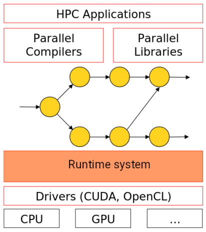
Dynamic Task Graph Adaptation with Recursive Tasks

Nathalie Furmento, Abdou Guermouche, Gwenolé Lucas, Thomas Morin, Samuel Thibault, Pierre-André Wacrenier

February 2024

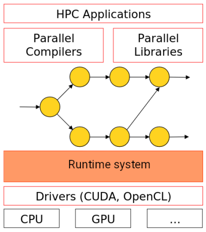
Task-based Programming

- Motivations:
  - > Portable frameworks.
  - > Exploit complex architectures.
- Applications: Directed Acyclic Graph (DAG).
- Runtime systems: scheduling, data management, communications, . . .



Task-based Programming

- Motivations:
  - Portable frameworks.
  - Exploit complex architectures.
- Applications: Directed Acyclic Graph (DAG).
- Runtime systems: scheduling, data management, communications, . . .

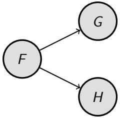


STF: Sequential Task Flow

- Dependencies:
  - Automatically inferred.
  - Order of submission.

```
F(a)
G(a, b)
H(a, c)
```

```
submit(F, a:RW)
submit(G, a:R, b:RW)
submit(H, a:R, c:RW)
wait_tasks_completion()
```



Submission

- Overhead: large number of non-ready tasks.
- Bottleneck: sequential insertion.
- Adaptability ? static task graphs.

Submission

- Overhead: large number of non-ready tasks.
- Bottleneck: sequential insertion.
- Adaptability ? static task graphs.

$\Rightarrow$ How to create more dynamic task-graphs ?

Submission

- Overhead: large number of non-ready tasks.
- Bottleneck: sequential insertion.
- Adaptability ? static task graphs.

$\Rightarrow$ How to create more dynamic task-graphs ? $\Rightarrow$ Recursive tasks graphs !

Submission

- Overhead: large number of non-ready tasks.
- Bottleneck: sequential insertion.
- Adaptability ? static task graphs.

$\Rightarrow$ How to create more dynamic task-graphs ? $\Rightarrow$ Recursive tasks graphs !

Granularity

- GPUs versus CPUs.
- Lack of parallelism versus Steady State.

$\Rightarrow$ Steering granularity dynamically ?

Submission

- Overhead: large number of non-ready tasks.
- Bottleneck: sequential insertion.
- Adaptability ? static task graphs.

$\Rightarrow$ How to create more dynamic task-graphs ? $\Rightarrow$ Recursive tasks graphs !

Granularity

- GPUs versus CPUs.
- **Lack of parallelism versus Steady State.**

$\Rightarrow$ Steering granularity dynamically ?

Objectives

- Adapt task implementation *at runtime*.
- No spurious synchronization.

Principles

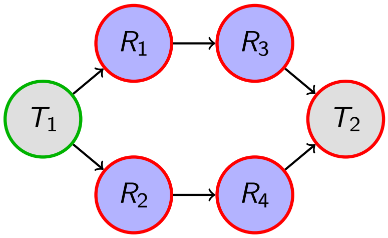
1. No limit for the hierarchy depth.
2. Fine-grained dependencies.
3. Transparent data management.

Objectives

- Adapt task implementation *at runtime*.
- No spurious synchronization.

Principles

1. No limit for the hierarchy depth.
2. Fine-grained dependencies.
3. Transparent data management.



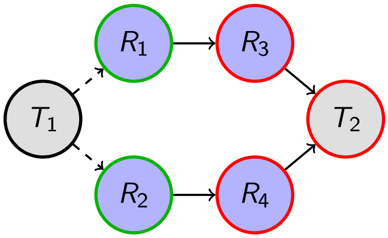
Objectives

- Adapt task implementation *at runtime*.
- No spurious synchronization.

Principles

1. No limit for the hierarchy depth.
2. Fine-grained dependencies.
3. Transparent data management.

- Recursive task execution:



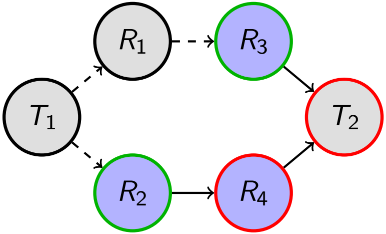
Objectives

- Adapt task implementation *at runtime*.
- No spurious synchronization.

Principles

1. No limit for the hierarchy depth.
2. Fine-grained dependencies.
3. Transparent data management.

- Recursive task execution:
  > Remain regular task.



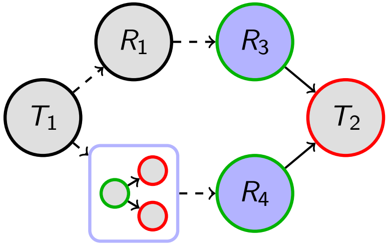
Objectives

- Adapt task implementation *at runtime*.
- No spurious synchronization.

Principles

1. No limit for the hierarchy depth.
2. Fine-grained dependencies.
3. Transparent data management.

- Recursive task execution:
  > Remain regular task.
  > Insert a subgraph: **split**.



