Northeastern University

EECE 7205 - Fundamentals of Computer Engineering
Project 2

PRAYAG SRIDHAR

NUID : 002080391

PROGRAM SETUP

1) First part is the header and the namespace

```cpp
#include <iostream>
#include <vector>
#include <limits>
#include <chrono>
#include <ctime>
#include <climits>
#include <algorithm>
#include <iterator>

using namespace std;
```

2) This part involves creating a new data structure '**JobUnit**' which denotes Tasks.

```cpp
class JobUnit {
public:
    int identifier;
    bool executeOnCloud;
    double priorityMetric;
    int finishLocal;
    int finishSending;
    int finishCloud;
    int finishReceiving;
    int readyLocal;
    int readySending;
    int readyCloud;
    int readyReceiving;
    int beginTime;
    int assignedResource;
    bool terminalJob;
    bool initialJob;
    int predCountReady;
    int sameChannelReady;
};
```

3) Utility max function helps in returning the maximum of 2 integers

```
int maxOfTwo(int a, int b) {
    return (a >= b) ? a : b;
}
```

4) The function sets each job's **identifier** and finds the minimal local execution time among all cores. If the minimal local execution time is greater than **limitThreshold**, it marks the job as '*executeOnCloud = true*', suggesting it should run on the cloud.

- **jobPool** holds all jobs.
- **execMatrix** is a 2D matrix where **execMatrix[i][j]** is the execution time of job i on core j.
- **limitThreshold** is a threshold that helps decide if a job should be executed on the cloud.

```
// Phase A: Initial Job Assignment
void initialJobAssignment(vector<JobUnit>& jobPool, const vector<vector<int>>& execMatrix, int limitThreshold)
{
    for (size_t i = 0; i < execMatrix.size(); i++) {
        jobPool[i].identifier = (int)i + 1;
        int minimalTime = execMatrix[i][0];
        for (size_t j = 0; j < execMatrix[i].size(); j++)
            if (execMatrix[i][j] < minimalTime)
                minimalTime = execMatrix[i][j];

        jobPool[i].executeOnCloud = (minimalTime > limitThreshold);
    }
}
```

5) This determines the tasks (jobs) which are initial (no incoming edges) and terminal (no outgoing edges).
The priority of each job (task) based on its average local execution time if **local**, or **limitThreshold** if on cloud, plus the highest priority of its successors.

```
void computePriorities(vector<JobUnit>& jobPool, const vector<vector<int>>& execMatrix, const vector<vector<int>>& dependencyGraph, int limitThreshold)
{
    size_t endIndex = jobPool.size() - 1;
    for (size_t idx = 0; idx < jobPool.size(); idx++) {
        int countSucc = 0;
        // Check if terminal job
        for (size_t j = 0; j < dependencyGraph[endIndex - idx].size(); j++)
            if (dependencyGraph[endIndex - idx][j] == 1)
                countSucc++;
        if (countSucc == 0)
            jobPool[endIndex - idx].terminalJob = true;

        countSucc = 0;
        // Check if initial job
        for (size_t j = 0; j < dependencyGraph.size(); j++)
            if (dependencyGraph[j][endIndex - idx] == 1)
                countSucc++;
        if (countSucc == 0)
            jobPool[endIndex - idx].initialJob = true;

        double maxPrio = 0;
        double avgTime;
        if (!jobPool[endIndex - idx].executeOnCloud) {
            double sumT = 0;
            for (size_t j = 0; j < execMatrix[endIndex - idx].size(); j++)
                sumT += execMatrix[endIndex - idx][j];
            avgTime = sumT / 3.0;
        }
        else {
            avgTime = limitThreshold;
        }

        for (size_t j = 0; j < dependencyGraph[endIndex - idx].size(); j++)
            if ((dependencyGraph[endIndex - idx][j] == 1) && (maxPrio < jobPool[j].priorityMetric))
                maxPrio = jobPool[j].priorityMetric;

        jobPool[endIndex - idx].priorityMetric = avgTime + maxPrio;
    }
}
```

6) This loop is about finding the job with the maximum priority

```
int findMaxPriorityIndex(const vector<JobUnit>& units)
{
    int maxIdx = 0;
    for (size_t i = 0; i < units.size(); i++)
        if (units[i].priorityMetric > units[maxIdx].priorityMetric)
            maxIdx = (int)i;
    return maxIdx;
}
```

7) This **localReadyCalc** function calculates when a job can start locally, based on the completion times of its prerequisites.

```
// Compute the earliest local ready time
int localReadyCalc(JobUnit& unit, const vector<JobUnit>& scheduled, const vector<vector<int>>& graph)
{
    int val = 0;
    if (!scheduled.empty()) {
        for (size_t i = 0; i < graph.size(); i++)
            if (graph[i][unit.identifier - 1] == 1)
                for (size_t j = 0; j < scheduled.size(); j++)
                    if ((scheduled[j].identifier == (int)i + 1) &&
                        (val < maxOfTwo(scheduled[j].finishLocal, scheduled[j].finishReceiving)))
                        val = maxOfTwo(scheduled[j].finishLocal, scheduled[j].finishReceiving);
    }
    return val;
}
```

8) cloudSendReadyCalc / cloudComputeReadyCalc / cloudReceiveReadyCalc: Similar logic to **localReadyCalc**, but these functions calculate when a job can start sending to the cloud, when it can start computing on the cloud, and when it can start receiving results from the cloud, based on the dependency graph and previously scheduled tasks.

```
int cloudSendReadyCalc(JobUnit& unit, const vector<JobUnit>& scheduled, const vector<vector<int>>& graph)
{
    int val = 0;
    if (!scheduled.empty()) {
        for (size_t i = 0; i < graph.size(); i++)
            if (graph[i][unit.identifier - 1] == 1)
                for (size_t j = 0; j < scheduled.size(); j++)
                    if ((scheduled[j].identifier == (int)i + 1) &&
                        (val < maxOfTwo(scheduled[j].finishLocal, scheduled[j].finishSending)))
                        val = maxOfTwo(scheduled[j].finishLocal, scheduled[j].finishSending);
    }
    return val;
}

// Cloud compute ready time
int cloudComputeReadyCalc(JobUnit& unit, const vector<JobUnit>& scheduled, const vector<vector<int>>& graph)
{
    int val = unit.finishSending;
    if (!scheduled.empty()) {
        for (size_t i = 0; i < graph.size(); i++)
            if (graph[i][unit.identifier - 1] == 1)
                for (size_t j = 0; j < scheduled.size(); j++)
                    if ((scheduled[j].identifier == (int)i + 1) &&
                        (val < maxOfTwo(unit.finishSending, scheduled[j].finishCloud)))
                        val = maxOfTwo(unit.finishSending, scheduled[j].finishCloud);
    }
    return val;
}

// Cloud receive ready time
int cloudReceiveReadyCalc(JobUnit& unit)
{
    return unit.finishCloud;
}
```

9) LocalScheduling: This tries all local cores to find the one that produces the earliest finishing time for this job, considering that it can't start before all its prerequisites are finished and that the core might already be occupied by previously scheduled tasks.

```
int localScheduling(JobUnit& unit, const vector<JobUnit>& S, const vector<vector<int>>& G, const vector<vector<int>>& timeMatrix)
{
    unit.readyLocal = localReadyCalc(unit, S, G);
    int minimalFinish = INT_MAX;
    int fTime;
    if (S.empty()) {
        for (size_t i = 0; i < timeMatrix[unit.identifier - 1].size(); i++) {
            fTime = unit.readyLocal + timeMatrix[unit.identifier - 1][i];
            if (fTime < minimalFinish) {
                minimalFinish = fTime;
                unit.assignedResource = (int)i;
            }
        }
        return minimalFinish;
    }

    for (size_t i = 0; i < timeMatrix[unit.identifier - 1].size(); i++) {
        fTime = unit.readyLocal + timeMatrix[unit.identifier - 1][i];
        int maxLocalF = 0;
        for (size_t j = 0; j < S.size(); j++)
            if ((S[j].assignedResource == (int)i) && (maxLocalF < S[j].finishLocal))
                maxLocalF = S[j].finishLocal;
        if (maxLocalF > unit.readyLocal)
            fTime = maxLocalF + timeMatrix[unit.identifier - 1][i];
        if (fTime < minimalFinish) {
            minimalFinish = fTime;
            unit.assignedResource = (int)i;
        }
    }
    return minimalFinish;
}
```

10) CloudScheduling:

