# Locality-Aware Scheduling of Independent Tasks for Runtime Systems

Maxime GONTHIER

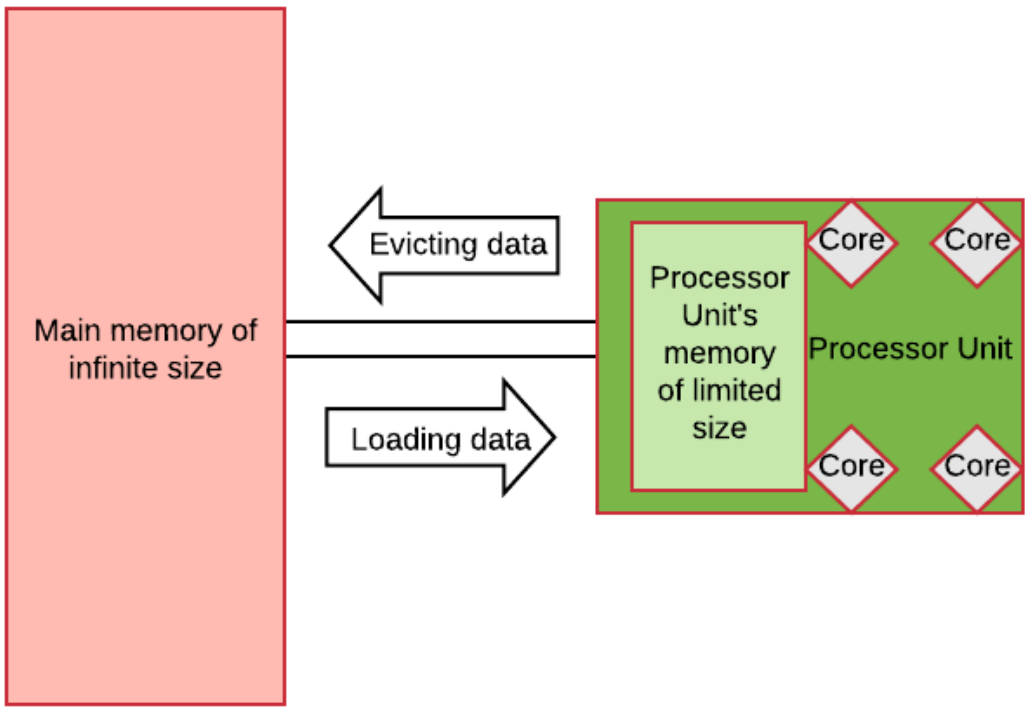
ENS Lyon - ROMA - Université de Bordeaux - LaBRI - STORM - Inria
Supervised by Samuel THIBAULT and Loris MARCHAL

July 2021 - Réunion plénière HPC-SCALABLE-ECOSYSTEM + SOLHARIS

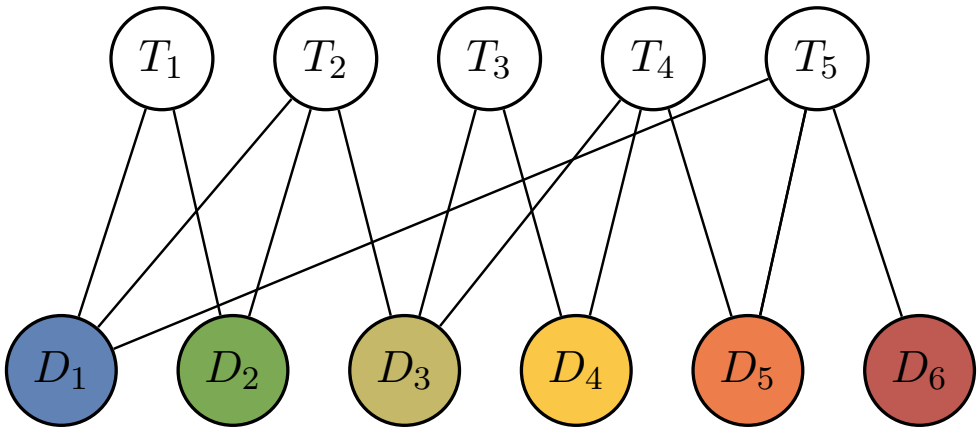
# Introduction

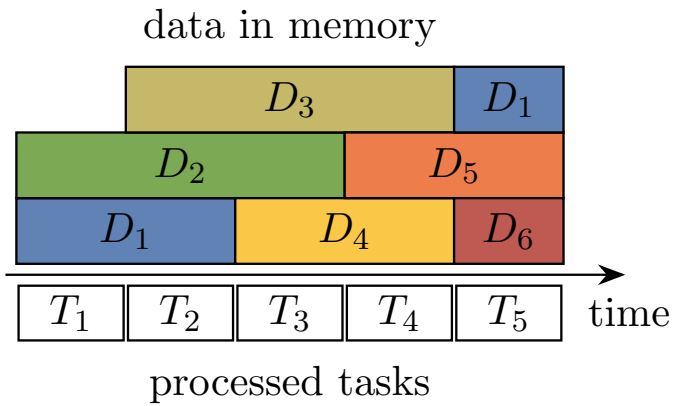
## Interest of our research

- Increasing demand for computing speed by HPC applications
- GPUs calculate quickly but their memory are too small for scientific computing



# Formalization of the problem




- Only input data. Outputs are ignored
- Which task order? $\Rightarrow \sigma(t)$
- When to load/evict data? evict $\Rightarrow \mathcal{V}(t)$ before processing of $T_{\sigma(t)}$

# Algorithms tested

## 2 algorithms from the state of the art

- Maximum Spanning Tree (MST)
- Reverse-Cuthill-McKee (RCM)

## 2 schedulers from StarPU

- Eager (our baseline)
- Deque Model Data Aware Ready (DMDAR)

## 1 new algorithm

- Hierarchical Fair Packing (HFP)

