Programming Heterogeneous & Distributed Architectures using Hierarchical Tasks

Mathieu Faverge, Nathalie Furmento, Abdou Guermouche, Gwenolé Lucas, Thomas Morin, Raymond Namyst, Samuel Thibault, Pierre-André Wacrenier

March 2023

Introduction - Context

Task Based Programming

- Task-based programming aims to provide portable frameworks capable of exploiting complex architectures.
- Applications are presented as a Directed Acyclic Graph (DAG).
- Runtime systems handle scheduling, communications, . . .

Introduction - Context

Task Based Programming

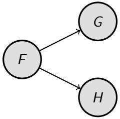
- Task-based programming aims to provide portable frameworks capable of exploiting complex architectures.
- Applications are presented as a Directed Acyclic Graph (DAG).
- Runtime systems handle scheduling, communications, . . .

StarPU

- StarPU rely on the *Sequential Task Flow* (STF) to create its DAGs.
- The STF infers dependencies from the order of submission of the tasks and data access modes.

```
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()
```



Introduction - Limitations. . .

. . . of tasks based programming

- GPUs and CPUs work best on different granularities.
- Some applications are too irregular to fit in a predetermined task-graph.
- Static task graphs limit adaptability during runtime.

. . . of the STF model

- Runtime overhead induced by a large number of non-ready tasks.
- The sequential insertion of tasks can bottleneck the execution of large DAG.

Introduction - Limitations. . .

. . . of tasks based programming

- GPUs and CPUs work best on different granularities.
- Some applications are too irregular to fit in a predetermined task-graph.
- Static task graphs limit adaptability during runtime.

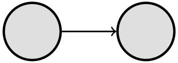
. . . of the STF model

- Runtime overhead induced by a large number of non-ready tasks.
- The sequential insertion of tasks can bottleneck the execution of large DAG.

⇒ How to create more dymanic task-graphs?

Introduction - State of the Art

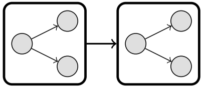
The following runtimes aim at creating more dynamic task-graphs by replacing tasks with an equivalent subgraph:



Runtime	Fine-grain Dependencies	Automatic Data Management	Heterogeneity
TaskFlow			
PaRSEC			
OmpSs			
IRIS			
libtask			
Our contribution			

Introduction - State of the Art

The following runtimes aim at creating more dynamic task-graphs by replacing tasks with an equivalent subgraph:



Runtime	Fine-grain Dependencies	Automatic Data Management	Heterogeneity
TaskFlow	✗	✗	✓
PaRSEC			
OmpSs			
IRIS			
libtask			
Our contribution			

The following runtimes aim at creating more dynamic task-graphs by replacing tasks with an equivalent subgraph:

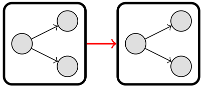


Figure: Barrier between parent tasks

Runtime	Fine-grain Dependencies	Automatic Data Management	Heterogeneity
TaskFlow	✗	✗	✓
PaRSEC	✗	✗	✓
OmpSs			
IRIS			
libtask			
Our contribution			

Introduction - State of the Art

The following runtimes aim at creating more dynamic task-graphs by replacing tasks with an equivalent subgraph:

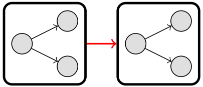


Figure: Barrier between parent tasks

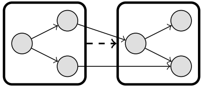


Figure: Fine-grain dependencies

Runtime	Fine-grain Dependencies	Automatic Data Management	Heterogeneity
TaskFlow	✗	✗	✓
PaRSEC	✗	✗	✓
OmpSs	✓	✗	✗
IRIS			
libtask			
Our contribution			

Introduction - State of the Art

The following runtimes aim at creating more dynamic task-graphs by replacing tasks with an equivalent subgraph:

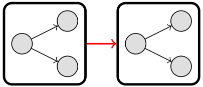


Figure: Barrier between parent tasks

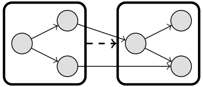


Figure: Fine-grain dependencies

Runtime	Fine-grain Dependencies	Automatic Data Management	Heterogeneity
TaskFlow	✗	✗	✓
PaRSEC	✗	✗	✓
OmpSs	✓	✗	✗
IRIS	✗	✓	✓
libtask			
Our contribution			

The following runtimes aim at creating more dynamic task-graphs by replacing tasks with an equivalent subgraph:

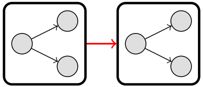


Figure: Barrier between parent tasks

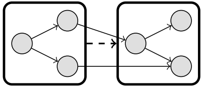


Figure: Fine-grain dependencies

Runtime	Fine-grain Dependencies	Automatic Data Management	Heterogeneity
TaskFlow	✗	✗	✓
PaRSEC	✗	✗	✓
OmpSs	✓	✗	✗
IRIS	✗	✓	✓
libtask	✗	✓	✓/✗
Our contribution			

Introduction - State of the Art

The following runtimes aim at creating more dynamic task-graphs by replacing tasks with an equivalent subgraph:

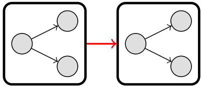


Figure: Barrier between parent tasks

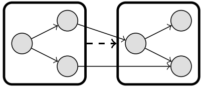


Figure: Fine-grain dependencies

Runtime	Fine-grain Dependencies	Automatic Data Management	Heterogeneity
TaskFlow	✗	✗	✓
PaRSEC	✗	✗	✓
OmpSs	✓	✗	✗
IRIS	✗	✓	✓
libtask	✗	✓	✓/✗
Our contribution	✓	✓	✓

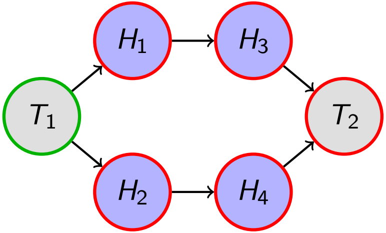
Hierarchical Tasks in StarPU

Objectives

- Adapt task granularity to devices
- Reduce the amount of active tasks in StarPU
- Dynamically adapt task implementation at runtime

