Sequential Scheduling of Dataflow Graphs for Memory Peak Minimization

Pascal Fradet, Alain Girault, Alexandre Honorat

April 27, 2023

Inria Grenoble — SPADES team

Context: DF4DL post-doc

DF4DL: DataFlow for Deep Learning
Deep Neural *Networks* $\Rightarrow$ DataFlow?

Context: DF4DL post-doc

DF4DL: DataFlow for Deep Learning
Deep Neural *Networks* $\Rightarrow$ DataFlow?

DataFlow, what for?
May compute guarantees on:

- liveness
- throughput
- memory usage
- real-time properties

Statically (e.g. SDF) or dynamically (e.g. RDF)

Context: DF4DL post-doc

DF4DL: DataFlow for Deep Learning
Deep Neural *Networks* $\Rightarrow$ DataFlow?

DataFlow, what for?
May compute guarantees on:

- liveness
- throughput
- memory usage
- real-time properties

Statically (e.g. SDF) or dynamically (e.g. RDF)

Our memory peak problem

Input
A directed acyclic task graph, with memory costs:

- on each edge (data I/O)
- on each node (computation)

Sequential execution without preemption, no timing properties.

Our memory peak problem

Input
A directed acyclic task graph, with memory costs:

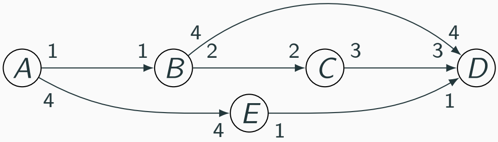
- on each edge (data I/O)
- on each node (computation)

Sequential execution without preemption, no timing properties.

Output

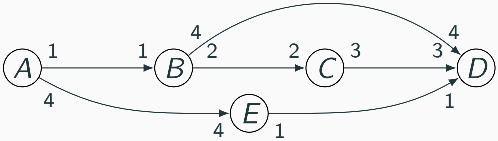
- a schedule minimizing the memory peak
- its corresponding memory peak

Example



When to execute E ?

Example



When to execute E **?** $A; E; B; C; D$

Previous known results

Complexity

[Sethi'73] PebbleGame is NP-complete (time)

[KS'74] Generate all Linear Extensions (linear in space)

[BW'91] Counting Linear Extensions is #P-complete ($\supseteq$ NP time)

Previous known results

Complexity

[Sethi'73] PebbleGame is NP-complete (time)

[KS'74] Generate all Linear Extensions (linear in space)

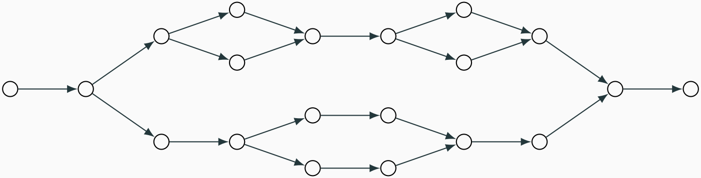
[BW'91] Counting Linear Extensions is #P-complete ($\supseteq$ NP time)

Specific cases

[Liu'87] trees in quadratic time

[KLMU'18] Series-Parallel DAG in cubic time

Example of an SP-DAG



(a sort of recursive fork-join graph)

“Optimal” graph transformations (contrib. 1)

Why graph transformations?

Key idea: reduce the combinatorial explosion by...

- reducing the number of nodes
- increasing the number of dependencies
  (from partial order to total order)

Why graph transformations?

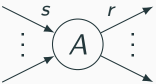
Key idea: reduce the combinatorial explosion by...

- reducing the number of nodes
- increasing the number of dependencies
  (from partial order to total order)

"Optimal" transformations
$\hookrightarrow$ preserve the minimal memory peak

From task graph to schedule graphs

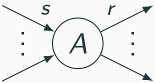
Internal representation: node Peak and Impact



Node $A^{\binom{\text{peak}}{\text{impact}}}$ produces r tokens and consumes s tokens.

From task graph to schedule graphs

Internal representation: node Peak and Impact



Node $A^{\binom{\text{peak}}{\text{impact}}}$ produces r tokens and consumes s tokens.

