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

EECE7205- FINAL PROJECT (2)

AKANKSHA AGNIHOTRI
(NUID : 002199302)

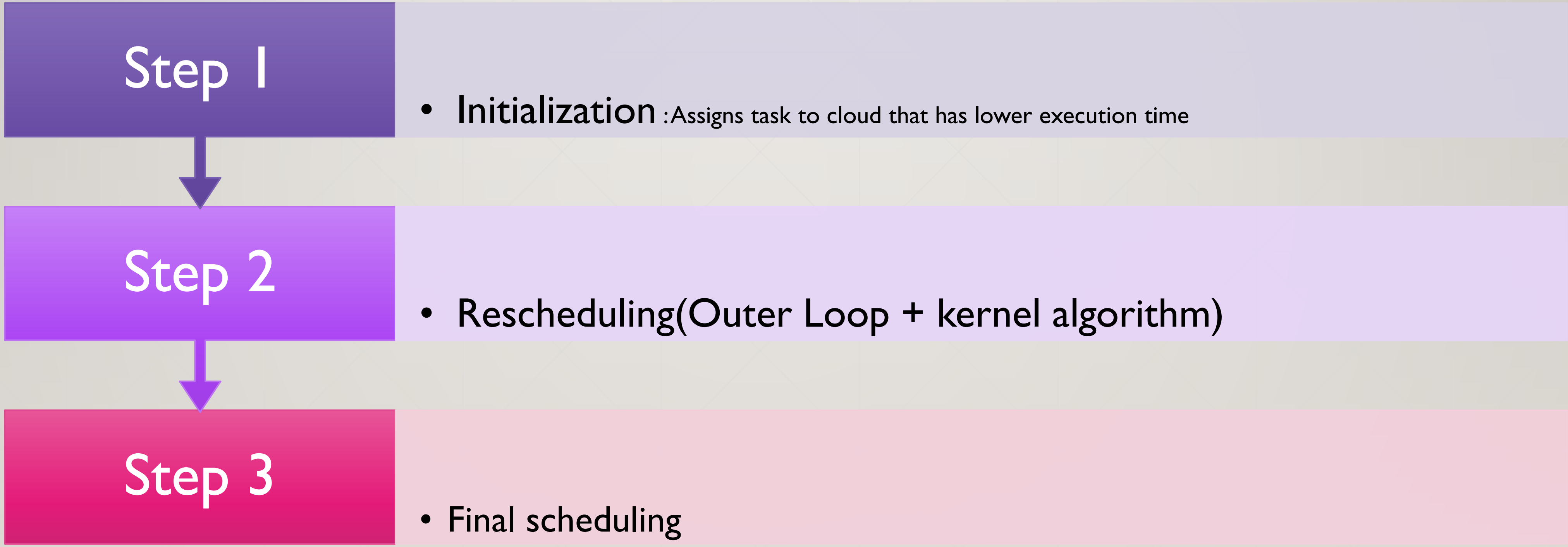
OVERVIEW OF THE PROJECT

- Abstract of the project
- MCC Task Scheduling System Model
- MCC Task Scheduling Algorithm
- Testing Samples
- Conclusion
- Code

ABSTRACT OF PROJECT

Thousands of tasks are implemented that needs to be scheduled on the cloud that consumes a lot of energy . Hence a task needs to be designed that minimizes the energy consumption of an application in a mobile device with access to the cloud under application completion time constraint .In this paper its done by minimal delay task scheduling in first step then we migrate the task towards cloud or the other local cores that reduces energy . Further to avoid high time complexity , a

MCC TASK SCHEDULING SYSTEM MODEL



Step 1

- Initialization :Assigns task to cloud that has lower execution time

Step 2

- Rescheduling(Outer Loop + kernel algorithm)

Step 3

- Final scheduling

MCC TASK SCHEDULING ALGORITHM

Step 1

Primary Assignment :

Task can be migrated from local core to another core in order to minimize the energy and time consumption

Step 2

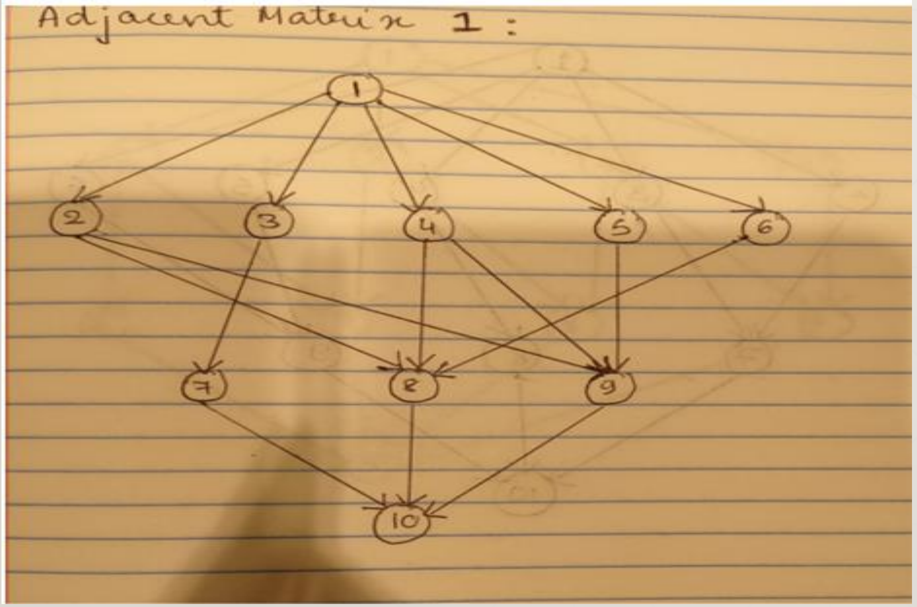
Task prioritizing : Priority is assigned by minimizing the time of completion of every task and also by taking the recursive sum of running time with maximum successor priority

Step 3

Kernel algorithm : reschedules task executions : original scheduling of the task graph is taken care in this

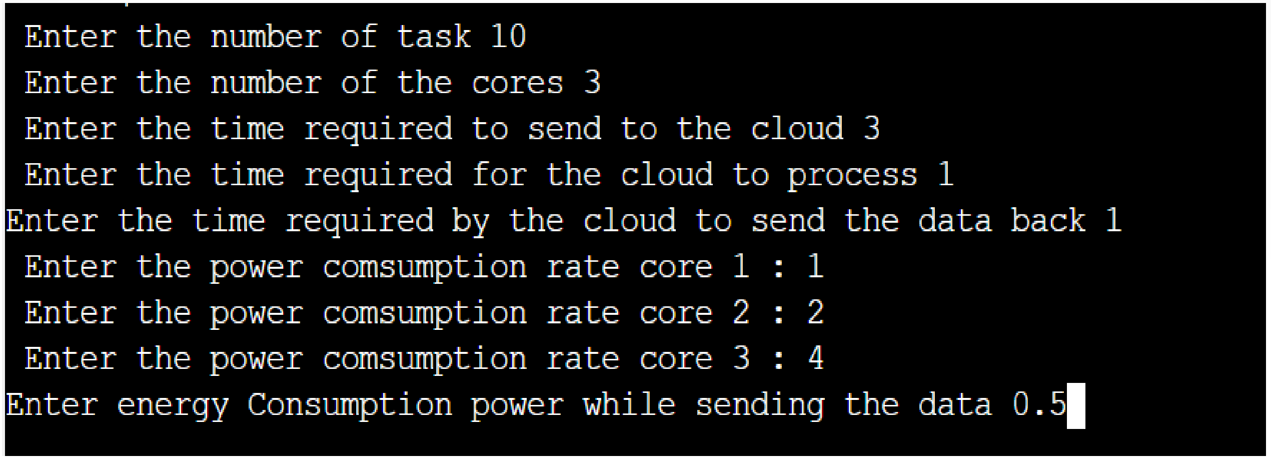
TEST SAMPLE AND RESULTS EXAMPLE 1(PAPER):

Adjacent matrix Task graph 1 :



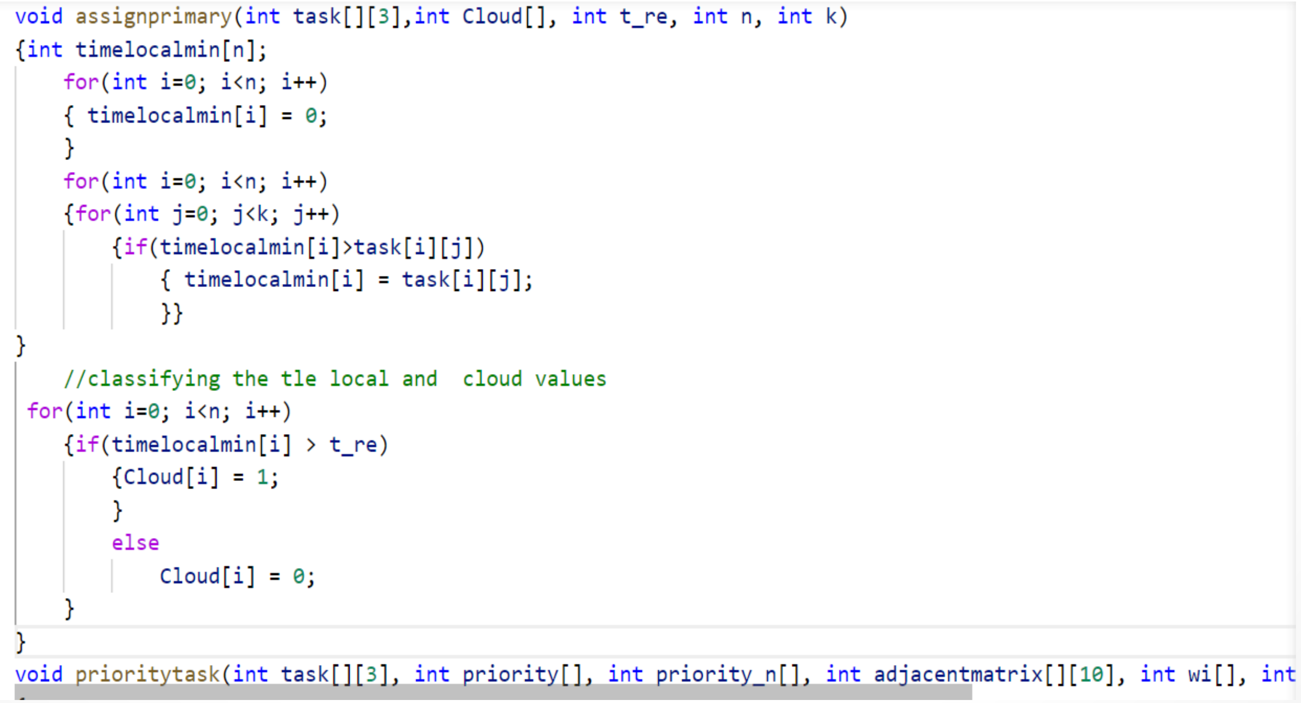
Task	Core1	Core2	Core3
1	9	7	5
2	8	6	5
3	6	5	4
4	7	5	3
5	5	4	2
6	7	6	4
7	8	5	3
8	6	4	2
9	5	3	2
10	7	4	2

General Input set :



```
 Enter the number of task 10
 Enter the number of the cores 3
 Enter the time required to send to the cloud 3
 Enter the time required for the cloud to process 1
Enter the time required by the cloud to send the data back 1
 Enter the power comsumption rate core 1 : 1
 Enter the power comsumption rate core 2 : 2
 Enter the power comsumption rate core 3 : 4
Enter energy Consumption power while sending the data 0.5
```

