Memory-Aware Scheduling of Tasks Sharing Data

Applied to limited distributed memory (GPUs) and limited shared memory (CPU)

Maxime GONTHIER

Supervised by Loris MARCHAL and Samuel THIBAULT

maxime.gonthier@ens-lyon.fr

LIP - ROMA - LaBRI - STORM - Inria
ENS de Lyon - Université de Bordeaux

March 31, 2023

ENS DE LYON université de BORDEAUX

Memory-Aware Scheduling of Tasks Sharing Data

Applied to limited distributed memory (GPUs) and limited shared memory (CPU)

Yet another Cholesky talk !

Maxime GONTHIER

Supervised by Loris MARCHAL and Samuel THIBAULT

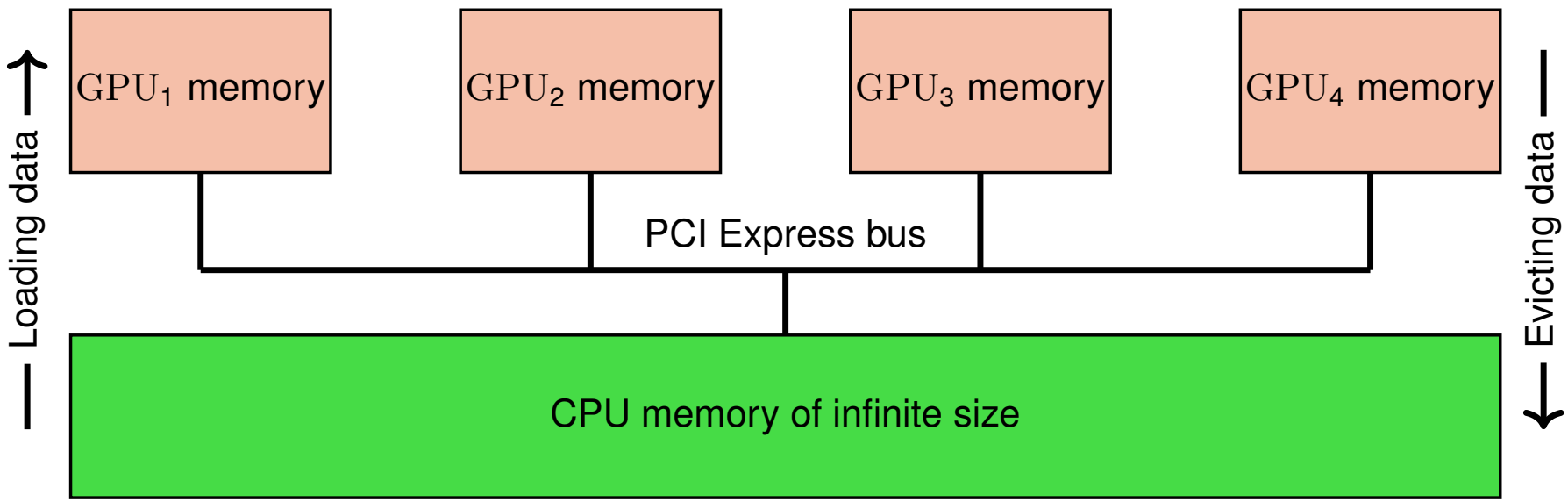
maxime.gonthier@ens-lyon.fr

LIP - ROMA - LaBRI - STORM - Inria
ENS de Lyon - Université de Bordeaux

March 31, 2023

ENS DE LYON université de BORDEAUX

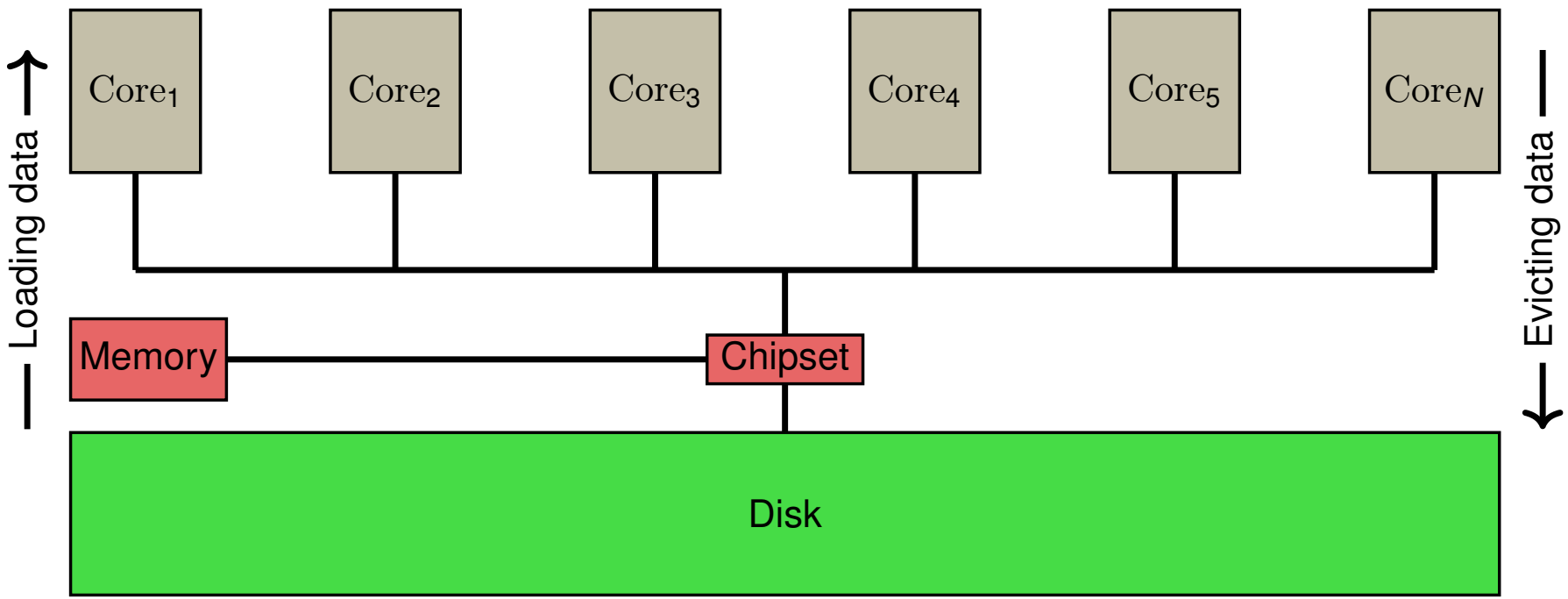
Why focus on memory-aware scheduling? Case 1: limited distributed memory (GPUs)



GPUs are fast but have a limited memory

They share the same PCI bus with limited bandwidth

Why focus on memory-aware scheduling? Case 2: limited shared memory (CPU)



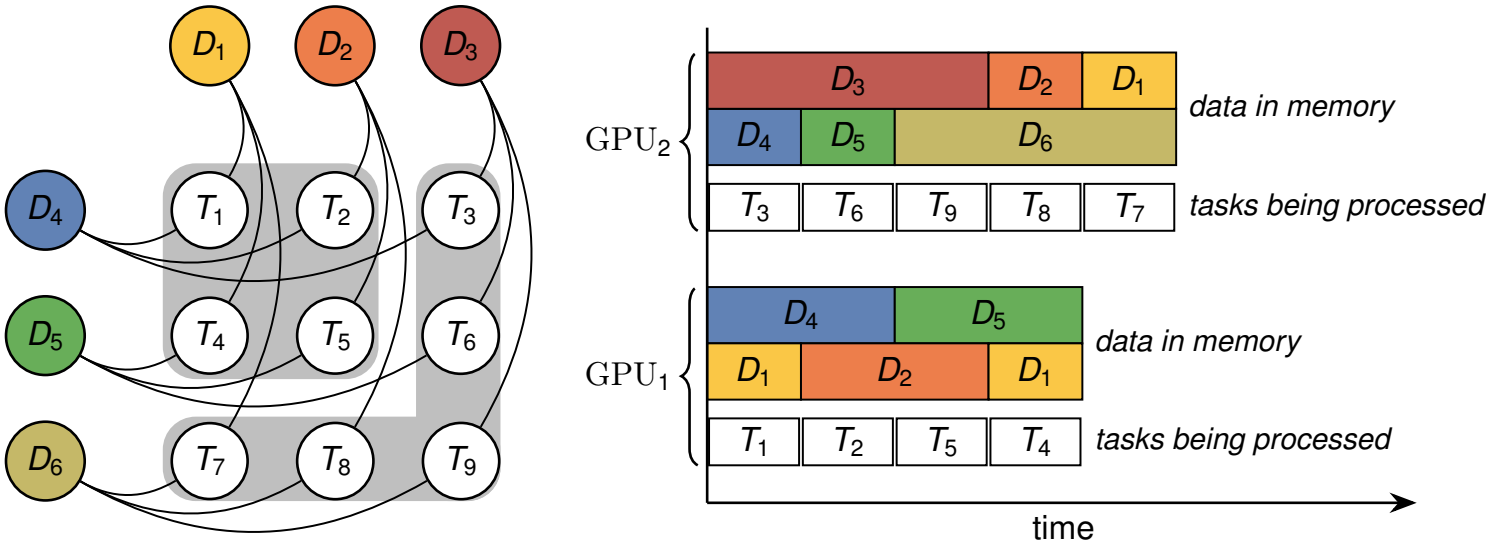
Communicating with the disk is much slower than with a PCI bus