# DMDAR

## Intuition

DMDAR (or tmdp) is an efficient scheduler built in StarPU

## Usage

- Schedules tasks where their completion times is expected to be minimal (dynamic version of HEFT)
- Allocates tasks as soon as they are available
- Uses a ready strategy to favor tasks whose data has already been loaded into memory
- Dynamic scheduling → use LRU eviction policy

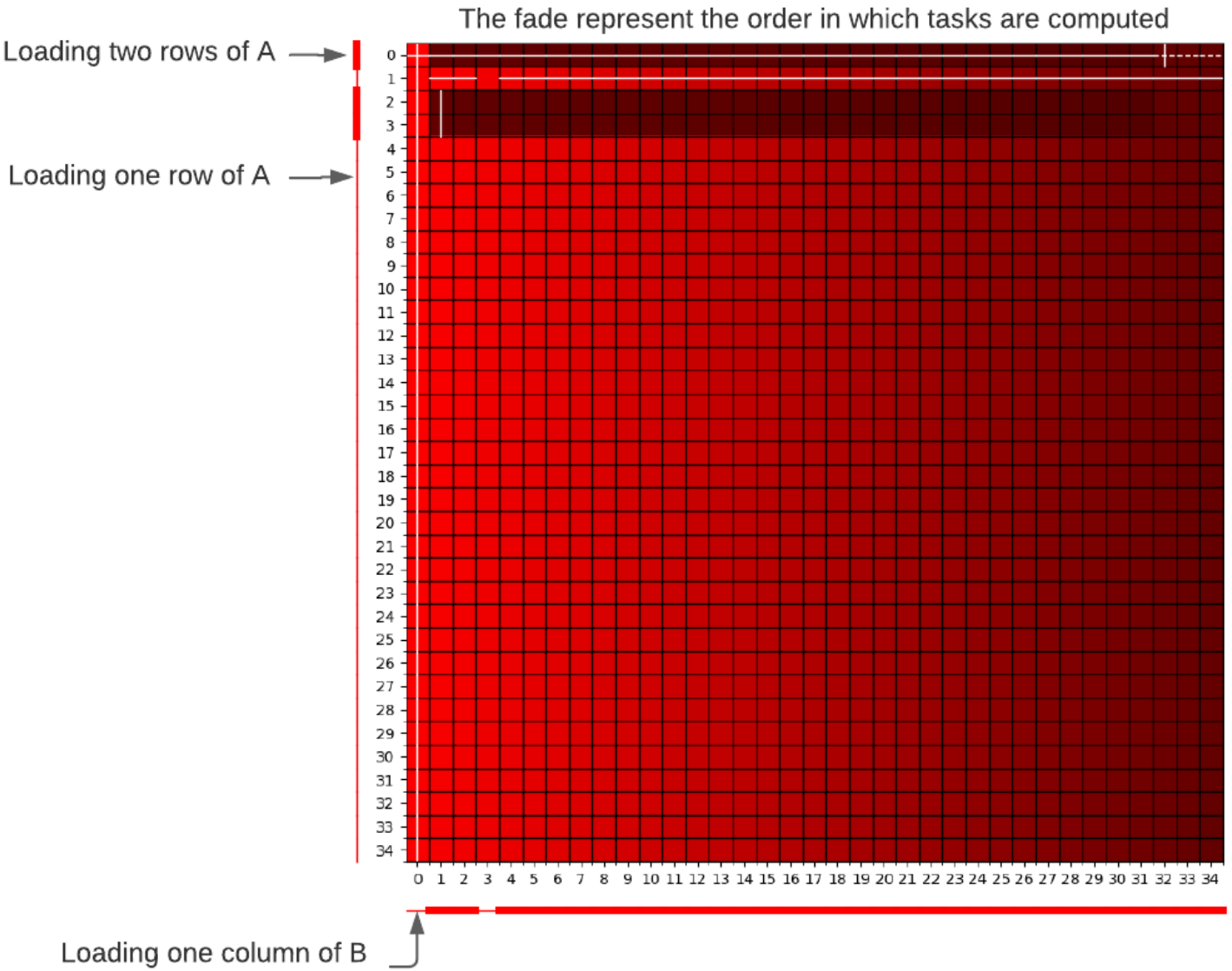



Figure: Visualisation of DMDAR's ordering.

# Hierarchical Fair Packing

## Intuition

Group tasks with common data into packages of maximum size $M = GPU_{RAM}$. Minimizing the number of packages will maximize data reuse.

## Complexity

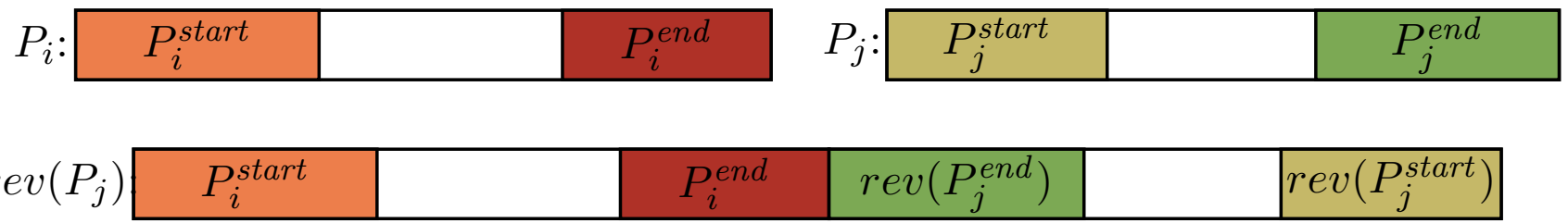
For a set of tasks $\mathbb{T}$ sharing data in $\mathbb{D}$. Partitioning tasks into at most $L$ packages $P_1, \ldots, P_L$ such that $|D(P_i)| \leq M$ for each package $P_i$ is an NP-complete problem.
Reduction from 3-partition