Principles

1. No limit for the hierarchy depth
2. Data management is transparent to the programmer
3. Dependencies connect tasks at the finest level possible



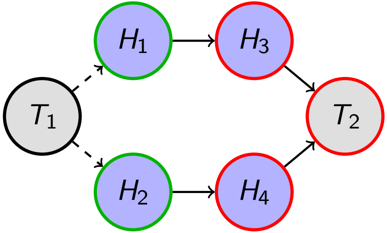
Hierarchical Tasks in StarPU

Objectives

- Adapt task granularity to devices
- Reduce the amount of active tasks in StarPU
- Dynamically adapt task implementation at runtime

Principles

1. No limit for the hierarchy depth
2. Data management is transparent to the programmer
3. Dependencies connect tasks at the finest level possible



Hierarchical Tasks in StarPU

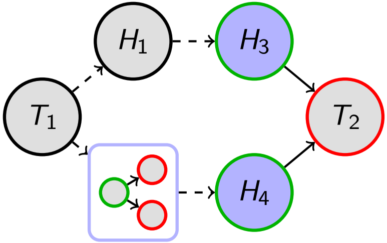
Objectives

- Adapt task granularity to devices
- Reduce the amount of active tasks in StarPU
- Dynamically adapt task implementation at runtime

Principles

1. No limit for the hierarchy depth
2. Data management is transparent to the programmer
3. Dependencies connect tasks at the finest level possible

- When executed, a hierarchical task *can* insert a subgraph.



Hierarchical Tasks in StarPU

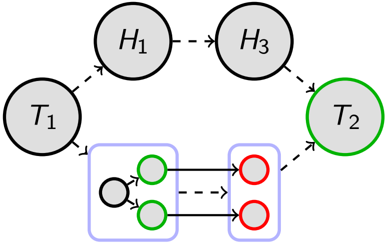
Objectives

- Adapt task granularity to devices
- Reduce the amount of active tasks in StarPU
- Dynamically adapt task implementation at runtime

Principles

1. No limit for the hierarchy depth
2. Data management is transparent to the programmer
3. Dependencies connect tasks at the finest level possible

- When executed, a hierarchical task *can* insert a subgraph.
- The dependencies in the subgraphs are infered from the data.



Hierarchical Tasks in StarPU

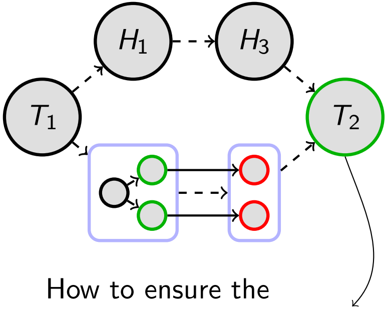
Objectives

- Adapt task granularity to devices
- Reduce the amount of active tasks in StarPU
- Dynamically adapt task implementation at runtime

Principles

1. No limit for the hierarchy depth
2. Data management is transparent to the programmer
3. Dependencies connect tasks at the finest level possible

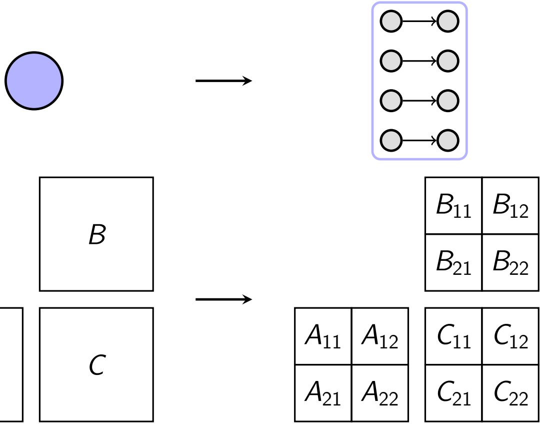
- When executed, a hierarchical task *can* insert a subgraph.
- The dependencies in the subgraphs are infered from the data.



How to ensure the correctness of the DAG?

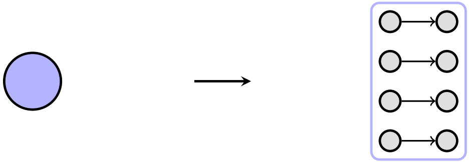
Hierarchical Tasks in StarPU

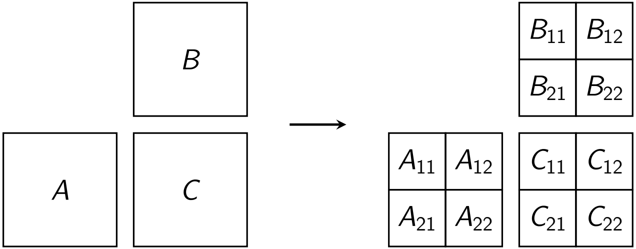
Matrix-matrix multiplication $C = C + A \times B$:



Hierarchical Tasks in StarPU

Matrix-matrix multiplication $C = C + A \times B$:





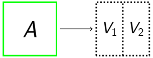
$\Rightarrow$ How to adapt data partitioning to suit hierarchical tasks?

Automatic Data Manager

```
[Data registration]
register(A)
partition_plan(A, vfilter, V)
partition_plan(A, hfilter, H)









.
```

Automatic Data Manager

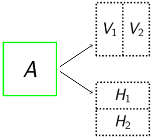
- The user describes data partitioning through *plan* operations.



```
[Data registration]
register(A)
partition_plan(A, vfilter, V)
partition_plan(A, hfilter, H)









.
```

Automatic Data Manager

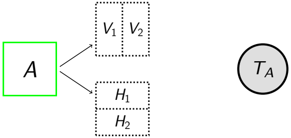
- The user describes data partitioning through *plan* operations.
- Multiple *plans* for the same data piece can coexist.



```
[Data registration]
register(A)
partition_plan(A, vfilter, V)
partition_plan(A, hfilter, H)









.
```

Automatic Data Manager

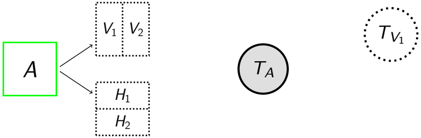
- The user describes data partitioning through *plan* operations.
- Multiple *plans* for the same data piece can coexist.



```
[Data registration]
register(A)
partition_plan(A, vfilter, V)
partition_plan(A, hfilter, H)

[Modification]
submit(T, A:RW)
for (i = 1 to NPARTS)
    submit(T, V[i]:RW)





.
```

Automatic Data Manager

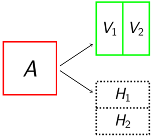
- The user describes data partitioning through *plan* operations.
- Multiple *plans* for the same data piece can coexist.
- Partitioning tasks are inserted to ensure that each task will use a valid data.

