The Locality Descriptor
A Holistic Cross-Layer Abstraction to Express Data Locality in GPUs

Nandita Vijaykumar
Eiman Ebrahimi, Kevin Hsieh, Phillip B. Gibbons, Onur Mutlu

Carnegie Mellon University  NVIDIA  ETH Zürich
Executive Summary

Exploiting data locality in GPUs is a challenging task

Performance Speedups:
26.6% (up to 46.6%) from cache locality
53.7% (up to 2.8x) from NUMA locality

Hardware

- Data Placement
- Cache Management
- CTA Scheduling
- Data Prefetching
- …
Outline

Why leveraging data locality is challenging?

Designing the Locality Descriptor

Evaluation
Data locality is critical to GPU performance

Two forms of data locality:

Reuse-based locality (cache locality)

NUMA locality

Reuse-based (Cache Locality)
Data locality is critical to GPU performance

Two forms of data locality:

Reuse-based locality (cache locality)

NUMA locality
The GPU execution and programming models are designed to explicitly express parallelism...

But there is no explicit way to express data locality

Exploiting data locality in GPUs is a challenging and elusive feat
A case study in leveraging data locality: **Histo**

![Diagram of data structure and CTA compute grid with CTAs along the Y dim sharing the same data and type of data locality: Inter-CTA]
Leveraging cache locality

CTA scheduling is required to leverage inter-CTA cache locality

CTA scheduling is insufficient: we also need other techniques
Leveraging **NUMA locality**

Exploiting NUMA locality requires both CTA scheduling and data placement
Today, leveraging data locality is challenging

As a programmer:
- No easy access to architectural techniques – CTA scheduling, cache management, data placement, etc.
- Even when using work-arounds, optimization is tedious and not portable

As the architect:
- Key program semantics are not available to the hardware

Where to place data?
Which CTAs to schedule together?
To make things worse:

There are many different locality types: Inter-CTA, inter-warp, intra-thread, ...

Each type requires a different set of architectural techniques:
- **Inter-CTA locality** requires CTA scheduling + prefetching
- **Intra-thread** locality requires cache management
- ...
The Locality Descriptor

A hardware-software abstraction to express and exploit data locality

Connects locality semantics to the underlying hardware techniques

Application

Software

Locality Descriptor

Hardware

New software interface

Data Placement

Cache Management

CTA Scheduling

Data Prefetching

Access to key program semantics

...
Goals in designing the Locality Descriptor

1) Supplemental and hint-based

2) Architecture-agnostic interface

3) Flexible and general

Inter-CTA, inter-warp, intra-thread, …

Software

Hardware

Locality Descriptor

Application

Data Placement
Cache Management
CTA Scheduling
Data Prefetching

…
LocalityDescriptor ldesc(X; Y, Z);

Designing the Locality Descriptor
An Overview: The components of the Locality Descriptor

LocalityDescriptor \texttt{ldesc}(A, \texttt{len}, \texttt{INTER-THREAD}, \texttt{tile}, \texttt{loc});

1) Data Structure

2) Locality Type

3) Tile Semantics

4) Locality Semantics
Outline

Why leveraging data locality is challenging?

Designing the Locality Descriptor

Evaluation
1. How to choose the basis of the abstraction?

*Key Idea: Use the data structure as the basis to describe data locality*

- Architecture-agnostic
- Each data structure is accessed the same way by all threads

A new instance is required for each important data structure

```
LocalityDescriptor ldesc(A);
```

*Data Structure*
2. How to communicate with hardware?

Locality type drives architecture mechanisms

Inter-CTA, inter-warp, intra-thread, ...

Architecture-agnostic interface

Software

Locality Descriptor

Hardware

Data Placement

Cache Management

CTA Scheduling

Data Prefetching

...
2. How to communicate with hardware?

Key Idea: Use locality type to drive underlying architectural techniques

Origin of locality (or locality type) causes the challenges in exploiting it

E.g.:
- Inter-CTA locality requires CTA scheduling as reuse is across threads
- Intra-thread locality requires cache management to avoid thrashing

Locality type is application-specific and known to the programmer
2. How to communicate with hardware?

Key Idea: Use **locality type** to drive underlying architectural techniques

Three fundamental types:
- INTER-THREAD
- INTRA-THREAD
- NO-REUSE

LocalityDescriptor ldesc(A); INTER-THREAD);
Driving underlying architectural techniques

Locality Type?

- INTER_THREAD
- NO_REUSE
- INTRA_THREAD

Determined based on more information

Cache Bypassing
CTA Scheduling (if NUMA)
Memory Placement (if NUMA)

CTA Scheduling
Cache Soft Pinning
Memory Placement (if NUMA)
3. How to describe locality?