# Description

0. Initially each task is an elementary package
1. Compute the number of common data $|\mathcal{D}(P_i) \cap \mathcal{D}(P_j)|$ on all pairs of packages
2. Consider packages $P_i$ with the fewest tasks
3. Merge couples $(P_i, P_j)$ which reach $\max(|\mathcal{D}(P_i) \cap \mathcal{D}(P_j)|)$ such that $|\mathcal{D}(P_i) \cup \mathcal{D}(P_j)| \leq M$
4. Repeat from step 1 until there are no more possible merge
5. Consider that $M = \infty$ and repeat from step 1 as long as there is more than 1 package

# Package flipping

Improve data locality inside a package without changing its content




The pair of sub-packages $(P_i^{end}, P_j^{end})$ is the one with the most shared input data.
$\rightarrow P_j$ is reversed before merging packages.

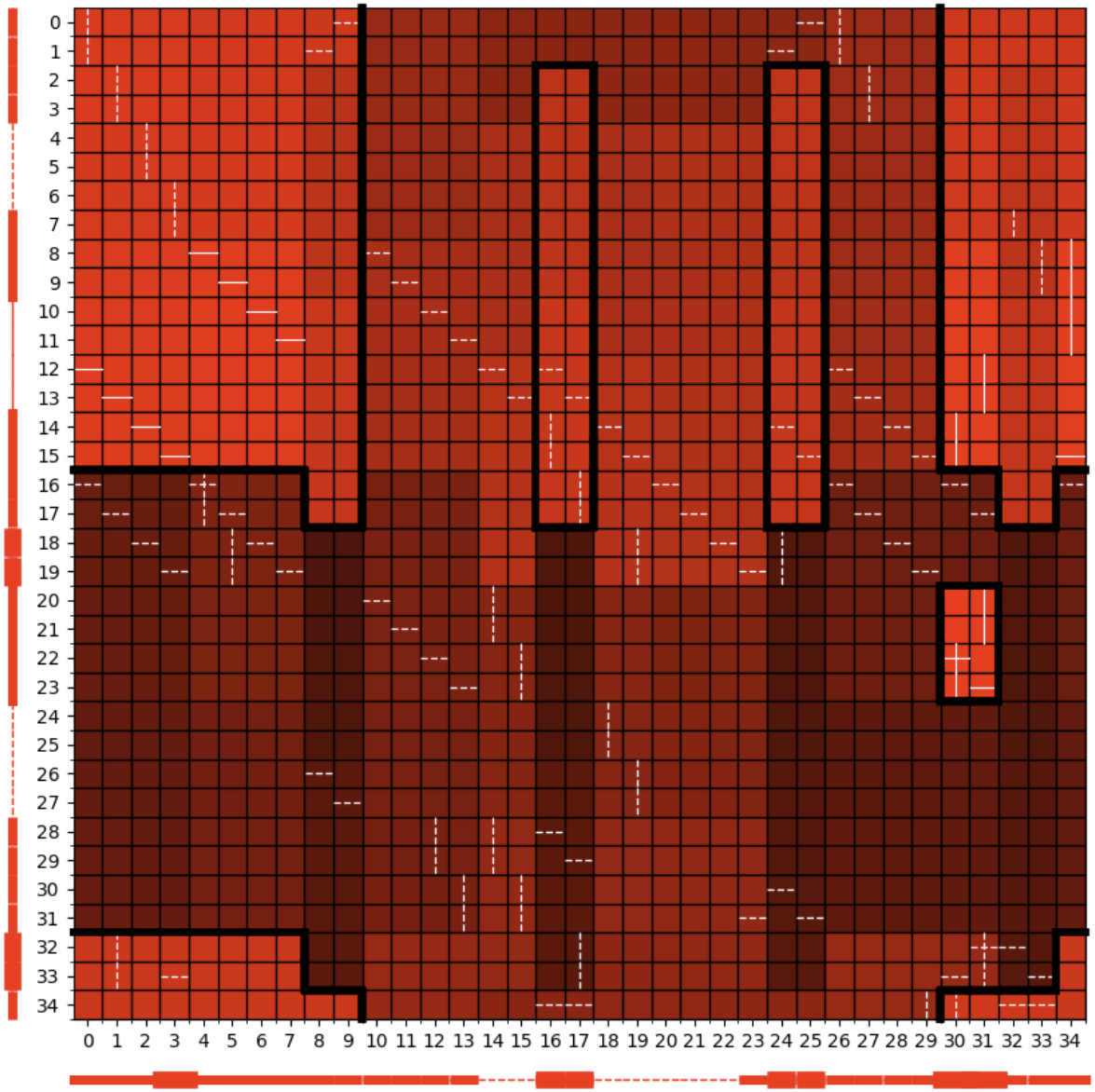



Figure: Visualisation of HFP's ordering.

# Belady's eviction policy

We denote by *MIN* the eviction policy that always evicts a data whose next use in $\sigma$ is the latest. For any eviction policy $\mathcal{V}$,

