Memory-aware Scheduling

Julien Herrmann

ENSEEIHT
3SN-B - Algorithmes HPC

December 7, 2023

Outline

1. Memory Usage and Performance on general task graphs

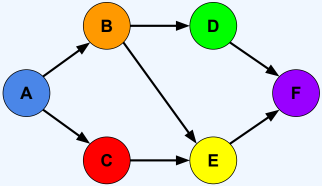
2. Tree-shaped task graphs
   - Sequential traversals
   - Parallel traversals

3. Backpropagation graphs
   - Single memory
   - Multiple memories
   - Parallel Processing

Memory Usage and Performance on general task graphs

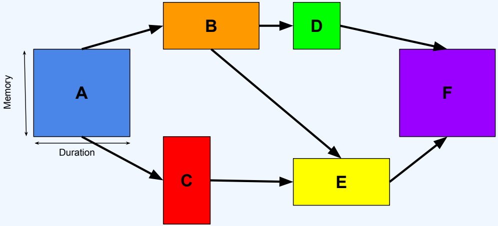
- Decompose an application (simulations, scientific computations,...) into tasks
- Data produced and used by tasks create dependencies
- Task graph : Directed Acyclic Graph (DAG)
  - nodes : computational tasks
  - edges : data dependencies between tasks
- Task mapping and scheduling done at runtime
- Numerous runtime projects:
  - StarPU (Inria Bordeaux) : dynamically schedules tasks on any computing ressource (CPU, GPU, *PU)
  - DAGUE, ParSEC (ICL Tennessee) : task graph expressed in symbolic form for linear algebra
  - StarSs (Barcelona), Xkaapi (Grenoble), and others...

- Consider a simple task graph



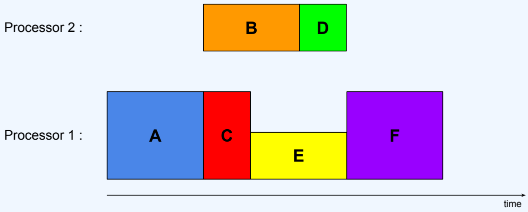
Task graph scheduling and memory

- Consider a simple task graph
- Tasks have duration and memory demands

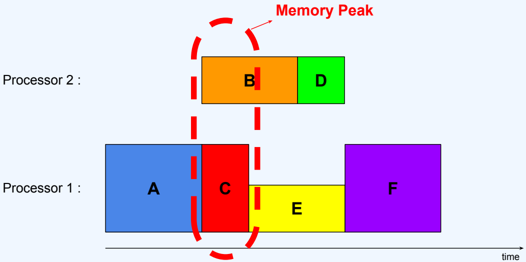


Task graph scheduling and memory

- Consider a simple task graph
- Tasks have duration and memory demands



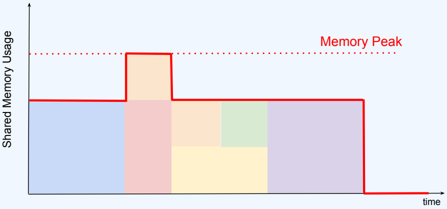
- Consider a simple task graph
- Tasks have duration and memory demands



- **Peak Memory :** maximum memory usage

Task graph scheduling and memory

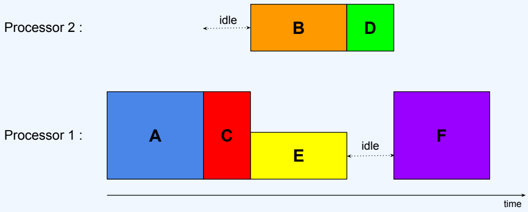
- Consider a simple task graph
- Tasks have duration and memory demands



- **Peak Memory :** maximum memory usage

Task graph scheduling and memory

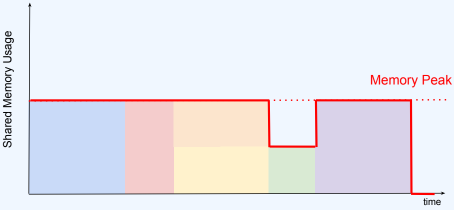
- Consider a simple task graph
- Tasks have duration and memory demands



- **Peak Memory :** maximum memory usage

Task graph scheduling and memory

- Consider a simple task graph
- Tasks have duration and memory demands



- Peak Memory : maximum memory usage
- Trade-off between Peak Memory and Performance

Research Problems

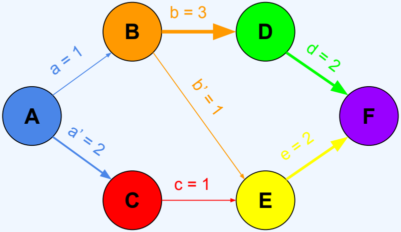
Several interesting questions:

- For sequential processing :
  - Minimum memory to process a graph
  - In case of memory shortage, minimum I/Os required
- In case of parallel processing :
  - Trade-off between memory and time
  - Makespan minimization under bounded memory

All of these problems are *NP-hard* on general graphs.
Sometimes restrict on simpler graphs:

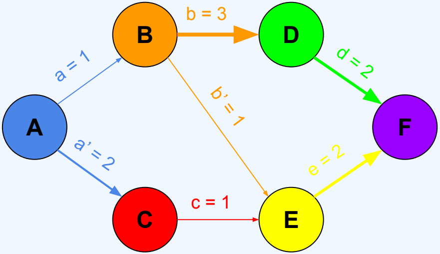
- Trees (single output, multiple inputs for each task)
  Arise in sparse linear algebra (sparse direct solvers), with large data to handle : memory is a problem
- Backpropagation graphs
  Arise in automatic differentiation, gradient descent, training of deep neural networks,...

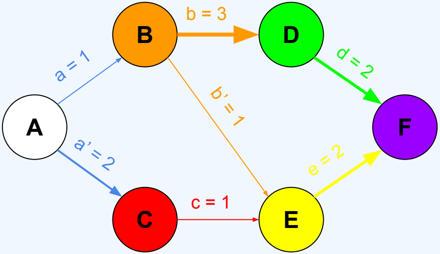
- Tasks have no execution data
- Output data have a memory size

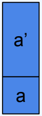


- Output data are kept in memory until every children is processed
- Even in the sequential case, scheduling influences the peak memory

Example



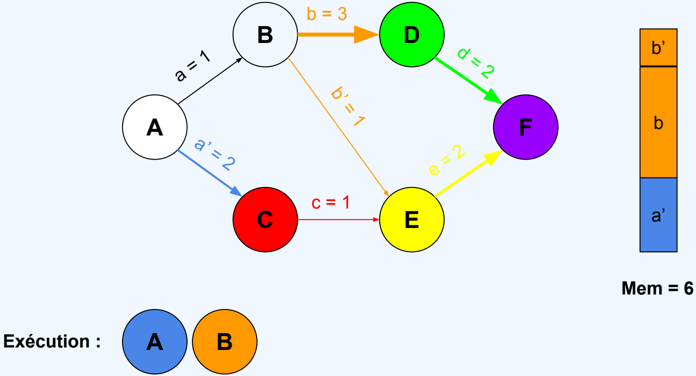




Mem = 3

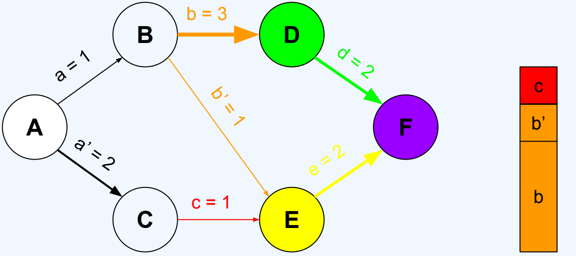
Exécution : **A**

Exécution : A C B E D F Mem Peak = 5



Exécution : A C B E D F Mem Peak = 5

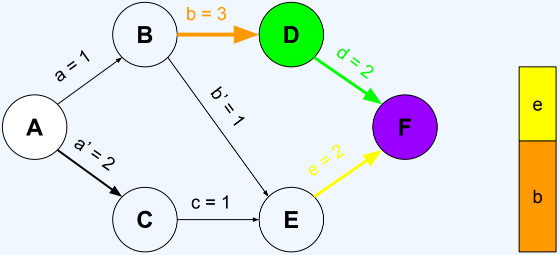
Example



Mem = 5

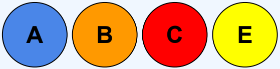
Exécution : **A** **B** **C**

Exécution : A C B E D F Mem Peak = 5

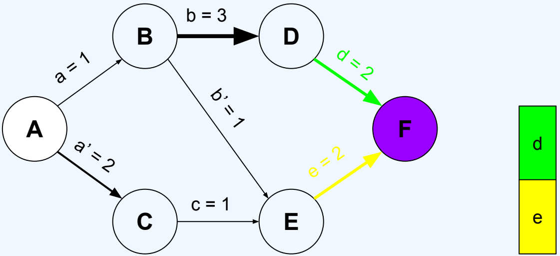


Mem = 5

Exécution :