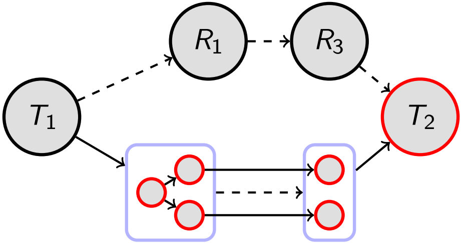
Objectives

- Adapt task implementation *at runtime*.
- No spurious synchronization.

Principles

1. No limit for the hierarchy depth.
2. Fine-grained dependencies.
3. Transparent data management.

- Recursive task execution:
  > Remain regular task.
  > Insert a subgraph: **split**.



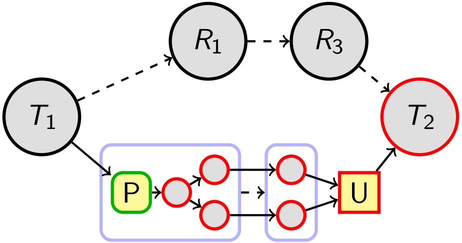
Objectives

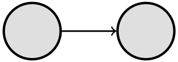
- Adapt task implementation *at runtime*.
- No spurious synchronization.

Principles

1. No limit for the hierarchy depth.
2. Fine-grained dependencies.
3. Transparent data management.
   > Automatic data partition.

- Recursive task execution:
  > Remain regular task.
  > Insert a subgraph: **split**.





Runtime	Fine-grain Dependencies	Automatic data Partition	Heterogeneity
TaskFlow			
PaRSEC			
IRIS			
OmpSs			
StarPU			

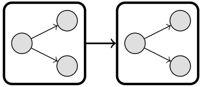


Figure: Barrier between parent tasks

Runtime	Fine-grain Dependencies	Automatic data Partition	Heterogeneity
TaskFlow	✗		
PaRSEC	✗		
IRIS	✗		
OmpSs			
StarPU			

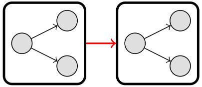


Figure: Barrier between parent tasks

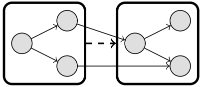


Figure: Fine-grain dependencies

Runtime	Fine-grain Dependencies	Automatic data Partition	Heterogeneity
TaskFlow	✗		
PaRSEC	✗		
IRIS	✗		
OmpSs	✓		
StarPU	✓		

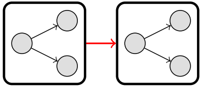


Figure: Barrier between parent tasks

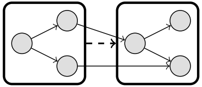


Figure: Fine-grain dependencies

Runtime	Fine-grain Dependencies	Automatic data Partition	Heterogeneity
TaskFlow	✗	✗	
PaRSEC	✗	✗	
IRIS	✗	✓	
OmpSs	✓		
StarPU	✓	✓	

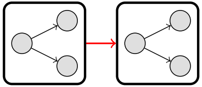


Figure: Barrier between parent tasks

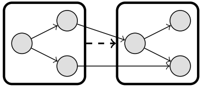


Figure: Fine-grain dependencies

Runtime	Fine-grain Dependencies	Automatic data Partition	Heterogeneity
TaskFlow	✗	✗	✓
PaRSEC	✗	✗	✓
IRIS	✗	✓	✓
OmpSs	✓		✗
StarPU	✓	✓	✓

Which task should we split?

When do we choose to split task?

Which task should we split?

Efficiency VS Completion Time

When do we choose to split task?

Which task should we split?

Efficiency VS Completion Time

When do we choose to split task?

Submission, execution, ...

Exploit informations

Exploit informations

1. Split efficiency.

Exploit informations

1. Split efficiency.
2. Current parallelism on Runtime System.

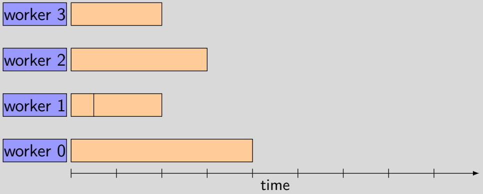
Exploit informations

1. **Split efficiency.**
2. Current parallelism on Runtime System.

Exploit informations

1. **Split efficiency.**
2. Current parallelism on Runtime System.

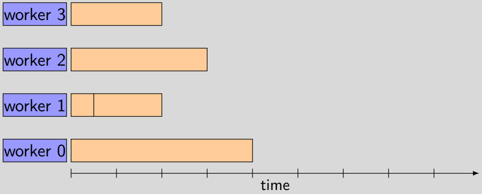
Split the task - Gantt chart



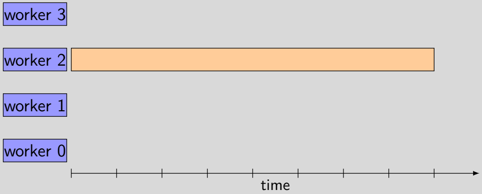
Exploit informations

1. **Split efficiency.**
2. Current parallelism on Runtime System.

Split the task - Gantt chart



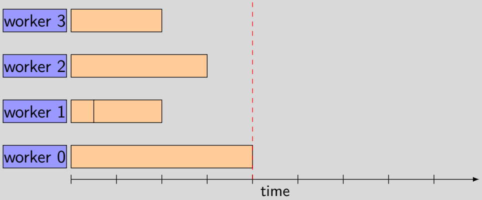
Not split the task - Gantt chart



Exploit informations

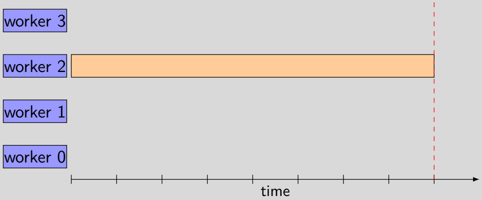
1. **Split efficiency.**
2. Current parallelism on Runtime System.

