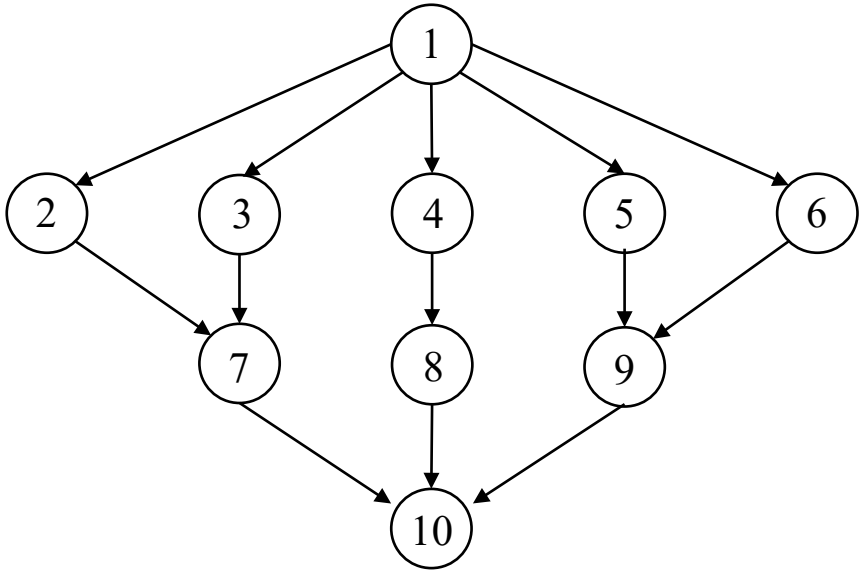
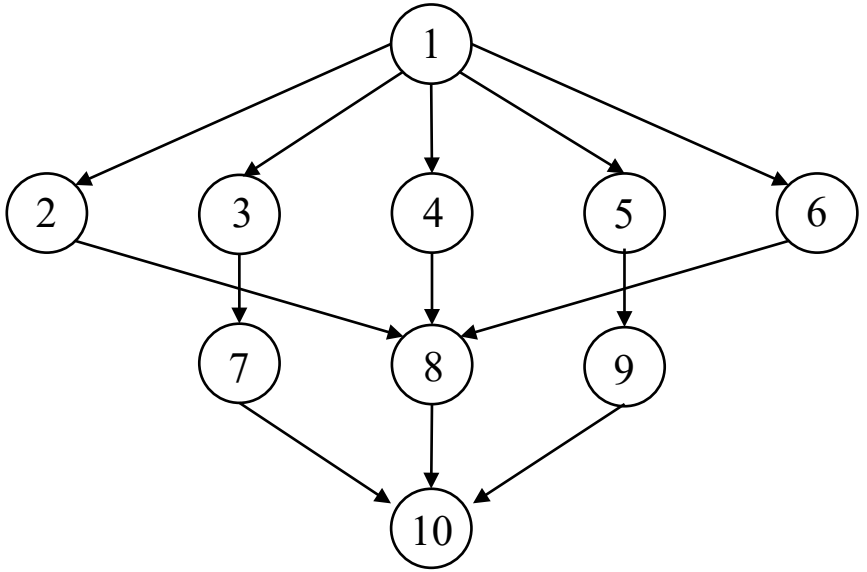
Energy and Performance-Aware Task Scheduling in a Mobile Cloud Computing Environment

