Memory-Aware Scheduling Of Tasks Sharing Data On Multiple GPUs

Maxime Gonthier, Loris Marchal, Samuel Thibault

Memory-Aware Scheduling Of Tasks Sharing Data On Multiple GPUs

Maxime GONTHIER - Samuel THIBAULT - Loris MARCHAL - maxime.gonthier@ens-lyon.fr - samuel.thibault@inria.fr - loris.marchal@ens-lyon.fr

Motivation: Extract peak performance from GPUs

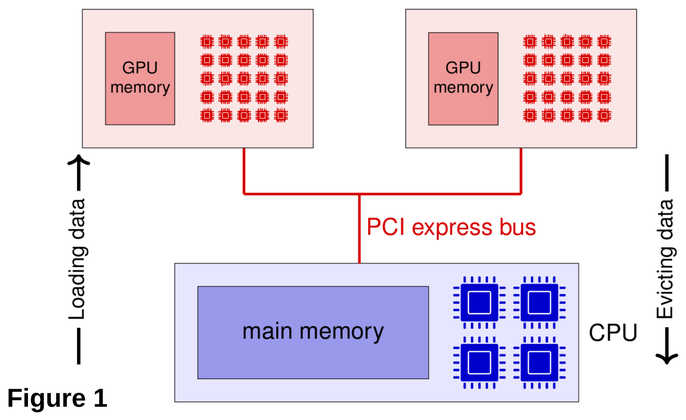
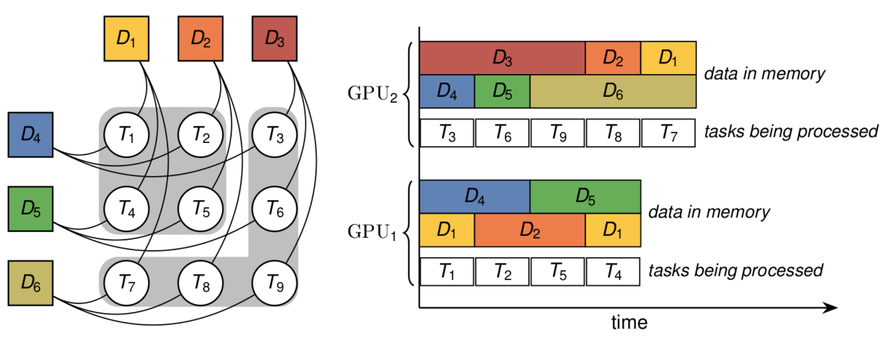


Figure 1

- Scientific application uses a huge quantity of data
- GPUs can calculate quickly but their memory are limited
- The PCI bus becomes a bottleneck when large amounts of data need to be loaded

To get peak performance we need to minimize the number of access to the main memory

Framework and objectives



- Independant tasks sharing data
- Same size data
- Multiple GPUs with limited memory

Figure 2: Example with a memory of size 2 data. The graph of input data dependencies is shown on the left. The figure on the right corresponds to the partition and schedule produced by the scheduler

Objective 1: Minimize the number of tasks on each GPU
Objective 2: Minimize the amount of data loads on each GPU

Algorithms from the state of the art

- **Using (hyper-)graph partitioning: hMETIS**
  - Hypergraph ⇒ Represents a data being shared by multiple tasks
  - hMETIS produces subsets of task that maximize the amount of edges (data shares) inside each subset
  - We add dynamic reordering and task stealing to improve performance
- **EAGER**

  Process tasks in the order in which it receives them ⇒ our baseline
- **Deque Model Data Aware Ready (DMDAR):**

  Schedules tasks where their completion times are expected to be minimal. Uses a ready strategy to favor tasks whose data has already been loaded into memory.