Split the task - Gantt chart



Completion time: 4 units.

Not split the task - Gantt chart

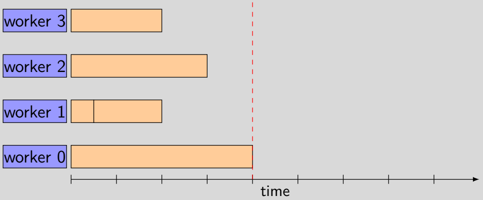


Completion time: 8 units.

Exploit informations

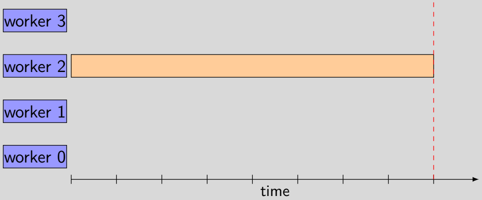
1. **Split efficiency.**
2. Current parallelism on Runtime System.

Split the task - Gantt chart



Completion time: 4 units.
Cumulated time: 11 units.

Not split the task - Gantt chart

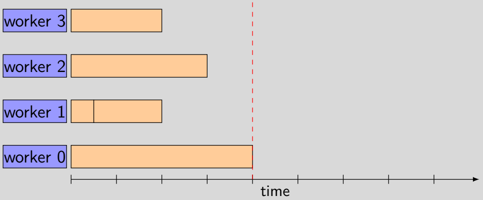


Completion time: 8 units.
Cumulated time: 8 units.

Exploit informations

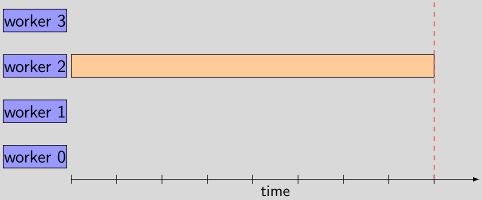
1. **Split efficiency.**
2. Current parallelism on Runtime System.

Split the task - Gantt chart



Completion time: 4 units.
Cumulated time: 11 units.

Not split the task - Gantt chart



Completion time: 8 units.
Cumulated time: 8 units.