Problem modeling

Steps before processing each task on a PU

1. PU_k wants to process task T
2. Some data are evicted from the memory of PU_k
3. Data required by T that are not yet in memory are loaded in the memory of PU_k
4. Task T is processed on PU_k

Example with 2D grid dependencies



GOALS:

- **Balancing** tasks among GPUs → Reduce total execution time
- **Ordering** tasks inside each GPU → Reduce data transfers
- Having a **generic** scheduler → Efficient with any application or memory limitation

Dynamic scheduler of StarPU: DMDAS

Strategy

Schedule tasks so their completion time is minimal based on computation + communication times and sorts tasks by priority order

+ Ready reordering heuristic on PU_k

From the list L of tasks allocated on PU_k, it will search the task $T \in L$ requiring the fewest data transfers

input : List L of tasks allocated on PU_k
while $L \neq \emptyset$ **do**
    Search first $T \in L$ requiring the fewest data transfers
    Wait for all data in $\mathcal{D}(T)$ to be in PU_k memory
    Start processing T
end

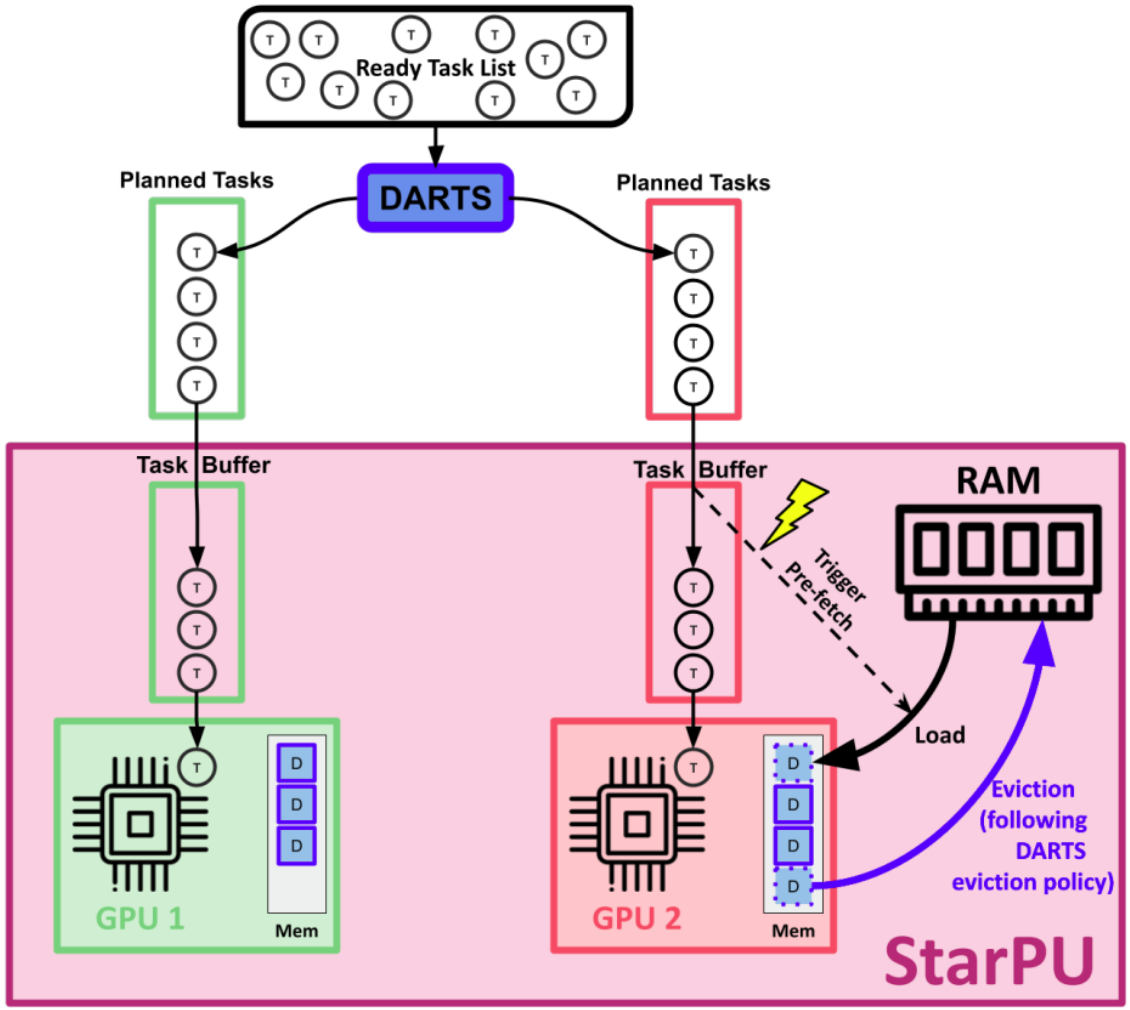
ENS DE LYON université de BORDEAUX

Work stealing from StarPU: LWS

Strategy

- One queue per worker
- Schedules tasks on the worker which released it by default (to re-use data)
- Steals tasks from neighbor worker's end of queue to load balance while maintaining locality
- Sort tasks by priority

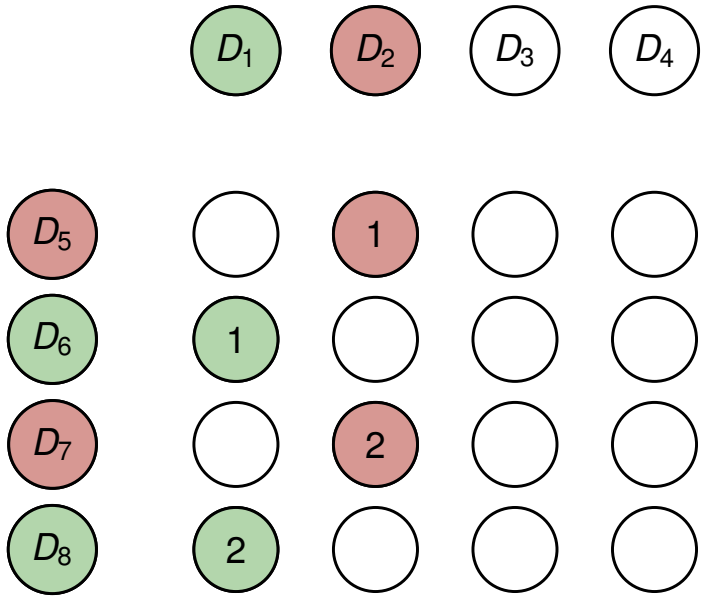
Data-Aware Reactive Task Scheduling: DARTS



If $taskBuffer_k$ is empty -> pop a task from $plannedTasks_k$
If $plannedTasks_k$ is empty -> call DARTS to fill $plannedTasks_k$

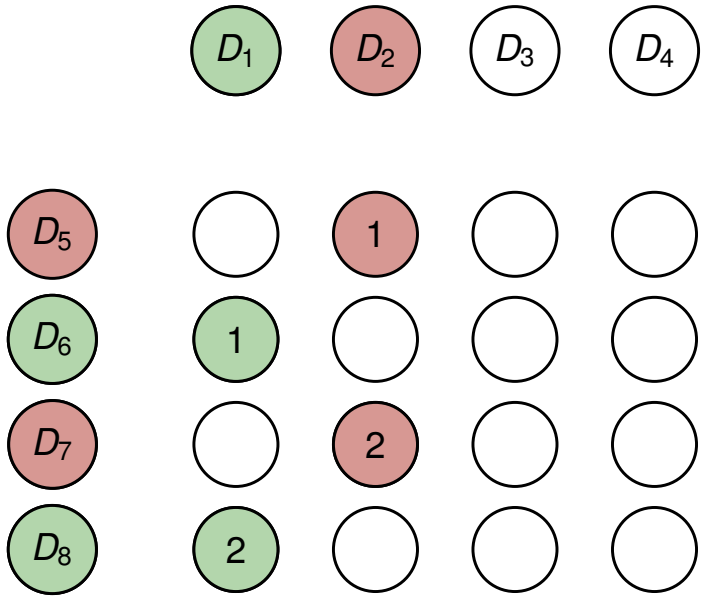
Intuition

- **Consider data movement before task allocation**
- **Perform as many tasks as possible with the data at hand**



