Memory-Aware Scheduling of Tasks Sharing Data on Multiple GPUs with Dynamic Runtime Systems
Maxime Gonthier, Loris Marchal, Samuel Thibault

To cite this version:
Maxime Gonthier, Loris Marchal, Samuel Thibault. Memory-Aware Scheduling of Tasks Sharing Data on Multiple GPUs with Dynamic Runtime Systems. 15th Scheduling for Large Scale Systems Workshop, Jun 2022, Fréjus, France. hal-04090618

HAL Id: hal-04090618
https://inria.hal.science/hal-04090618
Submitted on 5 May 2023

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Memory-Aware Scheduling of Tasks Sharing Data on Multiple GPUs with Dynamic Runtime Systems

15th Scheduling for Large Scale Systems Workshop - Fréjus - Juin 2022

Maxime GONTHIER - Loris MARCHAL - Samuel THIBAULT

maxime.gonthier@ens-lyon.fr
LIP - ROMA - LaBRI - STORM
Motivation: Extract peak performances from GPUs

- GPUs are fast but have a limited memory.
- GPUs share the same PCI bus with limited bandwidth.

---

Motivation: Extract peak performances from GPUs

- GPUs are fast but have a limited memory.
- GPUs share the same PCI bus with limited bandwidth.
Framework: An example with 2D grid dependencies

- Independent tasks sharing data
- Limited GPU memory
Goal: minimize makespan

- Balancing tasks among GPUs → Reduce total execution time
- Ordering tasks inside each GPU → Reduce data transfers
Problem modeling

$\text{GPU}_k$ wants to process task $T$

1. **Evict** selected data (possibly none) from the memory of $\text{GPU}_k$
2. **Load** data for $T$ that are not yet in memory
3. **Process** $T$ on $\text{GPU}_k$

**Hypothesis**

- Independant and homogeneous tasks
- Same size data
A bi-objective optimization problem

**Objective 1: Load Balancing**

\[ \text{minimize} \ max_k \ \text{number of tasks allocated to GPU}_k \]

**Objective 2: Data Movement**

Minimize the number of data loads from the main memory:

\[ \#\text{Loads}_k = \sum \text{data load for each } T \text{ computed on GPU}_k \]

\[ \text{minimize} \ \sum_k \#\text{Loads}_k \]
Algorithms

2 schedulers from STARPU
- EAGER (our baseline)
- Deque Model Data Aware Ready (DMDAR)

1 algorithm adapted from the literature
- hMETIS

Novel algorithm
- Data-Aware Reactive Task Scheduling (DARTS)
Dynamic scheduler of STARPU: DMDAR

Strategy

Schedule tasks so their completion time is minimal based on computation + data movement

+ Ready reordering heuristic on GPU_k

**input**: List L of tasks allocated on GPU_k

**while** L ≠ ∅ **do**

- Search first T ∈ L requiring the fewest data transfers
- Wait for all data in D(T) to be in GPU_k memory
- Start processing T

**end**
Using (hyper-)graph partitioning: hMETIS

Hypergraph → Represent a data being used by different tasks
Accurately represent data sharing

Strategy

1. Apply hMETIS to create tasks subsets
2. Each subset is allocated to a GPU
3. Use the Ready strategy
4. Dynamic load balancing using task stealing
DARTS: Task flow

DARTS is demand driven. It uses 2 queues. Task selection detailed next slide.

- \( \text{plannedTasks}_1 \)
- \( \text{plannedTasks}_2 \)
- \( \text{taskBuffer}_1 \)
- \( \text{taskBuffer}_2 \)

GPU 1 is asking for a task!
DARTS: Task flow

If $plannedTasks_k = \emptyset \rightarrow$ DARTS fill $plannedTasks_k$

$plannedTasks_1$

$plannedTasks_2$

$taskBuffer_1$

$taskBuffer_2$

$GPU_1$

$GPU_2$

$GPU_1$ is asking for a task!
DARTS: Task flow

If $\text{plannedTasks}_k = \emptyset \rightarrow$ DARTS fill $\text{plannedTasks}_k$

GPU1 is asking for a task!
DARTS: Task flow

If $\text{plannedTasks}_k = \emptyset \rightarrow$ DARTS fill $\text{plannedTasks}_k$

$\text{plannedTasks}_1$

$\text{plannedTasks}_2$

$\text{taskBuffer}_1$

$\text{taskBuffer}_2$

GPU$_1$

GPU$_2$

GPU$_1$ is asking for a task!
DARTS: Task flow

If $\text{taskBuffer}_k = \emptyset \rightarrow \text{GPU}_k$ will pop $\text{plannedTasks}_k$

- $\text{plannedTasks}_1$
- $\text{plannedTasks}_2$
- $\text{taskBuffer}_1$
- $\text{taskBuffer}_2$
- $\text{GPU}_1$
- $\text{GPU}_2$

$\text{GPU}_1$ is asking for a task!
DARTS: Task flow

If $\text{taskBuffer}_k = \emptyset \rightarrow \text{GPU}_k$ will pop $\text{plannedTasks}_k$

$\text{plannedTasks}_1$

$\text{plannedTasks}_2$

$\text{taskBuffer}_1$

$\text{taskBuffer}_2$

$\text{GPU}_1$

$\text{GPU}_2$

$\text{GPU}_1$ is asking for a task!
DARTS: Task flow

Data from $taskBuffer_k$ is prefetched

GPU

1

$plannedTasks_1$

GPU

2

$plannedTasks_2$

GPU

1

$taskBuffer_1$

GPU

2

$taskBuffer_2$

GPU

1 is asking for a task!