$$\#Loads(\sigma, MIN) \leq \#Loads(\sigma, \mathcal{V}).$$

Here the full set of tasks is available, so we can use Belady

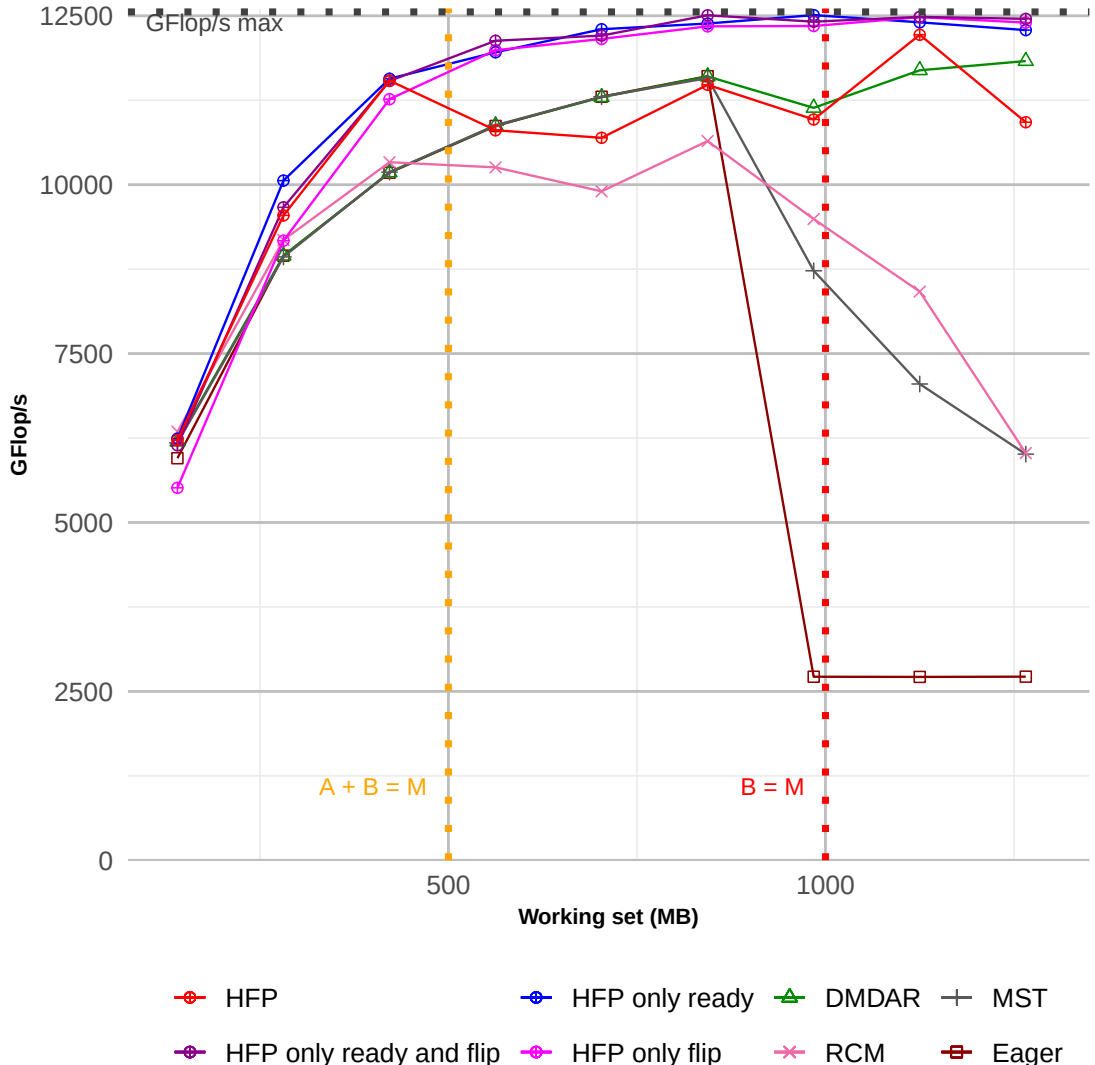
## Experimental settings

- Experiments on a tesla V100 GPU as well as simulations on Simgrid using the same performance model
- PCI bandwidth limited to $6000MB/s$
- GPU memory limited to $500MB$ in order to better distinguish the performance of different strategies even on small datasets

## Applications

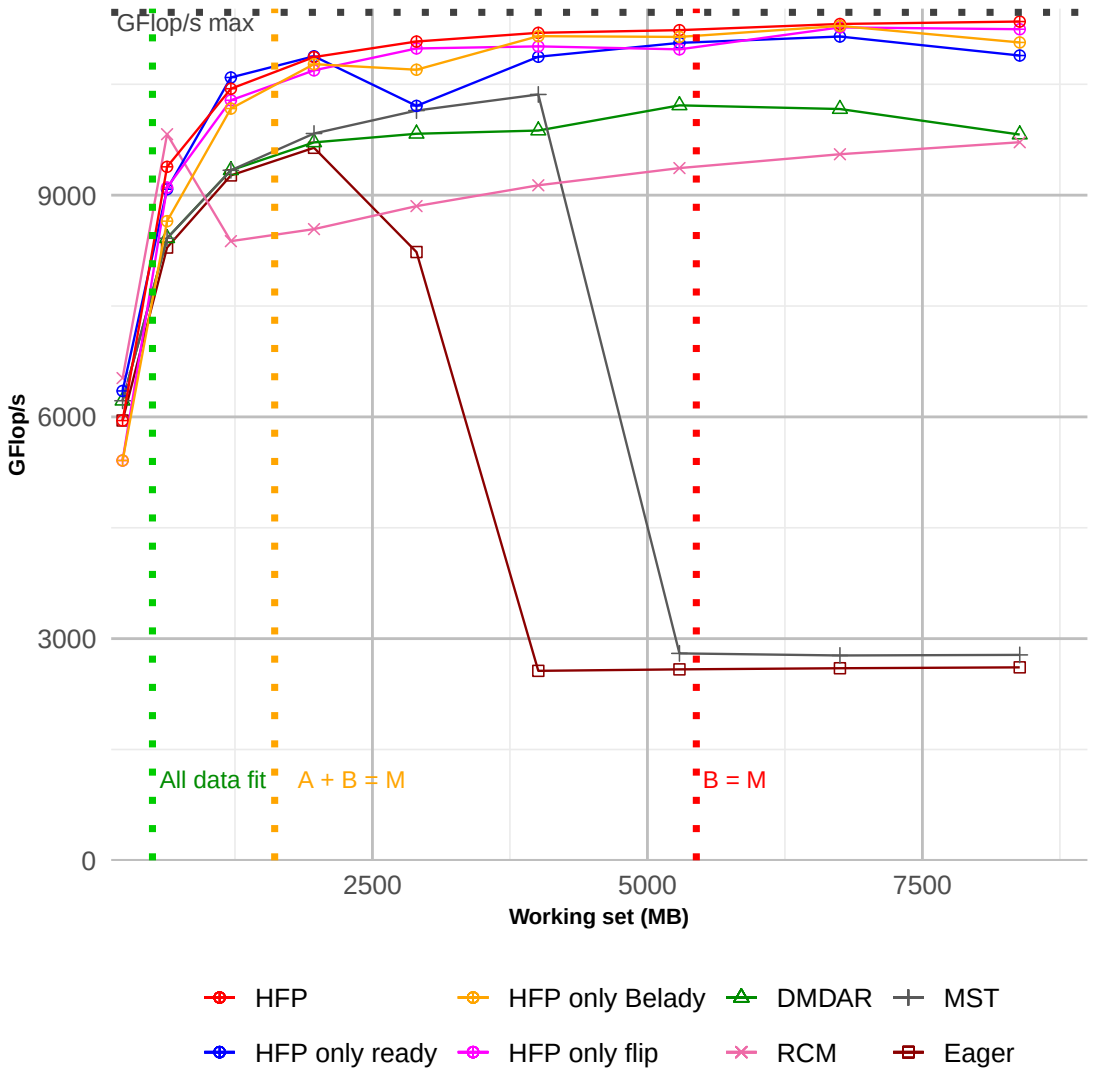
- **2D matrix multiplication**: Each task is the multiplication of one block-row of $A$ per one block-column of $B$
- **3D Matrix multiplication**: Each task requires one tile of $A$, $B$ and $C$

