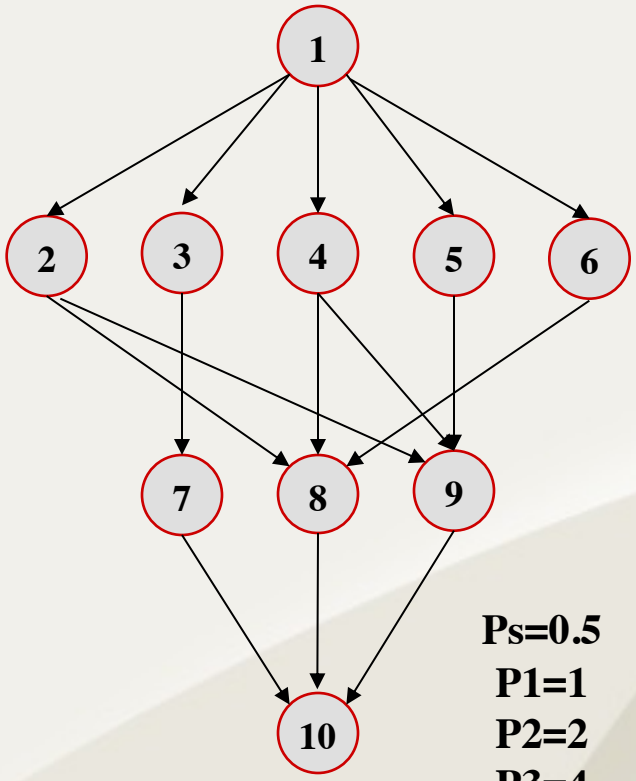
Northeastern UNIVERSITY

EECE-7205

PROJECT2

DANDAN LIN
ECE Department , Northeastern University
December 15,2021

Part 1:Original input



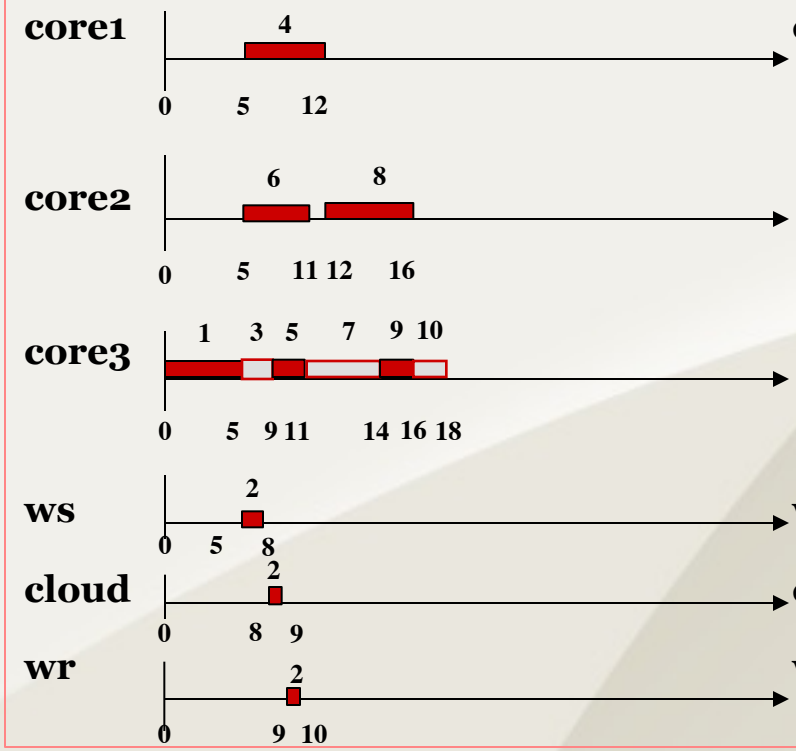
Ps=0.5
P1=1
P2=2
P3=4

Ts=3
Tc=1
Tr=1

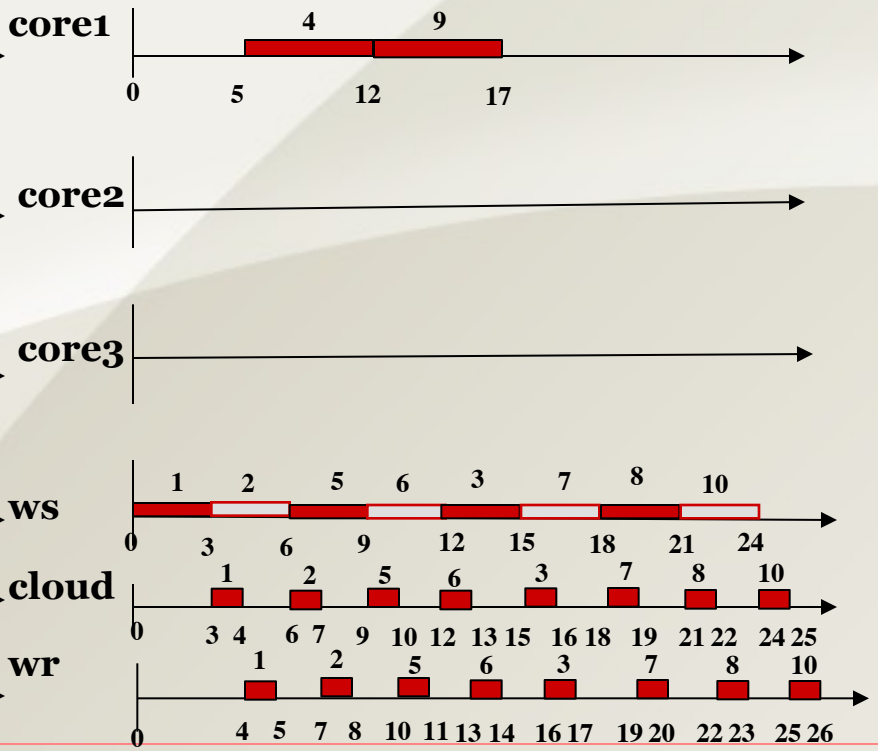
Task	core1	core2	core3
1	9	7	5
2	8	6	5
3	6	5	4
4	7	5	3
5	5	4	2
6	7	6	4
7	8	5	3
8	6	4	2
9	5	3	2
10	7	4	2

Original output

Initial algorithm:



MCC algorithm:



code output

```
initial--The result should be:
cloud: - - - - - 2 2 2
core1 :- - - - - 4 4 4 4 4 4 4
core2 :- - - - - 6 6 6 6 6 6 - 8 8 8 8
core3 :1 1 1 1 1 3 3 3 3 5 5 7 7 7 9 9 10 10
T total: 18
Energy total: 100.5
```

```
Best----The result should be:
cloud: 1 1 1 2 2 2 5 5 5 6 6 6 3 3 3 7 7 7 8 8 8 10 10 10
core1 :- - - - - 4 4 4 4 4 4 4 9 9 9 9 9
core2 :
core3 :
T total: 26
Energy total: 24
```

The initial result T total=18 and E total=100.5,but after using the MCC algorithm , the T total=26 and E total=24.