Initial values of Peak and Impact
$\text{impact} = r - s \in \mathbb{Z}$ and $\text{peak} \in \mathbb{N}$, or peak variants:

- $A^{\binom{\max(0,r-s)}{r-s}}$ in the Consume-Before-Produce model
- $A^{\binom{r}{r-s}}$ in the Produce-Before-Consume model

(no further need to edge attribute)

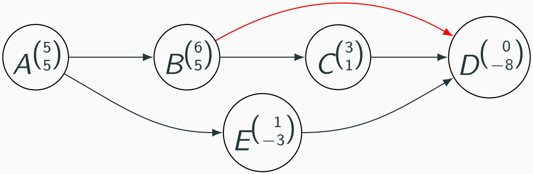
Peak of a schedule

Peak and Impact of a node sequence
Can be applied to any schedule.

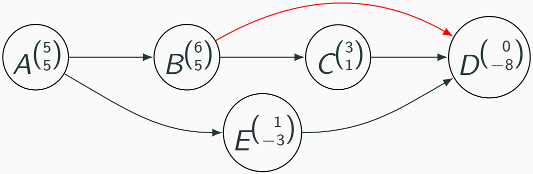
$$A^{\binom{p_a}{i_a}}; B^{\binom{p_b}{i_b}} = (A; B)^{\binom{max(p_a, p_b + i_a)}{i_a + i_b}} \tag{PI}$$

Theorem
Operation (PI) is associative.

Transitive reduction



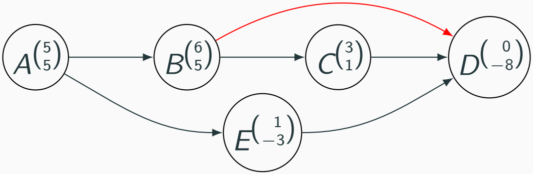
Transitive reduction



Simply remove all transitive edges.

(e.g. from B to D in red)

Transitive reduction

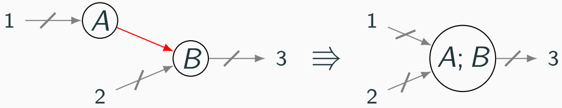


Simply remove all transitive edges.

(e.g. from B to D in red)

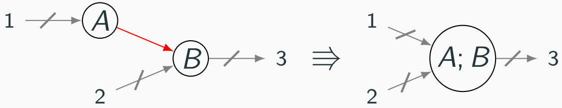
Does not modify node peak/impact!

Clustering rules (C1-C2): single successor/predecessor



$$\mathrm{Succ}(A) = \{B\} \;\wedge\; (i_A \geq 0) \;\wedge\; (p_B + i_A \geq p_A) \qquad \text{(C1)}$$

Clustering rules (C1-C2): single successor/predecessor



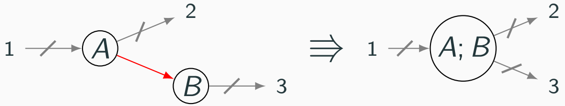
$$\mathrm{Succ}(A) = \{B\} \ \wedge \ (i_A \geq 0) \ \wedge \ (p_B + i_A \geq p_A) \qquad \text{(C1)}$$

Reduce the number of nodes!

Clustering rules (C1-C2): single successor/predecessor

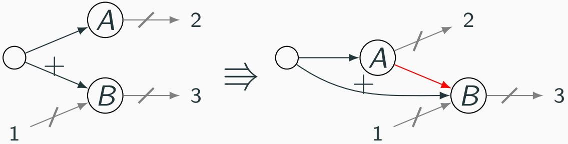
$$\mathrm{Succ}(A) = \{B\} \;\wedge\; (i_A \geq 0) \;\wedge\; (p_B + i_A \geq p_A) \qquad \text{(C1)}$$

Reduce the number of nodes!



$$\mathrm{Pred}(B) = \{A\} \;\wedge\; (i_B \leq 0) \;\wedge\; (p_A \geq p_B + i_A) \qquad \text{(C2)}$$