# GFlop/s on a 2D matrix multiplication using a Tesla V100 GPU



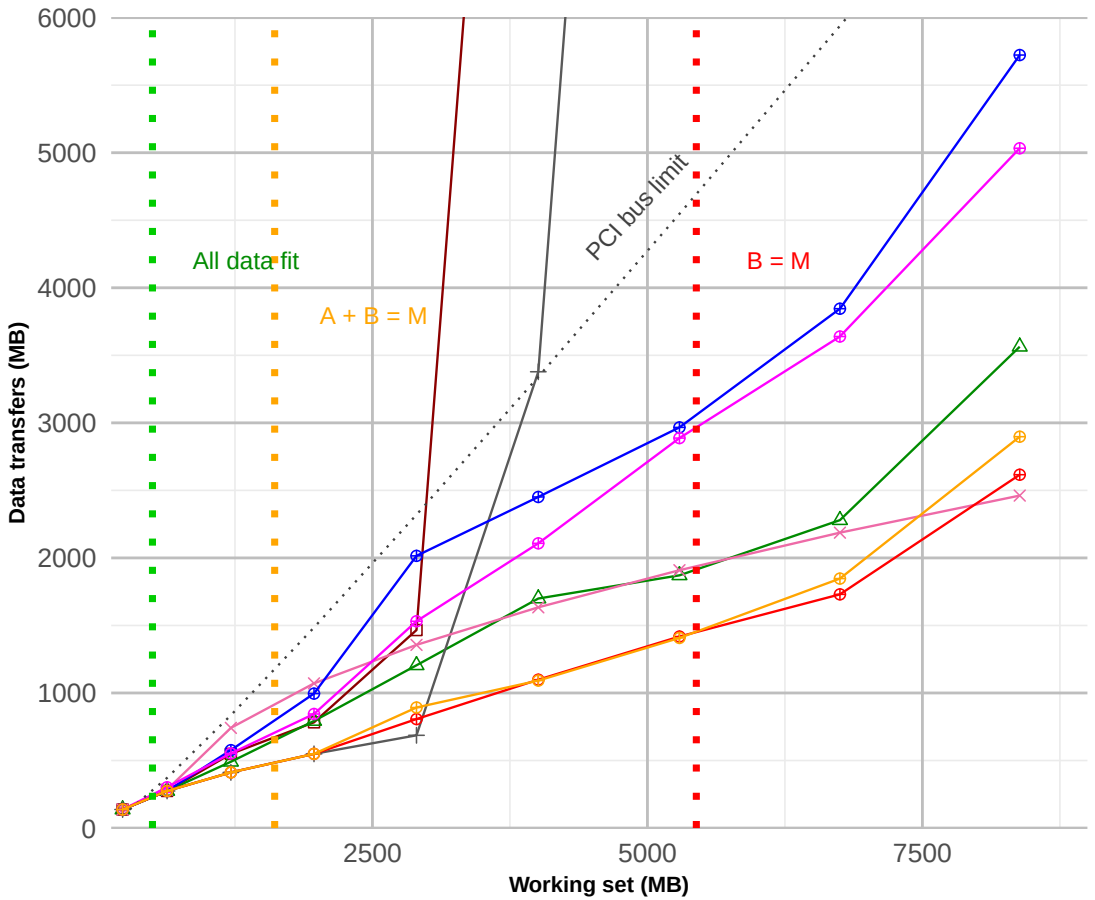

- Pathological matrix size corresponding to a little less than the size of the GPU memory
- HFP is 8% better than Dmdar
- The cost of Belady overcome it benefit

# GFlop/s on a 3D matrix multiplication on Simgrid




- DMDAR avoid the pathological case
- HFP is 11% better than Dmdar. Keeps gathering tasks forming a square

# Data transfers on a 3D matrix multiplication




(b) Amount of data transfers.