Completion Time versus Efficiency

Exploit informations

1. Split efficiency.
2. Current parallelism on Runtime System.

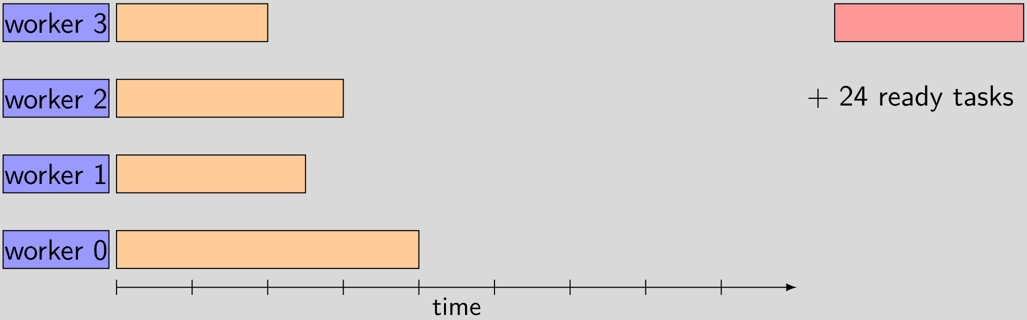
Exploit informations

1. Split efficiency.
2. **Current parallelism on Runtime System.**

Exploit informations

1. Split efficiency.
2. **Current parallelism on Runtime System.**

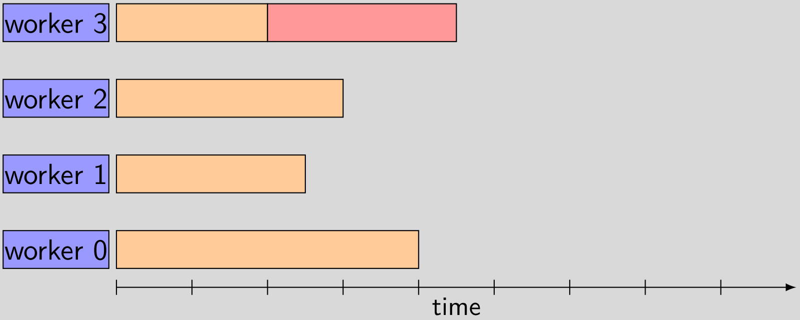
Situation 1: Steady State



Exploit informations

1. Split efficiency.
2. **Current parallelism on Runtime System.**

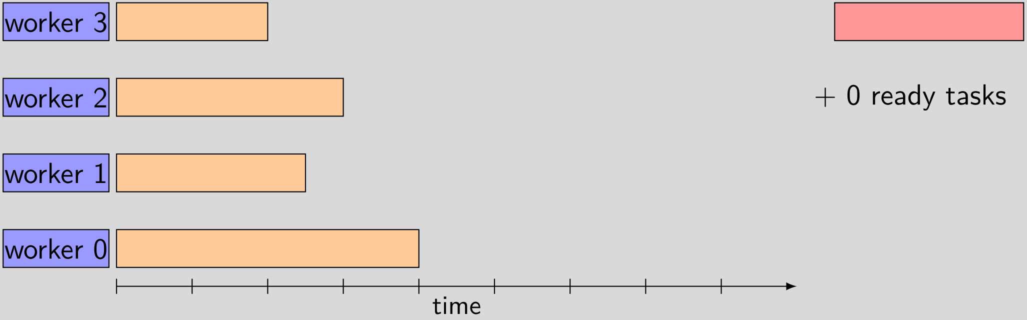
Situation 1: Steady State



Exploit informations

1. Split efficiency.
2. **Current parallelism on Runtime System.**

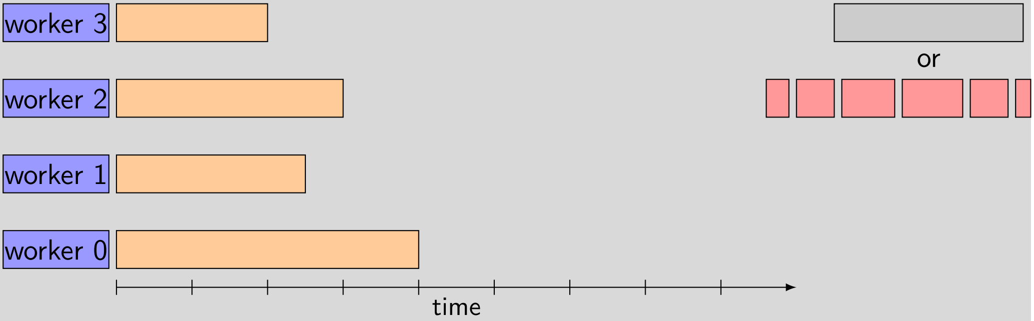
Situation 2



Exploit informations

1. Split efficiency.
2. **Current parallelism on Runtime System.**

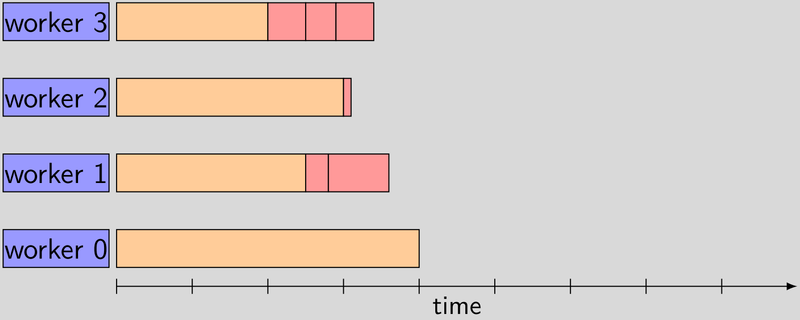
Situation 2 : Starvation



Exploit informations

1. Split efficiency.
2. **Current parallelism on Runtime System.**

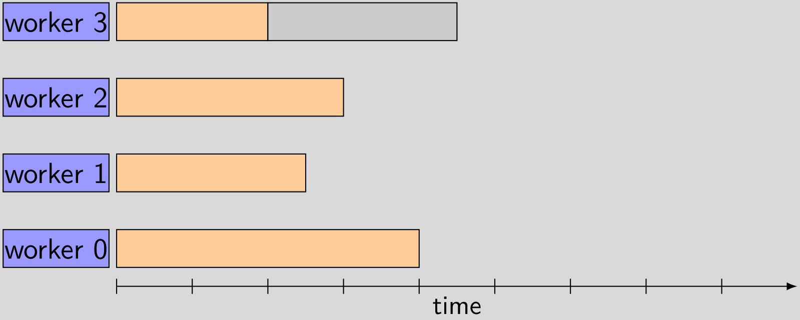