Exécution : A C B E D F Mem Peak = 5

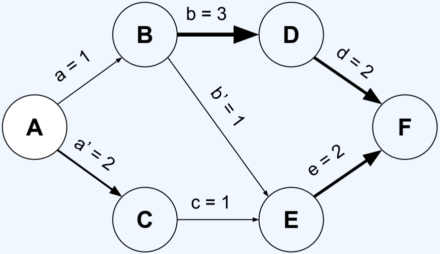


Mem = 4

Exécution : A B C E D

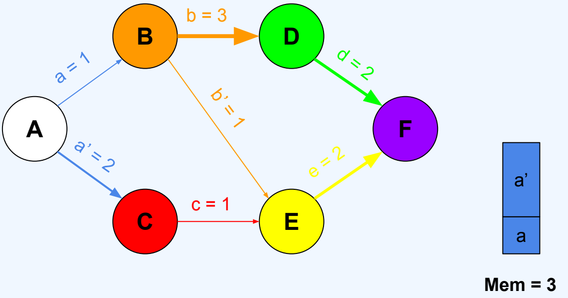
Exécution : A C B E D F Mem Peak = 5

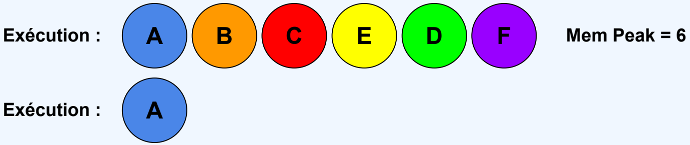
Example



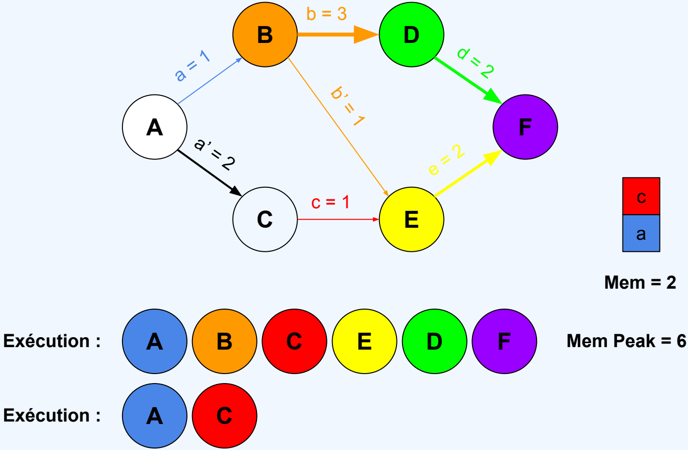
Exécution : A B C E D F **Mem Peak = 6**

Exécution : A C B E D F Mem Peak = 5

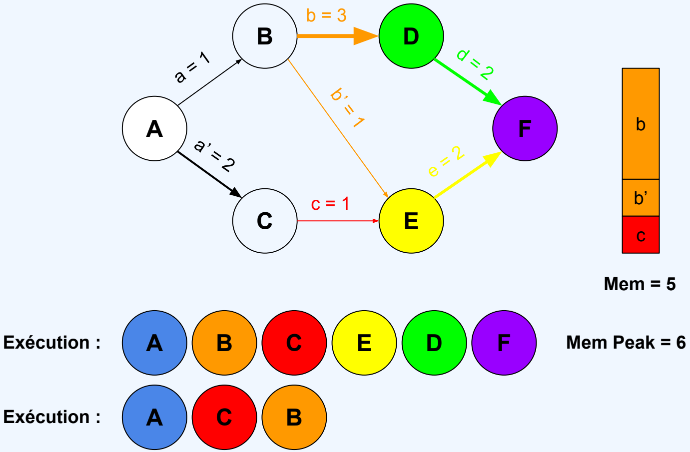

A
B
C
D
E
F
a = 1
a' = 2
b = 3
b' = 1
c = 1
d = 2
e = 2
a'
a
Mem = 3


Exécution :
A
B
C
E
D
F
Mem Peak = 6
Exécution :
A

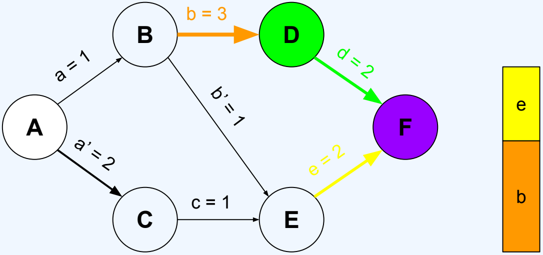
Example



B
b = 3
D
a = 1
d = 2
b' = 1
A
F
a' = 2
e = 2
c = 1
C
E
c
a
Mem = 2
Exécution :
A B C E D F
Mem Peak = 6
Exécution :
A C



B
b = 3
D
a = 1
d = 2
b' = 1
A
F
a' = 2
e = 2
C
c = 1
E
b
b'
c
Mem = 5
Exécution :
A
B
C
E
D
F
Mem Peak = 6
Exécution :
A
C
B

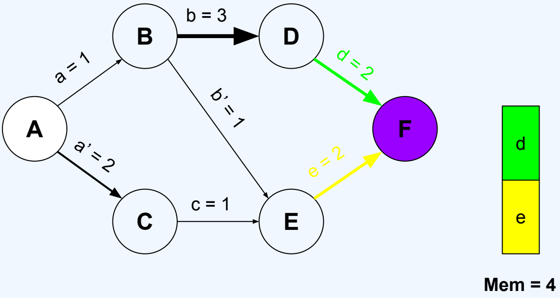


Mem = 5

Exécution : A B C E D F **Mem Peak = 6**

Exécution : A C B E

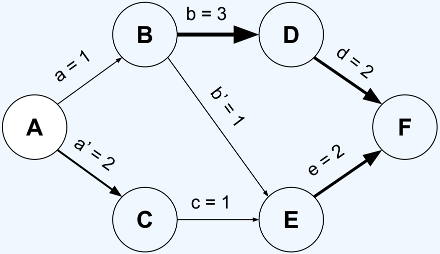
Example



Exécution : A B C E D F **Mem Peak = 6**

Exécution : A C B E D

EXAMPLE



Exécution : A B C E D F **Mem Peak = 6**

Exécution : A C B E D F **Mem Peak = 5**

Research Problems

Sequential processing of general DAGs:

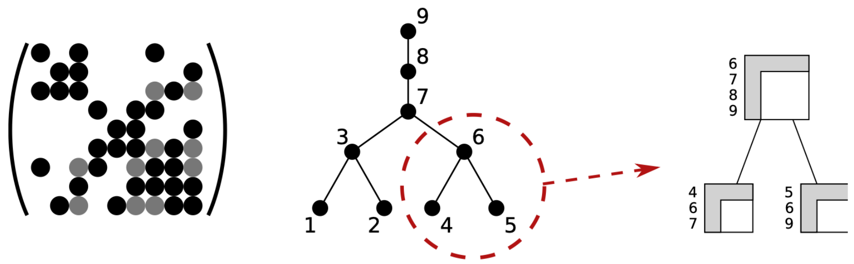
- Finding the topological order that minimize peak memory on general DAGs is **NP-complete**
- The problem is still **NP-complete** on **unitary** edges (pebble game)

The problem becomes polynomial when we restrict on some simpler graphs:

- Trees (single output, multiple inputs for each task)
  Arise in sparse linear algebra (sparse direct solvers), with large data to handle : memory is a problem
- Backpropagation graphs
  Arise in automatic differentiation, gradient descent, training of deep neural networks,...

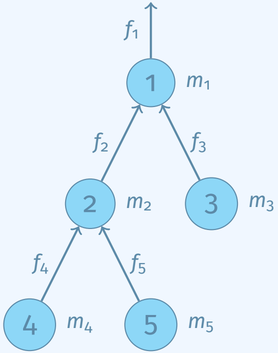
Tree-shaped task graphs

In the multifrontal method, when factorizing a sparse matrix, the order in which variables can be eliminated is expressed with a bottom-up elimination tree.



- Any **topological order** of the elimination tree leads to a correct factorization
- Parallelism : separate subtrees can be processed in parallel
- Output data have **large size** (increasing closer to the root)

Notations



- In-tree of n nodes
- Execution data of size m_i
- Output data of size f_i
- Leaf nodes have input data of null size

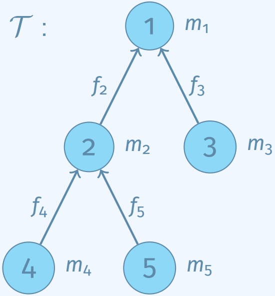
- Memory usage when executing node i:

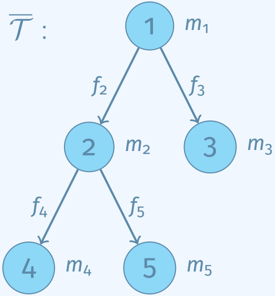
$$MemReq(i) = \left(\sum_{j \in Children(i)} f_j \right) + m_i + f_i$$

Theorem

Considering an in-tree $\mathcal{T}$ and a schedule σ_1 of $\mathcal{T}$, we can build a schedule σ_2 of the out-tree $\overline{\mathcal{T}}$ obtained by reversing all edges, with the same peak memory:

$$\sigma_2 = \mathit{reverse}(\sigma_1)$$



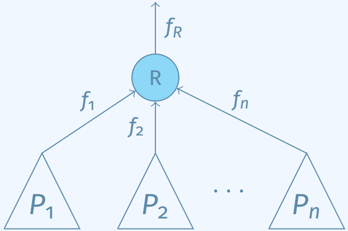


Post-order traversals for Tree

Post-order : entirely process one subtree after the other
(Deep First Search)

For each subtree $\mathcal{T}_i$:

- P_i : peak memory
- f_i : residual memory



For a given processing order of the subtrees $\mathcal{T}_1$, $\mathcal{T}_2$,... $\mathcal{T}_n$, the peak memory is:

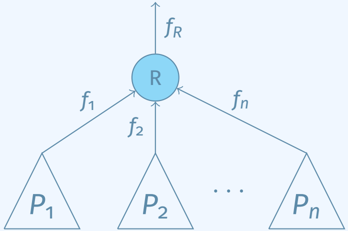
$$\max \left\{ P_1; \right.$$

Post-order traversals for Tree

Post-order : entirely process one subtree after the other
(Deep First Search)

