Locality-Aware Software Throttling for Sparse Matrix Operation on GPUs

Yanhao Chen¹, Ari B. Hayes¹, Chi Zhang², Timothy Salmon¹, Eddy Z. Zhang¹

¹. Rutgers University
². University of Pittsburgh
Sparse Matrix

- Sparse Linear Systems
  - CG
  - GMRES
- Physics Based Simulations
  - CFD
- Deep Learning Optimizations
  - Pruned Neural Networks

……
Sparse Matrix Operation

\[ y = Ax \]

- **Output Vector**
- **Input Vector**
- **Sparse Matrix**

\[ y_i = \text{reduce} \{ A_{ik} \odot x_k \} \]

\[ i \in [1, \ldots, m], k \in [1, \ldots, n] \]

**Binary Operator**
Sparse Matrix Vector Multiplication (SpMV)

\[ y_i = \text{reduce}\{A_{ik} \odot x_k\} \]

\[ \text{reduce} = \text{sum} \]
\[ \odot = \ast \]

\[ y_i = \text{sum}\{A_{i,k} \ast x_k\} \]

\[ i \in [1, \ldots, m], k \in [1, \ldots, n] \]

\[
\begin{array}{cccc}
A_{1,1} & A_{1,2} & A_{1,3} & A_{1,4} \\
0 & 0 & 0 & 0 \\
A_{3,1} & 0 & A_{3,3} & 0 \\
0 & A_{4,2} & 0 & A_{4,4}
\end{array}
\]

\[
\begin{array}{c}
x_1 \\
x_2 \\
x_3 \\
x_4
\end{array}
\]

\[
\begin{array}{c}
y_1 \\
y_2 \\
y_3 \\
y_4
\end{array}
\]

\[ y_3 = A_{3,1} \ast x_1 + A_{3,3} \ast x_3 \]
Single Source Shortest Path (SSSP)

$$y_i = \text{reduce}\{A_{ik} \odot x_k\}$$

.reduce = \min

$$y_i = \min\{A_{ki} + x_k\}$$

$$i \in [1, \ldots, m], k \in [1, \ldots, n]$$

$$y_5 = \min\{x_5, A_{3,5} + x_3, A_{4,5} + x_4\}$$

Adjacent Matrix $A$
Problem with Sparsity on GPUs

• Low data reuse is always a big problem

• e.g. SpMV
  – The input vector and the output vector can be reused a lot
  – They are usually too large to fit into GPU’s cache
  – The sparsity of the matrix causes irregular accesses of the vectors
  – This means low reuse of the data in the cache

\[ y_i = \text{sum}\{A_{i,k} \times x_k\} \]
Existing Methods to Improve Data Reuse on GPUs

• Warp Scheduling Policy
  – Throttling concurrent threads
  – Limits the number of active warps [Rogers+, MICRO’12]
  – DYNCTA: controls the number of CTAs [Kayiran+, PACT’13]

  Need Hardware Modification!

• Computation and Data Layout Transformation
  – Reduce irregular memory accesses
  – Improve Memory Coalescing [Zhang+, ICS’10]

  Only focus on Spatial Data Reuse!
Our Throttling framework for GPUs...

- Is the **First Software Throttling** implementation
- Is focused on **Temporal Data Reuse**
- Exploits the **Trade-off** between throttling performance and GPU throughput
SpMV with Software Throttling

\[
\begin{align*}
A_{1,1} & A_{1,2} & A_{1,3} & A_{1,4} \\
0 & 0 & 0 & 0 \\
A_{3,1} & 0 & A_{3,3} & 0 \\
0 & A_{4,2} & 0 & A_{4,4} \\
\end{align*}
\]

\[
\begin{align*}
x_1 & \quad x_2 & \quad x_3 & \quad x_4 \\
y_1 & \quad y_2 & \quad y_3 & \quad y_4 \\
\end{align*}
\]

\[
A \times x = y
\]
SpMV with Software Throttling

Matrix $A$ is bypassing the cache

$$A = \begin{bmatrix} A_{1,1} & A_{1,2} & A_{1,3} & A_{1,4} \\ 0 & 0 & 0 & 0 \\ A_{3,1} & 0 & A_{3,3} & 0 \\ 0 & A_{4,2} & 0 & A_{4,4} \end{bmatrix}$$

$$A \times \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix}$$
SpMV with Software Throttling

Matrix A is bypassing the cache

\[
\begin{bmatrix}
A_{1,1} & A_{1,2} & A_{1,3} & A_{1,4} \\
0 & 0 & 0 & 0 \\
A_{3,1} & 0 & A_{3,3} & 0 \\
0 & A_{4,2} & 0 & A_{4,4}
\end{bmatrix}
\]

\[
\begin{align*}
X &= \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} \\
Y &= \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix}
\end{align*}
\]

\[
\{ < x_1 \ y_1 > < x_2 \ y_1 > < x_3 \ y_1 > < x_4 \ y_1 > \\
< x_1 \ y_3 > < x_3 \ y_3 > < x_2 \ y_4 > < x_4 \ y_4 > \}
\]

