Parallel scheduling of task graphs under minimal memory constraints

Pascal Fradet, Alain Girault, Alexandre Honorat

& — Univ. Grenoble Alpes, France

SYNCHRON 2025 — Aussois — November 26th, 2025

Motivation

Memory constraints:

- All computing systems operate under some memory constraint.
- Often, one can buy more memory or limit the problem size.

Motivation

Memory constraints:

- All computing systems operate under some memory constraint.
- Often, one can buy more memory or limit the problem size.

But sometimes:

- Supply is too small: embedded systems, IoTs, wearable systems, ...
- Demand is too large: DNNs, scientific computing, ...

Motivation

Memory constraints:

- All computing systems operate under some memory constraint.
- Often, one can buy more memory or limit the problem size.

But sometimes:

- Supply is too small: embedded systems, IoTs, wearable systems, ...
- Demand is too large: DNNs, scientific computing, ...

Our goal:

Dynamic parallel scheduling of dataflow graphs under memory constraint

Our memory peak problem

Input

G A Directed Acyclic Task Graph (DAG), with memory costs attributes (on the edges) and execution times (on the nodes/tasks).

p The number of available homogeneous processors with shared memory.

$\mathbf{\Pi}$ The memory constraint.

Output

A parallel schedule **S** of **G** on the **p** processors, respecting the constraint memory peak(**S**) $\leq \mathbf{\Pi}$ and minimizing the execution time $C_{\max}$ (**S**).

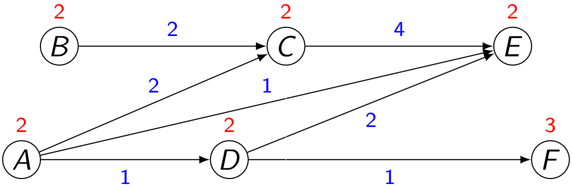
The memory peak(**S**) is the maximum amount of memory used during **S**.

Outline

- System model: Task graphs, memory peak, and schedule graphs
- First contribution: Optimal sequential scheduling for the memory peak
- Second contribution: Dynamic memory-aware parallel list scheduling
- Experimental evaluation: Success rate, speedup, and execution time
- Conclusion and future work

System model

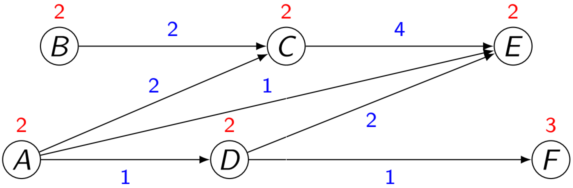
Task graph example



number of tokens

execution time

Task graph example



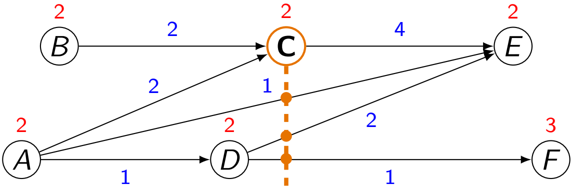
number of tokens

execution time

Produce-Before-Consume (PBC) shared memory model [MB'01]

When a task executes, first it reads its input tokens, it performs its execution, then it allocates memory for its output tokens (its result) and finally it frees the memory occupied by its input tokens.

Task graph example



number of tokens

execution time

memory peak$(A; B; D; \mathbf{C}; E; F)$ = cut weight

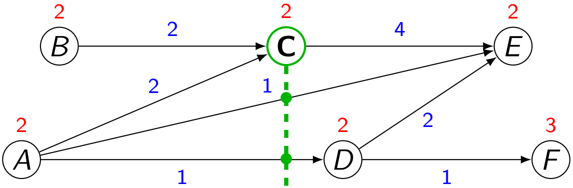
Produce-Before-Consume (PBC) shared memory model [MB'01]

When a task executes, first it reads its input tokens, it performs its execution, then it allocates memory for its output tokens (its result) and finally it frees the memory occupied by its input tokens.

Memory peak of two sequential schedules:

$$\text{memory peak}(A; B; D; \mathbf{C}; E; F) = 8 + 1 + 2 + 1 = 12$$

Task graph example



memory peak$(A; B; \mathbf{C}; D; E; F)$ = cut weight

number of tokens

execution time

Produce-Before-Consume (PBC) shared memory model [MB'01]

When a task executes, first it reads its input tokens, it performs its execution, then it allocates memory for its output tokens (its result) and finally it frees the memory occupied by its input tokens.

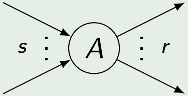
Memory peak of two sequential schedules:

$$\text{memory peak}(A; B; D; \mathbf{C}; E; F) = 8 + 1 + 2 + 1 = 12$$

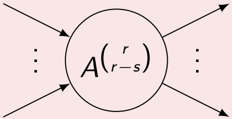
✔ $\text{memory peak}(A; B; \mathbf{C}; D; E; F) = 8 + 1 + 1 = 10$ ✔

Handling memory models: PBC, CBP, ... [MB'01]

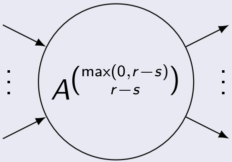
Arbitrary node/task in the input graph



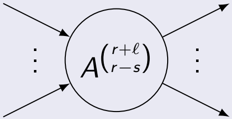
PBC model



CBP model



PBC with ℓ local variables model

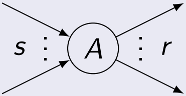


First contribution:

Memory-optimal sequential schedule
[FGH'24]

First transformation on **G**: put the memory attributes on the nodes

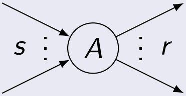
Initial task graph **G**: production/consumption per edge



PBC: Task A produces (allocates) its r output tokens and then consumes (frees) its s input tokens.

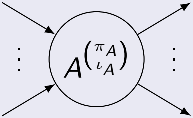
First transformation on **G**: put the memory attributes on the nodes

Initial task graph **G**: production/consumption per edge



PBC: Task A produces (allocates) its r output tokens and then consumes (frees) its s input tokens.