For each subtree $\mathcal{T}_i$:

- P_i : peak memory
- f_i : residual memory



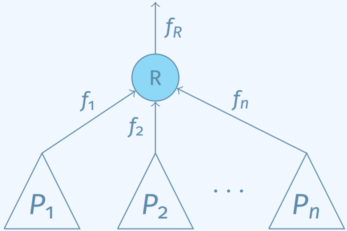
For a given processing order of the subtrees $\mathcal{T}_1$, $\mathcal{T}_2$,... $\mathcal{T}_n$, the peak memory is:

$$\max \left\{ P_1;\ f_1 + P_2; \right.$$

Post-order : entirely process one subtree after the other
(Deep First Search)

For each subtree $\mathcal{T}_i$:

- P_i : peak memory
- f_i : residual memory



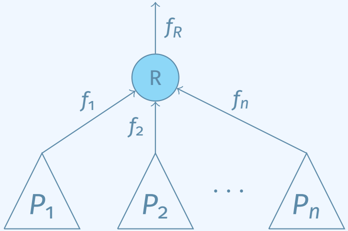
For a given processing order of the subtrees $\mathcal{T}_1$, $\mathcal{T}_2$,... $\mathcal{T}_n$, the peak memory is:

$$\max \left\{ P_1;\ f_1 + P_2;\ f_1 + f_2 + P_3; \ldots; \right.$$

Post-order : entirely process one subtree after the other (Deep First Search)

For each subtree $\mathcal{T}_i$:

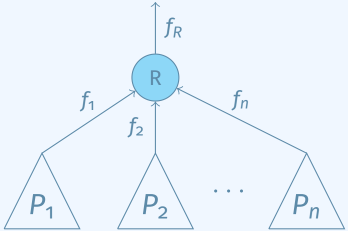
- P_i : peak memory
- f_i : residual memory



For a given processing order of the subtrees $\mathcal{T}_1$, $\mathcal{T}_2$,... $\mathcal{T}_n$, the peak memory is:

$$\max \left\{ P_1;\ f_1 + P_2;\ f_1 + f_2 + P_3; \dots; \sum_{i<n} f_i + P_n; \right.$$

Post-order : entirely process one subtree after the other
(Deep First Search)



For each subtree $\mathcal{T}_i$:

- P_i : peak memory
- f_i : residual memory

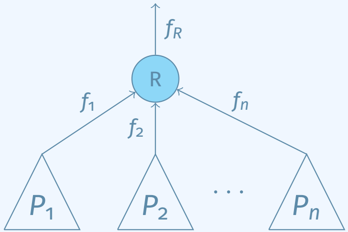
For a given processing order of the subtrees $\mathcal{T}_1$, $\mathcal{T}_2$,... $\mathcal{T}_n$, the peak memory is:

$$\max \left\{ P_1;\ f_1 + P_2;\ f_1 + f_2 + P_3;\ \ldots;\ \sum_{i<n} f_i + P_n;\ \sum_{i=1}^{n} f_i + m_R + f_R \right\}$$

Post-order : entirely process one subtree after the other (Deep First Search)

For each subtree $\mathcal{T}_i$:

- P_i : peak memory
- f_i : residual memory



For a given processing order of the subtrees $\mathcal{T}_1$, $\mathcal{T}_2$,... $\mathcal{T}_n$, the peak memory is:

$$\max\left\{\max_{j=1}^{n}\left(P_j + \sum_{i=1}^{j-1} f_i\right) ; \sum_{i=1}^{n} f_i + m_R + f_R\right\}$$

Theorem (Liu, Best Post-order Traversal)

The best post-order traversal is obtain by processing subtrees in non-increasing order of $P_i - f_i$.

Proof by contradiction.

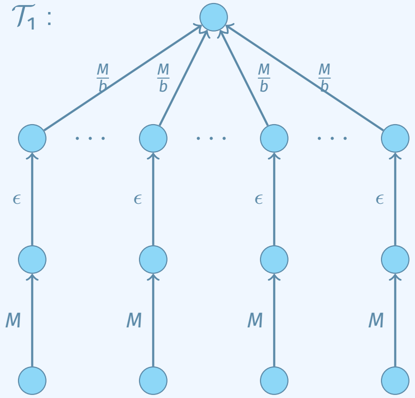
- Consider an optimal traversal which does not respect the order, that is to say:
  - subtree $\mathcal{T}_j$ is processed right before subtree $\mathcal{T}_k$
  - and $P_k - f_k \geq P_j - f_j$
- Transform the schedule step by step without increasing the peak memory

$\square$

Post-order is not optimal

Theorem (Post-order Traversals are arbitrary bad in the general case)

There is no constant K such that the best post-order traversal is a K-approximation in the general case.

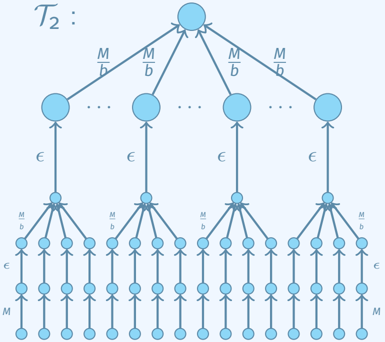


- Minimum post-order peak memory: $BPO_1 = M + \epsilon + (b-1)\frac{M}{b}$
- Minimum traversal peak memory: $BT_1 = M + \epsilon + (b-1)\epsilon$

Post-order is not optimal

Theorem (Post-order Traversals are arbitrary bad in the general case)

There is no constant K such that the best post-order traversal is a K-approximation in the general case.

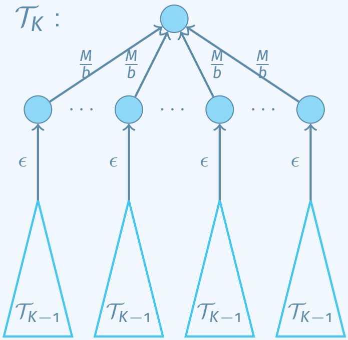


- Minimum post-order peak memory: $BPO_2 = M + \epsilon + 2(b-1)\frac{M}{b}$
- Minimum traversal peak memory: $BT_2 = M + \epsilon + 2(b-1)\epsilon$

Post-order is not optimal

Theorem (Post-order Traversals are arbitrary bad in the general case)

There is no constant K such that the best post-order traversal is a K-approximation in the general case.



- Minimum post-order peak memory: $BPO_K = M + \epsilon + K(b-1)\frac{M}{b}$
- Minimum traversal peak memory: $BT_K = M + \epsilon + K(b-1)\epsilon$
- Thus:

$$\frac{BPO_K}{BT_K} > K$$

Post-order is not optimal

Theorem (Post-order Traversals are arbitrary bad in the general case)

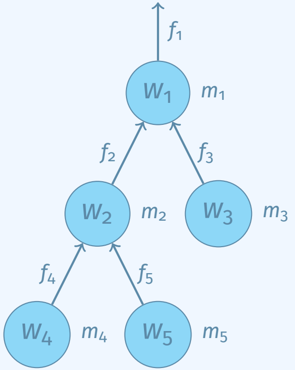
There is no constant K such that the best post-order traversal is a K-approximation in the general case.

The best post-order is not optimal in the general case but efficient in practice:

	actual assembly trees	random trees
Non-optimal traversals	4.2%	61%
Maximum increase compared to optimal	18%	22%
Average increase compared to optimal	1%	12%

Model for parallel tree processing

- P uniform processors
- Shared memory of size M
- Task i has execution times w_i
- Simultaneous processing of nodes induces larger memory
- Trade-off time vs. memory



When processing a tree with multiple processors, there are multiple sources of parallelism:

- Node parallelism : a task node can be processed using multiple processors.
$\Rightarrow$ It does not increase the memory usage but induces a lot of communications between processors.

- Tree parallelism : independent tasks can be processed at the same time by different processors
$\Rightarrow$ It increases the memory usage since their data coexists at the same time in the shared memory.

For tree-shaped task graph:

- Makespan minimization is NP-complete for general trees
- Makespan minimization is polynomial for unit-weigth task trees
- Memory minimization is polynomial for $w_i = 1$, $m_i = 0$, and $f_i = 1$ (pebble game model)

Theorem

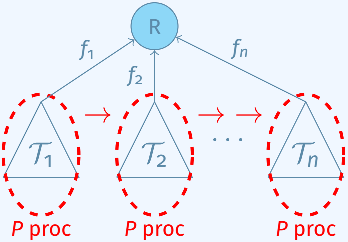
Deciding whether a tree can be scheduled with a bound M on memory and a bound C on makespan is NP-complete

Theorem

There is no algorithm that is both an α-approximation for makespan minimization and a β-approximation for peak memory minimization when scheduling tree-shaped task graphs

All-to-all mapping

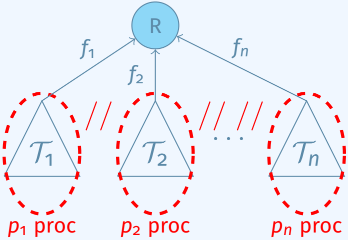
All-to-all mapping: post-order traversal of the tree, where all the processors work at every node (maximum node parallelism on every node)



- $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_n$ are processed in the best post-order using a all-to-all mapping scheduling
- Every processor executes root R in parallel

- Optimal memory scalability : same peak memory as the sequential execution
- No tree parallelism
- A lot of communications between processors

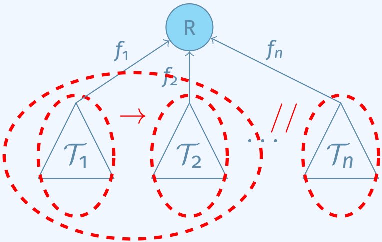
Proportional mapping: every subtrees are processed in parallel by a subset of processors proportional to their work load



- Subtree $\mathcal{T}_i$ is executed by p_i processors where: $p_i = P * \frac{W_{\mathcal{T}_i}}{W_{\mathcal{T}}}$ and $W_{\mathcal{T}} = \sum_{node \in \mathcal{T}} w_{node}$

- Good work-load balance to exploit tree parallelism
- Memory scalability can be arbitrary bad compared to the sequential execution: $P * M_{max}(P) >> M_{seq}$

Memory-aware mapping: (Agullo et al. [3]): aims at enforcing a given memory bound M_B on the peak memory



- Try to apply proportional mapping
- Check whether enough memory for each tree. If not, serialize them and update M_B:

- Ensures the given memory constraint and provides reliable estimates
- Tends to assign many processors on nodes at the top of the tree ⇒ performance issues on parallel nodes.

