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

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

December 23rd 2022
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.
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Introduction</td>
</tr>
<tr>
<td>2</td>
<td>Morpheus: Overview</td>
</tr>
<tr>
<td>3</td>
<td>Morpheus Controller</td>
</tr>
<tr>
<td>4</td>
<td>Extended LLC Kernel</td>
</tr>
<tr>
<td>5</td>
<td>Hit/Miss Prediction</td>
</tr>
<tr>
<td>6</td>
<td>Evaluation</td>
</tr>
<tr>
<td>7</td>
<td>Conclusion</td>
</tr>
</tbody>
</table>
Memory-Bound GPU Applications

![Graph showing speedup vs. number of GPU cores.](image)
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

Good performance benefit

Little to no performance benefit

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.
Morpheus’ Key Idea:
Repurpose under-utilized GPU cores’ private memory to extend the GPU’s LLC capacity.
Performance Benefits of a Larger LLC

Normalized IPC

1X-LLC  2X-LLC  4X-LLC

p-bfs  cfd  dwt2d  stencil  r-bfs  bprob  sgem  nw  page-r  kmeans  histo  mri-gri  spmv  ibm  gmean
Performance Benefits of a Larger LLC

More LLC generally improves performance
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
  - LLC Request
- Interconnection Network
  - New Hardware Unit Makes Forwarding Decision
  - Existing Hardware Unit Serves Fraction of LLC Requests
- Extended LLC Kernel Serves Fraction of LLC Requests
  - Core operating in Cache Mode
- LLC Partition
  - Morpheus Controller
    - Conventional LLC
Outline

1. Introduction
2. Morpheus: Overview
3. Morpheus Controller
4. Extended LLC Kernel
5. Hit/Miss Prediction
6. Evaluation
7. Conclusion
Morpheus Controller

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

Address Separation, Combinational Logic

1. Query Conventional LLC

2. 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
- Memory mapped

LLC Requests

Regular Loads/Stores
Outline

1 Introduction
2 Morpheus: Overview
3 Morpheus Controller
4 Extended LLC Kernel
5 Hit/Miss Prediction
6 Evaluation
7 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
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

- 32 Threads Operate in Parallel
- Query Tag
  - Compare
  - 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

32 Threads Operate in Parallel

Many Warps (e.g., 48) Operate in Parallel
LLC Timelines

Conventional LLC Hit

- Interconnection Network: 76ns
- Morpheus Controller (no hit/miss prediction): 0.7ns
- Morpheus Controller (with hit/miss prediction): 1.4ns
- Conventional LLC Access: 8ns
- Extended LLC Access: 21ns
- GDDR6X: 447ns

SM requests block 0ns to 773ns
LLC Timelines

Misses in the extended LLC are particularly expensive. Correctly predicting them can reduce the miss latency significantly.
Hit/Miss Predictor

Request for Extended LLC Address

- In BF1? Yes → Query Extended LLC
  - Hit? Yes → Cache Block Re-Used
  - Hit? No → Access DRAM → Send Response from Morpheus Controller to Requesting Core
- In BF1? No → Access DRAM

Request Handling

Bloom Filter Management

- Insert Cache Block into Extended LLC Set
  - Insert into BF1
  - Insert into BF2
  - $n := \text{num\_elems\_in}(BF2)$
    - $n \geq \text{associativity}$
      - No → Clear BF1
      - Yes → Swap BF1, BF2
  - Done with Request’s Updates
Is There A More Sophisticated Hit/Miss Predictor?
Hermes: Accelerating Long-Latency Load Requests via Perceptron-Based Off-Chip Load Prediction

Rahul Bera\textsuperscript{1} Konstantinos Kanellopoulos\textsuperscript{1} Shankar Balachandran\textsuperscript{2} David Novo\textsuperscript{3} Ataberk Olgun\textsuperscript{1} Mohammad Sadrosadati\textsuperscript{1} Onur Mutlu\textsuperscript{1}

\textsuperscript{1}ETH Zürich \textsuperscript{2}Intel Processor Architecture Research Lab \textsuperscript{3}LIRMM, Univ. Montpellier, CNRS

Long-latency load requests continue to limit the performance of modern high-performance processors. To increase the latency tolerance of a processor, architects have primarily relied on two key techniques: sophisticated data prefetchers and large on-chip caches. In this work, we show that: (1) even a sophisticated state-of-the-art prefetcher can only predict half of the off-chip load requests on average across a wide range of workloads, and (2) due to the increasing size and complexity of on-chip caches, a large fraction of the latency of an off-chip load request is spent accessing the on-chip cache hierarchy to solely determine that it needs to go off-chip.

