Northeastern University

EECE 7205 Fundanmentals of Computer Engineering Project 2

Liangshe Li

- **Part1 Program Setup**
- **Part2 Input and Output**
- **Part3 Conclusion**

Part1 Program Setup

This part, I will explain the lines in my code.

First is the head files and namespace I set:

Open

7205project2.cpp
~/source/ns-3.31/scratch

```
1 #include<iostream>
2 #include<vector>
3 #include<limits.h>
4 #include"Matrix.h"
5 #include<ctime>
6
7 using namespace std;
8 using namespace Numeric_lib;
9
```

Here I establish a new data structure called “task”.

```
struct task {
        int num; // the number of the task
        bool ct;  // judge whether the task is a cloud task
        double pri; //
        int FTl;  // the finish time of the task in a core
        int FTWS; // the finish time of the task in sending
        int FTC;   // the finish time of the task in cloud
        int FTWR;   // the finish time of the task in receiving
        int RTl; // the earlist  time that the task can start in local core
        int RTWS; // the earlist  time that the task can start in sending
        int RTC;  // the earlist  time that the task can start in cloud
        int RTWR; //the earlist  time that the task can start in receiving
        int ST; // the task's actual start time
        int chan; // illustrate which channel the task operate (local core = 0,1,2, cloud=3)
        bool exit;  //whether it is an exit task
        bool entry;  // whether it is an entry task
        int ready1;
        int ready2;
};
```

Here I explain the function of each parameter in the new data structure:

- **num:** the serial number of the task
- **ct:** judge whether the task is a cloud task
- **pri:** the priority level of the task
- **FTL:** if local schedule, the finish time of the task in local core
- **FTWS:** if cloud schedule, the finish sending time of the task in wireless channel
- **FTC:** if cloud schedule, the finish compute time of the task in cloud
- **FTWR:** if cloud schedule, the finish receiving time of the task in wireless channel
- **RTL:** if local schedule, the ready time of the task in local core
- **RTWS:** if cloud schedule, the ready sending time of the task in wireless channel
- **RTC:** if cloud schedule, the ready compute time of the task in cloud
- **RTWR:** if cloud schedule, the ready receiving time of the task in wireless channel
- **ST:** the actual start time of the task
- **chan:** the location of the task(if 0,1,2, is in local core 1,2,3; if 3, is in cloud)
- **exit:** judge whether the task is an exit task
- **entry:** judge whether the task is an entry task
- **ready1:** in kernel algorithm, the number of immediate predecessors of task which have not been scheduled
- **ready2:** in kernel algorithm, the number of tasks in the same channel before the task which have not been scheduled

Here is the Phase One in Step One: Primary assignment.

```
// Phase one in step one: primary assignment
void primary(vector<task>&ini, Matrix<int, 2>&ta,int t)
{
        int min;
        unsigned int i;
        unsigned int j;
        for (i = 0; i < ta.dim1(); i++)
        {
                ini[i].num = i + 1;
                min = ta(i, 0);
                for (j = 0; j < ta.dim2(); j++)
                        if (ta(i, j) < min)
                                min = ta(i, j);
                if (min > t)
                        ini[i].ct = 1;
                else
                        ini[i].ct = 0;
        }
}
```

Here is the Phase Two in Step One: Task prioritizing. (I also determine the value of exit and entry)

```
49
50 // Phase two in step one: task prioritizing
51 void prioritize(vector<task>&ini, Matrix<int, 2>&ta, Matrix<int, 2>&G,int t)
52 {
53         unsigned int i;
54         unsigned int j,m;
55         int k ;
56         double w;
57         double max;
58         m = ini.size() - 1;
59         for (i = 0; i < ini.size(); i++)
60         {
51                 k = 0;
52                 for (j = 0; j < G.dim2(); j++)
53                         if (G(m - i, j) == 1)
54                                 k = k + 1;
55                 if (k == 0)
56                         ini[m - i].exit = 1;
57                 k = 0;
58                 for (j = 0; j < G.dim2(); j++)
59                         if (G(j, m - i) == 1)
70                                 k = k + 1;
71                 if(k==0)
72                         ini[m - i].entry = 1;
73                 max = 0;
74                 w = 0;
75                 if (!(ini[m - i].ct))
76                 {
77                         for (j = 0; j < ta.dim2(); j++)
78                                 w = w + ta(m - i, j);
79                         w = w / 3;
30                 }
31                 else
32                         w = t;
33                 for (j = 0; j < G.dim2(); j++)
34                         if ((G(m - i, j) == 1) && (max < ini[j].pri))
35                                 max = ini[j].pri;
36                 ini[m - i].pri = w + max;
37         }
38 }
39
```

Here is the main function of Phase Three in Step One: Execution unit selection.
I use “mint” to compute the minimum finish time in local core and use “anot” to compute the minimum finish time in cloud. Then compare this two values to determine whether the task in cloud or local core.

```
53
54 void initials(vector<task>&S, vector<task>&ini, Matrix<int, 2>&ta, Matrix<int, 2>&G, int ts, int tc, int tr)
55 {
56         unsigned int i;
57         int t;
58         int maxp; // find the max priority in each iteration of ini
59         int mint;  // find the minimum finish time of local
60         int anot;  // perpare for another time (cloud)
61         t = ts + tc + tr;
62         for (i = 0; i < G.dim1(); i++)
63         {
64                 maxp = find_biggest_pri(ini);
65                 if (!ini[maxp].ct)
66                 {
67                         mint = locals(ini[maxp], S, G, ta);
68                         anot = clouds(ini[maxp], S, G, ts, tc, tr);
69                         if (anot < mint)
70                         {
71                                 ini[maxp].RTl = 0;
72                                 ini[maxp].FTl = 0;
73                                 ini[maxp].chan = 3;
74                                 ini[maxp].FTWR = anot;
75                                 ini[maxp].ST = anot - t;
76                         }
77                         else
78                         {
79                                 ini[maxp].FTC = 0;
80                                 ini[maxp].FTWS = 0;
81                                 ini[maxp].RTWS = 0;
```

```
Open                                                    7205project2.cp
                                                        ~/source/ns-3.31/scra
1                                       ini[maxp].RTl = 0;
2                                       ini[maxp].FTl = 0;
3                                       ini[maxp].chan = 3;
4                                       ini[maxp].FTWR = anot;
5                                       ini[maxp].ST = anot - t;
5                               }
7                               else
8                               {
9                                       ini[maxp].FTC = 0;
0                                       ini[maxp].FTWS = 0;
1                                       ini[maxp].RTWS = 0;
2                                       ini[maxp].RTC = 0;
3                                       ini[maxp].RTWR = 0;
4                                       ini[maxp].FTWR = 0;
5                                       ini[maxp].FTl = mint;
5                                       ini[maxp].ST = mint - ta(ini[maxp].num - 1, ini[maxp].chan);
7                               }
8                       }
9                       else
0                       {
1                               ini[maxp].FTl = 0;
2                               ini[maxp].RTl = 0;
3                               ini[maxp].chan = 3;
4                               ini[maxp].FTWR= clouds(ini[maxp], S, G, ts, tc, tr);
5                               ini[maxp].ST = ini[maxp].FTWR - t;
5                       }
7                       S.push_back(ini[maxp]);
8                       ini.erase(ini.begin() + maxp);
9               }
0 }
1
2 // return a task's finish time
```

