtime[minIndex];
   cout << "\nTask " << pTask + 1<< " was executed in " << minIndex + 1 << " with an endtime of
" << tendtime[minIndex] - initTime << endl;
   Total_time = tendtime[minIndex] - initTime;
   seq[p] = minIndex;
}

cout << "\nTotal Energy = " << energy << endl;      // Total energy of the time scheduled
Algorithm
cout << "\nTotal Time = " << Total_time << endl;     // Total time taken for the time scheduled
algorithm to be executed.
cout << "\nInitial Scheduling Result Seq = " << endl;
printarray1D(seq,V);

/* Initial Scheduling ends here */
/* Task Migration Algorithm - Working till Here */


int seq1[V];
//int time_power[V-1][C-1];         // store the totol power and total energy for all N*K
iterations.

for(int t=0; t<V; t++){             // Duplicating seq values for further manipulations
   seq1[t] = seq[t];
}


int presentTask;
int newEndtime[V];
int timeCore[C+1];
clear(newEndtime);              // clear junk values
int finishTime = 0;
```

```
int energyTotal = 0;
timeCore[0] = 0;
timeCore[1] = 0;
timeCore[2] = 0;
timeCore[3] = 0;
//vector <int> Total_energy1;
//vector <int> Total_time1;
//vector <double> finishTime2;
int Total_energy11[V][C+1];
int Total_time11[V][C+1];
for(int i=0; i<V; i++){            // each element in the sequence
  for(int j=0; j<C+1; j++){        // change each element for each core and iterate to calculate
total power and energy
      seq1[i] = j;
      //presentTask = priority_index[i];
      for(int k=0; k<V; k++){      // to iterate through all tasks in seq1
      presentTask = priority_index[k];
      // Here check if parents have executed the tasks, if not then change the newendtimes
values.
      for(int parent = 0; parent <V; parent++){
        if(graph[parent][presentTask] == 1){
          if(newEndtime[parent] > timeCore[seq[k]]){
            timeCore[seq1[k]] = newEndtime[parent];
          }
        }
      }
      // Calculate the finish time on the corresponding core
      if(seq1[k] == 3){                          // if it has to be executed on cloud, then tasks[task][3]
wouldn't exist
        timeCore[seq1[k]] = timeCore[seq1[k]] + T_re[k];
        energyTotal = energyTotal + power[seq1[k]] * T_re[k];
      }
      else
      {
      timeCore[seq1[k]] = timeCore[seq1[k]] + tasks[presentTask][seq1[k]];
      energyTotal = energyTotal + power[seq1[k]] * tasks[presentTask][seq1[k]];
  }
      newEndtime[k] = timeCore[seq1[k]];    // Update the new endtime over that task
      finishTime = maximum(timeCore);
      // Now clear the timeCore values for the next iteration
      timeCore[0] = 0;
      timeCore[1] = 0;
      timeCore[2] = 0;
      timeCore[3] = 0;
```

```
    }
    Total_energy11[i][j] = energyTotal;
    Total_time11[i][j] = finishTime;

    energyTotal = 0;
    finishTime = 0;
    reset(seq1, seq);
  }
    reset(seq1, seq);
  }



  cout << "\nTotal_energy \n" << endl;
  //printvector1D(Total_energy1);
  for(int i=0; i<V; i++){
    for(int j=0; j<=C; j++){
      cout << Total_energy11[i][j] << "\t\t";
    }
    cout << endl;
  }

  cout << "\nTotal_time \n" << endl;
  //printvector1D(Total_time1);
  for(int i=0; i<V; i++){
    for(int j=0; j<=C; j++){
      cout << Total_time11[i][j] << "\t\t";
    }
    cout << endl;
  }

  // Now Find power to time ratio. For each element in total_time and total_energy.
  float energyTimeRatio[V][C+1];  // To store each result
  cout << "\nEnergy Time Ratio " <<endl;
  for(int i=0; i<V; i++){
    for(int j=0; j<=C; j++){
      energyTimeRatio[i][j] = (float)Total_energy11[i][j] / (float)Total_time11[i][j];    // int float
casting.
      cout << energyTimeRatio[i][j] << "\t\t";
    }
    cout << endl;
  }
```

```
// Now calculate the least ratio and the seq1 that produced that results and then optimise that
resursivly untill we get fully optimised results.
float minRatio;
int xMin,yMin;

minRatio = energyTimeRatio[0][0];          // finding minRatio and its indices for first iteration
manually.
  xMin=0;
  yMin=0;
  for(int i=0;i<V;i++){
    for(int j=0; j<=C; j++){
      if(minRatio > energyTimeRatio[i][j]){
        minRatio =    energyTimeRatio[i][j];
        xMin = i;
        yMin = j;
      }
    }
  }
cout << "\nMin Ratio = " << minRatio << " was present at i,j = " << xMin << "  " << yMin << endl;

int seq2[V]; // New original seq
int seq3[V]; // New original seq that needs to be modified and iterated in optimize function.
copy(seq, seq2);