- Similar to **localScheduling** but splits the job (task) into phases
- Sends data to cloud, executes on cloud, and receives results.
- It calculates all these intervals considering other tasks that also use the cloud.

```
// Schedule job on cloud
int cloudScheduling(JobUnit& unit, const vector<JobUnit>& S, const vector<vector<int>>& G, int sendTime, int cloudTime, int recvTime)
{
    unit.readySending = cloudSendReadyCalc(unit, S, G);
    int totalDelay = sendTime + cloudTime + recvTime;
    int maxS = 0, maxC = 0, maxR = 0, finalF;

    if (S.empty()) {
        unit.finishSending = sendTime;
        unit.readyCloud = sendTime;
        unit.finishCloud = sendTime + cloudTime;
        unit.readyReceiving = sendTime + cloudTime;
        return totalDelay;
    }

    for (size_t i = 0; i < S.size(); i++)
        if (S[i].assignedResource == 3 && maxS < S[i].finishSending)
            maxS = S[i].finishSending;

    if (maxS > unit.readySending)
        unit.finishSending = maxS + sendTime;
    else
        unit.finishSending = unit.readySending + sendTime;

    unit.readyCloud = cloudComputeReadyCalc(unit, S, G);

    for (size_t i = 0; i < S.size(); i++)
        if (S[i].assignedResource == 3 && maxC < S[i].finishCloud)
            maxC = S[i].finishCloud;

    if (maxC > unit.readyCloud)
        unit.finishCloud = maxC + cloudTime;
    else
        unit.finishCloud = unit.readyCloud + cloudTime;

    unit.readyReceiving = cloudReceiveReadyCalc(unit);

    for (size_t i = 0; i < S.size(); i++)
        if (S[i].assignedResource == 3 && maxR < S[i].finishReceiving)
            maxR = S[i].finishReceiving;

    if (maxR > unit.readyReceiving)
        finalF = maxR + recvTime;
    else
        finalF = unit.readyReceiving + recvTime;

    return finalF;
}
```

11) jobFinishTime: Returns the absolute finishing time of a job (task).
12) printFinalSchedule: Prints each job, indicating whether it ran on a local core or the cloud and shows its start and finish times.
13) findCompletionTime: The completion time of the schedule is the maximum finish time among all terminal jobs (tasks).

```
int jobFinishTime(JobUnit& unit)
{
    return maxOfTwo(unit.finishLocal, unit.finishReceiving);
}

// Print the final schedule
void printFinalSchedule(const vector<JobUnit>& S)
{
    for (size_t i = 0; i < S.size(); i++) {
        int doneT = jobFinishTime((JobUnit&)S[i]);
        cout << "Job" << S[i].identifier << ": ";
        switch (S[i].assignedResource) {
        case 0:
            cout << "Local Resource " << S[i].assignedResource + 1 << ", ";
            break;
        case 1:
            cout << "Local Resource " << S[i].assignedResource + 1 << ", ";
            break;
        case 2:
            cout << "Local Resource " << S[i].assignedResource + 1 << ", ";
            break;
        case 3:
            cout << "Cloud Resource, ";
            break;
        default:
            break;
        }
        cout << "start: " << S[i].beginTime << ", finish: " << doneT << endl;
    }
}

// Determine overall completion time
int findCompletionTime(const vector<JobUnit>& S)
{
    int mTime = 0;
    for (size_t i = 0; i < S.size(); i++)
        if ((S[i].terminalJob) && (mTime < jobFinishTime((JobUnit&)S[i])))
            mTime = jobFinishTime((JobUnit&)S[i]);
    return mTime;
}
```

14) ComputeEnergy: This loop computes total energy based on which resource a job (task) used and how long it took. Here p1, p2, p3 is the power coefficients for local cores, ps is for the cloud service

15) updateReadyCount1 / updateReadyCount2: These maintain counts for how many prerequisites are ready and how many tasks on the same channel started before this one, used in more advanced scheduling and migration logic.

```
double computeEnergy(const vector<JobUnit>& S, int p1, int p2, int p3, double ps)
{
    double energy = 0;
    for (size_t i = 0; i < S.size(); i++) {
        int ft = jobFinishTime((JobUnit&)S[i]);
        switch (S[i].assignedResource) {
        case 0:
            energy += p1 * (ft - S[i].beginTime);
            break;
        case 1:
            energy += p2 * (ft - S[i].beginTime);
            break;
        case 2:
            energy += p3 * (ft - S[i].beginTime);
            break;
        case 3:
            energy += ps * (S[i].finishSending - S[i].beginTime);
            break;
        default:
            break;
        }
    }
    return energy;
}

// Compute ready counts
void updateReadyCount1(vector<JobUnit>& S, const vector<vector<int>>& G)
{
    for (size_t i = 0; i < S.size(); i++) {
        int c = 0;
        for (size_t j = 0; j < G.size(); j++)
            if (G[j][S[i].identifier - 1] == 1)
                for (size_t k = 0; k < S.size(); k++)
                    if (S[k].identifier == (int)j + 1)
                        c++;
        S[i].predCountReady = c;
    }
}

void updateReadyCount2(vector<JobUnit>& S)
{
    for (size_t i = 0; i < S.size(); i++) {
        int c = 0;
        for (size_t j = 0; j < S.size(); j++)
            if ((S[i].assignedResource == S[j].assignedResource) && (S[j].beginTime < S[i].beginTime))
                c++;
        S[i].sameChannelReady = c;
    }
}
```

16) **finalizeLocal**: Finalizes the local scheduling of a job once its local core is known.

17) **reScheduling**: Attempts to reschedule a job to a new resource (e.g., migrate from one local core to another or from local to cloud).

```
int finalizeLocal(JobUnit& v, const vector<JobUnit>& SN, const vector<vector<int>>& G, const vector<vector<int>>& execTimes)
{
    v.readyLocal = localReadyCalc(v, SN, G);
    int fTime;
    int maxLoc = 0;
    if (SN.empty())
        fTime = v.readyLocal + execTimes[v.identifier - 1][v.assignedResource];
    else {
        for (size_t i = 0; i < SN.size(); i++)
            if ((SN[i].assignedResource == v.assignedResource) && (maxLoc < SN[i].finishLocal))
                maxLoc = SN[i].finishLocal;
        if (maxLoc > v.readyLocal)
            fTime = maxLoc + execTimes[v.identifier - 1][v.assignedResource];
        else
            fTime = v.readyLocal + execTimes[v.identifier - 1][v.assignedResource];
    }
    return fTime;
}

void reScheduling(const vector<JobUnit>& S, vector<JobUnit>& SN, int newRes, JobUnit target, const vector<vector<int>>& G, const vector<vector<int>>& times, int ts, int tc, int tr)
{
    vector<JobUnit> temp = S;
    int fullDelay = ts + tc + tr;

    for (size_t i = 0; i < temp.size(); i++)
        if (target.identifier == temp[i].identifier) {
            temp[i].assignedResource = newRes;
            if (newRes == 3) {
                temp[i].finishLocal = 0;
                temp[i].readyLocal = 0;
            }
        }

    while (!temp.empty()) {
        updateReadyCount1(temp, G);
        updateReadyCount2(temp);
        size_t m = 0;
        while (m < temp.size() && (temp[m].predCountReady != 0) && (temp[m].sameChannelReady != 0))
            m++;
        if (temp[m].assignedResource == 3) {
            temp[m].finishReceiving = cloudScheduling(temp[m], SN, G, ts, tc, tr);
            temp[m].beginTime = temp[m].finishReceiving - fullDelay;
        }
        else {
            temp[m].finishLocal = finalizeLocal(temp[m], SN, G, times);
            temp[m].beginTime = temp[m].finishLocal - times[temp[m].identifier - 1][temp[m].assignedResource];
        }
        SN.push_back(temp[m]);
        temp.erase(temp.begin() + m);
    }
}
```

18) migrateJobs: This attempts migrating jobs (tasks) between resources to reduce energy or completion time. Checks if migrations produce better results and updates the schedule accordingly.

19) outerLoopRefine: Continues trying to improve the schedule by calling **migrateJobs** repeatedly until no further improvement in energy is found.

```
void migrateJobs(vector<JobUnit>& S, const vector<vector<int>>& G, const vector<vector<int>>& times, int ts, int tc, int tr, int p1, int p2, int p3, double ps, int timeMax)
{
    int origCompletion = findCompletionTime(S);
    double origEnergy = computeEnergy(S, p1, p2, p3, ps);
    double bestGain = 0;

    for (size_t i = 0; i < S.size(); i++) {
        int currRes = S[i].assignedResource;
        if (S[i].assignedResource != 3) {
            for (int newRes = 0; newRes < 4; newRes++) {
                if (newRes != currRes) {
                    vector<JobUnit> newSeq;
                    double oldE = computeEnergy(S, p1, p2, p3, ps);
                    reScheduling(S, newSeq, newRes, S[i], G, times, ts, tc, tr);
                    int newComp = findCompletionTime(newSeq);
                    double newE = computeEnergy(newSeq, p1, p2, p3, ps);
                    if ((newE < oldE) && (origCompletion >= newComp))
                        S = newSeq;
                    else if ((newE < oldE) && (newComp <= timeMax)) {
                        double ratio = (origEnergy - newE) / (newComp - origCompletion);
                        if (ratio > bestGain) {
                            bestGain = ratio;
                            S = newSeq;
                        }
                    }
                }
            }
        }
    }
}

void outerLoopRefine(vector<JobUnit>& S, const vector<vector<int>>& G, const vector<vector<int>>& times, int ts, int tc, int tr, int p1, int p2, int p3, double ps, int timeMax)
{
    double prevEn;
    double nextEn = 0;
    prevEn = computeEnergy(S, p1, p2, p3, ps);
    while (nextEn < prevEn) {
        prevEn = computeEnergy(S, p1, p2, p3, ps);
        migrateJobs(S, G, times, ts, tc, tr, p1, p2, p3, ps, timeMax);
        nextEn = computeEnergy(S, p1, p2, p3, ps);
    }
}
```