Here is the function which returns the maximum finish time of task in local core.

```
5
6 // if local schedule, return the smallest finish time
7 int locals(task &vi, vector<task>&S, Matrix<int, 2>&G, Matrix<int, 2>&ta)
8 {
9         vi.RTl = d_rtl(vi, S, G);
0         unsigned int i;
1         unsigned int j;
2         int mint=INT_MAX;
3         int ft;
4         int max = 0; // find a local core's biggest finish time
5         if (S.size()==0)
6         {
7                 for (i = 0; i < ta.dim2(); i++)
8                 {
9                         ft = ta(vi.num - 1, i);
0                         if (mint > ft)
1                         {
2                                 mint = ft;
3                                 vi.chan = i;
4                         }
5                 }
6                 return mint;
7         }
8         for (i = 0; i < ta.dim2(); i++)
9         {
0                 ft = vi.RTl + ta(vi.num - 1, i);
1                 max = 0;
2                 for (j = 0; j < S.size(); j++)
3                         if ((S[j].chan == i) && (max < S[j].FTl))
4                                 max = S[j].FTl;
5                 if(max>vi.RTl)
6                         ft=max+ ta(vi.num - 1, i);
7                 if (mint > ft)
8                 {
9                         mint = ft;
0                         vi.chan = i;
1                 }
2         }
3         return mint;
4 }
5
```

The first function returns the serial number of the task whose priority is biggest.
The third function returns RTL of the task if local schedule.

```
90 int find_biggest_pri(vector<task>&ini)
91 {
92         unsigned int i;
93         int max=0;
94         for (i = 0; i < ini.size(); i++)
95                 if (ini[i].pri > ini[max].pri)
96                         max = i;
97         return max;
98 }
99
00 // find the max in two numbers
01 int max2(int &m, int &n)
02 {
03         if (m >= n)
04                 return m;
05         else
06                 return n;
07 }
08
09 // if local schedule, return RTL
10 int d_rtl(task &vi, vector<task>&S, Matrix<int, 2>&G)
11 {
12         unsigned int i;
13         unsigned int j;
14         int max=0;
15         if (S.size()!=0)
16         {
17                 for (i = 0; i < G.dim2(); i++)
18                         if (G(i, vi.num - 1) == 1)
19                                 for (j = 0; j < S.size(); j++)
20                                         if ((S[j].num == i + 1)&&(max < max2(S[j].FTl, S[j].FTWR)))
21                                                 max = max2(S[j].FTl, S[j].FTWR);
22         }
23         return max;
24 }
25
```

Here is the function which returns the finish receiving time of task in wireless channel if cloud schedule. Also compute FTWS and FTC.

```
199                     mint = ft;
200                     vi.chan = i;
201             }
202     }
203     return mint;
204 }
205
206 // if cloud schedule, return the finish time
207 int clouds(task &vi, vector<task>&S, Matrix<int, 2>&G, int ts, int tc, int tr)
208 {
209     vi.RTWS = d_rtws(vi, S, G);
210     unsigned int i;
211     int maxs = 0;
212     int t;
213     int maxc = 0;
214     int maxr = 0;
215     int ft;
216     t = ts + tc + tr;
217     if (S.size()==0)
218     {
219             vi.FTWS = ts;
220             vi.RTC = ts;
221             vi.FTC = ts + tc;
222             vi.RTWR = ts+tc;
223             return t;
224     }
225     for(i=0;i<S.size();i++)
226             if (S[i].chan == 3)
227                     if (maxs < S[i].FTWS)
228                             maxs = S[i].FTWS;
229     if (maxs > vi.RTWS)
230             vi.FTWS = maxs + ts;
```

Open

```
223                     return t;
224     }
225     for(i=0;i<S.size();i++)
226             if (S[i].chan == 3)
227                     if (maxs < S[i].FTWS)
228                             maxs = S[i].FTWS;
229     if (maxs > vi.RTWS)
230             vi.FTWS = maxs + ts;
231     else
232             vi.FTWS = vi.RTWS + ts;
233     vi.RTC = d_rtc(vi, S, G);
234     for (i = 0; i < S.size(); i++)
235             if (S[i].chan == 3)
236                     if (maxc < S[i].FTC)
237                             maxc = S[i].FTC;
238     if (maxc > vi.RTC)
239             vi.FTC = maxc + tc;
240     else
241             vi.FTC = vi.RTC + tc;
242     vi.RTWR = d_rtwr(vi);
243     for (i = 0; i < S.size(); i++)
244             if (S[i].chan == 3)
245                     if (maxr < S[i].FTWR)
246                             maxr = S[i].FTWR;
247     if (maxr > vi.RTWR)
248             ft = maxr + tr;
249     else
250             ft = vi.RTWR + tr;
251     return ft;
252 }
253
254 void initials(vector<task>&S, vector<task>&ini, Matrix<int, 2>&ta, Matri
```

Here I compute RTWS, RTC, RTWR of the task if cloud schedule.

```
124 }
125
126 // if cloud schedule, return RTWS
127 int d_rtws(task &vi, vector<task>&S, Matrix<int, 2>&G)
128 {
129         unsigned int i;
130         unsigned int j;
131         int max=0;
132         if (S.size()!=0)
133         {
134                 for (i = 0; i < G.dim2(); i++)
135                         if (G(i, vi.num - 1) == 1)
136                                 for (j = 0; j < S.size(); j++)
137                                         if ((S[j].num == i + 1)&&(max < max2(S[j].FTl, S[j].FTWS)))
138                                                 max = max2(S[j].FTl, S[j].FTWS);
139         }
140         return max;
141 }
142
143 // if cloud schedule, return RTC
144 int d_rtc(task &vi, vector<task>&S, Matrix<int, 2>&G)
145 {
146         unsigned int i;
147         unsigned int j;
148         int max=vi.FTWS;
149         if (S.size()!=0)
150         {
151                 for (i = 0; i < G.dim2(); i++)
152                         if (G(i, vi.num - 1) == 1)
153                                 for (j = 0; j < S.size(); j++)
154                                         if ((S[j].num == i + 1)&&(max < max2(vi.FTWS, S[j].FTC)))
155                                                 max = max2(vi.FTWS, S[j].FTC);
156         }
157         return max;
158 }
159
160 // if cloud schedule, return RTWR
161 int d_rtwr(task &vi)
162 {
163         return vi.FTC;
164 }
165
```