By getting the loop=8, we can find that there is no further minimum. At the same time the T total is 26 which is smaller than 27 and the energy consumption is smaller too. Because of this ,we can know that the result is feasible.

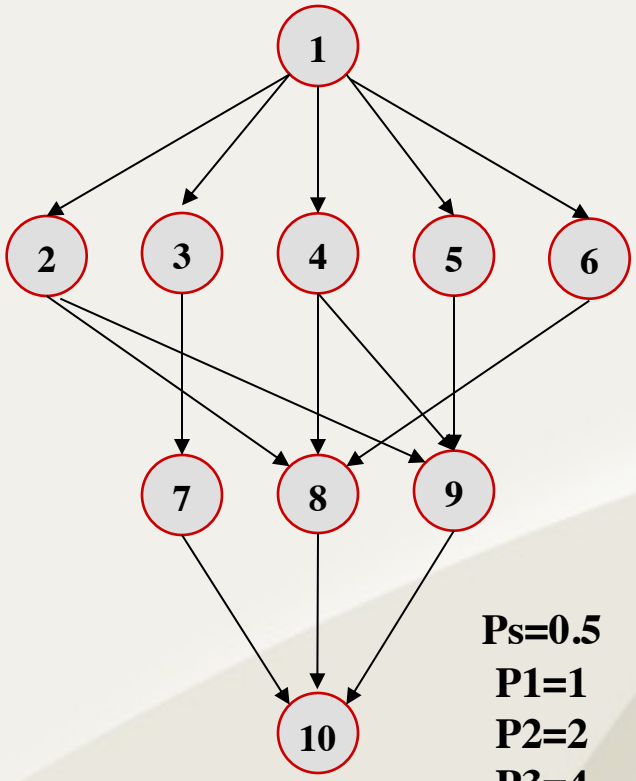
The whole result is like the picture in next slide.

```
The result should be:
cloud: - - - - - 2 2 2 - - - 9 9 9
core1 :- - - - - 3 3 3 3 3 3
core2 :- - - - - 4 4 4 4 4 8 8 8 8
core3 :1 1 1 1 1 6 6 6 6 5 5 7 7 7 - - 10 10
T total: 18
Energy total: 91
initial--The result should be:
cloud: - - - - - 2 2 2
core1 :- - - - - 4 4 4 4 4 4 4
core2 :- - - - - 6 6 6 6 6 6 - 8 8 8 8
core3 :1 1 1 1 1 3 3 3 3 5 5 7 7 7 9 9 10 10
T total: 18
Energy total: 100.5
Best----The result should be:
cloud: 1 1 1 2 2 2
core1 :- - - - - 4 4 4 4 4 4 4
core2 :- - - - - 6 6 6 6 6 6 - 8 8 8 8
core3 :- - - - - 3 3 3 3 5 5 7 7 7 9 9 10 10
T total: 18
Energy total: 82
Best----The result should be:
cloud: 1 1 1 2 2 2 3 3 3
core1 :- - - - - 4 4 4 4 4 4 4
core2 :- - - - - 6 6 6 6 6 6 - 8 8 8 8
core3 :- - - - - 5 5 - - - - 7 7 7 9 9 10 10
T total: 18
Energy total: 67.5
Best----The result should be:
cloud: 1 1 1 2 2 2 3 3 3 7 7 7
core1 :- - - - - 4 4 4 4 4 4 4
core2 :- - - - - 6 6 6 6 6 6 - 8 8 8 8
core3 :- - - - - 5 5 - - - - - 9 9 - - 10 10
```

```
T total: 18
Energy total: 57
Best----The result should be:
cloud: 1 1 1 2 2 2 6 6 6 3 3 3 7 7 7
core1 :- - - - - 4 4 4 4 4 4 4
core2 :- - - - - - - - - - - - 8 8 8 8
core3 :- - - - - 5 5 - - - - - 9 9 - - - 10 10
T total: 19
Energy total: 46.5
Best----The result should be:
cloud: 1 1 1 2 2 2 6 6 6 3 3 3 7 7 7
core1 :- - - - - 4 4 4 4 4 4 4 9 9 9 9 9
core2 :- - - - - - - - - - - - 8 8 8 8
core3 :- - - - - 5 5 - - - - - - - - - - 10 10
T total: 19
Energy total: 43.5
Best----The result should be:
cloud: 1 1 1 2 2 2 6 6 6 3 3 3 7 7 7 8 8 8
core1 :- - - - - 4 4 4 4 4 4 4 9 9 9 9 9
core2 :
core3 :- - - - - 5 5 - - - - - - - - - - - - 10 10
T total: 22
Energy total: 37
Best----The result should be:
cloud: 1 1 1 2 2 2 6 6 6 3 3 3 7 7 7 8 8 8 10 10 10
core1 :- - - - - 4 4 4 4 4 4 4 9 9 9 9 9
core2 :
core3 :- - - - - 5 5
T total: 23
Energy total: 30.5
Best----The result should be:
cloud: 1 1 1 2 2 2 5 5 5 6 6 6 3 3 3 7 7 7 8 8 8 10 10 10
core1 :- - - - - 4 4 4 4 4 4 4 9 9 9 9 9
```

```
Best----The result should be:
cloud: 1 1 1 2 2 2 5 5 5 6 6 6 3 3 3 7 7 7 8 8 8 10 10 10
core1 :- - - - - 4 4 4 4 4 4 4 9 9 9 9 9
core2 :
core3 :
T total: 26
Energy total: 24
no further minimized
no further minimized
sh: pause: command not found
Program ended with exit code: 0
```

Part 2:the other input(1)