Schedule graph **G**′: peak and impact per task (also per sequence)

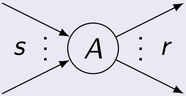


Notation $A^{\binom{\text{peak}}{\text{impact}}}$ with:

- peak $\pi_A = r \in \mathbb{N}$
- impact $\iota_A = r - s \in \mathbb{Z}$

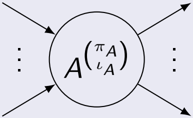
First transformation on **G**: put the memory attributes on the nodes

Initial task graph **G**: production/consumption per edge



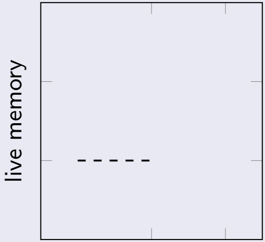
PBC: Task A produces (allocates) its r output tokens and then consumes (frees) its s input tokens.

Schedule graph **G′**: peak and impact per task (also per sequence)



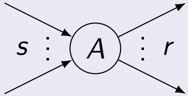
Notation $A^{\binom{\text{peak}}{\text{impact}}}$ with:

- peak $\pi_A = r \in \mathbb{N}$
- impact $\iota_A = r - s \in \mathbb{Z}$



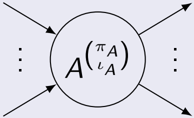
First transformation on **G**: put the memory attributes on the nodes

Initial task graph **G**: production/consumption per edge



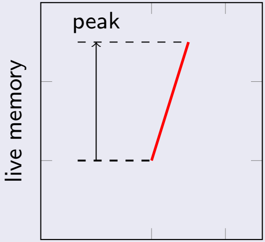
PBC: Task A produces (allocates) its r output tokens and then consumes (frees) its s input tokens.

Schedule graph **G′**: peak and impact per task (also per sequence)



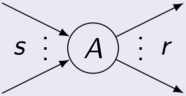
Notation $A^{\binom{\text{peak}}{\text{impact}}}$ with:

- peak $\pi_A = r \in \mathbb{N}$
- impact $\iota_A = r - s \in \mathbb{Z}$



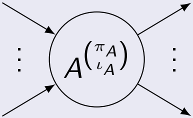
First transformation on **G**: put the memory attributes on the nodes

Initial task graph **G**: production/consumption per edge



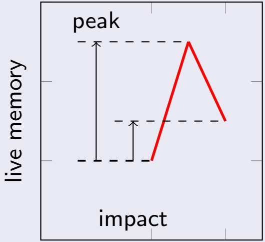
PBC: Task A produces (allocates) its r output tokens and then consumes (frees) its s input tokens.

Schedule graph **G′**: peak and impact per task (also per sequence)



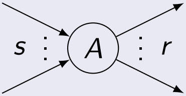
Notation $A^{\binom{\text{peak}}{\text{impact}}}$ with:

- peak $\pi_A = r \in \mathbb{N}$
- impact $\iota_A = r - s \in \mathbb{Z}$



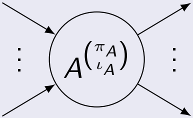
First transformation on **G**: put the memory attributes on the nodes

Initial task graph **G**: production/consumption per edge



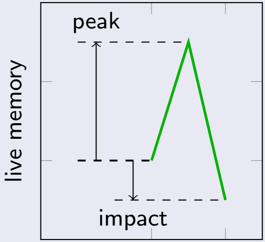
PBC: Task A produces (allocates) its r output tokens and then consumes (frees) its s input tokens.

Schedule graph **G′**: peak and impact per task (also per sequence)



Notation $A^{\binom{\text{peak}}{\text{impact}}}$ with:

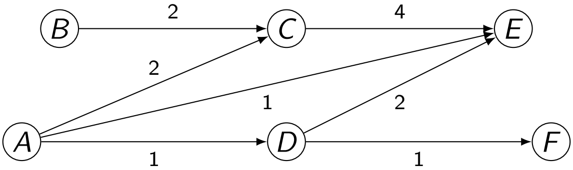
- peak $\pi_A = r \in \mathbb{N}$
- impact $\iota_A = r - s \in \mathbb{Z}$



Example: A task graph $\mathbf{G}$ and its schedule graph $\mathbf{G}'$

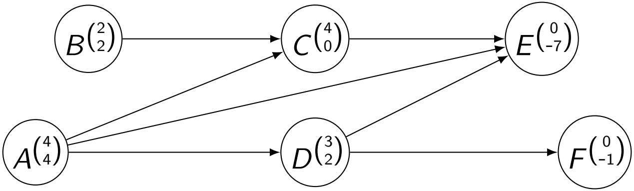
Initial task graph:

$\mathbf{G}$ =



Schedule graph (PBC model):

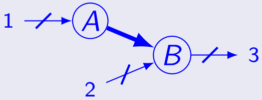
$\mathbf{G}'$ =



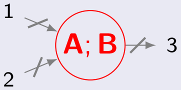
Works also when a node contains a sequence of tasks (a sub-schedule).