Sequentialization rule (S1): common predecessors

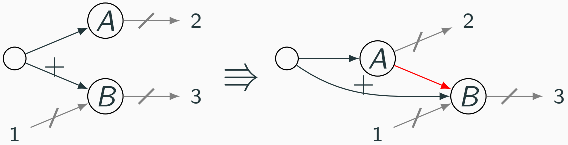


$$\mathrm{Pred}(A) \subseteq \mathrm{Pred}^{+}(B) \wedge (i_A \leq 0) \wedge (p_B \geq p_A) \qquad \text{(S1)}$$

Pred^{+} is the set of ancestors,

i.e. predecessors in transitive closure

Sequentialization rule (S1): common predecessors



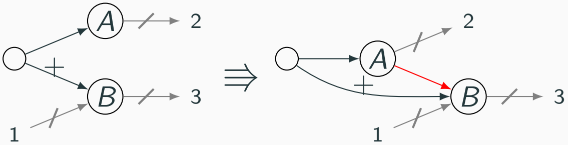
$$\mathrm{Pred}(A) \subseteq \mathrm{Pred}^+(B) \wedge (i_A \leq 0) \wedge (p_B \geq p_A) \qquad \text{(S1)}$$

Pred^+ is the set of ancestors,

i.e. predecessors in transitive closure

Increase the number of dependencies!

Sequentialization rule (S1): common predecessors



$$\mathrm{Pred}(A) \subseteq \mathrm{Pred}^{+}(B) \wedge (i_A \leq 0) \wedge (p_B \geq p_A) \qquad \text{(S1)}$$

Pred^{+} is the set of ancestors,
i.e. predecessors in transitive closure

Increase the number of dependencies!

(similar rule for common successors with Succ^{+})

Global algorithm

```
/* Takes a schedule graph G and compresses it until none of
   the transformations apply                                */
changed := false;
repeat
    repeat
        repeat
            clustering(G); ▷ O(n)
        until ¬ changed;
        basic_sequentialization(G); ▷ O(n^2)
    until ¬ changed;
    complete_sequentialization(G); ▷ O(n^3)
    transitive_reduction(G); ▷ O(n^3)
until ¬ changed;
```

Theoretical results

In general
Compressed graph always ensures at least one schedule having the minimal peak. (worst-case complexity: quartic time $\mathcal{O}(n^4)$)

Theoretical results

In general
Compressed graph always ensures at least one schedule having the minimal peak. (worst-case complexity: quartic time $\mathcal{O}(n^4)$)

Specific cases
If reduced to a single node, it contains one of the schedule ensuring minimal peak. This includes:

trees compressed to a single node (in quadratic time)

SP-DAG compressed to a single node (in cubic time)

Branch and Bound search (contrib. 2)

General idea of Branch and Bound

Explore linear extensions of the graph...
New *branch* at each scheduled node

(storing all the ready unexplored ones),

and continue with Depth-First-Search (DFS).

General idea of Branch and Bound

Explore linear extensions of the graph...
New *branch* at each scheduled node

(storing all the ready unexplored ones),

and continue with Depth-First-Search (DFS).

...but not all
New *bound* at each schedule having minimal peak

(stop DFS on next nodes implying a higher peak).

$\hookrightarrow$ backtrack to previous rank if no more unexplored nodes

General idea of Branch and Bound

Explore linear extensions of the graph...
New *branch* at each scheduled node

(storing all the ready unexplored ones),

and continue with Depth-First-Search (DFS).

...but not all
New *bound* at each schedule having minimal peak

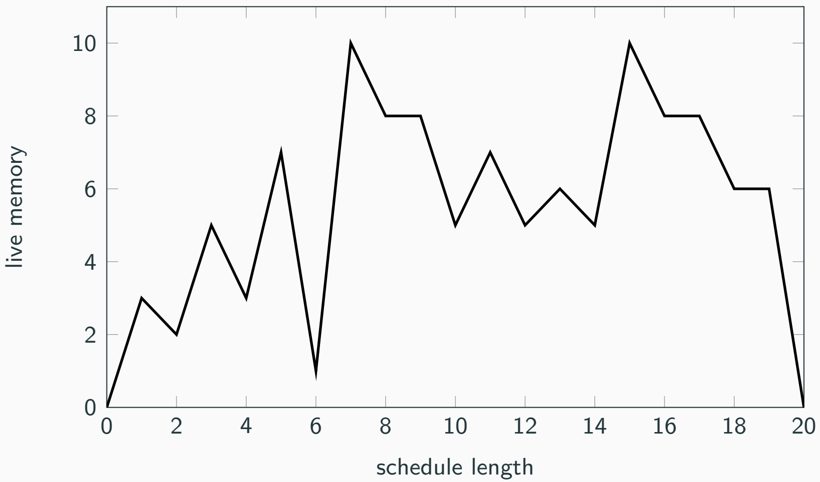
(stop DFS on next nodes implying a higher peak).

