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

P&S Processing-in-Memory
07.04.2022

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

SAFARI

ETH Zürich  UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN  UNIVERSITY OF TORONTO  UNIVERSITAS CACTI  UNIVERSIDAD DE MÁLAGA
Problem: Data movement is a major bottleneck in modern systems. However, it is unclear how to identify:
- different sources of data movement bottlenecks
- the most suitable mitigation technique (e.g., caching, prefetching, near-data processing) for a given data movement bottleneck

Goals:
1. Design a methodology to identify sources of data movement bottlenecks
2. Compare compute- and memory-centric data movement mitigation techniques

Key Approach: Perform a large-scale application characterization to identify key metrics that reveal the sources to data movement bottlenecks

Key Contributions:
- Experimental characterization of 77K functions across 345 applications
- A methodology to characterize applications based on data movement bottlenecks and their relation with different data movement mitigation techniques
- DAMOV: a benchmark suite with 144 functions for data movement studies
- Four case-studies to highlight DAMOV’s applicability to open research problems

Executive Summary
DAMOV: https://github.com/CMU-SAFARI/DAMOV
1. Data Movement Bottlenecks
2. Methodology Overview
3. Application Profiling
4. Locality-Based Clustering
5. Memory Bottleneck Analysis
6. Case Studies
# Outline

1. Data Movement Bottlenecks
2. Methodology Overview
3. Application Profiling
4. Locality-Based Clustering
5. Memory Bottleneck Analysis
6. Case Studies
Data movement bottlenecks happen because of:

- Not enough data **locality** → ineffective use of the cache hierarchy
- Not enough **memory bandwidth**
- High average **memory access time**
Data Movement Bottlenecks (2/2)

Compute-Centric Architecture

- Abundant DRAM bandwidth
- Shorter average memory access time

Near-Data Processing (NDP)

SAFARI
The goal of Near-Data Processing (NDP) is to mitigate data movement.

- Abundant DRAM bandwidth
- Shorter average memory access time
The goal of Near-Data Processing (NDP) is to mitigate data movement.

**UPMEM (2019)**

- Near-DRAM-banks processing for general-purpose computing
- 0.9 TOPS compute throughput\(^1\)

**Samsung FIMDRAM (2021)**

- Near-DRAM-banks processing for neural networks
- 1.2 TFLOPS compute throughput\(^2\)

---

When to Employ Near-Data Processing?

Near-Data Processing

Graph processing
(Tesseract$^1$)

Mobile consumer workloads
(GoogleWL$^2$)

Neural networks
(GoogleWL$^2$)

DNA sequence mapping
(GenASM$^3$; GRIM-Filter$^4$)

Databases
(Polynesia$^5$)

Time series analysis
(NATSA$^6$)

...
Identifying Memory Bottlenecks

- **Multiple approaches** to identify applications that:
  - suffer from data movement bottlenecks
  - take advantage of NDP

- **Existing approaches are not comprehensive enough**

---

**Roofline model**

- Performance (GOPS/s) vs. Arithmetic Intensity (OPS/byte)

**High LLC MPKI**

- NDP Speedup over CPU vs. Last-Level Cache MPKI
Limitations of Prior Approaches (1/2)

- **Roofline model** → identifies when an application is *bounded* by **compute** or **memory** units

![Diagram showing Roofline model]

- **Memory Roof**
  \[ y = BW \times AI \]

- **Compute Roof**
  \[ y = \text{Peak System Throughput} \]

- **Compute Bound** → Not suitable for NDP
- **Memory Bound** → Suitable for NDP

---

SAFARI
• **Roofline model** → identifies when an application is *bounded* by **compute** or **memory** units
Limitations of Prior Approaches (1/2)

- **Roofline model** → identifies when an application is *bounded* by **compute** or **memory** units

![Diagram](image)

- Memory Bound applications are **faster on CPU**, or performance depends ✗
- Compute Bound applications are **faster on CPU** ✓
- Compute Bound applications have **similar performance on CPU/NDP** or performance depends ✗

SAFARI
Limitations of Prior Approaches (1/2)

- **Roofline model** → identifies when an application is bounded by **compute** or **memory** units.