Key Idea: Partition the data structure and compute grid into tiles

LocalityDescriptor ldesc(A, INTER-THREAD); tile);

tile((X_tile, Y_len, 1),
    (1, GridSize.y, 1),
    (1, 0, 0));
Additional features of the Locality Descriptor

Locality type insufficient to inform underlying architectural techniques

(INTER-THREAD, INTRA-THREAD, NO-REUSE)

In addition, we also have Locality Semantics to include:

- Sharing Type
- Access Pattern

(COACCESED, REGULAR, X_len)

LocalityDescriptor ldesc(A, INTER-THREAD, tile); loc);
A decision tree to drive underlying techniques

**Locality Type?**

- **NEARBY**
  - **Sharing Type?**
    - **COACCESSSED**
      - **Access Pattern?**
        - **REGULAR**
          - CTA Scheduling
            - Guided Stride Prefetching
            - Memory Placement (if NUMA)
        - **IRREGULAR**
          - CTA Scheduling
            - Cache Soft Pinning
            - Memory Placement (if NUMA)
      - CTA Scheduling
        - Next-line Stride Prefetching
        - Memory Placement (if NUMA)
  - INTRA_THREAD
    - **Access Pattern?**
      - **REGULAR**
        - Cache Bypassing
          - CTA Scheduling (if NUMA)
          - Memory Placement (if NUMA)
      - **IRREGULAR**
        - Cache Hard Pinning
          - CTA Scheduling (if NUMA)
          - Memory Placement (if NUMA)

**INTER_THREAD**
Leveraging the Locality Descriptor

LocalityDescriptor ldesc(A, INTER-THREAD, tile, loc);

**Architectural techniques:**
1) CTA Scheduling
2) Prefetching
3) Data Placement
CTA Scheduling

Data Structure A

CTA Compute Grid
Leveraging the Locality Descriptor

LocalityDescriptor ldesc(A, INTER-THREAD, tile, loc);

Architectural techniques:
1) CTA Scheduling
2) Prefetching
3) Data Placement
Data Placement

Data Structure A

Cluster A
Cluster 1
Cluster 2
Cluster 3

Cluster Queues

Cluster 0
Cluster 1
Cluster 2
Cluster 3

SM 0
SM 1
SM 2
SM 3

L1D
L1D
L1D
L1D

Memory
Memory
Memory
Memory

NUMA Zone 0
NUMA Zone 1
NUMA Zone 2
NUMA Zone 3

CTA Compute Grid
Leveraging the Locality Descriptor

LocalityDescriptor $l_{desc}(A, \text{INTER-THREAD}, \text{tile}, \text{loc})$.

All threads stall because they wait on the same data.

Architectural techniques:
1) CTA Scheduling
2) Prefetching
3) Data Placement

Data Structure $A$

CTA Compute Grid
Outline

1. Why leveraging data locality is challenging?
2. The Locality Descriptor
3. Evaluation
Methodology

Evaluation Infrastructure: GPGPUSim v3.2.2
Workloads: Parboil, Rodinia, CUDA SDK, Polybench
System Parameters:

- **Shader Core**: 1.4 GHz; GTO scheduler [50]; 2 schedulers per SM, Round-robin CTA scheduler
- **SM Resources Registers**: 32768; Scratchpad: 48KB, L1: 32KB, 4 ways
- **Memory Model**: FR-FCFS scheduling [59, 60], 16 banks/channel
- **Single Chip System**: 15 SMs; 6 memory channels; L2: 768KB, 16 ways
- **Multi-Chip System**: 4 NUMA zones, 64 SMs (16 per zone); 32 memory channels; L2: 4MB, 16 ways; Inter-GPM Interconnect: 192 GB/s;
- **DRAM Bandwidth**: 768 GB/s (192 GB/s per module)
Locality descriptors are an effective means to leverage cache locality.

Different locality types require different optimizations. A single optimization is often insufficient.
Performance Impact: Leveraging NUMA Locality

- Baseline
- FirstTouch-Distrib
- LDesc-Placement
- LDesc

53.7% (up to 2.8x)
Conclusion

Problem:
GPU programming models have no explicit abstraction to express data locality
Leveraging data locality is a challenging task, as a result

Our Proposal: The Locality Descriptor
A HW-SW abstraction to explicitly express data locality
A architecture-agnostic and flexible SW interface to express data locality
Enables HW to leverage key program semantics to optimize locality

Key Results:
26.6% (up to 46.6%) performance speed up from leveraging cache locality
53.7% (up to 2.8x) performance speed up from leveraging NUMA locality
The Locality Descriptor
A Holistic Cross-Layer Abstraction to Express Data Locality in GPUs

Nandita Vijaykumar
Eiman Ebrahimi, Kevin Hsieh, Phillip B. Gibbons, Onur Mutlu

Carnegie Mellon University
NVIDIA
ETH Zürich