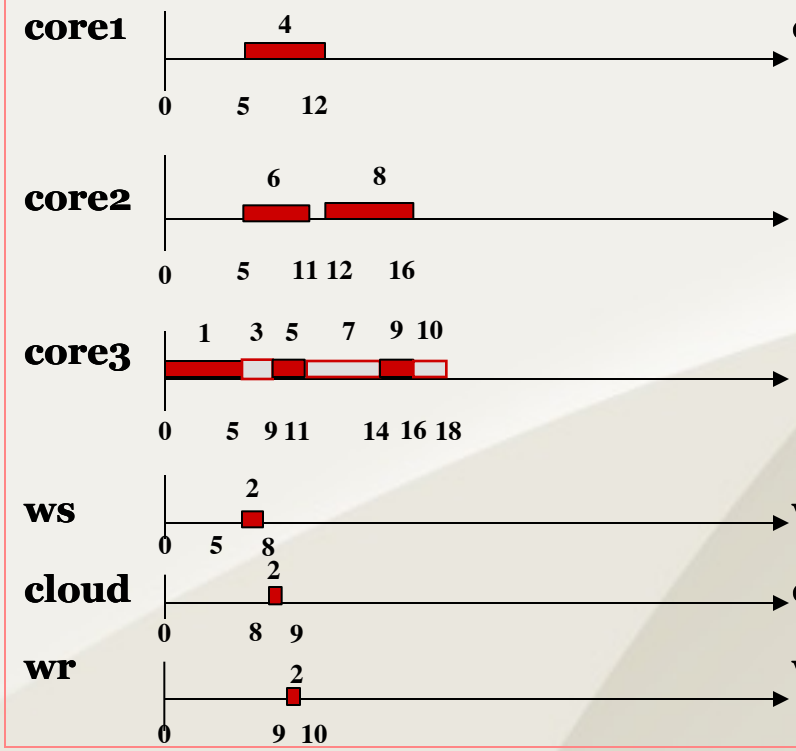
Ps=0.5
P1=1
P2=2
P3=4

Ts=3
Tc=1
Tr=1

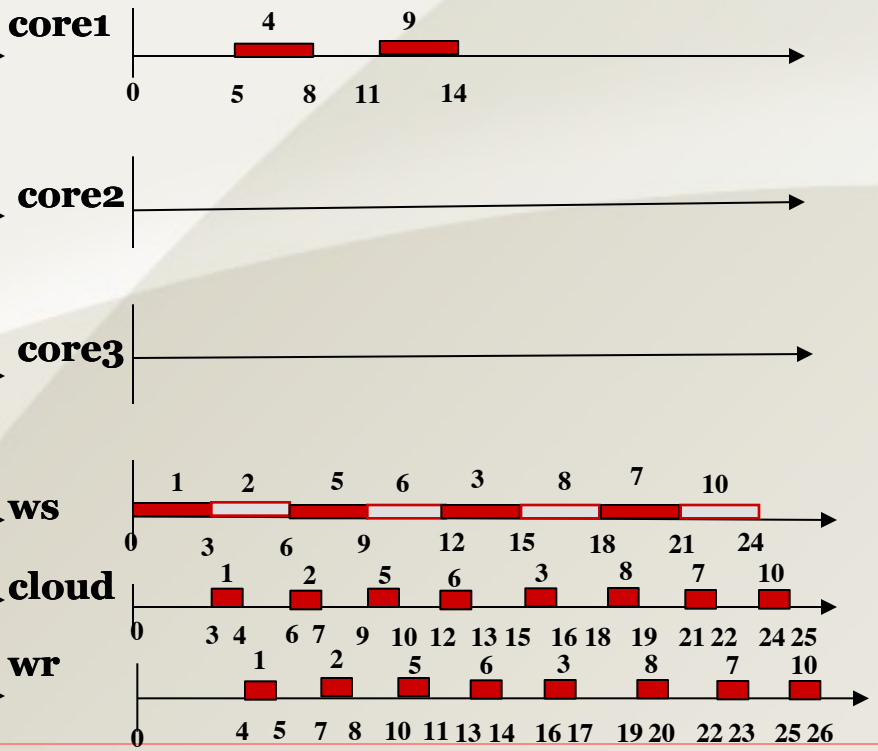
Task	core1	core2	core3
1	5	7	9
2	6	8	5
3	4	5	6
4	3	5	7
5	4	5	2
6	6	7	4
7	5	8	3
8	4	6	2
9	3	5	2
10	4	7	2

output

Initial algorithm:



MCC algorithm:



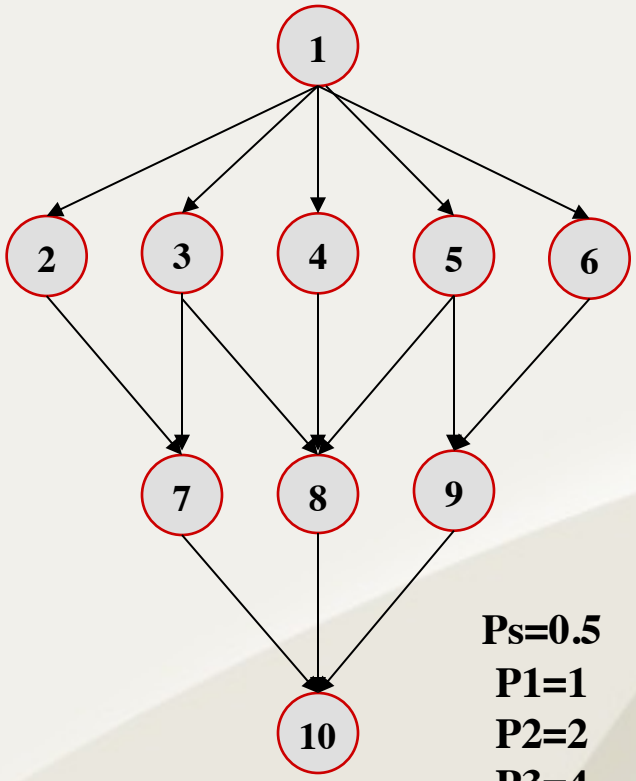
code output

```
initial--The result should be:
cloud: - - - - - 2 2 2
core1 :- - - - - 4 4 4 4 4 4 4
core2 :- - - - - 6 6 6 6 6 6 - 8 8 8 8
core3 :1 1 1 1 1 3 3 3 3 5 5 7 7 7 9 9 10 10
T total: 18
Energy total: 126.5
Best----The result should be:
cloud: 1 1 1 2 2 2 5 5 5 6 6 6 3 3 3 8 8 8 7 7 7 10 10 10
core1 :- - - - - 4 4 4 - - - 9 9 9
core2 :
core3 :
T total: 26
Energy total: 18
```