The first function returns the actual finish time of the task.
The second function prints the schedule of all tasks (involving the channel of task, the start and finish time of task)

```
301
302 // return a task's finish time
303 int find_ft(task&vi)
304 {
305         int max;
306         max = max2(vi.FTl, vi.FTWR);
307         return max;
308  }
309
310 // print the sequence S
311 void prints(vector<task>&S)
312 {
313         unsigned int i;
314         int k,m;
315         for (i = 0; i < S.size(); i++)
316         {
317                 k = 1 + S[i].chan;
318                 m = find_ft(S[i]);
319                 cout << "Task" << S[i].num << ": ";
320                 switch (S[i].chan)
321                 {
322                 case 0:
323                         cout << "local core" << k << ", ";
324                         break;
325                 case 1:
326                         cout << "local core" << k << ", ";
327                         break;
328                 case 2:
329                         cout << "local core" << k << ", ";
330                         break;
331                 case 3:
332                         cout << "cloud" << ", ";
333                         break;
334                 default:
335                         break;
336                 }
337                 cout << "start time is: " << S[i].ST << ",finish time is: "<<m<<endl;
338         }
339 }
340
341 // return the completion time of sequence S
```

The first function returns the total completion time of the schedule.
The second function returns the total energy consumed in the schedule.

```
39 }
40
41 // return the completion time of sequence S
42 int find_tcom(vector<task>&S)
43 {
44         unsigned int i;
45         int max=0;
46         for (i = 0; i < S.size(); i++)
47                 if ((S[i].exit) && (max < find_ft(S[i])))
48                         max = find_ft(S[i]);
49         return max;
50 }
51
52 // return the total energy of the sequence S
53 double find_en(vector<task>&S, int p1, int p2, int p3, double ps)
54 {
55         unsigned int i;
56         double ene=0;
57         for (i = 0; i < S.size(); i++)
58         {
59                 switch (S[i].chan)
60                 {
61                 case 0:
62                         ene = ene + p1 * (find_ft(S[i]) - S[i].ST);
63                         break;
64                 case 1:
65                         ene = ene + p2 * (find_ft(S[i]) - S[i].ST);
66                         break;
67                 case 2:
68                         ene = ene + p3 * (find_ft(S[i]) - S[i].ST);
69                         break;
70                 case 3:
71                         ene = ene + ps * (S[i].FTWS - S[i].ST);
72                         break;
73                 default:
74                         break;
75                 }
76         }
77         return ene;
78 }
79
```

Here is Step Two: Task Migration.
The outer loop repeats to run the “mcc” function until the energy consumed is minimized.

```
517 }
518
519 void outerloop(vector<task>&S, Matrix<int, 2>&G, Matrix<int, 2>&ta, int ts, int tc, int tr, int p1, int p2, int p3, double ps, int tmax)
520 {
521         double en;
522         double en1=0;
523         en = find_en(S, p1, p2, p3, ps);
524         while (en1<en)
525         {
526                 en= find_en(S, p1, p2, p3, ps);
527                 mcc(S, G, ta, ts, tc, tr, p1, p2, p3, ps, tmax);
528                 en1= find_en(S, p1, p2, p3, ps);
529         }
530 }
531
532 int main()
```

Here is the “mcc” function which involves kernel algorithm.

Open

7205project2.cpp
~/source/ns-3.31/scratch

```
474 void mcc(vector<task>&S, Matrix<int, 2>&G, Matrix<int, 2>&ta, int ts, int tc, int tr,int p1, int p2, int p3, double ps, int tmax)
475 {
476         unsigned int i, j;
477         int tcom;
478         int tcom2;
479         int a;
480         double en;
481         double en1;
482         double en2;
483         double ratio1=0;
484         double ratio2;
485         vector<task>SN;
486         tcom = find_tcom(S);
487         en = find_en(S, p1, p2, p3, ps);
488         for (i = 0; i < S.size(); i++)
489         {
490                 a = S[i].chan;
491                 if (S[i].chan != 3)
492                 {
493                         for (j = 0; j < 4; j++)
494                         {
495                                 if (j != a)
496                                 {
497                                         SN.erase(SN.begin(), SN.end());
498                                         en1 = find_en(S, p1, p2, p3, ps);
499                                         kernel(S, SN, j, S[i], G, ta, ts, tc, tr);
500                                         tcom2 = find_tcom(SN);
501                                         en2 = find_en(SN, p1, p2, p3, ps);
502                                         if ((en2 < en1) && (tcom >= tcom2))
503                                                 S = SN;
504                                         else if ((en2 < en1) && (tcom2 <= tmax))
505                                         {
506                                                 ratio2 = (en - en2) / (tcom2 - tcom);
507                                                 if (ratio2 > ratio1)
508                                                 {
509                                                         ratio1 = ratio2;
510                                                         S = SN;
511                                                 }
512                                         }
513                                 }
514                         }
515                 }
516         }
```

Here is the function which is the implementation of kernel algorithm.

```
433
434 void kernel(vector<task>&S, vector<task>&SN, int ktar, task vtar, Matrix<int, 2>&G, Matrix<int, 2>&ta,int ts, int tc, int tr)
435 {
436         unsigned int i;
437         int m;
438         int t;
439         t = ts + tc + tr;
440         vector<task>re;
441         re = S;
442         for (i = 0; i < re.size(); i++)
443                 if (vtar.num == re[i].num)
444                 {
445                         re[i].chan = ktar;
446                         if (ktar == 3)
447                         {
448                                 re[i].FTl = 0;
449                                 re[i].RTl = 0;
450                         }
451                 }
452         while (re.size()!=0)
453         {
454                 get_ready1(re, G);
455                 get_ready2(re);
456                 m = 0;
457                 while ((re[m].ready1!=0)&&(re[m].ready2 != 0))
458                         m = m + 1;
459                 if (re[m].chan == 3)
460                 {
461                         re[m].FTWR = clouds(re[m], SN, G, ts, tc, tr);
462                         re[m].ST = re[m].FTWR - t;
463                 }
464                 else
465                 {
466                         re[m].FTl = localse(re[m], SN, G, ta);
467                         re[m].ST = re[m].FTl - ta(re[m].num - 1, re[m].chan);
468                 }
469                 SN.push_back(re[m]);
470                 re.erase(re.begin() + m);
471         }
472 }
473
```