Qiulin Luo 002191479



Example 1

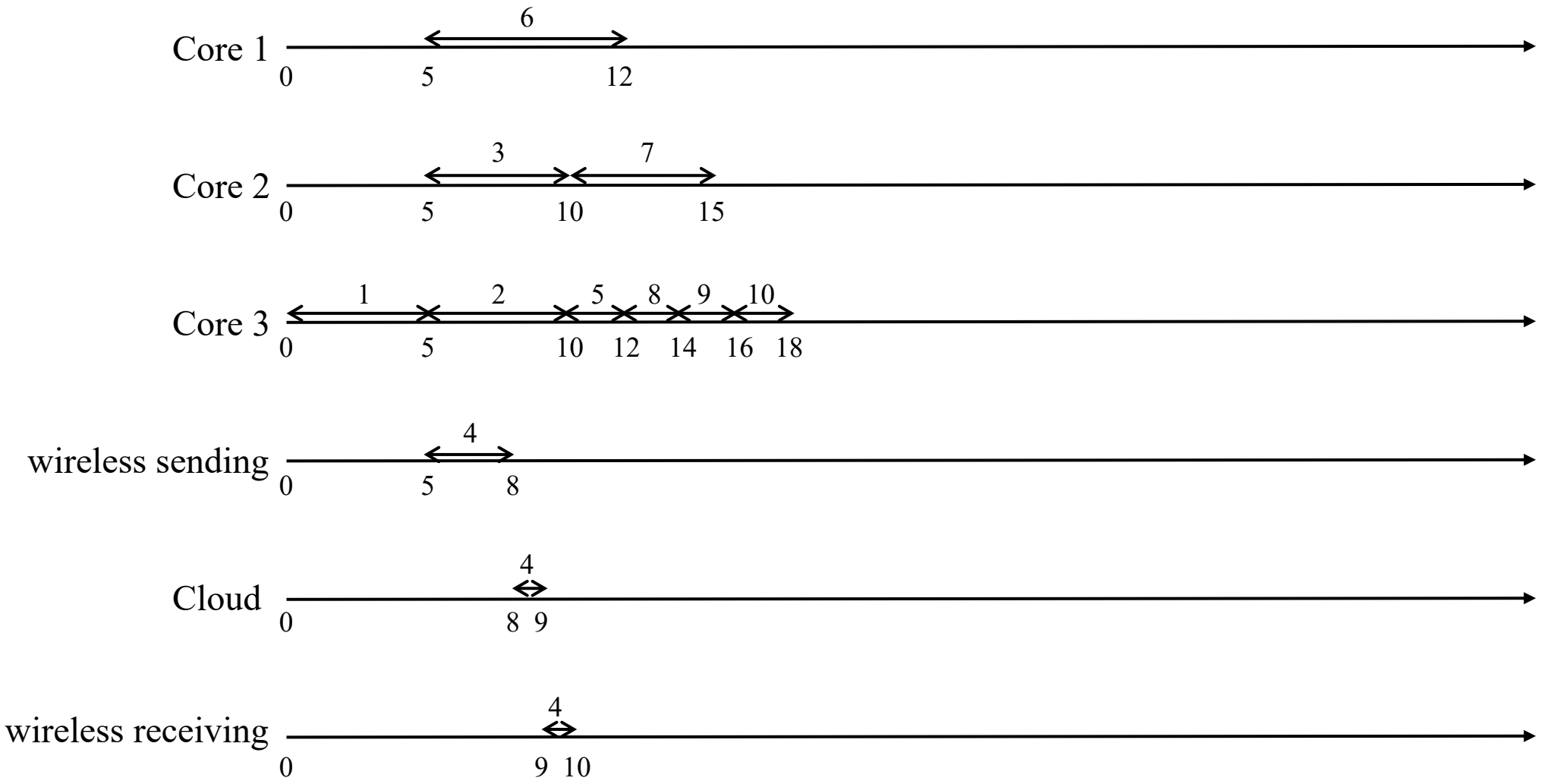


Example 2

Example 1

E: 100.5 T: 18

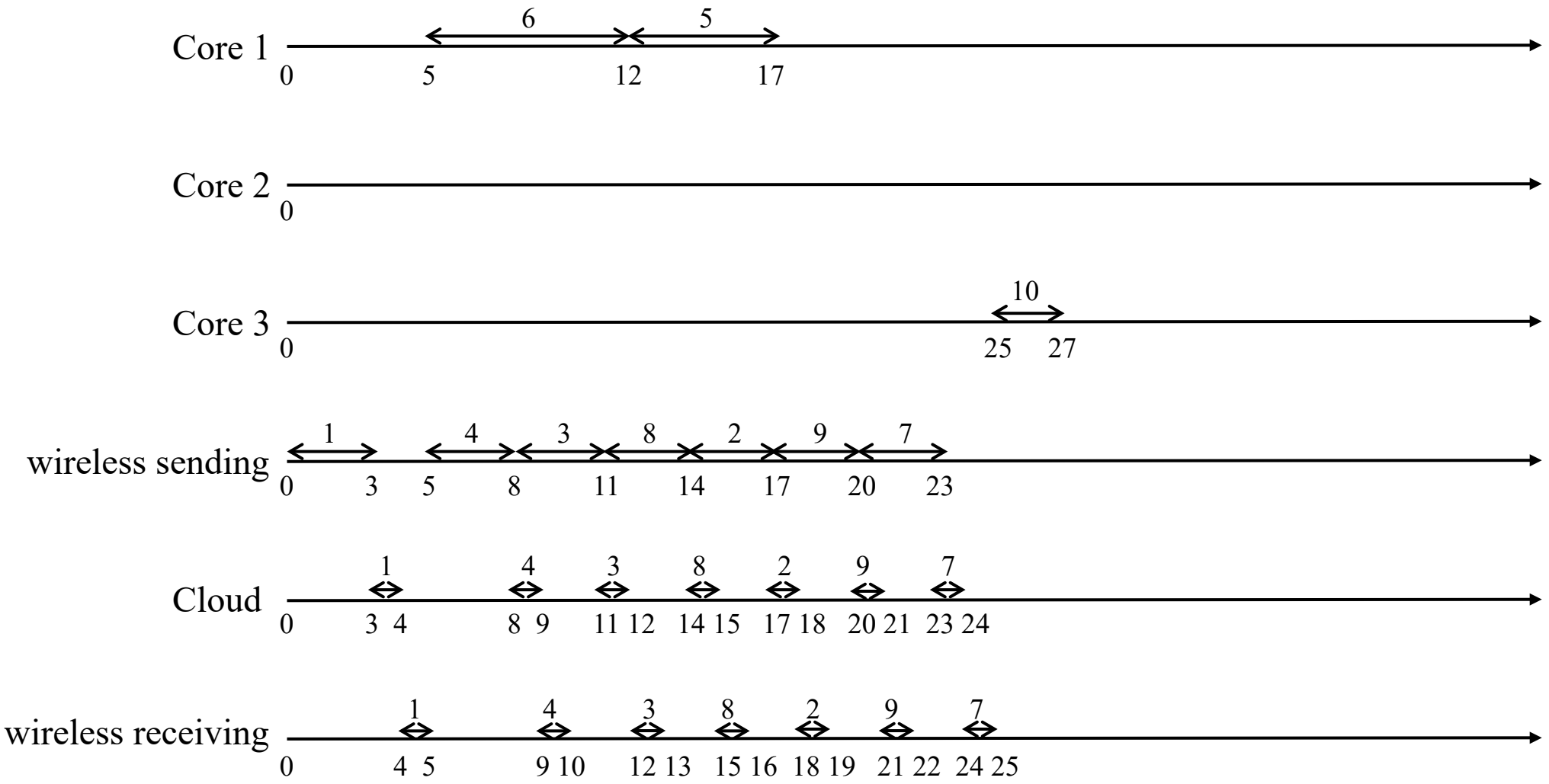
Initial Scheduling



Example 1

E: 27 T: 27

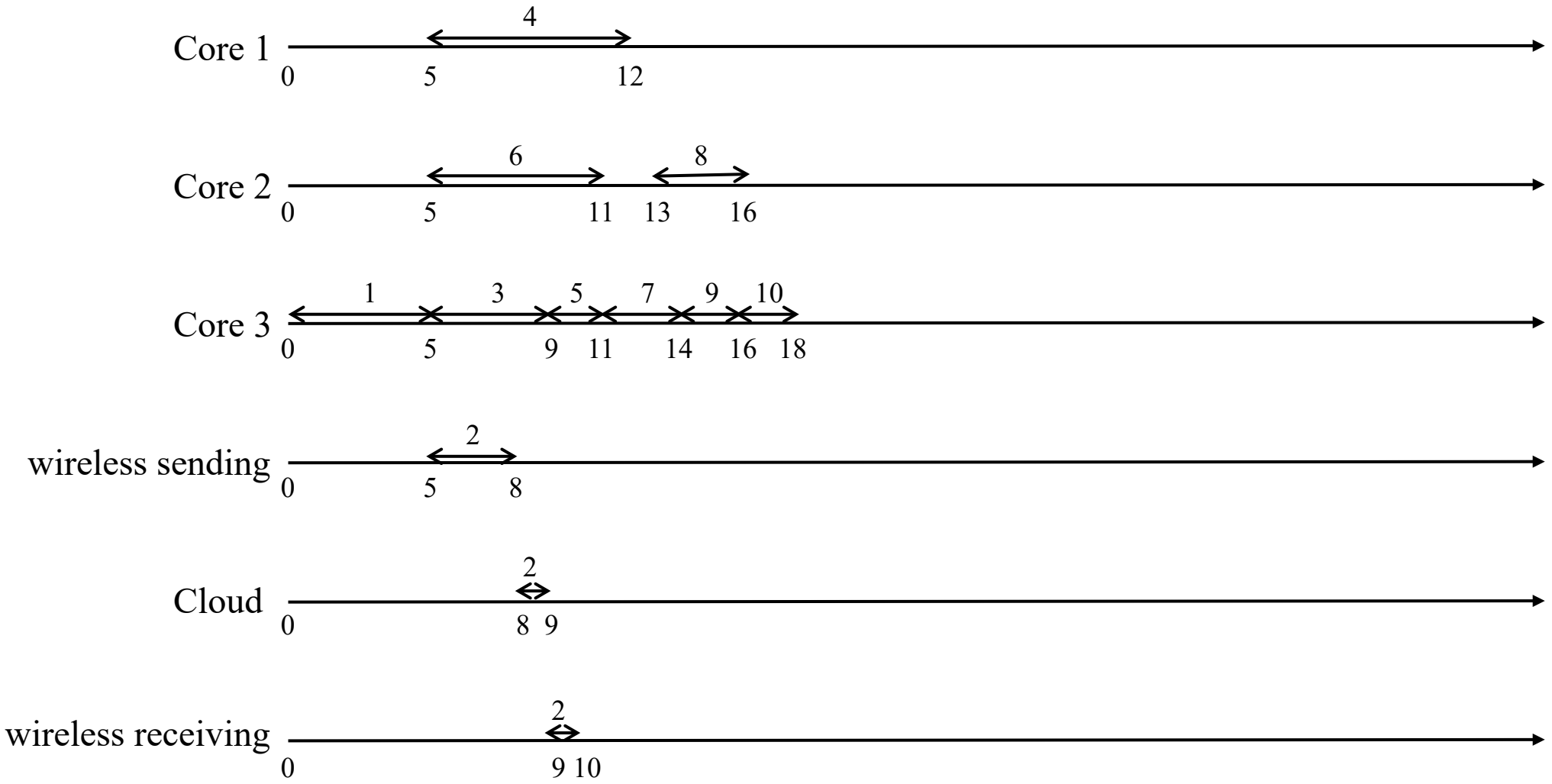
Final Scheduling



Example 2

E: 100.5 T: 18

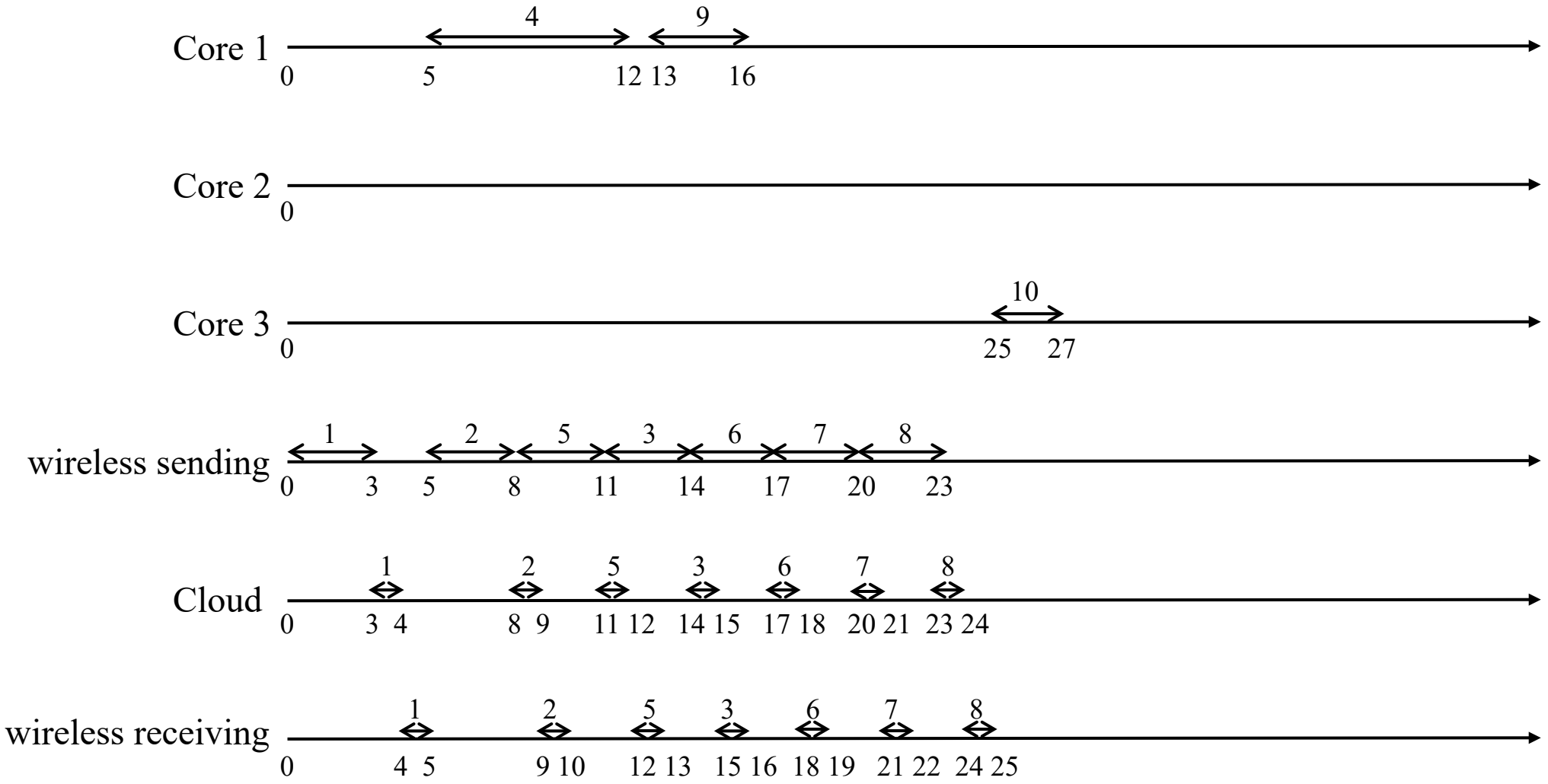
Initial Scheduling



Example 2

E: 27 T: 27

Final Scheduling



```
#include <windows.h>
#include <vector>
#include <iterator>
#include <cstdlib>
#include <stdio.h>
#include <iostream>
#include <stack>
#include <algorithm>
#include <stdlib.h>

using namespace std;

struct System_Initialization{
	int local_cores_number;
	int task_number;
	int cloud_sending_time;
	int cloud_computing_time;
	int cloud_sendingBack_time;
	int power_coefficient_local[3];
	float power_coefficient_cloud;
}thisSystem = {3, 10, 3, 1, 1, {1, 2, 4}, 0.5};

int * spilt_core(int t, int array1[10], int array2[10]){
	for(int k = 0; k < sizeof(array2); k++){
		if(array1[k] - t > 0) array2[k] = 1;
		else array2[k] = 0;
	}
	return array2;
}

int * priority_initialization(int prep[10], int avg[10], int a_m[10][10]){
	prep[thisSystem.task_number - 1] = avg[thisSystem.task_number - 1];
	for(int i = thisSystem.task_number - 1; i >= 0; i--){
		int flag = 0;
		for(int j = thisSystem.task_number - 1; j >= 0; j--){
			if(prep[j] > flag){
				if(a_m[i][j] == 1){
					flag = prep[j];
				}
			}
		}
		prep[i] = avg[i] + flag;
	}
	return prep;
```

```
}

int * array_obtain_element(int arrayA[10], int arrayB[10]){
	for(int k = 0; k < thisSystem.task_number; k++){
		arrayA[k] = arrayB[k];
	}
	return arrayA;
}

int task_insertion(int which_task, int which_core, int task_graph_adjacent_matrix[10][10], int total_finish_time_2[10]){
	int current_ready_time;
	if(which_core != thisSystem.local_cores_number){
		int compared_time = 0;
		for(int r = 0; r < thisSystem.task_number; r++){
			if(task_graph_adjacent_matrix[r][which_task] - 1 == 0){
				if(compared_time < total_finish_time_2[which_task]){
					compared_time = total_finish_time_2[which_task];
				}
			}
		}
		current_ready_time = compared_time;
	}
	else{
		int compared_sending_time = 0;
		for(int r = 0; r < thisSystem.task_number; r++){
			if(task_graph_adjacent_matrix[r][which_task] - 1 == 0){
				if(compared_sending_time < total_finish_time_2[r]){
				compared_sending_time = total_finish_time_2[r];
				}
			}
		}
		current_ready_time = compared_sending_time;
	}
	return current_ready_time;
}

int trace_insertion(vector<vector<int>> trace, int which_task, int which_core, int ready_time[10], int ready_time_2[10]){
	int insert_location;
	if(trace[which_core].size() > 1){
		if(ready_time_2[trace[which_core][0]] - ready_time[which_task] > 0){
			insert_location = 0;
		}
```

```
            else if(ready_time_2[trace[which_core][trace[which_core].size() - 1]] <= ready_time[which_task]){
                        insert_location = trace[which_core].size();
                    }
            else{
                for(int r = 0; r < trace[which_core].size() - 1; r++){
                    if(ready_time[which_task] >= ready_time_2[trace[which_core][r]]){
                        if(ready_time[which_task] <= ready_time_2[trace[which_core][r + 1]]){
                            insert_location = r + 1;
                        }
                    }
                }
            }
        }
        if(trace[which_core].size() <= 1){
            switch (trace[which_core].size()){
            case 0:
                insert_location = 0;
                break;
            case 1:
                if(ready_time_2[trace[which_core][0]] > ready_time[which_task]){
                    insert_location = 0;
                }
                else{
                    insert_location = 1;
                }
                break;
            }
        }
        return insert_location;
}

void calculate_target_time(int on_what_core[10], int core_finish_time[4], int local_cores_processing_time[10][3], int ready_time[10], int total_finish_time[10], int current_task, int compare1, int compare2){
        if(on_what_core[current_task] == thisSystem.local_cores_number){
            ready_time[current_task] = max(compare1,compare2);
            total_finish_time[current_task] = ready_time[current_task] + thisSystem.cloud_sending_time + thisSystem.cloud_computing_time + thisSystem.cloud_sendingBack_time;
            core_finish_time[on_what_core[current_task]] = ready_time[current_task] + thisSystem.cloud_sending_time;
```

```
	}
	else{
		ready_time[current_task] = max(compare1,compare2);
		total_finish_time[current_task] = ready_time[current_task] + local_cores_processing_time[current_task][on_what_core[current_task]];