20) initialScheduling: Builds an initial schedule by repeatedly taking the job (task) with the highest priority and deciding whether it runs locally or on cloud. It also compares local vs. cloud finishing times and chooses the better option initially.

```
void initialScheduling(vector<JobUnit>& finalSet, vector<JobUnit>& initialJobs, const vector<vector<int>>& times, const vector<vector<int>>& G, int ts, int tc, int tr)
{
    int fullLatency = ts + tc + tr;
    for (size_t i = 0; i < G.size(); i++) {
        int maxPidx = findMaxPriorityIndex(initialJobs);
        if (!initialJobs[maxPidx].executeOnCloud) {
            int localMin = localScheduling(initialJobs[maxPidx], finalSet, G, times);
            int cloudFin = cloudScheduling(initialJobs[maxPidx], finalSet, G, ts, tc, tr);
            if (cloudFin < localMin) {
                initialJobs[maxPidx].readyLocal = 0;
                initialJobs[maxPidx].finishLocal = 0;
                initialJobs[maxPidx].assignedResource = 3;
                initialJobs[maxPidx].finishReceiving = cloudFin;
                initialJobs[maxPidx].beginTime = cloudFin - fullLatency;
            }
            else {
                initialJobs[maxPidx].finishCloud = 0;
                initialJobs[maxPidx].finishSending = 0;
                initialJobs[maxPidx].readySending = 0;
                initialJobs[maxPidx].readyCloud = 0;
                initialJobs[maxPidx].readyReceiving = 0;
                initialJobs[maxPidx].finishReceiving = 0;
                initialJobs[maxPidx].finishLocal = localMin;
                initialJobs[maxPidx].beginTime = localMin - times[initialJobs[maxPidx].identifier - 1][initialJobs[maxPidx].assignedResource];
            }
        }
        else {
            initialJobs[maxPidx].finishLocal = 0;
            initialJobs[maxPidx].readyLocal = 0;
            initialJobs[maxPidx].assignedResource = 3;
            initialJobs[maxPidx].finishReceiving = cloudScheduling(initialJobs[maxPidx], finalSet, G, ts, tc, tr);
            initialJobs[maxPidx].beginTime = initialJobs[maxPidx].finishReceiving - fullLatency;
        }
        finalSet.push_back(initialJobs[maxPidx]);
        initialJobs.erase(initialJobs.begin() + maxPidx);
    }
}
```

21) main function:

- Prompts the user for number of tasks, cores, and various times and power parameters.
- Reads execution times and the dependency graph from the user.
- Creates initial job list and calls **initialJobAssignment** and **computePriorities**.
- Does an initial scheduling and prints results.
- Then tries to refine the schedule by calling **outerLoopRefine** to improve metrics
- Finally prints the improved schedule and results.

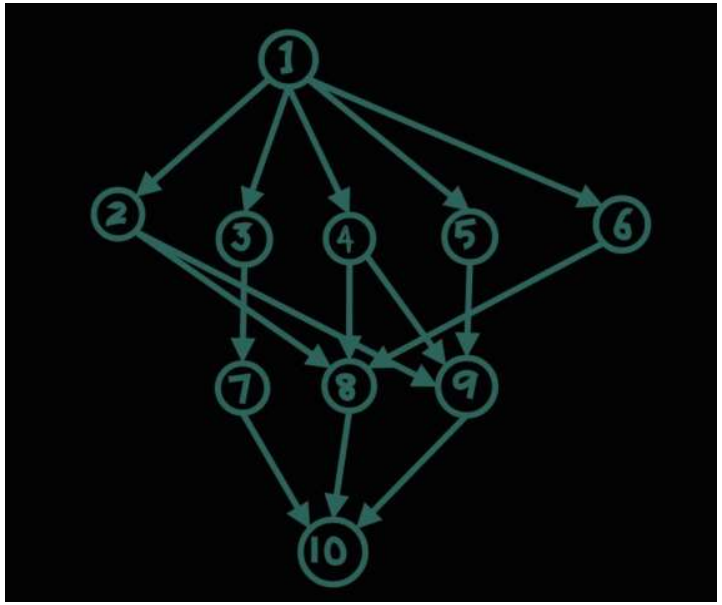
```
int main()
{
    int TASK_COUNT;
    int NATIVE_CORES;

    cout << "Enter number of tasks (TASK_COUNT): ";
    cin >> TASK_COUNT;
    cout << "Enter number of cores (NATIVE_CORES): ";
    cin >> NATIVE_CORES;

    int tSend, tCloud, tReceive;
    int tMaximum;
    int POWER_RES_1, POWER_RES_2, POWER_RES_3;
    double POWER_CLOUD;

    cout << "Enter sending time (tSend): ";
    cin >> tSend;
    cout << "Enter cloud computation time (tCloud): ";
    cin >> tCloud;
    cout << "Enter receiving time (tReceive): ";
    cin >> tReceive;
    cout << "Enter maximum allowed completion time (tMaximum): ";
    cin >> tMaximum;

    cout << "Enter power consumption for resource 1 (POWER_RES_1): ";
    cin >> POWER_RES_1;
    cout << "Enter power consumption for resource 2 (POWER_RES_2): ";
    cin >> POWER_RES_2;
    cout << "Enter power consumption for resource 3 (POWER_RES_3): ";
    cin >> POWER_RES_3;
    cout << "Enter cloud power consumption (POWER_CLOUD): ";
    cin >> POWER_CLOUD;
```

```
    vector<vector<int>> executionMatrix(TASK_COUNT, vector<int>(NATIVE_CORES, 0));
    cout << "Enter the execution times for " << TASK_COUNT << " jobs on "
         << NATIVE_CORES << " local units (one line per job):" << endl;
    for (int i = 0; i < TASK_COUNT; i++) {
        for (int j = 0; j < NATIVE_CORES; j++) {
            cin >> executionMatrix[i][j];
        }
    }

    vector<vector<int>> dependencyGraph(TASK_COUNT, vector<int>(TASK_COUNT, 0));
    cout << "Enter the dependency matrix (" << TASK_COUNT << "x" << TASK_COUNT << "):" << endl;
    for (int i = 0; i < TASK_COUNT; i++) {
        for (int j = 0; j < TASK_COUNT; j++) {
            cin >> dependencyGraph[i][j];
        }
    }

    vector<JobUnit> initialJobs(TASK_COUNT);
    vector<JobUnit> finalSchedule;

    int limitVal = tSend + tCloud + tReceive;

    initialJobAssignment(initialJobs, executionMatrix, limitVal);
    computePriorities(initialJobs, executionMatrix, dependencyGraph, limitVal);

    clock_t startRun, endRun;
    startRun = clock();
    initialScheduling(finalSchedule, initialJobs, executionMatrix, dependencyGraph, tSend, tCloud, tReceive);
    endRun = clock();

    cout << "Initial schedule:" << endl;
    printFinalSchedule(finalSchedule);
    double runTime = (double)(endRun - startRun) / (double)CLOCKS_PER_SEC * 1000.0;
    cout << "Total energy: " << computeEnergy(finalSchedule, POWER_RES_1, POWER_RES_2, POWER_RES_3, POWER_CLOUD) << endl;
    cout << "Completion time: " << findCompletionTime(finalSchedule) << endl;
    cout << "Initial scheduling time: " << runTime << " ms" << endl;

    startRun = clock();
    outerLoopRefine(finalSchedule, dependencyGraph, executionMatrix, tSend, tCloud, tReceive, POWER_RES_1, POWER_RES_2, POWER_RES_3, POWER_CLOUD, tMaximum);
    endRun = clock();

    cout << "After migrations:" << endl;
    printFinalSchedule(finalSchedule);
    runTime = (double)(endRun - startRun) / (double)CLOCKS_PER_SEC * 1000.0;
    cout << "Total energy: " << computeEnergy(finalSchedule, POWER_RES_1, POWER_RES_2, POWER_RES_3, POWER_CLOUD) << endl;
    cout << "Completion time: " << findCompletionTime(finalSchedule) << endl;
    cout << "Refinement time: " << runTime << " ms" << endl;
    cout << endl;
    cout << "Press Enter to continue..." << endl;
    cin.get();

    return 0;
}
```

EXAMPLE-1:

$T_{maximum} = 27$; $T_{send} = 3$; $T_{cloud} = 1$; $T_{receive} = 1$

$P_{res-1} = 1$; $P_{res-2} = 2$; $P_{res-3} = 4$; $P_{cloud} = 0.5$



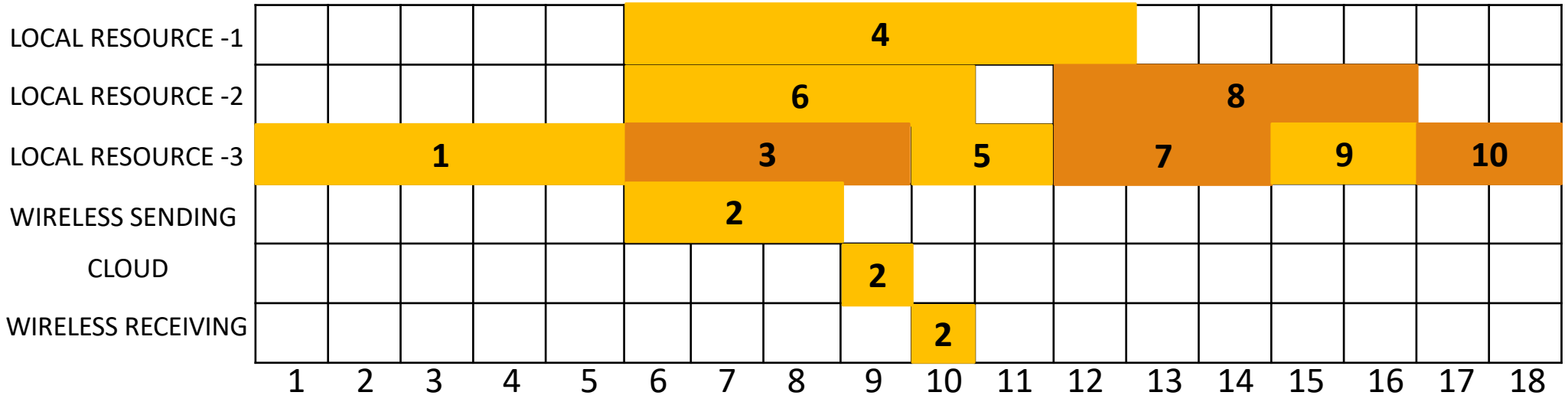
Time in each core	Localcore-1	Localcore-2	Localcore-3
Task 1	9	7	5
Task 2	8	6	5
Task 3	6	5	4
Task 4	7	5	3
Task 5	5	4	2
Task 6	7	6	4
Task 7	8	5	3
Task 8	6	4	2
Task 9	5	3	2
Task 10	7	4	2

Initial Scheduling

Time = 18; Energy = 100.5; Scheduling Time = 0 ms

```
Initial schedule:
Job1: Local Resource 3, start: 0, finish: 5
Job3: Local Resource 3, start: 5, finish: 9
Job2: Cloud Resource, start: 5, finish: 10
Job6: Local Resource 2, start: 5, finish: 11
Job4: Local Resource 1, start: 5, finish: 12
Job5: Local Resource 3, start: 9, finish: 11
Job7: Local Resource 3, start: 11, finish: 14
Job8: Local Resource 2, start: 12, finish: 16
Job9: Local Resource 3, start: 14, finish: 16
Job10: Local Resource 3, start: 16, finish: 18
Total energy: 100.5
Completion time: 18
Initial scheduling time: 0 ms
```

Output of the example after initial schedule

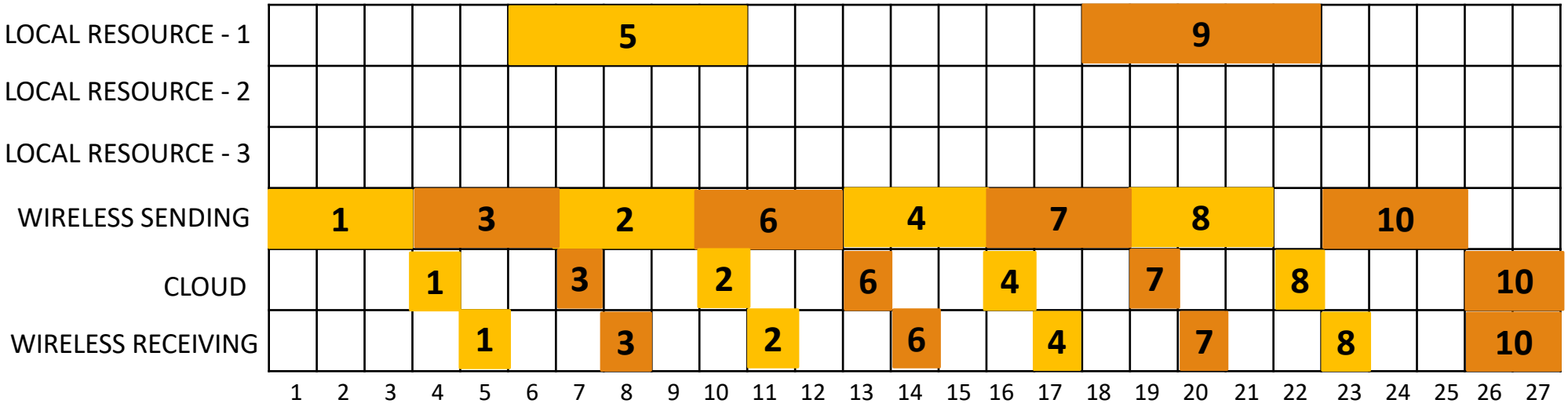


$E_1=(7x1)=7$; $E_2=(6+4)x2=20$;$E_3=(5+4+2+3+2+2)x4=72$;
$E_s=(3x0.5)=1.5$;
$E=7+20+72+1.5=100.5$

After Migrations:
Time = 27; Energy = 22; Run Time = 5 ms

```
After migrations:
Job1: Cloud Resource, start: 0, finish: 5
Job3: Cloud Resource, start: 3, finish: 8
Job2: Cloud Resource, start: 6, finish: 11
Job6: Cloud Resource, start: 9, finish: 14
Job4: Cloud Resource, start: 12, finish: 17
Job5: Local Resource 1, start: 5, finish: 10
Job7: Cloud Resource, start: 15, finish: 20
Job8: Cloud Resource, start: 18, finish: 23
Job9: Local Resource 1, start: 17, finish: 22
Job10: Cloud Resource, start: 22, finish: 27
Total energy: 22
Completion time: 27
Refinement time: 5 ms
```

Output of the example after Task Migration Schedule



$E_1=(5+5)*1=10$; $E_2=0$; $E_3=0$;

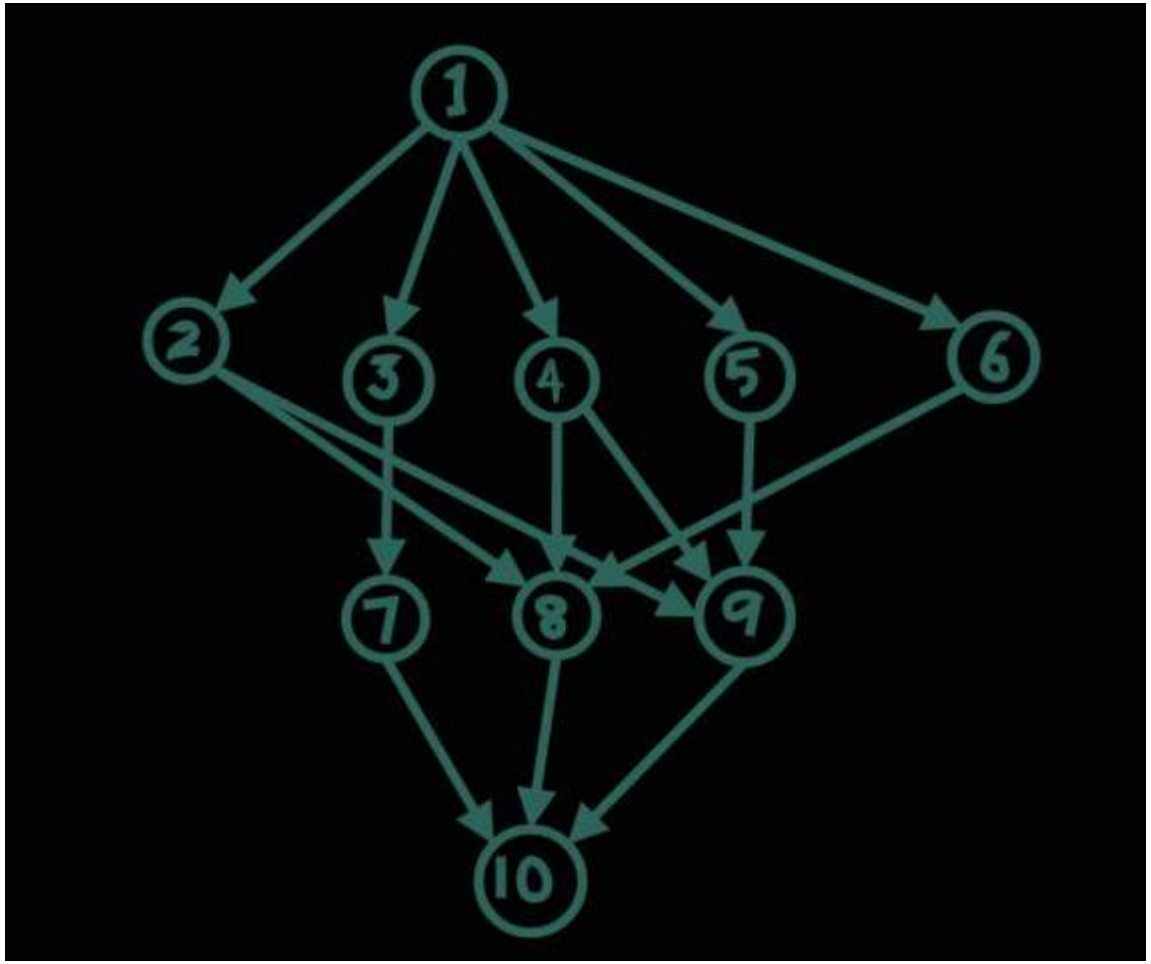
$E_s=(3*8*0.5)=12$;