Second transformation on $\mathbf{G}'$: Peak-preserving local compression rules [FGH'24]

Clustering rule (C1) (there is also a dual rule (C2))

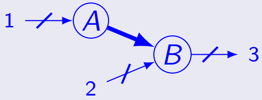


$\wedge\ (\iota_A \geq 0) \wedge (\pi_B + \iota_A \geq \pi_A) \Rrightarrow$

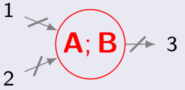


Second transformation on $\mathbf{G}'$: Peak-preserving local compression rules [FGH'24]

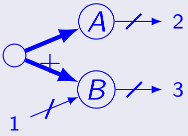
Clustering rule (C1) (there is also a dual rule (C2))



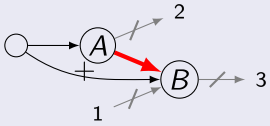
$\wedge\ (\iota_A \geq 0) \wedge (\pi_B + \iota_A \geq \pi_A) \Rrightarrow$



Sequentialization rule (S1) (there is also a dual rule (S2))

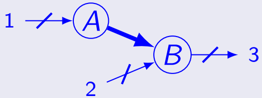


$\wedge(\iota_A \leq 0) \wedge (\pi_B \geq \pi_A) \Rrightarrow$

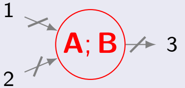


Second transformation on $\mathbf{G}'$: Peak-preserving local compression rules [FGH'24]

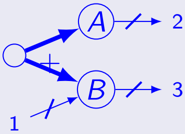
Clustering rule (C1) (there is also a dual rule (C2))



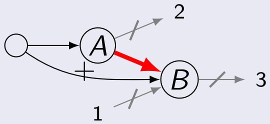
$\wedge \; (\iota_A \geq 0) \wedge (\pi_B + \iota_A \geq \pi_A) \Rrightarrow$



Sequentialization rule (S1) (there is also a dual rule (S2))



$\wedge (\iota_A \leq 0) \wedge (\pi_B \geq \pi_A) \Rrightarrow$



Properties

These four rules reduce the number of sequential schedules of $\mathbf{G}'$.
Simple topological and arithmetic conditions that can be checked locally.

Results on these transformations [FGH'24]

Properties

Memory peak preservation: Any compressed graph $\mathbf{G}''$ always admits at least one minimal memory peak schedule $\mathbf{S_o}$.

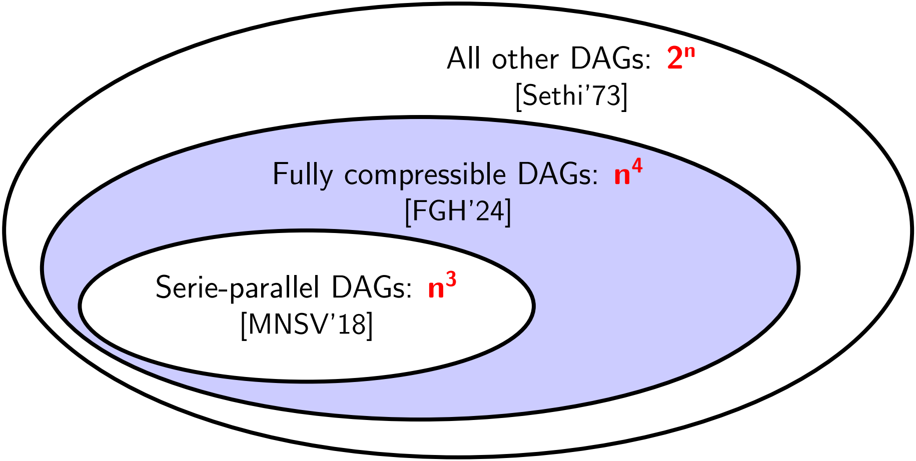
Computational complexity: Quartic in the size n of $\mathbf{G}'$: $\mathcal{O}(n^4)$.

Minimal memory peak schedule: For any graph $\mathbf{G}'$ compressed into a single node graph $\mathbf{G}''$ (in particular every SP-DAGs), the single node of $\mathbf{G}''$ contains a schedule with the minimal memory peak of $\mathbf{G}$.

Optimized Branch and Bound algorithm

For all graphs not compressed into a single node.

From polynomial to exponential complexity



Graph transformation algorithm

Compresses the schedule graph G until no transformation rule applies

repeat
  repeat
    repeat
      clustering(G); ▷ Rules (C1) and (C2), $\mathcal{O}(n)$
    until ¬ *changed*;
    basic_sequentialization(G); ▷ Rules (S1i) and (S2i), $\mathcal{O}(n^2)$
  until ¬ *changed*;
  complete_sequentialization(G); ▷ Rules (S1) and (S2), $\mathcal{O}(n^3)$
  transitive_reduction(G); ▷ $\mathcal{O}(n^3)$
until ¬ *changed*;

(S1i) and (S2i) are the "immediate successor" variants (much cheaper)

Experimental results

QMF benchmark [MB01] in SDF [LM87], fully expanded

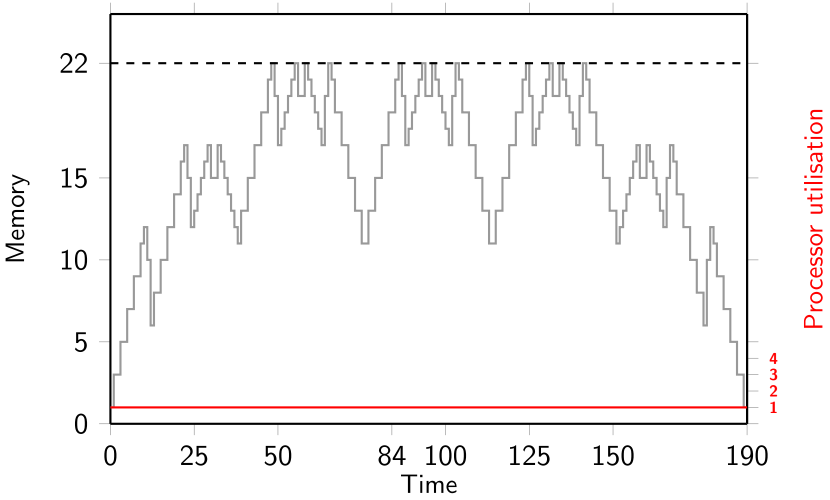
It is a signal processing application with parameterized size.
$\hookrightarrow$ task graphs up to $50,000$ tasks (no execution times provided)

filterbank	$\|G\|$	[MB'01]	[KLMU'18]	[ours]	sec.
qmf23_2d	78	22	18	**13**	0.007
qmf23_3d	324	63	53	**31**	0.06
qmf23_5d	4,536	492	405	**247**	6.7
qmf12_2d	40	9	10	**7**	0.003
qmf12_3d	112	16	20	**11**	0.009
qmf12_5d	704	58	79	**35**	0.1
qmf235_2d	190	55	45	**22**	0.03
qmf235_3d	1,300	240	133	**47**	0.7
qmf235_5d	50,000	5,690	1,190	**272**	802.5