Running at one time
SpMV with Software Throttling

Original Case

Cache Capacity: 4

Matrix A is bypassing the cache

\[
\begin{bmatrix}
A_{1,1} & A_{1,2} & A_{1,3} & A_{1,4} \\
A_{2,1} & 0 & 0 & 0 \\
0 & A_{3,3} & 0 & 0 \\
0 & A_{4,2} & 0 & A_{4,4}
\end{bmatrix}
\]

\[
\begin{align*}
X & = \\
\begin{bmatrix}
x_1 \\
x_2 \\
x_3 \\
x_4
\end{bmatrix} & \begin{bmatrix}
y_1 \\
y_2 \\
y_3 \\
y_4
\end{bmatrix}
\end{align*}
\]

\[
\{ <x_1,y_1> <x_2,y_1> <x_3,y_1> <x_4,y_1> \\
<x_1,y_3> <x_3,y_3> <x_2,y_4> <x_4,y_4> \}
\]

Running at one time
SpMV with Software Throttling

**Throttling Phase 1**

Cache Capacity: 4

\[
A = \begin{pmatrix}
A_{1,1} & A_{1,2} & A_{1,3} & A_{1,4} \\
0 & 0 & 0 & 0 \\
A_{3,1} & 0 & A_{3,3} & 0 \\
0 & A_{4,2} & 0 & A_{4,4}
\end{pmatrix}
\]

\[
X = \begin{pmatrix}
x_1 \\
x_2 \\
x_3 \\
x_4
\end{pmatrix}
\]

\[
Y = \begin{pmatrix}
y_1 \\
y_2 \\
y_3 \\
y_4
\end{pmatrix}
\]

\[
< x_1, y_1 > \quad < x_1, y_3 > \quad < x_3, y_1 > \quad < x_3, y_3 >
\]

\[
< x_2, y_1 > \quad < x_2, y_4 > \quad < x_4, y_1 > \quad < x_4, y_4 >
\]
SpMV with Software Throttling

Throttling Phase 1

Cache Capacity: 4

$x_1 y_1 x_3 y_3$

All Data Can Fit into the Cache

$< x_1 y_1 > < x_1 y_3 > < x_3 y_1 > < x_3 y_3 >$

$< x_2 y_1 > < x_2 y_4 > < x_4 y_1 > < x_4 y_4 >$

A

$\begin{array}{ccc}
A_{1,1} & A_{1,2} & A_{1,3} & A_{1,4} \\
0 & 0 & 0 & 0 \\
A_{3,1} & 0 & A_{3,3} & 0 \\
0 & A_{4,2} & 0 & A_{4,4} \\
\end{array}$

$x_1$

$x_2$

$x_3$

$x_4$

$y_1$

$y_2$

$y_3$

$y_4$

Time

2018 USENIX Annual Technical Conference
SpMV with Software Throttling

Throttling Phase 2

Cache Capacity: 4

\[
\begin{array}{cccc}
A_{1,1} & A_{1,2} & A_{1,3} & A_{1,4} \\
0 & 0 & 0 & 0 \\
A_{3,1} & 0 & A_{3,3} & 0 \\
0 & A_{4,2} & 0 & A_{4,4} \\
\end{array}
\]

\[
\begin{array}{c}
x_1 \\
x_2 \\
x_3 \\
x_4 \\
\end{array}
\quad \times \quad
\begin{array}{c}
x_1 \\
x_2 \\
x_3 \\
x_4 \\
\end{array}
\quad = \quad
\begin{array}{c}
y_1 \\
y_2 \\
y_3 \\
y_4 \\
\end{array}
\]

\[
< x_1 y_1 > < x_1 y_3 > < x_3 y_1 > < x_3 y_3 >
\]

\[
< x_2 y_1 > < x_2 y_4 > < x_4 y_1 > < x_4 y_4 >
\]

Time
SpMV with Software Throttling

Throttling Phase 2

Cache Capacity: 4

\[
\begin{bmatrix}
x_2 & y_1 & x_4 & y_4
\end{bmatrix}
\]

All Data Can Fit into the Cache

< $x_1 \ y_1$ > < $x_1 \ y_3$ > < $x_3 \ y_1$ > < $x_3 \ y_3$ >

< $x_2 \ y_1$ > < $x_2 \ y_4$ > < $x_4 \ y_1$ > < $x_4 \ y_4$ >

\[
\begin{array}{cccc}
A_{1,1} & A_{1,2} & A_{1,3} & A_{1,4} \\
0 & 0 & 0 & 0 \\
A_{3,1} & 0 & A_{3,3} & 0 \\
0 & A_{4,2} & 0 & A_{4,4} \\
\end{array}
\]

\[
\begin{bmatrix}
x_1 \\
x_2 \\
x_3 \\
x_4 \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
y_1 \\
y_2 \\
y_3 \\
y_4 \\
\end{bmatrix}
\]

Time
What We Need for Software Throttling

• An effective **partitioning algorithm**
  – All data items in one partition can fit into the cache
  – The interaction between different partitions are minimized