- HFP get better performance with more data transfers → better ditribution of transfers over time

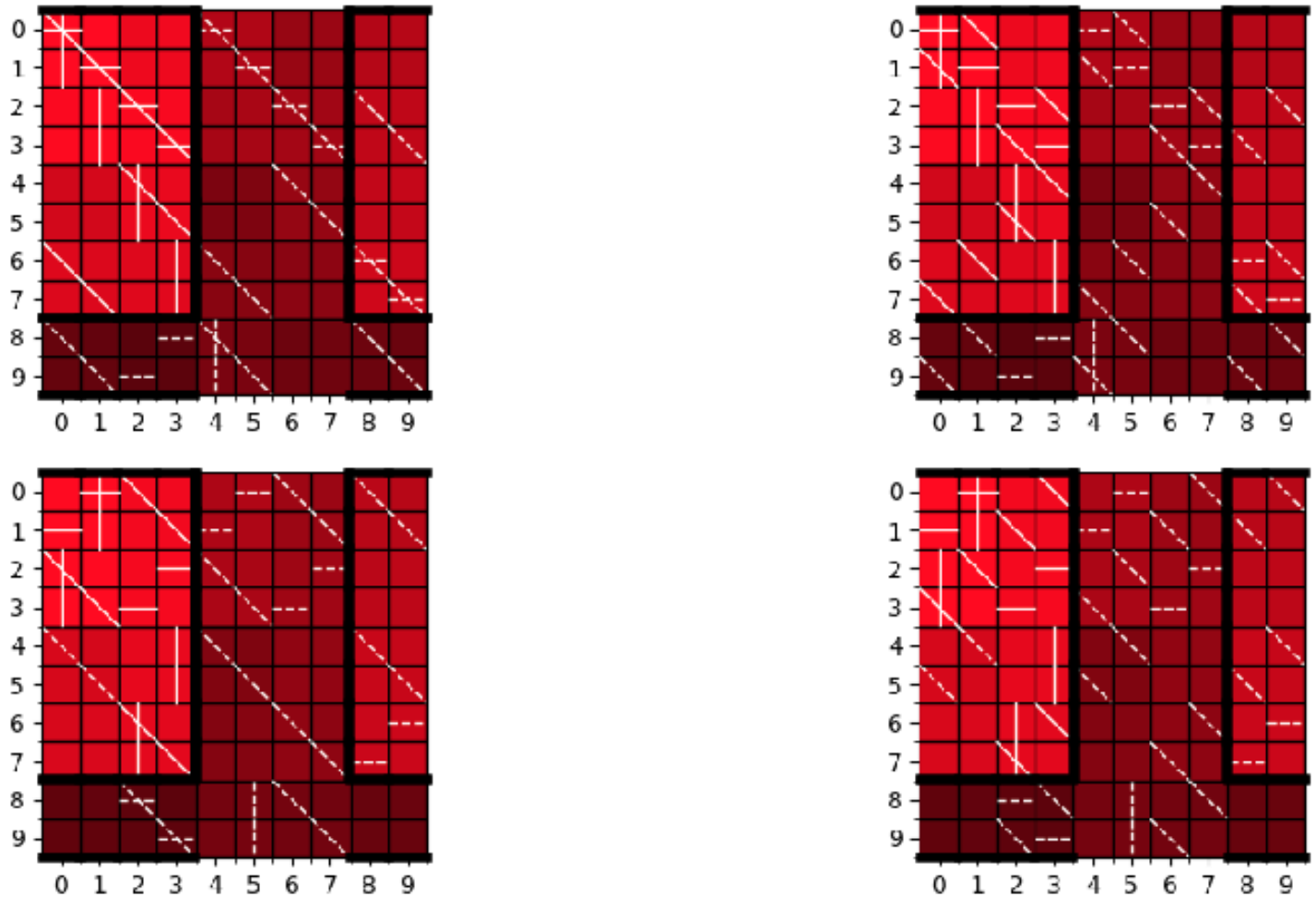



Figure: 3D matrix multiplication of depth 4.

