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
Executive Summary

Exploiting data locality in GPUs is a challenging task

Flexible, architecture-agnostic interface

Access to program semantics

Software

Hardware

Locality Descriptor

Application

Data Placement  Cache Management  CTA Scheduling  Data Prefetching  …
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
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
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

Data Structure A

CTA Compute Grid

CTAs
A case study in leveraging data locality: Histo
A case study in leveraging data locality: Histo

CTAs

CTAs along the Y dim share the same data

Type of data locality: Inter-CTA

Data Structure A

CTA Compute Grid
Leveraging cache locality

Data Structure A

CTA Compute Grid

Core

L1

Core

L1

Core

L1

CTA scheduling is required to leverage inter-CTA cache locality
CRA scheduling is required to leverage inter-CTA cache locality

CRA 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
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

New software interface

Access to key program semantics

Software

Locality Descriptor

Hardware

Data Placement

Cache Management

CTA Scheduling

Data Prefetching

...
Goals in designing the Locality Descriptor

1) Supplemental and hint-based

2) Architecture-agnostic interface

Application

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

3) Flexible and general

Locality Descriptor

Software

Hardware

Data Placement
Cache Management
CTA Scheduling
Data Prefetching
...

18
Designing the Locality Descriptor

LocalityDescriptor ldsc();
Designing the Locality Descriptor

LocalityDescriptor ldesc(X, Y, Z);
Designing the Locality Descriptor

LocalityDescriptor ldesc(X, Y, Z);

[Diagram showing relationships between Hardware, Data Placement, Cache Management, CTA Scheduling, Data Prefetching, and an ellipsis]
An Overview: The components of the Locality Descriptor

LocalityDescriptor  ldesc(A, len, INTER-THREAD, tile, 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?

- Architecture-agnostic interface
- Inter-CTA, inter-warp, intra-thread, ...
- Architecture-specific optimizations

**Application**

**Locality Descriptor**

- Data Placement
- Cache Management
- CTA Scheduling
- Data Prefetching
- ...

Software

Hardware
2. How to communicate with hardware?

Locality type drives architecture mechanisms

Architecture-agnostic interface

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

Architecture-specific optimizations

Application

Locality Descriptor

Software

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

Basic unit of locality:
1) Data Tile
2) Compute Tile
3) Mapping between them

```python
tile((X_tile, Y_len, 1),
     (1, GridSize.y, 1),
     (1, 0, 0));
```
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

(COACCESSSED, REGULAR, X_len)

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

Locality Type?

- INTER_THREAD
- INTRA_THREAD
- NO_REUSE

Sharing Type?

- NEARBY

Access Pattern?

- REGULAR
- IRREGULAR

CTA Scheduling
Next-line Stride Prefetching
Memory Placement (if NUMA)

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

Cache Hard Pinning
CTA Scheduling (if NUMA)
Memory Placement (if NUMA)

CTA Scheduling
Guided Stride Prefetching
Memory Placement (if NUMA)

CTA Scheduling
Cache Soft Pinning
Memory Placement (if NUMA)
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

Cluster 0
Cluster 1
Cluster 2
Cluster 3

Core 0
Core 1
Core 2
Core 3

L1D
L1D
L1D
L1D
Leveraging the Locality Descriptor

```
LocalityDescriptor ld_{desc}(A, INTER-THREAD, tile, loc);
```

Architectural techniques:
1) CTA Scheduling
2) Prefetching
3) Data Placement
Leveraging the Locality Descriptor

LocalityDescriptor \( ldesc(A, \text{INTER-THREAD}, \text{tile}, \text{loc}) \)

All threads stall because they wait on the same data

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

Why leveraging data locality is challenging?

The Locality Descriptor

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)
Performance speedup: leveraging cache locality

26.6% (up to 46.6%)

Baseline  LDesc-Sched  LDesc-Pref  LDesc-Cache  LDesc

<table>
<thead>
<tr>
<th>SK</th>
<th>DT</th>
<th>HS</th>
<th>D2D</th>
<th>C2D</th>
<th>SPMV</th>
<th>LIB</th>
<th>LMD</th>
<th>Geomean</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTER-THREAD (COACCESSSED)</td>
<td>INTER-THREAD (NEARBY)</td>
<td>INTRA-THREAD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Locality descriptors are an effective means to leverage cache locality.
Performance speedup: leveraging cache locality

Baseline  LDesc-Sched  LDesc-Pref  LDesc-Cache  LDesc

Speedup

$\text{SK}$  $\text{DT}$  $\text{HS}$  $\text{D2D}$  $\text{C2D}$  $\text{SPMV}$  $\text{LIB}$  $\text{LMD}$  $\text{Geomean}$

INTER-THREAD  (COACCESSSED)  INTER-THREAD  (NEARBY)  INTRA-THREAD

26.6% (up to 46.6%)
Performance speedup: leveraging cache locality

- **Baseline**
- **LDesc-Sched**
- **LDesc-Pref**
- **LDesc-Cache**
- **LDesc**

The diagram shows the speedup for different benchmarks and thread types:

- **SK**
- **DT**
- **HS**
- **D2D**
- **C2D**
- **SPMV**
- **LIB**
- **LMD**

Each benchmark is divided into three categories:

- **INTER-THREAD (COACCESSSED)**
- **INTER-THREAD (NEARBY)**
- **INTRA-THREAD**

The overall geometric mean speedup is 26.6% (up to 46.6%).
Performance speedup: leveraging cache locality

- **Baseline**
- **LDesc-Sched**
- **LDesc-Pref**
- **LDesc-Cache**
- **LDesc**

<table>
<thead>
<tr>
<th></th>
<th>SK</th>
<th>DT</th>
<th>HS</th>
<th>D2D</th>
<th>C2D</th>
<th>SPMV</th>
<th>LIB</th>
<th>LMD</th>
<th>Geomean</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speedup</td>
<td>1.5</td>
<td>1.2</td>
<td>1.4</td>
<td>1.3</td>
<td>1.2</td>
<td>1.1</td>
<td>1.1</td>
<td>1.2</td>
<td>1.3</td>
</tr>
</tbody>
</table>

- **INTER-THREAD (COACCESSSED)**
- **INTER-THREAD (NEARBY)**
- **INTRA-THREAD**
- **Geomean**
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