// Optimizing in a while loop until we get the desired results:
while(minRatio > 1.05){
  //seq1[xMin] = yMin;
  seq2[xMin] = yMin;   // new original sequence
  copy(seq2, seq3);    // a copy of the new original sequence to modify the code next.
  clear(newEndtime);   // all endtimes start from 0
  timeCore[0] = 0;     // clear all timeCore values.
  timeCore[1] = 0;
  timeCore[2] = 0;
  timeCore[3] = 0;

  vector <float> xy = optimize(seq2, seq3, Total_energy11, Total_time11, priority_index, graph,
timeCore, T_re, power, tasks);
  minRatio = xy[0];
   xMin = xy[1];
   yMin = xy[2];

}

}
```

```
int main(int argc, const char * argv[])
{

clock_t start, finish;
start = clock();
int graph[V][V] = {{0, 1, 1, 1, 1, 1, 0, 0, 0, 0}, // Example 1 Input Graph in terms of Adjecency
matrix
            {0, 0, 0, 0, 0, 0, 0, 1, 1, 0},
            {0, 0, 0, 0, 0, 0, 1, 0, 0, 0},
            {0, 0, 0, 0, 0, 0, 0, 1, 1, 0},
            {0, 0, 0, 0, 0, 0, 0, 0, 1, 0},
            {0, 0, 0, 0, 0, 0, 0, 1, 0, 0},
            {0, 0, 0, 0, 0, 0, 0, 0, 0, 1},
            {0, 0, 0, 0, 0, 0, 0, 0, 0, 1},
            {0, 0, 0, 0, 0, 0, 0, 0, 0, 1},
            {0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
            };

/*int graph[V][V] = {{0, 1, 1, 1, 1, 0, 0, 0, 0, 0}, // Example 2 Input Graph in terms of Adjecency
matrix
             {0, 0, 0, 0, 0, 1, 1, 0, 0, 0},
             {0, 0, 0, 0, 0, 0, 1, 1, 0, 0},
             {0, 0, 0, 0, 0, 0, 0, 1, 1, 0},
             {0, 0, 0, 0, 0, 0, 0, 0, 1, 0},
             {0, 0, 0, 0, 0, 0, 0, 0, 0, 1},
             {0, 0, 0, 0, 0, 0, 0, 0, 0, 1},
             {0, 0, 0, 0, 0, 0, 0, 0, 0, 1},
             {0, 0, 0, 0, 0, 0, 0, 0, 0, 1},
             {0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
             };*/

/*int graph[V][V] = {{0, 1, 1, 1, 0, 0, 0, 0, 0, 0}, // Example 3 Input Graph in terms of Adjecency
matrix
            {0, 0, 0, 0, 1, 0, 0, 0, 0, 0},
            {0, 0, 0, 0, 1, 1, 0, 0, 0, 0},
            {0, 0, 0, 0, 0, 1, 0, 0, 0, 0},
            {0, 0, 0, 0, 0, 0, 1, 1, 0, 0},
            {0, 0, 0, 0, 0, 0, 0, 1, 1, 0},
            {0, 0, 0, 0, 0, 0, 0, 0, 0, 1},
            {0, 0, 0, 0, 0, 0, 0, 0, 0, 1},
            {0, 0, 0, 0, 0, 0, 0, 0, 0, 1},
            {0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
```

```
        };*/

int tasks[10][10] =  {{9,7,5}, // The execution time of a given task on each core.
                {8,6,5},
                {6,5,4},
                {7,5,3},
                {5,4,2},
                {7,6,4},
                {8,5,3},
                {6,4,2},
                {5,3,2},
                {7,4,2},
                };

int Ts = 3;                        // Sending, Cloud and Receiving Time
int Tc = 1;
int Tr = 1;

Task_Scheduling(graph, tasks, Ts, Tc, Tr);

finish = clock();
cout << "\n\nProgram running time: " << (finish-start)/double(CLOCKS_PER_SEC)*1000 << "
ms" << endl;
return 0;

}
```

Output :

```
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
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 = 122

Total Time = 18

Initial Scheduling Result Seq =

2	**2**	**3**	**1**	**0**	**2**	**2**	**1**	**2**	**2**

Total_energy

91	**96**	**102**	**87**
92	**96**	**102**	**91**
105	**109**	**117**	**102**
99	**102**	**104**	**97**
102	**107**	**111**	**100**
99	**102**	**102**	**99**
98	**100**	**102**	**95**
100	**102**	**102**	**99**
99	**100**	**102**	**99**
101	**102**	**102**	**99**

Total_time

20	**18**	**16**	**16**
17	**16**	**16**	**16**
18	**16**	**16**	**16**
18	**16**	**16**	**16**

16 16 16 16
16 16 16 16
20 17 16 17
16 16 16 16
19 17 16 19
21 18 16 19

Energy Time Ratio
4.55 5.33333 6.375 5.4375
5.41176 6 6.375 5.6875
5.83333 6.8125 7.3125 6.375
5.5 6.375 6.5 6.0625
6.375 6.6875 6.9375 6.25
6.1875 6.375 6.375 6.1875
4.9 5.88235 6.375 5.58824
6.25 6.375 6.375 6.1875
5.21053 5.88235 6.375 5.21053
4.80952 5.66667 6.375 5.21053

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

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
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

minRatio in loop = 0.916667 xMin = 0 yMin = 3 E_total = 22 T_Total = 24

Program running time: 1.49 ms
Program ended with exit code: 0