Morpheus
Extending the Last Level Cache Capacity in GPU Systems with Idle GPU Core Resources

Sina Darabi, Mohammad Sadrosadati, Negar Akbarzadeh, Joël Lindegger, Mohammad Hosseini, Jisung Park, Juan Gómez-Luna, Onur Mutlu, Hamid Sarbazi-Azad

ETH zürich, SAFARI Research Group, IPM

October 3rd 2022
MICRO-55
Motivation & Goal

Memory-bound GPU applications leave some cores under-utilized

Our Goal:
Leverage the under-utilized cores to boost the performance and energy efficiency of memory-bound applications
Our Work

Morpheus

First technique that leverages some GPU cores’ private memories to extend the total GPU last-level cache (LLC) capacity.

Morpheus employs (1) a software helper kernel in cores and (2) a new hardware unit in the LLC partitions.

Morpheus significantly improves performance and energy efficiency of memory-bound GPU applications and enables a 4x larger LLC with the same LLC hardware.
Memory-Bound GPU Applications

![Graph showing Speedup vs. #GPU Cores](image-url)
Memory-Bound GPU Applications

- Good performance benefit up to 40 cores
- Little to no performance benefit from 40 to 68 cores
Memory-Bound GPU Applications

Conventional GPUs do not benefit (significantly) from all cores for memory-bound applications
**Our Goal**

Leverage the under-utilized cores to boost the performance and energy efficiency of memory-bound applications.
Outline

1 Introduction

2 Morpheus: Overview

3 Morpheus Controller

4 Extended LLC Kernel

5 Evaluation

6 Conclusion
Key Idea

On-Chip Memory of Conventional GPU

Well-Utilized Cores’ Private Memories

Under-Utilized Cores’ Private Memories

Conventional LLC

On-Chip Memory of Morpheus-enabled GPU

Well-Utilized Cores’ Private Memories

Under-Utilized Cores’ Private Memories

Extended LLC

Morpheus

Morpheus’ Key Idea:

Repurpose under-utilized GPU cores’ private memory to extend the GPU’s LLC capacity
GPU Core Execution Modes in Morpheus

- **Compute Mode**
  - Operates Conventionally
  - Application Kernel
  - Application Data
  - Execution Units
  - L1 Cache
  - Shared Memory
  - Register File

- **Cache Mode**
  - Lends Private Memory to Extend the LLC
  - Extended LLC Kernel
  - Controller State
  - Tag Array
  - Data Array
Serving an LLC Request in Morpheus

Application Kernel Generates LLC Request

Core operating in Compute Mode

Interconnection Network

Extended LLC Kernel Serves Fraction of LLC Requests

Core operating in Cache Mode

Interconnection Network

New Hardware Unit Makes Forwarding Decision

Existing Hardware Unit Serves Fraction of LLC Requests

Morpheus Controller

Conventional LLC

LLC Partition

Application Kernel Generates LLC Request

Core operating in Compute Mode

Interconnection Network

Extended LLC Kernel Serves Fraction of LLC Requests

Core operating in Cache Mode

Interconnection Network

New Hardware Unit Makes Forwarding Decision

Existing Hardware Unit Serves Fraction of LLC Requests

Morpheus Controller

Conventional LLC

LLC Partition
Outline

1. Introduction
2. Morpheus: Overview
3. Morpheus Controller
4. Extended LLC Kernel
5. Evaluation
6. Conclusion
Morpheus Controller

1. Forwarding Decision
2. Query Conventional LLC
3. Query Extended LLC
Morpheus Controller

1. Forwarding Decision
   *Address Separation, Combinational Logic*

2. Query Conventional LLC

3. Query Extended LLC
Morpheus Controller

1. Forwarding Decision
   Address Separation, Combinational Logic

2. Query Conventional LLC
   Physically Co-Located, Simple Forwarding

3. Query Extended LLC
Morpheus Controller

1. Forwarding Decision
   Address Separation, Combinational Logic

2. Query Conventional LLC
   Physically Co-Located, Simple Forwarding

3. Query Extended LLC
   Warp Status Table
Morpheus Controller Warp Status Table

Core operating in Compute Mode

Core operating in Cache Mode

Warp Status Table in the Morpheus Controller

• Tracks ongoing requests per set
  • E.g., read/write, tag, hit/miss

LLC Requests

Regular Loads/Stores

SAFARI
Outline

1 Introduction
2 Morpheus: Overview
3 Morpheus Controller
4 Extended LLC Kernel
5 Evaluation
6 Conclusion
Extended LLC Kernel