$\hookrightarrow$ backtrack to previous rank if no more unexplored nodes

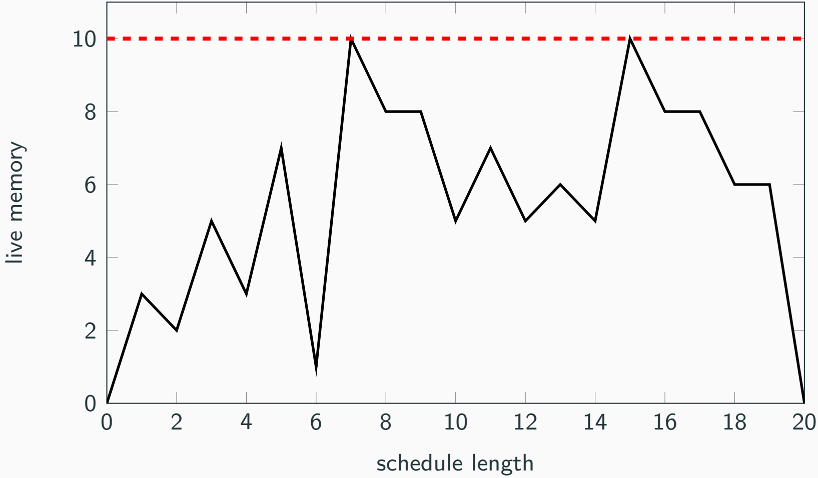
Optimizations (contributions)