The goal of this work is to accelerate off-chip load requests by removing the on-chip cache access latency from their critical path. To this end, we propose a new technique called Hermes, whose key idea is to: (1) accurately predict which load requests off-chip main memory (i.e., an off-chip load) often stalls the processor core by blocking the instruction retirement from the reorder buffer (ROB), thus limiting the core’s performance [88, 91, 92]. To increase the latency tolerance of a core, computer architects primarily rely on two key techniques. First, they employ increasingly sophisticated hardware prefetchers that can learn complex memory address patterns and fetch data required by future load requests before the core demands them [28, 32, 33, 35, 75]. Second, they significantly scale up the size of the on-chip cache hierarchy with each new generation of processors [10, 11, 16].

Key problem. Despite recent advances in processor core design, we observe two key trends in new processor designs that leave a significant opportunity for performance improvement on the table. First, even a sophisticated state-of-the-art


More in the Morpheus Paper: Mechanisms

- **Detailed mechanisms**
  - Morpheus Controller
  - Extended LLC kernel

- **Optimization techniques**
  - Indirect-MOV
  - Cache compression applied to Morpheus
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

NVIDIA RTX 3080
4x larger conventional LLC

Morpheus

Normalized Execution Time

Lower is Better

Memory-bound

Compute-bound

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.
Morpheus uses less or the same energy as the baseline for all applications.
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§ Joël Lindegger§ Negar Akbarzadeh† Mohammad Hosseini†
Jisung Park§.Down Juan Gómez-Luna§ Hamid Sarbazi-Azad†† Onur Mutlu§

†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
<table>
<thead>
<tr>
<th>Outline</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Introduction</td>
</tr>
<tr>
<td>2</td>
<td>Morpheus: Overview</td>
</tr>
<tr>
<td>3</td>
<td>Morpheus Controller</td>
</tr>
<tr>
<td>4</td>
<td>Extended LLC Kernel</td>
</tr>
<tr>
<td>5</td>
<td>Hit/Miss Prediction</td>
</tr>
<tr>
<td>6</td>
<td>Evaluation</td>
</tr>
<tr>
<td>7</td>
<td>Conclusion</td>
</tr>
</tbody>
</table>
### 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, Joël Lindegger, Negar Akbarzadeh, Mohammad Hosseini, Jisung Park, Juan Gómez-Luna, Hamid Sarbazi-Azad, Onur Mutlu

ETH Zürich
SAFARI Research Group
IPM

December 23rd 2022
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

Normalized IPC

1X-LLC 2X-LLC 4X-LLC

Normalized IPC
Morpheus’ Logical Data Flow

- **LLC Request**
  - **Morpheus Controller**
    - **Address Separation**
  - **Extended LLC Controller**
    - **Address Separation**
  - **Conventional LLC**
  - **New Components**
  - **Repurposed Components**
    - **L1 Cache**
    - **Shared Memory**
    - **Register File**
Hit/Miss Predictor

- **Request for Extended LLC Address**
  - In BF1?
    - Yes: Query Extended LLC
    - No: Access DRAM
      - Send Response from Morpheus Controller to Requesting Core

- **Request Handling**
  - Hit?
    - Yes: Cache Block Re-Used
    - No: In BF1?
      - Yes: Query Extended LLC
      - No: Access DRAM

- **Bloom Filter Management**
  - Insert Cache Block into Extended LLC Set
    - Insert into BF1
    - Insert into BF2
    - \( n := \text{num}_\text{elems}_\text{in}(BF2) \)
    - \( n \geq \text{associativity} \)
      - Yes: Clear BF1
      - No: Swap BF1, BF2