For parallel traversals of tree-shaped tasks graphs:

- Optimizing both memory and makespan is NP-complete
- Optimizing makespan under memory constraint is NP-complete
- No scheduling algorithm can be a constant factor approximation on both memory and time

Use of heuristics:

- All-to-all mapping : full node parallelism, no tree parallelism
- Proportional mapping : tree parallelism needing more memory
- Other memory-aware mappings

Backpropagation graphs

Automatic Differentiation

Ice-sheet model:

Model Algorithm (single timestep)

1. Evaluate driving stress $\tau_d = \rho g h \nabla s$
2. Solve for velocities

   DO i = 1, max_iter

   i. Evaluate nonlinear viscosity ν_i from iterate $\boldsymbol{u}_i$

   ii. Construct stress matrix $A\{\nu_i\}$

   iii. Solve linear system $A\, \boldsymbol{u}_{i+1} = \tau_d$

   iv. (Exit if converged)

   ENDDO
3. Evolve thickness (continuity eqn)

Automatic differentiation (AD) tools generate code for adjoint of operations

Simpler Version:

proc Model Algorithm(x_0)

begin

  Do stuff;

  for $i = 0$ *to* n **do**

    $\mathbf{x_{i+1}} = \mathbf{f_i}(\mathbf{x_i})$;

    Do stuff;

  end

  $\backslash * F(u_0) = f_n \circ f_{n-1} \circ \ldots \circ f_0(u_0) * \backslash$

  Compute $\nabla \mathbf{F}(\mathbf{x_0}).\mathbf{y}$;

end

A quick reminder about the gradient:

$$
\begin{aligned}
F(u_0) &= f_n \circ f_{n-1} \circ \ldots \circ f_1 \circ f_0(u_0) \\
\nabla F(u_0)\mathbf{y} &= \mathrm{J}f_0(u_0)^T \cdot \nabla(f_n \circ f_1)(u_1) \cdot \mathbf{y} \\
&= \mathrm{J}f_0(u_0)^T \cdot \mathrm{J}f_1(u_1)^T \cdot \ldots \cdot \mathrm{J}f_{n-1}(u_{n-1})^T \cdot \mathrm{J}f_n(u_n)^T \cdot \mathbf{y}
\end{aligned}
$$

$$
\begin{aligned}
\mathrm{J}f^T &= \text{Transpose Jacobian matrix of } f; \\
u_{i+1} &= f_i(u_i) = f_i\left(f_{i-1} \circ \ldots \circ f_0(u_0)\right).
\end{aligned}
$$

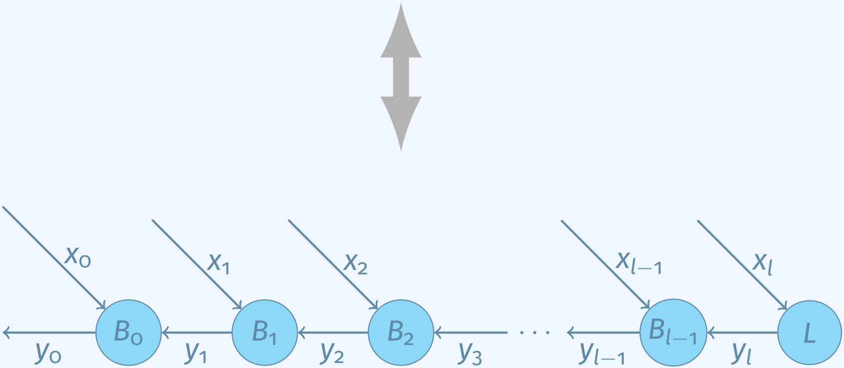
Task graph description

$$\mathbf{F_i(x_i) = x_{i+1}} \qquad \mathbf{i < l} \quad \textbf{(Forward Phase)}$$

$$B_i(x_i, y_{i+1}) = y_i \qquad i \leq l \quad \text{(Backward Phase)}$$

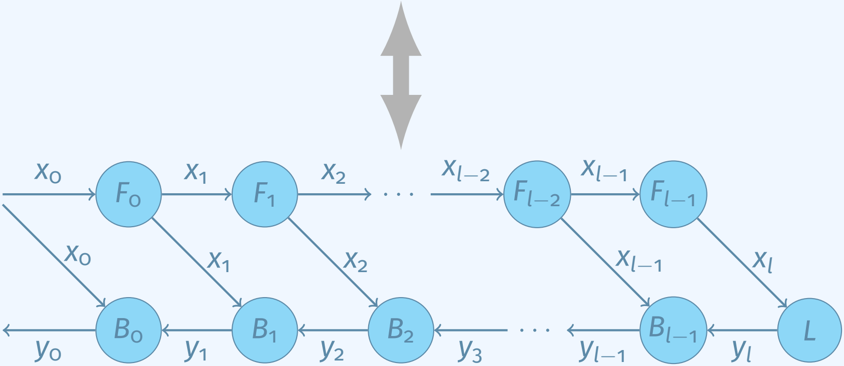
$$\xrightarrow{x_0} F_0 \xrightarrow{x_1} F_1 \xrightarrow{x_2} \cdots \xrightarrow{x_{l-2}} F_{l-2} \xrightarrow{x_{l-1}} F_{l-1}$$

$$F_i(x_i) = x_{i+1} \qquad i < l \quad \text{(Forward Phase)}$$
$$\mathbf{B_i(x_i, y_{i+1}) = y_i} \qquad \mathbf{i \leq l} \quad \textbf{(Backward Phase)}$$



Task graph description

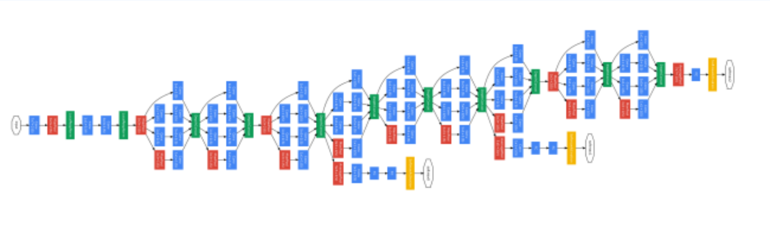
$$F_i(x_i) = x_{i+1} \qquad i < l \quad \text{(Forward Phase)}$$
$$B_i(x_i, y_{i+1}) = y_i \qquad i \leq l \quad \text{(Backward Phase)}$$



Relation to deep learning

When training a neural network:

- Forward phase: computes predicted output for each layers with respect to model weights
- Loss computation: difference between predicted output and expected output
- Backward phase: updates the model weights to minimize loss function

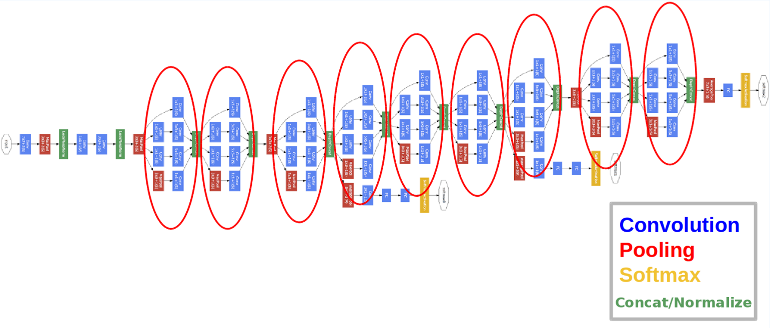


GoogleNet graph

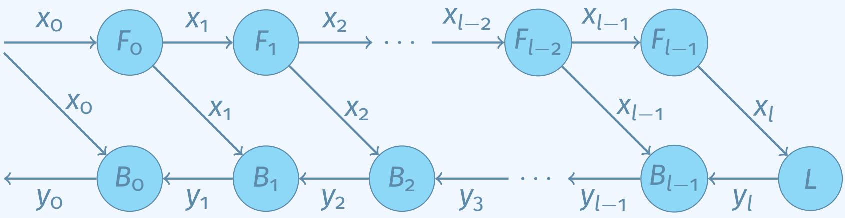
Relation to deep learning

When training a neural network:

- **Forward phase:** computes predicted output for each layers with respect to model weights
- **Loss computation:** difference between predicted output and expected output
- **Backward phase:** updates the model weights to minimize loss function