- a longer backtrack
- a smaller ready list

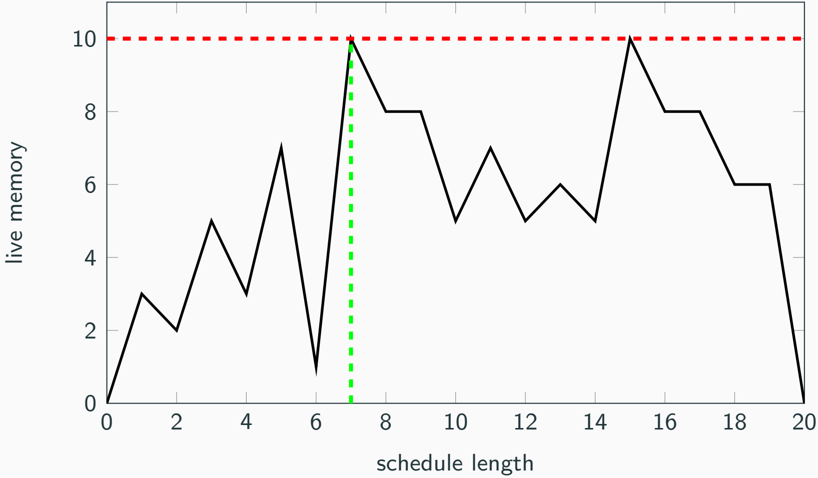
Peak backtrack optimization



Peak backtrack optimization

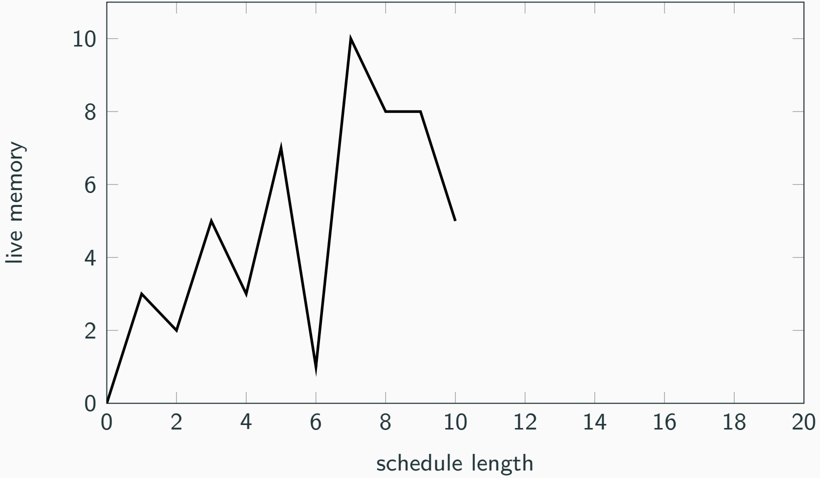


Peak backtrack optimization

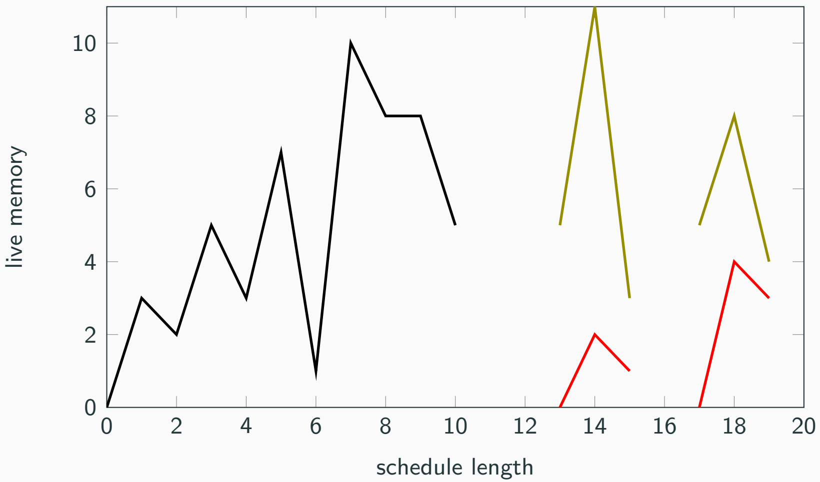


Backtrack until first peak!

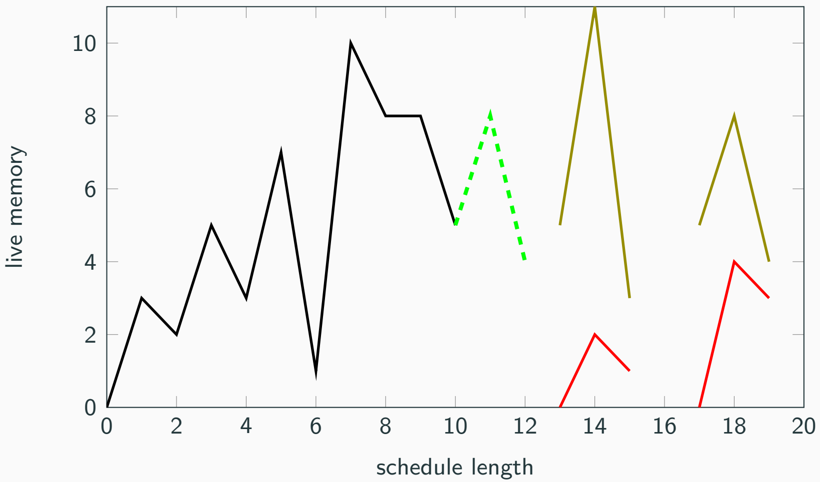
Negative impact optimization



Negative impact optimization



Negative impact optimization



Negative impact node first!

(smaller branching factor)

Practical results

Instance size: our Branch and Bound
Graph of ≈ 50 nodes always solved in ≤ 1 sec.

Time generally explodes if more than 100 nodes,

but B&B quickly finds at least one solution.

Practical results

Instance size: our Branch and Bound
Graph of ≈ 50 nodes always solved in ≤ 1 sec.

Time generally explodes if more than 100 nodes,

but B&B quickly finds at least one solution.