A novel strategy: Data-Aware Reactive Task Scheduling (DARTS)

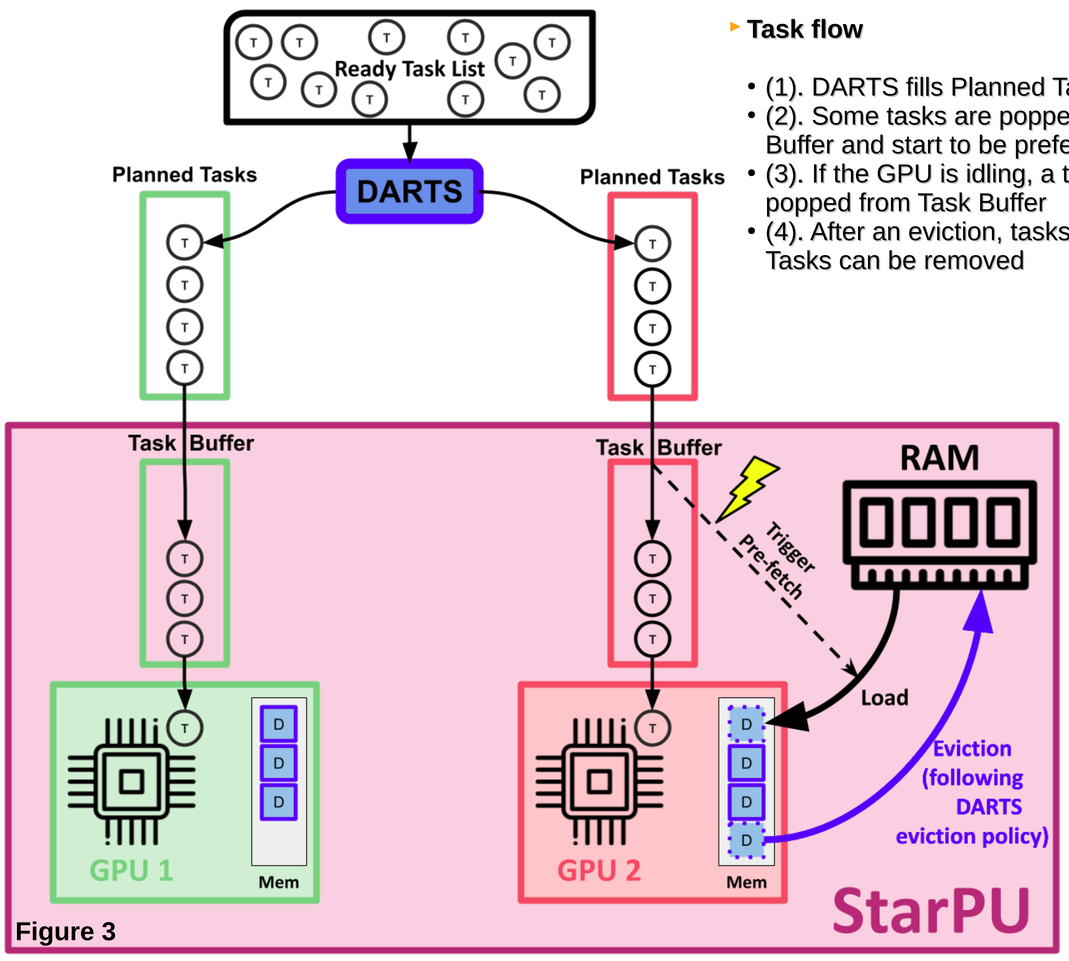


Figure 3

- **Task flow**
  - (1). DARTS fills Planned Tasks
  - (2). Some tasks are popped by Task Buffer and start to be prefetched
  - (3). If the GPU is idling, a task is popped from Task Buffer
  - (4). After an eviction, tasks in Planned Tasks can be removed

Intuition : Consider data locality before task allocation

- **Strategy**
  - (1).Find the optimal data D such that the number of tasks depending on D and on other data in memory is maximum
  - (2). Fill plannedTasks with these tasks
- **Our eviction policy : Least Used in the Future (LUF)**
  - If possible, evict data not useful for any task in taskBuffer and used by a minimal number of tasks in plannedTasks
  - Otherwise, evict data whose next usage is the furthest in the future in plannedTasks
  - Update plannedTasks