GPU Core

1. Query L1 Cache
   Load/Store Instructions

2. Query Shared Memory
   Software Engineering

3. Query Register File
   Data Layout across Threads for Parallelism

SAFARI
Extended LLC via Register File

Extended LLC Set

Thread #0 holds Tag #0

Block #0

Thread #1 holds Tag #1

Block #1

Thread #2 holds Tag #2

Block #2

Thread #31 holds Tag #31

Block #31
Extended LLC via Register File

Extended LLC Set

- Thread #0 holds Tag #0
- Thread #1 holds Tag #1
- Thread #2 holds Tag #2
- Thread #31 holds Tag #31

- Block #0
- Block #1
- Block #2
- Block #31

Query Tag

Compare

32 Threads Operate in Parallel
Extended LLC via Register File

32 Threads Operate in Parallel

Many Warps (e.g., 48) Operate in Parallel
More in the Paper: Mechanisms

- Detailed mechanisms
  - Morpheus Controller
  - Extended LLC kernel

- Optimization techniques
  - Hit/miss prediction
  - Indirect-MOV
  - Cache compression applied to Morpheus

More in the Paper: Mechanisms
Morpheus on arXiv
Outline

1. Introduction
2. Morpheus: Overview
3. Morpheus Controller
4. Extended LLC Kernel
5. Evaluation
6. Conclusion
### Methodology

- **17 representative applications**
  - 14 memory-bound
  - 3 compute-bound

- **Experimentally characterize the extended LLC**
  - On a real NVIDIA RTX 3080, as the extended LLC kernel requires no special hardware
  - Capacity, latency, bandwidth, energy/byte of the extended LLC

- **Simulate a Morpheus-enabled GPU using AccelSim** [Khairy+ ISCA'20]
  - Performance, energy efficiency, bandwidth utilization
Morpheus Performance

Morpheus performs **better** than or **equal** to the baseline for **all** applications.
**Morpheus Performance**

Morpheus performs **better** than or **equal** to the baseline for **all** applications.

Morpheus **accelerates memory-bound** applications by **39%** on average.

Morpheus **performs within 3%** of a **4x larger** conventional LLC.

**SAFARI**
Morpheus uses less or the same energy as the baseline for all applications
Morpheus Energy Efficiency

Morpheus uses **less** or the **same** energy as the baseline for all applications.

Morpheus **improves** energy efficiency by **58%** on average for **memory-bound** applications.

Morpheus’ energy efficiency is **within 6%** of a **4x larger** conventional LLC.
More in the Paper: Evaluation

Morpheus: Extending the Last Level Cache Capacity in GPU Systems Using Idle GPU Core Resources

Sina Darabi† Mohammad Sadrosadati§ Negar Akbarzadeh† Joël Lindegger§ Mohammad Hosseini‡ Jisung Park§†† Juan Gómez-Luna§ Onur Mutlu§ Hamid Sarbazi-Azad†‡

†Sharif University of Technology  §ETH Zürich
‡Institute for Research in Fundamental Sciences (IPM)  ††POSTECH
More in the Paper: Evaluation

- Characterization of extended LLC on a real GPU
- Performance and energy comparisons with more baselines
- Total LLC throughput with Morpheus
- On-chip and off-chip memory bandwidth utilization
- Storage and power overhead analysis
## Summary

**Motivation**
Memory-bound GPU applications leave some cores under-utilized

**Goal**
Leverage the under-utilized cores to boost the performance and energy efficiency of memory-bound applications

**Morpheus**
First technique that leverages some GPU cores’ private memories to extend the total GPU last-level cache capacity

**Key Results**
Morpheus outperforms state-of-the-art GPU architectures for memory-bound applications

- **39% performance** improvement (relative to RTX 3080)
- **58% energy efficiency** improvement (relative to RTX 3080)
- Morpheus performs within 3% of a 4x larger conventional LLC
- Morpheus provides a 4x larger LLC with the same LLC hardware

**Conclusion**
Morpheus effectively repurposes the private memory of under-utilized cores to extend the GPU LLC capacity
Morpheus
Extending the Last Level Cache Capacity in GPU Systems with Idle GPU Core Resources