GPU

2 is asking for a task!
DARTS: Task flow

In DARTS, there are two queues: `plannedTasks_1` and `plannedTasks_2`. If `plannedTasks_k` is empty, DARTS fills it. If `taskBuffer_k` is empty, the corresponding GPU will pop `plannedTasks_k`.

Data from `taskBuffer_k` is prefetched.

GPU_1 is asking for a task!
DARTS: Task flow

$p\text{lannedTasks}_1$

$\text{taskBuffer}_1$

$\text{GPU}_1$

$p\text{lannedTasks}_2$

$\text{taskBuffer}_2$

$\text{GPU}_2$

$\text{GPU}_2$ is asking for a task!
DARTS: Task flow

- **plannedTasks\textsubscript{1}**
- **plannedTasks\textsubscript{2}**
- **taskBuffer\textsubscript{1}**
- **taskBuffer\textsubscript{2}**

GPU\textsubscript{1} is asking for a task!

GPU\textsubscript{2} is asking for a task!
DARTS: Task flow

If \( \text{plannedTasks}_k = \emptyset \rightarrow \text{DARTS fill} \) \( \text{plannedTasks}_k \)

If \( \text{taskBuffer}_k = \emptyset \rightarrow \text{GPU}_k \) will pop \( \text{plannedTasks}_k \)

Data from \( \text{taskBuffer}_k \) is prefetched

GPU\(_1\) is asking for a task!

GPU\(_2\) is asking for a task!
DARTS: Task flow

plannedTasks\(_1\)

plannedTasks\(_2\)

taskBuffer\(_1\)

taskBuffer\(_2\)

GPU\(_1\)

GPU\(_2\)

GPU\(_2\) is asking for a task!
DARTS: Task flow

$plannedTasks_1$

$plannedTasks_2$

$taskBuffer_1$

$taskBuffer_2$

$GPU_1$

$GPU_2$
DARTS: Strategy

Intuition: consider data locality before task allocation

Perform as many tasks as possible with the data at hand

When no more available tasks with data at hand:

- Find data $D_{optimal}$ such that the number of tasks depending on $D_{optimal}$ and on other data already in memory is maximum
- $plannedTasks_k \leftarrow$ set of unprocessed tasks depending only on $D_{optimal}$ and on other data already in memory
DARTS: Eviction and optimizations

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

1. If possible, evict data not useful for any task in \(\text{taskBuffer}_k\) and used by a minimal number of tasks in \(\text{plannedTasks}_k\)
2. Otherwise, apply Belady’s rule on tasks already allocated
3. Update \(\text{plannedTasks}_k\)

Improvements

- **3inputs**: Extension to deal with a heterogenous number of data per task
- **OPTI**: Reduce scheduling complexity by stopping the search for \(D_{\text{optimal}}\) earlier
Visualization of DARTS processing order on the tiled dense Matrix Multiplication in outer product order
Experimental settings

- Real Tesla V100 GPUs and simulations on Simgrid
- PCI bandwidth not limited (12000 MB/s)
- GPU memory limited to 500 MB → To better distinguish performance on small datasets

Applications

- Tiled $(1 \times n)(n \times 1)$ dense and sparse Matrix Multiplication in outer product order: $C = A \times B$
- Tiled $(n \times n)$ Gemm: $C = A \times B$
- Tiled Cholesky decomposition (without dependencies) $(A = L \times L^T)$
Motivation and formalization

Algorithms

Experimental evaluation

Conclusion and future work

Tiled dense Matrix Multiplication in outer product order with 2 real Tesla V100 GPUs

- EAGER, hMETIS & DARTS: Pathological matrix size after the red line
- DMDAR: Conflict between prefetch and eviction
- DARTS+LUF outperforms DMDAR even with more data transfers
  → DARTS is better at overlapping communication and computations
Tiled dense Matrix Multiplication in outer product order with 4 real Tesla V100 GPUs
Cholesky decomposition with 4 real Tesla V100 GPUs

OPTI reduces scheduling time which improves performance

DMDAR suffers from a large scheduling time
Tiled Gemm with 4 Tesla V100 GPUs in simulation

![Graph showing performance metrics for different algorithms with working set and GFlop/s as variables.]

- **GFlop/s max**
- **Working set (MB)**
- **DARTS+LUF−3inputs**
- **DMDAR**
- **EAGER**
- **DARTS+LUF**
- **hMETIS+R**
Tiled sparse Matrix Multiplication in outer product order without memory limitation (32GB by GPU) with 4 real Tesla V100 GPUs

DARTS produces a processing order that best distributes transfers over time

hMETIS suffers from a significant partitioning cost
Conclusion and future work

Limiting data movements is crucial to extract the most out of GPUs.

Our contribution → **DARTS+LUF, focused on data locality**

DARTS achieves very 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**.

Areas for improvement:

- Reduce computational complexity.
- Consider tasks with dependencies.
- Take inter-GPU communications into account.
- Manage multiple MPI nodes.
Dynamic Scheduling Strategies for Matrix Multiplication

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