project progress report

ML-RCS algorithm & REST algorithm Implementation and improvements

Giorgia Fiscaletti, Peiyun Wu - November 27, 2019

Status of coding and algorithm design

We finished implementing ML-RCS algorithm and REST algorithm with the resource constraints that is defined by ourselves.

We wrote basically everything from scratch. Our code has four parts:

1) In the first part we read the input DFG txt files from the given code MR-LCS and the resources parameters from para_new.txt. The input graph in our program is represented as a dictionary where the keys are the node id and the values are the corresponding node objects. In each node object, the parent and children lists represent the dependencies in the DFG graph, while rfg_parent and rfg_children lists represent the dependencies in the resource flow graph for the REST algorithm. All the information needed (ASAP time, ALAP time, node type, etc.) are included in the attributes of the node class. Then we have a list resources which stores all the parameters of each FUs. The FUs are objects whose attributes are the resource parameters.

2) In the second part, we compute all the parameters needed in the ML-RCS algorithm extended with REST/e-REST. We implemented the following functions:

- ALAP(): function that computes the ALAP time of each node;
- ASAP(): function that computes the ASAP time of each node;
- get_levels(): function that divides all the nodes in the input graphs by levels; it is done by adding them to a utility data structure that is useful to easily compute REST and e-REST. The data structure is a dictionary whose key is the level and value is the list of nodes contained in that level;

3) In the third part, we implemented the REST algorithm. The procedure is implemented as in the slides from the lecture, with a top-down phase and a bottom-up phase. The levels dictionary mentioned before is used to visit the graph both in the first part of the algorithm (top-down) and the second part (bottom-up). In the first part of the REST

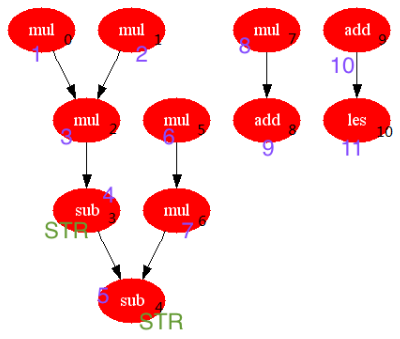
algorithm we build the resource flow graph by adding resource parents and resource children to the nodes.

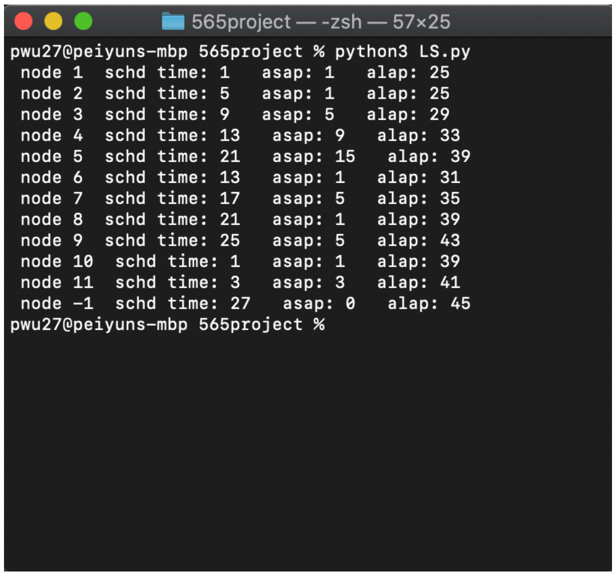
4) Finally, we have the ML-RCS list scheduling, which works on the data structures that we defined. The algorithm is implemented as illustrated during the lectures. It was integrated with the REST/e-REST algorithm, so that ties (in case two nodes eligible for selection have the same slack value and just one of them can be picked) are broken based on the e-REST values of the nodes, i.e. the one with the smallest e-REST is picked.

We also wrote a python script to read the resource constraints from the other group and write the constraints into the given input files. The reason of doing this is because the resource constraint files from the other group is raw files with redundant information. We want our project to have a very neat input files and those input files should be enough to start working on the ML-RCS problem.

Results and comparison

—ML_RCS results (with REST/e_REST)
We tested our algorithm on the inputs graphs. Here is an example:
We use all .txt files as inputs. The example is hal.txt.