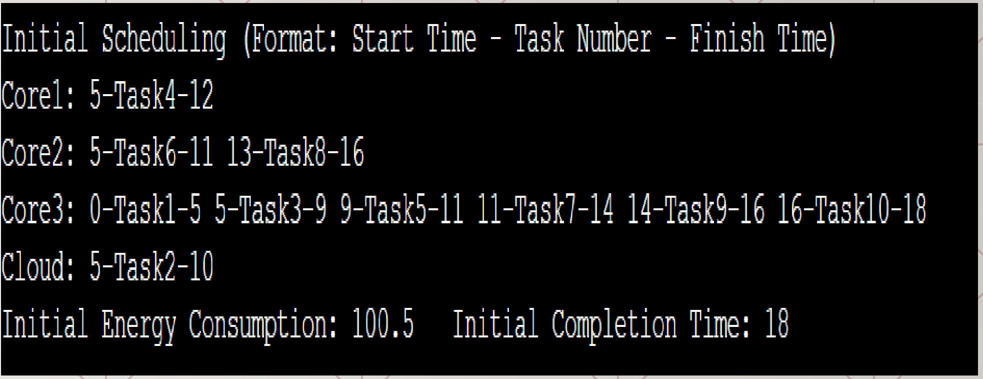
Primary assignment code :



```
void assignprimary(int task[][3],int Cloud[], int t_re, int n, int k)
{int timelocalmin[n];
    for(int i=0; i<n; i++)
    { timelocalmin[i] = 0;
    }
    for(int i=0; i<n; i++)
    {for(int j=0; j<k; j++)
        {if(timelocalmin[i]>task[i][j])
            { timelocalmin[i] = task[i][j];
            }}
}
    //classifying the tle local and  cloud values
 for(int i=0; i<n; i++)
    {if(timelocalmin[i] > t_re)
        {Cloud[i] = 1;
        }
        else
            Cloud[i] = 0;
    }
}
void prioritytask(int task[][3], int priority[], int priority_n[], int adjacentmatrix[][10], int wi[], int
```

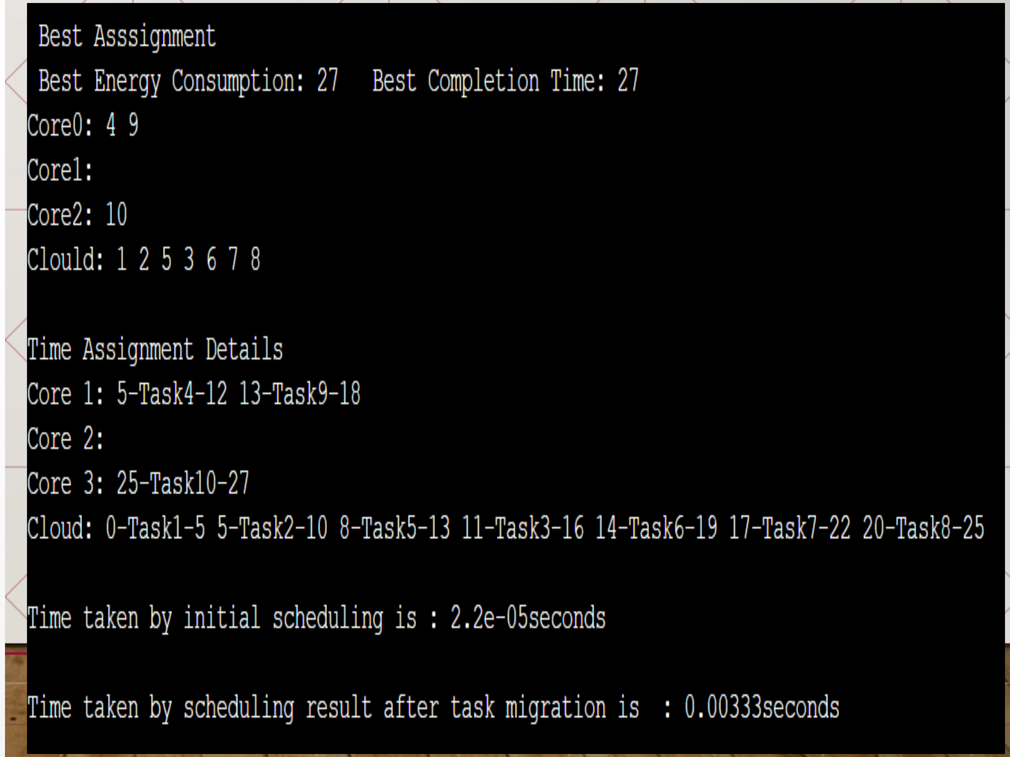
OUTPUT EXAMPLE 1:

PART 1:



```
Initial Scheduling (Format: Start Time - Task Number - Finish Time)
Core1: 5-Task4-12
Core2: 5-Task6-11 13-Task8-16
Core3: 0-Task1-5 5-Task3-9 9-Task5-11 11-Task7-14 14-Task9-16 16-Task10-18
Cloud: 5-Task2-10
Initial Energy Consumption: 100.5   Initial Completion Time: 18
```

PART 2 :



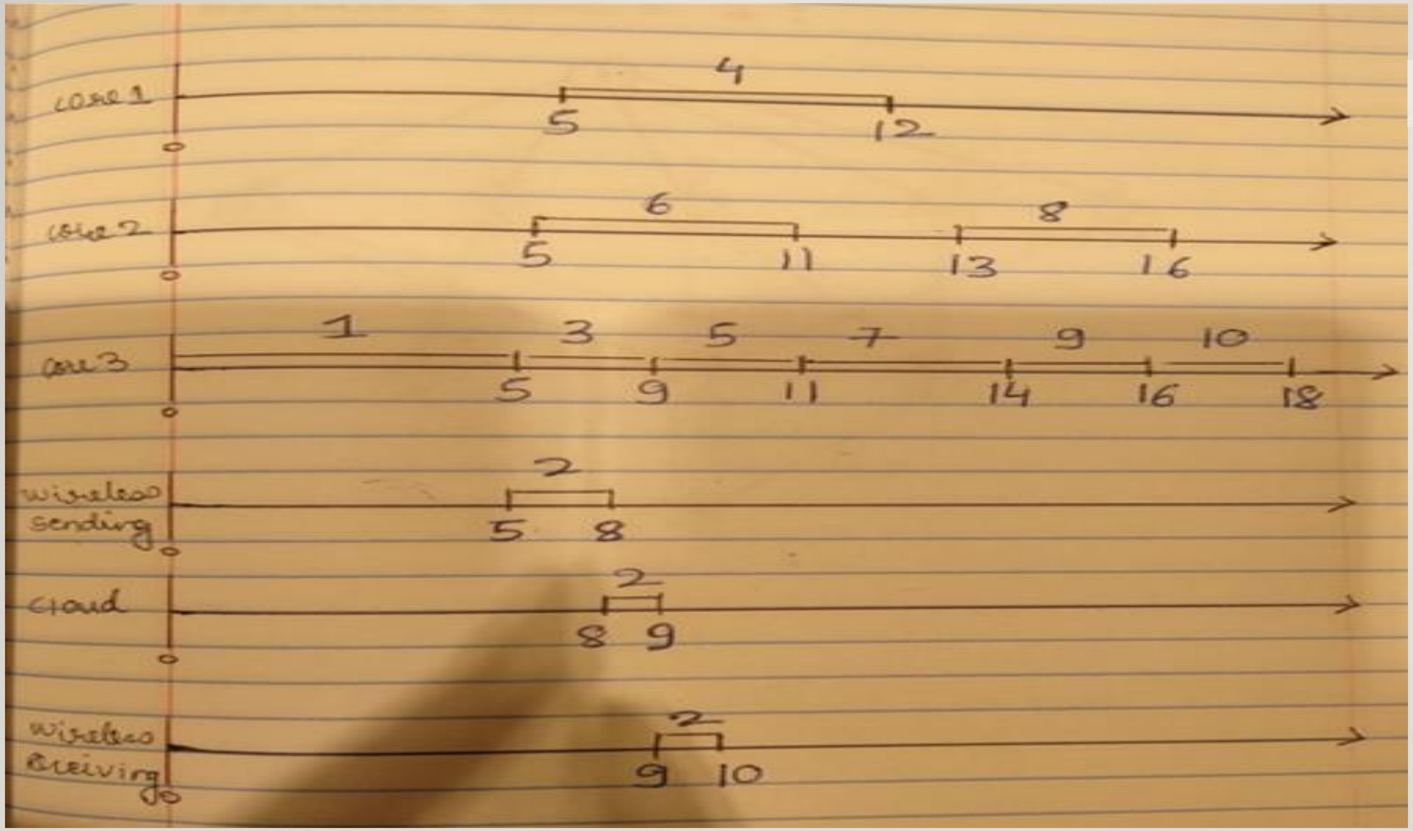
```
 Best Assignment
 Best Energy Consumption: 27   Best Completion Time: 27
Core0: 4 9
Core1:
Core2: 10
Clould: 1 2 5 3 6 7 8

Time Assignment Details
Core 1: 5-Task4-12 13-Task9-18
Core 2:
Core 3: 25-Task10-27
Cloud: 0-Task1-5 5-Task2-10 8-Task5-13 11-Task3-16 14-Task6-19 17-Task7-22 20-Task8-25

Time taken by initial scheduling is : 2.2e-05seconds

Time taken by scheduling result after task migration is  : 0.00333seconds
```