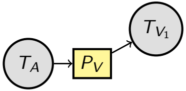


```
[Data registration]
register(A)
partition_plan(A, vfilter, V)
partition_plan(A, hfilter, H)

[Modification]
submit(T, A:RW)
for (i = 1 to NPARTS)
    submit(T, V[i]:RW)





.
```

Automatic Data Manager

- The user describes data partitioning through *plan* operations.
- Multiple *plans* for the same data piece can coexist.
- Partitioning tasks are inserted to ensure that each task will use a valid data.

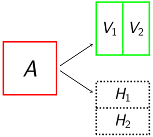


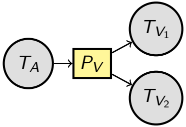


```
[Data registration]
register(A)
partition_plan(A, vfilter, V)
partition_plan(A, hfilter, H)

[Modification]
submit(T, A:RW)
for (i = 1 to NPARTS)
    submit(T, V[i]:RW)






.
```

Automatic Data Manager

- The user describes data partitioning through *plan* operations.
- Multiple *plans* for the same data piece can coexist.
- Partitioning tasks are inserted to ensure that each task will use a valid data.

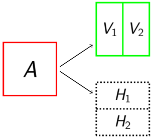


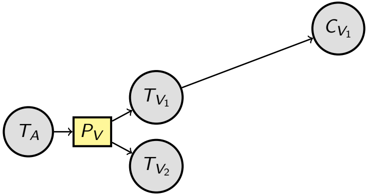


```
[Data registration]
register(A)
partition_plan(A, vfilter, V)
partition_plan(A, hfilter, H)

[Modification]
submit(T, A:RW)
for (i = 1 to NPARTS)
    submit(T, V[i]:RW)






.
```

Automatic Data Manager

- The user describes data partitioning through *plan* operations.
- Multiple *plans* for the same data piece can coexist.
- Partitioning tasks are inserted to ensure that each task will use a valid data.

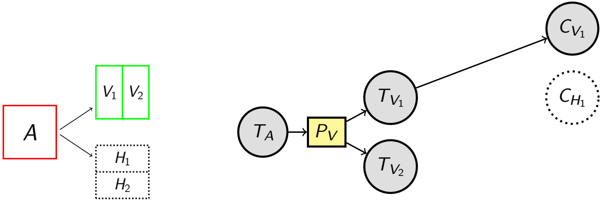




```
[Data registration]
register(A)
partition_plan(A, vfilter, V)
partition_plan(A, hfilter, H)

[Modification]
submit(T, A:RW)
for (i = 1 to NPARTS)
    submit(T, V[i]:RW)
[Verification]
for (i = 1 to NPARTS)
    submit(C, V[i]:R)
    submit(C, H[i]:R)

wait_tasks_completion()
```

Automatic Data Manager

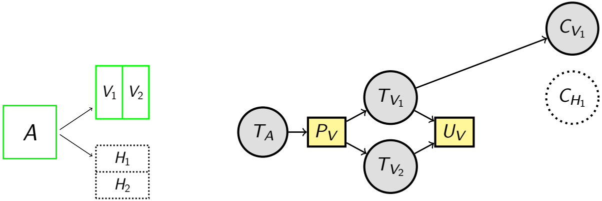
- The user describes data partitioning through *plan* operations.
- Multiple *plans* for the same data piece can coexist.
- Partitioning tasks are inserted to ensure that each task will use a valid data.



```
[Data registration]
register(A)
partition_plan(A, vfilter, V)
partition_plan(A, hfilter, H)

[Modification]
submit(T, A:RW)
for (i = 1 to NPARTS)
    submit(T, V[i]:RW)
[Verification]
for (i = 1 to NPARTS)
    submit(C, V[i]:R)
    submit(C, H[i]:R)

wait_tasks_completion()
```

Automatic Data Manager

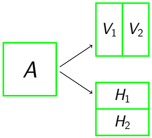
- The user describes data partitioning through *plan* operations.
- Multiple *plans* for the same data piece can coexist.
- Partitioning tasks are inserted to ensure that each task will use a valid data.

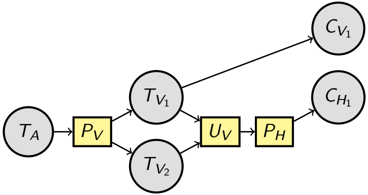


```
[Data registration]
register(A)
partition_plan(A, vfilter, V)
partition_plan(A, hfilter, H)

[Modification]
submit(T, A:RW)
for (i = 1 to NPARTS)
    submit(T, V[i]:RW)
[Verification]
for (i = 1 to NPARTS)
    submit(C, V[i]:R)
    submit(C, H[i]:R)

wait_tasks_completion()
```

Automatic Data Manager

- The user describes data partitioning through *plan* operations.
- Multiple *plans* for the same data piece can coexist.
- Partitioning tasks are inserted to ensure that each task will use a valid data.