The initial result T total=18 and E total=126.5,but after using the MCC algorithm , the T total=26 and E total=18.
The whole result is like the picture in next slide.

```
The result should be:
cloud: - - - - - 2 2 2 - - - 9 9 9
core1 :1 1 1 1 1 3 3 3 - 8 8 8 - - 10 10 10 10
core2 :- - - - - 4 4 4 4 4
core3 :- - - - - 6 6 6 6 5 5 7 7 7
T total: 20
Energy total: 66
initial--The result should be:
cloud: - - - - - 2 2 2
core1 :- - - - - 4 4 4 4 4 4 4
core2 :- - - - - 6 6 6 6 6 6 - 8 8 8 8
core3 :1 1 1 1 1 3 3 3 3 5 5 7 7 7 9 9 10 10
T total: 18
Energy total: 126.5
Best----The result should be:
cloud: 1 1 1 2 2 2
core1 :- - - - - 4 4 4 4 4 4 4
core2 :- - - - - 6 6 6 6 6 6 - 8 8 8 8
core3 :- - - - - 3 3 3 3 5 5 7 7 7 9 9 10 10
T total: 18
Energy total: 92
Best----The result should be:
cloud: 1 1 1 2 2 2 3 3 3
core1 :- - - - - 4 4 4 4 4 4 4
core2 :- - - - - 6 6 6 6 6 6 - 8 8 8 8
core3 :- - - - - 5 5 - - - - 7 7 7 9 9 10 10
T total: 18
Energy total: 69.5
Best----The result should be:
cloud: 1 1 1 2 2 2 3 3 3 7 7 7
core1 :- - - - - 4 4 4 4 4 4 4
core2 :- - - - - 6 6 6 6 6 6 - 8 8 8 8
core3 :- - - - - 5 5 - - - - - 9 9 - - 10 10
T total: 18
Energy total: 69.5
Best----The result should be:
cloud: 1 1 1 2 2 2 3 3 3 7 7 7
core1 :- - - - - 4 4 4 4 4 4 4
core2 :- - - - - 6 6 6 6 6 6 - 8 8 8 8
core3 :- - - - - 5 5 - - - - - 9 9 - - 10 10
T total: 18
Energy total: 59
Best----The result should be:
cloud: 1 1 1 2 2 2 3 3 3 7 7 7
core1 :- - - - - 4 4 4 - - - 8 8 8 8
core2 :- - - - - 6 6 6 6 6 6
core3 :- - - - - 5 5 - 9 9 - - - - - 10 10
T total: 17
Energy total: 51
Best----The result should be:
cloud: 1 1 1 2 2 2 3 3 3 7 7 7
core1 :- - - - - 4 4 4 9 9 9 8 8 8 8
core2 :- - - - - 6 6 6 6 6 6
core3 :- - - - - 5 5 - - - - - - - - 10 10
T total: 17
Energy total: 46
Best----The result should be:
cloud: 1 1 1 2 2 2 6 6 6 3 3 3 7 7 7
core1 :- - - - - 4 4 4 9 9 9 8 8 8 8
core2 :
core3 :- - - - - 5 5 - - - - - - - - - - 10 10
T total: 19
Energy total: 33.5
Best----The result should be:
cloud: 1 1 1 2 2 2 6 6 6 3 3 3 7 7 7 10 10 10
core1 :- - - - - 4 4 4 9 9 9 8 8 8 8
core2 :
core3 :- - - - - 5 5
T total: 20
Energy total: 27
Best----The result should be:
cloud: 1 1 1 2 2 2 5 5 5 6 6 6 3 3 3 7 7 7 10 10 10
core1 :- - - - - 4 4 4 - - - 9 9 9 8 8 8 8
core2 :
core3 :
T total: 23
Energy total: 20.5
Best----The result should be:
cloud: 1 1 1 2 2 2 5 5 5 6 6 6 3 3 3 8 8 8 7 7 7 10 10 10
core1 :- - - - - 4 4 4 - - - 9 9 9
core2 :
core3 :
T total: 26
Energy total: 18
```

Part 2:the other input(2)



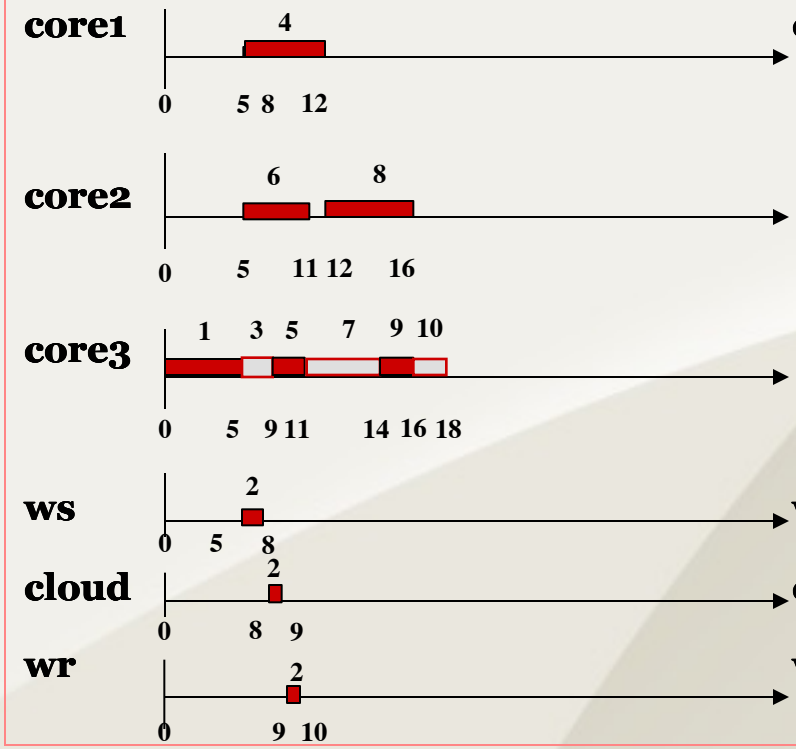
Ps=0.5
P1=1
P2=2
P3=4

Ts=3
Tc=1
Tr=1

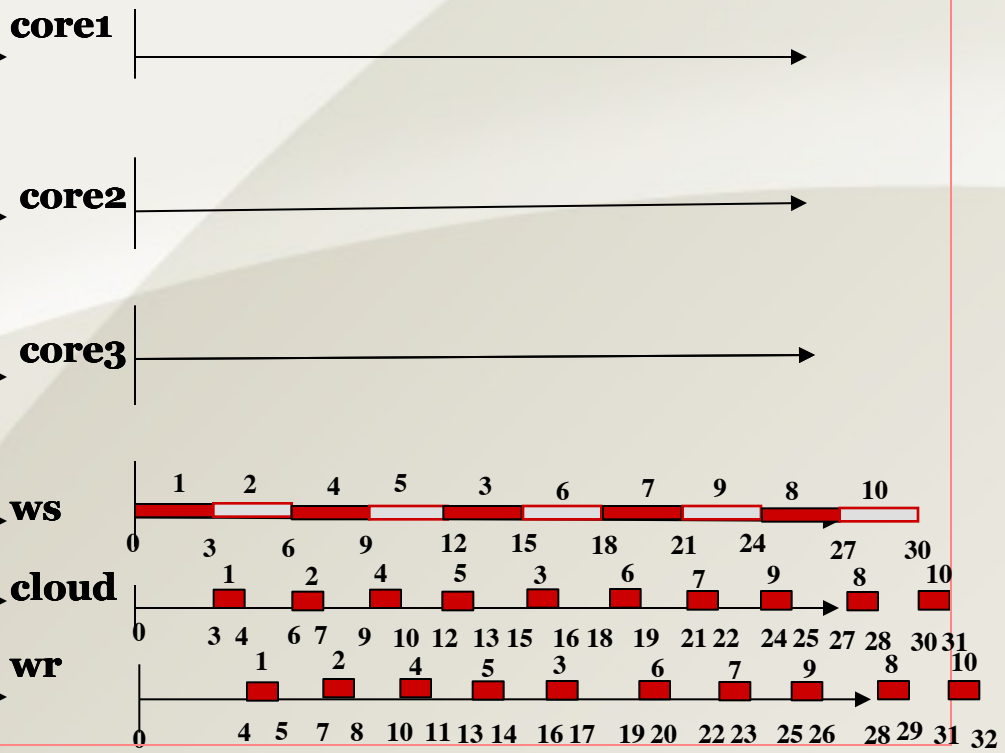
Task	core1	core2	core3
1	7	9	5
2	5	6	3
3	5	4	2
4	7	5	3
5	5	4	2
6	8	6	4
7	4	5	3
8	6	4	2
9	5	3	2
10	7	4	2

output

Initial algorithm:



MCC algorithm:



code output

```
initial--The result should be:
cloud: - - - - - 2 2 2
core1 :- - - - - 4 4 4 4 4 4 4
core2 :- - - - - 6 6 6 6 6 6 - 8 8 8 8
core3 :1 1 1 1 1 3 3 3 3 5 5 7 7 7 9 9 10 10
T total: 18
Energy total: 92.5
```

```
Best----The result should be:
cloud: 1 1 1 2 2 2 4 4 4 5 5 5 3 3 3 6 6 6 7 7 7 9 9 9 8 8 8 10 10 10
core1 :
core2 :
core3 :
T total: 32
Energy total: 15
```

The initial result T total=18 and E total=100.5,but after using the MCC algorithm , the T total=26 and E total=24.

The whole result is like the picture in next slide.

```
initial--The result should be:
cloud: - - - - - 2 2 2
core1 :- - - - - 4 4 4 4 4 4 4
core2 :- - - - - 6 6 6 6 6 6 - 8 8 8 8
core3 :1 1 1 1 1 3 3 3 3 5 5 7 7 7 9 9 10 10
T total: 18
Energy total: 92.5
Best----The result should be:
cloud: 1 1 1 2 2 2
core1 :- - - - - 4 4 4 4 4 4 4
core2 :- - - - - 6 6 6 6 6 6 - 8 8 8 8
core3 :- - - - - 3 3 3 3 5 5 7 7 7 9 9 10 10
T total: 18
Energy total: 74
Best----The result should be:
cloud: 1 1 1 2 2 2 6 6 6
core1 :- - - - - 4 4 4 4 4 4 4
core2 :- - - - - - - - - - - - - 8 8 8 8
core3 :- - - - - 3 3 3 3 5 5 7 7 7 9 9 10 10
T total: 18
Energy total: 63.5
Best----The result should be:
cloud: 1 1 1 2 2 2 6 6 6 7 7 7
core1 :- - - - - 4 4 4 4 4 4 4
core2 :- - - - - - - - - - - - - 8 8 8 8
core3 :- - - - - 3 3 3 3 5 5 9 9 - - - 10 10
T total: 18
Energy total: 53
Best----The result should be:
cloud: 1 1 1 2 2 2 6 6 6 7 7 7 8 8 8
core1 :- - - - - 4 4 4 4 4 4 4
core2 :
core3 :- - - - - 3 3 3 3 5 5 9 9 - - - - 10 10
T total: 19
Energy total: 46.5
Best----The result should be:
cloud: 1 1 1 2 2 2 6 6 6 7 7 7 8 8 8
core1 :- - - - - 4 4 4 4 4 4 4 9 9 9 9 9
core2 :
core3 :- - - - - 3 3 3 3 5 5 - - - - - - 10 10
T total: 19
Energy total: 43.5
Best----The result should be:
cloud: 1 1 1 2 2 2 3 3 3 6 6 6 7 7 7 8 8 8
core1 :- - - - - 4 4 4 4 4 4 4 - - 9 9 9 9 9
core2 :
core3 :- - - - - 5 5 - - - - - - - - - - - - - 10 10
T total: 22
Energy total: 37
Best----The result should be:
cloud: 1 1 1 2 2 2 3 3 3 6 6 6 7 7 7 8 8 8 - 10 10 10
core1 :- - - - - 4 4 4 4 4 4 4 - - 9 9 9 9 9
core2 :
core3 :- - - - - 5 5
T total: 24
Energy total: 30.5
Best----The result should be:
cloud: 1 1 1 2 2 2 5 5 5 3 3 3 6 6 6 7 7 7 8 8 8 - 10 10 10
core1 :- - - - - 4 4 4 4 4 4 4 - - - - - 9 9 9 9 9
core2 :
core3 :
T total: 27
Energy total: 24
Best----The result should be:
cloud: 1 1 1 2 2 2 4 4 4 5 5 5 3 3 3 6 6 6 7 7 7 8 8 8 - 10 10 10
core1 :- - - - - - - - - - - - - - - - - - - 9 9 9 9 9
core2 :
core3 :
T total: 30
Energy total: 18.5
Best----The result should be:
cloud: 1 1 1 2 2 2 4 4 4 5 5 5 3 3 3 6 6 6 7 7 7 9 9 9 8 8 8 10 10 10
core1 :
core2 :
core3 :
T total: 32
Energy total: 15
sh: pause: command not found
Program ended with exit code: 0
```

Part 3: code

First: parameter

```
//
//  main.cpp
//  project2
//
//  Created by DANDAN LIN on 11/27/21.
//
#include<iostream>
#include<stdlib.h>
#include<vector>
#include<limits.h>
#include <stdlib.h>
#include <iomanip>
#include <algorithm>
#include <iterator>
using namespace std;
```

```
struct graph {
    int k;
    int end_time;
    char finish;
    graph *next;


};
vector<graph> ready1;
vector<graph> ready2;
vector<int> cloud;
vector<float> priority;
vector<double> Energy;
struct use_sort {
    float time;
    int num_task;
};


struct core {
    int task_num;
    int start;
    int end;
    core *next;
};
```

```
struct save_min {
    int T_min;
    int E_min;
};


struct final_result {
    vector<core> final;
    double Energy;
    int time;
};
int minimum_t;
double ori_e;
double ori_e1;
```