STEP 1 : INITIAL ASSIGNMENT

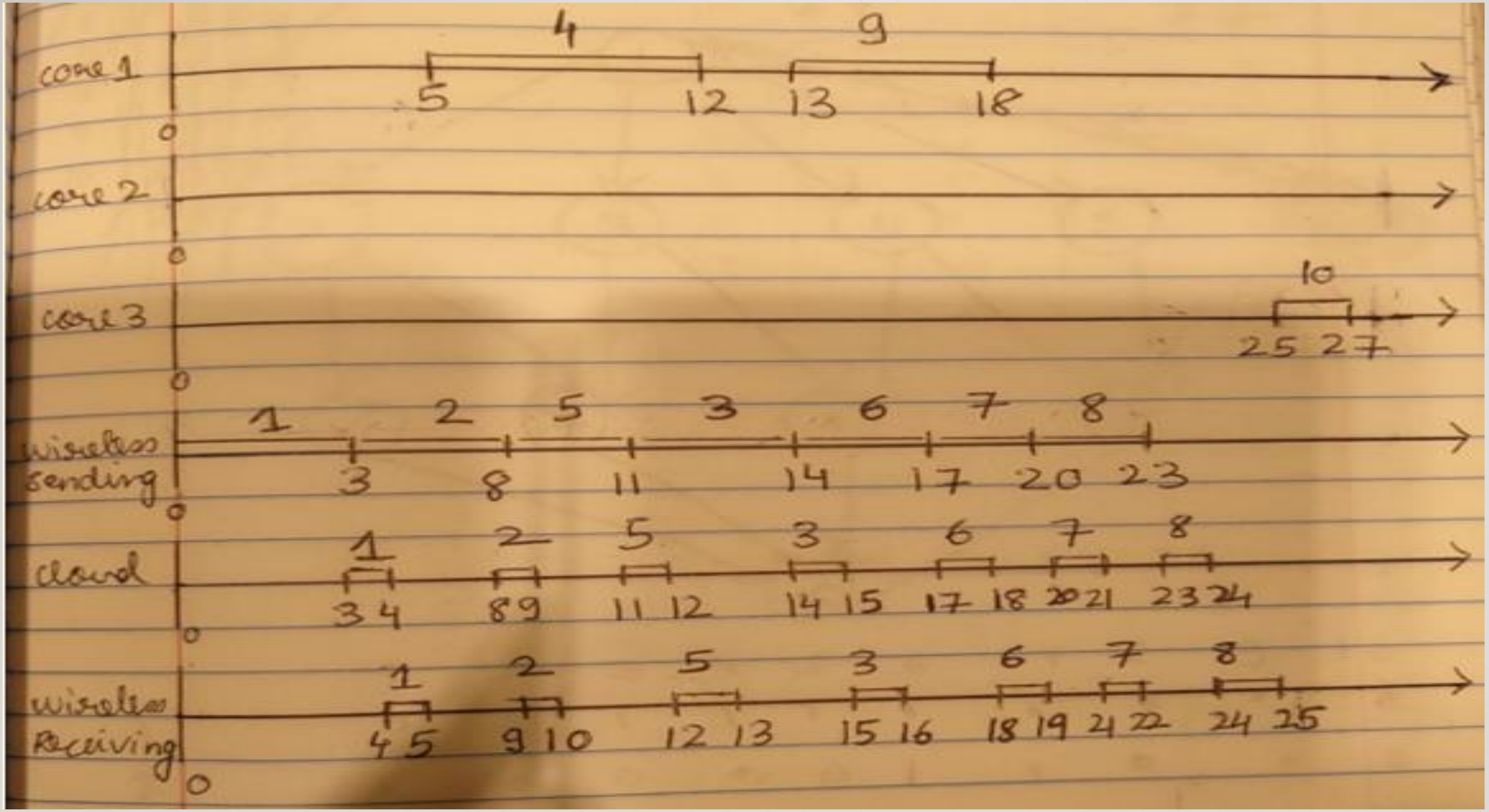


Total Energy consumed (Energy_total)= **100.5 units**

TotalExecutiontime(Time_total)=**18units**

Program running time for initial scheduling : 2.2E^(-05) sec.
Thus, the maximum time for the next step should not exceed Timemax = 1.5·Ttotal = 27 units

After running outer loop and Kernel algorithm



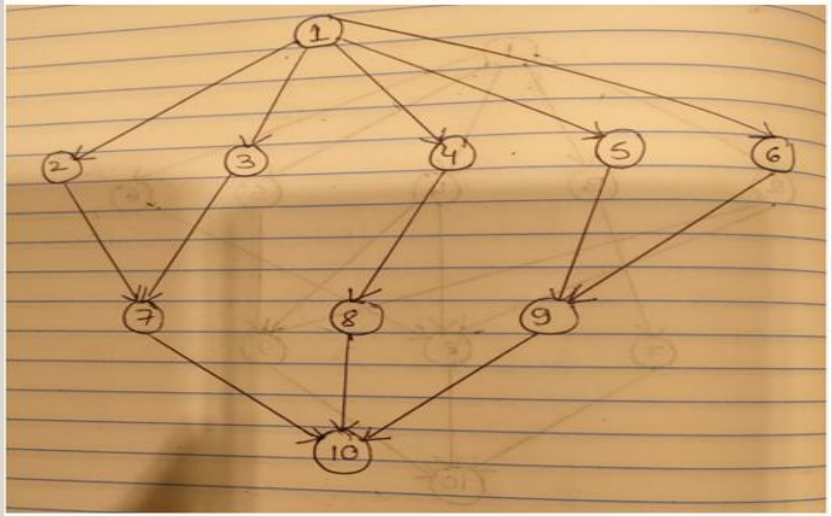
Total Energy consumed (Energy_total)= **27 units**

TotalExecutiontime(Time_total)=**27 units**

Total Program running time : 0.00333 sec.

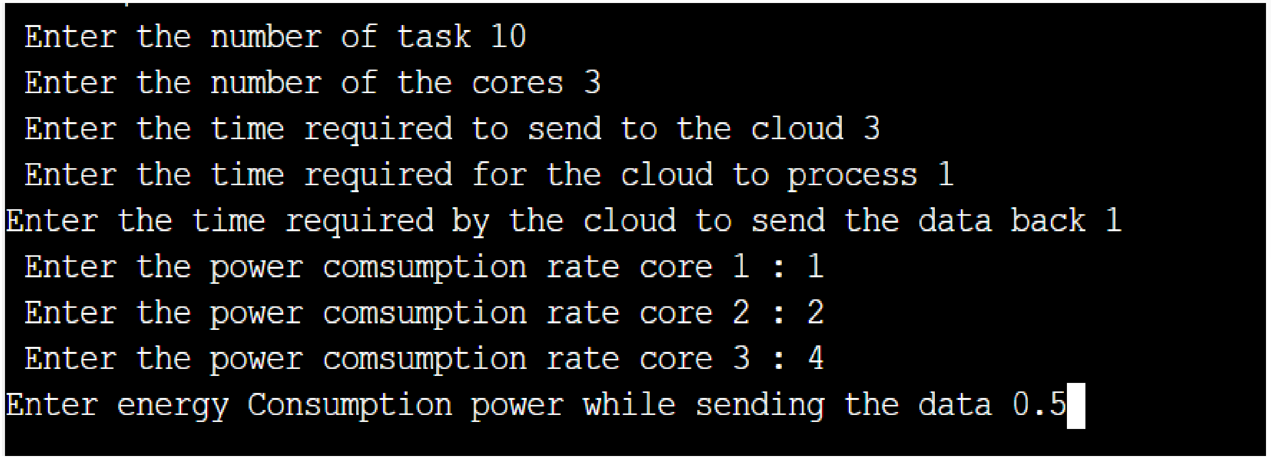
TEST RESULTS AND SAMPLES EXAMPLE 2 :

Adjacent matrix Task graph 2 :



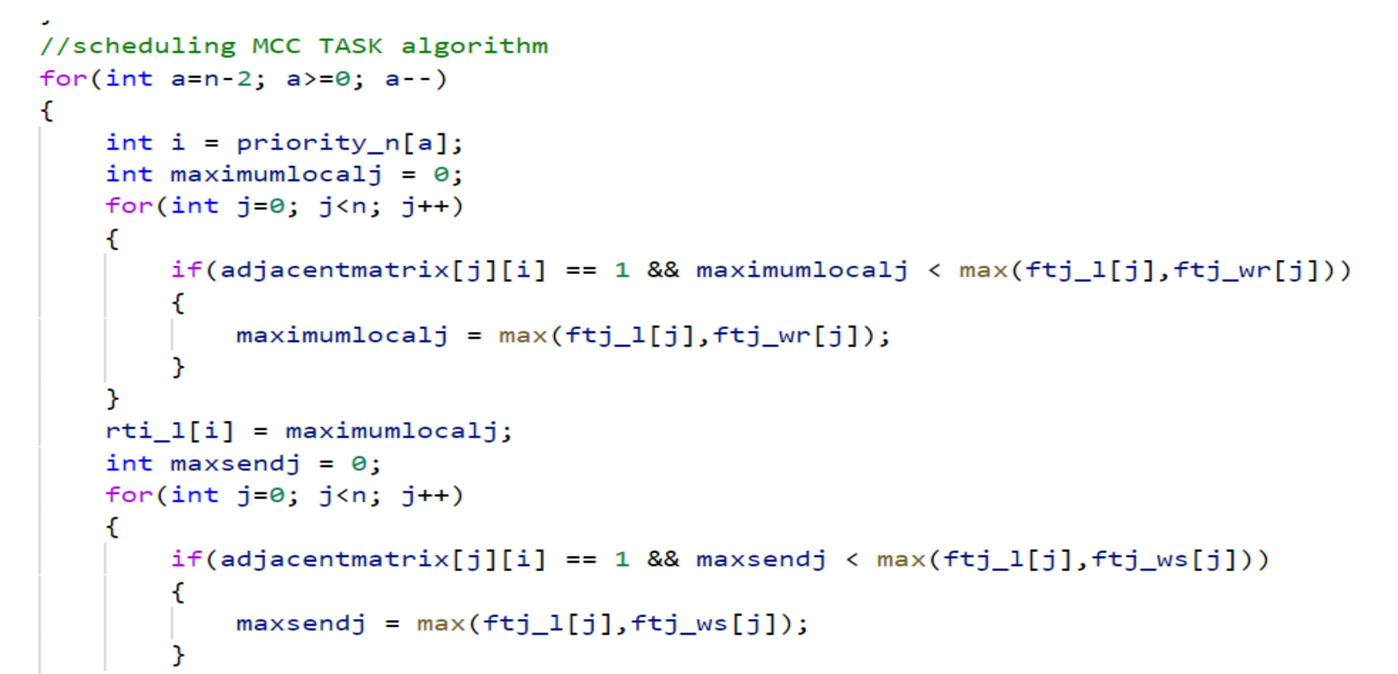
Task	Core1	Core2	Core3
1	8	6	4
2	9	7	5
3	5	4	3
4	7	5	3
5	5	4	2
6	6	5	3
7	7	5	4
8	6	4	2
9	4	3	2
10	7	6	5

General Input set :



```
 Enter the number of task 10
 Enter the number of the cores 3
 Enter the time required to send to the cloud 3
 Enter the time required for the cloud to process 1
Enter the time required by the cloud to send the data back 1
 Enter the power comsumption rate core 1 : 1
 Enter the power comsumption rate core 2 : 2
 Enter the power comsumption rate core 3 : 4
Enter energy Consumption power while sending the data 0.5
```

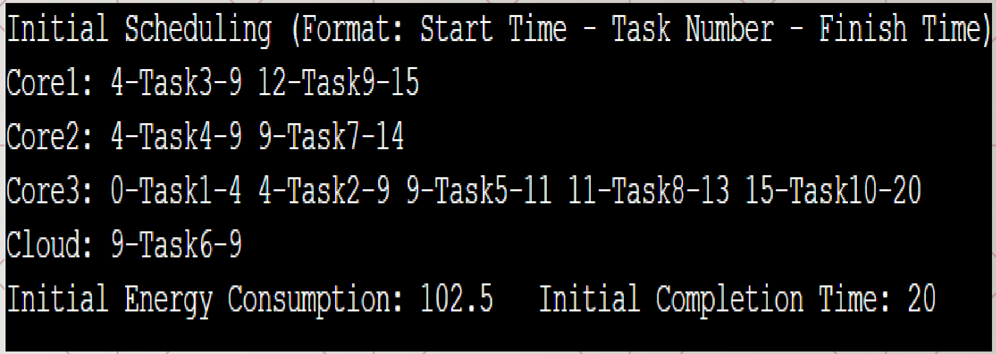
Initial code for scheduling MCC task :



```
//scheduling MCC TASK algorithm
for(int a=n-2; a>=0; a--)
{
    int i = priority_n[a];
    int maximumlocalj = 0;
    for(int j=0; j<n; j++)
    {
        if(adjacentmatrix[j][i] == 1 && maximumlocalj < max(ftj_l[j],ftj_wr[j]))
        {
            maximumlocalj = max(ftj_l[j],ftj_wr[j]);
        }
    }
    rti_l[i] = maximumlocalj;
    int maxsendj = 0;
    for(int j=0; j<n; j++)
    {
        if(adjacentmatrix[j][i] == 1 && maxsendj < max(ftj_l[j],ftj_ws[j]))
        {
            maxsendj = max(ftj_l[j],ftj_ws[j]);
        }
```