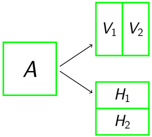


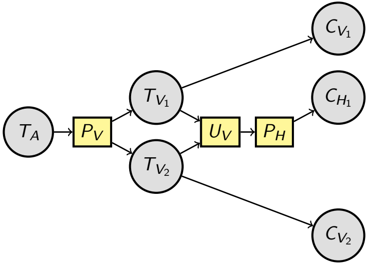


```
[Data registration]
register(A)
partition_plan(A, vfilter, V)
partition_plan(A, hfilter, H)

[Modification]
submit(T, A:RW)
for (i = 1 to NPARTS)
    submit(T, V[i]:RW)
[Verification]
for (i = 1 to NPARTS)
    submit(C, V[i]:R)
    submit(C, H[i]:R)

wait_tasks_completion()
```

Automatic Data Manager

- The user describes data partitioning through *plan* operations.
- Multiple *plans* for the same data piece can coexist.
- Partitioning tasks are inserted to ensure that each task will use a valid data.

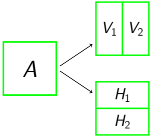


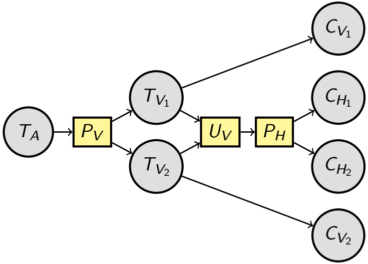


```
[Data registration]
register(A)
partition_plan(A, vfilter, V)
partition_plan(A, hfilter, H)

[Modification]
submit(T, A:RW)
for (i = 1 to NPARTS)
    submit(T, V[i]:RW)
[Verification]
for (i = 1 to NPARTS)
    submit(C, V[i]:R)
    submit(C, H[i]:R)

wait_tasks_completion()
```

Automatic Data Manager

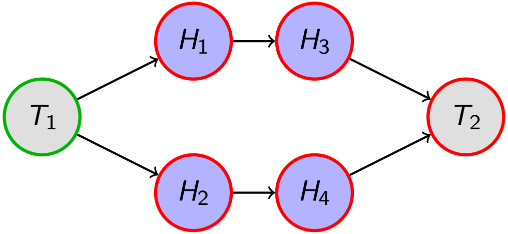
- The user describes data partitioning through *plan* operations.
- Multiple *plans* for the same data piece can coexist.
- Partitioning tasks are inserted to ensure that each task will use a valid data.



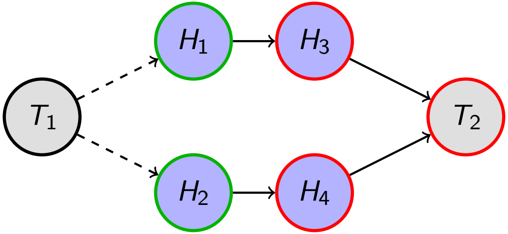


```
[Data registration]
register(A)
partition_plan(A, vfilter, V)
partition_plan(A, hfilter, H)

[Modification]
submit(T, A:RW)
for (i = 1 to NPARTS)
    submit(T, V[i]:RW)
[Verification]
for (i = 1 to NPARTS)
    submit(C, V[i]:R)
    submit(C, H[i]:R)

wait_tasks_completion()
```

Correctness of a Hierarchical DAG

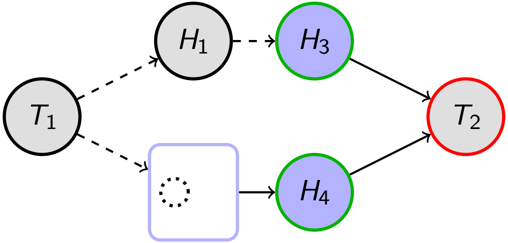


Correctness of a Hierarchical DAG



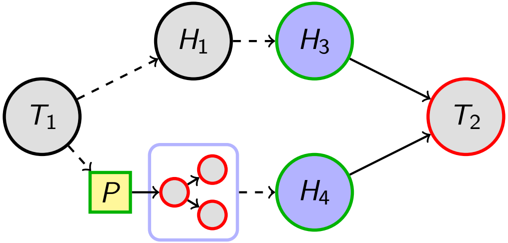
Correctness of a Hierarchical DAG

- Partitioning tasks are added, if needed, at the submission of a subtask.



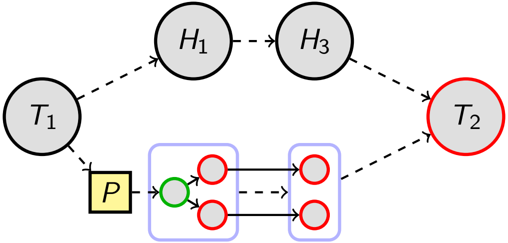
Correctness of a Hierarchical DAG

- Partitioning tasks are added, if needed, at the submission of a subtask.



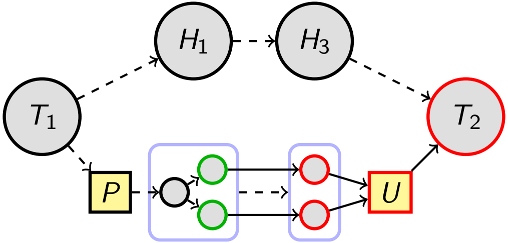
Correctness of a Hierarchical DAG

- Partitioning tasks are added, if needed, at the submission of a subtask.



Correctness of a Hierarchical DAG

- Partitioning tasks are added, if needed, at the submission of a subtask.
- Unpartitioning tasks are added, if needed, before a regular task. They enforce the correctness of the DAG.



Benchmarks - Overhead

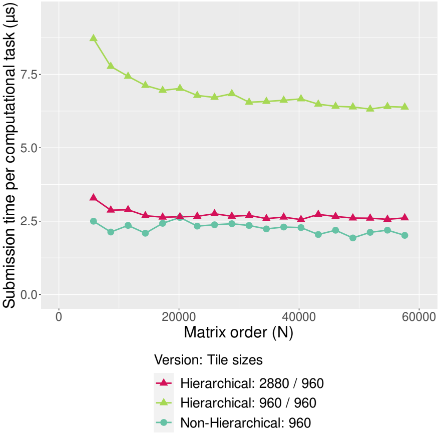


Figure: Submission cost of computational tasks for the matrix-matrix multiplication kernel.