- Roofline model **does not accurately account** for the **NDP suitability** of memory-bound applications.

  - **Memory Bound** applications are **faster on NDP**.
  - **Compute Bound** applications have **similar performance** on CPU/NDP or performance **depends**.
Limitations of Prior Approaches (2/2)

• Application with a last-level cache $\text{MPKI} > 10$ → memory intensive and benefits from NDP

![Graph showing NDP Speedup over CPU vs LLC MPKI]
Limitations of Prior Approaches (2/2)

- Application with a last-level cache MPKI > 10 → memory intensive and benefits from NDP

Applications with low MPKI can be faster on NDP; have similar performance on CPU/NDP or; performance can depend.

Applications with low MPKI are faster on CPU.

Applications with high MPKI are faster on NDP.

SAFARI
Limitations of Prior Approaches (2/2)

- Application with a last-level cache MPKI > 10 → memory intensive and benefits from NDP

Applications with low MPKI can be faster on NDP;

LLC MPKI does not accurately account for the NDP suitability of memory-bound applications
Identifying Memory Bottlenecks

• Multiple approaches to identify applications that:
  - suffer from data movement bottlenecks
  - take advantage of NDP

• Existing approaches are not comprehensive enough
The Problem

- Multiple approaches to identify applications that:
  - suffer from data movement bottlenecks
  - take advantage of NDP

No available methodology can comprehensively:

- **identify** data movement bottlenecks
- **correlate** them with the **most suitable** data movement mitigation mechanism
Our Goal

- **Our Goal:** develop a methodology to:
  - methodically identify sources of data movement bottlenecks
  - comprehensively compare compute- and memory-centric data movement mitigation techniques
Outline

1. Data Movement Bottlenecks
2. Methodology Overview
3. Application Profiling
4. Locality-Based Clustering
5. Memory Bottleneck Analysis
6. Case Studies
Key Approach

• New **workload characterization methodology** to analyze:
  - data movement bottlenecks
  - suitability of different data movement mitigation mechanisms

• Two main profiling strategies:

  **Architecture-independent profiling:**
  characterizes the memory behavior **independently** of the underlying **hardware**

  **Architecture-dependent profiling:**
  evaluates the **impact of the system configuration** on the memory behavior
Methodology Overview

User Input
- Target Application
- Source Code

Step 1
Application Profiling

Profiler
roi_begin, roi_end

DAMOV-SIM Simulator

Memory Traces
Scalability Analysis

# Cores

Step 2
Locality-based Clustering

Step 3
Memory Bottleneck Class.

Methodology Output
- Memory Bottleneck Classes

SAFARI
Methodology Overview

**Step 1: Application Profiling**
- **Profiler**
  - roi_begin
  - roi_end

**User Input**
- Source Code
- Target Application

**Methodology Output**
- Memory Bottleneck Classes
- Profiler

**DAMOV-SIM Simulator**
- Memory Traces
- Scalability Analysis
- # Cores

**Step 2: Locality-based Clustering**
- High
- Low

**Step 3: Memory Bottleneck Class**
- High
- Low
Step 1: Application Profiling

Goal: Identify application functions that suffer from data movement bottlenecks

Hardware Profiling Tool: Intel VTune

MemoryBound: CPU is stalled due to load/store
Methodology Overview

**Step 1: Application Profiling**

- User Input: Target Application
- Source Code
- Profiler

**Memory Traces**

- roi_begin
- roi_end

**Step 2: Locality-based Clustering**

- **Spatial Locality**
- **Temporal Locality**

**Step 3: Memory Bottleneck Classifcation**

- High Temp. Locality
- Low Temp. Locality

**DAMOV-SIM Simulator**

- ld 0xFF
- st 0xAF
- ld 0xFF
- st 0xAF
- ld 0xFF
Step 2: Locality-Based Clustering

- **Goal:** analyze application’s memory characteristics

**Spatial Locality**

Memory Trace

| 0 | 1 | 2 | 3 | 4 | 5 |

Stride profile \((1)++=1\)