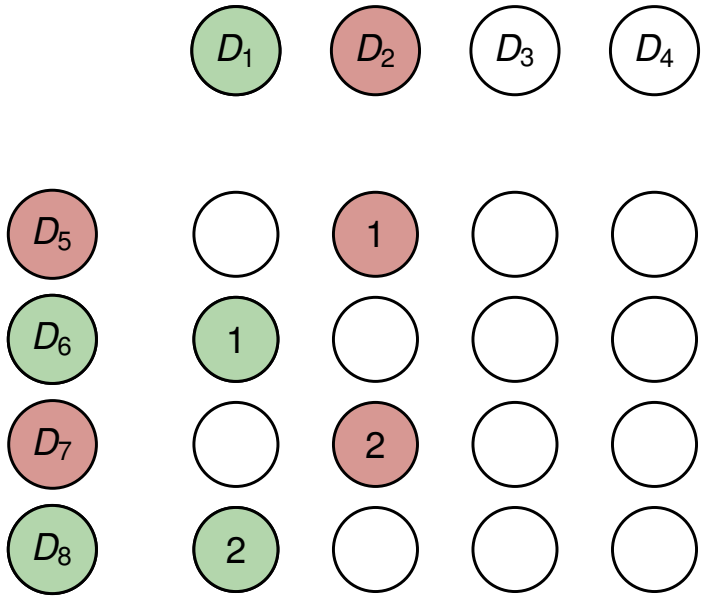
Intuition

- **Consider data movement before task allocation**
- **Perform as many tasks as possible with the data at hand**



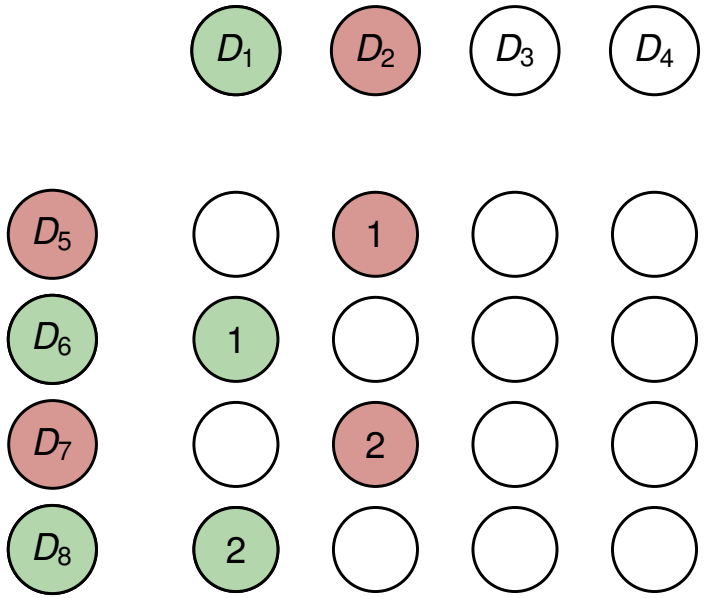
Intuition

- **Consider data movement before task allocation**
- **Perform as many tasks as possible with the data at hand**



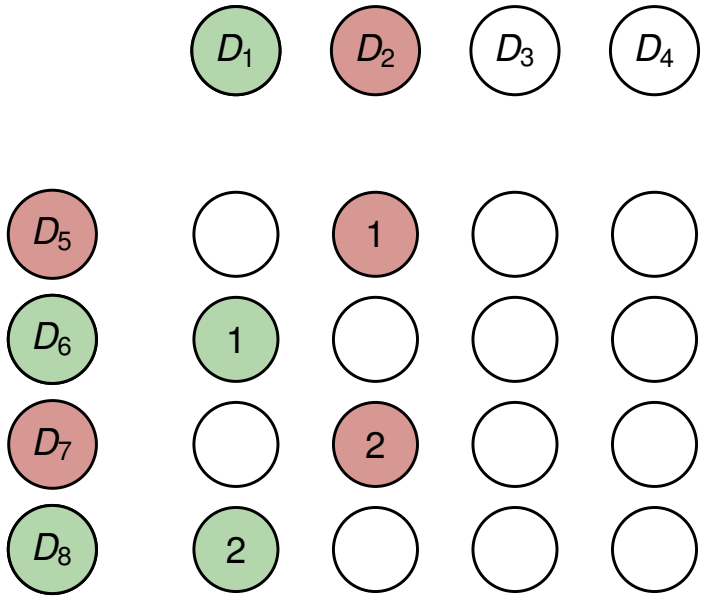
Intuition

- **Consider data movement before task allocation**
- **Perform as many tasks as possible with the data at hand**



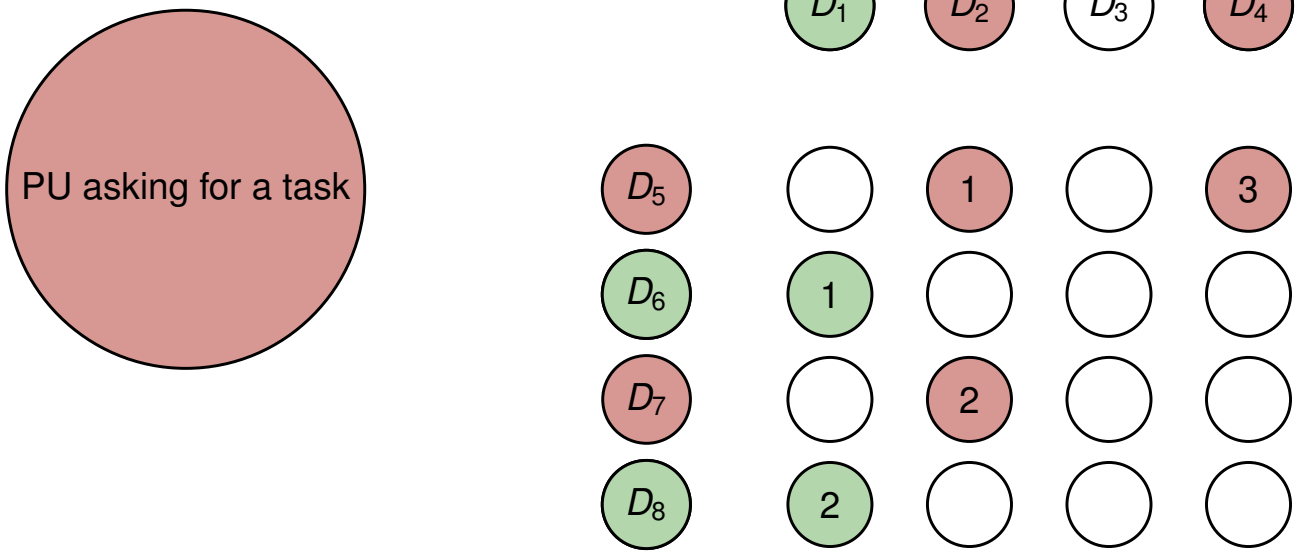
Intuition

- **Consider data movement before task allocation**
- **Perform as many tasks as possible with the data at hand**



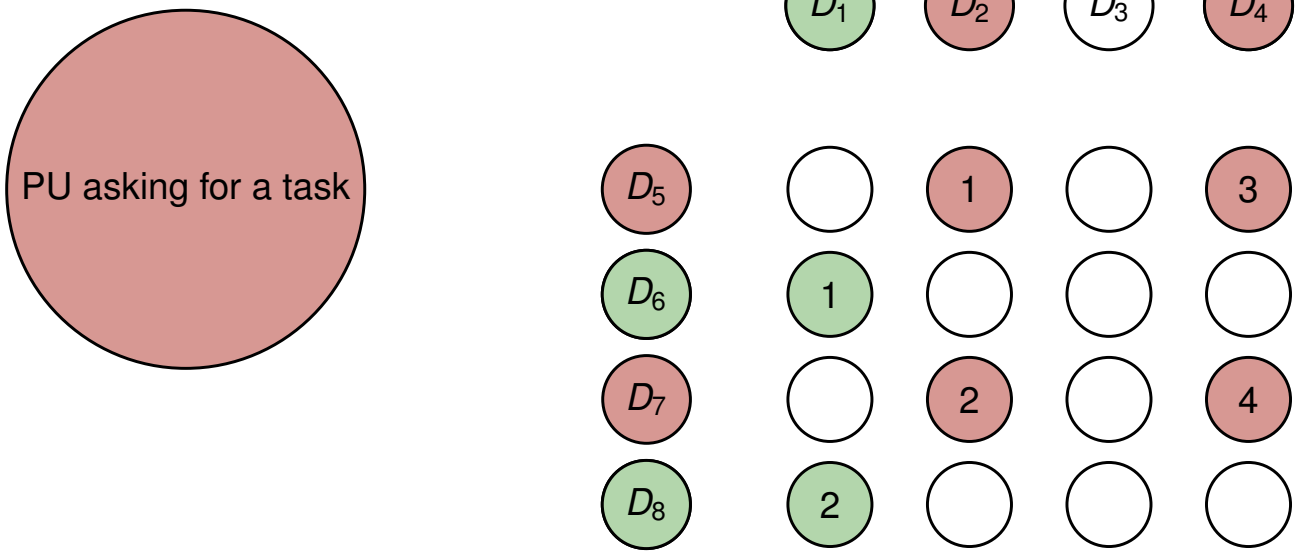
Intuition

- **Consider data movement before task allocation**
- **Perform as many tasks as possible with the data at hand**