Situation 2 : Starvation



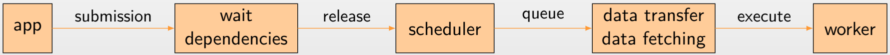
Exploit informations

1. Split efficiency.
2. **Current parallelism on Runtime System.**

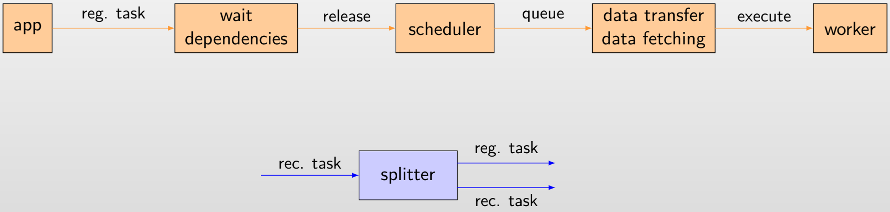
Situation 2 : Starvation



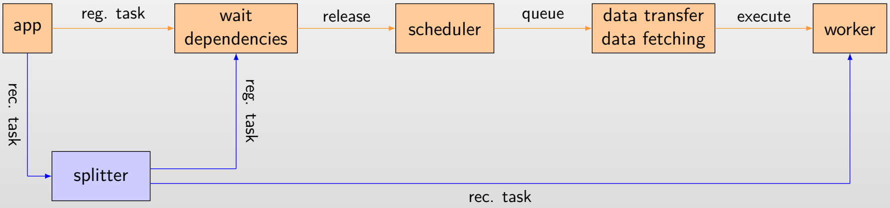
Task life path



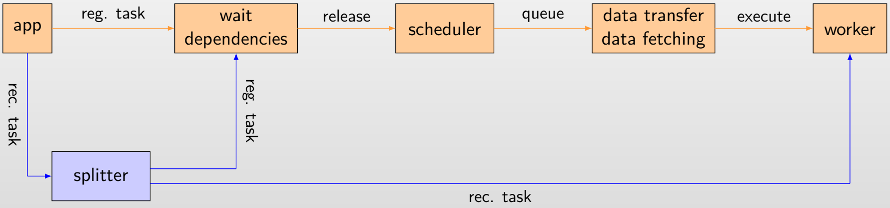
Adding the splitter



Position of the splitter - at submission

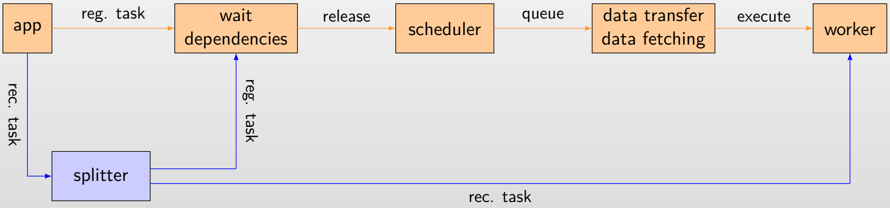


Position of the splitter - at submission



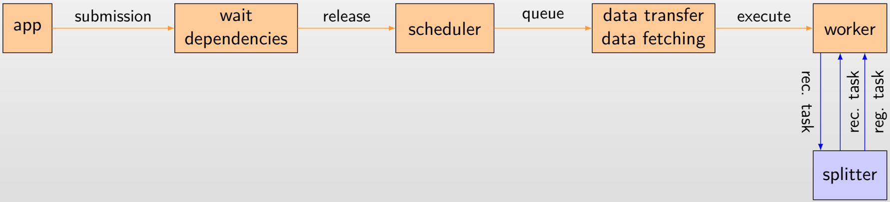
- Easy.

Position of the splitter - at submission

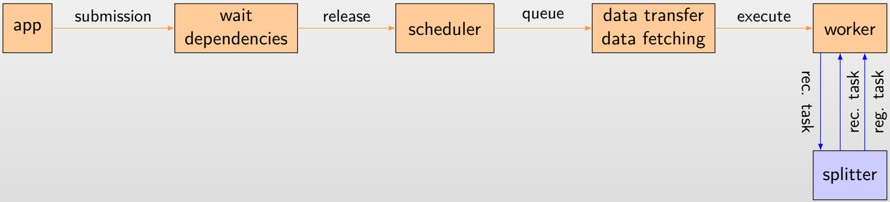


- Easy.
- Lack of information.

Position of the splitter - Execution

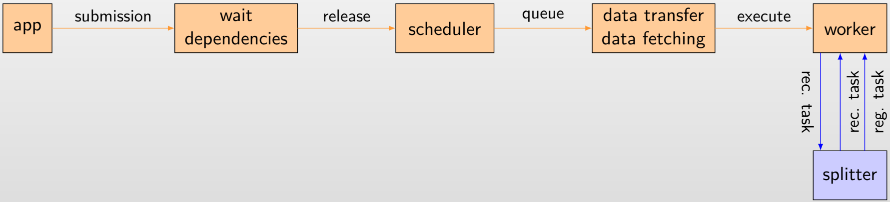


Position of the splitter - Execution



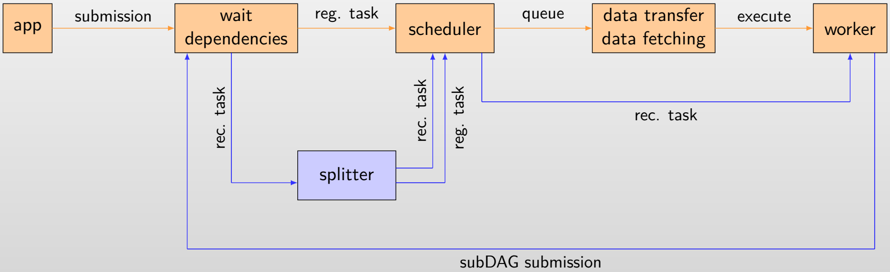
- Runtime information.

Position of the splitter - Execution



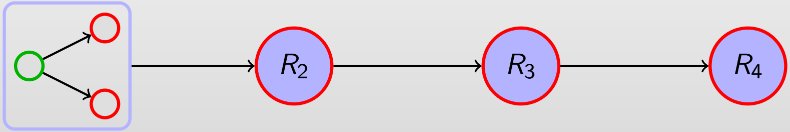
- Runtime information.
- Useless data transfer: cancel decision.