**Low spatial locality**

**High spatial locality**

---

Step 2: Locality-Based Clustering

- **Goal:** analyze application’s memory characteristics

**Spatial Locality**

- Memory Trace: 0 1 2 3 4 5
  - Stride profile(1) += 1

**Temporal Locality**

- Memory Trace: A A A A A
  - Reuse profile(4) += 1

Methodology Overview

**User Input**
- Target Application
- Source Code

**Step 1**
Application Profiling
- Profiled Region (roi_begin, roi_end)

**Profiler**

**Methodology Output**
- Memory Bottleneck Classes
- LFMR
- High
- Low

**Step 2**
Locality-based Clustering
- High
- Low

**Step 3**
Memory Bottleneck Class
- Arithmetic Intensity
- LLC MPKI
- Last-to-First Miss Ratio (LFMR)

**DAMOV-SIM Simulator**
- Memory Traces
- Scalability Analysis

**Safari**
Step 3: Memory Bottleneck Classification (1/2)

**Arithmetic Intensity (AI)**
- floating-point/arithmetic operations per L1 cache lines accessed
  → shows computational intensity per memory request

**LLC Misses-per-Kilo-Instructions (MPKI)**
- LLC misses per one thousand instructions
  → shows memory intensity

**Last-to-First Miss Ratio (LFMR)**
- LLC misses per L1 misses
  → shows if an application benefits from L2/L3 caches
Step 3: Memory Bottleneck Classification (2/2)

• **Goal:** identify the specific sources of data movement bottlenecks

- DAMOV-SIM Simulator
  - Scalability Analysis
  - Integrated ZSim and Ramulator

- **Scalability Analysis:**
  - 1, 4, 16, 64, and 256 out-of-order/in-order host and NDP CPU cores
  - 3D-stacked memory as main memory

DAMOV-SIM: [https://github.com/CMU-SAFARI/DAMOV](https://github.com/CMU-SAFARI/DAMOV)
Outline

1. Data Movement Bottlenecks
2. Methodology Overview
3. Application Profiling
4. Locality-Based Clustering
5. Memory Bottleneck Analysis
6. Case Studies
Step 1: Application Profiling

- We analyze **345 applications** from distinct domains:
  - Graph Processing
  - Deep Neural Networks
  - Physics
  - High-Performance Computing
  - Genomics
  - Machine Learning
  - Databases
  - Data Reorganization
  - Image Processing
  - Map-Reduce
  - Benchmarking
  - Linear Algebra
  ...

SAFARI
Memory Bound Functions

- We analyze 345 applications from distinct domains
- Selection criteria: clock cycles $> 3\%$ and Memory Bound $> 30\%$

We find 144 functions from a total of 77K functions and select:
- 44 functions → apply steps 2 and 3
- 100 functions → validation
1. Data Movement Bottlenecks
2. Methodology Overview
3. Application Profiling
4. Locality-Based Clustering
5. Memory Bottleneck Analysis
6. Case Studies
Step 2: Locality-Based Clustering

We use K-means to cluster the applications across both spatial and temporal locality, forming two groups:

1. Low locality applications (in orange)
2. High locality applications (in blue)
Step 2: Locality-Based Clustering

We use K-means to cluster the applications across both spatial and temporal locality, forming two groups:

1. Low locality applications (in orange)
2. High locality applications (in blue)

The closer a function is to the **bottom-left corner** → less likely it is to **take advantage** of a deep cache hierarchy.
Outline

1. Data Movement Bottlenecks
2. Methodology Overview
3. Application Profiling
4. Locality-Based Clustering
5. Memory Bottleneck Analysis
6. Case Studies
Step 3: Memory Bottleneck Analysis

Memory Bottleneck Class

- High Temporal Locality
  - High LFMR
    - High MPKI
      - Low AI
      - Low Memory Bottleneck Class
    - Low MPKI
      - Low AI
      - Low Memory Bottleneck Class
  - Low LFMR
    - Decreasing MPKI
      - Low AI
      - Low Memory Bottleneck Class
    - Increasing MPKI
      - Low AI
      - Low Memory Bottleneck Class