Sina Darabi, Mohammad Sadrosadati, Negar Akbarzadeh, Joël Lindegger, Mohammad Hosseini, Jisung Park, Juan Gómez-Luna, Onur Mutlu, Hamid Sarbazi-Azad

October 3rd 2022
MICRO-55
Backup Slides

- Performance Benefits of a Larger LLC
- Memory-bound vs. Compute-bound Applications
- Morpheus’ Logical Data Flow
- LLC Timelines
- Hit/Miss Predictor
- Extended LLC Query Logic Unit
- Registerfile Layout
- Cache Compression
- Indirect-MOV Algorithm
- Indirect-MOV Instruction
- Extended LLC Characterization
- Simulation Parameters
- Number of Compute Mode Cores
- Morpheus LLC Capacity
- Morpheus Energy and Power
- Hit/Miss Prediction Benefits
- Evaluated Applications
- Extended LLC Tag Lookup Algorithm
Performance Benefits of a Larger LLC

The graph shows the normalized IPC (Instructions Per Cycle) for different workloads with different LLC sizes: 1X-LLC, 2X-LLC, and 4X-LLC. The workloads include p-bfs, cfd, dwt2d, stencil, r-bfs, bprob, sgem, nw, page-r, kmeans, histo, mri-gri, spmv, lbm, and gmean. The normalized IPC values range from 1 to 2, indicating the performance benefits of a larger LLC compared to the baseline.
Memory-bound vs. Compute-bound Applications

Memory-bound

- p-bfs
- stencil
- sgem
- kmeans

Compute-bound

- lib
- hotsp
- lbm
- mri-q

Normalized IPC vs. Number of SMs
Morpheus’ Logical Data Flow

- **Morpheus Controller**
  - Address Separation

- **Conventional LLC**
  - Forward

- **Extended LLC Controller**
  - Address Separation
  - Forward

- **New Components**
  - Query

- **Repurposed Components**
  - Forward

- **L1 Cache**
  - Query

- **Shared Memory**
  - Query

- **Register File**
Hit/Miss Predictor

Request for Extended LLC Address

In BF1?
Yes → Query Extended LLC
No → Access DRAM

Hit?
Yes → Cache Block Re-Used
No → Send Response from Morpheus Controller to Requesting Core

Bloom Filter Management

Insert Cache Block into Extended LLC Set

Cache Block Re-Used

Insert into BF1
Insert into BF2

\( n := \text{num\_elems\_in(BF2)} \)

\( n \geq \text{associativity} \)

No → Clear BF1, BF2
Yes → Done with Request’s Updates

Done with Request’s Updates

Send Response from Morpheus Controller to Requesting Core
Extended LLC Query Logic Unit

Request from Core in Compute Mode

Request Queue

Warp Status Table

<table>
<thead>
<tr>
<th>Set ID</th>
<th>Tag</th>
<th>Requesting SM</th>
<th>Busy</th>
<th>Op</th>
<th>Result</th>
<th>Data Pointer</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0xB3...</td>
<td>42</td>
<td>1</td>
<td>Write</td>
<td>-</td>
<td>15</td>
</tr>
<tr>
<td>1</td>
<td>0xD7...</td>
<td>3</td>
<td>0</td>
<td>Read</td>
<td>Hit</td>
<td>0</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>255</td>
<td>0x5F...</td>
<td>12</td>
<td>0</td>
<td>Read</td>
<td>Miss</td>
<td>-</td>
</tr>
</tbody>
</table>

128 Bytes

Write Data Buffer

Read Data Buffer
Registerfile Layout

Extended LLC’s Logical Layout

- **Set 0**
  - Word 0
  - **0**: 32 LLC Blocks
  - Metadata Block M0
  - Metadata Block M31
  - LRU Counter
  - D V Block Tag
  - 4 bytes
  - 12 bits
  - 2 bits
  - 18 bits

Extended LLC’s Layout in Register File

- **Warp 0**
  - **Thread 0**
  - **R0**: 32 Data-Array Registers
  - **R31**: 1 Metadata Register
  - **R32**: 9 Auxiliary Registers
  - **R33**: Metadata Block M31
  - **R41**: 4 bytes
Cache Compression

Extended LLC’s Logical Layout

Set 0 | Word 0 | ... | Word 31
---|---|---|---
Block 0
Block 1
Block 2
Block 3

4 bytes

Extended LLC’s Layout across Registers with Compression

<table>
<thead>
<tr>
<th>Warp 0</th>
<th>Thread 0</th>
<th>...</th>
<th>Thread 31</th>
</tr>
</thead>
<tbody>
<tr>
<td>R0</td>
<td>Low Compression Level</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>High Compression Level</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Uncompressed</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1 byte