		core_finish_time[on_what_core[current_task]] = total_finish_time[current_task];
	}
}

void operate_stack(stack<int> task_stack, vector<vector<int>> trace, int record_1[10], int record_2[10], int stack_record[10], int on_what_core[10], int local_cores_processing_time[10][3], int task_graph_adjacent_matrix[10][10], int core_finish_time[4], int total_finish_time[10], int ready_time[10]){
	for(;;){
		int task_now = task_stack.top();
		task_stack.pop();
		if(on_what_core[task_now] == thisSystem.local_cores_number){
			int compared_sending_time = 0;
			for(int r = 0; r < thisSystem.task_number; r++){
				if(task_graph_adjacent_matrix[r][task_now] == 1){
					if(compared_sending_time < total_finish_time[r]){
						compared_sending_time = total_finish_time[r];
					}
				}
			}
			ready_time[task_now] = compared_sending_time;
		}
		else{
			int compared_local_time = 0;
			for(int r = 0; r < thisSystem.task_number; r++){
				if(task_graph_adjacent_matrix[r][task_now] == 1){
					if(compared_local_time < total_finish_time[r]){
						compared_local_time = total_finish_time[r];
					}
				}
			}
			ready_time[task_now] = compared_local_time;
		}
		if(on_what_core[task_now] == thisSystem.local_cores_number){
			ready_time[task_now] = max(core_finish_time[on_what_core[task_now]], ready_time[task_now]);
			total_finish_time[task_now] = ready_time[task_now] + thisSystem.cloud_sending_time + thisSystem.cloud_computing_time +
```

```
thisSystem.cloud_sendingBack_time;
                core_finish_time[on_what_core[task_now]] = ready_time[task_now] +
thisSystem.cloud_sending_time;
            }
            else{
                ready_time[task_now]                                              =
max(core_finish_time[on_what_core[task_now]],ready_time[task_now]);
                total_finish_time[task_now]      =      ready_time[task_now]      +
local_cores_processing_time[task_now][on_what_core[task_now]];
                core_finish_time[on_what_core[task_now]]                          =
total_finish_time[task_now];
            }
            for(int r = 0; r < thisSystem.task_number; r++){
                if(task_graph_adjacent_matrix[task_now][r] == 1){
                    record_1[r] = record_1[r] - 1;
                }
            }
            record_2[task_now] = 1;
            if(trace[on_what_core[task_now]].size() > 1){
                for(int r = 1; r < trace[on_what_core[task_now]].size(); r++){
                    if(trace[on_what_core[task_now]][r - 1] == task_now){
                        record_2[trace[on_what_core[task_now]][r]] = 0;
                    }
                }
            }
            for(int r = 0; r < thisSystem.task_number; r++){
                if(record_1[r] == 0){
                    if(record_2[r] == 0){
                        if(stack_record[r] == 0){
                            task_stack.push(r);
                            stack_record[r] = 1;
                        }
                    }
                }
            }
            if(task_stack.size() <= 0){
                break;
            }
        }
}

void transitive(int initial_core_location[10], int settled_core_location[10], int
initial_start_time[10], int settled_start_time[10], int initial_finish_time[10], int
settled_finish_time[10]){
```

```
    for(int r = 0; r < thisSystem.task_number; r++){
        settled_core_location[r] =initial_core_location[r];
        settled_start_time[r] = initial_start_time[r];
        settled_finish_time[r] = initial_finish_time[r];
    }
}

void print_result(vector<vector<int>> result_set, int start[10], int end[10], float final_E,
float final_T){
    printf("\nResult:\n");
    printf("Cost of Energy: %.2f Cost of Time: %.2f", final_E, final_T);
    for(int i = 0; i < result_set.size(); i++){
        if(i == 3) printf("\n3 -> ");
        else printf("\n%d -> ", i);
        for(int j = 0; j < result_set[i].size(); j++){
            printf("( %d %d %d ) ",start[result_set[i][j]], result_set[i][j] + 1,
end[result_set[i][j]]);
        }
    }
    printf("\n");
}

void Phase_One(int local_cores_processing_time[10][3], int cloud_or_local[10]){
    int Time_Max_Cloud = thisSystem.cloud_sending_time +
thisSystem.cloud_computing_time + thisSystem.cloud_sendingBack_time;
    int Inf_Local[10] = {0};
    for(int i = 0; i < thisSystem.task_number; ++i){
        for(int j = 0; j < thisSystem.local_cores_number; ++j){
            if(Inf_Local[i] > local_cores_processing_time[i][j]) Inf_Local[i] =
local_cores_processing_time[i][j];
        }
    }
    int * CoL;
