Memory-Aware Scheduling Of Tasks Sharing Data On Multiple GPUs
Maxime Gonthier, Samuel Thibault, Loris Marchal

To cite this version:
Maxime Gonthier, Samuel Thibault, Loris Marchal. Memory-Aware Scheduling Of Tasks Sharing Data On Multiple GPUs. HiPEAC ACACES 2022 - 18th International Summer School on Advanced Computer Architecture and Compilation for High-performance Embedded Systems, Jul 2022, Fiuggi, Italy. hal-04090607

HAL Id: hal-04090607
https://inria.hal.science/hal-04090607
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

Motivation: Extract peak performance from GPUs

- 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

- 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

- 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)

- 2 queues of tasks
  - Tasks in plannedTasks can be prefetched and can’t be removed
  - Tasks in GPU are being computed
  - Tasks in taskBuffer are being prefetched and can’t be removed
  - Tasks in GPU are being computed

Flow of tasks

- (1). DARTS fills plannedTasks
  - (2). Some tasks are popped by taskBuffer and start to be prefetched
  - (3). If the GPU is idling, a task is popped from taskBuffer

Intuition: Consider data locality before task allocation

- Strategy
  - (1). Find the optimal data D such that the number of tasks depending on D and 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 be limited by a number of tasks in plannedTasks
  - Otherwise, apply Belady’s rule on tasks already allocated
  - Update plannedTasks

Experimental settings

- Tesla V100 GPUs
- 12000 MB/s PCI bandwidth
- GPU memory limited to 500 MB to better distinguish the performance of different strategies even on small datasets

Experimental evaluations

- Large scheduling cost for all algorithms
- DMDAR suffers from a conflict between prefetches and evictions
- DARTS+LUF stay close to the asymptotic goal!

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 multiple GPUs induce a large amount of data transfers which reduces performance → Limiting data movement is crucial to extract the most out of GPUs.

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

Areas for improvement:

- Consider tasks with dependencies
- Consider inter-GPU communications (NVLink)
- Manage multiple MPI nodes

Summary and conclusion