Position of the splitter - trade-off

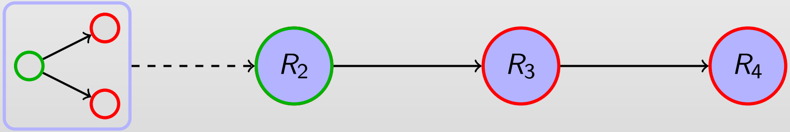


Recursive Task Path - Release dependency

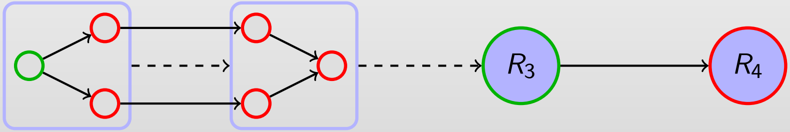
Recursive Task Path - Release dependency



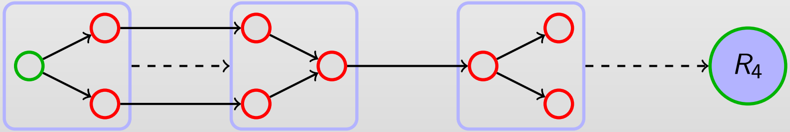
Recursive Task Path - Release dependency



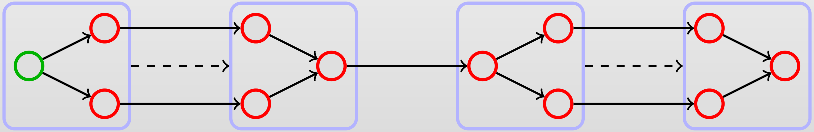
Recursive Task Path - Release dependency



Recursive Task Path - Release dependency

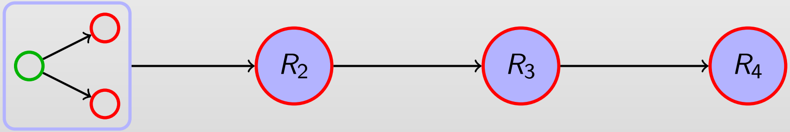


Recursive Task Path - Release dependency

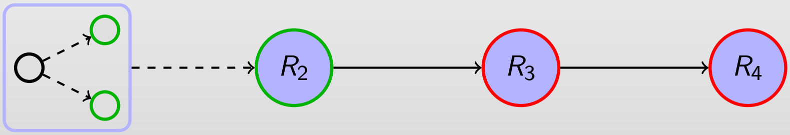


Over-synchronization solution

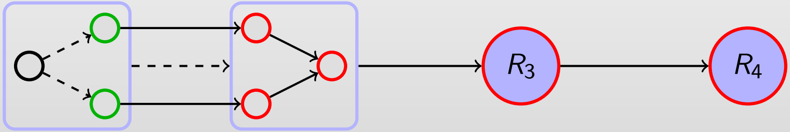
Over-synchronization solution



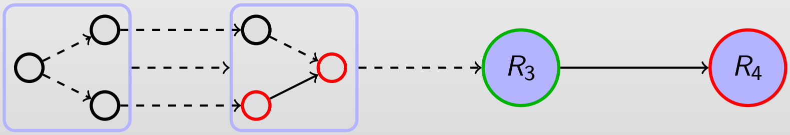
Over-synchronization solution



Over-synchronization solution



Over-synchronization solution



The tests were run on PlaFRIM's `bora` nodes:

- 2x 18-core Cascade Lake Intel Xeon Skylake Gold 6240 @ 2.6 GHz
- 192 GB (5.3 GB/core) (@2933 MHz)
- Scheduler : Locality-aware Work-Stealing (LWS)

The tests were run on PlaFRIM's `bora` nodes:

- 2x 18-core Cascade Lake Intel Xeon Skylake Gold 6240 @ 2.6 GHz
- 192 GB (5.3 GB/core) (@2933 MHz)
- Scheduler : Locality-aware Work-Stealing (LWS)

Tile sizes choosen :

- 1120 : "big" : the most efficient.
- 280 : "small": no lack of parallelism.
- 560 : "mid": trade-off.

The tests were run on PlaFRIM's `bora` nodes:

- 2x 18-core Cascade Lake Intel Xeon Skylake Gold 6240 @ 2.6 GHz
- 192 GB (5.3 GB/core) (@2933 MHz)
- Scheduler : Locality-aware Work-Stealing (LWS)

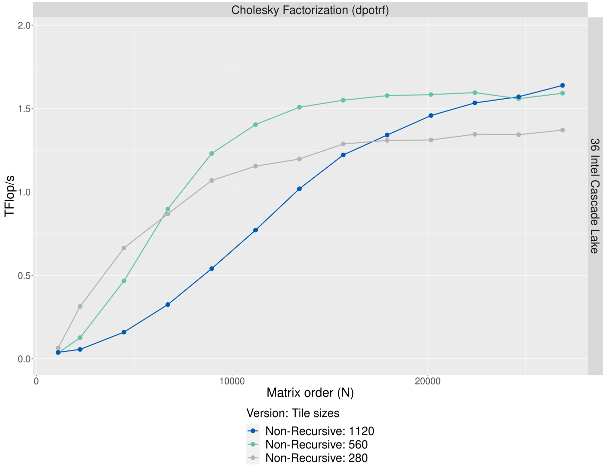
Tile sizes choosen :

- 1120 : "big" : the most efficient.
- 280 : "small": no lack of parallelism.
- 560 : "mid": trade-off.

We split a task if:

- $N_{ready} \leq 4N_{cores}$
- Split efficiency $\geq 50\%$.