4 bytes
Indirect-MOV Algorithm

Algorithm 2 Indirect-MOV Algorithm

Input: BLOCK_INDEX:int \(R_{aux3}\)
Output: Requested Extended LLC Block \(R_{aux0}\)

1: procedure \textsc{Indirect-MOV} //The goal is to implement register indirect access, reading from a register whose index is determined by accessing the value in another register. This procedure is critical for accessing data-array registers in the extended LLC kernel.

2: \(T_{list} : \text{Branch Targets } L_0, L_1, L_2, \ldots, L_{31};\) //Define 32 branch targets, each is allocated to access a specific register index.

3: @p brx.idx \(R_{aux3}, T_{list};\) //Branch to label \(L_i\) specified by the target LLC block index \(i=R_{aux3}\)

4: \(L_0:\)
5: \text{MOV } R_0, R_{aux0} //Access data-array register \(R_0\) if target LLC block index is 0
6: return

7: \(L_1:\)
8: \text{MOV } R_1, R_{aux0} //Access data-array register \(R_1\) if target LLC block index is 1
9: return

10: ...
11: \(L_{31}:\)
12: \text{MOV } R_{31}, R_{aux0} //Access data-array register \(R_{31}\) if target LLC block index is 31
13: return
14: end procedure
Indirect-MOV Instruction
Extended LLC Characterization

(a) Capacity (KiB) vs. Number of Warps
(b) Latency (ns) vs. Number of Warps
(c) Bandwidth (GB/s) vs. Number of Warps
(d) Energy/Byte (J/Byte) vs. Number of Warps
# Simulation Parameters

- **NVIDIA RTX 3080**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of SMs</td>
<td>68</td>
</tr>
<tr>
<td>Scheduler</td>
<td>Two-Level [18, 38]</td>
</tr>
<tr>
<td>GPU Memory Interface</td>
<td>320-bit GDDR6X [39]</td>
</tr>
<tr>
<td>GPU Memory Capacity</td>
<td>10 GiB</td>
</tr>
<tr>
<td>Conventional LLC Capacity</td>
<td>5 MiB</td>
</tr>
<tr>
<td>L1/Shared-Memory Capacity</td>
<td>128 KiB per SM</td>
</tr>
<tr>
<td>Register File Capacity</td>
<td>256 KB per SM</td>
</tr>
</tbody>
</table>
## Number of Compute Mode Cores

Table 3: Number of GPU cores executing application threads for different evaluated systems (#available GPU cores is 68).

<table>
<thead>
<tr>
<th>Application</th>
<th>p-bfs</th>
<th>cfd</th>
<th>dwt2d</th>
<th>stencil</th>
<th>r-bfs</th>
<th>bprob</th>
<th>sgem</th>
<th>nw</th>
<th>page-r</th>
<th>kmeans</th>
<th>histo</th>
<th>mri-gri</th>
<th>spmv</th>
<th>ibm</th>
<th>lib</th>
<th>hotsp</th>
<th>mri-q</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBL</td>
<td>68</td>
<td>68</td>
<td>68</td>
<td>68</td>
<td>68</td>
<td>68</td>
<td>68</td>
<td>68</td>
<td>24</td>
<td>53</td>
<td>34</td>
<td>42</td>
<td>34</td>
<td>68</td>
<td>68</td>
<td>68</td>
<td>68</td>
</tr>
<tr>
<td>Morpheus-Basic</td>
<td>32</td>
<td>42</td>
<td>42</td>
<td>50</td>
<td>34</td>
<td>39</td>
<td>48</td>
<td>18</td>
<td>42</td>
<td>37</td>
<td>47</td>
<td>36</td>
<td>44</td>
<td>32</td>
<td>68</td>
<td>68</td>
<td>68</td>
</tr>
<tr>
<td>Morpheus-ALL</td>
<td>40</td>
<td>55</td>
<td>54</td>
<td>56</td>
<td>37</td>
<td>41</td>
<td>54</td>
<td>26</td>
<td>46</td>
<td>47</td>
<td>52</td>
<td>43</td>
<td>47</td>
<td>36</td>
<td>68</td>
<td>68</td>
<td>68</td>
</tr>
</tbody>
</table>
Morpheus LLC Capacity (NVIDIA RTX 3080)