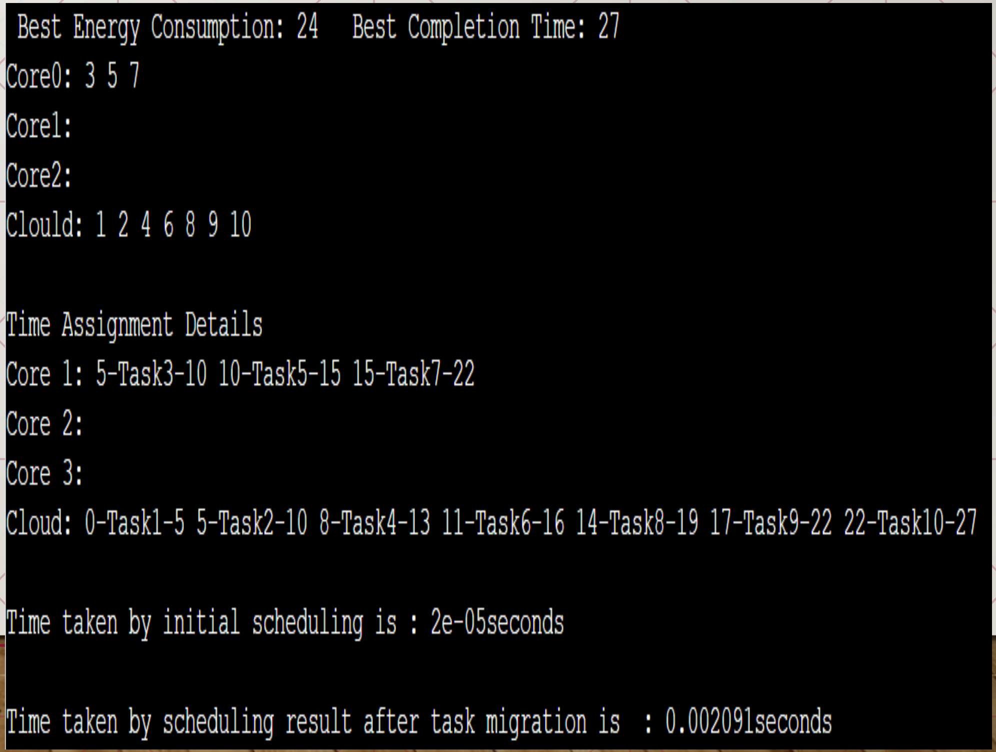
OUTPUT EXAMPLE 2 :

PART 1:



```
Initial Scheduling (Format: Start Time - Task Number - Finish Time)
Core1: 4-Task3-9 12-Task9-15
Core2: 4-Task4-9 9-Task7-14
Core3: 0-Task1-4 4-Task2-9 9-Task5-11 11-Task8-13 15-Task10-20
Cloud: 9-Task6-9
Initial Energy Consumption: 102.5   Initial Completion Time: 20
```

PART 2 :



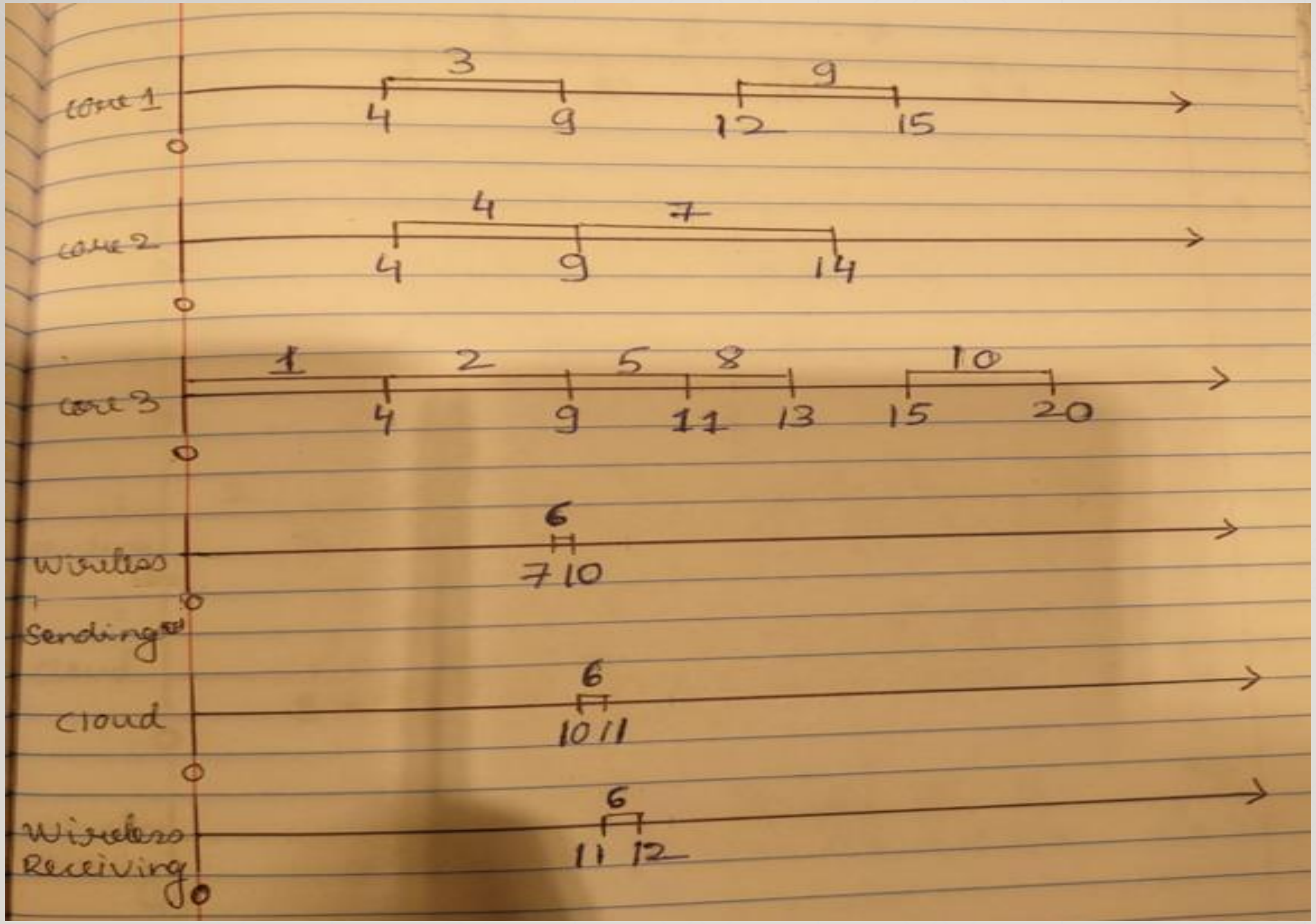
```
 Best Energy Consumption: 24   Best Completion Time: 27
Core0: 3 5 7
Core1:
Core2:
Clould: 1 2 4 6 8 9 10

Time Assignment Details
Core 1: 5-Task3-10 10-Task5-15 15-Task7-22
Core 2:
Core 3:
Cloud: 0-Task1-5 5-Task2-10 8-Task4-13 11-Task6-16 14-Task8-19 17-Task9-22 22-Task10-27

Time taken by initial scheduling is : 2e-05seconds

Time taken by scheduling result after task migration is  : 0.002091seconds
```

Initial Assignment

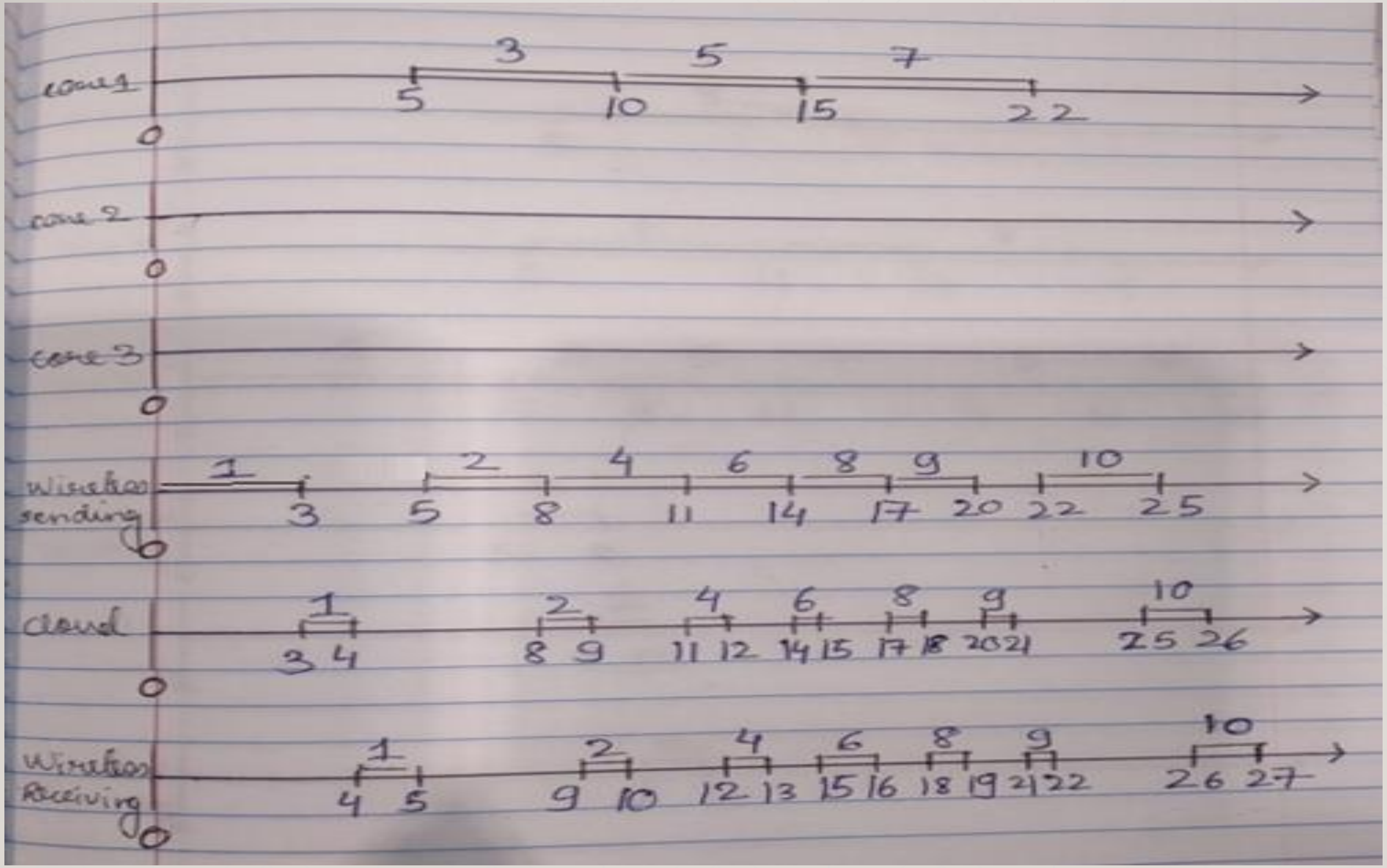


Total Energy consumed (Energy_total)= **102.5 units**

TotalExecutiontime(Time_total)=**20units**

Program running time for initial scheduling : 2E^(-05) sec.

