CTA-Aware Prefetching and Scheduling for GPU

Gunjae Koo*, Hyeran Jeon†, Zhenhong Liu‡, Nam Sung Kim‡, Murali Annavaram*

*University of Southern California
†San Jose State University
‡University of Illinois at Urbana-Champaign
Memory Latency – GPU Compute Hurdle

**GPU execution model**
- Can hide tens of cycles by quick context switching

---

Warp 0
Warp 1
Warp 2
Warp 3

```
MUL R2, R1, 8
ADD R4, R2, R3
LD R5, [R4+0]
SUB R6, R5, R1
```
Memory Latency – GPU Compute Hurdle

**GPU demand fetch latency**
• **Hundreds of cycles if fetched from LLC or DRAM**

![Diagram showing memory latency and GPU demand fetch latency]
GPU demand fetch latency

- Not hidden by warp throttling
- Critical performance hurdle: pipeline stalls for long cycles

Pipeline stalls by memory operations [Kim (HPCA'16)] 38% (average), 63% (memory intensive)
Prefetch: A Solution for Long Data Fetch Latency

Prefetch can be a solution for long data fetch latency in GPU

Prefetch performance factors

- Coverage
- Accuracy
- Timeliness
**Intra-warp stride prefetching**

- **Applied for iterative global loads in a loop**
- **Limited coverage**: loops are rare in GPU kernels
- **Bad timeliness**: early prefetch

```c
for (int i=0; i < n; i++)
    y[i] = a*x[i] + y[i];

int i = blockIdx.x * blockDim.x + threadIdx.x;
    y[i] = a*x[i] + y[i]
```
Limitations of GPU Prefetch

**Inter-warp prefetching**

- Regular strides observed among threads

```c
for (int i=0; i < n; i++)
    y[i] = a*x[i] + y[i];
```
Limitations of GPU Prefetch

Inter-warp prefetching

- Regular strides observed among threads
- **Low accuracy:** discrepancy across CTA boundaries
- **Bad timeliness:** base address estimation for each CTA
CTA-Aware Prefetcher and Scheduler (CAPS)

- CTA-aware prefetcher
  - Base address of each CTA
  - Common stride per load between warps
- Prefetch-aware warp scheduler
  - Reorganizing warp execution priority to detect required information for prefetcher
  - Warp-wakeup
```c
#define INDEX(i, j, j_off) (i + __mul24(j, j_off))

__shared__ float u1[3*KOFF];

i = threadIdx.x;
j = threadIdx.y;

i = INDEX(i, blockIdx.x, BLOCK_X);
j = INDEX(j, blockIdx.y, BLOCK_Y);
indg = INDEX(i, j, pitch);

active = (i<NX) && (j<NY);

if (active) u1[ind+KOFF] = d_u1[indg];

......
```
```c
#define INDEX(i, j, j_off)  (i + __mul24(i, j_off))

__shared__ float u1[3*KOFF];

i = threadIdx.x;
j = threadIdx.y;
i = INDEX(i, blockIdx.x, BLOCK_X);
j = INDEX(j, blockIdx.y, BLOCK_Y);
indg = INDEX(i, j, pitch);
active = (i < NX) && (j < NY);
if (active) u1[ind+KOFF] = d_u1[indg];
```

Indg = threadIdx.x + blockIdx.x * BLOCK_X + (threadIdx.y + blockIdx.y * BLOCK_Y) * pitch

\[
C_1 \leftarrow blockIdx.x \times BLOCK_X \\
C_2 \leftarrow blockIdx.y \times BLOCK_Y \\
C_3 \leftarrow pitch
\]

\[
i = threadIdx.x + C_1 + (threadIdx.y + C_2) \times C_3 \\
= threadIdx.x + threadIdx.y \times C_3 + (C_1 + C_2 \times C_3)
\]

Function of thread IDs

Function of CTA IDs
```c
#define INDEX(i, j, j_off) (i + __mul24(i, j_off))

__shared__ float u1[3*KOFF];

i = threadIdx.x;

j = threadIdx.y;

i = INDEX(i, blockIdx.x, BLOCK_X);

j = INDEX(j, blockIdx.y, BLOCK_Y);

indg = INDEX(i, j, pitch);

active = (i < NX) && (j < NY);

if (active) u1[ind + KOFF] = d_u1[indg];
```

*Indg = threadIdx.x + blockIdx.x * BLOCK_X + 
(threadIdx.y + blockIdx.y * BLOCK_Y) * pitch*

*\( C_1 \leftarrow blockIdx.x \times BLOCK_X \)

*\( C_2 \leftarrow blockIdx.y \times BLOCK_Y \)

*\( C_3 \leftarrow pitch \)

= threadIdx.x + \( C_1 \) + (threadIdx.y + \( C_2 \)) \times \( C_3 \)

= threadIdx.x + threadIdx.y \times \( C_3 \) + (\( C_1 \) + \( C_2 \) \times \( C_3 \))