Experimental evaluations on independent taskset

- Tesla V100 GPUs with a 12000 MB/s PCI bandwidth
- GPU memory limited to 500 MB to better distinguish the performance even on small datasets
- 10 runs per experiment (deviation < 2%)

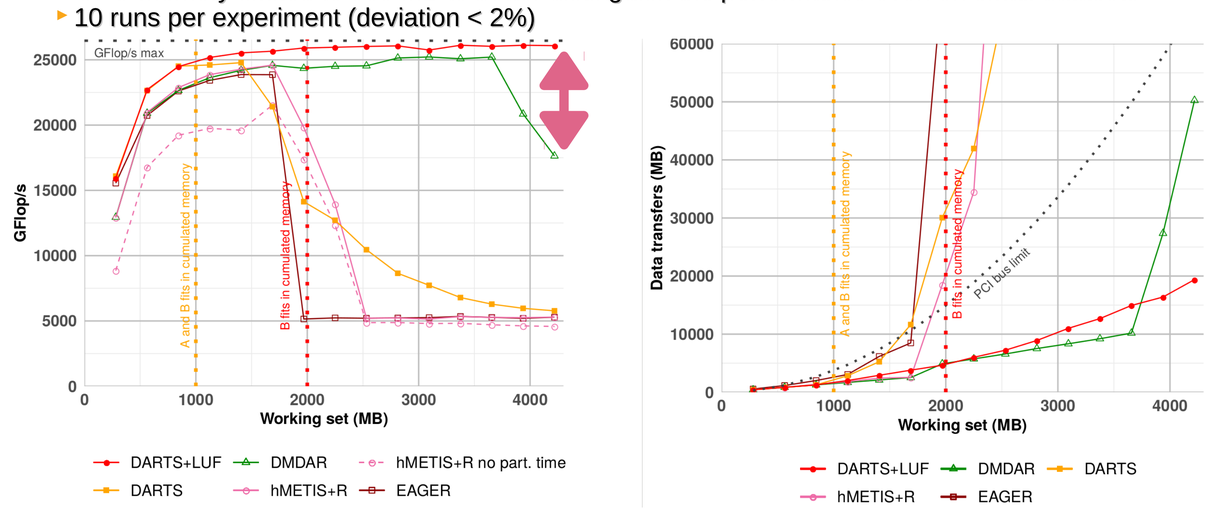


Figure 4: GFlop/s (left) and data transfers (right) on a tiled (1 x n) Matrix Multiplication in outer product order with 2 Tesla V100 GPUs

Dotted horizontal black line: maximum GFlop/s the GPUs can achieve
Red dotted vertical line: the cumulated GPUs memory can fit exactly one of the two input matrices
Orange dotted line: It can accommodate both input matrices
Black diagonal dotted line: maximum number of transfers that can be done during the minimum time for computation -> the hard limitation induced by the PCI bus bandwidth

- Pathological matrix size due to LRU's eviction policy when the memory is a constraint (after the red dotted line) for EAGER, hMETIS and DARTS
- DMDAR suffers from a conflict between prefetches and evictions
- DARTS+LUF stays close to the asymptotic goal !

DARTS nicely overlaps communications and computations

- hMETIS suffers from a large scheduling cost
- DARTS can be used with a reduced complexity (DARTS+OPTI) by stopping the search for the optimal data earlier.
- Even without memory limitation, DARTS outperforms DMDAR

DARTS produces a processing order that best distributes transfers over time ⇒ Better performance even without memory limitation

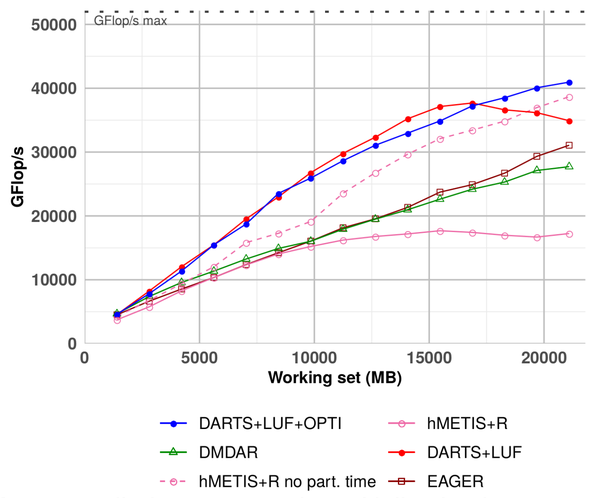


Figure 5: Tiled sparse Matrix Multiplication in outer product order without memory limitation (32GB by GPU) with 4 Tesla V100 GPUs