Second contribution:

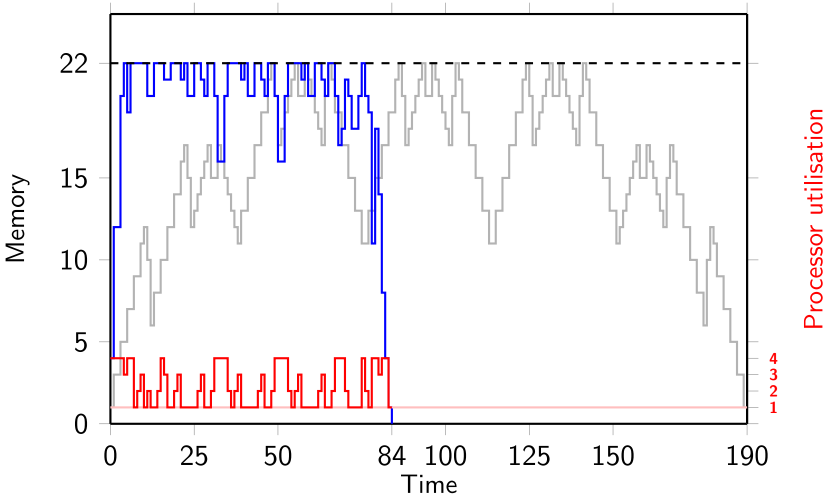
Dynamic memory-aware parallel list scheduling [FGH'25]

Goal: Execute tasks in parallel as long as we are below Π



From a "wide mountain" (optimal sequential schedule) ...

Goal: Execute tasks in parallel as long as we are below Π



From a “wide mountain” (optimal sequential schedule) ...

to a “narrow plateau” (parallel schedule)

Dynamic memory-aware ready list parallel scheduler

if (*a task completes*) **then**
    $p \leftarrow$ nbIdleProcessors();
    $\mathcal{L}_{ready} \leftarrow$ getReadyList() ; ▷ (sorted) list of ready tasks
    while ($p > 0$ **and** size($\mathcal{L}_{ready}$) > 0) **do**
        $X \leftarrow$ pop($\mathcal{L}_{ready}$);
        if canSched(X, Π) **then** ▷ check that if X is scheduled now, the new memory peak is $\leq \Pi$
            $p \leftarrow p - 1$;
            launch(X); ▷ X is launched immediately on the first idle processor
        else
            break or **continue** ; ▷ depends on the variant

Three variants depending on how the three instructions are implemented.

Our three scheduler variants

Scheduler-V1: Follow the sequential order of $\mathbf{S_o}$,
otherwise wait for the next scheduling instant.
↪ See the paper.

Scheduler-V2: Ready List sorted w.r.t. bottom levels.
↪ See next slides.

Scheduler-V3: Adaptive aggregation of the sequential order of $\mathbf{S_o}$
and the sequential order based on the bottom levels.
↪ See the paper.

Scheduler-V2: Ready List sorted w.r.t. bottom levels

Variant specification

ready list $\mathcal{L}_{ready}$: sorted according to the *bottom levels* ($b\ell$)

memory check: on the current memory peak and on the peak of remaining sequential schedule (initialized with $\mathbf{S_o}$)

continue: if the current task cannot be scheduled, then try the next task in $\mathcal{L}_{ready}$

Scheduler-V2: Ready List sorted w.r.t. bottom levels

Variant specification

ready list $\mathcal{L}_{ready}$: sorted according to the *bottom levels* ($b\ell$)

memory check: on the current memory peak and on the peak of remaining sequential schedule (initialized with $\mathbf{S_o}$)

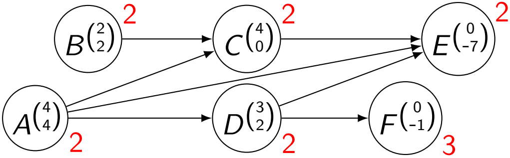
continue: if the current task cannot be scheduled, then try the next task in $\mathcal{L}_{ready}$

Bottom level [Hu'61]

Efficient node ordering based on the critical path: Backward computation

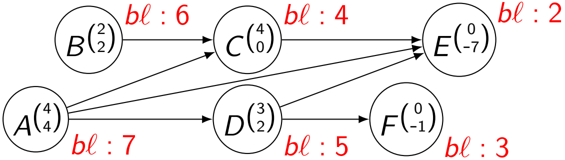
$$b\ell(X) = ET_X + \max_{Y \in \mathrm{Succ}(X)} \{b\ell(Y)\}$$

Scheduler-V2: Enforcing the memory constraint $\mathbf{\Pi} = 10$



Scheduler-V2: Enforcing the memory constraint $\mathbf{\Pi} = 10$

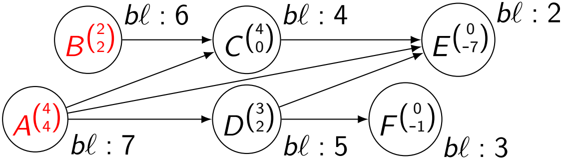
$\mathbf{S_o} = A; B; C; D; E; F$



Scheduler-V2: Enforcing the memory constraint $\mathbf{\Pi} = 10$

$\mathbf{S_o} = A; B; C; D; E; F$

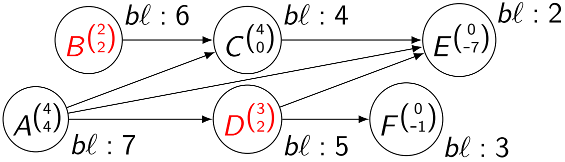
$\mathcal{L}_{ready} = [A; B]$

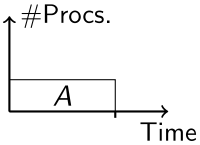


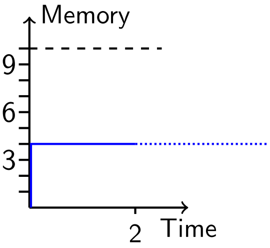
Scheduler-V2: Enforcing the memory constraint $\mathbf{\Pi} = 10$