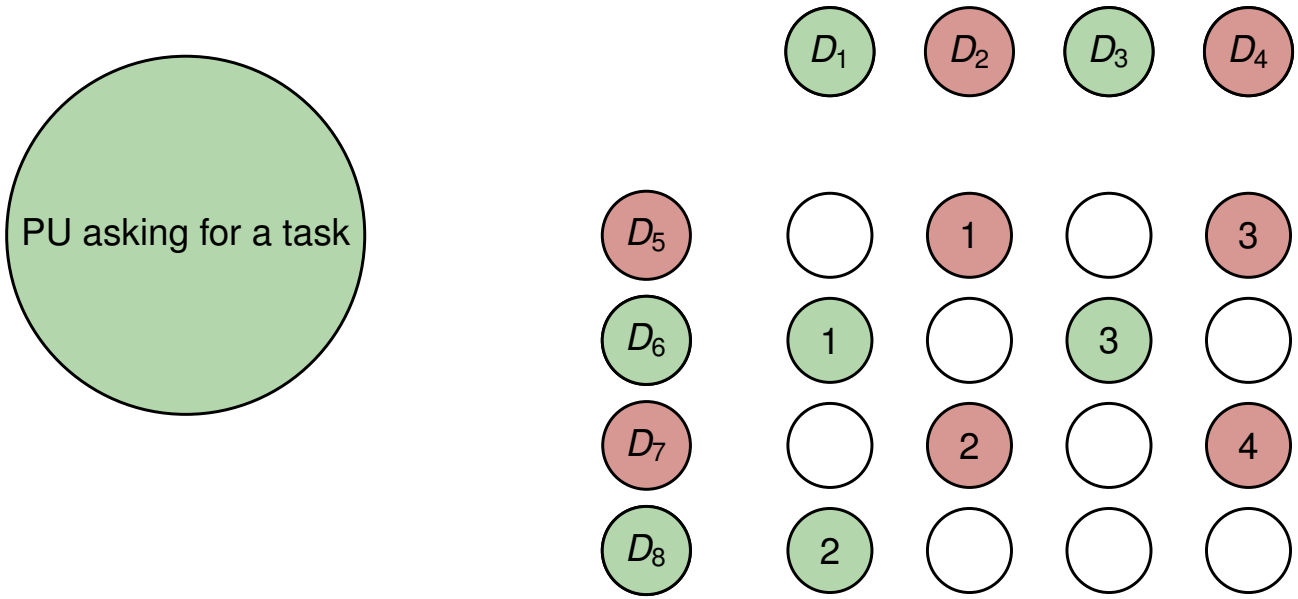
Intuition

- **Consider data movement before task allocation**
- **Perform as many tasks as possible with the data at hand**



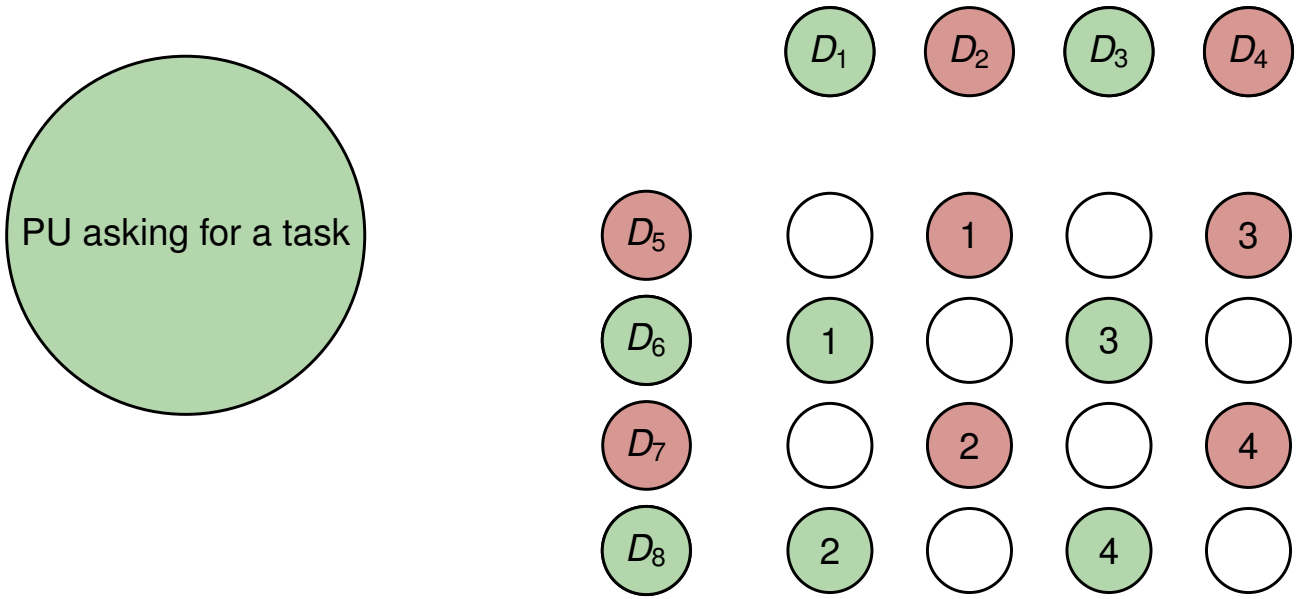
Intuition

- **Consider data movement before task allocation**
- **Perform as many tasks as possible with the data at hand**



Intuition

- **Consider data movement before task allocation**
- **Perform as many tasks as possible with the data at hand**



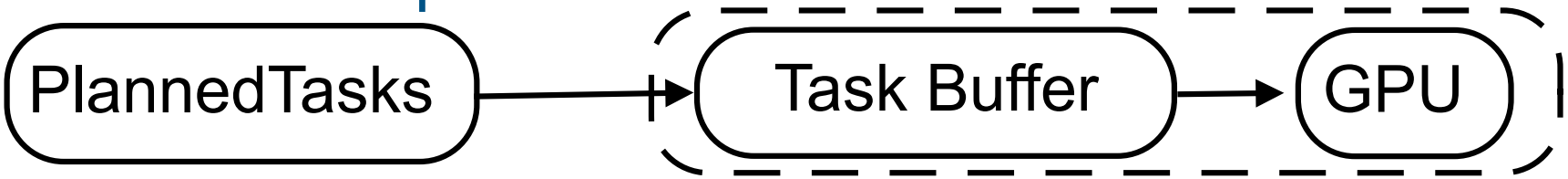
Strategy

Filling $plannedTasks_k$

Everything is distributed. The scheduling is done by PU_k when it needs new tasks.

- Find $D_{opt} \in dataNotInMem_k$ such that the number of tasks depending on D_{opt} and on other data already in memory is maximum (if there is a tie: tiebreak with priority, transfer time and most total unprocessed task)
- $plannedTasks_k \leftarrow$ set of unprocessed tasks depending only on D_{opt} and on other data already in memory
- Remove D_{opt} from $dataNotInMem_k$
- If D_{opt} does not allow to process at least one "free" task -> Select a random unprocessed task T

Eviction and optimizations



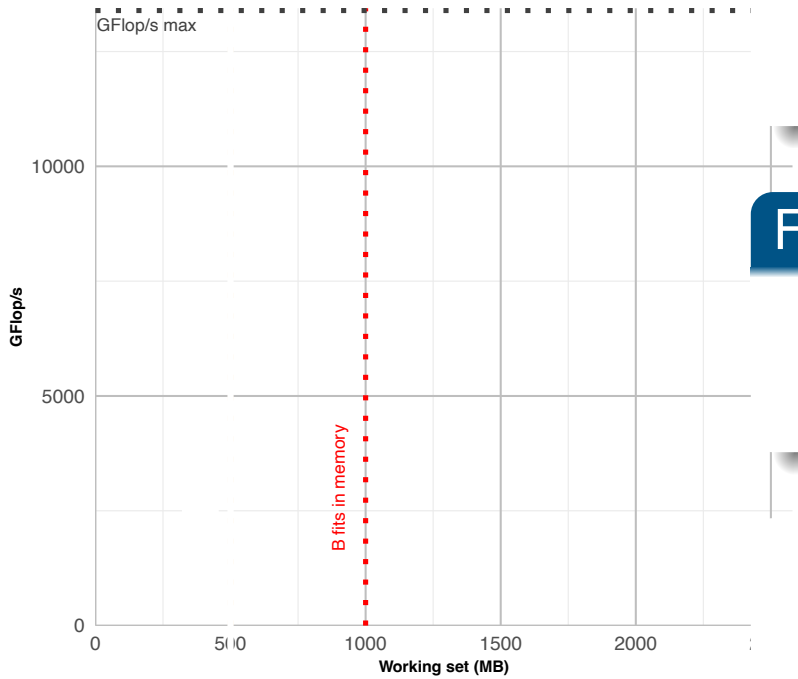
Our eviction policy: LUF (Least Used in the Future)