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

Name	Processor	GPU	Memory
INTEL-V100	2 x INTEL XEON GOLD 6142, 16 cores, 2.6GHz	2 x NVIDIA V100 (16GB)	384GB
AMD-A100	2 x AMD ZEN3 EPYC 7513, 32 cores, 2.6GHz	2 x NVIDIA A100 (40GB)	512GB

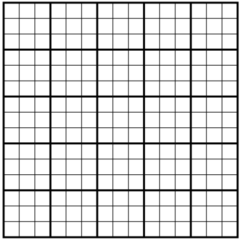


Figure: Full matrix partitioning.

Benchmarks - Matrix-Matrix Multiplication

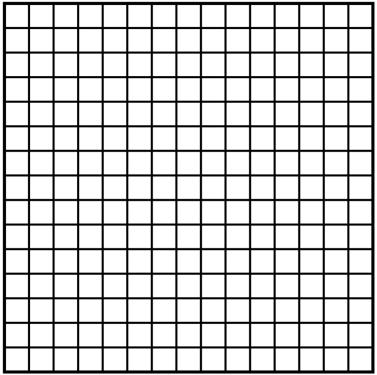


Figure: Tile size of 960.

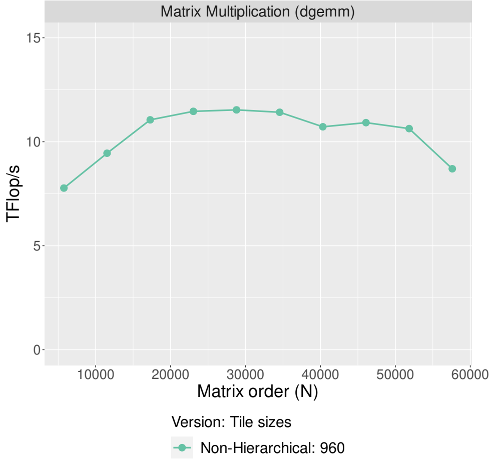


Figure: Matrix-matrix multiplication kernel with a fixed percentage of hierarchical tasks on INTEL-V100.

Benchmarks - Matrix-Matrix Multiplication

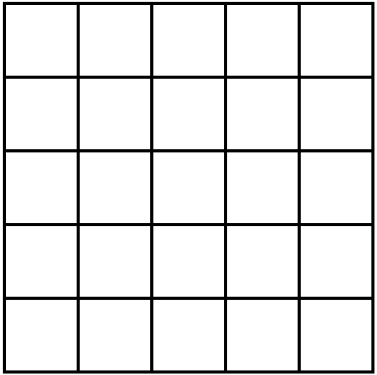


Figure: Tile size of 2880.

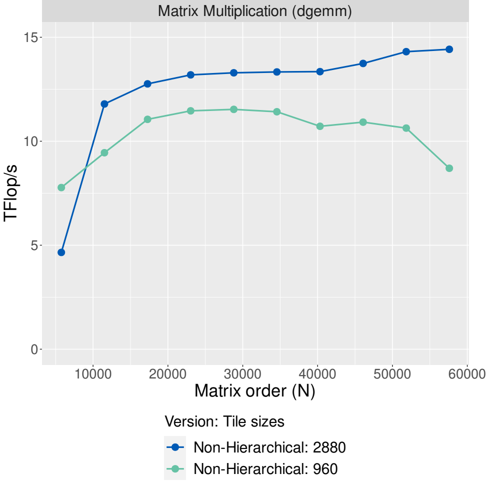


Figure: Matrix-matrix multiplication kernel with a fixed percentage of hierarchical tasks on INTEL-V100.

Benchmarks - Matrix-Matrix Multiplication

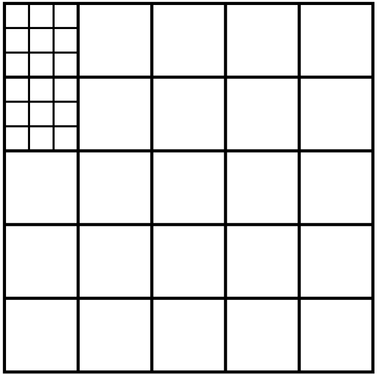


Figure: 10% recursive matrix partitioning.

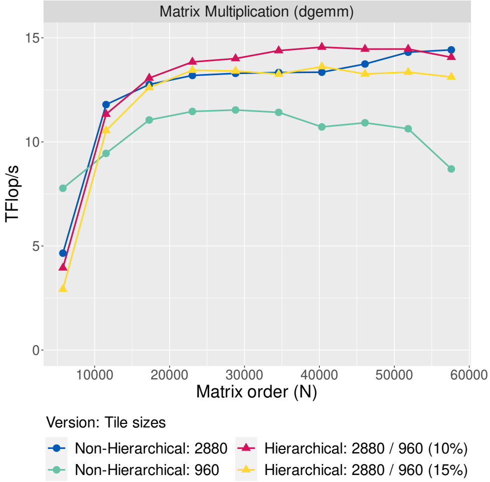


Figure: Matrix-matrix multiplication kernel with a fixed percentage of hierarchical tasks on INTEL-V100.

Benchmarks - LU Factorization without Pivoting

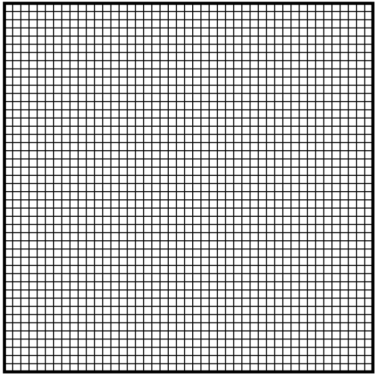


Figure: Tile size of 320.

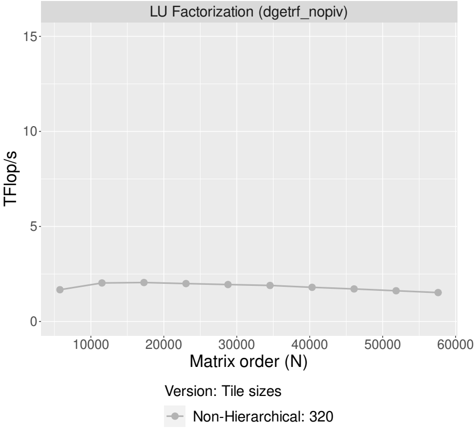


Figure: LU factorization kernel with a diagonal repartition of hierarchical tasks on INTEL-V100.