Benchmarks - Cholesky Factorization



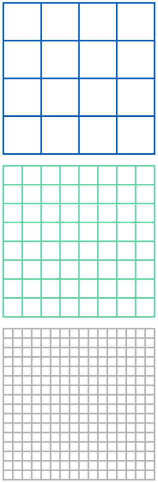
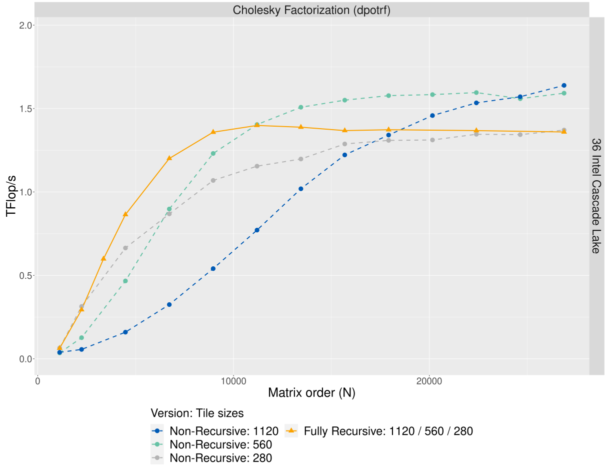


Figure: Performance comparison between different Cholesky Factorization versions.



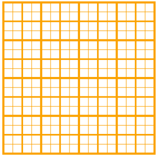
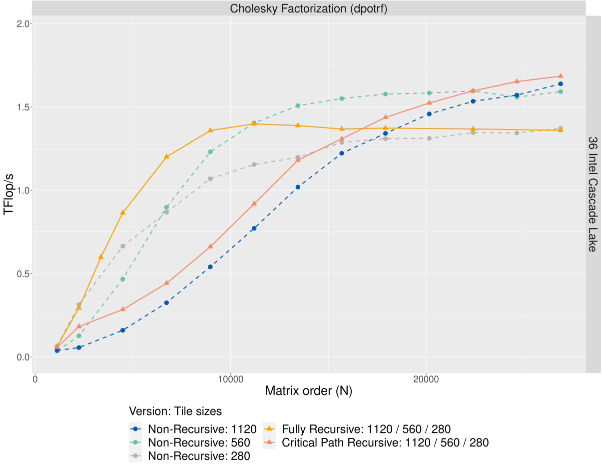


Figure: Performance comparison between different Cholesky Factorization versions.

Benchmarks - Cholesky Factorization



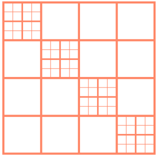
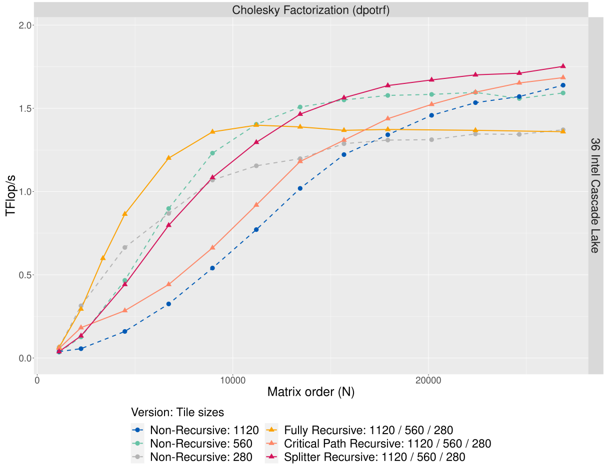


Figure: Performance comparison between different Cholesky Factorization versions.



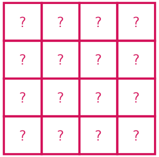


Figure: Performance comparison between different Cholesky Factorization versions.

Benchmarks - Cholesky Factorization

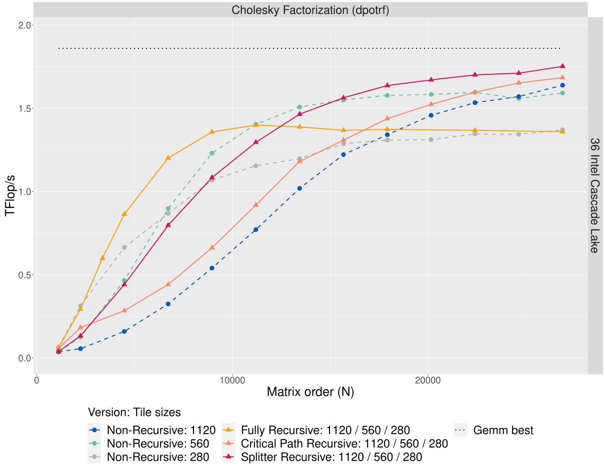


Figure: Performance comparison between different Cholesky Factorization versions.