After running outer loop and Kernel algorithm



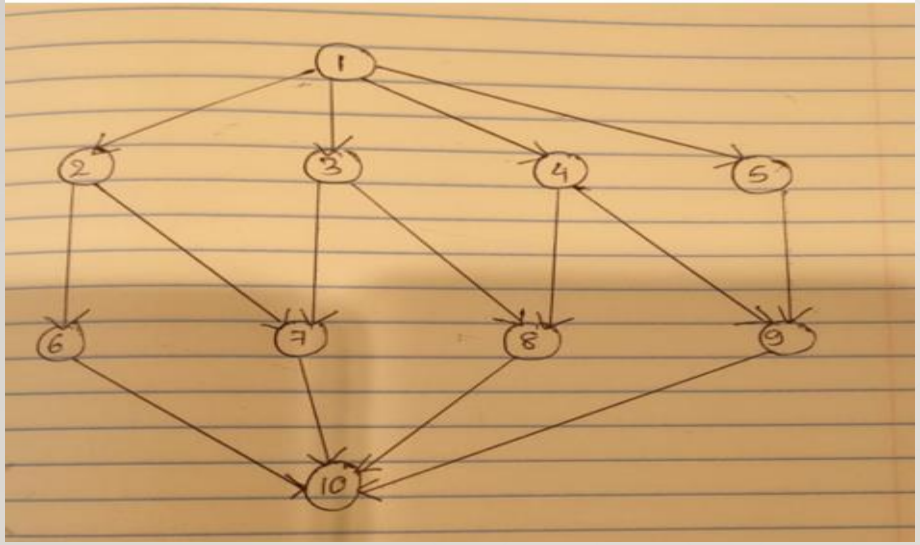
Total Energy consumed (Energy_total)= **24 units**

TotalExecutiontime(Time_total)=**27 units**

Total Program running time : 0.002091 sec.

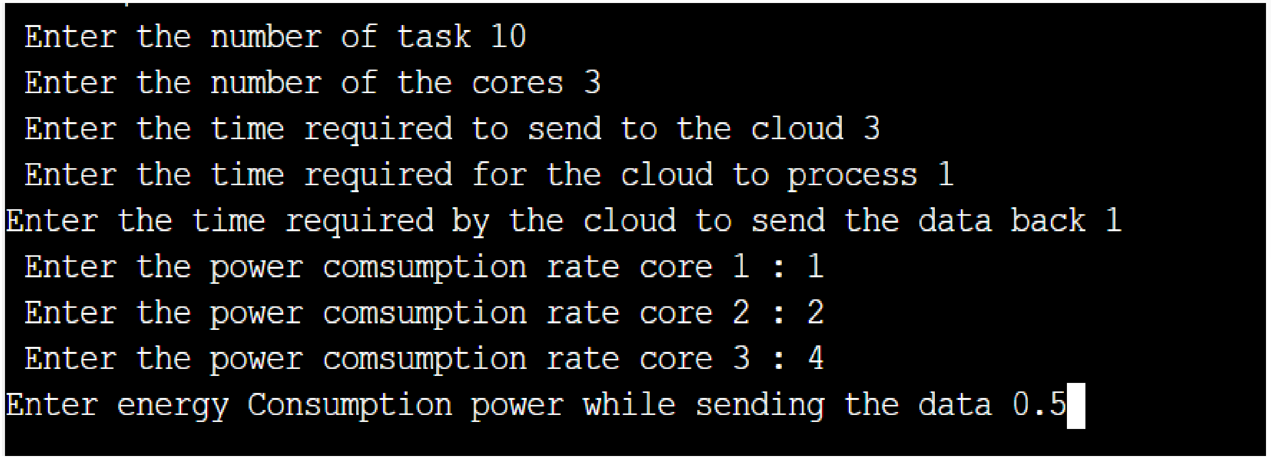
TEST RESULTS AND SAMPLES EXAMPLE 3:

Adjacent matrix Task graph 3 :



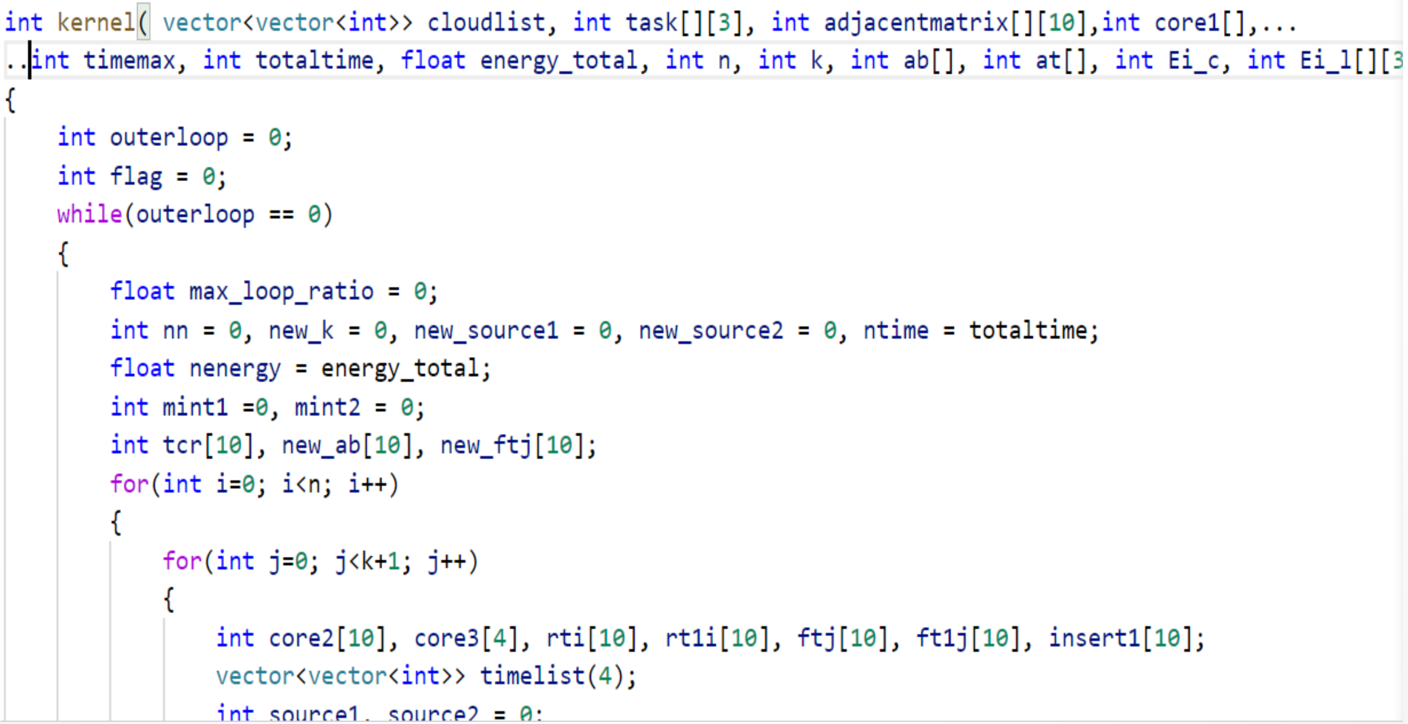
Task	Core1	Core2	Core3
1	7	5	3
2	4	3	2
3	6	5	4
4	5	4	2
5	7	6	5
6	4	3	2
7	5	4	3
8	8	5	3
9	9	7	4
10	8	6	5

General Input set :



```
 Enter the number of task 10
 Enter the number of the cores 3
 Enter the time required to send to the cloud 3
 Enter the time required for the cloud to process 1
Enter the time required by the cloud to send the data back 1
 Enter the power comsumption rate core 1 : 1
 Enter the power comsumption rate core 2 : 2
 Enter the power comsumption rate core 3 : 4
Enter energy Consumption power while sending the data 0.5
```

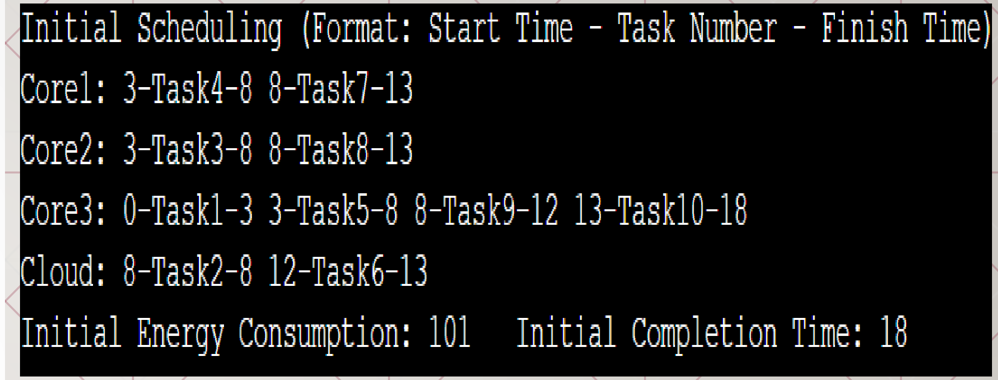
Kernel algorithm starting code :



```
int kernel( vector<vector<int>> cloudlist, int task[][3], int adjacentmatrix[][10],int core1[],...
..int timemax, int totaltime, float energy_total, int n, int k, int ab[], int at[], int Ei_c, int Ei_l[][3
{
    int outerloop = 0;
    int flag = 0;
    while(outerloop == 0)
    {
        float max_loop_ratio = 0;
        int nn = 0, new_k = 0, new_source1 = 0, new_source2 = 0, ntime = totaltime;
        float nenergy = energy_total;
        int mint1 =0, mint2 = 0;
        int tcr[10], new_ab[10], new_ftj[10];
        for(int i=0; i<n; i++)
        {
            for(int j=0; j<k+1; j++)
            {
                int core2[10], core3[4], rti[10], rt1i[10], ftj[10], ft1j[10], insert1[10];
                vector<vector<int>> timelist(4);
                int source1, source2 = 0;
```

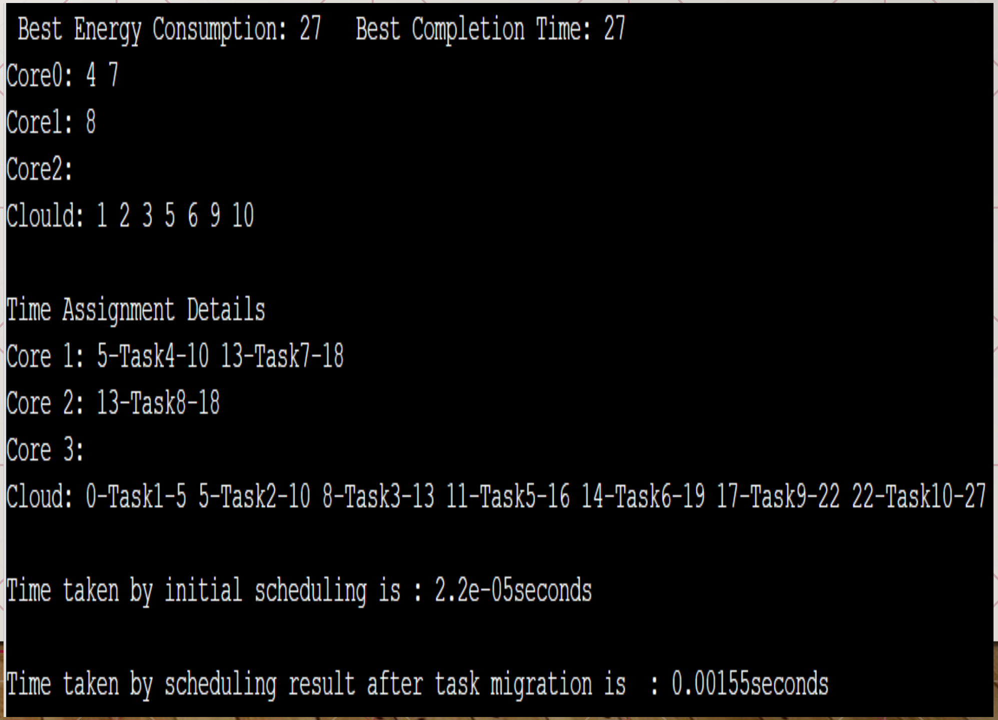
OUTPUT EXAMPLE 3:

PART 1:



```
Initial Scheduling (Format: Start Time - Task Number - Finish Time)
Core1: 3-Task4-8 8-Task7-13
Core2: 3-Task3-8 8-Task8-13
Core3: 0-Task1-3 3-Task5-8 8-Task9-12 13-Task10-18
Cloud: 8-Task2-8 12-Task6-13
Initial Energy Consumption: 101   Initial Completion Time: 18
```

PART 2 :



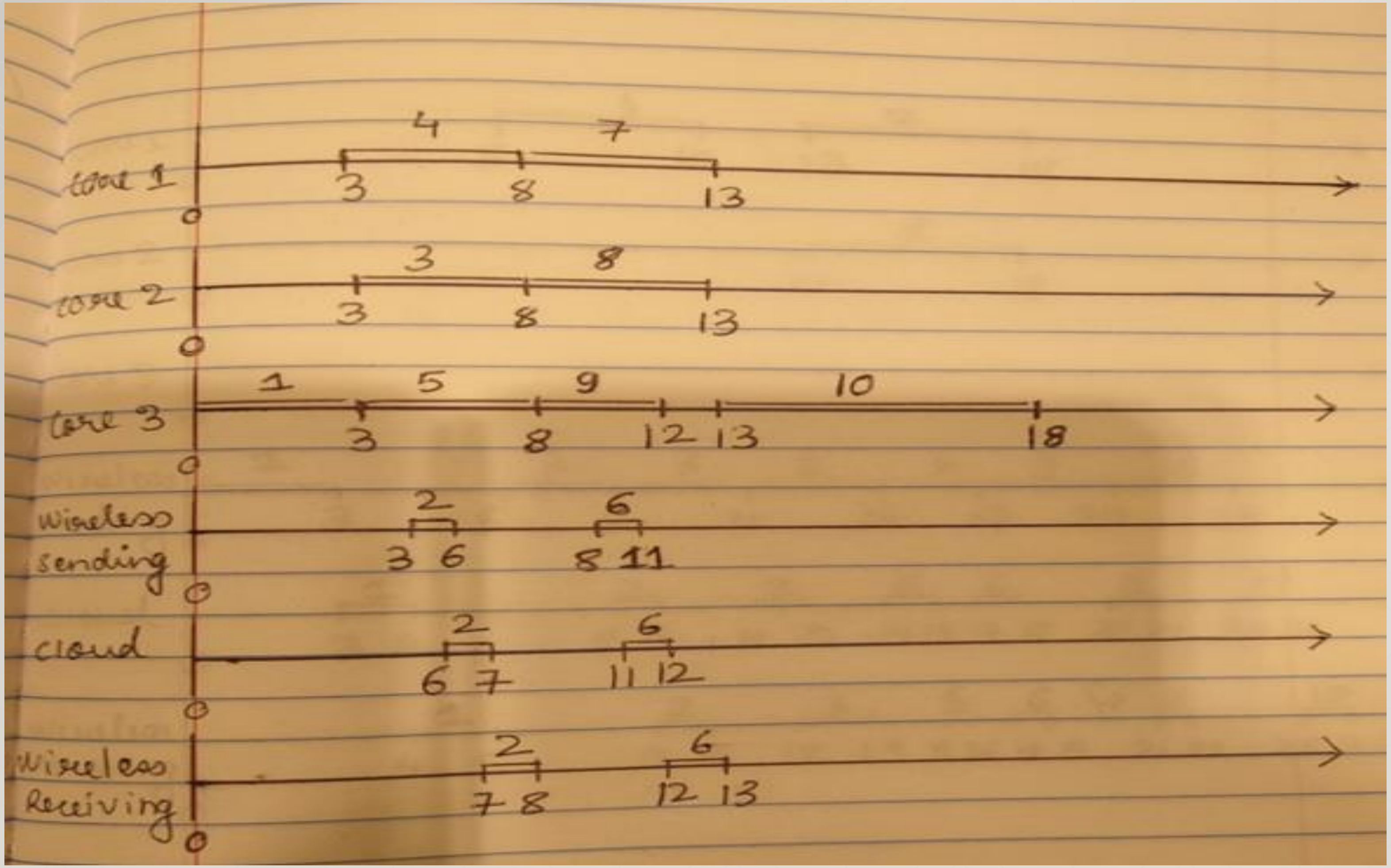
```
 Best Energy Consumption: 27   Best Completion Time: 27
Core0: 4 7
Core1: 8
Core2:
Clould: 1 2 3 5 6 9 10


Time Assignment Details
Core 1: 5-Task4-10 13-Task7-18
Core 2: 13-Task8-18
Core 3:
Cloud: 0-Task1-5 5-Task2-10 8-Task3-13 11-Task5-16 14-Task6-19 17-Task9-22 22-Task10-27


Time taken by initial scheduling is : 2.2e-05seconds


Time taken by scheduling result after task migration is  : 0.00155seconds
```

STEP 1 : INITIAL ASSIGNMENT

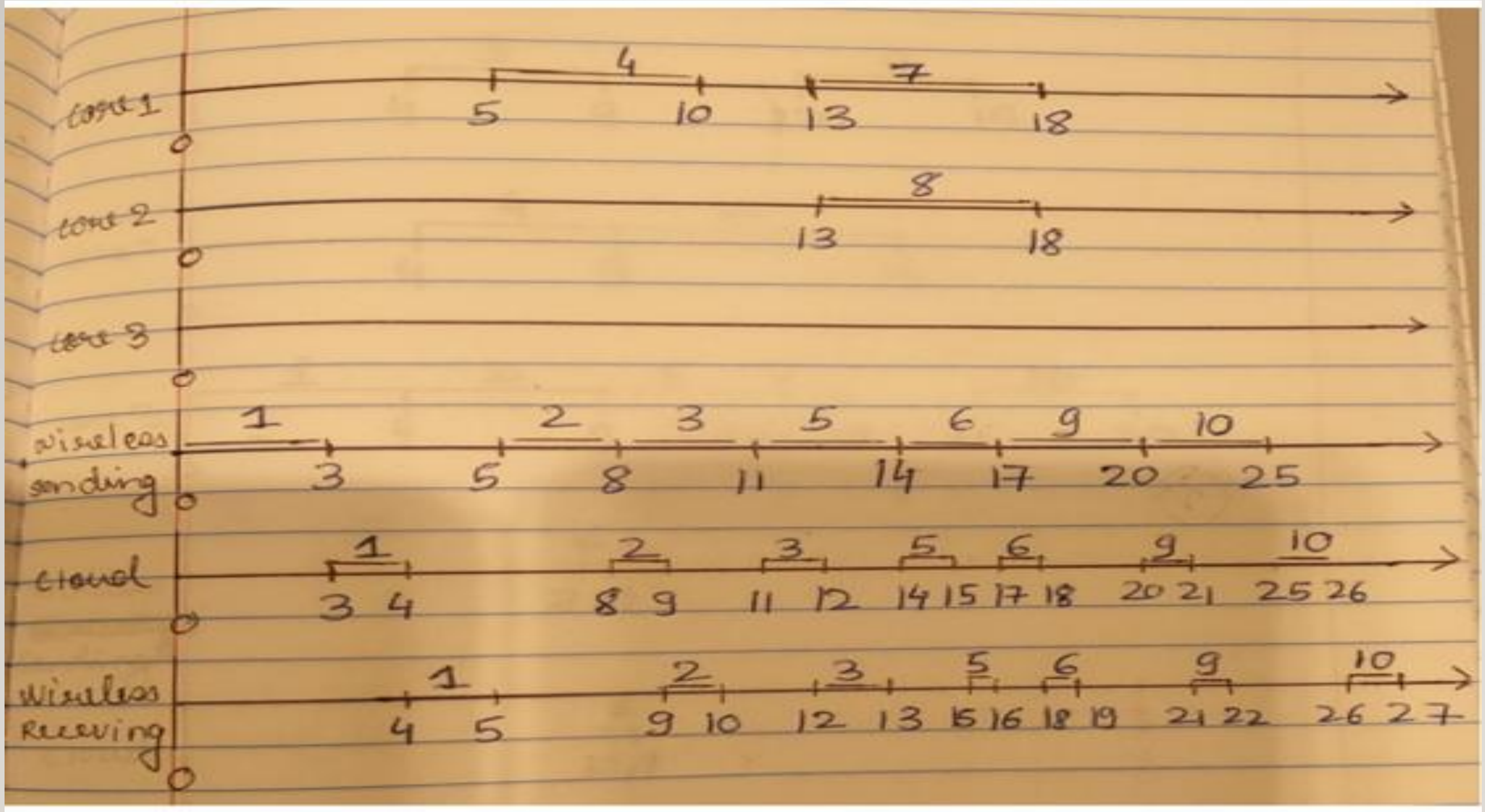


Total Energy consumed (Energy_total)= **101 units**

TotalExecutiontime(Time_total)=**18units**

Program running time for initial scheduling : 2.2E^(-0.5) sec.

After running outer loop and Kernel algorithm



Total Energy consumed (Energy_total)= **27 units**

TotalExecutiontime(Time_total)=**27 units**

Total Program running time : 0.00155 sec.

CONCLUSION :

Simulation results in significant energy reduction with overall completion time constraint satisfied. Minimal delay scheduling helps in energy reduction by migrating task from local core and cloud . Linear time rescheduling algorithm is proposed for task migration and overall computation complexity is effectively reduced .