SAFARI
Step 3: Memory Bottleneck Analysis

Memory Bottleneck Class

1a: DRAM Bandwidth
1b: DRAM Latency
1c: L1/L2 Cache Capacity

2a: L3 Cache Contention
2b: L1 Cache Capacity
2c: Compute-Bound

Temporal Locality

High
Low

LFMR

Decreasing
Increasing

MPKI

AI

High
Low

Low
High

Low
Low

SAFARI
Step 3: Memory Bottleneck Analysis

Six classes of data movement bottlenecks:

- each class ↔ data movement mitigation mechanism

Memory Bottleneck Class

1a: DRAM Bandwidth
1b: DRAM Latency
1c: L1/L2 Cache Capacity
2a: L3 Cache Contention
2b: L1 Cache Capacity
2c: Compute-Bound
Step 3: Memory Bottleneck Analysis

Temporal Locality

- LFMR
- MPKI
- AI

Memory Bottleneck Class

1a: DRAM Bandwidth
- Low
- High
- Low

1b: DRAM Latency
- Low
- High
- Low

1c: L1/L2 Cache Capacity
- Low
- Low
- Low

2a: L3 Cache Contention
- Low
- High
- Low

2b: L1 Cache Capacity
- Low
- Low
- Low

2c: Compute-Bound
- High
- Low
- Low

SAFARI
Class 1a: DRAM Bandwidth Bound (1/2)

- High MPKI $\rightarrow$ high memory pressure
- Host scales well until bandwidth saturates
- NDP scales **without saturating** alongside attained bandwidth

**DRAM bandwidth bound applications:**
NDP does better because of the higher internal DRAM bandwidth
Class 1a: DRAM Bandwidth Bound (2/2)

- High LFMR → L2 and L3 caches are inefficient
- Host’s energy consumption is dominated by cache look-ups and off-chip data transfers
- NDP provides **large system energy reduction** since it does not access L2, L3, and off-chip links

**DRAM bandwidth bound applications:**
NDP does better because it eliminates off-chip I/O traffic
Step 3: Memory Bottleneck Analysis

**Temporal Locality**

- LFMR: Low
  - MPKI: Low
    - Increasing
      - LFMR: High
      - MPKI: Low
      - AI: Low
      - 2b: L1 Cache Capacity
  - AI: Low
    - 2c: Compute-Bound

- LFMR: High
  - MPKI: Low
    - Decreasing
      - LFMR: Low
      - MPKI: Low
      - AI: Low
      - 1b: DRAM Latency
  - AI: Low
    - 1c: L1/L2 Cache Capacity
Class 1b: DRAM Latency Bound

- High LFMR → L2 and L3 caches are inefficient
- Host scales well but NDP performance is always higher
- NDP performs better than host because of its lower memory access latency

**DRAM latency bound applications:**
host performance is hurt by the cache hierarchy and off-chip link
Step 3: Memory Bottleneck Analysis

Memory Bottleneck Class

1a: DRAM Bandwidth

1b: DRAM Latency

1c: L1/L2 Cache Capacity

2a: L3 Cache Contention

2b: L1 Cache Capacity

2c: Compute-Bound
Class 1c: L1/L2 Cache Capacity

- Decreasing LFMR \(\rightarrow\) L2/L3 caches turn efficient
- NDP scales better than the host at low core counts
- Host scales better than NDP at high core counts
- Host performs better than NDP at high core counts since it reduces memory access latency via data caching

**L1/L2 cache capacity bottlenecked applications:**
NDP is higher performance when the aggregated cache size is small
Step 3: Memory Bottleneck Analysis

**Memory Bottleneck Class**

1a: DRAM Bandwidth
1b: DRAM Latency
1c: L1/L2 Cache Capacity

2a: L3 Cache Contention
2b: L1 Cache Capacity
2c: Compute-Bound
Class 2a: L3 Cache Contention

- Increasing LFMR $\rightarrow$ L2/L3 caches turn inefficient
- Host scales better than the NDP at low core counts
- NDP scales better than host at high core counts
- NDP performs better than host at high core counts since it reduces memory access latency

**L3 cache contention bottlenecked applications:** at high core counts, applications turn into DRAM latency-bound
Step 3: Memory Bottleneck Analysis