$\mathbf{S_o} = A; B; C; D; E; F$

$\mathcal{L}_{ready} = [B; D]$



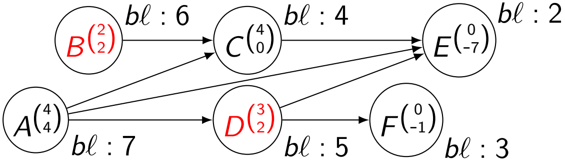


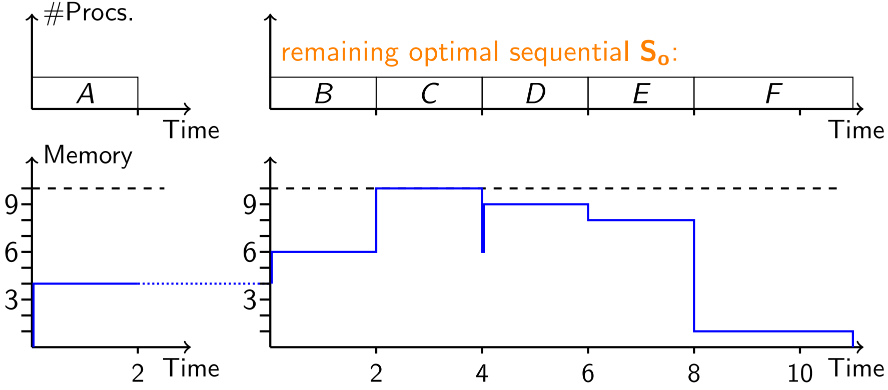


Scheduler-V2: Enforcing the memory constraint $\mathbf{\Pi} = 10$

$\mathbf{S_o} = A; B; C; D; E; F$

$\mathcal{L}_{ready} = [B; D]$

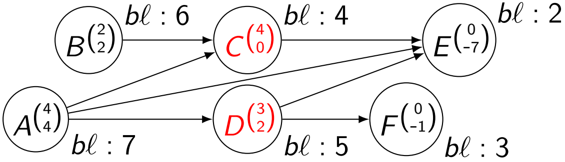


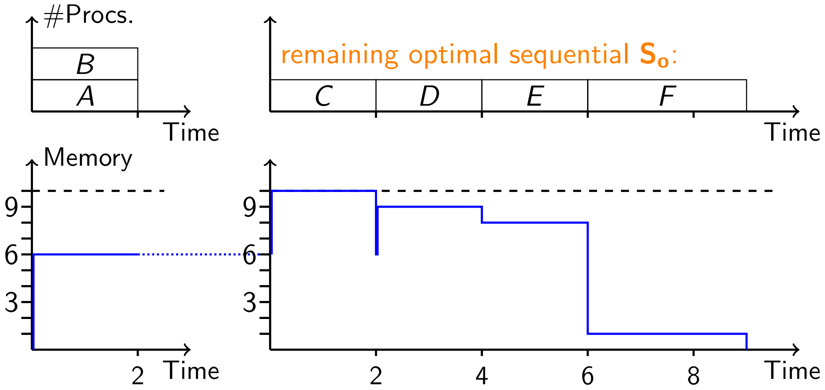


Scheduler-V2: Enforcing the memory constraint $\Pi = 10$

$\mathbf{S_o} = A; B; C; D; E; F$

$\mathcal{L}_{ready} = [D; C]$

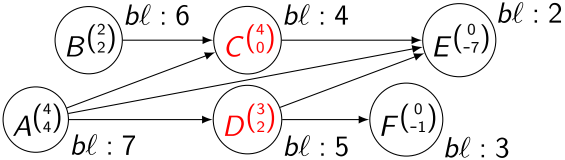


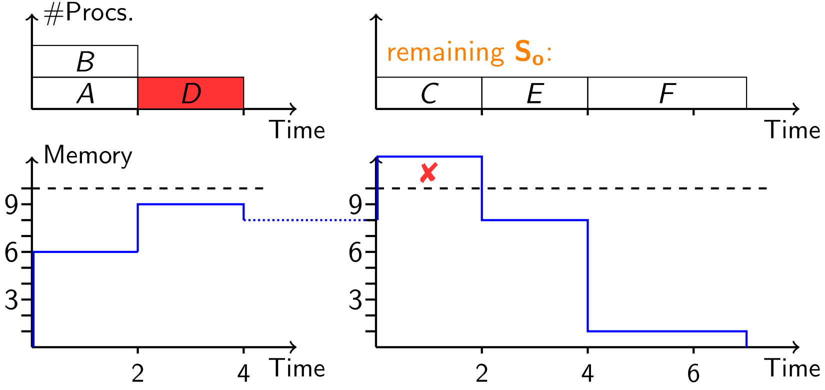


Scheduler-V2: Enforcing the memory constraint $\Pi = 10$

$\mathbf{S_o} = A; B; C; D; E; F$

$\mathcal{L}_{ready} = [D; C]$

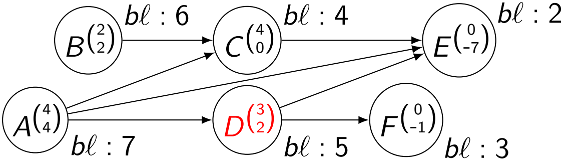


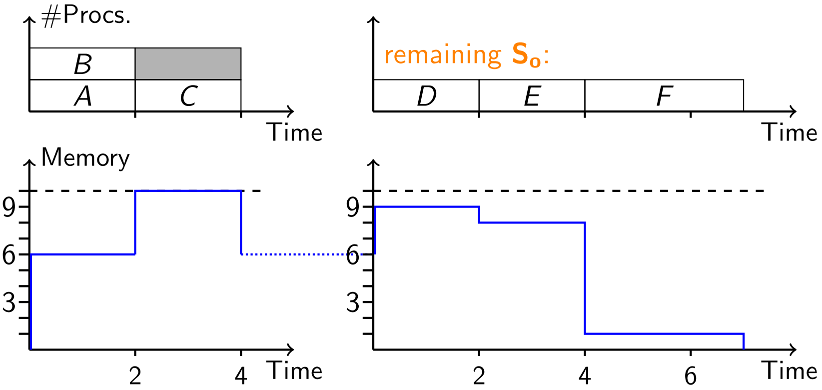


Scheduler-V2: Enforcing the memory constraint $\mathbf{\Pi} = 10$

$\mathbf{S_o} = A; B; C; D; E; F$

$\mathcal{L}_{ready} = [D]$

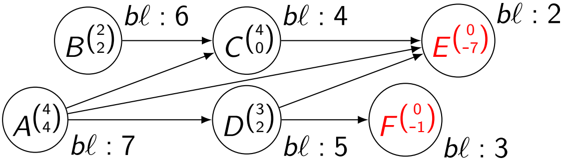


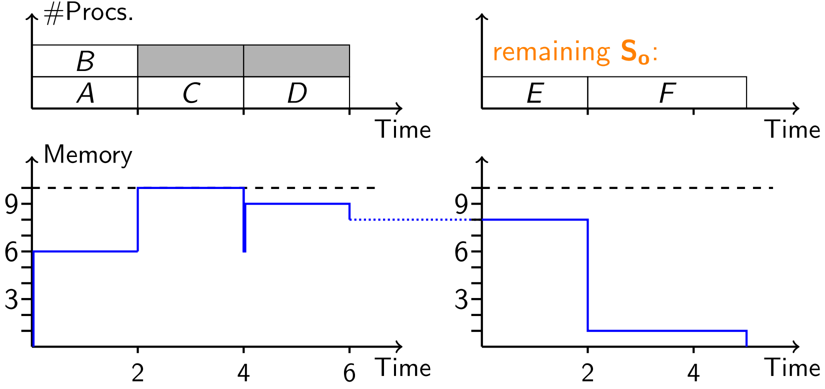


Scheduler-V2: Enforcing the memory constraint $\mathbf{\Pi} = 10$

$\mathbf{S_o} = A; B; C; D; E; F$

$\mathcal{L}_{ready} = [F; E]$

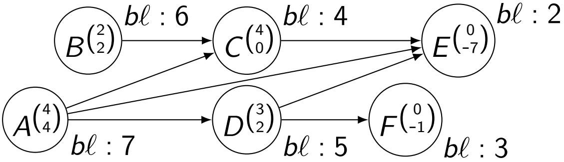


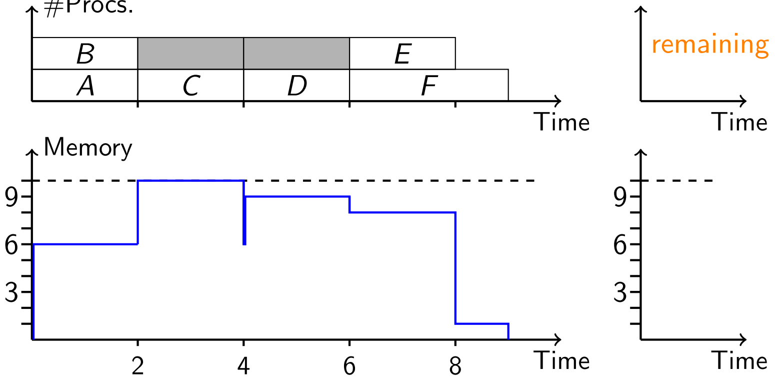


Scheduler-V2: Enforcing the memory constraint $\mathbf{\Pi} = 10$

$\mathbf{S_o} = A; B; C; D; E; F$

$\mathcal{L}_{ready} = \emptyset$





Benchmarks and state-of-the-art algorithms

Pegasus benchmark [Silva+14]

Random generator of task graphs mocking real scientific workflow applications.
$\hookrightarrow$ 120 task graphs of 50 and 100 tasks