CODE :

```
#include <iostream>
#include <algorithm>
#include <stack>
#include <vector>
#include <list>
#include <climits>
#include <ctime>
using namespace std;

void assignprimary(int task[][3],int Cloud[], int t_re, int n, int k)
{
    int timelocalmin[n];
    for(int i=0; i<n; i++)
    {
        timelocalmin[i] = 0;
    }
    for(int i=0; i<n; i++)
    {
        for(int j=0; j<k; j++)
        {
            if(timelocalmin[i]>task[i][j])
            {
                timelocalmin[i] = task[i][j];
            }
        }
    }
    //classifying the tle local and  cloud values
    for(int i=0; i<n; i++)
    {
        if(timelocalmin[i] > t_re)
        {
            Cloud[i] = 1;
        }
        else
            Cloud[i] = 0;
    }
}

void prioritytask(int task[][3], int priority[], int priority_n[], int
adjacentmatrix[][10], int wi[], int Cloud[], int t_re, int n, int k)
{
    for(int i=0; i<n; i++)
    {
        if(Cloud[i] == 1)
        {
            wi[i] = t_re;
        }
        else
        {
            int total = 0;
            for(int j=0; j<k; j++)
            {
                total += task[i][j];

            }
            wi[i] = total/k;

        }
    }

    priority[n-1] = wi[n-1];
    for(int i=n-1; i>=0; i--)
    {
        int maximumj = 0;
        for(int j=n-1; j>=0; j--)
        {
            if(adjacentmatrix[i][j] == 1 && priority[j] > maximumj)
            {
                maximumj = priority[j];
            }
        }
        priority[i] = wi[i] + maximumj;
    }
    //calculating the priority  and sorting priority
    vector<pair<int,int>> vect;
    for (int i=0; i<n; i++)
    {
        vect.push_back(make_pair(priority[i],i));
    }
    sort(vect.begin(), vect.end());
    for(int i=0; i<n; i++)
    {
        priority_n[i] = vect[i].second;
    }

}
```

```
void execution(int task[][3],int priority_n[], int adjacentmatrix[][10], int
Cloud[], int rti_l[], int rti_c[], int rti_ws[], int ftj_ws[], int ftj_wr[], int
ftj_l[], int ftj[], int core[], int core1[], int n, int k, int t_s, int t_r, int t_c,
vector<vector<int>> cloudlist)
{
    int ar = priority_n[n-1];
    rti_l[ar] = 0;
    rti_ws[ar] = 0;
    ftj_ws[ar] = rti_ws[ar] + t_s;
    rti_c[ar] = ftj_ws[ar];
    if(Cloud[ar] == 1)
    {
        ftj_wr[ar] = rti_c[ar] + t_c + t_r;
        ftj_l[ar] = 0;
        ftj[ar] = ftj_wr[ar];
        core[3] = ftj[ar];
        cloudlist[0].push_back(ar);
        core1[ar] = 3;
    }
    else
    {
        int timelocalmin = INT_MAX;
        int source;
        for(int i=0; i<k; i++)
        {
            if(task[ar][i]<timelocalmin)
            {
                timelocalmin = task[ar][i];
                source = i;
            }
        }
        ftj_l[ar] = rti_l[ar] + timelocalmin;
        ftj_wr[ar] = rti_c[ar] + t_c + t_r;
        if(ftj_l[ar] <= ftj_wr[ar])
        {
            ftj[ar] = ftj_l[ar];
            ftj_wr[ar] = 0;
            core[3] = ftj_ws[ar];
            cloudlist[source+1].push_back(ar);
            core1[ar] = source;
        }
        else
        {
            ftj[ar] = ftj_wr[ar];
            ftj_l[ar] = 0;
            core[source] = ftj[ar];
            cloudlist[0].push_back(ar);
            core1[ar] = 3;
        }
    }
    //scheduling MCC TASK algorithm
    for(int a=n-2; a>=0; a--)
    {
        int i = priority_n[a];
        int maximumlocalj = 0;
        for(int j=0; j<n; j++)
        {
            if(adjacentmatrix[j][i] == 1 && maximumlocalj < max(ftj_l[j],ftj_wr[j]))
            {
                maximumlocalj = max(ftj_l[j],ftj_wr[j]);
            }
        }
        rti_l[i] = maximumlocalj;
        int maxsendj = 0;
        for(int j=0; j<n; j++)
        {
            if(adjacentmatrix[j][i] == 1 && maxsendj < max(ftj_l[j],ftj_ws[j]))
            {
                maxsendj = max(ftj_l[j],ftj_ws[j]);
            }
        }
        rti_ws[i] = maxsendj;
        ftj_ws[i] = max(core[3],rti_ws[i]) + t_s;
        int maxcloudj = 0;
        for(int j=0; j<n; j++)
        {
            if(adjacentmatrix[j][i] == 1 && maxcloudj < ftj_wr[j]-t_r)
            {
                maxcloudj = ftj_wr[j]-t_r;
            }
        }
        rti_c[i] = max(ftj_ws[i],maxcloudj);
        if(Cloud[i] == 1)
        {
            ftj_wr[i] = rti_c[i] + t_c + t_r;
            ftj[i] = ftj_wr[i];
            ftj_l[i] = 0;
            core[3] = ftj_ws[i];
            cloudlist[0].push_back(i);
            core1[i] = 3;
        }
        else
        {
            int rti, source;
            int g = INT_MAX;
            for(int j=0; j<k; j++)
            {
                rti = max(rti_l[i],core[j]);
                if(g > rti + task[i][j])
                {
                    g = rti + task[i][j];
                    source = j;
                }
            }
            rti_l[i] = g - task[i][source];
            ftj_l[i] = g;
            ftj_wr[i] = rti_c[i] + t_c + t_r;
            if(ftj_l[i] <= ftj_wr[i])
            {
                ftj[i] = ftj_l[i];
                ftj_wr[i] = 0;
                core[source] = ftj[i];
                cloudlist[source+1].push_back(i);
                core1[i] = source;
            }
            else
            {
                ftj[i] = ftj_wr[i];
                ftj_l[i] = 0;
                core[3] = ftj[i];
                cloudlist[0].push_back(i);
                core1[i] = 3;
            }
        }

    }
}

int kernel( vector<vector<int>> cloudlist, int task[][3], int
adjacentmatrix[][10],int core1[], int timemax, int totaltime, float energy_total, int
n, int k, int ab[], int at[], int Ei_c, int Ei_l[][3])
{
    int outerloop = 0;
    int flag = 0;
    while(outerloop == 0)
    {
        float max_loop_ratio = 0;
        int nn = 0, new_k = 0, new_source1 = 0, new_source2 = 0, ntime = totaltime;
        float nenergy = energy_total;
        int mint1 =0, mint2 = 0;
        int tcr[10], new_ab[10], new_ftj[10];
        for(int i=0; i<n; i++)
        {
            for(int j=0; j<k+1; j++)
            {
                int core2[10], core3[4], rti[10], rt1i[10], ftj[10], ft1j[10],
insert1[10];
                vector<vector<int>> timelist(4);
                int source1, source2 = 0;
                for(int i=0; i<10; i++)
                {
                    rti[i] = 0;
                    ftj[i] = 0;
                    core2[i] = core1[i];
                    ft1j[i] = at[i];
                    rt1i[i] = ab[i];
                    insert1[i] = 0;
                }
                for(int a=0; a<cloudlist.size(); a++)
                {

                    core3[a] = 0;
                    for(int b=0; b<cloudlist[a].size(); b++)
                    {
                        timelist[a].push_back(cloudlist[a][b]);
                    }
                }
                int presentcore = core1[i];
                for(int a=0; a<timelist[presentcore].size(); a++)
                {
                    if(timelist[presentcore][a] == i)
                    {
                        source1 = a;
                    }
                }
                timelist[presentcore].erase(timelist[presentcore].begin()+source1
);

                // ready time of target task
                if(j == 3)
                {
```

```
    int maxsendj = 0;
    for(int a=0; a<n; a++)
    {
        if(adjacentmatrix[a][i] == 1 && maxsendj < ft1j[a])
        {
            maxsendj = ft1j[a];
        }
    }
    rti[i] = maxsendj;
}
else
{
    int maximumlocalj = 0;
    for(int a=0; a<n; a++)
    {
        if(adjacentmatrix[a][i] == 1 && maximumlocalj < ft1j[a])
        {
            maximumlocalj = ft1j[a];
        }
    }
    rti[i] = maximumlocalj;
}
core2[i] = j;
if(timelist[j].size() == 0)
{
    source2 = 0;
}
else if(timelist[j].size() == 1)
{
    if(rt1i[timelist[j][0]] > rti[i])
    {
        source2 = 0;
    }
    else
    {
        source2 = 1;
    }
}
else
{
    if(rt1i[timelist[j][0]] > rti[i])
    {
        source2 = 0;
    }
```

```
                    else if(rt1i[timelist[j][timelist[j].size()-1]] <= rti[i])
                    {
                        source2 = timelist[j].size();
                    }
                    else
                    {
                        for(int b=0; b<timelist[j].size()-1; b++)
                        {
                            if(rti[i]>=rt1i[timelist[j][b]] &&
rti[i]<=rt1i[timelist[j][b+1]])
                            {
                                source2 = b+1;
                            }
                        }
                    }
                }
                timelist[j].insert(timelist[j].begin()+source2,i);
                // initialize ready1 and ready2
                int ready1[10], ready2[10];
                for(int a=0; a<10; a++)
                {
                    ready1[a] = 0;
                }
                for(int a=0; a<10; a++)
                {
                    for(int b=0; b<10; b++)
                    {
                        if(adjacentmatrix[a][b] == 1)
                        {
                            ready1[b] += 1;
                        }
                    }
                    ready2[a] = 1;
                }
                for(int a=0; a<4; a++)
                {
                    if(timelist[a].size()>0)
                    {
                        ready2[timelist[a][0]] = 0;
                    }
                }
                //defining the stack
                stack<int> sknew;
                for(int a=0; a<10; a++)
                {
                    if(ready1[a] == 0 && ready2[a] == 0 && insert1[a] == 0)
                    {
                        sknew.push(a);
                        insert1[a] = 1;

                    }
                }
                int present1 = sknew.top();
                sknew.pop();
                rti[present1] = 0;
                if(core2[present1] == 3)
                {
                    rti[present1] = max(core3[core2[present1]],rti[present1]);
                    ftj[present1] = rti[present1] + 5;
                    core3[core2[present1]] = rti[present1] + 3;
                }
                else
                {
                    rti[present1] = max(core3[core2[present1]],rti[present1]);
                    ftj[present1] = rti[present1] +
task[present1][core2[present1]];
                    core3[core2[present1]] = ftj[present1];
                }

                for(int a=0; a<10; a++)
                {
                    if(adjacentmatrix[present1][a] == 1)
                    {
                        ready1[a] -= 1;
                    }
                }
                ready2[present1] = 1;
                if(timelist[core2[present1]].size()>1)
                {
                    for(int a=1; a<timelist[core2[present1]].size(); a++)
                    {
                        if(timelist[core2[present1]][a-1] == present1)
                        {
                            ready2[timelist[core2[present1]][a]] = 0;
                        }
                    }
                }

                for(int a=0; a<10; a++)
                {
                    if(ready1[a] == 0 && ready2[a] == 0 && insert1[a] == 0)
                    {
                        sknew.push(a);
                        insert1[a] = 1;

                    }
                }
                while(sknew.size() != 0)
                {
                    int present = sknew.top();
                    sknew.pop();
                    //ready time of current task
                    if(core2[present] == 3)
                    {
                        int maxsendj1 = 0;
                        for(int a=0; a<n; a++)
                        {
                            if(adjacentmatrix[a][present] == 1 && maxsendj1 <
ftj[a])

                            {
                                maxsendj1 = ftj[a];
                            }
                        }
                        rti[present] = maxsendj1;
                    }
                    else
                    {
                        int maximumlocalj1 = 0;
                        for(int a=0; a<n; a++)
                        {
                            if(adjacentmatrix[a][present] == 1 && maximumlocalj1 <
ftj[a])

                            {
                                maximumlocalj1 = ftj[a];
                            }
                        }
                        rti[present] = maximumlocalj1;
                    }
                    if(core2[present] == 3)
                    {
                        rti[present] = max(core3[core2[present]],rti[present]);
                        ftj[present] = rti[present] + 5;
                        core3[core2[present]] = rti[present] + 3;
                    }
                    else
                    {
                        rti[present] = max(core3[core2[present]],rti[present]);
                        ftj[present] = rti[present] +
task[present][core2[present]];
                        core3[core2[present]] = ftj[present];
                    }
                    //updating ready1 and ready2
                    for(int a=0; a<10; a++)
                    {
                        if(adjacentmatrix[present][a] == 1)
                        {
                            ready1[a] -= 1;
                        }
                    }
                    ready2[present] = 1;
                    if(timelist[core2[present]].size()>1)
                    {
                        for(int a=1; a<timelist[core2[present]].size(); a++)
                        {
                            if(timelist[core2[present]][a-1] == present)
                            {
                                ready2[timelist[core2[present]][a]] = 0;
                            }
                        }
                    }
                    for(int a=0; a<10; a++)
                    {
                        if(ready1[a] == 0 && ready2[a] == 0 && insert1[a] == 0)
                        {
                            sknew.push(a);
                            insert1[a] = 1;

                        }
                    }

                }

                int presenttime = ftj[n-1];
                int presentenergy = 0;
                for(int a=0; a<10; a++)
                {
                    if(core2[a] == 3)
                    {
                        presentenergy += Ei_c;
                    }
                    else
                    {
                        presentenergy += Ei_l[a][core2[a]];
                    }
                }

                if(presenttime <= totaltime && presentenergy < nenergy)
                {
                    mint1 = 1;
                    nn = i;
                    new_k = j;
                    new_source1 = source1;
                    new_source2 = source2;
                    ntime = presenttime;
                    nenergy = presentenergy;

                    for(int a=0; a<10; a++)
                    {
                        tcr[a] = core2[a];
                        new_ab[a] = rti[a];
                        new_ftj[a] = ftj[a];
                     }
                }
                if(presenttime > totaltime && presenttime <= timemax && mint1 == 0
&& presentenergy < energy_total && max_loop_ratio< double((energy_total -
presentenergy) / (presenttime - totaltime)))
                {
                    max_loop_ratio = double((energy_total - presentenergy) /
(presenttime - totaltime));
                    mint2 = 1;
                    nn = i;
                    new_k = j;
                    new_source1 = source1;
                    new_source2 = source2;
                    ntime = presenttime;
                    nenergy = presentenergy;
                    for(int a=0; a<10; a++)
                    {
                        tcr[a] = core2[a];
                        new_ab[a] = rti[a];
                        new_ftj[a] = ftj[a];
                    }
                }
            }
        }
        if(mint1 != 1 && mint2 != 1)
        {
            outerloop = 1;
        }
        else
        {
            cloudlist[core1[nn]].erase(cloudlist[core1[nn]].begin()+new_source1);
            cloudlist[new_k].insert(cloudlist[new_k].begin()+new_source2,nn);
            totaltime = ntime;
            energy_total = nenergy;
            for(int a=0; a<10; a++)
            {
                core1[a] = tcr[a];
                ab[a] = new_ab[a];
                at[a] = new_ftj[a];
            }
            if(mint1 != 1 && mint2 != 1)
            {
                outerloop = 1;
            }
            flag += 1;
            cout<<flag<<"th step "<<endl;
            cout<<" Current Operation: Insert Task "<<nn+1<<" to Core 
"<<new_k+1<<endl;
            cout<<" Current Completion Time: "<<totaltime<< "   Current Energy 
Consumption: "<<energy_total<<endl;
        }
    }
    cout<<endl;
    cout<<" Best Asssignment "<<endl;
    cout<<" Best Energy Consumption: "<<energy_total<<"   Best Completion Time: 
"<<totaltime<<endl;
    for(int i=0; i<cloudlist.size(); i++)
    {
        if(i == 3)
        {
            cout<<"Clould: ";
        }
        else{
            cout<<"Core"<<i<<": ";
        }
        for(int j=0; j<cloudlist[i].size(); j++)
        {
            cout<<cloudlist[i][j]+1<<" ";
        }
        cout<<endl;
    }
    cout<<endl;
    cout<<"Time Assignment Details"<<endl;
    for(int i=0; i<cloudlist.size(); i++)
    {
        if(i == 3)
        {
            cout<<"Cloud: ";
        }
        else{
            cout<<"Core "<<i+1<<": ";
        }
        for(int j=0; j<cloudlist[i].size(); j++)
        {
            cout<<ab[cloudlist[i][j]]<<"-Task"<<cloudlist[i][j]+1<<"-"<<at[cloudl
ist[i][j]]<<" ";
        }
        cout<<endl;
    }
}

int main()
{
   int task[][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}};
    /*int task[][3]={{8,6,4},
                     {9,7,5},
                     {5,4,3},
                     {7,5,3},
                     {5,4,2},
                     {6,5,3},
                     {7,5,4},
                     {6,4,2},
                     {4,3,2},
                     {7,6,5}};*/
    /*int task[][3]={{7,5,3},
                     {4,3,2},
                     {6,5,4},
                     {5,4,2},
                     {7,6,5},
                     {4,3,2},
                     {5,4,3},
                     {8,5,3},
                     {9,7,4},
                     {8,6,5}};*/

    int adjacentmatrix[][10]={{0,1,1,1,1,1,0,0,0,0},
                    {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 adjacentmatrix[][10]={{0,1,1,1,1,1,0,0,0,0},
                    {0,0,0,0,0,0,1,0,0,0},
                    {0,0,0,0,0,0,1,0,0,0},
                    {0,0,0,0,0,0,0,1,0,0},
                    {0,0,0,0,0,0,0,0,1,0},
                    {0,0,0,0,0,0,0,0,1,0},
                    {0,0,0,0,0,0,0,0,0,1},
                    {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 adjacentmatrix[][10]={{0,1,1,1,1,0,0,0,0,0},
                              {0,0,0,0,0,1,1,0,0,0},
                              {0,0,0,0,0,0,1,1,0,0},
                              {0,0,0,0,0,0,0,1,1,0},
                              {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 Cloud[10];
    int priority[10], priority_n[10], wi[10], core[4], core1[10];
    int rti_l[10], rti_c[10], rti_ws[10], ftj_ws[10], ftj_wr[10], ftj_l[10],
ftj[10];
    vector<vector<int>> cloudlist(4);
```

```
for(int i=0; i<10; i++)
{
    Cloud[i] = 0;
    priority[i] = 0;
    priority_n[i] = 0;
    wi[i] = 0;
    rti_l[i] = 0;
    rti_ws[i] = 0;
    rti_c[i] = 0;
    ftj_ws[i] = 0;
    ftj_wr[i] = 0;
    ftj_l[i] = 0;
    ftj[i] = 0;
    core1[i] = 0;
}
for(int i=0; i<4; i++)
{
    core[i] = 0;
}
int k = 3, n = 10;
int t_s = 3, t_c = 1, t_r = 1;
int k1,n1;
int t_s1,t_r1,t_c1;
cout<<" Enter the number of task ";
cin>>n1;
cout<<" Enter the number of the cores ";
cin>>k1;
cout<<" Enter the time required to send to the cloud ";
cin>>t_s1;
cout<<" Enter the time required for the cloud to process ";
cin>>t_c1;
cout<<"Enter the time required by the cloud to send the data back ";
cin>>t_r1;
int t_re = t_s + t_c + t_r;
int Ei_l[10][3];
int pc[] = {1,2,4};
float pcs = 0.5;
int pc1[100];
for(int z=1 ; z<=k ;z++)
{
    cout<<" Enter the power comsumption rate core "<<z<<" : ";
    cin>>pc1[z];
}
float pcs1;
```

```
  cout<<"Enter energy Consumption power while sending the data ";
  cin>>pcs1;
  float Ei_c = pcs * t_s;
  for(int i=0; i<10; i++)
  {
   for(int j=0; j<3; j++)
   {
     Ei_l[i][j] = pc[j] * task[i][j];
   }
  }
  clock_t time_req1;
  clock_t time_req2;
  time_req1 = clock();
  time_req2 = clock();
  assignprimary(task,Cloud,t_re,n,k);
  prioritytask(task,priority,priority_n,adjacentmatrix,wi,Cloud,t_re,n,k);
  execution(task,priority_n,adjacentmatrix,Cloud,rti_l,rti_c,rti_ws,ftj_ws,ftj_
wr,ftj_l,ftj,core,core1,n,k,t_s,t_r,t_c,cloudlist);
  time_req2 = clock() - time_req2;
  for(int i=0; i<4; i++)
  {
    for(int j=0; j<10; j++)
    {
      if(core1[j] == i)
      {
        cloudlist[i].push_back(j);
      }
    }
  }
  float energy_total = 0;
  for(int i=0; i<10; i++)
  {
    if(core1[i] == 3)
    {
      energy_total += Ei_c;
    }
    else
    {
      energy_total += Ei_l[i][core1[i]];
    }
  }
  int ab[10];
  for(int i=0; i<10; i++)
  {
    ab[i] = max(rti_l[i],rti_ws[i]);
    }

    int timemin = ftj[n-1];
    int timemax = 27;
    cout<<"Initial Scheduling (Format: Start Time - Task Number - Finish Time)"<<endl;
    for(int i=0; i<cloudlist.size(); i++)
    {
        if(i == 3)
        {
            cout<<"Cloud: ";
        }
        else{
            cout<<"Core"<<i+1<<": ";
        }
        for(int j=0; j<cloudlist[i].size(); j++)
        {
            cout<<ab[cloudlist[i][j]]<<"-Task"<<cloudlist[i][j]+1<<"-"<<ftj[cloud
list[i][j]]<<" ";
        }
        cout<<endl;
    }
    cout<<"Initial Energy Consumption: "<<energy_total<<"   Initial Completion
Time: "<<timemin<<endl;
    cout<<endl;
    kernel(cloudlist,task,adjacentmatrix,core1,timemax,timemin,energy_total,10,3,
ab,ftj,Ei_c,Ei_l);
    time_req1 = clock() - time_req1;
    cout << endl << "Time taken by initial scheduling is : " <<(double)time_req2 /
CLOCKS_PER_SEC<<"seconds"<< endl;
    cout << endl << "Time taken by scheduling result after task migration is  : "
<<(double)time_req1 / CLOCKS_PER_SEC<<"seconds"<< endl;
    return 0;
}
```