Temporal Locality

LFMR

High

MPKI

High

MPKI

Low

AI

Low

1a: DRAM Bandwidth

1b: DRAM Latency

1c: L1/L2 Cache Capacity

Low

LFMR

Decreasing

MPKI

Low

AI

Low

2a: L3 Cache Contention

2b: L1 Cache Capacity

Low

2c: Compute-Bound

High

SAFARI
Class 2b: L1 Cache Capacity

- Low LFMR, MPKI; high temporal locality → efficient L2/L3 caches, low memory intensity
- Low AI → few operations per byte
- Host and NDP performance are similar → L1 dominates average memory access time

L1 cache capacity bottlenecked applications:
NDP can be used to reduce the host overall SRAM area
Step 3: Memory Bottleneck Analysis

Memory Bottleneck Class

1a: DRAM Bandwidth
1b: DRAM Latency
1c: L1/L2 Cache Capacity
2a: L3 Cache Contention
2b: L1 Cache Capacity
2c: Compute-Bound

Temporal Locality

LFMR

MPKI

AI
Class 2c: Compute-Bound

- Low LFMR, MPKI; high temporal locality → efficient L2/L3 caches, low memory intensity

- High AI → many operations per byte

- Host performs better than NDP because computation dominates execution time

Compute-bound applications: benefit highly from cache hierarchy; NDP is not a good fit
Step 3: Memory Bottleneck Analysis

**Temporal Locality**

- **LFMR**
  - High: Low → Decreasing
  - Low: Low → Increasing

- **MPKI**
  - High: Low → Decreasing
  - Low: Low → Increasing

- **AI**
  - High: Low → Decreasing
  - Low: Low → Increasing

**Memory Bottleneck Class**

1a: **DRAM Bandwidth**

1b: **DRAM Latency**

1c: **L1/L2 Cache Capacity**

2a: **L3 Cache Contention**

2b: **L1 Cache Capacity**

2c: **Compute-Bound**
Step 3: Memory Bottleneck Analysis

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

GERALDO F. OLIVEIRA¹, JUAN GÓMEZ-LUNA¹, LOIS OROSA¹, SAUGATA GHOSE², NANDITA VIJAYKUMAR³, IVAN FERNANDEZ¹,⁴, MOHAMMAD SADROSADATI¹, and ONUR MUTLU¹

¹ETH Zürich, Switzerland
²University of Illinois Urbana-Champaign, USA
³University of Toronto, Canada
⁴University of Malaga, Spain

Corresponding author: Geraldo F. Oliveira (e-mail: geraldod@inf.ethz.ch).
Methodology Validation

• **Goal**: evaluate the **accuracy** of our workload characterization methodically on a large set of functions

• Two-phase validation:

  **Phase 1:** calculate thresholds \((T)\)
  
  - Temporal Locality
  - Arithmetic Intensity
  - LLC MPKI
  - Last-to-First Miss Ratio
  
  Calculate **44 functions**

  **Phase 2:** calculate accuracy
  
  - \(T_{\text{Temporal Locality}}\)
  - \(T_{\text{Arithmetic Intensity}}\)
  - \(T_{\text{LLC MPKI}}\)
  - \(T_{\text{Last-to-First Miss Ratio}}\)
  
  Classify **100 functions**

  **High accuracy:**
  
  our methodology accurately classifies 97% of functions into one of the six memory bottleneck classes
More in the Paper

• Effect of the last-level cache size
  - Large L3 cache size (e.g., 512 MB) can mitigate some cache contention issues

• Summary of our workload characterization methodology
  - Including workload characterization using in-order host/NDP cores

• Limitations of our methodology

• Benchmark diversity
More in the Paper

- Effect of the last-level cache size
  - Large L3 cache size (e.g., 512 MB) can mitigate some cache

Summary of our workload characterization methodology
  - Including workload characterization using in-order host/NDP

Limitations of our methodology

Benchmark diversity
Outline

1. Data Movement Bottlenecks
2. Methodology Overview
3. Application Profiling
4. Locality-Based Clustering
5. Memory Bottleneck Analysis
6. Case Studies
Case Studies

• Many **open questions related to NDP system designs**\(^8\):
  - Interconnects
  - Data mapping and allocation
  - NDP core design (accelerators, general-purpose cores)
  - Offloading granularity
  - Programmability
  - Coherence
  - System integration
  - ...