    CoL = spilt_core(Time_Max_Cloud, Inf_Local, cloud_or_local);
    for(int q = 0; q < thisSystem.task_number; q++){
        cloud_or_local[q] = * (CoL + q);
    }
}

void Phase_Two(int local_cores_processing_time[10][3], int priority[10], int
sorted_priority_index[10], int task_graph_adjacent_matrix[10][10], int
average_time[10], int cloud_or_local[10]){
    for(int i = 0; i < thisSystem.task_number; ++i){
        switch (cloud_or_local[i]){
```

```
            case 1:
                average_time[i] = thisSystem.cloud_sending_time +
thisSystem.cloud_computing_time + thisSystem.cloud_sendingBack_time;
                break;
            case 0:
                int total = 0;
                for(int k = 0; k < thisSystem.local_cores_number; ++k) total = total +
local_cores_processing_time[i][k];
                average_time[i] = total / thisSystem.local_cores_number;
                break;
            }
        }
        int * prep_tmp;
        prep_tmp =
priority_initialization(priority,average_time,task_graph_adjacent_matrix);
        for(int q = 0; q < thisSystem.task_number; q++){
            priority[q] = * (prep_tmp + q);
        }
        vector<pair<int,int>> key_value_pair;
        for (int r = 0; r < thisSystem.task_number; r++)
key_value_pair.push_back(make_pair(priority[r],r));
        sort(key_value_pair.begin(), key_value_pair.end());
        for(int p = 0; p < thisSystem.task_number; p++) sorted_priority_index[p] =
key_value_pair[p].second;
}

void Phase_Three(vector<vector<int>> trace, int local_cores_processing_time[10][3],
int sorted_priority_index[10], int task_graph_adjacent_matrix[10][10], int
cloud_or_local[10], int ready_time_local[10], int ready_time_cloud_computing[10],
int ready_time_cloud_sending[10], int finish_time_cloud_sending[10], int
finish_time_cloud_return[10], int finish_time_local[10], int total_finish_time[10], int
core_finish_time[4], int on_what_core[10]){
        ready_time_local[sorted_priority_index[9]] = 0;
        ready_time_cloud_sending[sorted_priority_index[9]] = 0;
        finish_time_cloud_sending[sorted_priority_index[9]] =
ready_time_cloud_sending[sorted_priority_index[9]] +
thisSystem.cloud_sending_time;
        ready_time_cloud_computing[sorted_priority_index[9]] =
finish_time_cloud_sending[sorted_priority_index[9]];
        switch (cloud_or_local[sorted_priority_index[9]]){
        case 1:
            finish_time_cloud_return[sorted_priority_index[9]] =
ready_time_cloud_computing[sorted_priority_index[9]] +
thisSystem.cloud_computing_time + thisSystem.cloud_sendingBack_time;
```

```
            finish_time_local[sorted_priority_index[9]] = 0;
            total_finish_time[sorted_priority_index[9]] =
finish_time_cloud_return[sorted_priority_index[9]];
            core_finish_time[3] = total_finish_time[sorted_priority_index[9]];
            trace[0].push_back(sorted_priority_index[9]);
            on_what_core[sorted_priority_index[9]] = thisSystem.local_cores_number;
            break;
        case 0:
            int tmp;
            int Inf_Local = 100000;
            for(int i = 0; i < thisSystem.local_cores_number; i++){
                if(local_cores_processing_time[sorted_priority_index[9]][i] -
Inf_Local < 0){
                    Inf_Local =
local_cores_processing_time[sorted_priority_index[9]][i];
                    tmp = i;
                }
            }
            finish_time_local[sorted_priority_index[9]] =
ready_time_local[sorted_priority_index[9]] + Inf_Local;
            finish_time_cloud_return[sorted_priority_index[9]] =
ready_time_cloud_computing[sorted_priority_index[9]] +
thisSystem.cloud_computing_time + thisSystem.cloud_sendingBack_time;
            if(finish_time_local[sorted_priority_index[9]] -
finish_time_cloud_return[sorted_priority_index[9]] <= 0){
                total_finish_time[sorted_priority_index[9]] =
finish_time_local[sorted_priority_index[9]];
                finish_time_cloud_return[sorted_priority_index[9]] = 0;
                core_finish_time[3] =
finish_time_cloud_sending[sorted_priority_index[9]];
                trace[tmp + 1].push_back(sorted_priority_index[9]);
                on_what_core[sorted_priority_index[9]] = tmp;
            }
            else{
                total_finish_time[sorted_priority_index[9]] =
finish_time_cloud_return[sorted_priority_index[9]];
                finish_time_local[sorted_priority_index[9]] = 0;
                core_finish_time[tmp] = total_finish_time[sorted_priority_index[9]];
                trace[0].push_back(sorted_priority_index[9]);
                on_what_core[sorted_priority_index[9]] = 3;
            }
            break;
        }
```

```
