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

DataFlow, what for?
May compute guarantees on:

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

Statically (e.g. SDF) or dynamically (e.g. RDF)
DF4DL: DataFlow for Deep Learning
Deep Neural Networks ⇒ 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
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 (⊆ NP time)

Specific cases

[Liu’87] trees in quadratic time

[KLMU’18] Series-Parallel DAG in cubic 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
preserve the minimal memory peak
Internal representation: node Peak and Impact

Node $A^{(\text{peak})}$ produces $r$ tokens and consumes $s$ tokens.
Internal representation: node Peak and Impact

Node $A_{\text{peak}}^{\text{impact}}$ 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:

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

(no further need to edge attribute)
Peak and Impact of a node sequence
Can be applied to any schedule.

\[ A^{(p_a)}_{i_a}; B^{(p_b)}_{i_b} = (A; B)^{\max(p_a, p_b+i_a)}_{i_a+i_b} \] (PI)

**Theorem**
Operation (PI) is associative.
Transitive reduction

Simply remove all transitive edges. (e.g. from B to D in red)

Does not modify node peak/impact!
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

\[
\text{Succ}(A) = \{B\} \land (i_A \geq 0) \land (p_B + i_A \geq p_A) \quad (C1)
\]
Clustering rules (C1-C2): single successor/predecessor

\[ \text{Succ}(A) = \{B\} \land (i_A \geq 0) \land (p_B + i_A \geq p_A) \quad \text{(C1)} \]

Reduce the number of nodes!
Clustering rules (C1-C2): single successor/predecessor

Succ($A$) = \{ $B$ \} \land (i_A \geq 0) \land (p_B + i_A \geq p_A) \quad (C1)

Reduce the number of nodes!

Pred($B$) = \{ $A$ \} \land (i_B \leq 0) \land (p_A \geq p_B + i_A) \quad (C2)
Sequentialization rule (S1): common predecessors

\[ \text{Pred}(A) \subseteq \text{Pred}^+(B) \land (i_A \leq 0) \land (p_B \geq p_A) \]  

(S1)

Pred\(^+\) is the set of ancestors, i.e. predecessors in transitive closure
Sequentialization rule (S1): common predecessors

\[
\text{Pred}(A) \subseteq \text{Pred}^+(B) \land (i_A \leq 0) \land (p_B \geq p_A) \tag{S1}
\]

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

Increase the number of dependencies!
Sequentialization rule (S1): common predecessors

\[ \text{Pred}(A) \subseteq \text{Pred}^+(B) \land (i_A \leq 0) \land (p_B \geq p_A) \] (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 \*/

1 $\text{changed} := \text{false}$;
2 repeat
3 \hspace{1em} repeat
4 \hspace{2em} repeat
5 \hspace{3em} clustering($G$); $\triangleright \mathcal{O}(n)$
6 \hspace{2em} until $\neg \text{changed}$;
7 \hspace{2em} basic\_sequentialization($G$); $\triangleright \mathcal{O}(n^2)$
8 \hspace{2em} until $\neg \text{changed}$;
9 \hspace{1em} complete\_sequentialization($G$); $\triangleright \mathcal{O}(n^3)$
10 \hspace{1em} transitive\_reduction($G$); $\triangleright \mathcal{O}(n^3)$
11 \hspace{1em} until $\neg \text{changed}$;
Theoretical results

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

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

**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)
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).

→ 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).
← backtrack to previous rank if no more unexplored nodes

Optimizations (contributions)
- a longer backtrack
- a smaller ready list
Peak backtrack optimization

![Graph showing the relationship between live memory and schedule length.](image-url)
Peak backtrack optimization

![Graph showing live memory over schedule length](image-url)
Peak backtrack optimization

Backtrack until first peak!
Negative impact optimization

![Graph showing the relationship between schedule length and live memory. The graph indicates a peak in live memory at schedule length 6, followed by a decline.](image)
Negative impact optimization

![Graph showing live memory versus schedule length](image-url)
Negative impact optimization

Negative impact node first!
(smaller branching factor)
Practical results

**Instance size: our Branch and Bound**
Graph of \( \approx 50 \) nodes always solved in \( \leq 1 \) 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 $\approx 50$ nodes always solved in $\leq 1$ sec.
Time generally explodes if more than 100 nodes, but B&B quickly finds at least one solution.

**Instance size: all linear extensions**
$\approx 15$ nodes to be solved in $\leq 1$ sec.
**Practical results**

**Instance size: our Branch and Bound**
Graph of $\approx 50$ nodes always solved in $\leq 1$ sec.
Time generally explodes if more than 100 nodes, but B&B quickly finds at least one solution.

**Instance size: all linear extensions**
$\approx 15$ nodes to be solved in $\leq 1$ 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 | $|G|$ | [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 |

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 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 |

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    | $|G|$ | [MB’01] | [KLMU’18] | [ours] | $G^C$ | 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).
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)


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