Benchmarks - LU Factorization without Pivoting

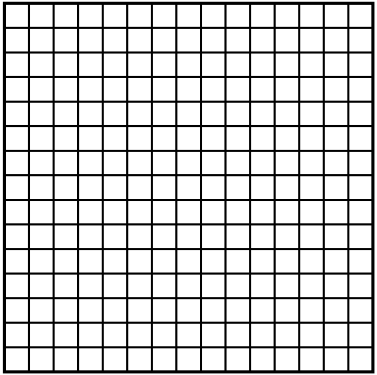


Figure: Tile size of 960.

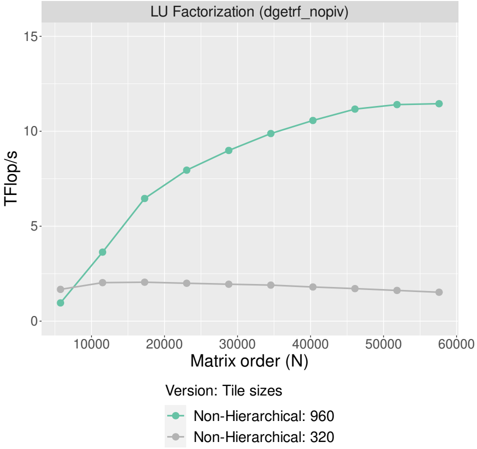


Figure: LU factorization kernel with a diagonal repartition of hierarchical tasks on INTEL-V100.

Benchmarks - LU Factorization without Pivoting

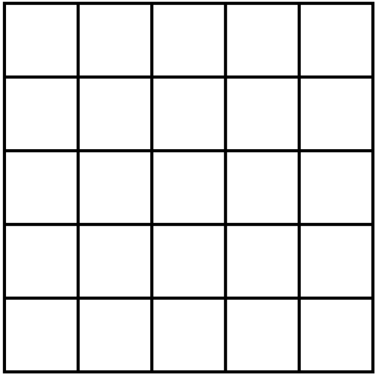


Figure: Tile size of 2880.

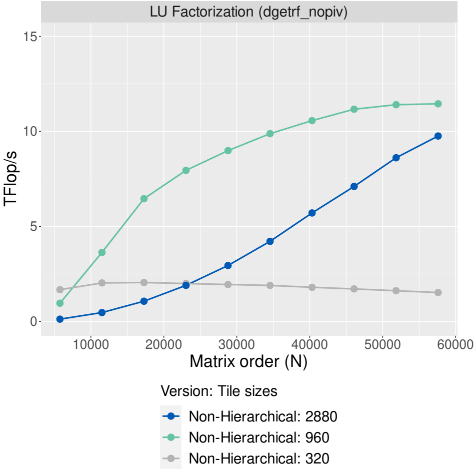


Figure: LU factorization kernel with a diagonal repartition of hierarchical tasks on INTEL-V100.

Benchmarks - LU Factorization without Pivoting

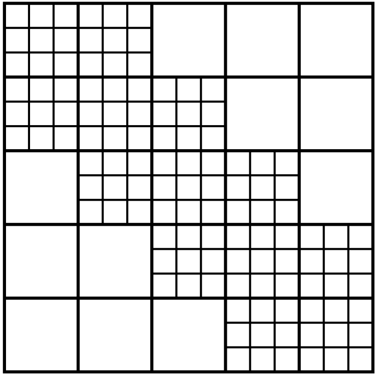


Figure: Diagonal matrix partitioning.

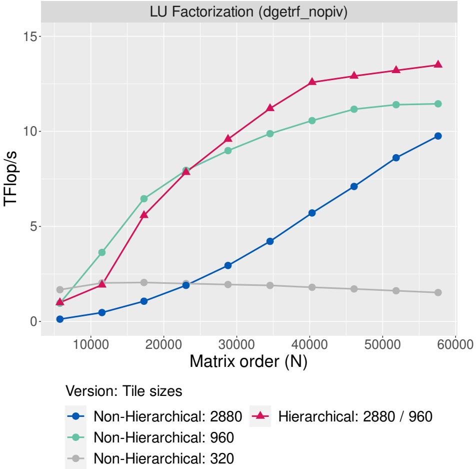


Figure: LU factorization kernel with a diagonal repartition of hierarchical tasks on Intel-V100.

Benchmarks - LU Factorization without Pivoting

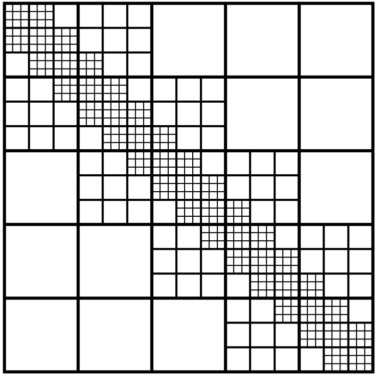


Figure: Diagonal matrix partitioning.

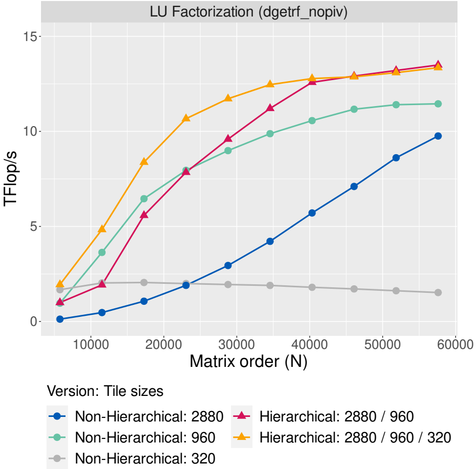


Figure: LU factorization kernel with a diagonal repartition of hierarchical tasks on INTEL-V100.