= threadIdx.x + threadIdx.y \times \( C_3 \) + \( \Theta \)

**Stride between warps**

**Base address of each CTA**
CTA-aware prefetcher
- Estimates prefetch addresses for trailing warp executions
- Base address of CTA + (stride between warps) × distance

<table>
<thead>
<tr>
<th>CTA 0</th>
<th>Warp 0</th>
<th>Warp 1</th>
<th>Warp 2</th>
<th>Warp 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTA 1</td>
<td>Warp 4</td>
<td>Warp 5</td>
<td>Warp 6</td>
<td>Warp 7</td>
</tr>
<tr>
<td>CTA 2</td>
<td>Warp 8</td>
<td>Warp 9</td>
<td>Warp 10</td>
<td>Warp 11</td>
</tr>
</tbody>
</table>

Base address of CTA0
Base address of CTA1
Base address of CTA2
CTA-Aware Prefetcher

**CTA-aware prefetcher**
- Estimates prefetch addresses for trailing warp executions
- \( \text{Base address of CTA} + (\text{stride between warps}) \times \text{distance} \)

![Diagram showing CTA and warp layout with prefetch addresses](image-url)
CTA-Aware Prefetcher

CTA-aware prefetcher hardware

- **PerCTA table**: CTA base addresses, leading warp ID
- **DIST table**: stride, misprediction count

<table>
<thead>
<tr>
<th>CTA 0</th>
<th>W0</th>
<th>W1</th>
<th>W2</th>
<th>W3</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTA 1</td>
<td>W4</td>
<td>W5</td>
<td>W6</td>
<td>W7</td>
</tr>
<tr>
<td>CTA 2</td>
<td>W8</td>
<td>W9</td>
<td>W10</td>
<td>W11</td>
</tr>
</tbody>
</table>

- Dist table: stride, misprediction cnt

- PerCTA table:
  - base addr, lead wid
  - base addr, lead wid
  - base addr, lead wid

- Prefetch request generator
Timeliness matters

- Prefetch is pending until detecting strides and CTA bases
- Reordering warp executions to detect required info quickly
- Warp-wakeup to prevent eviction of prefetched data
Conventional two-level scheduler

- Warps are fairly assigned to ready queue
- CTA base addresses are detected late

**CTA0**
- W0
- W1
- W2

**CTA1**
- W3
- W4
- W5

**CTA2**
- W6
- W7
- W8

**Ready queue**
- W0
- W1
- W2
- W3

**Pending queue**
- W4
- W5
- W6
- W7
- W8

*ld.global (cache miss)*
Conventional two-level scheduler

- Warps are fairly assigned to ready queue
- CTA base addresses are detected late
Conventional two-level scheduler

- Warps are fairly assigned to ready queue
- CTA base addresses are detected late
Prefetch-Aware Scheduler

**Conventional two-level scheduler**
- **Warps are fairly assigned to ready queue**
- **CTA base addresses are detected late**

### Scheduling Graph

- CTA0: W0, W1, W2
- CTA1: W3, W4, W5
- CTA2: W6, W7, W8

- **Ready queue**: W1, W2, W3, W4
- **Pending queue**: W5, W6, W7, W8, W0

- **PerCTA**
  - **Base0**
  - **DIST**
  - **stride**

- **Pr(W2)**
**Conventional two-level scheduler**

- Warps are fairly assigned to ready queue
- CTA base addresses are detected late

```
CTA0  W0  W1  W2
CTA1  W3  W4  W5
CTA2  W6  W7  W8

Ready queue:
W2  W3  W4  W5

Pending queue:
W6  W7  W8  W0  W1

Pr(W2)
```

PerCTA:
- **Base0**
- **DIST**
- **stride**
Prefetch-Aware Scheduler

**Conventional two-level scheduler**

- Warps are fairly assigned to ready queue
- CTA base addresses are detected late

![Scheduler diagram](image)

- CTA0: W0 W1 W2
- CTA1: W3 W4 W5
- CTA2: W6 W7 W8

- **Ready queue:** W3 W4 W5 W6
- **Pending queue:** W7 W8 W0 W1 W2

- **PerCTA:** Base0 Base1
- **DIST:** stride
Conventional two-level scheduler

- Warps are fairly assigned to ready queue
- CTA base addresses are detected late

- Warps are fairly assigned to ready queue
- CTA base addresses are detected late
Prefetch-Aware Scheduler

**Conventional two-level scheduler**
- Warps are fairly assigned to ready queue
- CTA base addresses are detected late

**Ready queue**
- W0, W1, W2
- W4, W5, W6, W7

**Pending queue**
- W8, W0, W1, W2, W3

**Compute phase (W4~W7)**
- W4, W5, W6, W7

**PerCTA**
- Base0
- Base1
- Dist
- Stride

**CTA0**
- W0, W1, W2

**CTA1**
- W3, W4, W5

**CTA2**
- W6, W7, W8
**Prefetch-aware scheduler**

- Reorganizes warp priorities to detect CTA addresses quickly
- Required information can be detected early

