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

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 $\rightarrow$ Reduce total execution time
- Ordering tasks inside each GPU $\rightarrow$ Reduce data transfers

data in memory

tasks being processed

time

Balancing tasks among GPUs $\rightarrow$ Reduce total execution time
Ordering tasks inside each GPU $\rightarrow$ 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**

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

**Objective 2: Data Movement**

Minimize the number of data loads from the main memory:

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

\[
\text{minimize } \sum_k \#\text{Loads}_k
\]
## 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 $\text{GPU}_k$

**input**: List $L$ of tasks allocated on $\text{GPU}_k$

**while** $L \neq \emptyset$ **do**

- Search first $T \in L$ requiring the fewest data transfers
- Wait for all data in $D(T)$ to be in $\text{GPU}_k$ memory
- Start processing $T$

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

Hypergraph $\rightarrow$ 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.
DARTS: Task flow

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

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

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

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

\[ \begin{align*}
\text{plannedTasks}_1 \\
\text{taskBuffer}_1 \\
\text{GPU}_1
\end{align*} \]

\[ \begin{align*}
\text{plannedTasks}_2 \\
\text{taskBuffer}_2 \\
\text{GPU}_2
\end{align*} \]

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

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

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

If $taskBuffer_k = \emptyset \rightarrow GPU_k$ will pop $plannedTasks_k$

GPU_1 is asking for a task!
DARTS: Task flow

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

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

Data from $taskBuffer_k$ is prefetched

$plannedTasks_1$

$plannedTasks_2$

$taskBuffer_1$

$taskBuffer_2$

GPU$_1$

GPU$_2$

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

\[ \text{plannedTasks}_1 \]

\[ \text{plannedTasks}_2 \]

\[ \text{taskBuffer}_1 \]

\[ \text{taskBuffer}_2 \]

\[ \text{GPU}_1 \]

\[ \text{GPU}_2 \]

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

\[
\text{plannedTasks}_1 \quad \text{plannedTasks}_2
\]

\[
\text{taskBuffer}_1 \quad \text{taskBuffer}_2
\]

\[
\text{GPU}_1 \quad \text{GPU}_2
\]

\[
\text{GPU}_2 \text{ is asking for a task!}
\]
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**
  - plannedTasks$_1$
  - plannedTasks$_2$

- **taskBuffer**
  - taskBuffer$_1$
  - taskBuffer$_2$

- **GPU**
  - GPU$_1$
  - GPU$_2$

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

If plannedTasks$_k$ = ∅ → DARTS fill plannedTasks$_k$

If taskBuffer$_k$ = ∅ → GPU$_k$ will pop plannedTasks$_k$

Data from taskBuffer$_k$ is prefetched.

GPU$_1$ is asking for a task!

GPU$_2$ is asking for a task!
DARTS: Task flow

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

plannedTasks\textsubscript{2} \rightarrow \text{GPU} \text{2} \text{is asking for a task!}

taskBuffer\textsubscript{1} \rightarrow \text{GPU} \text{1} \text{will pop plannedTasks}_1

\text{taskBuffer}_1 \text{ is prefetched from GPU}_1

\text{taskBuffer}_2 \text{ is prefetched from GPU}_2

\text{STARPU}

\text{GPU}_1 \rightarrow \text{GPU}_1 \text{ will pop plannedTasks}_1

\text{GPU}_2 \rightarrow \text{GPU}_2 \text{ will pop plannedTasks}_2
DARTS: Task flow

- **plannedTasks**
  - GPU1
  - GPU2

- **taskBuffer**
  - GPU1
  - GPU2

DARTS is demand driven. It uses 2 queues. Task selection detailed next slide.
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 $taskBuffer_k$ and used by a minimal number of tasks in $plannedTasks_k$
2. Otherwise, apply Belady’s rule on tasks already allocated
3. Update $plannedTasks_k$

Improvements

- **3inputs**: Extension to deal with a heterogenous number of data per task
- **OPTI**: Reduce scheduling complexity by stopping the search for $D_{optimal}$ 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) \((A = L \times L^T)\)
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

- 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