Benchmarks - Cholesky Factorization

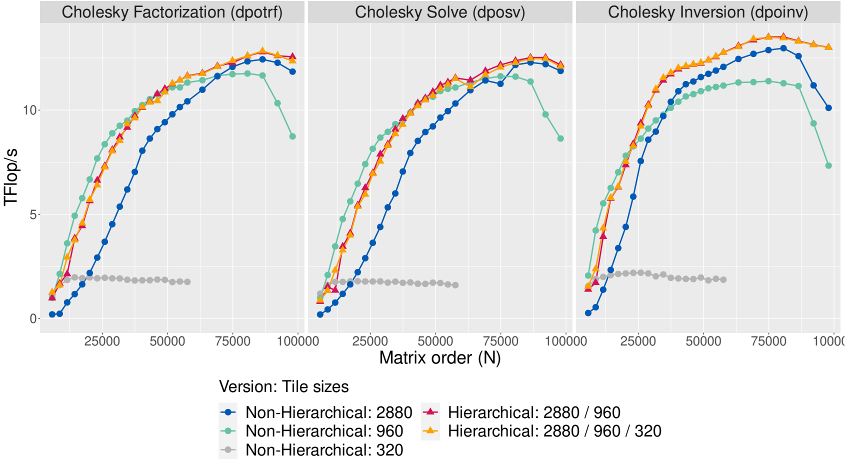


Figure: Cholesky type operations (DPOTRF, DPOSV, DPOINV) kernel with diagonal distribution of the hierarchical tasks on INTEL-V100.

Benchmarks - Best Performance - LU Factorization no Pivoting

- Non-hierarchical version: Various tile sizes and number of stream (1, 2, 4, 8).
- Hierarchical version: Various levels of partitioning.

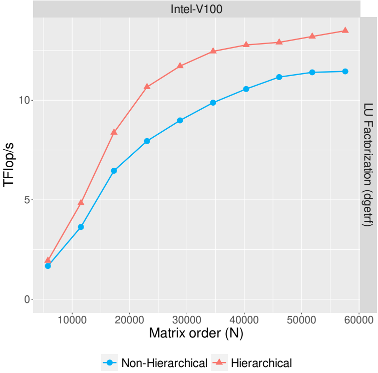


Figure: Aggregation of best results for LU factorization on INTEL-V100.

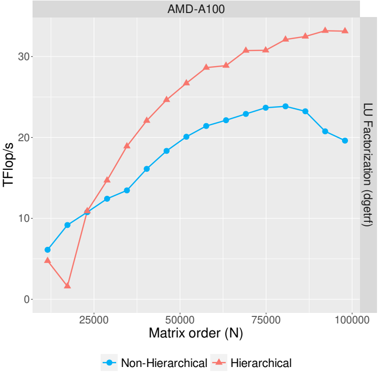


Figure: Aggregation of best results for LU factorization on AMD-A100.

StarPU + MPI

- Thomas Morin extends hierarchical tasks to the distributed context for his intership.

StarPU + MPI

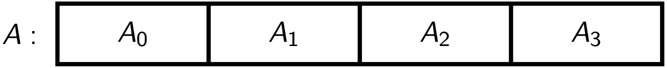
- Thomas Morin extends hierarchical tasks to the distributed context for his intership.
- In StarPU+MPI, data must be distributed among the nodes.

A :	A_0	A_1	A_2	A_3

StarPU + MPI

- Thomas Morin extends hierarchical tasks to the distributed context for his intership.
- In StarPU+MPI, data must be distributed among the nodes.

⇒ How to submit a hierarchical task on A that will be processed on different nodes?



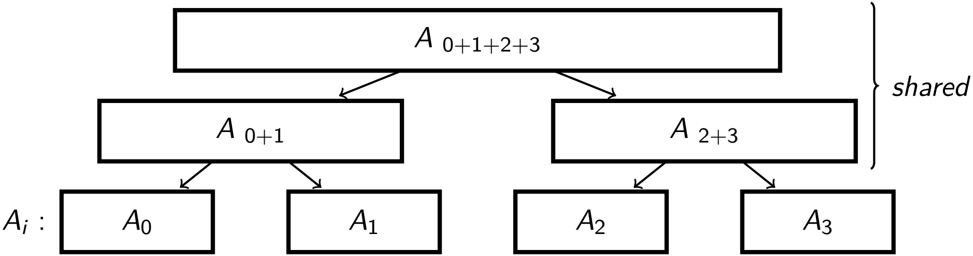
A : A_0 A_1 A_2 A_3

StarPU + MPI

- Thomas Morin extends hierarchical tasks to the distributed context for his intership.
- In StarPU+MPI, data must be distributed among the nodes.

⇒ How to submit a hierarchical task on A that will be processed on different nodes?

- He introduced the notion of *shared data*.

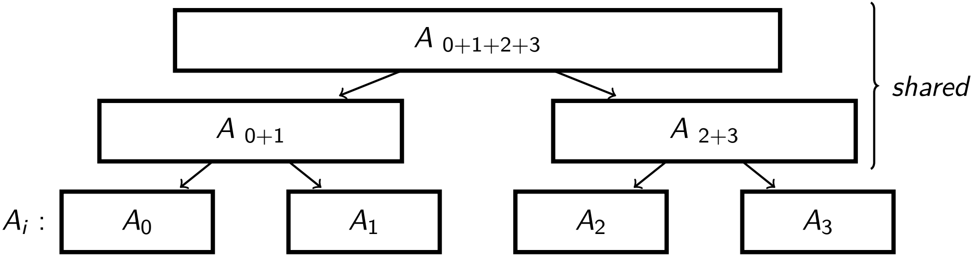


StarPU + MPI

- Thomas Morin extends hierarchical tasks to the distributed context for his intership.
- In StarPU+MPI, data must be distributed among the nodes.