# Parallel computing with HFP

## Two goals

- Partition task among multiple GPUs
- Order tasks for each partition

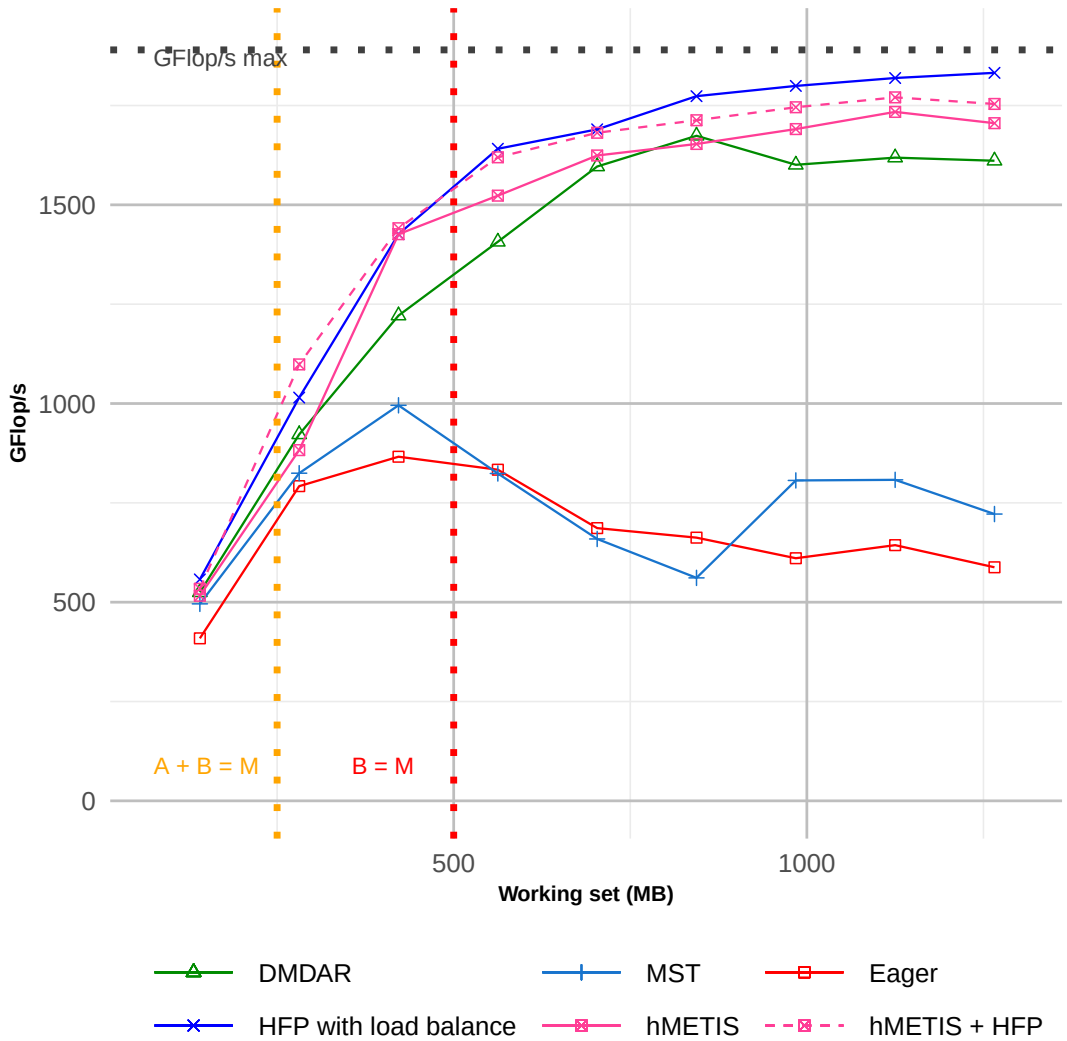
## Partitionning with HFP

1. Apply HFP to form as much packages as there are GPUs
2. Balance expected computation time between each packages
3. Assign one package to each GPU

## Partitionning with hMETIS

1. Use hMETIS to build as much partition as there are GPUs
2. Apply HFP on each partition
3. Assign one partition to each GPU

# GFlop/s on a 2D matrix multiplication on Simgrid with 3 GPUs




- HFP with load balance as well as HFP with hMETIS are better than DMDAR

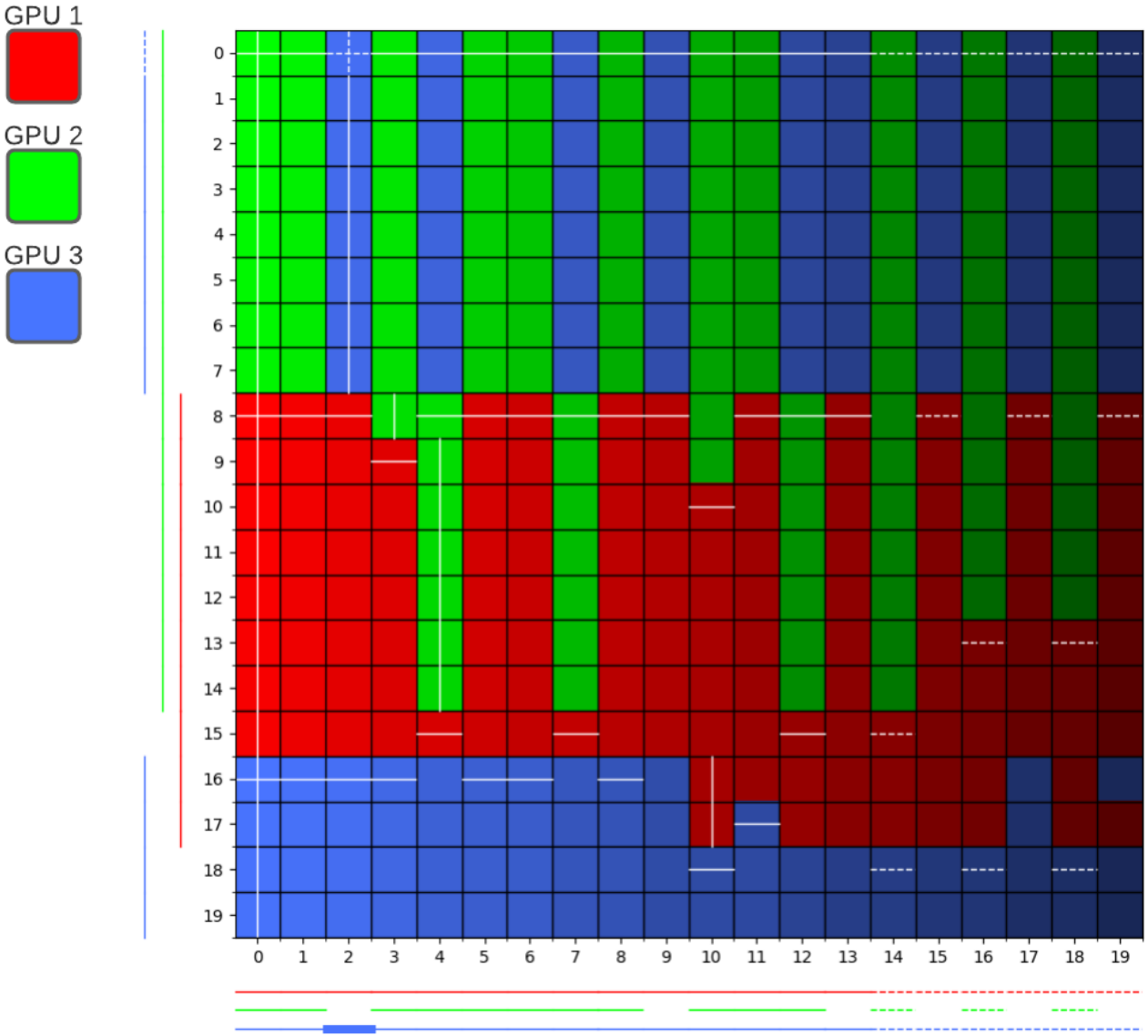



Figure: Visualisation of DMDAR's partitioning and ordering.