QMF benchmark [MB01] in SDF [LM87], fully expanded

It is a signal processing application with parameterized size.
$\hookrightarrow$ task graphs up to $50,000$ tasks (no execution times provided)

Benchmarks and state-of-the-art algorithms

Pegasus benchmark [Silva+14]

Random generator of task graphs mocking real scientific workflow applications.
$\hookrightarrow$ 120 task graphs of 50 and 100 tasks

QMF benchmark [MB01] in SDF [LM87], fully expanded

It is a signal processing application with parameterized size.
$\hookrightarrow$ task graphs up to $50,000$ tasks (no execution times provided)

State-of-the-art: [MNSV'18]+[BMRT'20]

Add dummy edges to prevent graph cuts above the memory constraint $\mathbf{\Pi}$:

- MBL Heuristic based on Min Bottom Levels
- RO Heuristic based on a sub-optimal sequential schedule

Experimental results

- 1. Success rates
- 2. Speedups
- 3. Execution times

Download MASTAG at
`https://gitlab.inria.fr/spades-pub/mastag`

1. Success rate to meet the memory constraint $\mathbf{\Pi}$ (Pegasus, 4 and 8 processors)

min. peak: $\mathbf{\Pi}$ = minimal memory peak of $\mathbf{S_o}$ [FGH'24].

midway: $\mathbf{\Pi}$ = average between the minimal memory peak of $\mathbf{S_o}$ and the memory peak obtained by a Critical Path list scheduling.

objective	4 processors				8 processors			
	MBL	RO	RO+$\mathbf{S_o}$	[ours]	MBL	RO	RO+$\mathbf{S_o}$	[ours]
min. peak	0%	7%	49%	**100%**	6%	21%	66%	**100%**
midway	4%	33%	75%	**100%**	8%	44%	92%	**100%**

By construction, our schedulers never fail !
Thanks to the memory check based on the suffix of $\mathbf{S_o}$.