    for(int task = thisSystem.task_number - 2; task >= 0; task--){
        int compared_time = 0;
        for(int j = 0; j < thisSystem.task_number; j++){
            if(task_graph_adjacent_matrix[j][sorted_priority_index[task]] == 1){
                if(compared_time < max(finish_time_local[j],finish_time_cloud_return[j])){
                    compared_time = max(finish_time_local[j],finish_time_cloud_return[j]);
                }
            }
        }
        ready_time_local[sorted_priority_index[task]] = compared_time;
        int compared_sending_time = 0;
        for(int j = 0; j < thisSystem.task_number; j++){
            if(task_graph_adjacent_matrix[j][sorted_priority_index[task]] == 1){
                if(compared_sending_time < max(finish_time_local[j],finish_time_cloud_sending[j])){
                    compared_sending_time = max(finish_time_local[j],finish_time_cloud_sending[j]);
                }
            }
        }
        ready_time_cloud_sending[sorted_priority_index[task]] = compared_sending_time;
        finish_time_cloud_sending[sorted_priority_index[task]] = max(core_finish_time[3],ready_time_cloud_sending[sorted_priority_index[task]]) + thisSystem.cloud_sending_time;
        int compared_computing_time = 0;
        for(int j = 0; j < thisSystem.task_number; j++){
            if(task_graph_adjacent_matrix[j][sorted_priority_index[task]] == 1){
                if(compared_computing_time < finish_time_cloud_return[j] - 1){
                    compared_computing_time = finish_time_cloud_return[j] - 1;
                }
            }
        }
        ready_time_cloud_computing[sorted_priority_index[task]] = max(finish_time_cloud_sending[sorted_priority_index[task]],compared_computing_time);
        switch (cloud_or_local[sorted_priority_index[task]]){
        case 1:
            finish_time_cloud_return[sorted_priority_index[task]] = ready_time_cloud_computing[sorted_priority_index[task]] + thisSystem.cloud_computing_time + thisSystem.cloud_sendingBack_time;
            total_finish_time[sorted_priority_index[task]] =
```

```
finish_time_cloud_return[sorted_priority_index[task]];
                finish_time_local[sorted_priority_index[task]] = 0;
                core_finish_time[3] =
finish_time_cloud_sending[sorted_priority_index[task]];
                trace[0].push_back(sorted_priority_index[task]);
                on_what_core[sorted_priority_index[task]] = 3;
                break;
            case 0:
                int tmp;
                int Ini_Local = 100000;
                int ready_time;
                for(int j = 0; j < thisSystem.local_cores_number; j++){
                    ready_time =
max(ready_time_local[sorted_priority_index[task]],core_finish_time[j]);
                    if(Ini_Local - (ready_time +
local_cores_processing_time[sorted_priority_index[task]][j]) > 0){
                        Ini_Local = ready_time +
local_cores_processing_time[sorted_priority_index[task]][j];
                        tmp = j;
                    }
                }
                ready_time_local[sorted_priority_index[task]] = Ini_Local -
local_cores_processing_time[sorted_priority_index[task]][tmp];
                finish_time_local[sorted_priority_index[task]] = Ini_Local;
                finish_time_cloud_return[sorted_priority_index[task]] =
ready_time_cloud_computing[sorted_priority_index[task]] + 2;
                if(finish_time_local[sorted_priority_index[task]] -
finish_time_cloud_return[sorted_priority_index[task]] <= 0){
                    total_finish_time[sorted_priority_index[task]] =
finish_time_local[sorted_priority_index[task]];
                    finish_time_cloud_return[sorted_priority_index[task]] = 0;
                    core_finish_time[tmp] =
total_finish_time[sorted_priority_index[task]];
                    trace[tmp+1].push_back(sorted_priority_index[task]);
                    on_what_core[sorted_priority_index[task]] = tmp;
                }
                else{
                    total_finish_time[sorted_priority_index[task]] =
finish_time_cloud_return[sorted_priority_index[task]];
                    finish_time_local[sorted_priority_index[task]] = 0;
                    core_finish_time[3] =
total_finish_time[sorted_priority_index[task]];
                    trace[0].push_back(sorted_priority_index[task]);
                    on_what_core[sorted_priority_index[task]] = 3;
```

```
                }
                break;
            }
        }
}