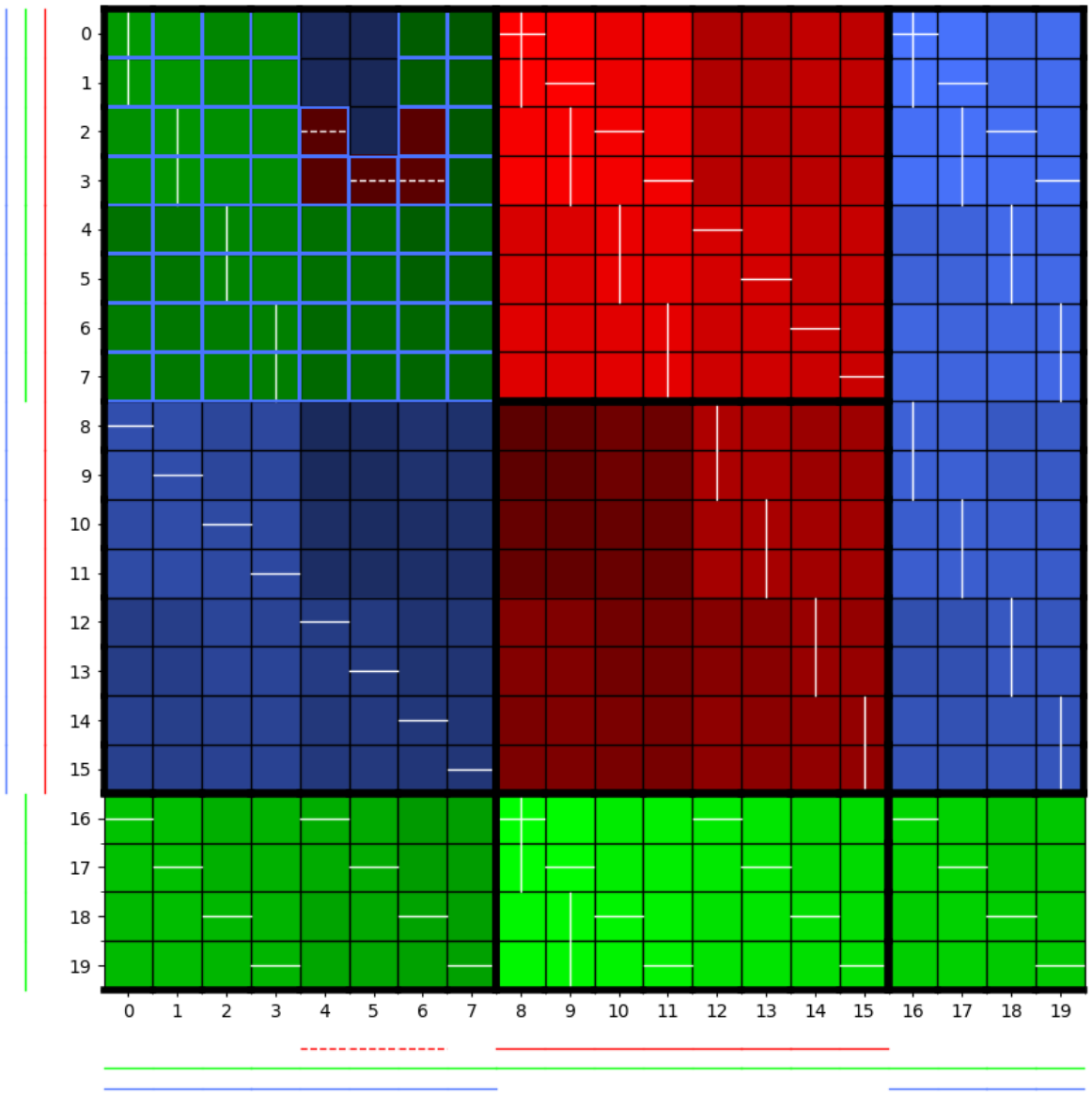



Figure: Visualisation of HFP's partitioning and ordering.

Goal: ordering tasks sharing data in order to avoid moving data as much as possible.

## Our contributions

- New heuristic based on gathering tasks with similar data into packages. Allows on average a 8.3% (resp. 11%) improvement for a 2D (resp. 3D) matrix multiplication
- Optimal eviction scheme, based on Belady's rule
- HFP is good in distributing data transfer over time to increase transfer/computation overlap
- HFP is easily adapted to parallel computing

## Future works

- Adapting Belady's rule to the Ready dynamic task reordering to integrate it in native executions
- Adapt our algorithms to scenarios with a "hot start"
- Use applications with inter-task dependencies

# Thank you

# Maximum Spanning Tree (MST)

## Intuition

From "*Locality-aware task management for unstructured parallelism: A quantitative limit study*" Yoo, Hughes, Kim, Chen, Kozyrakis.
Following Prim's algorithm on a graph where nodes are tasks and edges are data share.
Select the incident edge with largest weight → increase the data reuse between the previous tasks and the next one to process.

## Usage

Order the vertices according to their order of inclusion in the spanning tree

# Reverse-Cuthill-McKee (RCM)

## Intuition

- Permutes a sparse matrix into a minimal band matrix
- Vertices sharing an edge are at most $k$ edges away
- $T_i$ processed at time $t$ has all its "neighbours" processed in the interval $[t-k; t+k]$.

## Usage

Reversing the obtained order is known to improve the performance of the Cuthill–McKee algorithm.

## Theorem

$\#Loads(\sigma, MIN) = \#Loads(\bar{\sigma}, MIN)$
But performance (in Gflop/s) of RCM is better in practice.