1. If possile, try to evict data not useful for any task in $taskBuffer_k$ and used by a minimal number of tasks in $plannedTasks_k$
2. Else, apply Belady's optimal rule on tasks already allocated
3. Then, update $plannedTasks_k$

Improvements

- Using the transfer time and task's length to favor CUDA-CUDA transfers

Experimental settings



For limited distributed memory (GPUs)

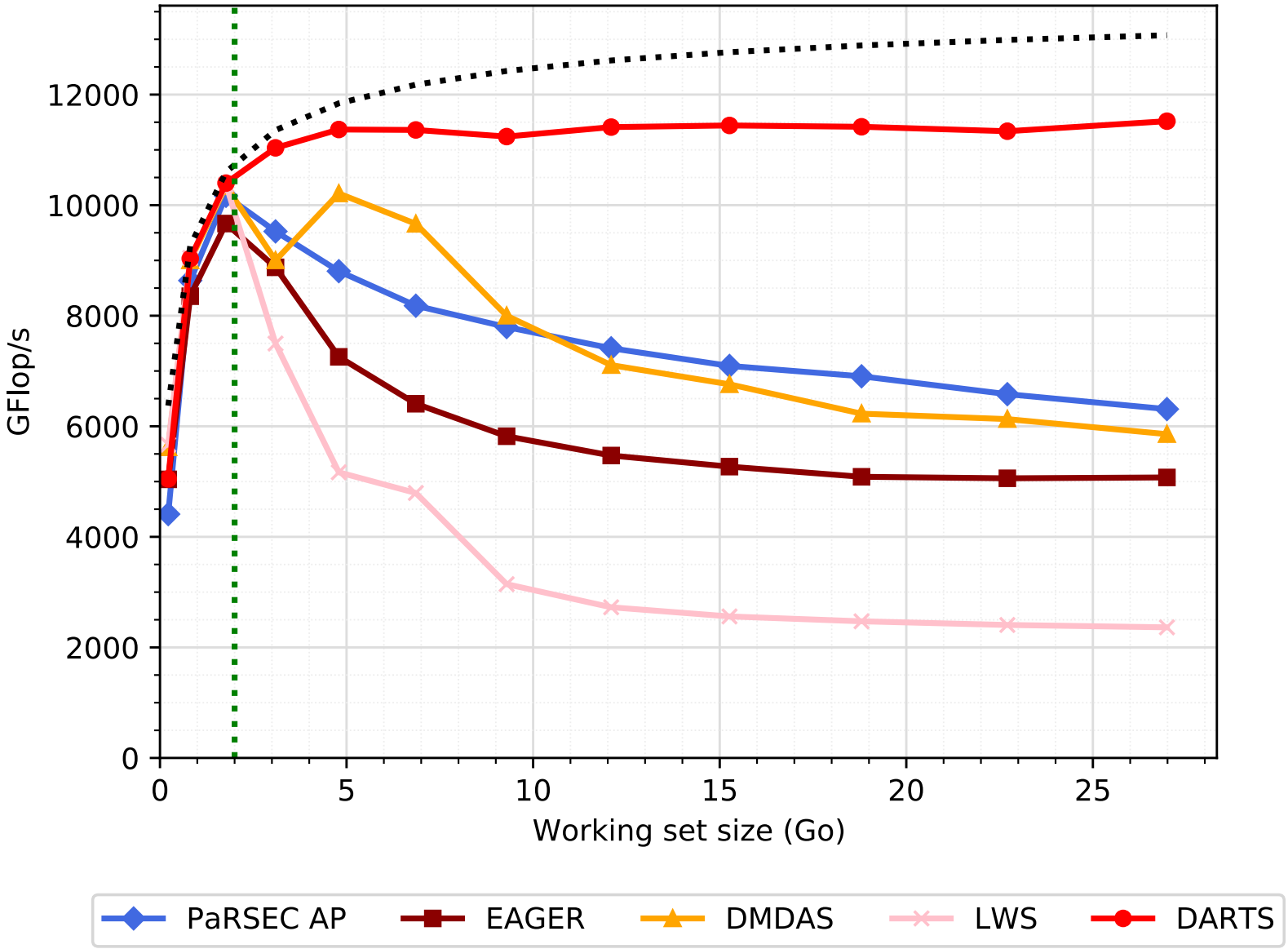
- Tesla V100 GPUs
- PCI bandwidth of 12000 MB/s

For limited shared memory (CPU)

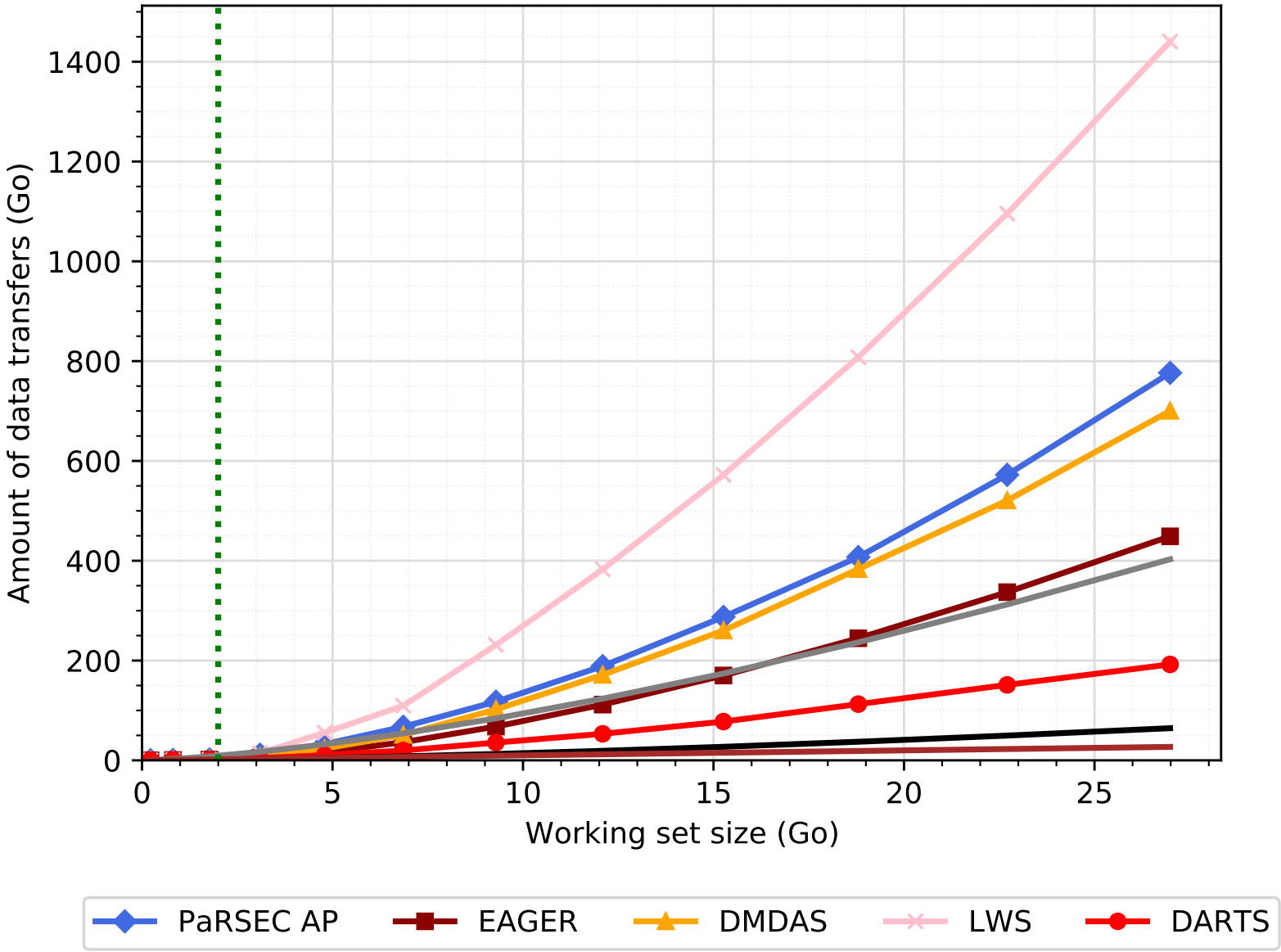
- AMD EPYC 7642: 48 cores/CPU
- Disk transfer bandwith: 300 MB/s