void Reschedule(vector<vector<int>> trace, int local_cores_processing_time[10][3], int flag[10], int flag_2[10], int task_graph_adjacent_matrix[10][10], int on_what_core[10], int time_maximum, int total_time, float total_energy, int start_time[10], int end_time[10], int Energy_Cost_Cloud, float Energy_Cost_Local[10][3]){
    for(;;){
        double deduction_rate = 0;
        float dynamic_total_time = total_time;
        float dynamic_energy = total_energy;
        int flag1 = 0;
        int flag2 = 0;
        int target_task = 0;
        int target_core = 0;
        int insert_location_1 = 0;
        int insert_location_2 = 0;
        int on_what_core_3[10] = {0};
        int target_start_time[10] = {0};
        int dynamic_target_finish_time[10] = {0};

        for(int i = 0; i < thisSystem.task_number; i++){
            for(int j = 0; j < thisSystem.local_cores_number + 1; j++){
                int displacement_1 = 0;
                int displacement_2 = 0;
                int on_what_core_2[10] = {0};
                int core_finish_time_2[4] = {0};
                int ready_time[10] = {0};
                int ready_time_2[10] = {0};
                int total_finish_time[10] = {0};
                int total_finish_time_2[10] = {0};
                int stack_record[10] = {0};
                int current_core = on_what_core[i];
                int record_1[10] = {0};
                int record_2[10] = {0};
                stack<int> task_stack;
                vector<vector<int>> trace_2(4);

                int * obtain1, * obtain2, * obtain3;
                obtain1 = array_obtain_element(on_what_core_2,on_what_core);
```

```
                    for(int g = 0; g < thisSystem.task_number; g++){
                        on_what_core_2[g] = * (obtain1 + g);
                    }
                    obtain2 = array_obtain_element(total_finish_time_2,end_time);
                    for(int g = 0; g < thisSystem.task_number; g++){
                        total_finish_time_2[g] = * (obtain2 + g);
                    }
                    obtain3 = array_obtain_element(ready_time_2,start_time);
                    for(int g = 0; g < thisSystem.task_number; g++){
                        ready_time_2[g] = * (obtain3 + g);
                    }
                    trace_2.assign(trace.begin(), trace.end());
                    for(int m = 0; m < trace_2[current_core].size(); m++){
                        if(trace_2[current_core][m] == i){
                            displacement_1 = m;
                        }
                    }
                    trace_2[current_core].erase(trace_2[current_core].begin()          +
displacement_1);
                    ready_time[i] = task_insertion(i, j, task_graph_adjacent_matrix,
total_finish_time_2);
                    on_what_core_2[i] = j;
                    displacement_2 = trace_insertion(trace_2, i, j, ready_time,
ready_time_2);
                    trace_2[j].insert(trace_2[j].begin()+displacement_2,i);

                    int * obtain4, *obtain5;
                    obtain4 = array_obtain_element(record_1,flag_2);
                    for(int g = 0; g < thisSystem.task_number; g++){
                        record_1[g] = * (obtain4 + g);
                    }
                    obtain5 = array_obtain_element(record_2,flag);
                    for(int g = 0; g < thisSystem.task_number; g++){
                        record_2[g] = * (obtain5 + g);
                    }
                    for(int r = 0; r < thisSystem.local_cores_number + 1; r++){
                        if(trace_2[r].size() > 0){
                            record_2[trace_2[r][0]] = 0;
                        }
                    }
                    for(int r = 0; r < thisSystem.task_number; r++){
                        if(record_1[r] == 0){
                            if(record_2[r] == 0){
                                if(stack_record[r] == 0){
```

```
                                    task_stack.push(r);
                                    stack_record[r] = 1;
                                }
                            }
                        }
                    }
                    int current_task = task_stack.top();
                    task_stack.pop();
                    ready_time[current_task] = 0;
                    int finish_time_compare = core_finish_time_2[on_what_core_2[current_task]];
                    int ready_time_compare = ready_time[current_task];
                    calculate_target_time(on_what_core_2, core_finish_time_2, local_cores_processing_time, ready_time, total_finish_time, current_task, finish_time_compare, ready_time_compare);

                    for(int r = 0; r < thisSystem.task_number; r++){
                        if(task_graph_adjacent_matrix[current_task][r] == 1){