• **Goal:** demonstrate how DAMOV is useful to study NDP system designs

---

Case Studies

Load Balance and Inter-Vault Communication on NDP

NDP Accelerators and Our Methodology

Different Core Models on NDP Architectures

Fine-Grained NDP Offloading
Load Balance and Inter-Vault Communication on NDP

portion of the memory requests an NDP core issues go to remote vaults → increases the memory access latency for the NDP core

NDP Accelerators and Our Methodology

NDP accelerator is faster than compute-centric accelerator for Class 1a and 1b applications; slower for Class 2c

→ key observations hold for other NDP architectures

Different Core Models on NDP Architectures

using in-order cores limits performance of some applications → static instruction scheduling cannot exploit memory parallelism

Fine-Grained NDP Offloading

few basic blocks are responsible for most LLC misses → offloading such basic blocks to NDP are enough to improve performance
Load Balance and Inter-Vault Communication on NDP

NDP Accelerators and Our Methodology
NDP accelerator is faster than compute-centric accelerator for Class 1a and 1b applications; slower for Class 2c
→ key observations hold for other NDP architectures

Different Core Models on NDP Architectures

Fine-Grained NDP Offloading
Case Studies (3/4)

Load Balance and Inter-Vault Communication on NDP

NDP Accelerators and Our Methodology

Different Core Models on NDP Architectures

- using in-order cores limits performance of some applications
- static instruction scheduling cannot exploit memory parallelism

Fine-Grained NDP Offloading
Load Balance and Inter-Vault Communication on NDP

NDP Accelerators and Our Methodology

Different Core Models on NDP Architectures

Fine-Grained NDP Offloading

- Few basic blocks are responsible for most of LLC misses
- Offloading such basic blocks to NDP are enough to improve performance
Case Studies

Load Balance and Inter-Vault Communication on NDP

A portion of the memory requests an NDP core issues go to remote vaults → increases the memory access latency for the NDP core.

NDP Accelerators and Our Methodology

NDP accelerator is faster than compute-centric accelerator for Class 1a and 1b applications; slower for Class 2c → key observations hold for other NDP architectures.

Different Core Models on NDP Architectures

Using in-order cores limits performance of some applications → static instruction scheduling cannot exploit memory parallelism.

Fine-Grained NDP Offloading

Few basic blocks are responsible for most of LLC misses → offloading such basic blocks to NDP are enough to improve performance.
Case Studies

Load Balance and Inter-Vault Communication on NDP

NDP Accelerators and Our Methodology

NDP accelerator is faster than compute-centric accelerator for Class 1a and 1b applications; slower for Class 2c

→ key observations hold for other NDP architectures

Different Core Models on NDP Architectures

Fine-Grained NDP Offloading
NDP Accelerators and Our Methodology

**Goal:** evaluate compute-centric versus NDP accelerators

![Diagram of compute-centric and NDP accelerators with custom accelerators and DRAM](image)

**Graph: Speedup Over Compute-Centric Accelerator**

- 1a: DRK Yolo
- 1b: PLYalu
- 2c: PLY3mm

NDP Accelerators and Our Methodology

• Goal: evaluate compute-centric versus NDP accelerators

The performance of NDP accelerators are in line with the characteristics of the memory bottleneck classes:

our memory bottleneck classification can be applied to study other types of system configurations

Case Studies

Load Balance and Inter-Vault Communication on NDP

portion of the memory requests an NDP core issues go to remote vaults
→ increases the memory access latency for the NDP core

NDP Accelerators and Our Methodology

NDP accelerator is faster than compute-centric accelerator for Class 1a and 1b applications; slower for Class 2c
→ key observations hold for other NDP architectures

Different Core Models on NDP Architectures

using in-order cores limits performance of some applications
→ static instruction scheduling cannot exploit memory parallelism

Fine-Grained NDP Offloading

few basic blocks are responsible for most of LLC misses
→ offloading such basic blocks to NDP are enough to improve performance
Case Studies