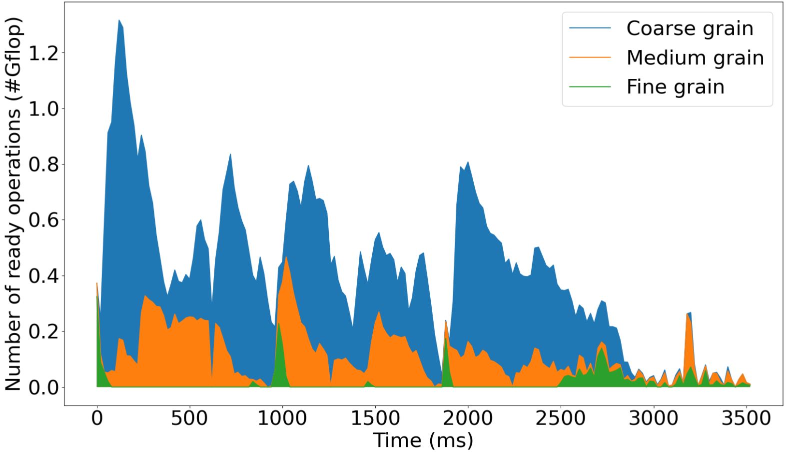
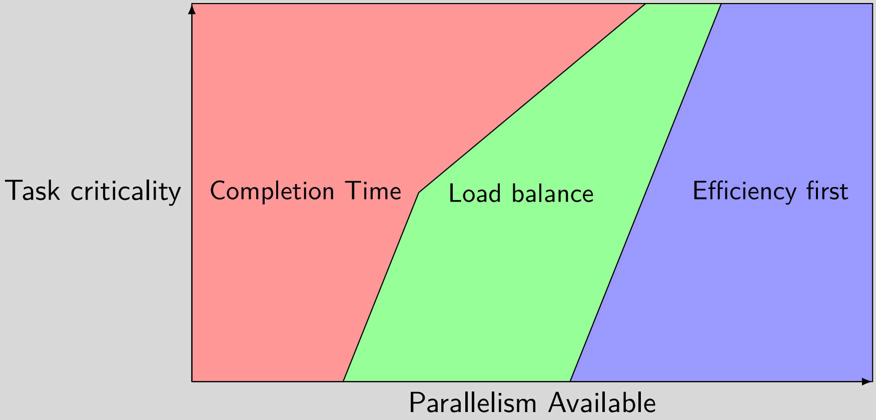


Figure: Flops evolution according to execution time during recursive-splitter Cholesky Factorization execution, with matrix of size 26880.

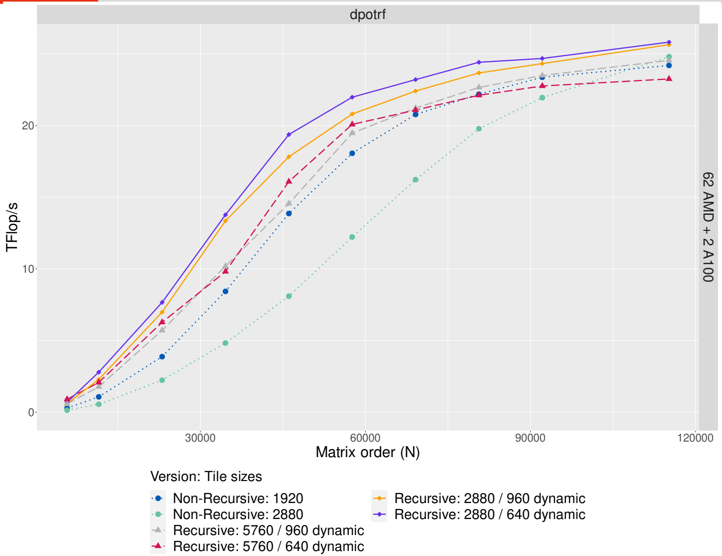
- Recursive tasks:
  - Insert subgraph at runtime.
  - More dynamic DAG.
- Splitting task dynamically brings different questions:
  - Which task sould we split.
  - When do we choose to split.

Future Work

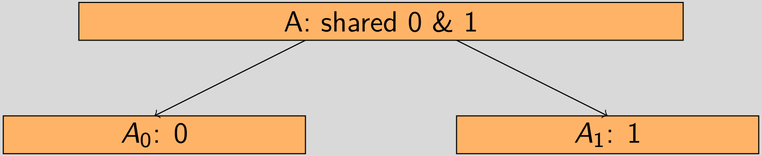
- Scheduling questions:
  - How should we split tasks ?
- Extend current work:
  - Heterogeneous platforms.
  - Distributed recursive tasks.


Task criticality
Completion Time
Load balance
Efficiency first
Parallelism Available

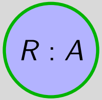
Heterogeneous



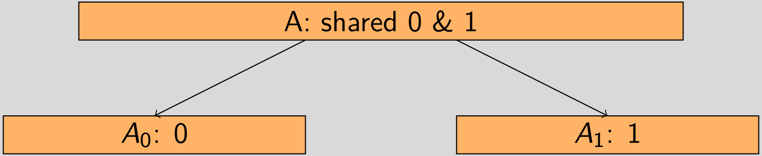
Shared data



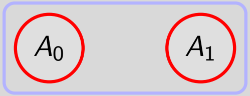
Auto-pruning



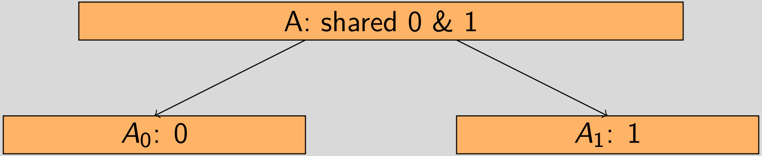
Shared data



Auto-pruning



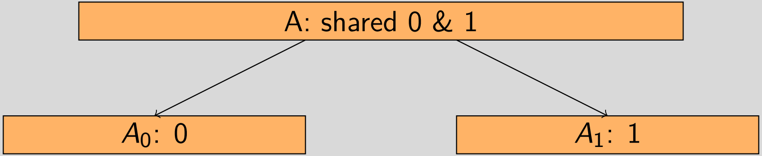
Shared data



Auto-pruning

Node 0 :

Shared data



Auto-pruning

Node 1 :