Main():

```
int main(){
   int i;
   int j;
   int n=3;
   int m=10;
   int e=15;
   int Ts=3;
   int Tc=1;
   int Tr=1;
   int power_rate[]={1,2,4};
   double Ps=0.5;
   graph g;
   int array[10][3]={{9,7,5},
                     {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}};
    vector<vector<int> > time_array(m);
```

```
int array2[15][2]={{1,2},
                   {1,3},
                   {1,4},
                   {1,5},
                   {1,6},
                   {2,8},
                   {2,9},
                   {3,7},
                   {4,8},
                   {4,9},
                   {5,9},
                   {6,8},
                   {7,10},
                   {8,10},
                   {9,10}};
   for (i = 0; i < m; i++){
      time_array[i].resize(n);
      priority.push_back(0);
   }
   for (i = 0; i < m; i++){
      g.k = i + 1;
      g.end_time = 0;
      g.finish = 'f';
      g.next = NULL;
      ready1.push_back(g);
      ready2.push_back(g);
      for (j = 0; j < n; j++){
         time_array[i][j]=array[i][j];//t
he graph of task
      }
   }
```

```
 vector<vector<int> >
row_graph(e);
   for (i = 0; i < e; i++){
      row_graph[i].resize(2);
      for (j = 0; j < 2; j++){
         row_graph[i][j]=array2[i][j];
      }
      graph *temp = new graph;
      graph *Node = new graph;
      graph *Node2 = new graph;
      graph *temp2 = new graph;
      Node->k = row_graph[i][0];
      Node->next = NULL;
      Node2->k = row_graph[i][1];
      Node2->next = NULL;
      if (ready1[row_graph[i][1] -
1].next == NULL){
         ready1[row_graph[i][1] -
1].next = Node;
      }
      else{
         temp =
ready1[row_graph[i][1] - 1].next;
         while (temp->next != NULL){
            temp = temp->next;
         }
         temp->next = Node;
      }
      if (ready2[row_graph[i][0] -
1].next == NULL){
         ready2[row_graph[i][0] -
1].next = Node2;
      }
```

```
            else{
        temp2 = ready2[row_graph[i][0] -
1].next;
        while (temp2->next != NULL){
          temp2 = temp2->next;
        }
        temp2->next = Node2;
      }
    }

     Energy.push_back(Ps);
    for (i = 0; i < n; i++){
      double y;
      y=power_rate[i];
      Energy.push_back(y);
    }
    int T_cloud;
    T_cloud = Ts + Tc + Tr;
    primary_assignment(time_array, m, n,
T_cloud);//phase1 primary_assignment
    Task_prioritizing(time_array, m, n,
T_cloud);//phase 2
    vector<use_sort> select_result(m);
    vector<core> core_result;
core_result = unit_selection(select_result,
 m, n, time_array, T_cloud, Ts); //phase3
minimum_t =
calculate_time(core_result, T_cloud -
Ts);
    ori_e = calculate_energy(core_result,
time_array, Ts);
    ori_e1 = ori_e;
    vector<save_min> min_g;//save and
print the first time_min schedule
    save_min temp_min;
    temp_min = print_core(core_result, n,
time_array, Ts,
calculate_time(core_result, T_cloud -
Ts), calculate_energy(core_result,
time_array, Ts));
    min_g.push_back(temp_min);
    vector<core> scheduled_result;
    scheduled_result =
save_vec(core_result,
core_result.size());
for (i = 0; i < 10; i++){
       scheduled_result =
kernel(scheduled_result,
order_priority(select_result),
time_array, T_cloud, Ts, Tc);
    }//repeat  untill no further minimized
    system("pause");
}
```

Function:primary_assignment

```
void primary_assignment(vector<vector<int> > A, int m, int n, int T_cloud){
   int i;
   int j;
   int min;
   for (i = 0; i < m; i++){
      min = INT_MAX;
      cloud.push_back(0);
      for (j = 0; j < n; j++){
         if (A[i][j] < min){
            min = A[i][j];
         }
      }
      if (min > T_cloud){
         cloud[i] = 1;
      }
   }
}// determine if the task is on the cloud or just on local
```

Northeastern UNIVERSITY

Task_prioritizing:

```
int if_zero(){
   int i;
   for (i = 0; i < priority.size(); i++){
      if (priority[i] == 0){//determine if priority is 0
         return 1;
      }
   }
   return 0;
}
void Task_prioritizing(vector<vector<int>> A, int m,
int n, int Tcloud){
   int i;
   int j;
   int sum;
   int max;
   for (i = 0; i < m; i++)
   {
      if (ready2[i].next == NULL)
      {
         sum = 0;
         if (cloud[i] == 0)
         {
            for (j = 0; j < n; j++)
            {
               sum = sum + A[i][j];
            }
            priority[i] = sum / n;//calculate w
         }
         else
         {
            priority[i] = Tcloud;
         }
         ready2[i].finish = 't';
      }
   }
   while (if_zero()){
      for (i = 0; i < m; i++){
         if (priority[i] != 0){
            continue;
         }
         sum = 0;
         max = 0;
         graph* temp = new graph;
         temp = ready2[i].next;
         while (temp != NULL){
            if (ready2[temp->k - 1].finish == 't'){
               if (max < priority[temp->k - 1])
               {
                  max = priority[temp->k - 1];
               }
               temp = temp->next;
            }
            else{
               break;
            }
         }
         if (temp == NULL && max != 0){
            if (cloud[i] == 0){
               for (j = 0; j < n; j++)
               {
                  sum = sum + A[i][j];
               }
               priority[i] = sum / n + max;
            }
            else{
               priority[i] = Tcloud + max;
            }
            ready2[i].finish = 't';
         }
      }
   }
}
```

```
int max(vector<use_sort> A){
   int i;
   int max = INT_MIN;
   for (i = 0; i < A.size(); i++){
      if (max < A[i].time){
         max = A[i].time;
      }
   }
   return max;
}


vector<use_sort> counting_sort(vector<use_sort> A,
int max){
   int i, j;
   vector<use_sort> B(A.size());
   vector<int> C(max + 2);
   for (i = 0; i < A.size(); i++){
      B[i] = A[i];
   }
   for (j = 0; j < max + 2; j++){
      C[j] = 0;
   }
   for (i = 0; i < B.size(); i++){
      C[B[i].time] = C[B[i].time] + 1;
   }
   for (j = 1; j < max + 2; j++){
      C[j] = C[j - 1] + C[j];
   }
   for (i = 0; i <= A.size() - 1; i++){
      A[C[B[i].time] - 1] = B[i];
      C[B[i].time] = C[B[i].time] - 1;
   }
   return A;
}
```

```
vector<use_sort> order_priority(vector<use_sort>
A){//order the priority vector
   int i;
   for (i = 0; i < priority.size(); i++){
      A[i].time = priority[i];
      A[i].num_task = i + 1;
   }
   A = counting_sort(A, max(A)); //sort the priority


   return A;
}
```

