EURO 4SEE

Optimizing Deep Learning Systems for Hardware

Assoc. Prof. Erdem AKAGÜNDÜZ, METU

ncc@ulakbim.gov.tr

Part II : Hardware & Memory Hierarchy

- PII.a : Existing Solution: CPU, GPU, TPU, FPGA, ASIC basics
- PII.b : Memory hierarchy, bandwidth bottlenecks, movement costs
- PII.c : Precision

What Are We Optimizing?

- Deep learning frameworks run on powerful hardware, yet performance varies.
- The bottleneck is often not the math itself, but how data is stored and moved.
- To optimize deep learning (or any algorithm), we must first understand:
  - Where data lives (memory hierarchy).
  - How fast it moves (bandwidth, latency).
  - How costly movement is compared to computation.

Why talk about memory?

- In deep learning, most of the cost is not in raw arithmetic, but in moving data.
  - Optimizing performance = minimizing expensive memory transfers.
- To understand this, we must study the memory hierarchy and where bottlenecks occur.

Memory Hierarchy

- Memory hierarchy is the organization of storage components based on speed, size, and cost.
  - At the top: small, very fast, expensive (e.g., CPU registers, L1 cache).
  - At the bottom: large, slow, cheap (e.g., DRAM, SSD, HDD).
  - Principle: Data is moved between these levels to balance performance and capacity.
- Key trade-off: The faster the memory, the less of it we have.

Who controls this?

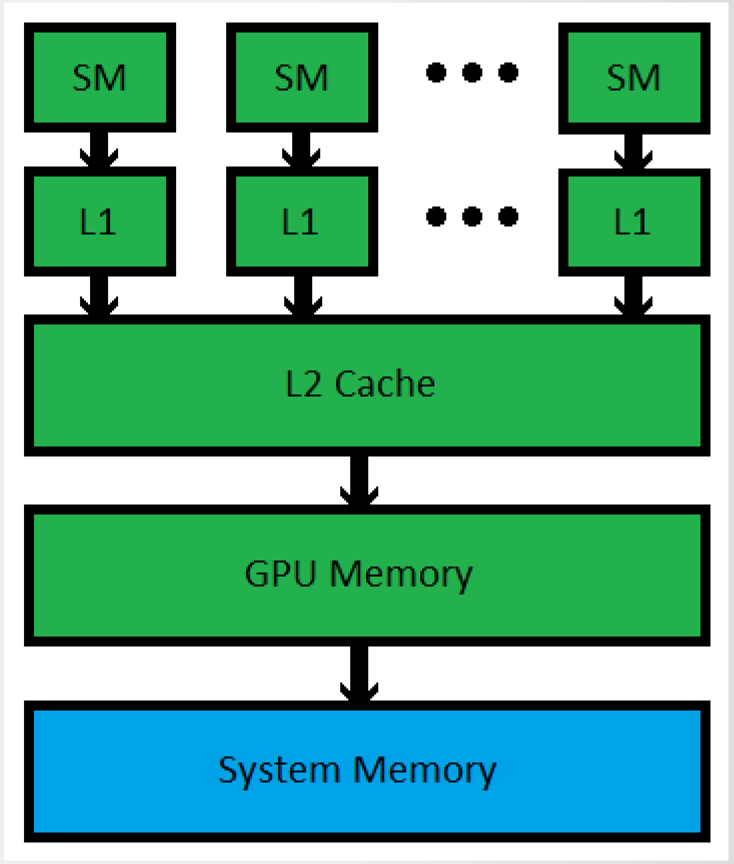
- So the framework asks for an operation.
  - The runtime (CUDA, cuDNN, MKL) maps it efficiently to the memory hierarchy.
  - The hardware executes it with its built-in cache and memory logic.
- Understanding the basic memory architecture of whatever system you’re programming for is necessary to create high performance applications.
- Most desktop systems consist of large amounts of system memory connected to a single CPU, which may have 2 or three levels or fully coherent cache.

Who controls this?