The first function is to compute the value of ready1.
The second function is to compute the value of ready2.

```
380 //compute all the ready1 in a sequence
381 void get_ready1(vector<task>&S, Matrix<int, 2>&G)
382 {
383         unsigned int i, j, k;
384         int m;
385         for (i = 0; i < S.size(); i++)
386         {
387                 m = 0;
388                 for (j = 0; j < G.dim2(); j++)
389                         if (G(j, S[i].num-1) == 1)
390                                 for (k = 0; k < S.size(); k++)
391                                         if (S[k].num == j + 1)
392                                                 m = m + 1;
393                 S[i].ready1 = m;
394         }
395 }
396
397 //compute all the ready2 in a sequence
398 void get_ready2(vector<task>&S)
399 {
400         unsigned int i, j;
401         int m;
402         for (i = 0; i < S.size(); i++)
403         {
404                 m = 0;
405                 for (j = 0; j < S.size(); j++)
406                         if ((S[i].chan == S[j].chan) && (S[j].ST < S[i].ST))
407                                 m = m + 1;
408                 S[i].ready2 = m;
409         }
410 }
411
```

Here is where I compute the running time of the program (in the main function). I use “rt” to compute running time.

```
ta1(9, 2) = 2;
primary(ini1, ta1, t1);
prioritize(ini1, ta1, G1,t1);
start = clock();
initials(S1, ini1, ta1, G1, ts1, tc1, tr1);
end = clock();
cout << "Initial schedule:" << endl;
prints(S1);
rt = (double)(end - start) / (double)(CLOCKS_PER_SEC)*(double)(1000.000000);
cout << " Now the total energy is: " << find_en(S1, p11, p12, p13, ps1)<<endl;
 cout << " Now the completion time is: " << find_tcom(S1) << endl;
cout << "The running time of initial schedule of Graph 1 is "<<rt<<" ms"<< endl;
start = clock();
outerloop(S1, G1, ta1, ts1, tc1, tr1, p11, p12, p13, ps1, tmax1);
end = clock();
cout << "After Task Migration:" << endl;
prints(S1);
rt = (double)(end - start) / (double)(CLOCKS_PER_SEC)*(double)(1000.000000);
cout << " Now the total energy is: " << find_en(S1, p11, p12, p13, ps1) << endl;
 cout << " Now the completion time is: " << find_tcom(S1) << endl;
cout << "The running time of task migration of Graph 1 is "<<rt<<" ms"<< endl;
cout << endl;
int N2 = 8;  // the number of tasks
```

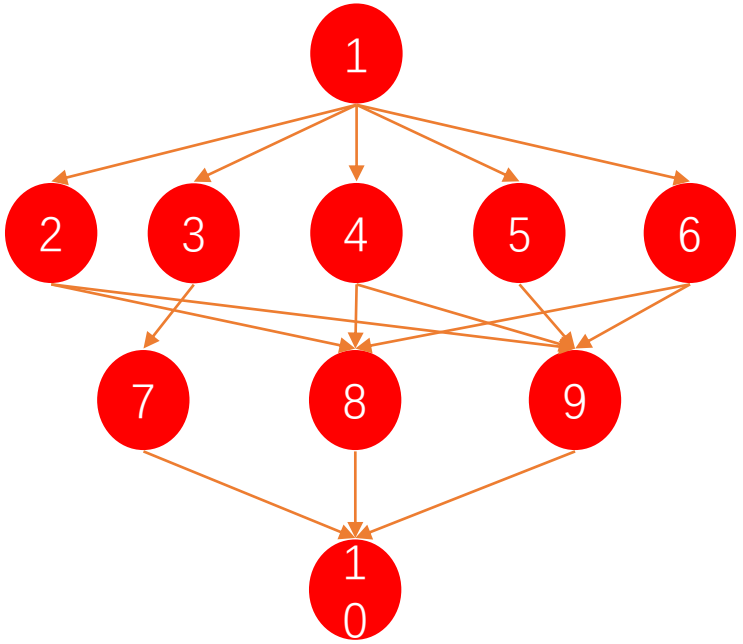
Part2 Input and Output

- This part I will use two examples to verify the correctness of mu codes.

- The first example is the same with paper [1].

- The second example is a good example to show how the cloud can save energy for the whole schedule.