$E=10+12=22$

EXAMPLE-2:

$T_{maximum} = 45$; $T_{send} = 3$; $T_{cloud} = 1$; $T_{receive} = 1$

$P_{res-1} = 1$; $P_{res-2} = 2$; $P_{res-3} = 4$; $P_{cloud} = 0.5$



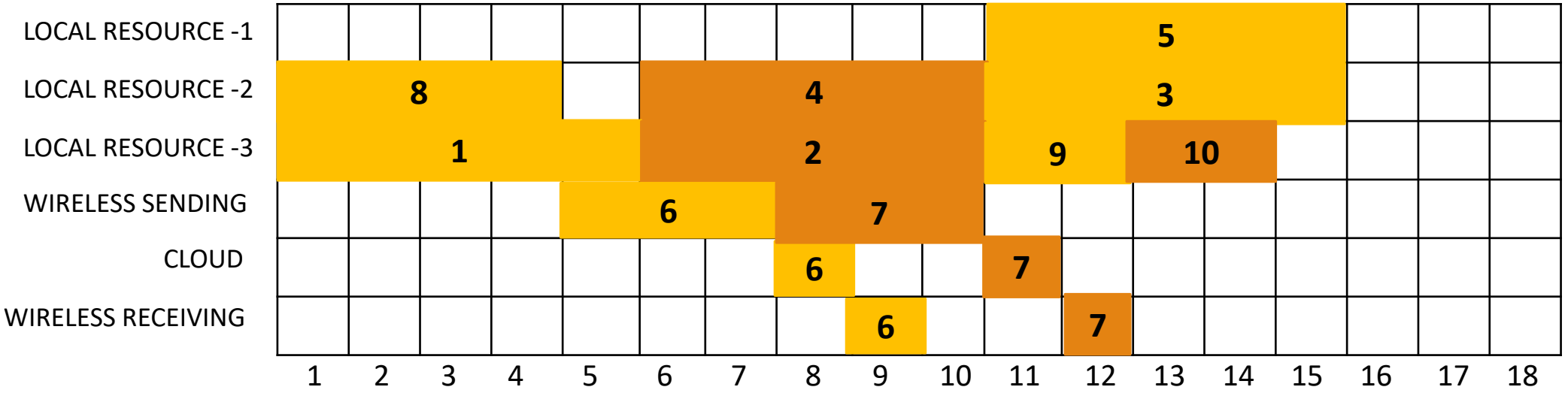
Time in each core	Localcore-1	Localcore-2	Localcore-3
Task 1	9	7	5
Task 2	8	6	5
Task 3	6	5	4
Task 4	7	5	3
Task 5	5	4	2
Task 6	7	6	4
Task 7	8	5	3
Task 8	6	4	2
Task 9	5	3	2
Task 10	7	4	2

```
Initial schedule:
Job1: Local Resource 3, start: 0, finish: 5
Job2: Local Resource 3, start: 5, finish: 10
Job8: Local Resource 2, start: 0, finish: 4
Job6: Cloud Resource, start: 4, finish: 9
Job4: Local Resource 2, start: 5, finish: 10
Job9: Local Resource 3, start: 10, finish: 12
Job7: Cloud Resource, start: 7, finish: 12
Job3: Local Resource 2, start: 10, finish: 15
Job10: Local Resource 3, start: 12, finish: 14
Job5: Local Resource 1, start: 10, finish: 15
Total energy: 92
Completion time: 14
Initial scheduling time: 0 ms
```

Initial Scheduling

Time = 14; Energy = 92; Scheduling Time = 0 ms

Output of the example after initial schedule:

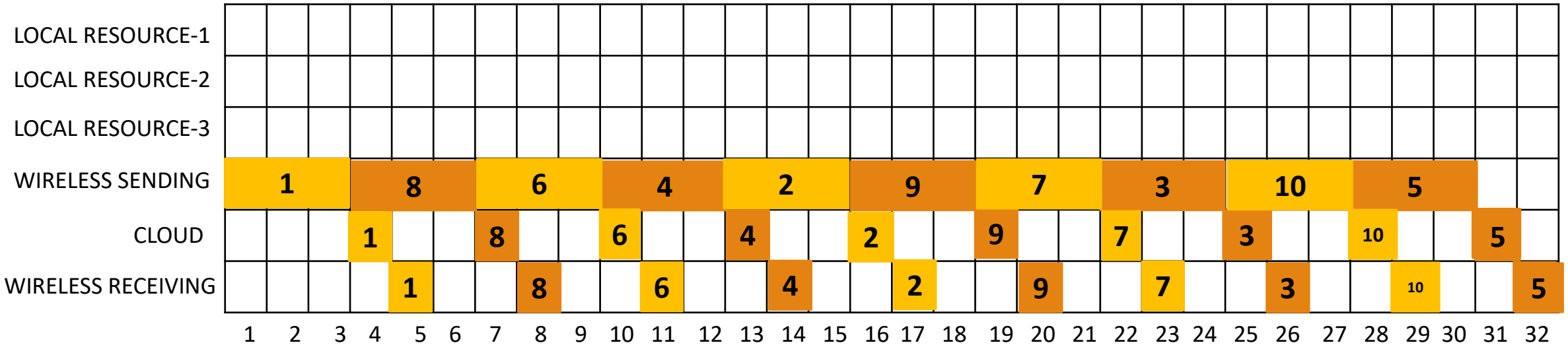


$E_1=(5x1)=5$; $E_2=(4+5+5)x2=28$;$E_3=(5+5+2+2)x4=56$;
$E_s=(3x0.5X2)=3$;
E=5+28+56+3=92

After Migrations:
Time = 29; Energy = 15; Run Time = 4 ms

```
After migrations:
Job1: Cloud Resource, start: 0, finish: 5
Job8: Cloud Resource, start: 3, finish: 8
Job6: Cloud Resource, start: 6, finish: 11
Job4: Cloud Resource, start: 9, finish: 14
Job2: Cloud Resource, start: 12, finish: 17
Job9: Cloud Resource, start: 15, finish: 20
Job7: Cloud Resource, start: 18, finish: 23
Job3: Cloud Resource, start: 21, finish: 26
Job10: Cloud Resource, start: 24, finish: 29
Job5: Cloud Resource, start: 27, finish: 32
Total energy: 15
Completion time: 29
Refinement time: 4 ms
```

Output of the example after Task Migration schedule



$E_1 + E_2 + E_3 = 0$

$E_S = (3*10*0.5) = 15$

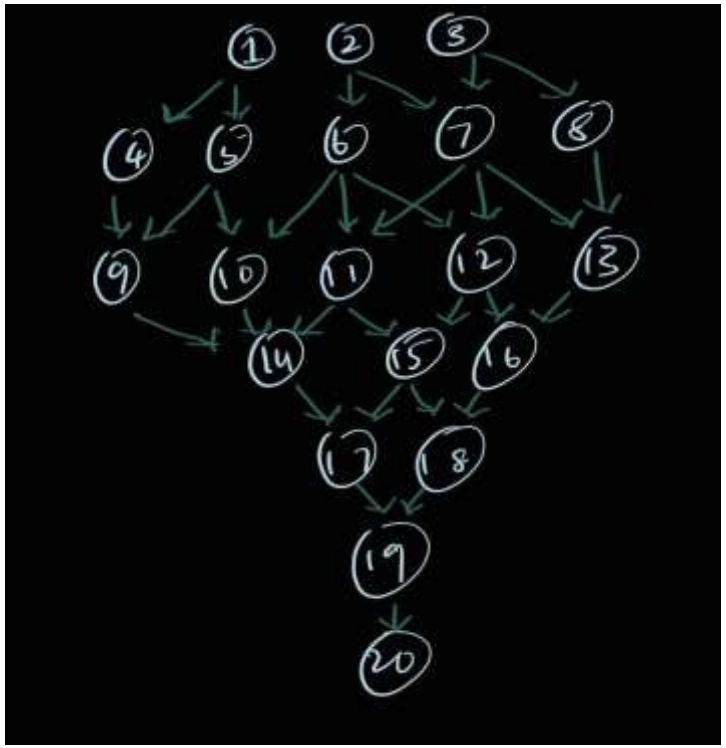
$E = E_1 + E_2 + E_3 + E_S = 15$

EXAMPLE-3:

$T_{max}=50$

$T_s=3$; $T_c=1$; $T_r=1$;

$P_1=1$; $P_2=2$; $P_3=4$; $P_s=0.5$;