![Diagram showing CTA0, CTA1, CTA2, Ready queue, and Pending queue with corresponding warp priorities](image)
**Prefetch-aware scheduler**

- Reorganizes warp priorities to detect CTA addresses quickly
- Required information can be detected early

---

CTA0: W0, W1, W2
CTA1: W3, W4, W5
CTA2: W6, W7, W8

Ready queue: W0, W1, W3, W6
Pending queue: W2, W4, W5, W7, W8

PerCTA: Base0

Base0
**Prefetch-aware scheduler**

- Reorganizes warp priorities to detect CTA addresses quickly
- Required information can be detected early

ready to execute

```plaintext
CTA0  W0  W1  W2
CTA1  W3  W4  W5
CTA2  W6  W7  W8
```
**Prefetch-aware scheduler**

- Reorganizes warp priorities to detect CTA addresses quickly
- Required information can be detected early
**Prefetch-aware scheduler**

- Reorganizes warp priorities to detect CTA addresses quickly
- Required information can be detected early

<table>
<thead>
<tr>
<th>CTA0</th>
<th>W0</th>
<th>W1</th>
<th>W2</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTA1</td>
<td>W3</td>
<td>W4</td>
<td>W5</td>
</tr>
<tr>
<td>CTA2</td>
<td>W6</td>
<td>W7</td>
<td>W8</td>
</tr>
</tbody>
</table>

**Ready queue**

- W3
- W6
- W2
- W4

**Pending queue**

- W5
- W7
- W8
- W0
- W1

**PerCTA**

- Base0
- Base1
- Base2
- DIST
- stride
Prefetch-aware scheduler

- Reorganizes warp priorities to detect CTA addresses quickly
- Required information can be detected early

CTA0: W0 W1 W2
CTA1: W3 W4 W5
CTA2: W6 W7 W8

Ready queue:
- W2 W4 W5 W7

Pending queue:
- W8 W0 W1 W3 W6

Compute phase (W2, W4, W5, W7):
- Pr(W2)
- Pr(W4)
- Pr(W7)
- Pr(W5)
- Pr(W8)
Prefetch-aware scheduler

- Reorganizes warp priorities to detect CTA addresses quickly
- Required information can be detected early
Prefetch-aware scheduler

- Reorganizes warp priorities to detect CTA addresses quickly
- Required information can be detected early
### Evaluation

- **GPGPU-Sim v3.2.2**
- **Configuration: NVIDIA GTX480 (Fermi architecture)**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core</td>
<td>15 cores, 32 SIMT lanes, 1400MHz</td>
</tr>
<tr>
<td>Resources / SM</td>
<td>48 concurrent warps, 8 concurrent CTAs</td>
</tr>
<tr>
<td>Register file</td>
<td>128KB</td>
</tr>
<tr>
<td>Shared memory</td>
<td>48KB</td>
</tr>
<tr>
<td>Scheduler</td>
<td>two-level scheduler (8 ready warps)</td>
</tr>
<tr>
<td>L1I cache</td>
<td>2KB, 128B line, 4-way</td>
</tr>
<tr>
<td>L1D cache</td>
<td>16KB, 128B line, 4-way, LRU, 32 MSHR entries</td>
</tr>
<tr>
<td>L2 cache</td>
<td>64KB per partitions (12 partitions), 128B line, 8-way, LRU, 32 MSHR entries</td>
</tr>
<tr>
<td>DRAM</td>
<td>924MHz GDDR5, 6 channels, FR-FCFS</td>
</tr>
</tbody>
</table>
Evaluation: Performance

- **Performance is improved by 8%**

![](image_url)

- Irregular applications: 6%
Concurrent CTAs

- Exploit more parallelism & hardware resources
- CAPS is effective if more concurrent CTAs are running

Effective for a single CTA
Accuracy of prefetcher

- Inaccurate prefetch requests degrade performance
- CAPS provides high prefetching accuracy

Accuracy: 97%
**Timeliness**

- Early prefetch: prefetched data is evicted before demand requests
- Prefetch distance: cycle gaps between prefetch and demand fetch

**Early prefetch ratio (%)**

- INTRA
- INTER
- MTA
- CAPS
- CAPS w/o Wakeup

**Average cycles**

- LRR
- TLV
- PA-TLV

- Mean values: 64.3, 145.0, 172.7
**CTA-aware prefetcher and scheduler**

- Accurate prefetching address estimation by detecting CTA base addresses and strides
- Better timeliness with prefetch-aware warp scheduling
- Improves performance of GPU
Thank you

CTA-Aware Prefetching and Scheduling for GPU

Gunjae Koo, Hyeran Jeon, Zhenhong Lie, Nam Sung Kim, Murali Annavaram

✉ gunjae.koo@usc.edu  🚀 http://gunjaekoo.com