Input of the first example



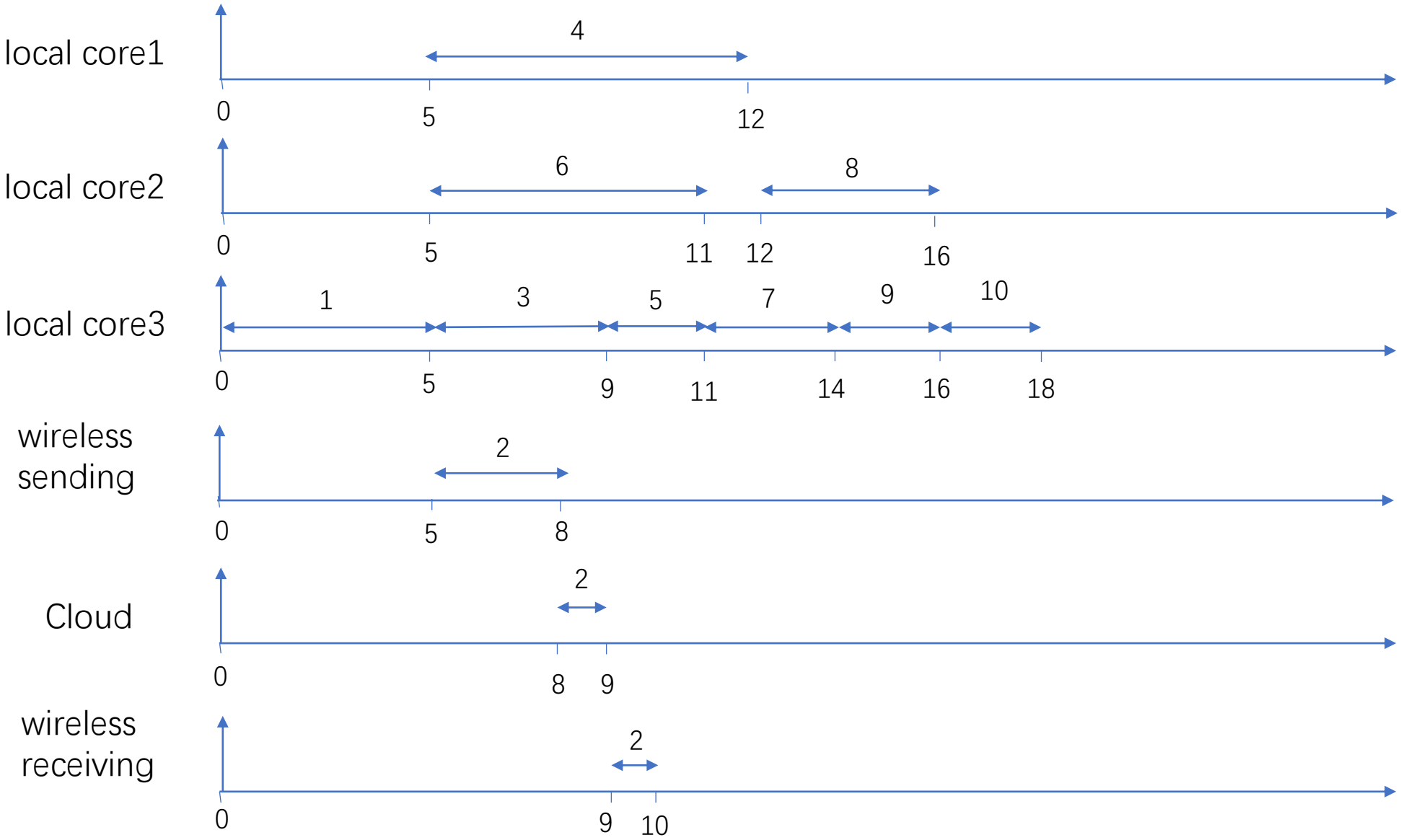
$$T_{max} = 27$$

$$\begin{cases} T^S = 3; \\ T^C = 1; \\ T^R = 1; \end{cases}$$

$$\begin{cases} P_1 = 1; \\ P_2 = 2; \\ P_3 = 4; \\ P_s = 0.5; \end{cases}$$

Time in each core	local core1	local core2	local core3
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