Load Balance and Inter-Vault Communication on NDP

portion of the memory requests an NDP core issues go to remote vaults

→ increases the memory access latency for the NDP core

NDP Accelerators and Our Methodology

NDP accelerator is faster than compute-centric accelerator for Class 1a and 1b applications; slower for Class 2c

→ key observations hold for other NDP architectures

Different Core Models on NDP Architectures

using inorder cores limits performance of some applications

→ static instruction scheduling cannot exploit memory parallelism

Fine-Grained NDP Offloading

few basic blocks are responsible for most of LLC misses

→ offloading such basic blocks to NDP are enough to improve performance

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

GERALDO F. OLIVEIRA¹, JUAN GÓMEZ-LUNA¹, LOIS OROSA¹, SAUGATA GHOSE², NANDITA VIJAYKUMAR³, IVAN FERNANDEZ¹,⁴, MOHAMMAD SADROSADATI¹, and ONUR MUTLU¹

¹ETH Zürich, Switzerland
²University of Illinois Urbana-Champaign, USA
³University of Toronto, Canada
⁴University of Malaga, Spain

Corresponding author: Geraldo F. Oliveira (e-mail: geraldod@inf.ethz.ch).
DAMOV is Open-Source

- We open-source our benchmark suite and our toolchain

DAMOV-SIM

DAMOV Benchmark

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

DAMOV is a benchmark suite and a methodical framework targeting the study of data movement bottlenecks in modern applications. It is intended to study new architectures, such as near-data processing.

The DAMOV benchmark suite is the first open-source benchmark suite for main memory data movement-related studies, based on our systematic characterization methodology. This suite consists of 144 functions representing different sources of data movement bottlenecks and can be used as a baseline benchmark set for future data-movement mitigation research. The applications in the DAMOV benchmark suite belong to popular benchmark suites, including BWA, Chai, Darknet, GASE, Hardware Effects, Hashjoin, HPCC, HPCG, Ligra, PARSEC, Parboil, PolyBench, Phoenix, Rodinia, SPLASH-2, STREAM.

SAFARI
DAMOV is Open-Source

We open-source our benchmark suite and our toolchain.

Get DAMOV at: https://github.com/CMU-SAFARI/DAMOV

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

DAMOV is a benchmark suite and a methodical framework targeting the study of data movement bottlenecks in modern applications. It is intended to study new architectures, such as near-data processing.

The DAMOV benchmark suite is the first open-source benchmark suite for main memory data movement-related studies, based on our systematic characterization methodology. This suite consists of 144 functions representing different sources of data movement bottlenecks and can be used as a baseline benchmark set for future data-movement mitigation research. The applications in the DAMOV benchmark suite belong to popular benchmark suites, including BWA, Chai, Darknet, GASE, Hardware Effects, HashJoin, HPCC, HPCG, Ligra, PARSEC, Parboil, PolyBench, Phoenix, Rodinia, SPLASH-2, STREAM.
• **Problem:** Data movement is a major bottleneck in modern systems. However, it is **unclear** how to identify:
  - different sources of data movement bottlenecks
  - the most suitable mitigation technique (e.g., caching, prefetching, near-data processing) for a given data movement bottleneck

• **Goals:**
  1. Design a methodology to **identify** sources of data movement bottlenecks
  2. Compare compute- and memory-centric data movement mitigation techniques

• **Key Approach:** Perform a large-scale application characterization to identify **key metrics** that reveal the sources to data movement bottlenecks

• **Key Contributions:**
  - Experimental characterization of 77K functions across 345 applications
  - A **methodology** to characterize applications based on data movement bottlenecks and their relation with different data movement mitigation techniques
  - **DAMOV:** a **benchmark suite** with 144 functions for data movement studies
  - Four case-studies to highlight DAMOV’s applicability to open research problems

**SAFARI**

DAMOV: [https://github.com/CMU-SAFARI/DAMOV](https://github.com/CMU-SAFARI/DAMOV)
DAMOV: A New Methodology and Benchmark Suite for Evaluating Data Movement Bottlenecks

P&S Processing-in-Memory
07.04.2022

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

SAFARI

ETH Zürich  I  UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN  UNIVERSITY OF TORONTO  UNIVERSITAS CATTAMONACI  UNIVERSIDAD DE MÁLAGA