unit_selection

```
vector<core> unit_selection(vector<use_sort> A, int m, int n,
vector<vector<int>> time_array, int T_cloud, int T_s){
   int i;
   int j;
   int min_end=0;
   int min_start=0;
   int min_end_cloud=0;
   int index=0;
   int begin = 0;// begin time
   graph* temp = new graph;
   graph on_cloud;
   on_cloud.end_time = 0;
   on_cloud.k = 0;
   on_cloud.next = NULL;
   vector<core> core_result(n + 1);  //save tasks in each core,
which is shown as figure 3 in paper
   for (i = 0; i < n + 1; i++){
      core_result[i].start = 0;
      core_result[i].end = 0;
      core_result[i].task_num = 0;
      core_result[i].next = NULL;
   }//initial the core_result
   A = order_priority(A);
   for (i = A.size() - 1; i >= 0; i--){
      temp = ready1[A[i].num_task - 1].next;
      begin = 0;
      while (temp != NULL && ready1[temp->k - 1].finish ==
't'){
         if (begin < ready1[temp->k - 1].end_time){
            begin = ready1[temp->k - 1].end_time;
         }
         temp = temp->next;
      }
if (temp == NULL){
         min_end = INT_MAX;
         core* sud = new core;
         core* g = new core;
         if (cloud[A[i].num_task - 1] == 0){//schedule the task
on local core
            for (j = n; j >= 0; j--){
               if (j != 0){
                  if (min_end > core_result[j].end +
time_array[A[i].num_task - 1][j - 1]){
                     if (core_result[j].end < begin){
                        min_start = begin;
                        min_end = begin +
time_array[A[i].num_task - 1][j - 1];
                        index = j;
                     }
                     else{
                        min_start = core_result[j].end;
                        min_end = core_result[j].end +
time_array[A[i].num_task - 1][j - 1];
                     }
                     index = j;
                  }
               }
               else {//schedule the task on cloud
                  if (min_end > core_result[j].end + T_cloud){
                     if (core_result[j].end < begin){
                        min_start = begin;
                        min_end = begin + T_cloud;
                        min_end_cloud = begin + T_s;
                     }
                     else{
                        min_start = core_result[j].end;
                        min_end = core_result[j].end + T_cloud;
                        min_end_cloud = begin + T_s;
                     }
                     index = j;
                  }
               }
            }
         }
```

```
else{//for cloud task
            if (min_end > core_result[0].end + T_cloud){
               if (core_result[0].end < begin){
                  min_start = begin;
                  min_end = begin + T_cloud;
                  min_end_cloud = begin + T_s;
               }
               else{
                  min_start = core_result[0].end;
                  min_end = core_result[0].end + T_cloud;
                  min_end_cloud = begin + T_s;
               }
               index = 0;
            }
         }
         sud = core_result[index].next;
         g->next = NULL;
         g->start = min_start;
         if (index != 0){
            g->end = min_end;
         }
         else{
            g->end = min_end_cloud;
         }
         g->task_num = A[i].num_task;
         if (sud == NULL){
            core_result[index].next = g;
         }
         else{
            while (sud->next != NULL){
               sud = sud->next;
            }
            sud->next = g;
         }
         core_result[index].end = min_end;
         ready1[A[i].num_task - 1].finish = 't';
         ready1[A[i].num_task - 1].end_time = min_end;
      }
   }
   return core_result;
}
```

Print core:

```
save_min print_core(vector<core> A, int n,
vector<vector<int> > time_array, int T_s, int time_r,
double energy_r){
   int i;
   int j;
   int time=0;
   int time_result=0;
   double energy=0;
   save_min result;
   cout << "The result should be:" << std::endl;
   for (i = 0; i < n + 1; i++){
      time = 0;
      if (i == 0){
         cout << "cloud: ";
      }
      else{
         cout << "core" << i << " :";
      }
      core *temp = new core;
      core *pre = new core;
      temp = A[i].next;
      pre = NULL;
      while (temp != NULL){
         if (pre == NULL){
            for (j = 0; j < temp->start; j++){
               cout << "- ";
               time = time + 1;
            }
         }
         else{
            for (j = pre->end; j < temp->start; j++){
               cout << "- ";
               time = time + 1;
            }
         }
            for (j = temp->start; j < temp->end; j++){
               cout << temp->task_num << " ";
               time = time + 1;
               if (i == 0){
                  energy = energy + Energy[0];//cloud
energy
               }
               else{
                  energy = energy + Energy[i];//local energy
               }
            }
            pre = temp;
            temp = temp->next;
         }
         cout << endl;
         if (time > time_result){
            time_result = time; //running time
         }
      }
      cout << "T total: " << time_r << endl;
      cout << "Energy total: " << energy_r << endl;
      result.T_min = time_result;
      result.E_min = energy;
      return result;
}//print every core
```