Experimental evaluations with dependencies

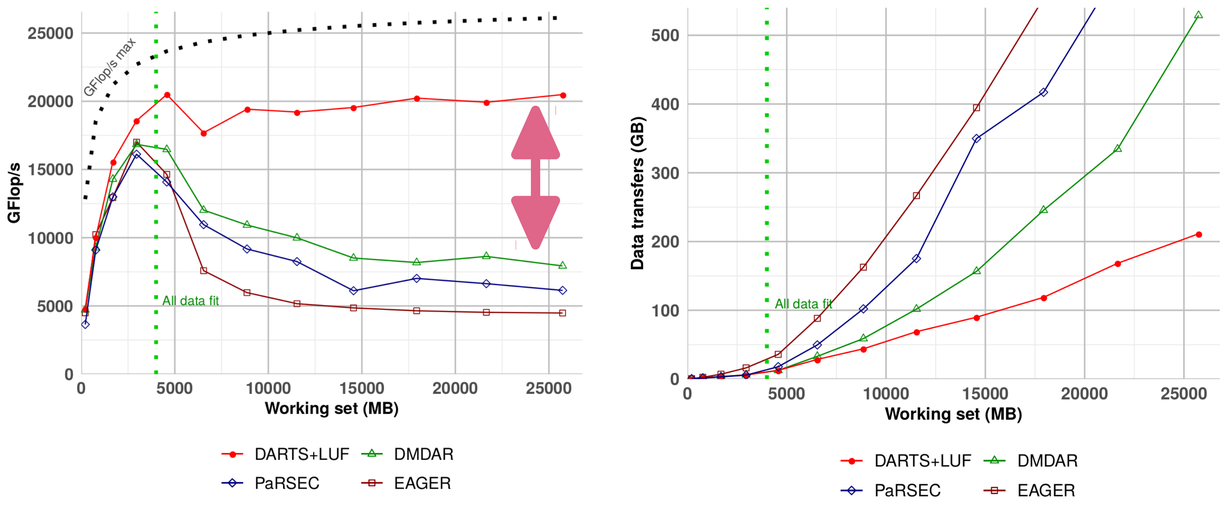


Figure 6: Cholesky decomposition with 2 Tesla V100 GPU

- When choosing the optimal data with dependencies, DARTS takes into account the transfer time, priority and length of task associated with each data.
- DARTS achieves better performances once the memory is a constraint (on the right of the green dotted line) and behaves as DMDAR when the memory is not limited (on the left). It also greatly reduces data transfers. DARTS outperforms the scheduler from PaRSEC, a state-of-the-art runtime.

Summary and conclusion

- **Motivations :**
  - High-performance computing applications, like weather and climate forecasting, have an increasing demand in computer power
  - GPUs have a large computation power but have a limited memory. This, coupled with the share of a single PCI express bus by multilple GPUs induce a large amount of data transfers which reduces performance ⇒ Limiting data movement is crucial to extract the most out of GPUs
- ⇒ We provide a new algorithm (DARTS) and a new eviction policy (LUF) focused on data locality
- **Results : DARTS+LUF achieves good performance because it :**
  - Limits data transfers thanks to the finding of an optimal data and an adapted eviction policy
  - Overlaps communication and computations by distributing transfers over time
  - Can be used with a reduced complexity
  - Can deal with priorities
- **Areas for improvement:**
  - Manage multiple MPI nodes
  - Use other out-of-core applications (DARTS is not specific to GPUs!)

You can learn more about my work on my website