Output of the first example after initial schedule

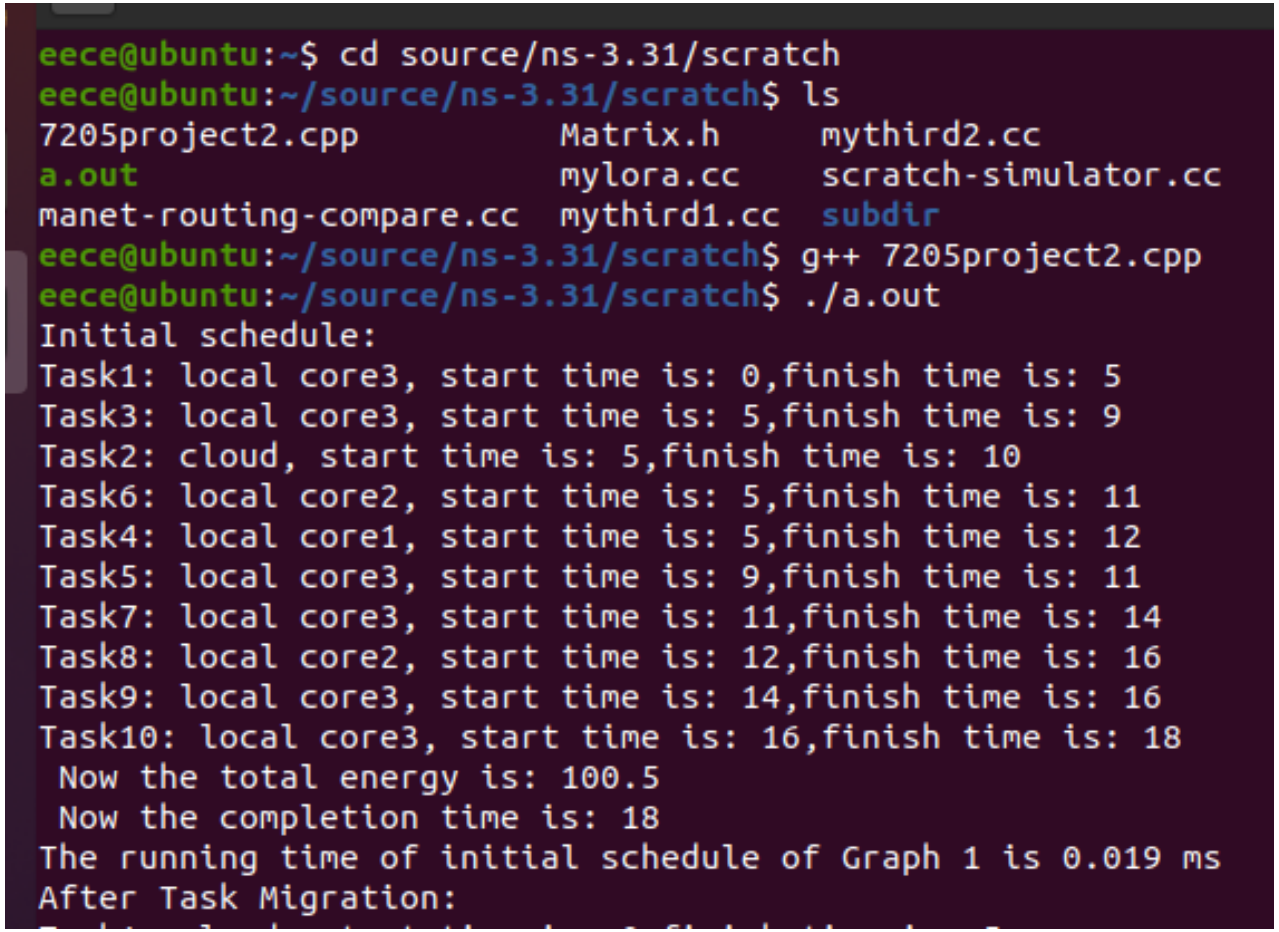


After initial schedule

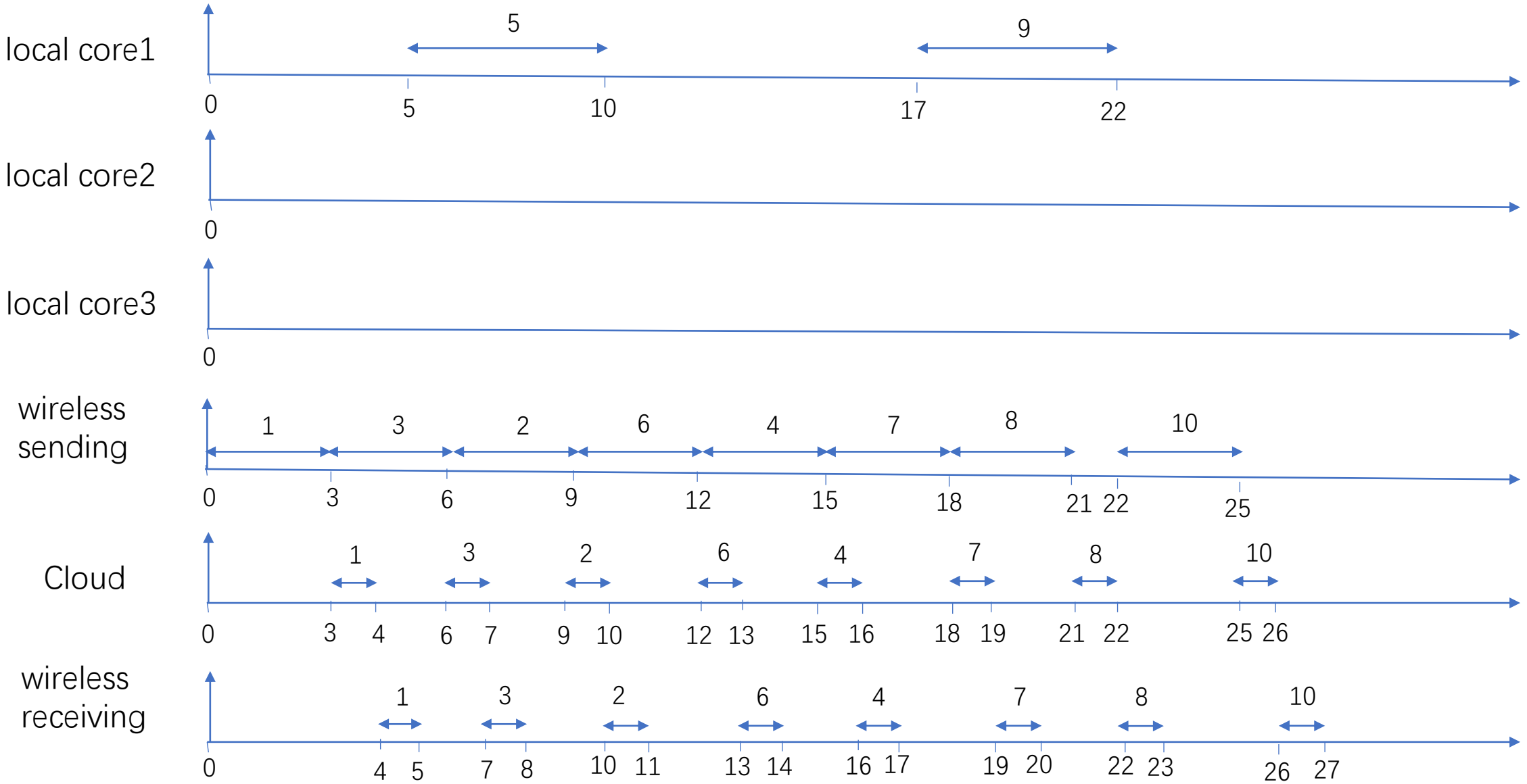
The completion time of the schedule is 18.

The total energy consumed is 100.5.

The running time of the initial schedule program is 0.019 ms.



Output of the first example after task migration



After task migration

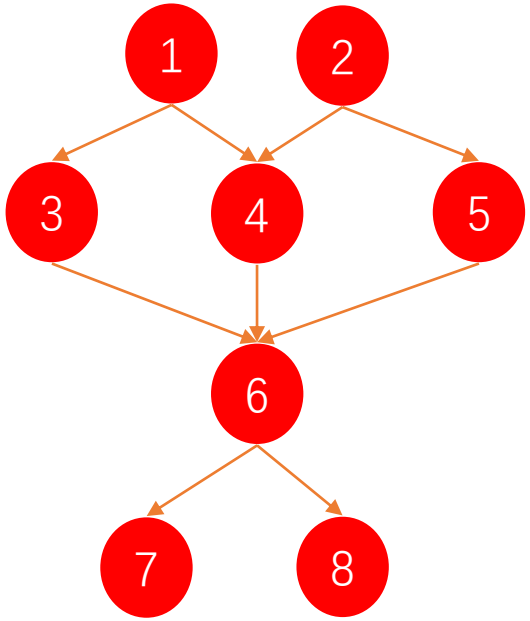
The completion time of the schedule is 27.

The total energy consumed is 22.

The running time of the initial schedule program is 1.208 ms.

```
The running time of initial schedule of Graph 1 is 0.019 ms
After Task Migration:
Task1: cloud, start time is: 0,finish time is: 5
Task3: cloud, start time is: 3,finish time is: 8
Task2: cloud, start time is: 6,finish time is: 11
Task6: cloud, start time is: 9,finish time is: 14
Task4: cloud, start time is: 12,finish time is: 17
Task5: local core1, start time is: 5,finish time is: 10
Task7: cloud, start time is: 15,finish time is: 20
Task8: cloud, start time is: 18,finish time is: 23
Task9: local core1, start time is: 17,finish time is: 22
Task10: cloud, start time is: 22,finish time is: 27
 Now the total energy is: 22
 Now the completion time is: 27
The running time of task migration of Graph 1 is 1.208 ms
```