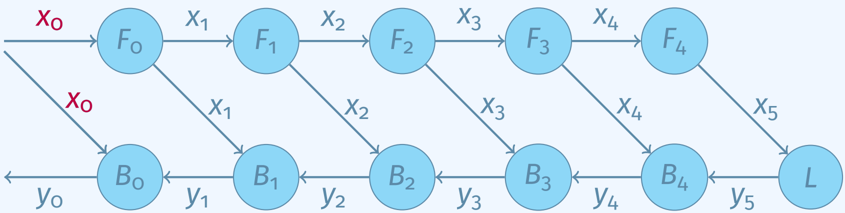


Backpropagation Graphs



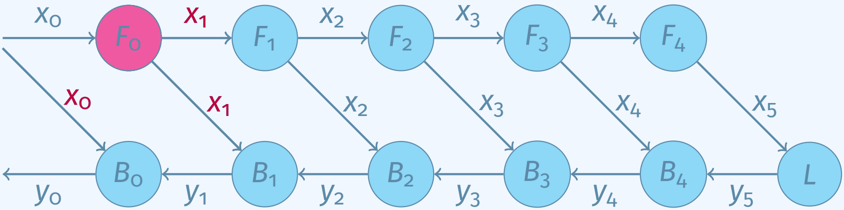
- No graph parallelism (linear structure)
- Intermediate data (x_i and y_i) have large sizes
- Intermediate data can be useful much later in execution
- Intermediate data can not all fit in memory at the same time
- Initial state : x_0 is stored in memory
- Objective : compute y_0

Model of execution : Store all



Memory :	x_0					

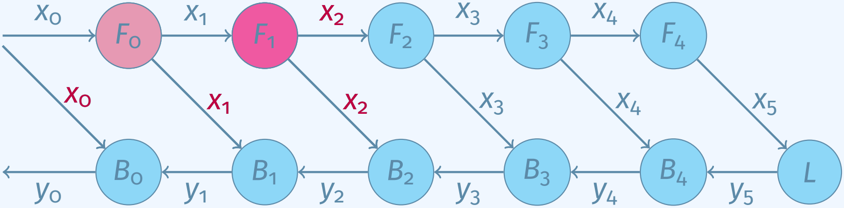
Model of execution : Store all



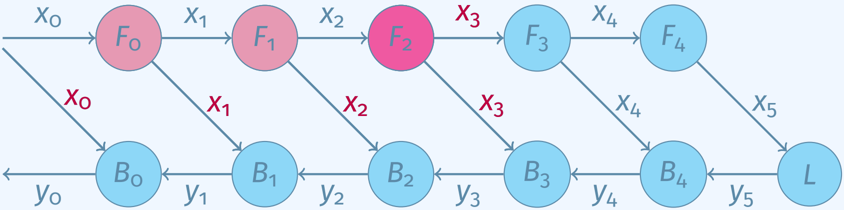
Memory :

x_0	x_1				

Model of execution : Store all



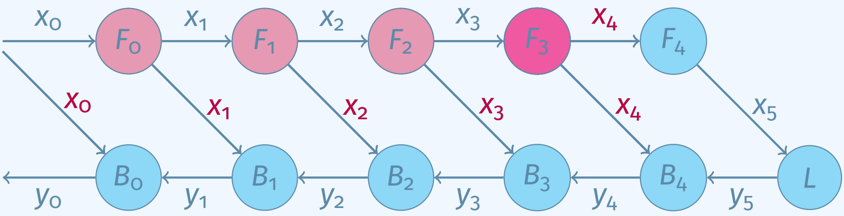
Memory :	x_0	x_1	x_2			



Memory :

x_0	x_1	x_2	x_3		

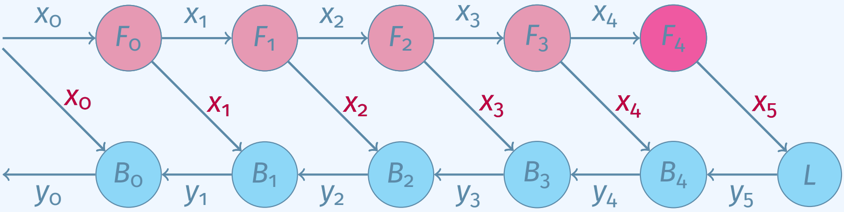
Model of execution : Store all



Memory :

x_0	x_1	x_2	x_3	x_4	

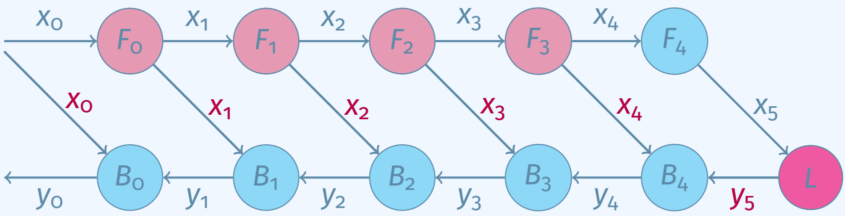
Model of execution : Store all



Memory :

x_0	x_1	x_2	x_3	x_4	x_5

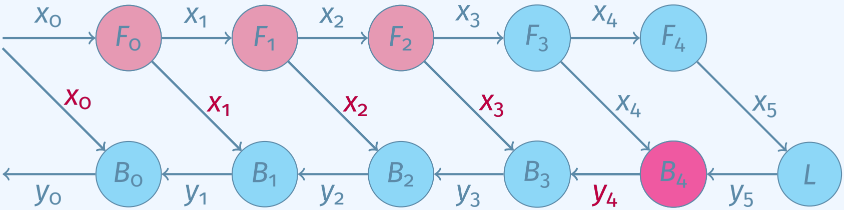
Model of execution : Store all



Memory :

x_0	x_1	x_2	x_3	x_4	y_5

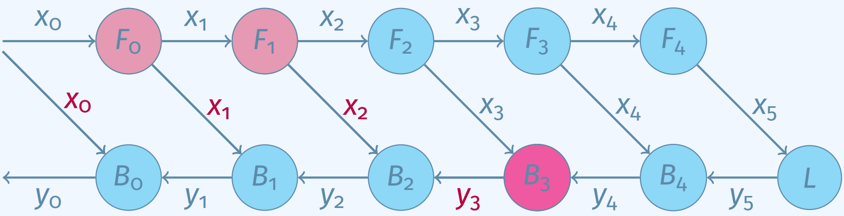
Model of execution : Store all



Memory :

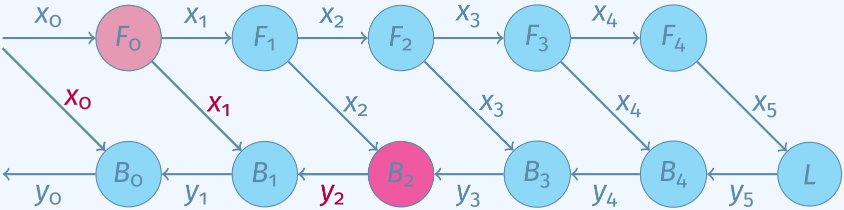
x_0	x_1	x_2	x_3	y_4	

Model of execution : Store all



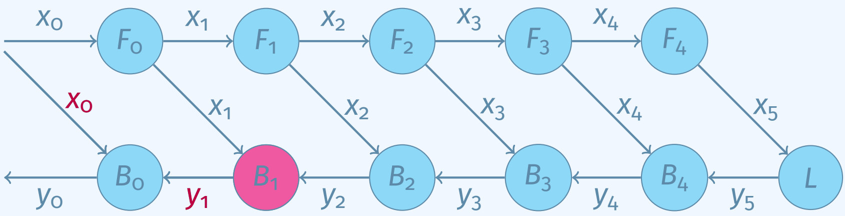
Memory :						
	x_0	x_1	x_2	y_3		

Model of execution : Store all



Memory :	x_0	x_1	y_2			

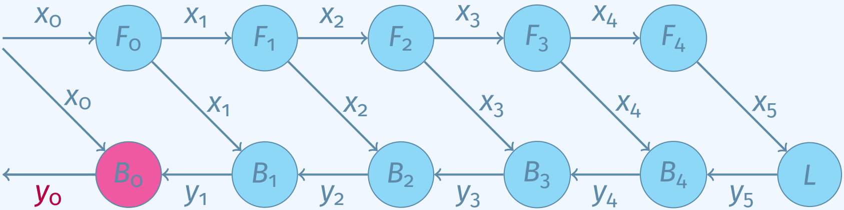
Model of execution : Store all



Memory :

x_0	y_1				

Model of execution : Store all



Memory :

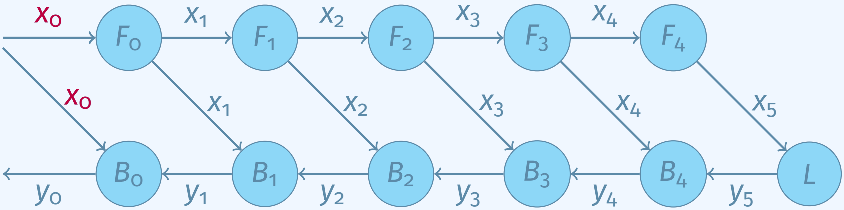
y_0					

For $l = 5$ forward steps

Strategy Store all (memory expensive):

