Revisiting dynamic DAG scheduling under memory constraints for shared-memory platforms

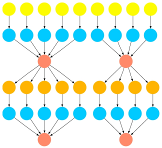
Gabriel Bathie[1] Loris Marchal[1]
Yves Robert[1,2] Samuel Thibault[3]

1:Laboratoire LIP, ENS Lyon, CNRS, Inria and Univ. Lyon, France
2. University Tennessee Knoxville, USA
3: Inria Bordeaux and Univ. Bordeaux, France.

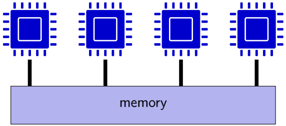
May 18, 2020

Processing DAGs with Limited Memory

Schedule task graphs
with large data:



On a parallel platform
with limited shared memory:



- First option: design a good static scheduler:
  - NP-complete, non-approximable
  - Cannot react to unpredicted changes in the platform or inaccuracies in task timings
- Second option (this work):
  - Limit memory consumption of any dynamic scheduler
  - Target: runtime systems
  - Without impacting parallelism too much

Memory model

Task graphs with:

- Vertex weights w_i: task (estimated) durations
- Edge weights $m_{i,j}$: data sizes

Simple memory model: at the beginning of a task

- Inputs are freed (instantaneously)
- Outputs are allocated

At the end of a task: outputs stay in memory

Memory model

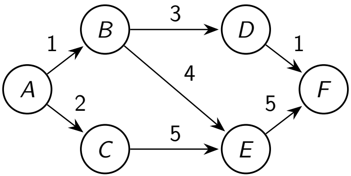
Task graphs with:

- Vertex weights w_i: task (estimated) durations
- Edge weights $m_{i,j}$: data sizes

Simple memory model: at the beginning of a task

- Inputs are freed (instantaneously)
- Outputs are allocated

At the end of a task: outputs stay in memory



Memory used:

Memory model

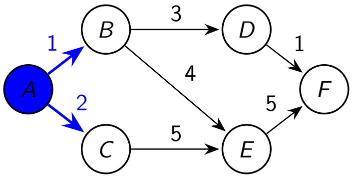
Task graphs with:

- Vertex weights w_i: task (estimated) durations
- Edge weights $m_{i,j}$: data sizes

Simple memory model: at the beginning of a task

- Inputs are freed (instantaneously)
- Outputs are allocated

At the end of a task: outputs stay in memory



Memory used: 1+2=3

Memory model

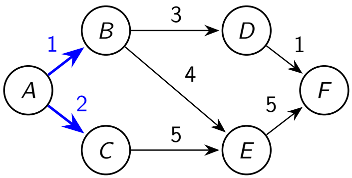
Task graphs with:

- Vertex weights w_i: task (estimated) durations
- Edge weights $m_{i,j}$: data sizes

Simple memory model: at the beginning of a task

- Inputs are freed (instantaneously)
- Outputs are allocated

At the end of a task: outputs stay in memory



Memory used: 1+2=3

Memory model

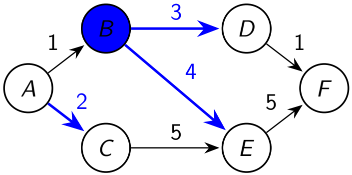
Task graphs with:

- Vertex weights w_i: task (estimated) durations
- Edge weights $m_{i,j}$: data sizes

Simple memory model: at the beginning of a task

- Inputs are freed (instantaneously)
- Outputs are allocated

At the end of a task: outputs stay in memory



Memory used: 3+4+2=9

Memory model

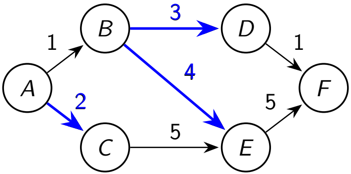
Task graphs with:

- Vertex weights w_i: task (estimated) durations
- Edge weights $m_{i,j}$: data sizes

Simple memory model: at the beginning of a task

- Inputs are freed (instantaneously)
- Outputs are allocated

At the end of a task: outputs stay in memory

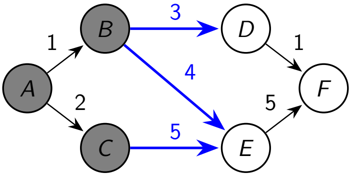


Memory used: 3+4+2=9

Computing the maximum memory peak

Topological cut: (S, T) with:

- S include the source node, T include the target node
- No edge from T to S
- Weight of the cut = weight of all edges from S to T



Any topological cut corresponds to a possible state when all nodes in S are completed or being processed.