Input of the second example



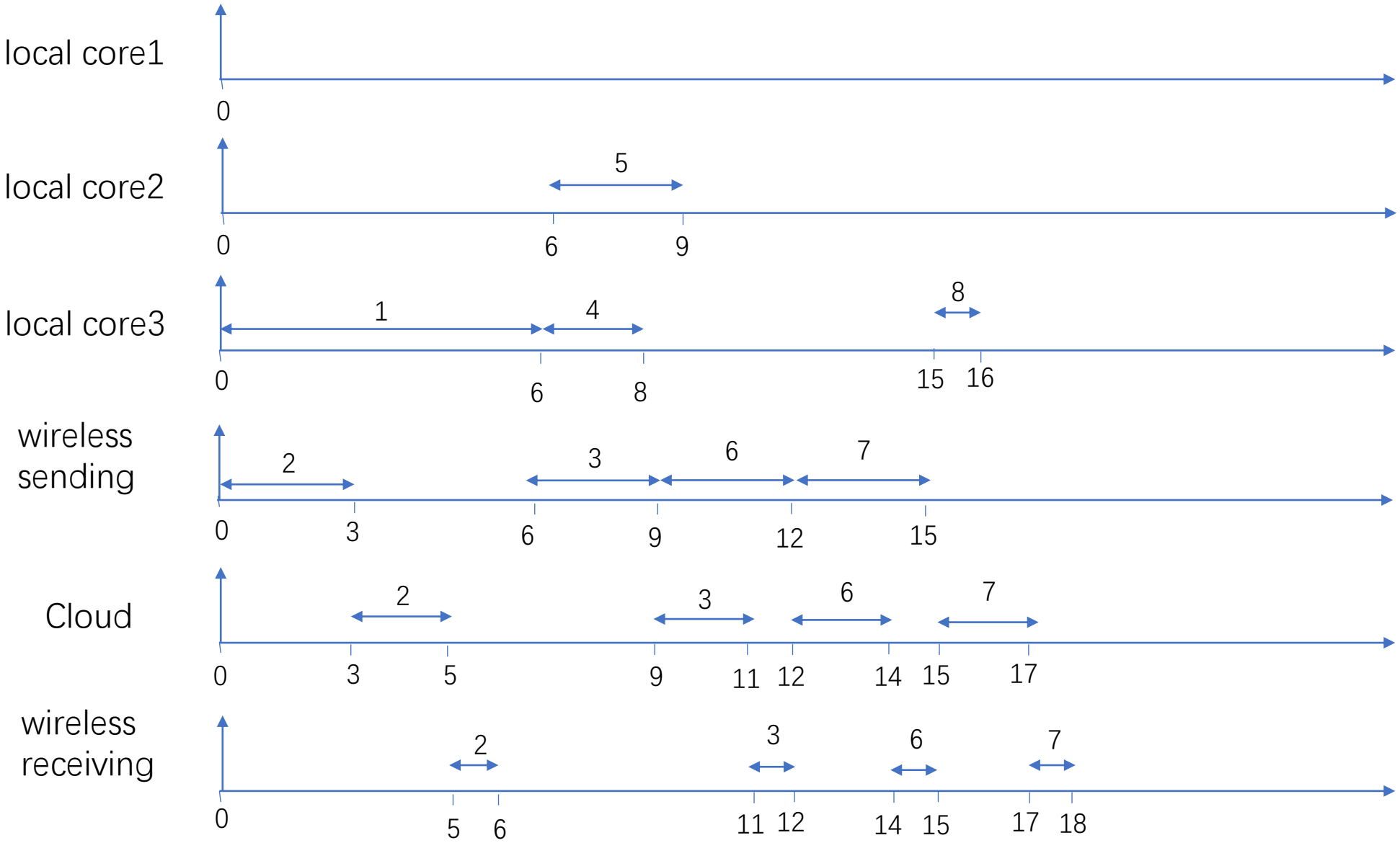
Time in each core	local core1	local core2	local core3
Task 1	9	8	6
Task 2	8	7	6
Task 3	10	9	7
Task 4	6	4	2
Task 5	5	3	1
Task 6	7	5	4
Task 7	15	11	10
Task 8	6	4	1

$$T_{max} = 27$$

$$\begin{cases} T^S = 3; \\ T^C = 2; \\ T^R = 1; \end{cases}$$

$$\begin{cases} P_1 = 2; \\ P_2 = 3; \\ P_3 = 5; \\ P_s = 1.5; \end{cases}$$

Output of the second example after initial schedule



After initial schedule

The completion time of the schedule is 18.

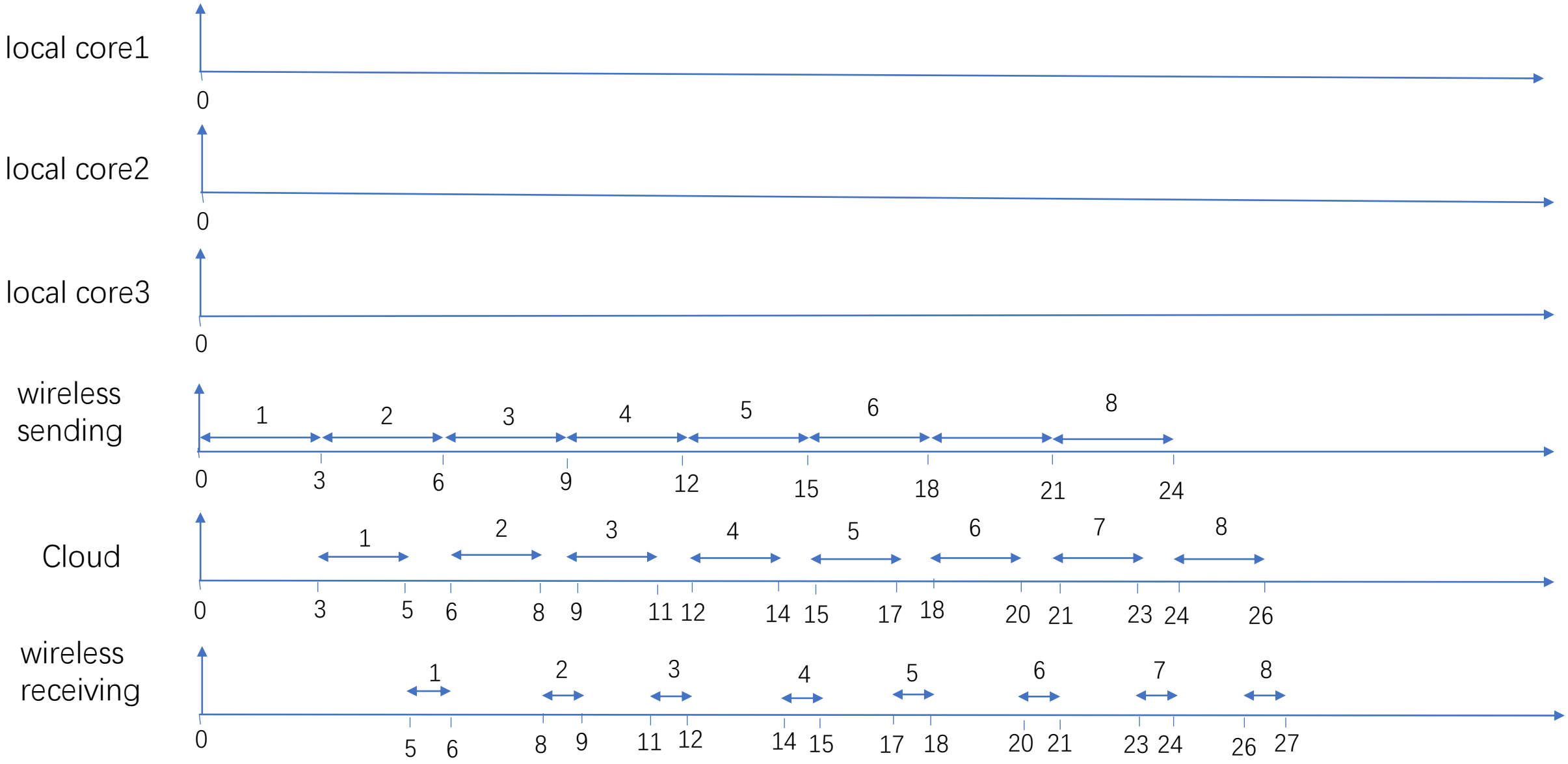
The total energy consumed is 72.