- The hardware itself (CPU/GPU/TPU) enforces the memory hierarchy
  - whether you have registers, caches, DRAM, etc.
- Compilers and runtime libraries (e.g., CUDA for NVIDIA GPUs, ROCm for AMD, XLA for TPUs, MKL for CPUs) decide how tensors are placed and moved within that hierarchy.
- Deep learning frameworks (PyTorch, TensorFlow, JAX) don’t directly manage caches or registers
  - instead, they call into CUDA/cuDNN, MKL, etc., which in turn optimize data layout and movement.

Example: NVIDIA Fermi

- This is a basic diagram of the memory structure in a modern system using NVIDIA's Fermi architecture.
- Each SM (streaming microprocessor) processes a stream of threads concurrently.
- "Streaming" emphasizes that the SM handles continuous flows of lightweight threads, keeping the GPU cores busy while hiding memory latency.



Bandwidth Bottleneck

- When the rate of moving data between memory and compute units is slower than the compute capability, hardware sits idle.
  - Even if FLOPs are plentiful, performance drops if the data cannot arrive fast enough.
- Parallelism helps:
  - More threads/cores can hide latency by working on other data while waiting.
  - GPUs/TPUs exploit massive parallelism to keep arithmetic units busy despite memory delays.
- Remember: Optimizing DL is not just about arithmetic—it's about moving data efficiently to sustain compute.

Movement Costs?

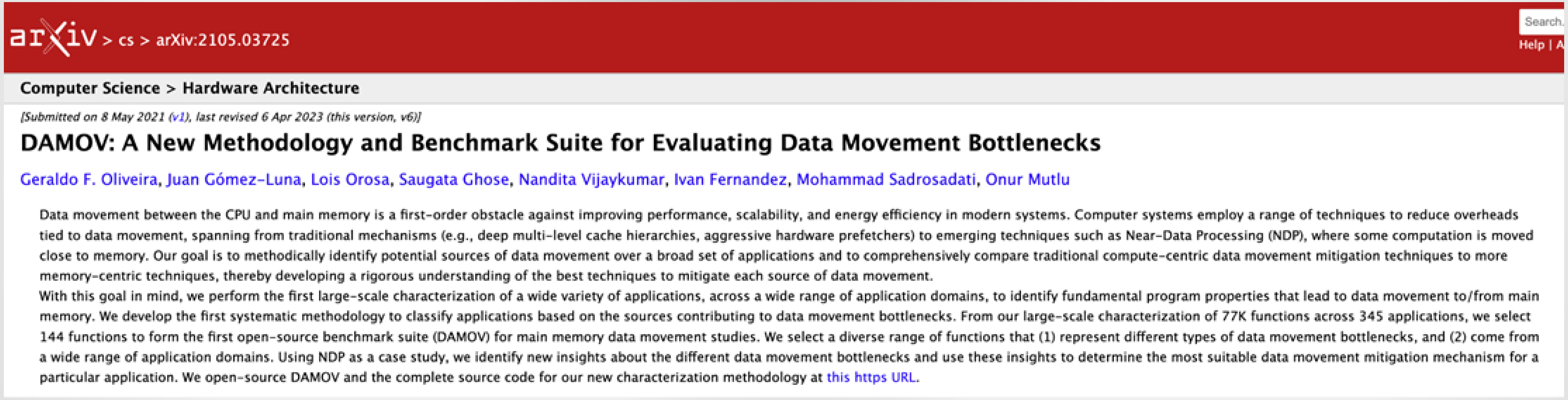
- Matrix multiplications (training/inference) are compute-bound if data fits in cache. **But is never does!**
- When data is too large, they become memory-bound (performance drops).
- Activations, weights, and gradients are constantly moved between memory levels.
- The cost of training often scales with communication, not just computation.

Movement Costs?

- We are not just optimizing FLOPs → we are optimizing data movement.
- Understanding hierarchy = identifying where frameworks waste resources.

Movement Costs?

- We are not just optimizing FLOPs → we are optimizing data movement.
- Understanding hierarchy = identifying where frameworks waste resources.


arXiv > cs > arXiv:2105.03725

Search.