                            record_1[r] = record_1[r] - 1;
                        }
                    }
                    record_2[current_task] = 1;
                    if(trace_2[on_what_core_2[current_task]].size() > 1){
                        for(int r = 1; r < trace_2[on_what_core_2[current_task]].size(); r++){
                            if(trace_2[on_what_core_2[current_task]][r - 1] == current_task){

record_2[trace_2[on_what_core_2[current_task]][r]] = 0;
                            }
                        }
                    }
                    for(int r = 0; r < thisSystem.task_number; r++){
                        if(record_1[r] == 0){
                            if(record_2[r] == 0){
                                if(stack_record[r] == 0){
                                    task_stack.push(r);
                                    stack_record[r] = 1;
                                }
                            }
                        }
                    }
                    operate_stack(task_stack, trace_2, record_1, record_2, stack_record, on_what_core_2, local_cores_processing_time,
```

```
task_graph_adjacent_matrix, core_finish_time_2, total_finish_time, ready_time);
                    float final_T = total_finish_time[thisSystem.task_number - 1];
                    float final_E = 0;
                    for(int r = 0; r < thisSystem.task_number; r++){
                        if(on_what_core_2[r] == thisSystem.local_cores_number) final_E = final_E + Energy_Cost_Cloud;
                        else final_E = final_E + Energy_Cost_Local[r][on_what_core_2[r]];
                    }
                    double tmp_rate = double((total_energy - final_E) / (final_T - total_time));
                    if(final_T <= total_time){
                        if(final_E < dynamic_energy){
                            flag1 = 1;
                            target_task = i;
                            target_core = j;
                            insert_location_1 = displacement_1;
                            insert_location_2 = displacement_2;
                            dynamic_total_time = final_T;
                            dynamic_energy = final_E;
                            transitive(on_what_core_2, on_what_core_3, ready_time, target_start_time, total_finish_time, dynamic_target_finish_time);
                        }
                    }
                    else if(final_T <= time_maximum){
                        if(final_E < total_energy && flag1 == 0){
                            if(deduction_rate < tmp_rate){
                                deduction_rate = tmp_rate;
                                flag2 = 1;
                                target_task = i;
                                target_core = j;
                                insert_location_1 = displacement_1;
                                insert_location_2 = displacement_2;
                                dynamic_total_time = final_T;
                                dynamic_energy = final_E;
                                transitive(on_what_core_2, on_what_core_3, ready_time, target_start_time, total_finish_time, dynamic_target_finish_time);
                            }
                        }
                    }
                }
            }
            if(flag1 + flag2 == 0) break;
            else{
```

```
trace[on_what_core[target_task]].erase(trace[on_what_core[target_task]].begin()+inse
rt_location_1);

trace[target_core].insert(trace[target_core].begin()+insert_location_2,target_task);
                    total_time = dynamic_total_time;
                    total_energy = dynamic_energy;
                    transitive(on_what_core_3, on_what_core, target_start_time, start_time,
dynamic_target_finish_time, end_time);
                    if(flag1 + flag2 == 0) break;
                }
        }
        print_result(trace, start_time, end_time, total_energy, total_time);
}

int main(){
        int  local_cores_processing_time[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}};

        int task_graph_adjacent_matrix_1[10][10] = {{0,1,1,1,1,0,0,0,0,0},
                                                    {0,0,0,0,0,1,1,0,0,0},
                                                    {0,0,0,0,0,0,1,0,0,0},
                                                    {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},
                                                    {0,0,0,0,0,0,0,0,0,1},
                                                    {0,0,0,0,0,0,0,0,0,1},
                                                    {0,0,0,0,0,0,0,0,0,0}};