The running time of the initial schedule program is 0.011 ms.

```
The running time of task migration of Graph 1 is 1.208 ms

Initial schedule:
Task1: local core3, start time is: 0,finish time is: 6
Task2: cloud, start time is: 0,finish time is: 6
Task3: cloud, start time is: 6,finish time is: 12
Task4: local core3, start time is: 6,finish time is: 8
Task5: local core2, start time is: 6,finish time is: 9
Task6: cloud, start time is: 9,finish time is: 15
Task7: cloud, start time is: 12,finish time is: 18
Task8: local core3, start time is: 15,finish time is: 16
 Now the total energy is: 72
 Now the completion time is: 18
The running time of initial schedule of Graph 2 is 0.011 ms
After all of Task Migration:
```

Output of the second example after task migration

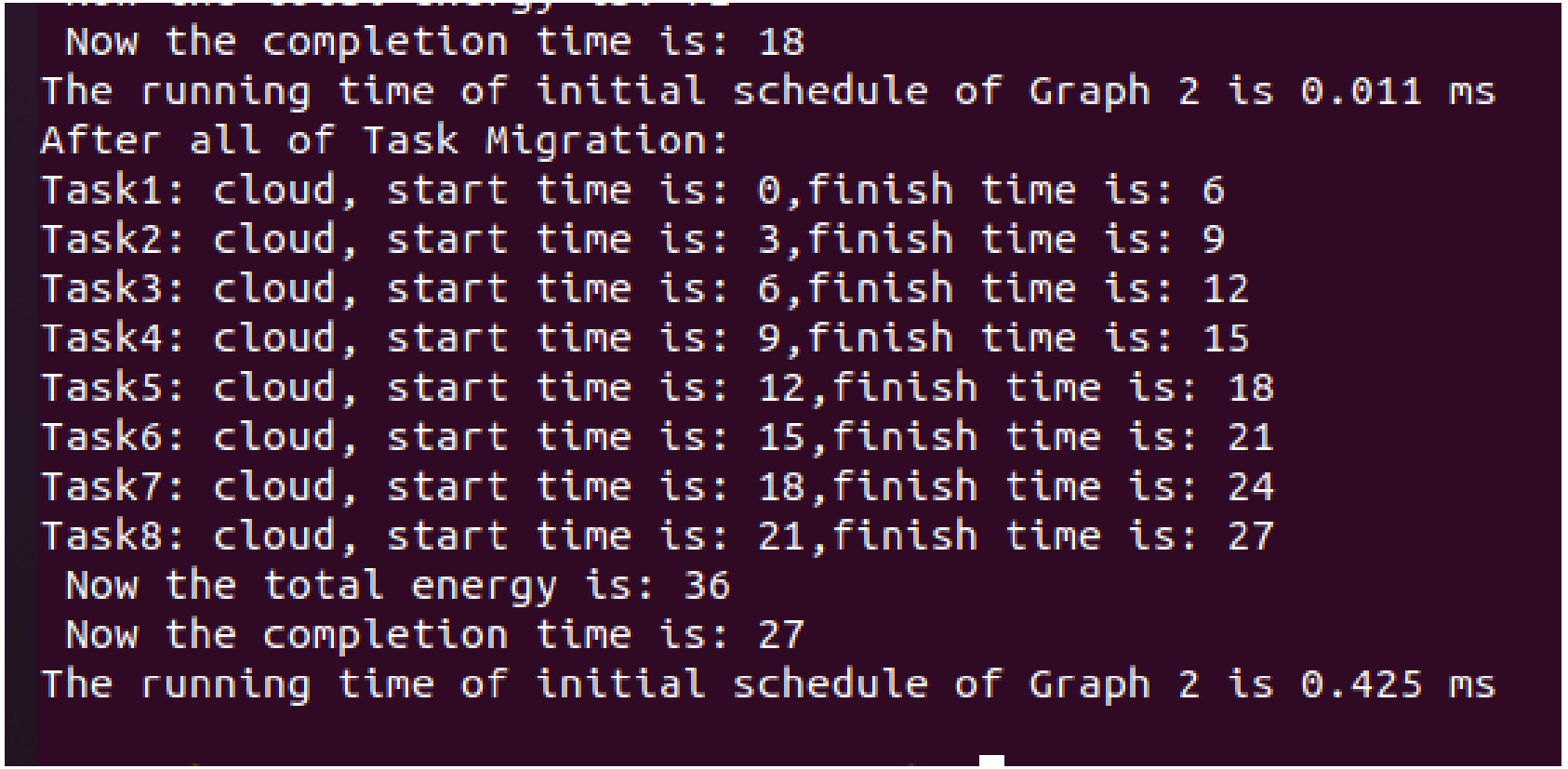


After task migration

The completion time of the schedule is 27.

The total energy consumed is 36.

The running time of the initial schedule program is 0.425 ms.



Part3 Conclusion

- Achieve the goal which is to minimize the consumption of the mobile device.
- Apply one example in paper [1].
- The result of another example is absolutely correct.
- The result of the first example is little different from paper [1]

References

[1] X. Lin, Y. Wang, Q. Xie and M. Pedram, "Energy and Performance-Aware Task Scheduling in a Mobile Cloud Computing Environment," *2014 IEEE 7th International Conference on Cloud Computing*, Anchorage, AK, 2014, pp. 192-199, doi: 10.1109/CLOUD.2014.35.