Two equivalent questions:

- What is the maximum memory of any parallel execution?
- What is the topological cut with maximum weight?

Computing the maximum topological cut

Predict the maximal memory of any dynamic scheduling

$\Leftrightarrow$

Compute the maximal topological cut

Two algorithms from [Marchal et al, JPDC'19]:

- Linear program + rounding
- Direct algorithm based on MaxFlow/MinCut

Downsides:

- Large running time: $O(|V|^2|E|)$ or solving a LP
- May include edges corresponding to the (parallel) computing of more than p tasks
- Max. Top Cut $\equiv$ maximum memory of any dynamic scheduling with infinite number of processors

Maximum memory with *p*-processors

Definition (p-MaxTopCut)

Given a graph with black/red edges and a number p of processor, what is the maximal weight of a topological cut including at most p red edges ?

Theorem

Computing the p-MaxTopCut is NP-complete

Proof.

Reduction from k-MSI □

Case of Series-Parallel graphs

Pseudo Polynomial Time algorithm:

$$M(Edge(m,r),k) = m, \forall k \geq 1, \forall r \in \{True, False\} \tag{1}$$

$$M(Edge(m, True), 0) = -\infty \tag{2}$$

$$M(Edge(m, False), 0) = m \tag{3}$$

$$M(Serie(G_1, G_2), k) = \max\{M(G_1, k), M(G_2, k)\} \tag{4}$$

$$M(Par(G_1, G_2), k) = \max_{j=0\ldots k}\{M(G_1, j) + M(G_2, k-j)\} \tag{5}$$

Compute $M(H, k)$ for all H, for all $k = 0 \ldots p$. With memoization: runs in time $\mathcal{O}(|E|p^2)$.

General Case: ILP formulation for p-MaxTopCut

The following linear program solves the problem exactly:

$$\max \sum_{(i,j)\in E} m_{i,j} d_{i,j} \tag{6}$$

$$\forall (i,j) \in E, \quad d_{i,j} = p_i - p_j \tag{7}$$

$$\forall (i,j) \in E, \quad d_{i,j} \geq 0 \tag{8}$$

$$p_s = 1, \qquad p_t = 0 \tag{9}$$

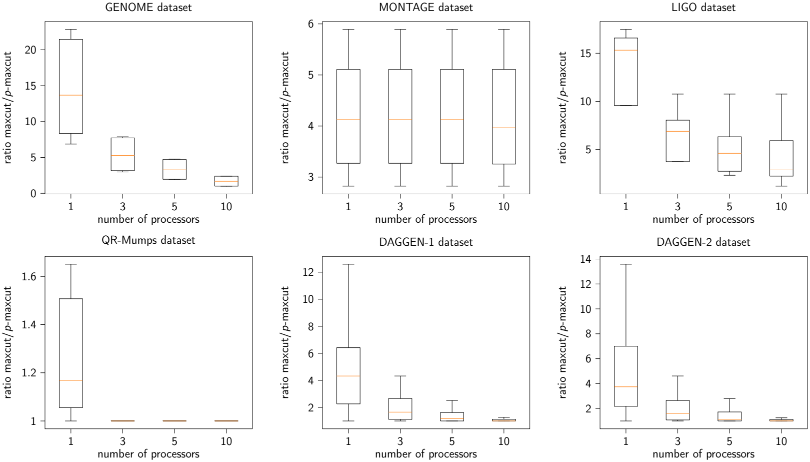
$$\sum_{(i,j)\in E} isred_{i,j} d_{i,j} \leq p \tag{10}$$

$$\forall i, p_i \in \{0,1\} \tag{11}$$

Heuristic relaxation: change Equation (11) to $\forall i, p_i \in [0,1]$.
$\rightarrow$ Linear program over rational numbers, efficiently solvable.

Simulation and Results

Measuring the gap between MaxTopCut ($p = \infty$) vs. p-MaxTopCut



Code available at `https://github.com/GBathie/PMaxcut`.

Conclusion

Contributions

- MaxTopCut (former approach) significantly overestimates the maximum memory compared to proposed p-MaxTopCut
- Computing pMaxTopCut is NP-hard ☹
- Proposed heuristic (Linear Program) very efficient to compute p-MaxTopCut in practice (see paper) ☺

Future work

- Design efficient strategies to reduce peak memory with p processors
- Concentrate on special class of dynamic schedulers, that favor low memory-consuming tasks