Help | A

Computer Science > Hardware Architecture

[Submitted on 8 May 2021 (v1), last revised 6 Apr 2023 (this version, v6)]

DAMOV: A New Methodology and Benchmark Suite for Evaluating Data Movement Bottlenecks

Geraldo F. Oliveira, Juan Gómez-Luna, Lois Orosa, Saugata Ghose, Nandita Vijaykumar, Ivan Fernandez, Mohammad Sadrosadati, Onur Mutlu

Data movement between the CPU and main memory is a first-order obstacle against improving performance, scalability, and energy efficiency in modern systems. Computer systems employ a range of techniques to reduce overheads tied to data movement, spanning from traditional mechanisms (e.g., deep multi-level cache hierarchies, aggressive hardware prefetchers) to emerging techniques such as Near-Data Processing (NDP), where some computation is moved close to memory. Our goal is to methodically identify potential sources of data movement over a broad set of applications and to comprehensively compare traditional compute-centric data movement mitigation techniques to more memory-centric techniques, thereby developing a rigorous understanding of the best techniques to mitigate each source of data movement.
With this goal in mind, we perform the first large-scale characterization of a wide variety of applications, across a wide range of application domains, to identify fundamental program properties that lead to data movement to/from main memory. We develop the first systematic methodology to classify applications based on the sources contributing to data movement bottlenecks. From our large-scale characterization of 77K functions across 345 applications, we select 144 functions to form the first open-source benchmark suite (DAMOV) for main memory data movement studies. We select a diverse range of functions that (1) represent different types of data movement bottlenecks, and (2) come from a wide range of application domains. Using NDP as a case study, we identify new insights about the different data movement bottlenecks and use these insights to determine the most suitable data movement mitigation mechanism for a particular application. We open-source DAMOV and the complete source code for our new characterization methodology at this https URL.

Roofline Plot

arXiv > cs > arXiv:2105.03725

Computer Science > Hardware Architecture

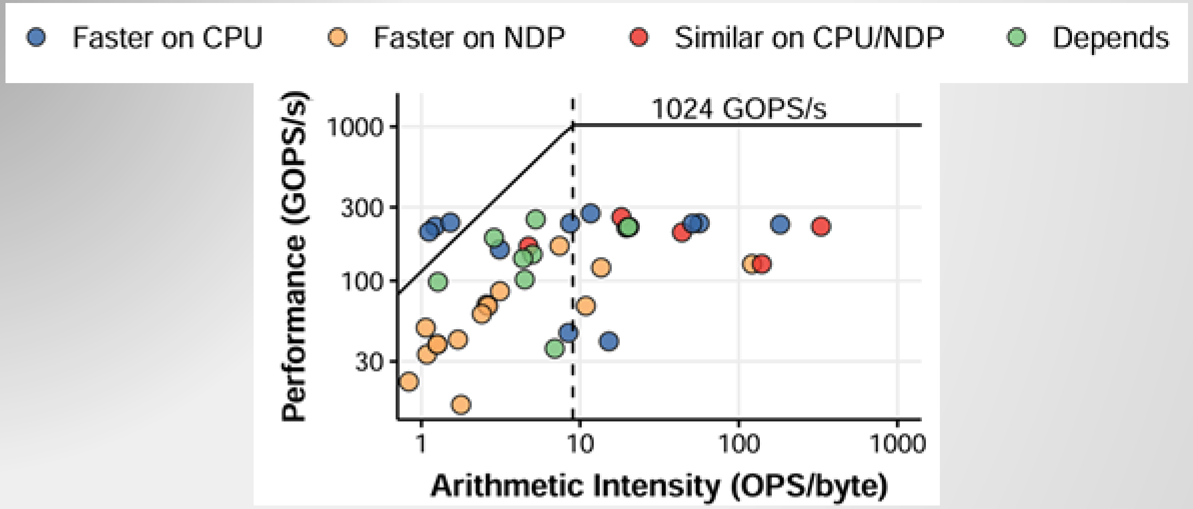
DAMOV: A New Methodology and Benchmark Suite for Evaluating Data Movement Bottlenecks

Geraldo F. Oliveira, Juan Gómez-Luna, Lois Orosa, Saugata Ghose, Nandita Vijaykumar, Ivan Fernandez, Mohammad Sadrosadati, Onur Mutlu

4SEE

- X-axis: Arithmetic Intensity (OPS/byte)
  - Represents the number of operations per byte of data transferred.
  - Indicates how computationally intensive an application is relative to its memory usage.