OUTPUT 1:

Enter the number of task 10

Enter the number of the cores 3

Enter the time required to send to the cloud 3

Enter the time required for the cloud to process 1

Enter the time required by the cloud to send the data back 1

Enter the power comsumption rate core 1 : 1

```
  Enter the power comsumption rate core 2 : 2

  Enter the power comsumption rate core 3 : 4

Enter energy Consumption power while sending the data 0.5

Initial Scheduling (Format: Start Time - Task Number - Finish Time)

Core1: 5-Task4-12

Core2: 5-Task6-11 13-Task8-16

Core3: 0-Task1-5 5-Task3-9 9-Task5-11 11-Task7-14 14-Task9-16 16-Task10-18

Cloud: 5-Task2-10

Initial Energy Consumption: 100.5      Initial Completion Time: 18


1th step

  Current Operation: Insert Task 1 to Core 4

  Current Completion Time: 18      Current Energy Consumption: 81

2th step

  Current Operation: Insert Task 6 to Core 4

  Current Completion Time: 19      Current Energy Consumption: 70

3th step

  Current Operation: Insert Task 8 to Core 4

  Current Completion Time: 20      Current Energy Consumption: 63

4th step

  Current Operation: Insert Task 7 to Core 4

  Current Completion Time: 21      Current Energy Consumption: 52

5th step

  Current Operation: Insert Task 3 to Core 2

  Current Completion Time: 21      Current Energy Consumption: 46

6th step

  Current Operation: Insert Task 3 to Core 4

  Current Completion Time: 24      Current Energy Consumption: 37

7th step
```

```
  Current Operation: Insert Task 5 to Core 4
  Current Completion Time: 27      Current Energy Consumption: 30
8th step
  Current Operation: Insert Task 9 to Core 1
  Current Completion Time: 27      Current Energy Consumption: 27

  Best Asssignment
  Best Energy Consumption: 27      Best Completion Time: 27
Core0: 4 9
Core1:
Core2: 10
Clould: 1 2 5 3 6 7 8

Time Assignment Details
Core 1: 5-Task4-12 13-Task9-18
Core 2:
Core 3: 25-Task10-27
Cloud: 0-Task1-5 5-Task2-10 8-Task5-13 11-Task3-16 14-Task6-19 17-Task7-22 20-Task8-25

Time taken by initial scheduling is : 2.1e-05seconds

Time taken by scheduling result after task migration is    : 0.002039seconds
```