  - Done with Request’s Updates
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>0x3B...</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>
Registerfile Layout

Extended LLC’s Logical Layout

<table>
<thead>
<tr>
<th>Set 0</th>
<th>Word 0</th>
<th>⋮</th>
<th>Word 31</th>
<th>Metadata</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block 0</td>
<td>⋮</td>
<td>⋮</td>
<td>⋮</td>
<td>M0</td>
</tr>
<tr>
<td>Block 31</td>
<td>⋮</td>
<td>⋮</td>
<td>M31</td>
<td></td>
</tr>
</tbody>
</table>

48 LLC Sets

32 LLC Blocks

Metadata Block M31

LRU Counter 12 bits
DV 2 bits
Block Tag 18 bits

4 bytes

Extended LLC’s Layout in Register File

<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>⋮</td>
<td>⋮</td>
<td>⋮</td>
</tr>
<tr>
<td>⋮</td>
<td>32 Data-Array Registers</td>
<td>⋮</td>
<td>⋮</td>
</tr>
<tr>
<td>R31</td>
<td>⋮</td>
<td>1 Metadata Register</td>
<td>⋮</td>
</tr>
<tr>
<td>R32</td>
<td>⋮</td>
<td>9 Auxiliary Registers</td>
<td>⋮</td>
</tr>
<tr>
<td>R33</td>
<td>⋮</td>
<td>⋮</td>
<td>⋮</td>
</tr>
<tr>
<td>R41</td>
<td>⋮</td>
<td>⋮</td>
<td>⋮</td>
</tr>
</tbody>
</table>

48 Warps

4 bytes
Cache Compression

Extended LLC’s Logical Layout

<table>
<thead>
<tr>
<th>Set 0</th>
<th>Word 0</th>
<th>•••</th>
<th>Word 31</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Block 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Block 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Block 3</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R0</td>
<td></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>
</tbody>
</table>

High Compression Level

Low Compression Level

Uncompressed

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

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

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

<table>
<thead>
<tr>
<th>Warp: 2</th>
<th>Op: Indirect-MOV</th>
</tr>
</thead>
<tbody>
<tr>
<td>Valid: 1</td>
<td>Register: R33</td>
</tr>
<tr>
<td>Valid: 1</td>
<td>Register: *R33</td>
</tr>
<tr>
<td>Valid: 0</td>
<td>Register: -</td>
</tr>
</tbody>
</table>

Register File Bank

Warp ID

Src Reg ID Immediate

Indirect Src Reg ID

Indirect Src Ready

New Component
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>
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-gr</th>
<th>spmv</th>
<th>lbm</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></td>
<td></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
Hit/Miss Prediction Benefits

No-Prediction  Bloom-Filter  Perfect-Prediction

Norm. exec. time

p-bfs  cfd  dwt2d  stencil  r-bfs  bprob  sgem  nw  page-r  kmeans  histo  mri-gri  spmv  lbm  gmean
## 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: $R_{aux1} \leftarrow Valid(R_M)$ //ensure the block is valid  
3: $R_{aux1} \leftarrow R_{aux1} \&\& (R_{aux0} == Tag(R_M))$ //match request tag to metadata  
4: $R_{aux2} \leftarrow ballot_sync(0xffffffff, R_{aux1})$ //share $R_{aux1}$ between all threads as a 32-bit vector  
5: if ($R_{aux2}$) then //one of the bits is non-zero because there was a hit  
6: $R_{aux3} \leftarrow _ffs(R_{aux2}) - 1$ //get the 0-based index of the non-zero bit  
7: HIT $\leftarrow$ True  
8: BLOCK_INDEX $\leftarrow R_{aux3}$  
9: if (thread_idx $== R_{aux3}$) then //reset the LRU counter of the hit block  
10: $LRU\_Counter(R_M) \leftarrow 0xfff$  
11: else //decrement the LRU counters of all other blocks  
12: $LRU\_Counter(R_M) \leftarrow LRU\_Counter(R_M) - 1$  
13: end if  
14: else  
15: HIT $\leftarrow$ False  
16: end if  
17: end procedure