- PU memory limited to 2 $GB \rightarrow$ To better distinguish performance on small datasets

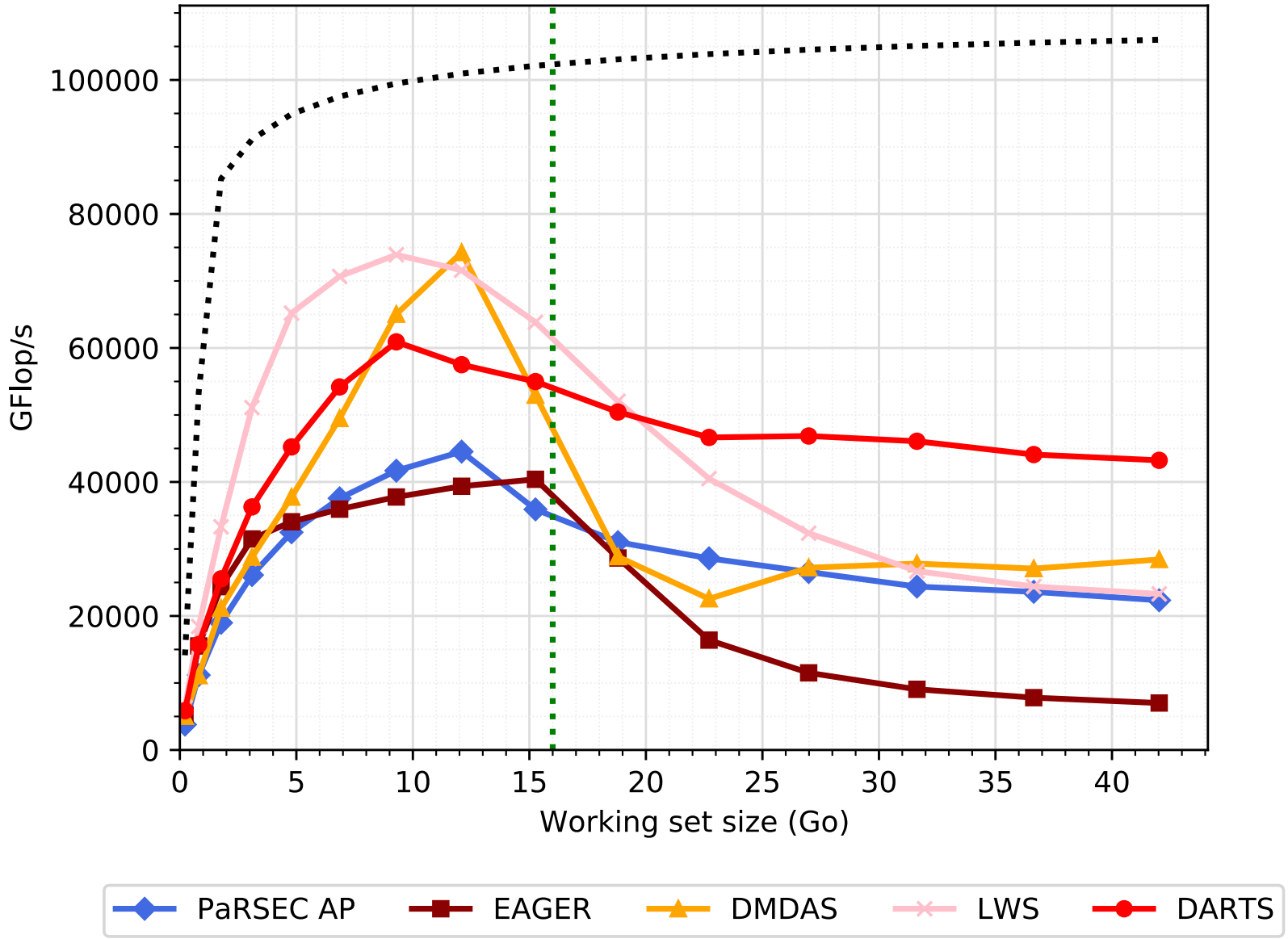
Cholesky with 2 GPUs



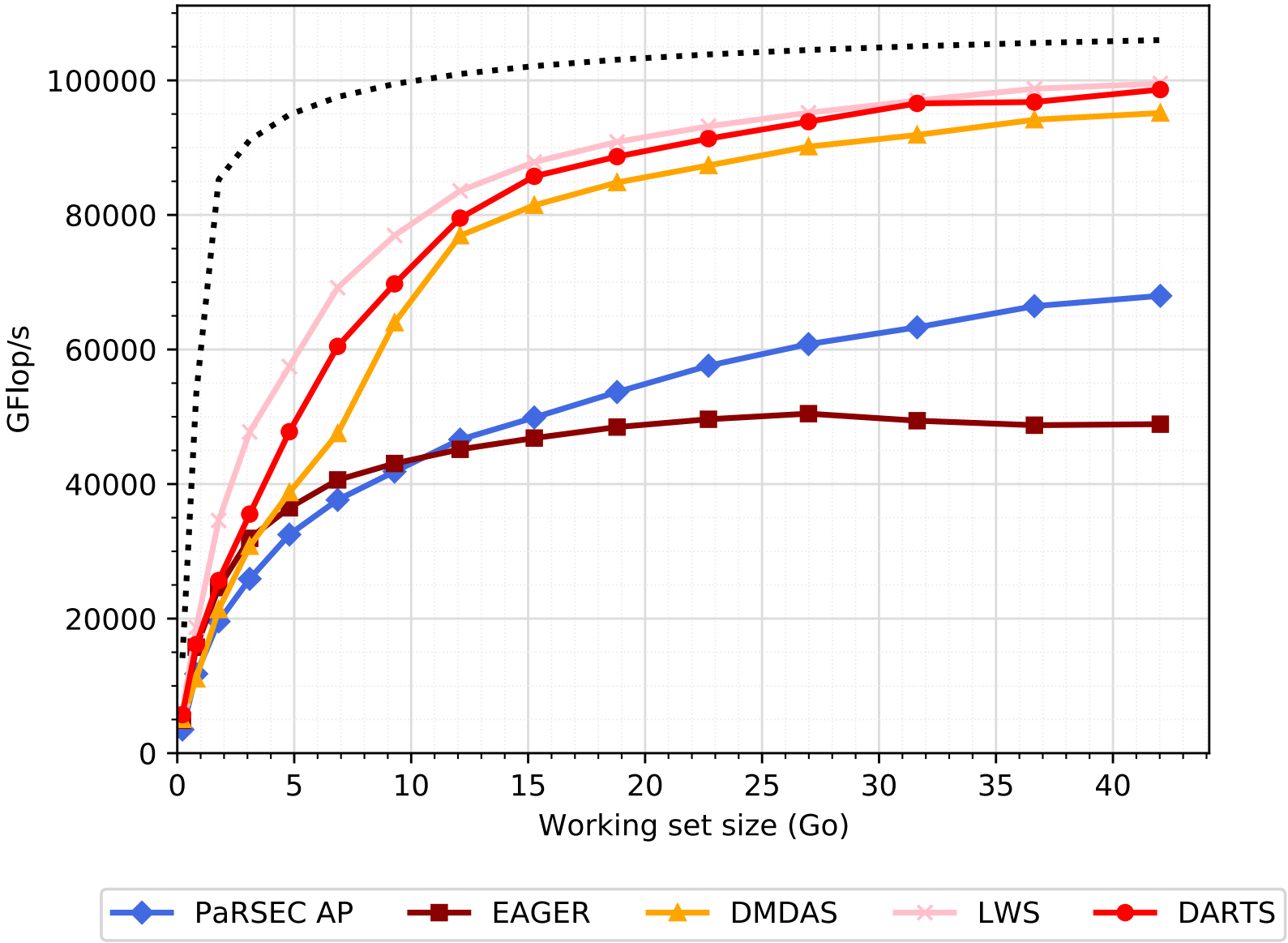