2. Speedups (Pegasus, 4 processors min. peak)

$\text{Speedup} = C_{\max}(\mathbf{S_o}) \;/\; C_{\max}(\text{Method})$

10^{th} sample	RO	RO+$\mathbf{S_o}$	[ours]-V1	[ours]-V2	[ours]-V3
Ligo_50	Fail	**3.09**	1.49	3.04	2.78
Ligo_100	Fail	Time Out	1.45	**3.57**	3.53
Montage_50	**3.57**	3.56	2.43	**3.57**	**3.57**
Montage_100	Fail	3.12	2.44	**3.13**	3.03
Genome_50	Fail	1.08	1.08	**1.26**	1.08
Genome_100	Fail	Time Out	1.08	1.26	**1.28**
Average 120 samples	N/A	N/A	1.64	**2.68**	2.62
(Average 8 procs.)	N/A	N/A	1.91	**3.80**	3.68

V2, V3 with mem. check on the suffix of $\mathbf{S_o}$ are very competitive !

Following strictly a precomputed seq. order (V1, RO) is not good !

3. Preprocessing and scheduling runtimes (QMF, 8 processors)

preprocessing (in sec.): time to add the dummy edges (RO) and/or to compute a sequential schedule (all)

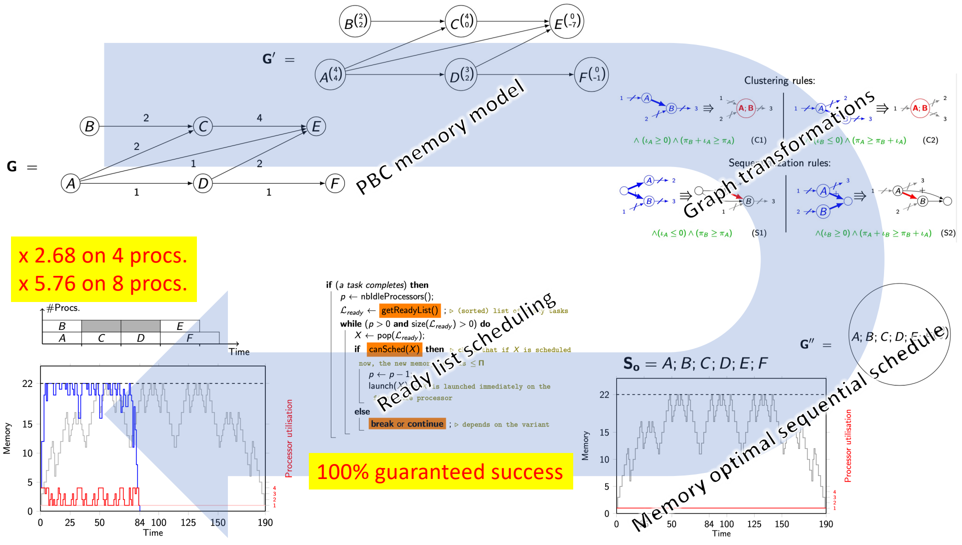
scheduling (in sec.): time to schedule (including the memory checks [ours])

sample	#tasks	RO		[ours]-V3	
		preproc.	sched.	preproc.	sched.
QMF_235_d2	190	358.29	0.01	0.04	0.01
QMF_235_d3	1,300	Time Out	N/A	0.76	0.19
QMF_235_d4	8,250	Out of Memory	N/A	25.17	6.51
QMF_235_d5	50,000	Out of Memory	N/A	828.27	396.28

Our scheduler V3 handles up to 50,000 tasks !

Wrap-up

Summary of the whole scheduling method



First contribution: Memory-optimal sequential schedule

- System model able to handle many memory models
- Four graph transformations
- Graph transformations based on simple topological and arithmetic conditions that can be checked locally
- In many cases, able to reduce the graph to a single node graph that contains one memory-optimal sequential schedule
- Optimized Branch-and-Bound algorithm to find a memory-optimal sequential schedule: It explores the next nodes having the lower peak and backtracks otherwise, taking into account the max
- Significant improvement over the state of the art

Second contribution: Dynamic memory-aware parallel list scheduling

- Ready list-scheduling that relies on any sequential schedule to meet the memory constraint $\mathbf{\Pi}$
- Guarantees that the memory constraint $\mathbf{\Pi}$ is always met !
- Very good speedups (on average **2.68** on 4 procs. and **5.76** on 8 procs.) and low execution times

Future work

- Implementation into a runtime scheduler (e.g. StarPU)
- More precise task memory usage: memory profile in function of time
- Study application to register minimization in compiling
- Study complementary techniques to limit memory usage e.g., offloading and rematerialization

Bibliographic references

[Hu'61] *Parallel sequencing and assembly line problems*, T.C. Hu (1961)

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

[LM'87] *Synchronous Data Flow*, E.A. Lee and D.G. Messerschmitt (1987)

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

[KLMU'18] *Scheduling series-parallel task graphs to minimize peak memory*, E. Kayaaslan et al. (2018)

[Silva+14] *Community resources for enabling research in distributed scientific workflows* R.F. da Silva et al. (2014)

[SBS'14] *Bounded memory scheduling of dynamic task graphs*, D. Sbîrlea et al. (2014)

[MNSV'18] *Parallel scheduling of dags under memory constraints*, L. Marchal et al. (2018)

[BMRT'20] *Revisiting dynamic DAG scheduling under memory constraints for shared-memory platforms*, G. Bathie et al. (2020)

[FGH'24] *Graph Transformations for Memory Peak Minimization by Scheduling*, P. Fradet et al. (2024)

Computing the memory peak of a sequential schedule (1)

Peak and impact of a sequence of 2 nodes

$$A^{\binom{\pi_a}{\iota_a}}; B^{\binom{\pi_b}{\iota_b}} = (A; B)^{\binom{max(\pi_a, \pi_b + \iota_a)}{\iota_a + \iota_b}} \tag{PI}$$

Property: Associativity

Operation (PI) is associative.