- Y-axis: Performance (GOPS/s)
  - Denotes the number of billions of operations per second.
- Reflects the computational throughput of the application.



Roofline Plot

arXiv > cs > arXiv:2105.03725

Computer Science > Hardware Architecture

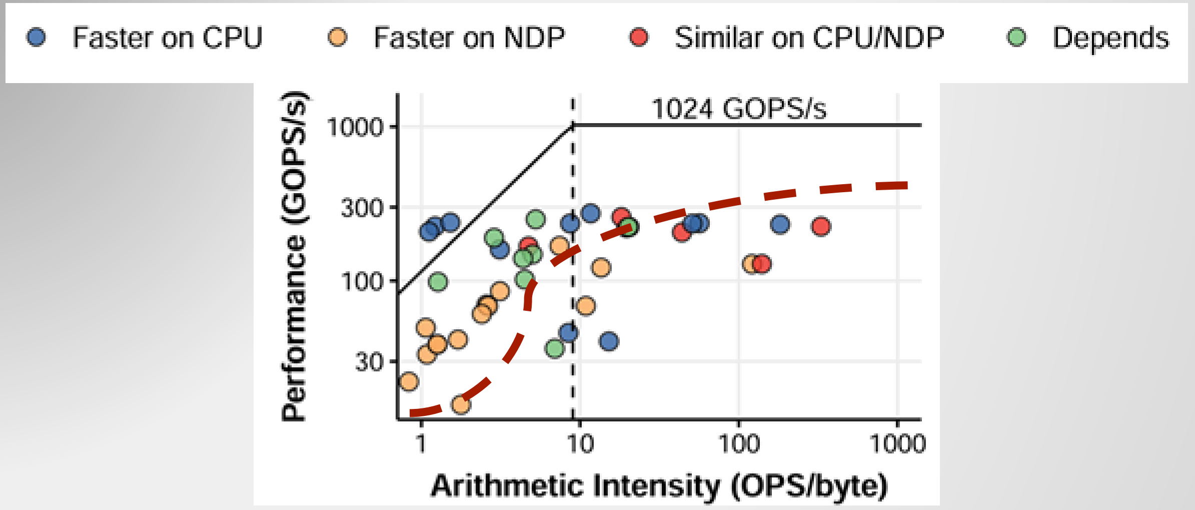
[Submitted on 8 May 2021 (v1), last revised 6 Apr 2023 (this version, v6)]

DAMOV: A New Methodology and Benchmark Suite for Evaluating Data Movement Bottlenecks

Geraldo F. Oliveira, Juan Gómez-Luna, Lois Orosa, Saugata Ghose, Nandita Vijaykumar, Ivan Fernandez, Mohammad Sadrosadati, Onur Mutlu

- X-axis: Arithmetic Intensity (OPS/byte)
  - Represents the number of operations per byte of data transferred.
  - Indicates how computationally intensive an application is relative to its memory usage.

- Y-axis: Performance (GOPS/s)
  - Denotes the number of billions of operations per second.
- Reflects the computational throughput of the application.



Next: Part II.b

- PII.a : Existing Solution: CPU, GPU, TPU, FPGA, ASIC basics
- PII.b : Memory hierarchy, bandwidth bottlenecks, movement costs
- PII.c : Precision

EURO 4SEE

Thanks!

Co-funded by
the European Union

EuroHPC
Joint Undertaking

This project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 101191697. The JU receives support from the Digital Europe Programme and Germany, Türkiye, Republic of North Macedonia, Montenegro, Serbia, Bosnia and Herzegovina.