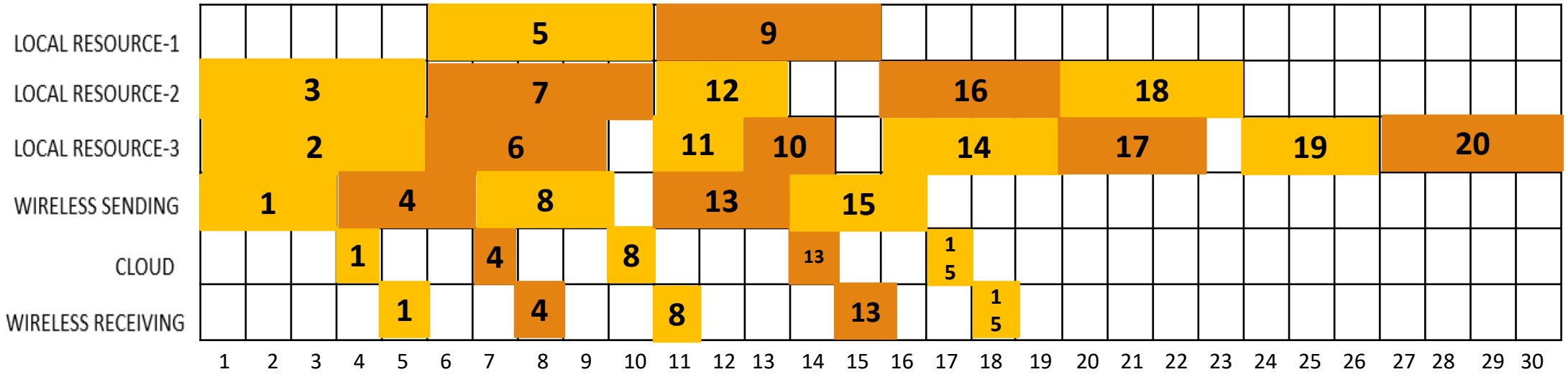
Time in each core	Localcore-1	Localcore-2	Localcore-3
Task 1	9	7	5
Task 2	8	6	5
Task 3	6	5	4
Task 4	7	5	3
Task 5	5	4	2
Task 6	7	6	4
Task 7	8	5	3
Task 8	6	4	2
Task 9	5	3	2
Task 10	7	4	2
Task 11	7	4	2
Task 12	6	3	2
Task 13	5	3	3
Task 14	8	7	4
Task 15	9	8	5
Task 16	6	4	2
Task 17	7	5	3
Task 18	8	4	2
Task 19	9	6	3
Task 20	7	6	4

Initial Scheduling

Time =30; Energy =167.5; Runtime =0 ms

```
Initial schedule:
Job2: Local Resource 3, start: 0, finish: 5
Job3: Local Resource 2, start: 0, finish: 5
Job1: Cloud Resource, start: 0, finish: 5
Job6: Local Resource 3, start: 5, finish: 9
Job7: Local Resource 2, start: 5, finish: 10
Job4: Cloud Resource, start: 3, finish: 8
Job5: Local Resource 1, start: 5, finish: 10
Job11: Local Resource 3, start: 10, finish: 12
Job8: Cloud Resource, start: 6, finish: 11
Job12: Local Resource 2, start: 10, finish: 13
Job10: Local Resource 3, start: 12, finish: 14
Job9: Local Resource 1, start: 10, finish: 15
Job13: Cloud Resource, start: 10, finish: 15
Job15: Cloud Resource, start: 13, finish: 18
Job14: Local Resource 3, start: 15, finish: 19
Job16: Local Resource 2, start: 15, finish: 19
Job17: Local Resource 3, start: 19, finish: 22
Job18: Local Resource 2, start: 19, finish: 23
Job19: Local Resource 3, start: 23, finish: 26
Job20: Local Resource 3, start: 26, finish: 30
Total energy: 167.5
Completion time: 30
Initial scheduling time: 0 ms
```

Output of the example after initial schedule:



E_1= (5+5)x1 = 10; E_2=(5+5+3+4+4)x2=42;

E_3=(5+4+2+2+4+3+3+4)x4=108;

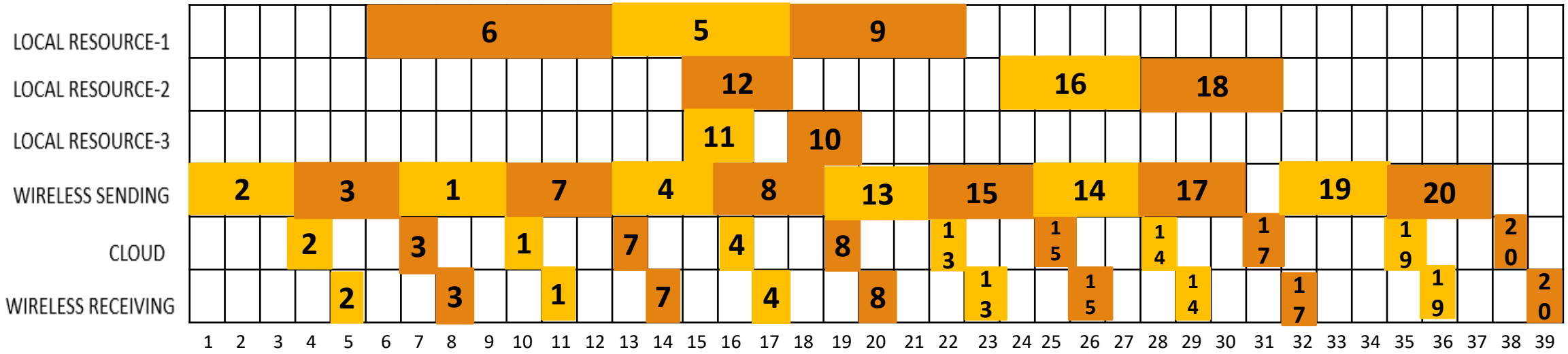
E_S=(3x0.5X5)=7.5;

E= 167.5

After Migrations:
Time = 39; Energy = 73; Run Time = 86 ms

```
After migrations:
Job2: Cloud Resource, start: 0, finish: 5
Job3: Cloud Resource, start: 3, finish: 8
Job1: Cloud Resource, start: 6, finish: 11
Job6: Local Resource 1, start: 5, finish: 12
Job7: Cloud Resource, start: 9, finish: 14
Job4: Cloud Resource, start: 12, finish: 17
Job5: Local Resource 1, start: 12, finish: 17
Job11: Local Resource 3, start: 14, finish: 16
Job8: Cloud Resource, start: 15, finish: 20
Job12: Local Resource 2, start: 14, finish: 17
Job10: Local Resource 3, start: 17, finish: 19
Job9: Local Resource 1, start: 17, finish: 22
Job13: Cloud Resource, start: 18, finish: 23
Job15: Cloud Resource, start: 21, finish: 26
Job14: Cloud Resource, start: 24, finish: 29
Job16: Local Resource 2, start: 23, finish: 27
Job17: Cloud Resource, start: 27, finish: 32
Job18: Local Resource 2, start: 27, finish: 31
Job19: Cloud Resource, start: 31, finish: 36
Job20: Cloud Resource, start: 34, finish: 39
Total energy: 73
Completion time: 39
Refinement time: 86 ms
```

Output of the example after Task Migration schedule



E_1= (7+5+5)x1 = 17; E_2=(3+4+4)x2=22;

E_3=(2+2)x4=16;

E_S=(3x0.5X12)=18;

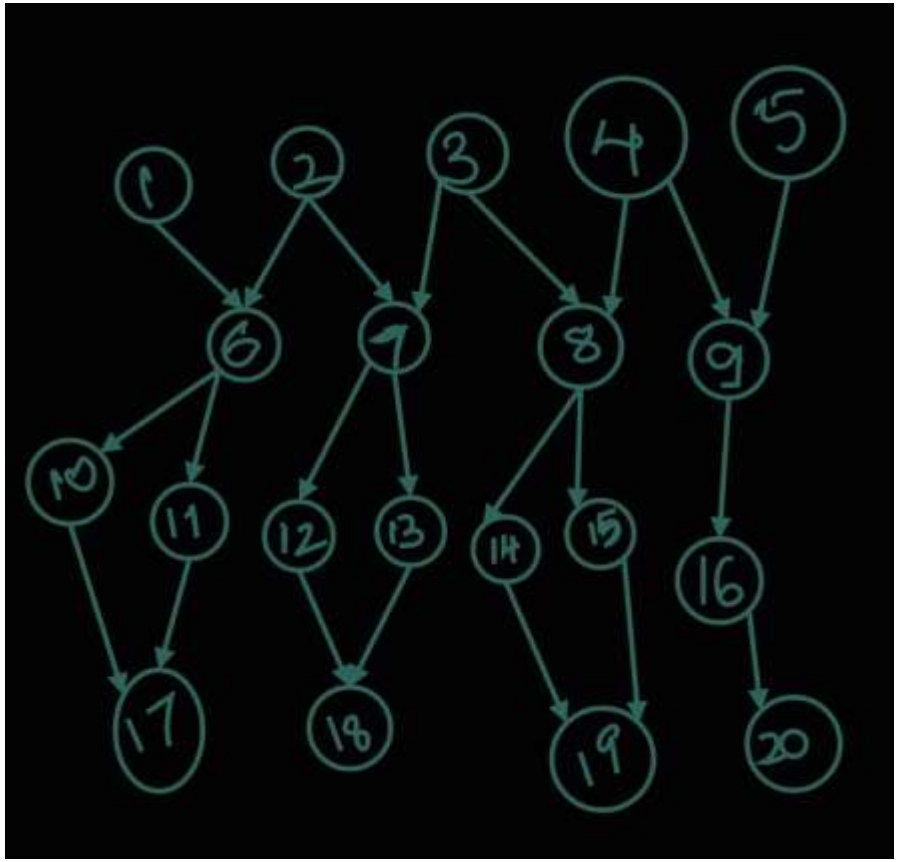
E= 73

EXAMPLE-4:

T_{max}=50

T_s=3; T_c=1; T_r=1;

P_1=1; P_2=2; P_3=4; P_s=0.5



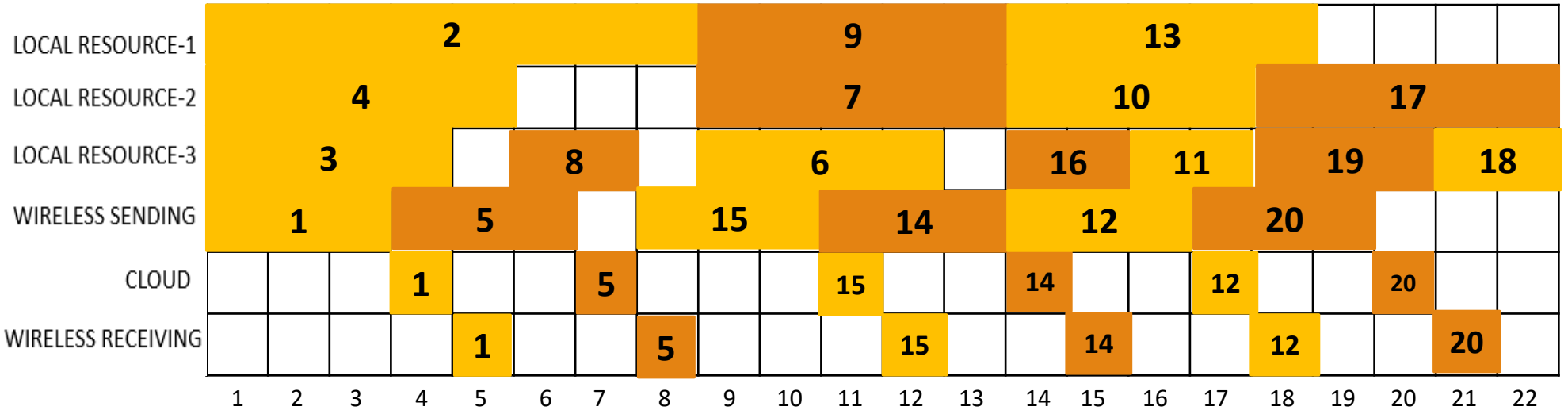
Time in each core	Localcore-1	Localcore-2	Localcore-3
Task 1	9	7	5
Task 2	8	6	5
Task 3	6	5	4
Task 4	7	5	3
Task 5	5	4	2
Task 6	7	6	4
Task 7	8	5	3
Task 8	6	4	2
Task 9	5	3	2
Task 10	7	4	2
Task 11	7	4	2
Task 12	6	3	2
Task 13	5	3	3
Task 14	8	7	4
Task 15	9	8	5
Task 16	6	4	2
Task 17	7	5	3
Task 18	8	4	2
Task 19	9	6	3
Task 20	7	6	4

Initial Scheduling

Time =22; Energy =141; Runtime =0 ms

```
Initial schedule:
Job3: Local Resource 3, start: 0, finish: 4
Job4: Local Resource 2, start: 0, finish: 5
Job1: Cloud Resource, start: 0, finish: 5
Job2: Local Resource 1, start: 0, finish: 8
Job8: Local Resource 3, start: 5, finish: 7
Job5: Cloud Resource, start: 3, finish: 8
Job6: Local Resource 3, start: 8, finish: 12
Job7: Local Resource 2, start: 8, finish: 13
Job15: Cloud Resource, start: 7, finish: 12
Job9: Local Resource 1, start: 8, finish: 13
Job14: Cloud Resource, start: 10, finish: 15
Job16: Local Resource 3, start: 13, finish: 15
Job10: Local Resource 2, start: 13, finish: 17
Job11: Local Resource 3, start: 15, finish: 17
Job12: Cloud Resource, start: 13, finish: 18
Job13: Local Resource 1, start: 13, finish: 18
Job19: Local Resource 3, start: 17, finish: 20
Job20: Cloud Resource, start: 16, finish: 21
Job17: Local Resource 2, start: 17, finish: 22
Job18: Local Resource 3, start: 20, finish: 22
Total energy: 141
Completion time: 22
Initial scheduling time: 0 ms
```

Output of the example after the initial scheduling



E_1= (8+5+5)x1 = 18;

E_2=(5+5+4+5)x2 = 38;

E_3=(4+2+4+2+2+3+2)x4 = 76;

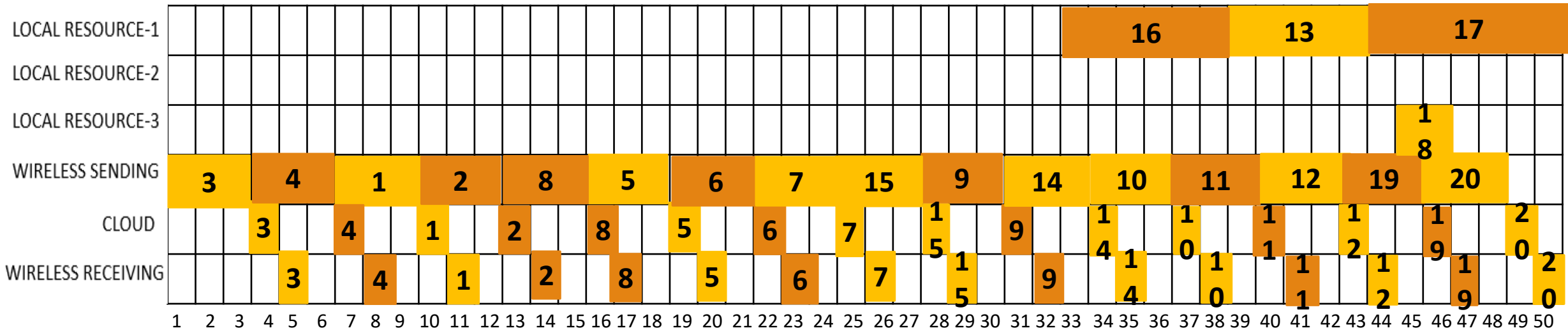
E_s=(3x0.5X6)=9;

E= E_1 + E_2 + E_3 + E_s = 141

After Migrations:
Time = 50; Energy = 50; Ref Time = 47 ms

```
After migrations:
Job3: Cloud Resource, start: 0, finish: 5
Job4: Cloud Resource, start: 3, finish: 8
Job1: Cloud Resource, start: 6, finish: 11
Job2: Cloud Resource, start: 9, finish: 14
Job8: Cloud Resource, start: 12, finish: 17
Job5: Cloud Resource, start: 15, finish: 20
Job6: Cloud Resource, start: 18, finish: 23
Job7: Cloud Resource, start: 21, finish: 26
Job15: Cloud Resource, start: 24, finish: 29
Job9: Cloud Resource, start: 27, finish: 32
Job14: Cloud Resource, start: 30, finish: 35
Job16: Local Resource 1, start: 32, finish: 38
Job10: Cloud Resource, start: 33, finish: 38
Job11: Cloud Resource, start: 36, finish: 41
Job12: Cloud Resource, start: 39, finish: 44
Job13: Local Resource 1, start: 38, finish: 43
Job19: Cloud Resource, start: 42, finish: 47
Job20: Cloud Resource, start: 45, finish: 50
Job17: Local Resource 1, start: 43, finish: 50
Job18: Local Resource 3, start: 44, finish: 46
Total energy: 50
Completion time: 50
Refinement time: 47 ms
```

Output of the example after Task Migration schedule



E_1= (6+5+7)x1 = 18;

E_2= 0;

E_3=2x4 = 8;

E_s=(3x0.5X16)=24;

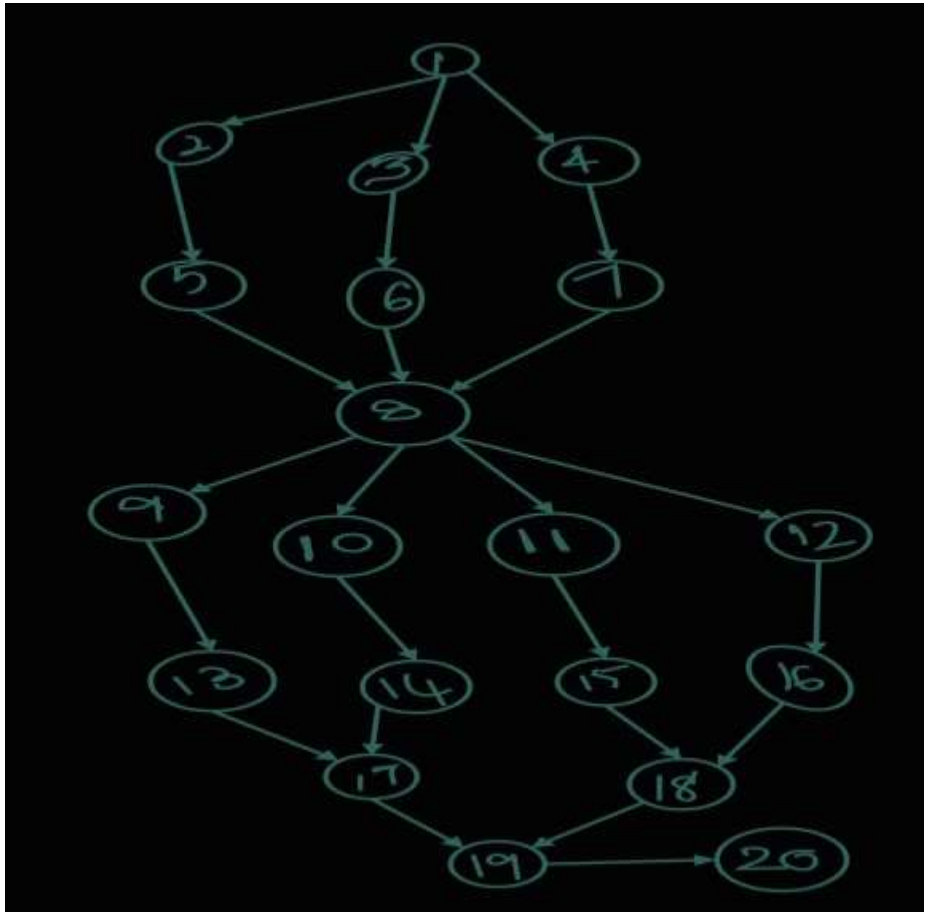
E= E_1 + E_2 + E_3 + E_s = 50

EXAMPLE 5:

T_{max}= 40

T_s=3; T_c=1; T_r=1;

P_1=1; P_2=2; P_3=4; P_s=0.5



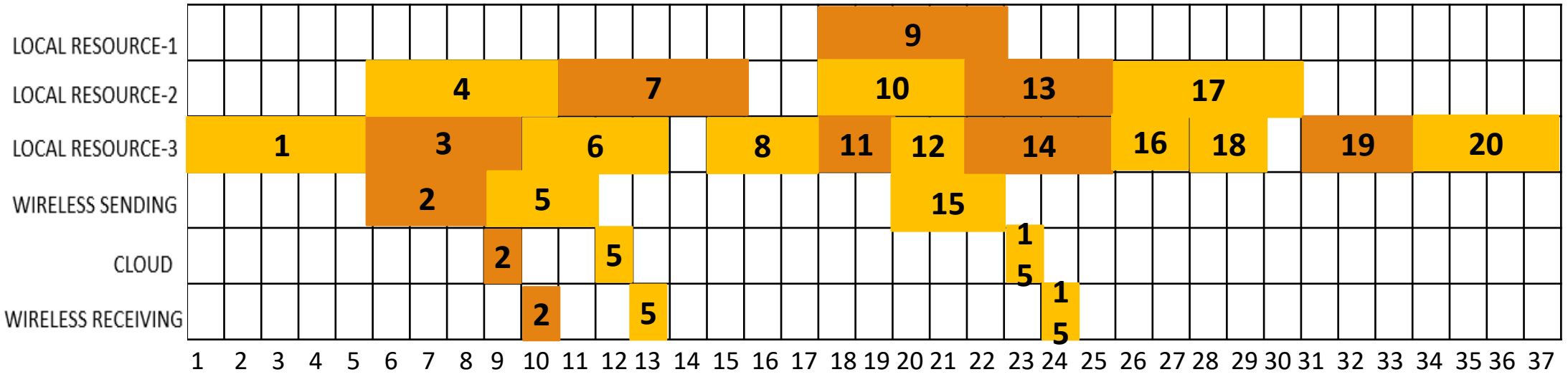
Time in each core	Localcore-1	Localcore-2	Localcore-3
Task 1	9	7	5
Task 2	8	6	5
Task 3	6	5	4
Task 4	7	5	3
Task 5	5	4	2
Task 6	7	6	4
Task 7	8	5	3
Task 8	6	4	2
Task 9	5	3	2
Task 10	7	4	2
Task 11	7	4	2
Task 12	6	3	2
Task 13	5	3	3
Task 14	8	7	4
Task 15	9	8	5
Task 16	6	4	2
Task 17	7	5	3
Task 18	8	4	2
Task 19	9	6	3
Task 20	7	6	4

Initial Scheduling

Time =37; Energy =189.5; Runtime = 0 ms

```
Initial schedule:
Job1: Local Resource 3, start: 0, finish: 5
Job3: Local Resource 3, start: 5, finish: 9
Job4: Local Resource 2, start: 5, finish: 10
Job2: Cloud Resource, start: 5, finish: 10
Job6: Local Resource 3, start: 9, finish: 13
Job7: Local Resource 2, start: 10, finish: 15
Job5: Cloud Resource, start: 8, finish: 13
Job8: Local Resource 3, start: 15, finish: 17
Job11: Local Resource 3, start: 17, finish: 19
Job10: Local Resource 2, start: 17, finish: 21
Job12: Local Resource 3, start: 19, finish: 21
Job9: Local Resource 1, start: 17, finish: 22
Job15: Cloud Resource, start: 19, finish: 24
Job14: Local Resource 3, start: 21, finish: 25
Job13: Local Resource 2, start: 22, finish: 25
Job16: Local Resource 3, start: 25, finish: 27
Job17: Local Resource 2, start: 25, finish: 30
Job18: Local Resource 3, start: 27, finish: 29
Job19: Local Resource 3, start: 30, finish: 33
Job20: Local Resource 3, start: 33, finish: 37
Total energy: 189.5
Completion time: 37
Initial scheduling time: 0 ms
```

Output of the example after the initial scheduling



E_1= 5 x1 = 5;

E_2=(5+5+4+3+5)x2 = 44;

E_3=(5+4+4+2+2+4+2+2+3+4+2)x4 = 136;

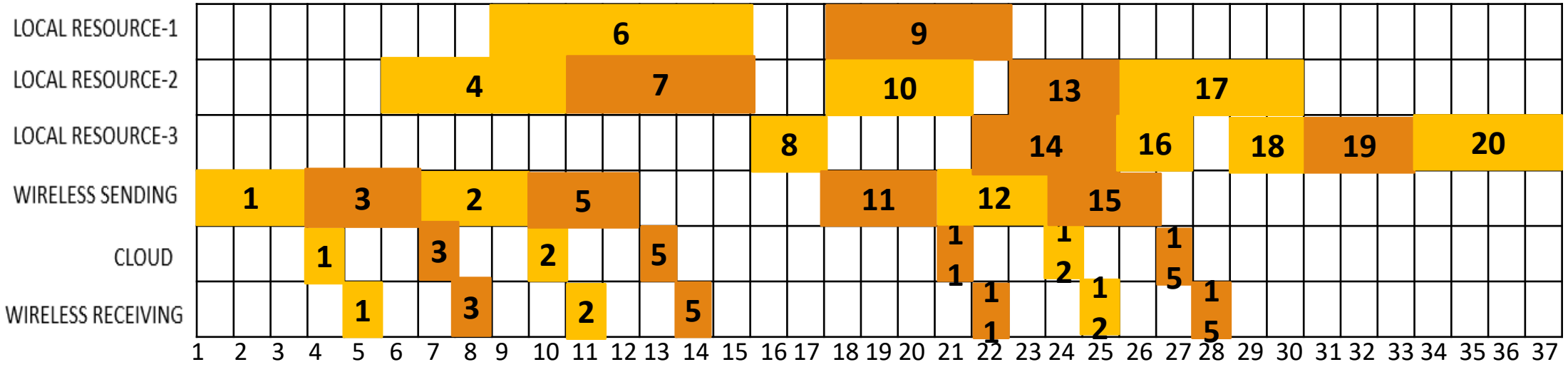
E_s=(3x0.5X3)=4.5;

E= E_1 + E_2 + E_3 + E_s = 189.5

After Migrations:
Time = 37; Energy = 134.5; Ref Time = 26 ms

```
Initial scheduling time: 0 ms
After migrations:
Job1: Cloud Resource, start: 0, finish: 5
Job3: Cloud Resource, start: 3, finish: 8
Job4: Local Resource 2, start: 5, finish: 10
Job2: Cloud Resource, start: 6, finish: 11
Job6: Local Resource 1, start: 8, finish: 15
Job7: Local Resource 2, start: 10, finish: 15
Job5: Cloud Resource, start: 9, finish: 14
Job8: Local Resource 3, start: 15, finish: 17
Job11: Cloud Resource, start: 17, finish: 22
Job10: Local Resource 2, start: 17, finish: 21
Job12: Cloud Resource, start: 20, finish: 25
Job9: Local Resource 1, start: 17, finish: 22
Job15: Cloud Resource, start: 23, finish: 28
Job14: Local Resource 3, start: 21, finish: 25
Job13: Local Resource 2, start: 22, finish: 25
Job16: Local Resource 3, start: 25, finish: 27
Job17: Local Resource 2, start: 25, finish: 30
Job18: Local Resource 3, start: 28, finish: 30
Job19: Local Resource 3, start: 30, finish: 33
Job20: Local Resource 3, start: 33, finish: 37
Total energy: 134.5
Completion time: 37
Refinement time: 26 ms
```

Output of the example after Task Migration schedule



E_1= (7+5)x1 = 12;

E_2= (5+5+4+3+5) x 2 = 44;

E_3=(2+4+2+2+3+4)x4 = 68;

E_s=(3x0.5X7)=10.5;

E= E_1 + E_2 + E_3 + E_s = 134.5