Kernel:

```
vector<core> kernel(vector<core> scheduled_result,
vector<use_sort> select_result, vector<vector<int> > time_array,
int T_cloud, int T_s, int T_c){
   int i;
   int j;
   int s;
   int a;
   int b;
   int c = 0;
   int d;
   vector<core> schedule;
   vector<final_result> kernel_result;
   for (i = 0; i < scheduled_result.size(); i++){
      core* pre;
      core* temp;
      temp = scheduled_result[i].next;
      pre = NULL;
      while (temp != NULL){
         for (j = 0; j < scheduled_result.size(); j++){
            core* temp1;
            core* temp11;
            core* pre1;
            core* pre11;
            core* pre2;
            core* temp2;
            schedule = save_vec(scheduled_result,
scheduled_result.size());
            temp1 = schedule[i].next;
            temp11 = schedule[i].next;//
            pre1 = NULL;
            if (pre != NULL){
               while (temp1 != NULL && temp->task_num != temp1-
>task_num){
                  pre1 = temp1;
                  temp1 = temp1->next;   // save pre and
temp for not change
                  temp11 = temp11->next;
               }
            }
            if (temp1 != NULL){
               if (temp1->next != NULL){
                  temp1->next->end = temp1->next->end
- temp1->next->start;
                  if (pre1 != NULL){
                     temp1->next->start = pre1->end;
                  }
                  else{
                     temp1->next->start = 0;
                  }
                  temp1->next->end = temp1->next-
>start + temp1->next->end;
                  pre11 = temp11->next;
                  temp11 = temp11->next->next;
                  while (temp11 != NULL){
                     temp11->end = temp11->end - temp11-
>start;
                     temp11->start = pre11->end;// inorder
to make them one by one
                     temp11->end = temp11->start +
temp11->end;
                     pre11 = temp11;
                     temp11 = temp11->next;
                  }
               }
            }
```

```
c = 0;
            d = 0;
            if (j != i){  // not to move it to it's original sequence
               temp2 = schedule[j].next;
               pre2 = NULL;
               if (temp2 == NULL){
                  if (pre1 == NULL){
                     schedule[i].next = temp1->next;  //delete task
i in original core
                  }
                  else{
                     pre1->next = temp1->next;   //delete task i in
original core
                  }
                  temp1->next = schedule[j].next;
                  schedule[j].next = temp1;
                  temp1->end = temp1->end - temp1->start;
                  temp1->start = 0;
                  if (j != 0){
                     temp1->end = time_array[temp1->task_num -
1][j - 1] + temp1->start;
                  }
                  else{
                     temp1->end = temp1->start + T_s;
                  }
               }
               else{
                  while (temp2 != NULL){//move the task to core
j
                     a = 0;
                     b = 0;
                     if ((temp2->start >= temp1->start) && c ==
0){//original process
                                                   if (pre2 ==
NULL){
                              if (pre1 == NULL){
                                 schedule[i].next = temp1-
>next;  //delete task i in original core
                              }
                              else{
                                 pre1->next = temp1-
>next;   //delete task i in original core
                              }
if (temp2->start == temp1->start){
                                 for (s = 0; s < select_result.size();
s++){
                                    if (select_result[s].num_task
== temp1->task_num)
                                       a = s; //select_result[s].time
                                    if (select_result[s].num_task
== temp2->task_num)
                                       b = s; //select_result[s].time
                                 }
                              }
                              if (temp2->start > temp1->start){
                                 temp1->next = temp2;
                                 schedule[j].next = temp1;
                                 temp1->end = temp1->end -
temp1->start;
                                 temp1->start = 0;
                                 if (j != 0){
                                    temp1->end = temp1->start +
time_array[temp1->task_num - 1][j - 1];
                                 }
                                 else{
                                    temp1->end = temp1->start +
T_s;
                                 }

                              }
```

```
else if (temp2->start <= temp1->start){
                        temp1->next = temp2->next;
                        temp2->next = temp1;
                        temp1->start = temp2->end;
                    }
                    if (j != 0){
                        temp1->end = temp1->start
+ time_array[temp1->task_num - 1][j - 1];
                    }
                    else{
                        temp1->end = temp1->start
+ T_s;
                    }
                }
else{
                    if (pre1 == NULL){
                        schedule[i].next = temp1-
>next;
                    }
                    else{
                        pre1->next = temp1->next;
                    }
                    //if two task start at same time,
then order them
                    if (temp2->start == temp1-
>start){
                        for (s = 0; s <
select_result.size(); s++){
                            if
(select_result[s].num_task == temp1->task_num)
                                a = s;
//select_result[s].time
                            if
(select_result[s].num_task == temp2->task_num)
                                b = s;
//select_result[s].time
                        }
                    }
```

```
if (temp2->start > temp1->start){
                        temp1->next = temp2;
                        pre2->next = temp1;
                        temp1->start = pre2->end;
                    }
                    else if (temp2->start <= temp1->start){
                        temp1->next = temp2->next;
                        temp2->next = temp1;
                        temp1->start = temp2->end;
                    }
                    if (j != 0){
                        temp1->end = temp1->start +
time_array[temp1->task_num - 1][j - 1];
                    }
                    else{
                        temp1->end = temp1->start + T_s;
                    }


                }
                c = 1;
            }
            else if (temp2->next == NULL && c == 0){
                if (pre1 == NULL){
                    schedule[i].next = temp1->next;  //delete task i
in original core
                }
                else{
                    pre1->next = temp1->next;   //delete task i in
original core
                }
                temp1->next = temp2->next;
                temp2->next = temp1;
                temp1->start = temp2->end;
                if (j != 0){
                    temp1->end = temp1->start +
time_array[temp1->task_num - 1][j - 1];
                }
```

```
                    else{
                        temp1->end = temp1->start + T_s;
                    }


                    d = 1;
                    c = 1;
                }
if (c > 0){
                    if (c == 1){
                        if (d != 1 && (temp2->start > temp1-
>start)){
                            temp2->start = temp1->end;
                        }
                        if (j != 0){
                            temp2->end = temp2->start +
time_array[temp2->task_num - 1][j - 1];
                        }
                        else{
                            temp2->end = temp2->start +
T_s;
                        }
                        c = c + 1;
                    }
                    else{
                        if (temp2->start < pre2->end){
                            temp2->start = pre2->end;
                        }
                        if (j != 0){
                            temp2->end = temp2->start +
time_array[temp2->task_num - 1][j - 1];
                        }
                        else{
                            temp2->end = temp2->start +
T_s;
                        }
                    }
                }
                    pre2 = temp2;
                    temp2 = temp2->next;
                }
            }
 schedule = prepare_time_energy(schedule,
time_array, select_result, j, T_cloud - T_s, T_s, T_c);
            schedule = prepare_time_energy(schedule,
time_array, select_result, j, T_cloud - T_s, T_s, T_c);
            int time;
            time = calculate_time(schedule, T_cloud -
T_s);
            schedule = prepare_time_energy(schedule,
time_array, select_result, j, T_cloud - T_s, T_s, T_c);
            while (time != calculate_time(schedule,
T_cloud - T_s)){
                time = calculate_time(schedule, T_cloud
- T_s);
                schedule =
prepare_time_energy(schedule, time_array,
select_result, j, T_cloud - T_s, T_s, T_c);
            }
            final_result p;
            p.final = schedule;
            p.Energy = calculate_energy(schedule,
time_array, T_s);
            p.time = calculate_time(schedule, T_cloud -
T_s);
            kernel_result.push_back(p);


        }
    }
    pre = temp;
    temp = temp->next;
  }
 }
  return chose_best(kernel_result, scheduled_result,
calculate_energy(scheduled_result, time_array, T_s),
calculate_time(scheduled_result, T_cloud - T_s),
time_array, T_s, T_cloud);
}
```

Northeastern UNIVERSITY

Other function used in the main function:

```
vector<core> chose_best(vector<final_result> kernel_result, vector<core>
    scheduled_result, double ori_energy, int ori_time, vector<vector<int> > time_array,
    int T_s, int T_cloud){
    int i;
    int index=-1;
    double rate=0;
    for (i = 0; i < kernel_result.size(); i++){
        if (kernel_result[i].time <= ori_time && kernel_result[i].Energy < ori_energy){
            ori_energy = kernel_result[i].Energy;
            index = i;
        }
    }
    if (index == -1){
        for (i = 0; i < kernel_result.size(); i++){
            if (ori_energy > kernel_result[i].Energy  && kernel_result[i].time <=
                (minimum_t + 8 * minimum_t / 18) && kernel_result[i].time > ori_time){
                if (rate < (double(ori_energy - kernel_result[i].Energy) /
                    (double(kernel_result[i].time - ori_time)))){
                    rate = (double(ori_energy - kernel_result[i].Energy) /
                        (double(kernel_result[i].time - ori_time)));
                    index = i;
                }
            }

        }
    }
    if (index != -1){
        cout << "Best----";
        print_core(kernel_result[index].final, kernel_result[index].final.size()
            - 1, time_array, T_s, calculate_time(kernel_result[index].final,
            T_cloud - T_s), calculate_energy(kernel_result[index].final,
            time_array, T_s));

        return kernel_result[index].final;
    }
    else{
        cout << "no further minimized" << endl;
        return scheduled_result;
    }
}
```

```
double calculate_energy(vector<core> schedule, vector<vector<int> > time_array, int T_s){
    int i;
    double energy=0;
    for (i = 0; i < schedule.size(); i++){
        core* temp;
        temp = schedule[i].next;
        while (temp != NULL){
            if (i == 0){
                energy = energy + Energy[0] * T_s;
            }
            else{
                energy = energy + Energy[i] * time_array[temp->task_num - 1][i - 1];
            }
            temp = temp->next;
        }

    }
    return energy;
}
int calculate_time(vector<core> schedule, int T_cr)
{
    int i;
    int max=0;
    int middle;
    for (i = 0; i < schedule.size(); i++){
        core* temp = new core;
        if (schedule[i].next != NULL){
            temp = schedule[i].next;
            while (temp->next != NULL){
                temp = temp->next;
            }
            middle = temp->end;
            if (i == 0){
                middle = middle + T_cr;
            }
            if (max < middle){
                max = middle;
            }
        }
    }
     return max;
}
```