Instance size: all linear extensions
≈ 15 nodes to be solved in ≤ 1 sec.

Practical results

Instance size: our Branch and Bound
Graph of ≈ 50 nodes always solved in ≤ 1 sec.

Time generally explodes if more than 100 nodes,

but B&B quickly finds at least one solution.

Instance size: all linear extensions
≈ 15 nodes to be solved in ≤ 1 sec.

Important parameter: sort function for ready list
Will impact on the peak quality of first DFS.

Experiments

Memory peak for Satellite

satellite	$\|G\|$	[RWM'95]	[MB'01]	[KLMU'18]	[ours]	sec.
flat SAS	22	1,920	—	**1,680**	**1,680**	0.01
SDF	4,515	—	991	**960**	**960**	24.5

Memory peak for Satellite

satellite	$\lvert G \rvert$	[RWM'95]	[MB'01]	[KLMU'18]	[ours]	sec.
flat SAS	22	1,920	—	**1,680**	**1,680**	0.01
SDF	4,515	—	991	**960**	**960**	24.5

Previous runtime for flat SAS [RWM'95]:

4 days (and wrong result) with ILP

Memory peak for Satellite

satellite	$\lvert G\rvert$	[RWM'95]	[MB'01]	[KLMU'18]	[ours]	sec.
flat SAS	22	1,920	—	**1,680**	**1,680**	0.01
SDF	4,515	—	991	**960**	**960**	24.5

Previous runtime for flat SAS [RWM'95]:

4 days (and wrong result) with ILP

Peaks of [MB'01] and [KLMU'18] are over-estimated.

Memory peak for QMF Filterbank

Filterbank	$\lvert G \rvert$	[MB'01]	[KLMU'18]	[ours]	$\lvert G^{\mathbf{C}} \rvert$	sec.
qmf23_2d	90	22	27	**14**	1	0.07
qmf23_3d	378	63	81	**32**	1	0.6
qmf23_5d	5,346	492	709	**248**	1	445.4
qmf12_2d	40	9	10	**7**	1	0.02
qmf12_3d	112	16	20	**11**	1	0.06
qmf12_5d	704	58	79	**35**	1	1.7
qmf235_2d	250	55	78	**24**	24	0.3
qmf235_3d	1,750	240	189	**47**†	285	T/O
qmf235_5d	68,750	5,690	—	**3,401**†	—	T/O

Gray = Wrong qmf version but similar peaks

(always reduced to 1 node on correct version, no T/O)

Conclusion

Conclusion

New peak-preserving transformations
Always compress trees and SP-DAG into a single node,

and many more graphs too (at worst quartic time).

Conclusion

New peak-preserving transformations
Always compress trees and SP-DAG into a single node,

and many more graphs too (at worst quartic time).

New optimal Branch and Bound
Handle instances up to 50 nodes in a few seconds.

Conclusion

New peak-preserving transformations
Always compress trees and SP-DAG into a single node,
and many more graphs too (at worst quartic time).

New optimal Branch and Bound
Handle instances up to 50 nodes in a few seconds.

Future work

- checkpointing a.k.a. rematerialization
  (a.k.a. reversible PebbleGame?)
- apply same kind of transformations to other problems?

References

[Sethi'73] *Complete Register Allocation Problems*, R. Sethi (1973)

[KS'74] *A structured program to generate all topological sorting arrangements*, D. Knuth and J. L. Szwarcfiter (1974)

[BW'91] *Counting Linear Extensions is #P-Complete*, G. Brightwell and P. Winkler (1991)

[Liu'87] *An Application of Generalized Tree Pebbling to Sparse Matrix Factorization*, J. W. H. Liu (1987)

[KLMU'18] *Scheduling series-parallel task graphs to minimize peak memory*, E. Kayaaslan, T. Lambert, L. Marchal and B. Uçar (2018)

[RWM'95] *Scheduling for optimum data memory compaction in block diagram oriented software synthesis*, S. Ritz, M. Willems and H. Meyr (1995)

[MB'01] *Shared buffer implementations of signal processing systems using lifetime analysis techniques*, P.K. Murthy and S.S. Bhattacharyya (2001)