        int task_graph_adjacent_matrix_2[10][10] = {{0,1,1,0,0,0,0,0,0,0},
                                                    {0,0,0,1,1,0,0,0,0,0},
                                                    {0,0,0,0,1,0,0,0,0,0},
                                                    {0,0,0,0,0,1,0,1,0,0},
                                                    {0,0,0,0,0,0,1,0,1,0},
                                                    {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_or_local[10] = {0};
        int priority[10] = {0};
        int sorted_priority_index[10] = {0};
```

```
        int average_time[10] = {0};
        int core_finish_time[4] = {0};
        int on_what_core[10] = {0};
        int ready_time_local[10] = {0};
        int ready_time_cloud[10] = {0};
        int ready_time_cloud_sending[10] = {0};
        int finish_time_cloud_sending[10] = {0};
        int finish_time_cloud_return[10] = {0};
        int finish_time_local[10] = {0};
        int total_finish_time[10] = {0};
        int start_time[10] = {0};
        int input_a_m[10][10] = {0};
        int flag[10] = {1, 1, 1, 1, 1, 1, 1, 1, 1, 1};
        int flag_2[10] = {0};
        float total_energy = 0;
        int time_maximum = 27;
        int myChoose;
        vector<vector<int>> trace(4);
        float Energy_Cost_Local[10][3];
        float Energy_Cost_Cloud = thisSystem.power_coefficient_cloud *
thisSystem.cloud_sending_time;

        for(int i = 0; i < thisSystem.task_number; i++){
           for(int j = 0; j < thisSystem.local_cores_number; j++){
                Energy_Cost_Local[i][j] = thisSystem.power_coefficient_local[j] *
local_cores_processing_time[i][j];
           }
        }
        printf("Please choose an example: 1 or 2.\n");
        scanf("%d",&myChoose);
        switch (myChoose){
        case 1:
            for(int i = 0; i < thisSystem.task_number; i++){
                for(int j = 0; j < thisSystem.task_number; j++){
                    input_a_m[i][j] = task_graph_adjacent_matrix_1[i][j];
                }
            }
            break;
        case 2:
            for(int i = 0; i < thisSystem.task_number; i++){
                for(int j = 0; j < thisSystem.task_number; j++){
                    input_a_m[i][j] = task_graph_adjacent_matrix_2[i][j];
                }
            }
```

```
            break;
        }
        for(int r = 0; r < thisSystem.task_number; r++){
            for(int p = 0; p < thisSystem.task_number; p++){
                if(input_a_m[r][p] == 1){
                    flag_2[p] = flag_2[p] + 1;
                }
            }
        }

        LARGE_INTEGER initial_time;
        double cpu_run_time = 0;
        LARGE_INTEGER end_time;
        double f2;
        LARGE_INTEGER f;
        QueryPerformanceFrequency(&f);
        f2 = (double)f.QuadPart;
        QueryPerformanceCounter(&initial_time);
        Phase_One(local_cores_processing_time, cloud_or_local);
        Phase_Two(local_cores_processing_time, priority, sorted_priority_index,
input_a_m, average_time, cloud_or_local);
        Phase_Three(trace, local_cores_processing_time, sorted_priority_index,
input_a_m, cloud_or_local, ready_time_local, ready_time_cloud,
ready_time_cloud_sending, finish_time_cloud_sending, finish_time_cloud_return,
finish_time_local, total_finish_time, core_finish_time, on_what_core);
        for(int i = 0; i <= thisSystem.local_cores_number; i++){
            for(int j = 0; j < thisSystem.task_number; j++){
                if(on_what_core[j] == i){
                    trace[i].push_back(j);
                }
            }
        }
        for(int i = 0; i < thisSystem.task_number; i++){
            if(on_what_core[i] == 3){
                total_energy += Energy_Cost_Cloud;
            }
            else{
                total_energy += Energy_Cost_Local[i][on_what_core[i]];
            }
        }
        for(int i = 0; i < thisSystem.task_number; i++){
            start_time[i] = max(ready_time_local[i], ready_time_cloud_sending[i]);
        }
        print_result(trace, start_time, total_finish_time, total_energy,
```

```
total_finish_time[thisSystem.task_number - 1]);
        Reschedule(trace, local_cores_processing_time, flag, flag_2, input_a_m, on_what_core, time_maximum, total_finish_time[thisSystem.task_number - 1], total_energy, start_time, total_finish_time, Energy_Cost_Cloud, Energy_Cost_Local);
        QueryPerformanceCounter(&end_time);

        cpu_run_time = ((double)end_time.QuadPart - (double)initial_time.QuadPart) / f2;
        printf("The cpu running time of example 1 is %fs\n", cpu_run_time);

        return 0;
}
```