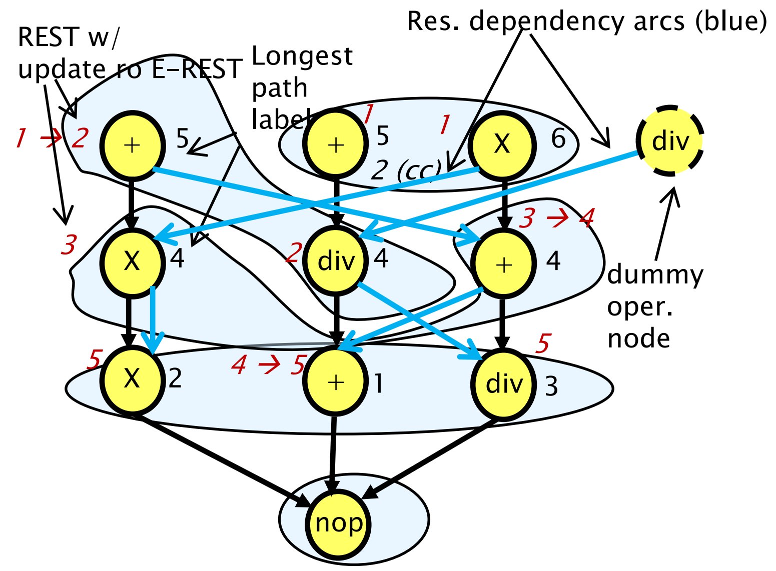
```
pwu27@peiyuns-mbp 565project % python3 LS.py
 node 1  schd time: 1   asap: 1   alap: 25
 node 2  schd time: 5   asap: 1   alap: 25
 node 3  schd time: 9   asap: 5   alap: 29
 node 4  schd time: 13   asap: 9   alap: 33
 node 5  schd time: 21   asap: 15   alap: 39
 node 6  schd time: 13   asap: 1   alap: 31
 node 7  schd time: 17   asap: 5   alap: 35
 node 8  schd time: 21   asap: 1   alap: 39
 node 9  schd time: 25   asap: 5   alap: 43
 node 10  schd time: 1   asap: 1   alap: 39
 node 11  schd time: 3   asap: 3   alap: 41
 node -1  schd time: 27   asap: 0   alap: 45
pwu27@peiyuns-mbp 565project %
```

Purple numbers are ids of nodes
STRs substitute sub nodes because the graph is not consistent with input hal.txt.

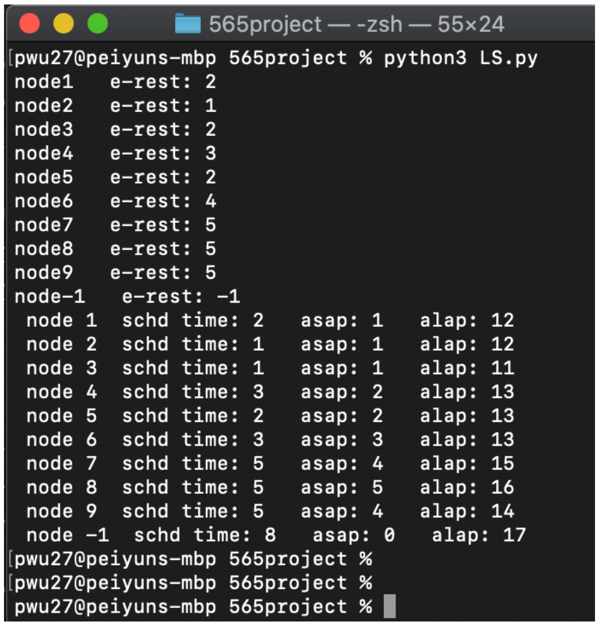
mul.delay=4, str.delay=6, add.delay=2

—REST results

To show the result of REST, we test the algorithm on the graph from the class. For this test we have to modify para_new.txt file to match the delay (. Resources: 1 + (1 cc) , 1 X (2 cc), 1 div (3 cc))



Our result:



```
[pwu27@peiyuns-mbp 565project % python3 LS.py
node1   e-rest: 2
node2   e-rest: 1
node3   e-rest: 2
node4   e-rest: 3
node5   e-rest: 2
node6   e-rest: 4
node7   e-rest: 5
node8   e-rest: 5
node9   e-rest: 5
node-1   e-rest: -1
 node 1  schd time: 2   asap: 1   alap: 12
 node 2  schd time: 1   asap: 1   alap: 12
 node 3  schd time: 1   asap: 1   alap: 11
 node 4  schd time: 3   asap: 2   alap: 13
 node 5  schd time: 2   asap: 2   alap: 13
 node 6  schd time: 3   asap: 3   alap: 13
 node 7  schd time: 5   asap: 4   alap: 15
 node 8  schd time: 5   asap: 5   alap: 16
 node 9  schd time: 5   asap: 4   alap: 14
 node -1  schd time: 8   asap: 0   alap: 17
[pwu27@peiyuns-mbp 565project %
[pwu27@peiyuns-mbp 565project %
pwu27@peiyuns-mbp 565project %
```

Node id is counted from top to bottom from left to right :

1(+)   2(+)   3(x)
4(x)   5(div) 6(+)
7(x)   8(+)   9(div)
-1(dummy node)

Plans for the future of the project

We plan to implement the algorithm with different resource constraints and compare the result. We will also show the comparison of ML_RCS list scheduling result with and without REST so that we will see the influence of REST. Once we finish all the implementation stated earlier, we will extend/improve the algorithm for better results.