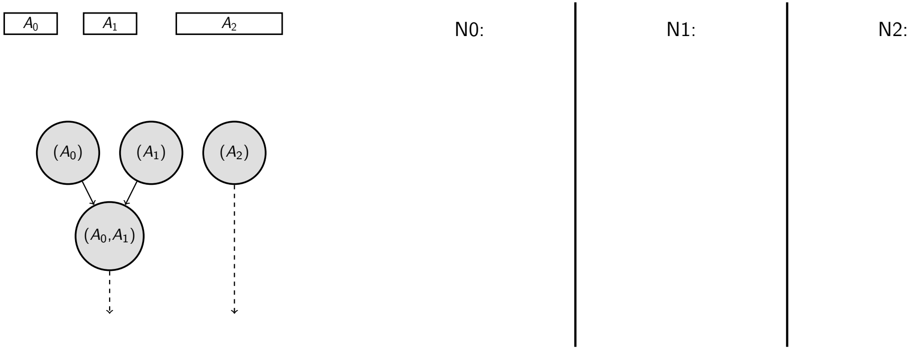
⇒ How to submit a hierarchical task on A that will be processed on different nodes?

- He introduced the notion of *shared data*.
- Whether a data is *shared* is automatically deduced when the A_i are registered.



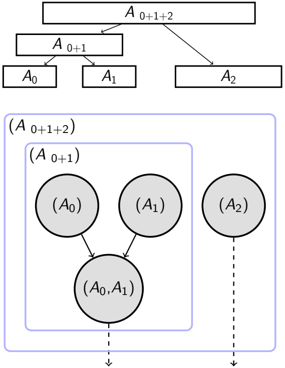
StarPU + MPI - Automatic Pruning

- Using shared data allows us to automatically prune the distributed DAG of tasks irrelevant for certain nodes.



StarPU + MPI - Automatic Pruning

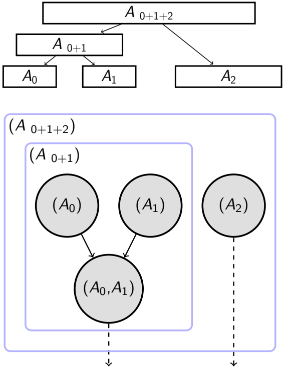
- Using shared data allows us to automatically prune the distributed DAG of tasks irrelevant for certain nodes.

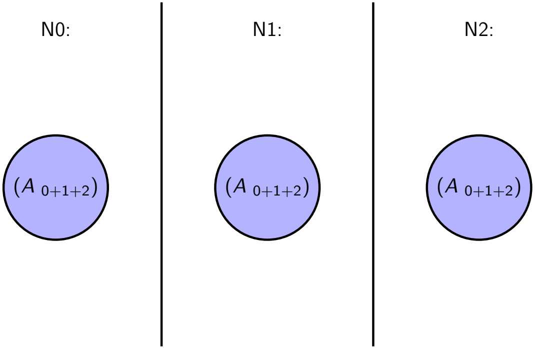


N0: | N1: | N2:

StarPU + MPI - Automatic Pruning

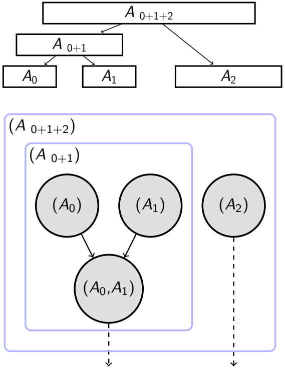
- Using shared data allows us to automatically prune the distributed DAG of tasks irrelevant for certain nodes.

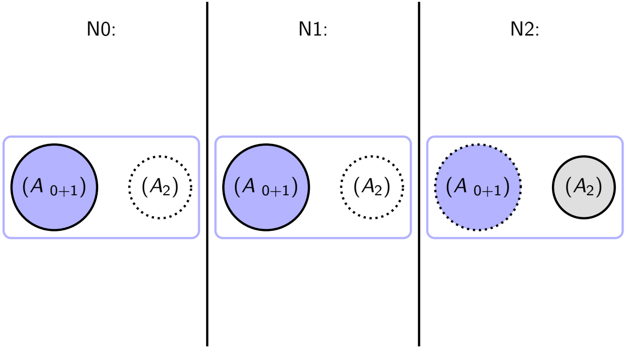




StarPU + MPI - Automatic Pruning

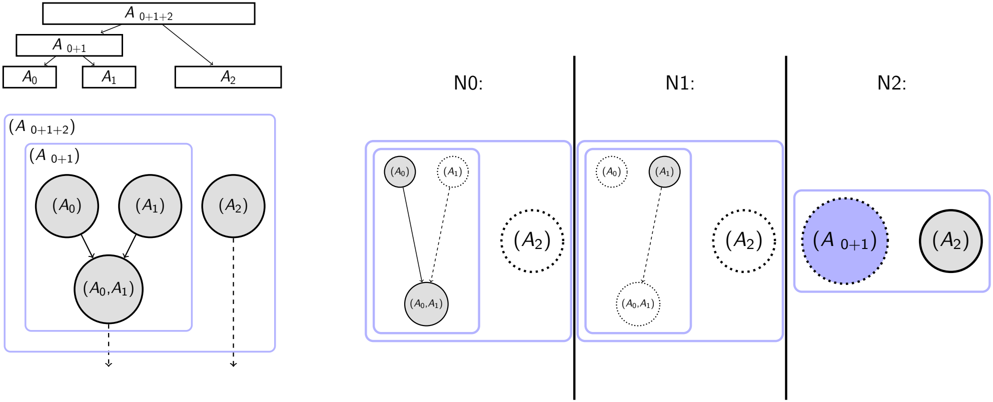
- Using shared data allows us to automatically prune the distributed DAG of tasks irrelevant for certain nodes.





StarPU + MPI - Automatic Pruning

- Using shared data allows us to automatically prune the distributed DAG of tasks irrelevant for certain nodes.



Conclusion

- Hierarchical tasks can insert a subgraph at runtime, resulting in a more dynamic DAG.
- Data management is handled automatically and contributes to the correctness of hierarchical DAGs.
- Hierarchical tasks can be used with StarPU+MPI and allow for automatic pruning of the DAG.

Future Work

- Scheduling questions:
  - When should we insert a subgraph ?
  - Where should we execute it ?
  - Using which implementation ?
- Testing with applications benefitting more from dynamic task graphs.
- In distributed computing, there are investigations to be made around data transfers (delay them, use higher level data, etc).