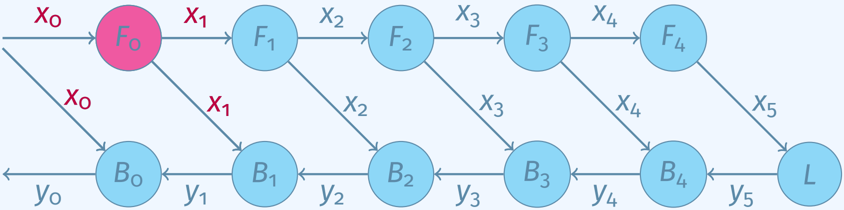
- Peak Memory : 6
- Recomputation : 0

Model of execution : Recompute all



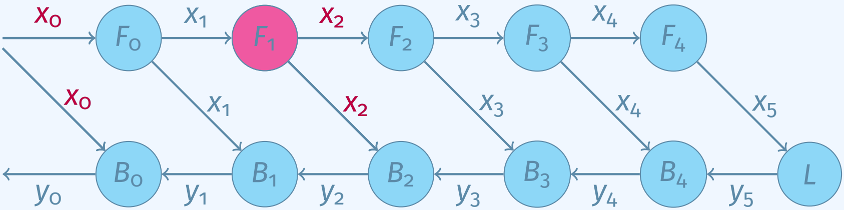
Memory :

x_0		

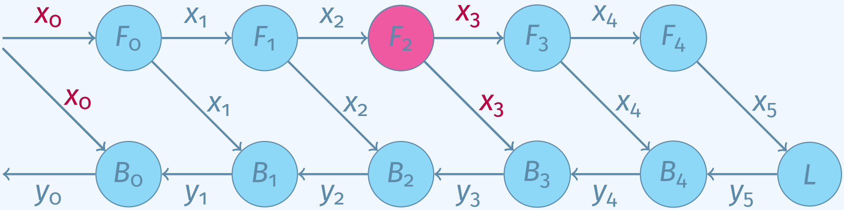

x_0
F_0
x_1
F_1
x_2
F_2
x_3
F_3
x_4
F_4
x_0
x_1
x_2
x_3
x_4
x_5
B_0
B_1
B_2
B_3
B_4
L
y_0
y_1
y_2
y_3
y_4
y_5

Memory : | x_0 | x_1 | |

Model of execution : Recompute all



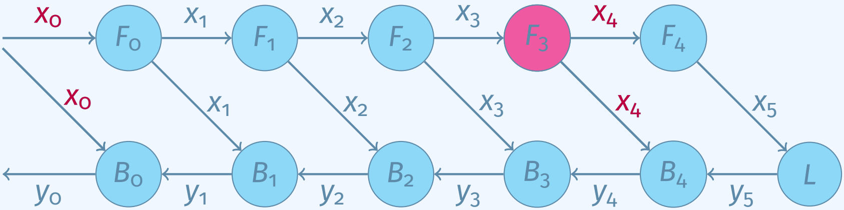
Memory :			
	x_0	x_2	


x_0
F_0
x_1
F_1
x_2
F_2
x_3
F_3
x_4
F_4
x_0
x_1
x_2
x_3
x_4
x_5
B_0
B_1
B_2
B_3
B_4
L
y_0
y_1
y_2
y_3
y_4
y_5

Memory :

x_0	x_3	

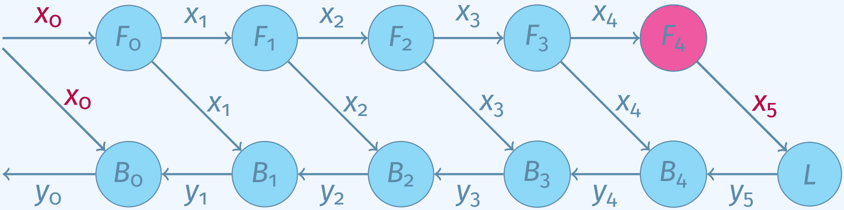
Model of execution : Recompute all



Memory :

x_0	x_4	

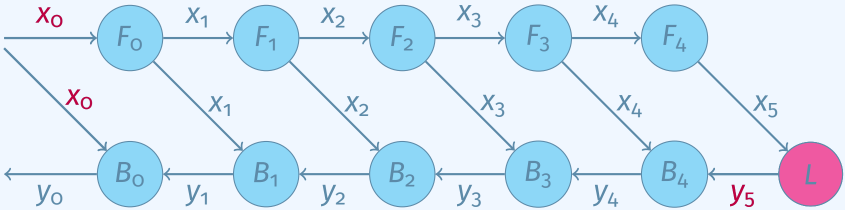
Model of execution : Recompute all



Memory :

x_0	x_5	

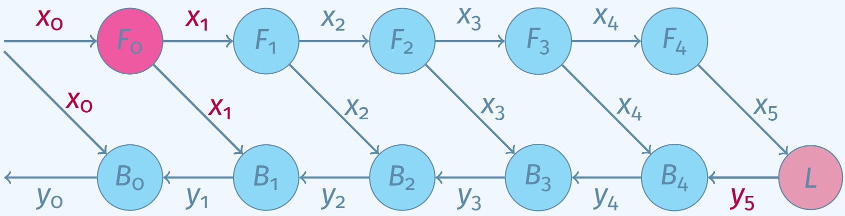
Model of execution : Recompute all



Memory :

x_0		y_5

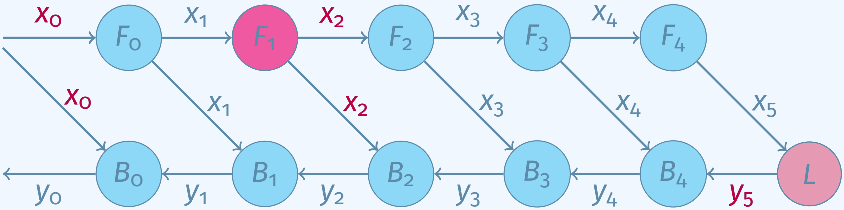
Model of execution : Recompute all



Memory :

x_0	x_1	y_5

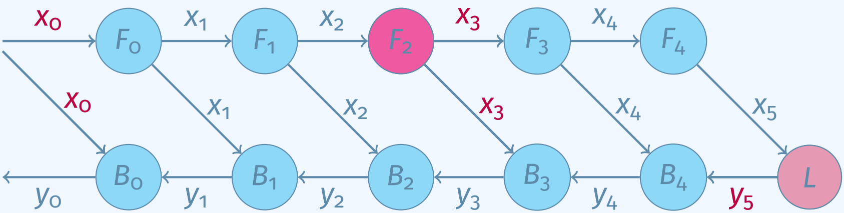
Model of execution : Recompute all



Memory :

x_0	x_2	y_5

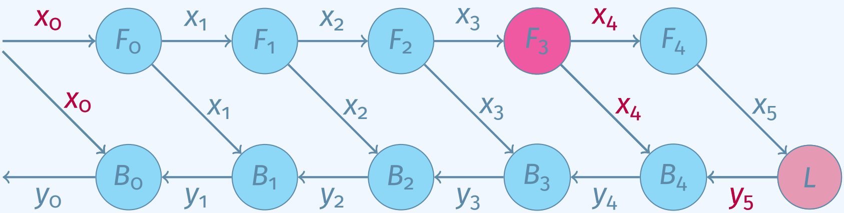
Model of execution : Recompute all



Memory :

x_0	x_3	y_5

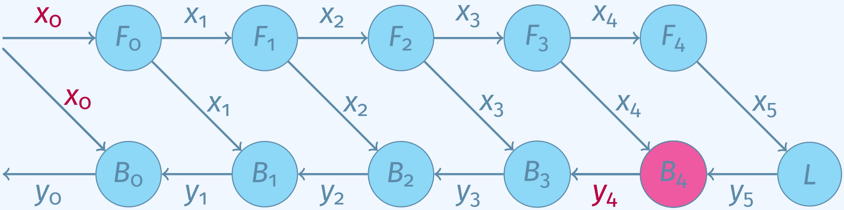
Model of execution : Recompute all



Memory :

x_0	x_4	y_5

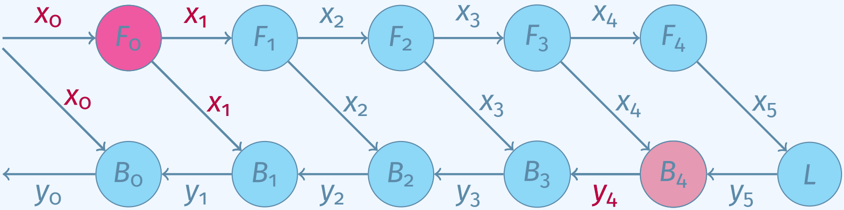
Model of execution : Recompute all



Memory :

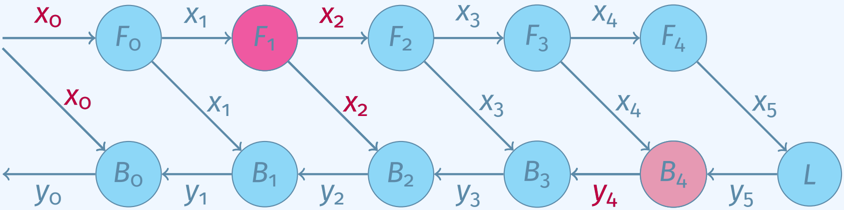
x_0		y_4

Model of execution : Recompute all



Memory : | x_0 | x_1 | y_4 |
|---|---|---|

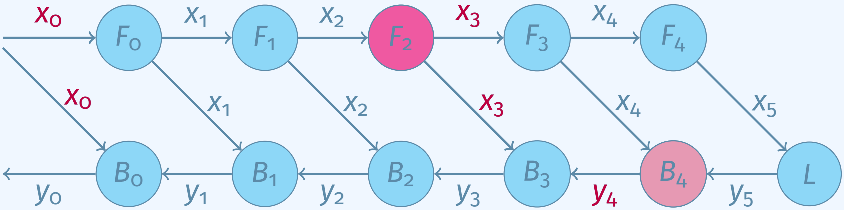
Model of execution : Recompute all



Memory :

x_0	x_2	y_4

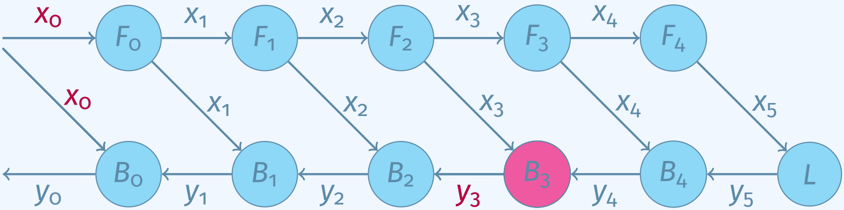
Model of execution : Recompute all



Memory :