Cholesky with 2 GPUs



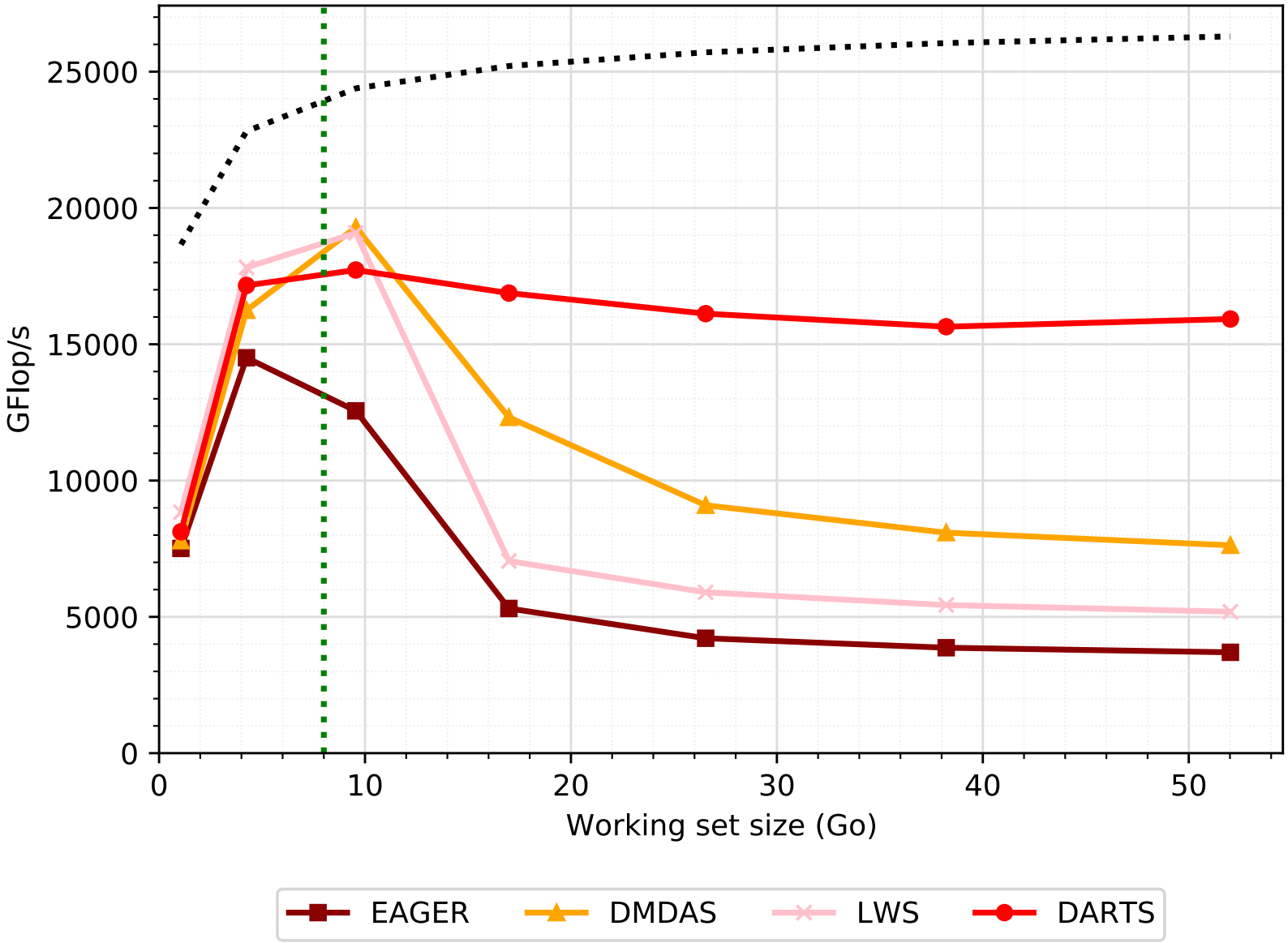
Cholesky with 8 GPUs



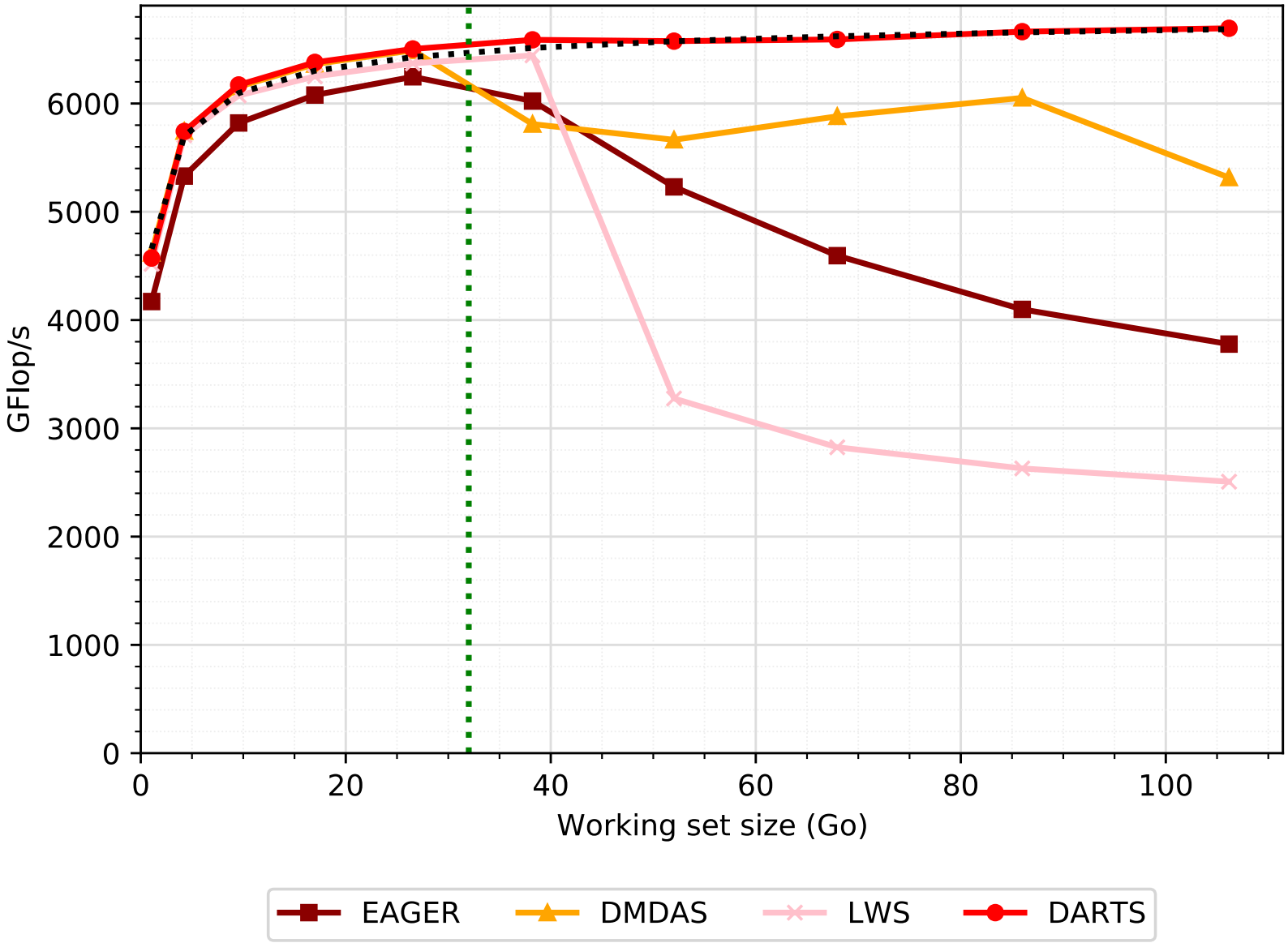
Cholesky with 8 GPUs and no memory limitation