- **Conventional LLC:** 5 MiB

- **Extended LLC Per Core in Cache Mode:** 328 KiB
  - 128 KiB from L1
  - 256 KiB from Register File
    - 200 KiB usable to extended LLC data

- **Number of Cache Mode Cores:** Application Dependent
  - E.g. 50 for Needleman-Wunsch (nw) => 16 MiB
  - Theoretically up to 67 => 21.4 MiB
Morpheus Energy and Power

![Graph showing normalized execution time and normalized performance per watt for various benchmarks across different configurations.]
## Evaluated Applications

<table>
<thead>
<tr>
<th>Application</th>
<th>Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Breadth-First Search [6]</td>
<td>p-bfs</td>
<td>Memory-bound</td>
</tr>
<tr>
<td>Computational fluid dynamics [5]</td>
<td>cfd</td>
<td>Memory-bound</td>
</tr>
<tr>
<td>Discrete Wavelet Transform (2D) [5]</td>
<td>dwt2d</td>
<td>Memory-bound</td>
</tr>
<tr>
<td>Stencil [6]</td>
<td>stencil</td>
<td>Memory-bound</td>
</tr>
<tr>
<td>Breadth-First Search [5]</td>
<td>r-bfs</td>
<td>Memory-bound</td>
</tr>
<tr>
<td>Back Propagation [5]</td>
<td>bprob</td>
<td>Memory-bound</td>
</tr>
<tr>
<td>sgemm [6]</td>
<td>sgem</td>
<td>Memory-bound</td>
</tr>
<tr>
<td>Needleman-Wunsch [5]</td>
<td>nw</td>
<td>Memory-bound</td>
</tr>
<tr>
<td>Page Rank [40]</td>
<td>page-r</td>
<td>Memory-bound</td>
</tr>
<tr>
<td>Histogram [6]</td>
<td>histo</td>
<td>Memory-bound</td>
</tr>
<tr>
<td>Sparse-Matrix Dense-Vector Multiplication [6]</td>
<td>spmv</td>
<td>Memory-bound</td>
</tr>
<tr>
<td>Lattice-Boltzmann [6]</td>
<td>lbm</td>
<td>Memory-bound</td>
</tr>
<tr>
<td>LIBOR Monte Carlo [41]</td>
<td>lib</td>
<td>Compute-bound</td>
</tr>
<tr>
<td>HotSpot [5]</td>
<td>hotsp</td>
<td>Compute-bound</td>
</tr>
</tbody>
</table>
Algorithm 1 Extended LLC Tag Lookup – Register File

Input: Extended LLC Request’s Tag ($R_{aux0}$)
Output: HIT:bool, and if HIT=True, BLOCK_INDEX:int ($R_{aux3}$)

1: procedure TAG_LOOKUP //executed by an extended LLC kernel warp of 32 threads
2: \hspace{1em} $R_{aux1} \leftarrow \text{Valid}(R_M)$ //ensure the block is valid
3: \hspace{1em} $R_{aux1} \leftarrow R_{aux1} \&\& (R_{aux0} == \text{Tag}(R_M))$ //match request tag to metadata
4: \hspace{1em} $R_{aux2} \leftarrow \_\text{ballot\_sync}(0xffffffff, R_{aux1})$ //share $R_{aux1}$ between all threads as a 32-bit vector
5: \hspace{1em} if ($R_{aux2}$) then //one of the bits is non-zero because there was a hit
6: \hspace{2em} $R_{aux3} \leftarrow \_\text{ffs}(R_{aux2}) - 1$ //get the 0-based index of the non-zero bit
7: \hspace{2em} HIT \leftarrow True
8: \hspace{2em} BLOCK_INDEX \leftarrow R_{aux3}
9: \hspace{2em} if (thread_idx == $R_{aux3}$) then //reset the LRU counter of the hit block
10: \hspace{3em} LRU\_Counter(R_M) \leftarrow 0xff
11: \hspace{2em} else //decrement the LRU counters of all other blocks
12: \hspace{3em} LRU\_Counter(R_M) \leftarrow LRU\_Counter(R_M) - 1
13: \hspace{2em} end if
14: \hspace{2em} else
15: \hspace{3em} HIT \leftarrow False
16: \hspace{2em} end if
17: end procedure