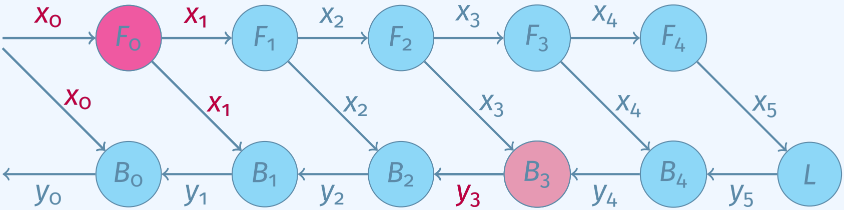
x_0	x_3	y_4

Model of execution : Recompute all



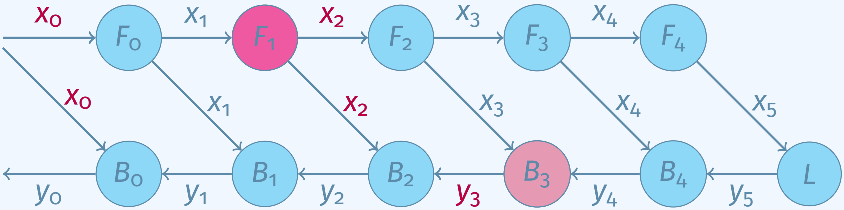
Memory :

x_0		y_3



Memory :

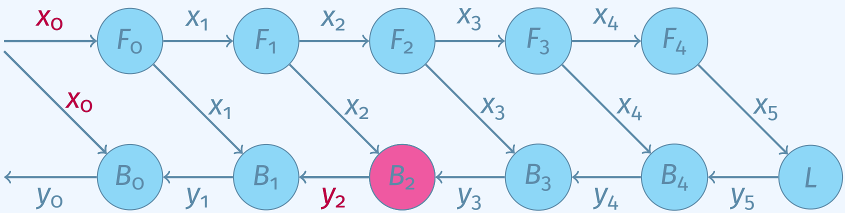
x_0	x_1	y_3



Memory :

x_0	x_2	y_3

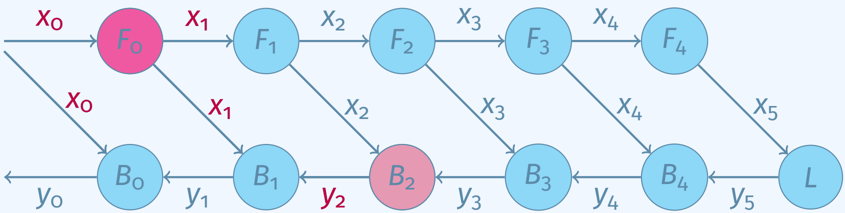
Model of execution : Recompute all



Memory :

x_0		y_2

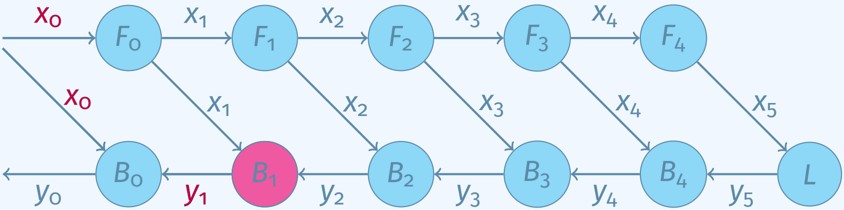
Model of execution : Recompute all



Memory :

x_0	x_1	y_2

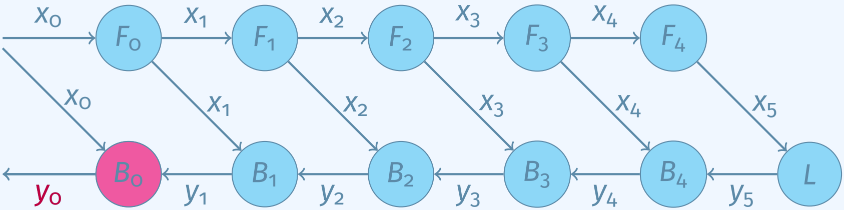
Model of execution : Recompute all



Memory :

x_0		y_1

Model of execution : Recompute all



Memory :

y_0		

For $l = 5$ forward steps

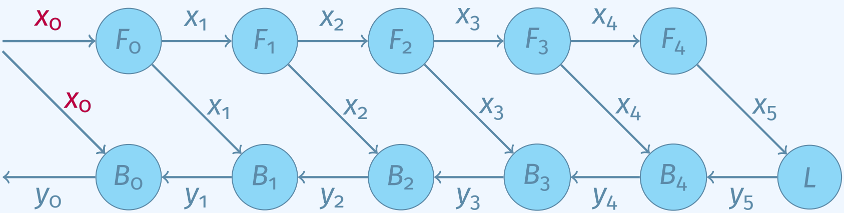
Strategy Store all (memory expensive):

- Peak Memory : 6
- Recomputation : 0

Strategy Recompute all (compute expensive):

- Peak Memory : 3
- Recomputation : 10

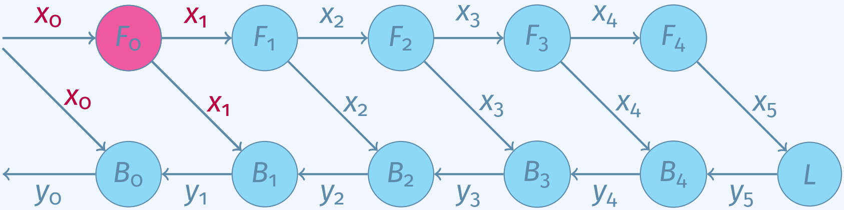
Execution : Store some / Recompute some



Memory :

x_0			

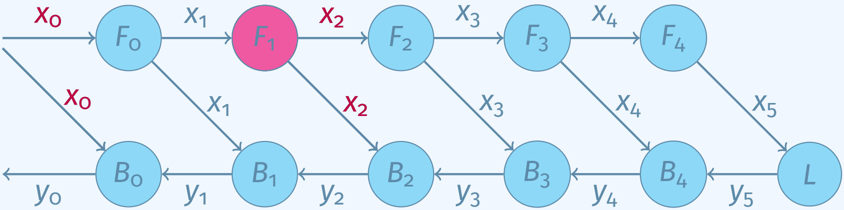
Execution : Store some / Recompute some



Memory :

x_0	x_1		

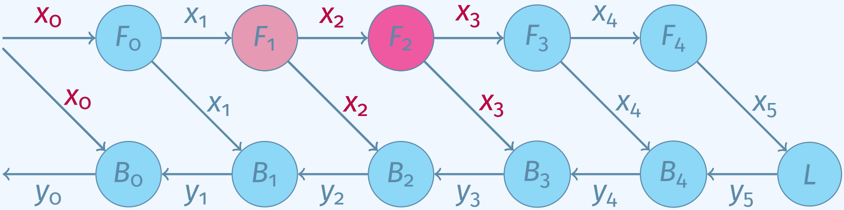
Execution : Store some / Recompute some



Memory :

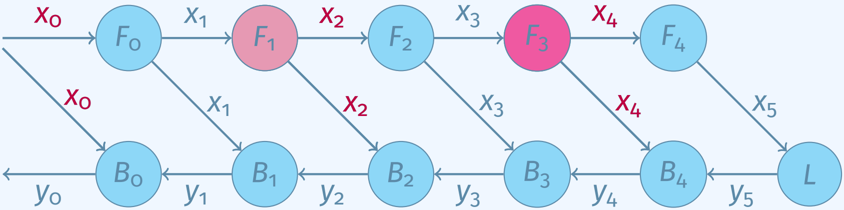
x_0	x_2		

Execution : Store some / Recompute some



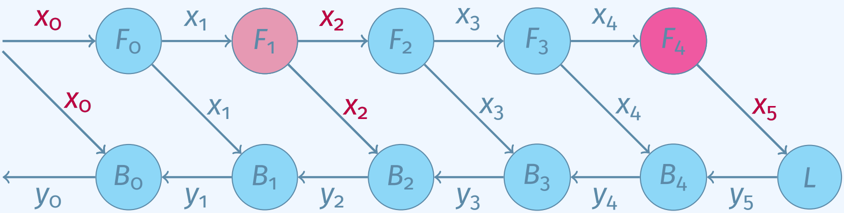
Memory :	x_0	x_2	x_3	

Execution : Store some / Recompute some



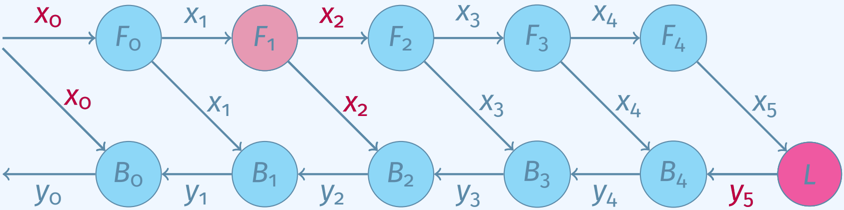
Memory :

x_0	x_2	x_4	



Memory :

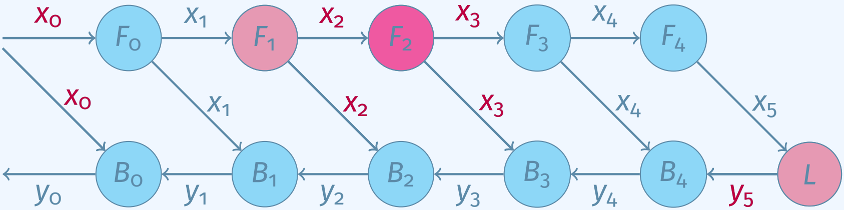
x_0	x_2	x_5	



Memory :