Corollary

The memory peak of a sequential schedule can be computed in linear time.

Computing the memory peak of a sequential schedule (2)

Peak and impact of a sequence of 2 tasks

$$A^{\binom{\pi_a}{\iota_a}}\ ;\ B^{\binom{\pi_b}{\iota_b}} = (A;B)^{\binom{max(\pi_a,\pi_b+\iota_a)}{\iota_a+\iota_b}} \qquad \text{(PI)}$$

$$\underbrace{A^{\binom{4}{4}}\ ;\ B^{\binom{2}{2}}}\ ;\ \underbrace{C^{\binom{4}{0}}\ ;\ D^{\binom{3}{2}}}\ ;\ \underbrace{E^{\binom{0}{-7}}\ ;\ F^{\binom{0}{-1}}}$$

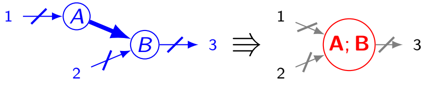
$$\underbrace{(A;B)^{\binom{6}{6}}\ ;\ (C;D)^{\binom{4}{2}}}\ ;\ (E;F)^{\binom{0}{-8}}$$

$$\underbrace{(A;B;C;D)^{\binom{10}{8}}\ ;\ (E;F)^{\binom{0}{-8}}}$$

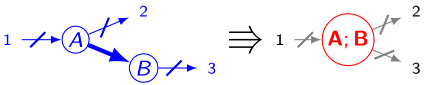
$$(A;B;C;D;E;F)^{\binom{10}{0}}$$

Peak-preserving compression: Four rules to reduce them all

Two clustering rules:

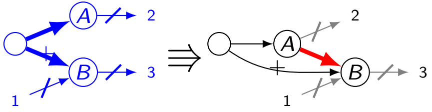


$\wedge\ (\iota_A \geq 0) \wedge (\pi_B + \iota_A \geq \pi_A)$ (C1)

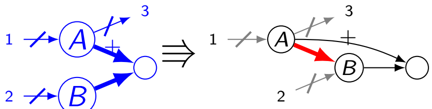


$\wedge\ (\iota_B \leq 0) \wedge (\pi_A \geq \pi_B + \iota_A)$ (C2)

Two sequentialization rules:



$\wedge (\iota_A \leq 0) \wedge (\pi_B \geq \pi_A)$ (S1)



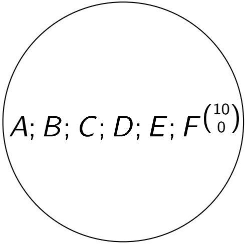
$\wedge (\iota_B \geq 0) \wedge (\pi_A + \iota_B \geq \pi_B + \iota_A)$ (S2)

Properties

These four rules reduce the number of sequential schedules of $\mathbf{G}'$.
Simple topological and arithmetic conditions that can be checked locally.

Fully compressed graph $\mathbf{G}''$

$\mathbf{G}'' =$



Scheduler-V1: Follow the sequential order, otherwise wait

Variant specification

ready list: sorted as in the optimal sequential schedule $\mathbf{S_o}$

memory check: on the current memory peak only

break: if the current task cannot be scheduled,
wait until the next scheduling instant

$$\mathsf{canSched}(X, \Pi) \stackrel{\text{def}}{=} \mathsf{transientMem}(t) + \pi_X \leq \mathbf{\Pi} \tag{1}$$

$$\mathsf{transientMem}(t) \stackrel{\text{def}}{=} \sum_{Y \in \text{completed}(t)} \iota_Y + \sum_{Z \in \text{running}(t)} \pi_Z \tag{2}$$

Scheduler-V3: Adaptive aggregation of two orders

Variant specification

ready list: sorted according to $b\ell$ and to $\mathbf{S_o}$

memory check: on the current memory peak and on the remaining sequential peak (initialized with $\mathbf{S_o}$)

continue: if the current task cannot be scheduled try the next ready task

Linear aggregation of two normalized orders w.r.t. $\mathbf{S_o}$ and $b\ell$:

$$Score(X) = r\, O_{\mathbf{S_o}}(X) + (1-r)\, O_{b\ell}(X) \quad (3)$$

The ratio r depends on the memory constraint $\mathbf{\Pi}$, on the schedule obtained with a Critical Path heuristics, and on $\mathbf{S_o}$:

$$r = \frac{\pi_{CP} - \mathbf{\Pi}}{\pi_{CP} - \pi_{\mathbf{S_o}}} \quad (4)$$

Scheduler-V3: normalization of objectives

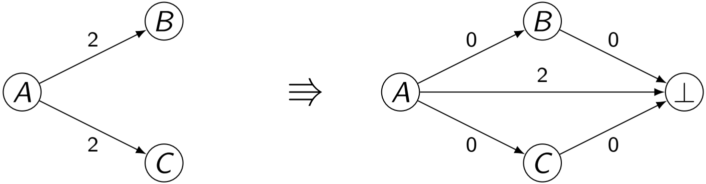
For bottom levels

$$O_{b\ell}(X) = \frac{b\ell(X)}{\max_{X \in \mathcal{L}_{ready}} b\ell(X)} \quad (5)$$

For sequential order

$$O_{\mathbf{S_o}}(X) = \frac{1}{i} \quad \text{where } i \text{ is } X\text{'s index in } S^r. \quad (6)$$

Shared output data transformation



Garbage collectors (⊥) should be executed ASAP.