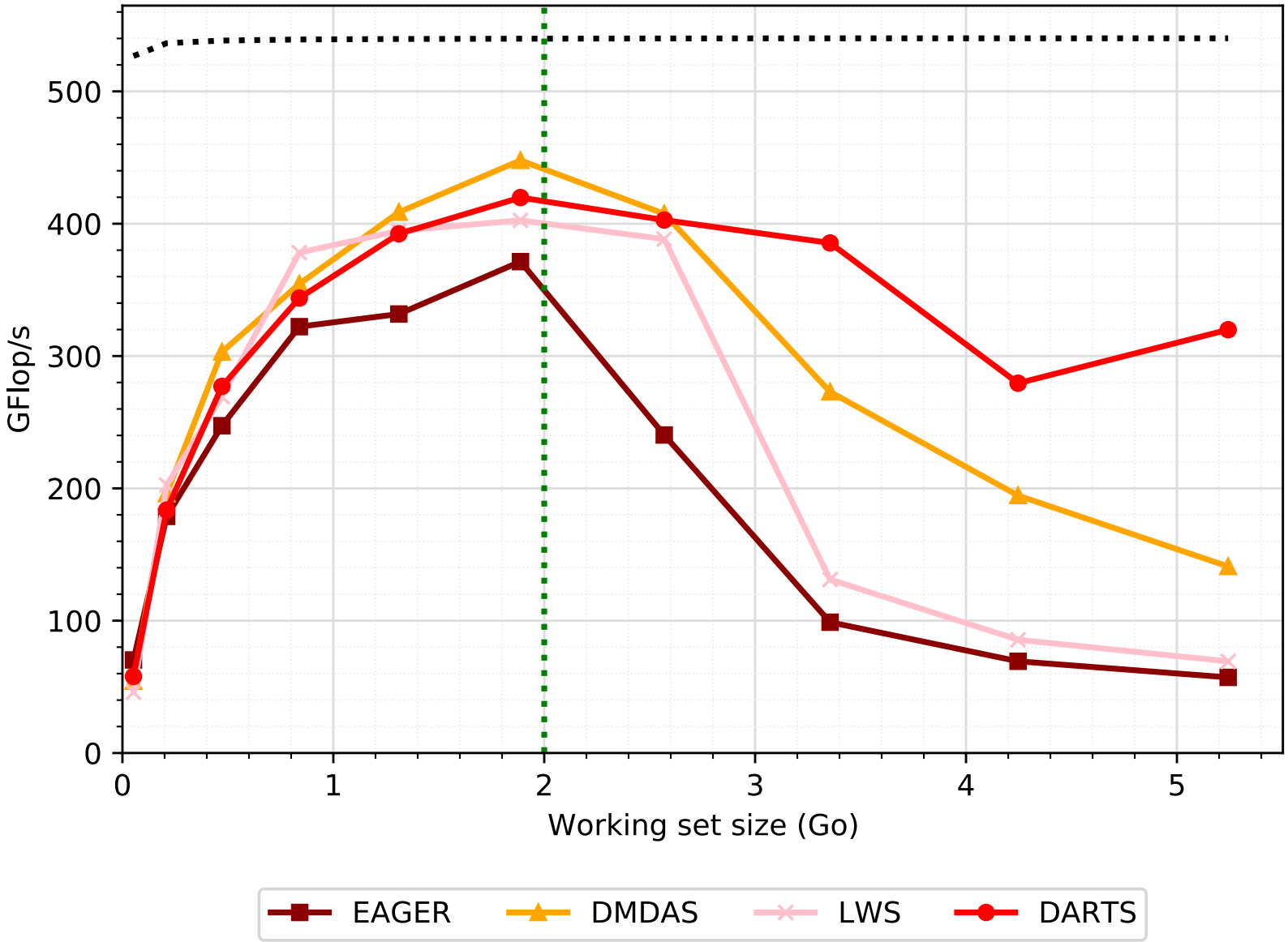
LU with 4 GPUs



LU with 1 GPU and no memory limitation



LU with 48 CPU cores



Conclusion

Limiting data movements is crucial

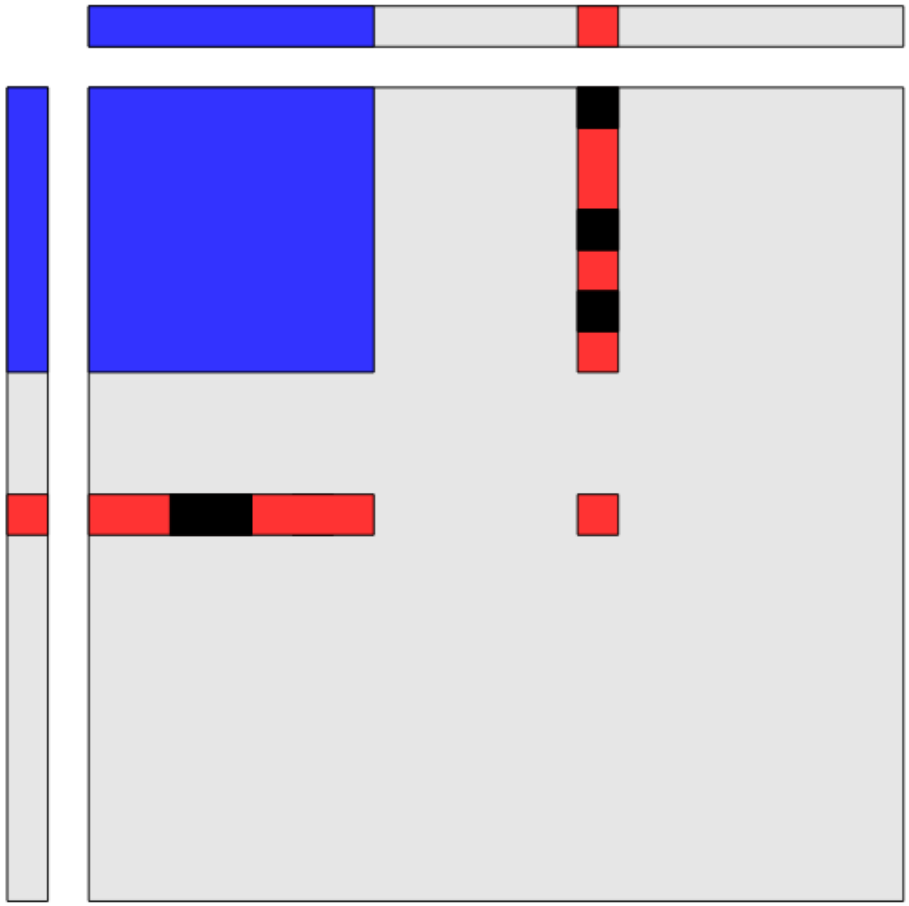
➔ **New strategy: DARTS, focused on data movement**

Tested on 2D matrix multiplication, Cholesky decomposition, LU, GEMM and sparse matrix multiplication ➔ DARTS can achieve good performance when the memory is not constrained and always get very good performance when it is a scarce resource. Thanks to StarPU we observed that DARTS can perform using GPUs and CPU.

Future works

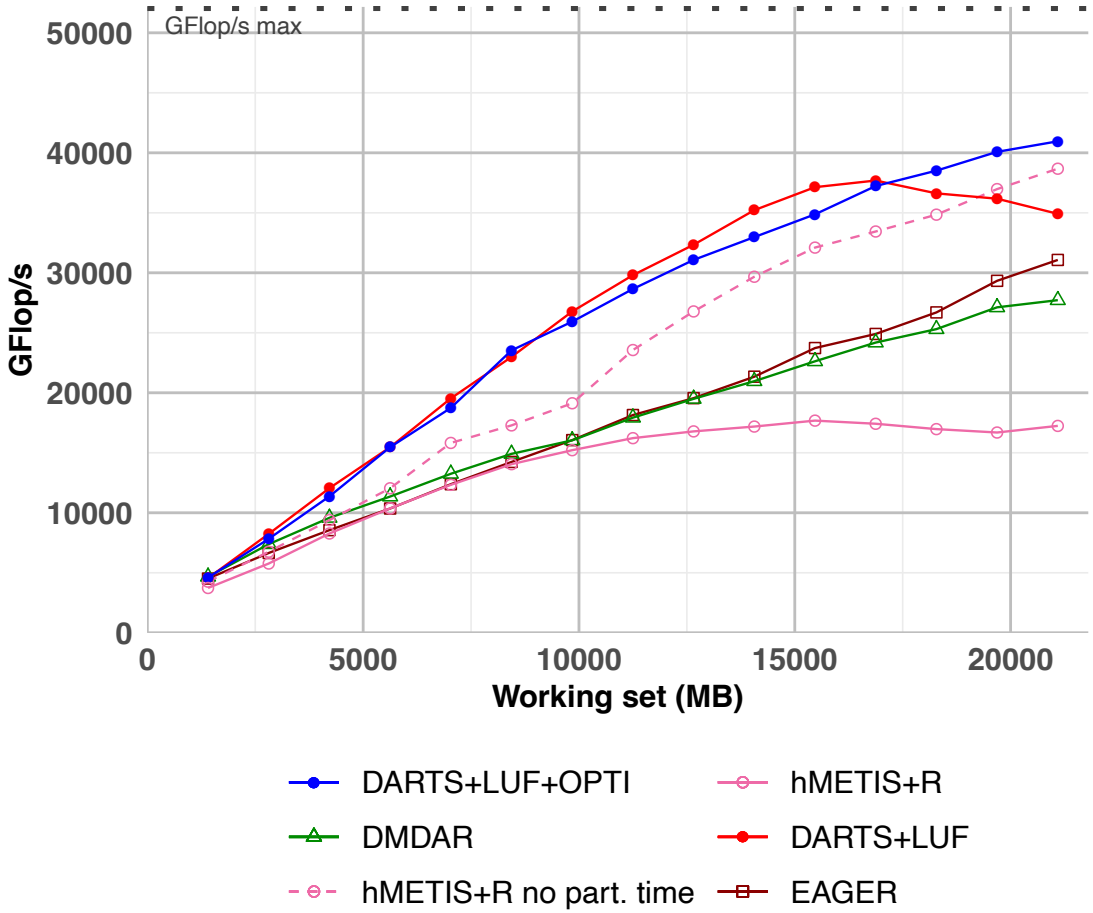
- Test DARTS with other out-of-core applications
- Using one instance of DARTS per memory node
- Improve computational complexity. It's holding us back to get good performances on application with a large number of ready task
- Manage multiple MPI nodes

Joker slide



Source: Analysis of Dynamic Scheduling Strategies for Matrix Multiplication on Heterogeneous Platforms - Marchal - Beaumont

Sparse 2D matrix multiplication without memory limitation (32GB by GPU) with 4 Tesla V100 GPUs



- No memory constraint
- DARTS produces a **processing order that best distributes transfers over time**