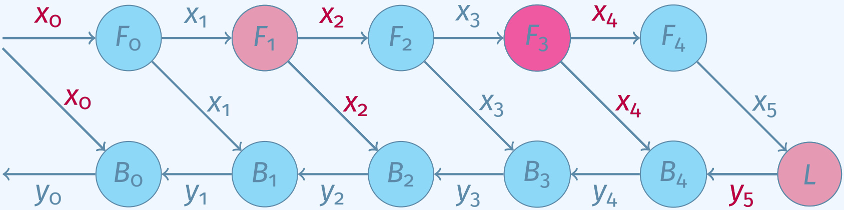
x_0	x_2		y_5

Execution : Store some / Recompute some



Memory :				
	x_0	x_2	x_3	y_5

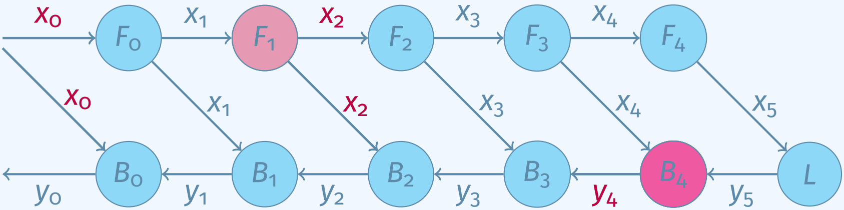
Execution : Store some / Recompute some



Memory :

x_0	x_2	x_4	y_5

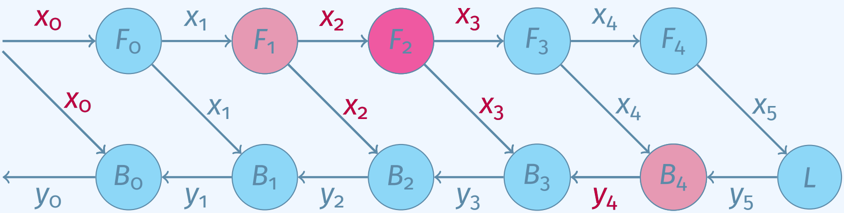
Execution : Store some / Recompute some



Memory :

x_0	x_2		y_4

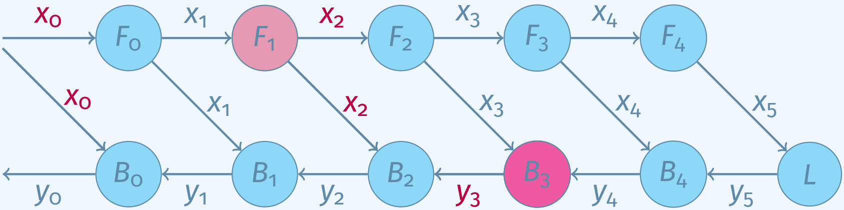
Execution : Store some / Recompute some



Memory :

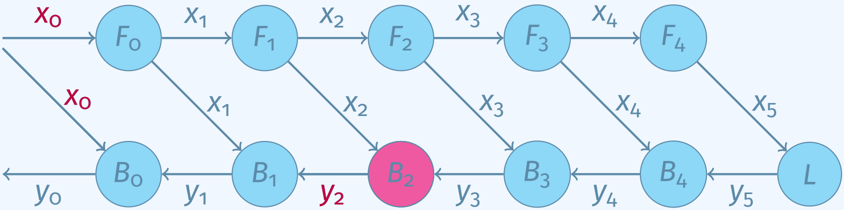
x_0	x_2	x_3	y_4

Execution : Store some / Recompute some



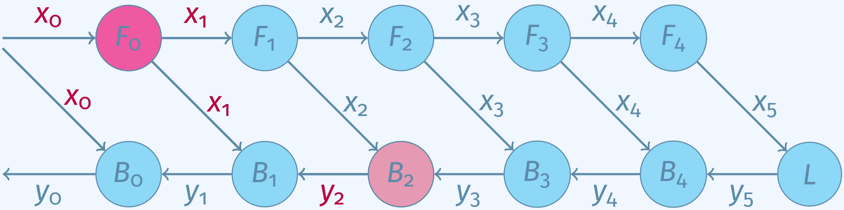
Memory :				
	x_0	x_2		y_3

Execution : Store some / Recompute some



Memory :

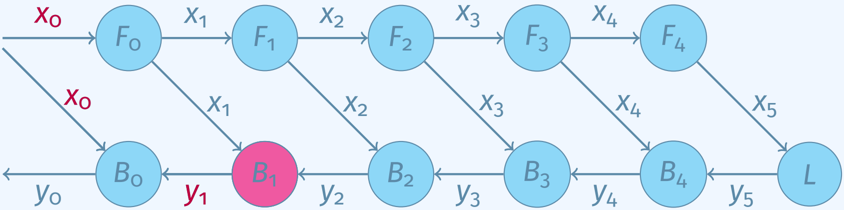
x_0			y_2



Memory :

x_0	x_1		y_2

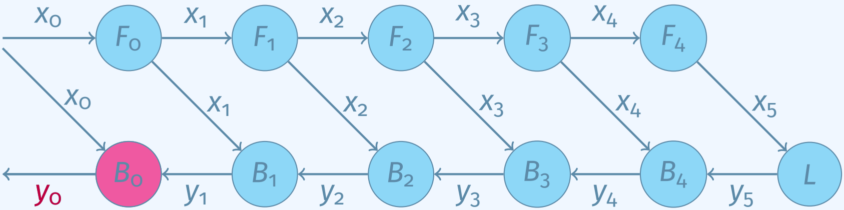
Execution : Store some / Recompute some



Memory :

x_0			y_1

Execution : Store some / Recompute some



Memory :

y_0			

For $l = 5$ forward steps

Strategy Store all (memory expensive):

- Peak Memory : 6
- Recomputation : 0

Strategy Recompute all (compute expensive):

- Peak Memory: 3
- Recomputation : 10

Strategy Store some / Recompute some (hybrid):

- Peak Memory: 4
- Recomputation : 4

Scheduling Problem

Application Parameters:

- l : number of forward steps in the BP graph
- x_i : memory size of intermediate value i
- wf, wb : computational cost of forward and backward steps

Memory Parameters:

- w_m: writing cost in memory
- r_m: reading cost in memory
- c_m : size of the memory

Question : Which intermediate data should we store in memory and which should we evict and recompute later?

Homogeneous model + single free memory

Application Parameters:

- l : arbitrary number of steps
- $\mathbf{x_i} = \mathbf{1}$: unitary size of intermediate value
- $\mathbf{wf_i} = \mathbf{1}$, $\mathbf{wb_i} = \mathbf{1}$: homogeneous forward and backward steps

Memory Parameters:

- $\mathbf{w_m} = \mathbf{0}$: free writing cost in memory
- $\mathbf{r_m} = \mathbf{0}$: free reading cost in memory
- c_m : arbitrary memory size

Griewank and Walther, 2000: $\mathit{Revolve}(l, c_m)$, optimal algorithm with c_m memory slots on homogeneous backpropagation graphs and free memory

Heterogeneous model + K memories

We consider K different memories with arbitrary reading and writing costs:

Application Parameters:

- l : arbitrary number of steps
- x_i : arbitrary memory size of intermediate value
- wf_i, wb_i : arbitrary computational cost for forward and backward steps

Memory Parameters:

- $\mathbf{w_m^{(k)}}$: writing cost into memory k
- $\mathbf{r_m^{(k)}}$: reading cost from memory k
- $\mathbf{c_m^{(k)}}$: size of memory k

Aupy, Herrmann, Hovland, Robert, 2015: Optimal algorithm for two level of storage: cheap bounded memory and costly unbounded disks.
Aupy, Herrmann, 2019: Library of optimal schedules for any number of storage level.

Parallel Processing of Backpropagation graphs

There are multiple parallelization techniques for Deep Neural Networks:

- **Model parallelism:** the model is partitioned on the architecture
  - Intra-layer parallelism : partition individual layers across workers
  - Inter-layer parallelism : pipelining
  - ...
- **Data parallelism:** the model is replicated on several workers, and each worker computes a micro-batch
  - ZeroDP
  - Fully-shared DP
  - ...

How do we optimize memory usage in these parallel frameworks to train deeper networks or bigger batches? $\Rightarrow$ INTERNSHIP

Refrences

[1] Liu, J. W.
On the storage requirement in the out-of-core multifrontal method for sparse factorization.
ACM Transactions on Mathematical Software (TOMS), 1986

[2] Liu, J. W.
An application of generalized tree pebbling to sparse matrix factorization.
SIAM Journal on Algebraic Discrete Methods, 1987

[3] Agullo, E., Guermouche, A., L'Excellent, J. Y.
Reducing the I/O volume in sparse out-of-core multifrontal methods.
SIAM Journal on Scientific Computing, 2010

[4] Griewank, A., Walther, A.
Revolve: an implementation of checkpointing for the reverse or adjoint mode of computational differentiation.
ACM Transactions on Mathematical Software (TOMS), 2000

[5] Aupy, G., Herrmann, J., Hovland, P., Robert, Y.
Optimal multistage algorithm for adjoint computation.
SIAM Journal on Scientific Computing, 2016

[6] Aupy G., Herrmann, J.
H-Revolve: a framework for adjoint computation on synchronous hierarchical platforms.
ACM Transactions on Mathematical Software (TOMS), 2020

[7] Huang, Y., Cheng, Y., Bapna, A., Firat, O., Chen, D., Chen, M., ...
Gpipe: Efficient training of giant neural networks using pipeline parallelism.
Advances in neural information processing systems, 2019