• Applicable **scheduling methods**
  – Handle the trade-off between throttling and throughput
What We Need for Software Throttling

• An effective **partitioning algorithm**
  – All data items in one partition can fit into the cache
  – The interaction between different partitions are minimized

• Applicable **scheduling methods**
  – Handle the trade-off between throttling and throughput
Graph Representation

- **Graph Edge Partition Model**
  - Places an emphasis on **Data**
  - Node $\rightarrow$ Data object
  - Edge $\rightarrow$ Interaction between data

```
\begin{array}{c|cccc}
  & 1 & 2 & 3 & 4 \\
\hline
1 & A_{1,1} & A_{1,2} & A_{1,3} & A_{1,4} \\
2 & 0 & 0 & 0 & 0 \\
3 & A_{3,1} & 0 & A_{3,3} & 0 \\
4 & 0 & A_{4,2} & 0 & A_{4,4} \\
\end{array}
```

\[ X = AXY \]
Why Edge Partition Model?

1. Better **load balancing**
   - *PowerGraph* [OSDI’12], *Streaming Edge Partition* [KDD’14], *SPAC* [SIGMETRICS’17]
     - Balanced vertex partition is sometimes **NOT equal** to balanced workload

2. **Quantifying** the communication cost

3. Applies to **a large class of** parallel applications
   - *N-body*, *CFD*, *Sparse Linear Algebra*, *Graph Analytics*, …
Different Edge Partition Models

Load Balanced Partition

Partition 1

$Y_1 \rightarrow X_1$
$Y_2 \rightarrow X_2$
$Y_3 \rightarrow X_3$
$Y_4 \rightarrow X_4$

Partition 2

# Edges: 3

Data Balanced Partition

Partition 1

$Y_1 \rightarrow X_1$
$Y_2 \rightarrow X_2$
$Y_3 \rightarrow X_3$
$Y_4 \rightarrow X_4$

Partition 2

# Edges: 3

Cache-Fit Partition

2018 USENIX Annual Technical Conference
Data Balanced Partition

- Recursive **Bisection** Framework
  - 2-way Load Balanced Edge Partition
    - **SPAC** [Li+, SIGMETRICS’17]
    - Minimize vertex replica (data copy)

Cache Capacity: 4
Data Balanced Partition

• Recursive **Bisection** Framework
  – 2-way Load Balanced Edge Partition
    • **SPAC** [Li+, SIGMETRICS’17]
    – Minimize vertex replica (data copy)
Data Balanced Partition

• Recursive **Bisection** Framework
  – 2-way Load Balanced Edge Partition
    • **SPAC** [Li+, SIGMETRICS’17]
    – Minimize vertex replica (data copy)
Data Balanced Partition

• Recursive **Bisection** Framework
  – 2-way Load Balanced Edge Partition
  • **SPAC** [Li+, SIGMETRICS’17]
  – Minimize vertex replica (data copy)

Cache Capacity: 4
Data Balanced Partition

• Recursive **Bisection** Framework
  – 2-way Load Balanced Edge Partition
  • **SPAC** [Li+, SIGMETRICS’17]
  – Minimize vertex replica (data copy)

Cache Capacity: 4
What We Need for Software Throttling

• A good partitioning algorithm
  – All data items in one partition can fit into the cache
  – The interaction between different partitions are minimized

• Applicable scheduling methods
  – Handle the trade-off between throttling and throughput
Effective scheduling methods

• Four different scheduling methods
  – Cache-Fit (CF)
  – Cache-Fit Queue (CF-Q)
  – Split-Join (SJ)
  – Split-Join Queue (SJ-Q)
Effective scheduling methods

• Four different scheduling methods
  – Cache-Fit (CF)
  – Cache-Fit Queue (CF-Q)
  – Split-Join (SJ)
  – Split-Join Queue (SJ-Q)
Cache-Fit (CF) Scheduling

- Isolate the computation of different Cache-Fit Partitions
- Run one Cache-Fit Partition at one time

CUDA Function

**Original:** \( \text{Kernel}<<<\text{blocknum}, \text{blockdim}>>>(\text{TL}, N); \)

**Phase 1:** \( \text{Kernel}<<<\text{blocknum}, \text{blockdim}>>>(\text{TL'[0]}, P_0); \)

**Phase 2:** \( \text{Kernel}<<<\text{blocknum}, \text{blockdim}>>>(\text{TL'[1]}, P_1); \)

**......**

**Phase k:** \( \text{Kernel}<<<\text{blocknum}, \text{blockdim}>>>(\text{TL'[k]}, P_k); \)

Strict Barriers

- TL: tuple list
- N: # of tuples
- TL': new tuple list
- \( P_i \): # of tuples in TL[i]
Low Pipeline Utilization

Cache Capacity: 4

Kernel 1 -- Partition A

4 Working Threads
Low Throughput

Cache Capacity: 4

Kernel 2 -- Partition B

IDLE

4 Working Threads
Low Pipeline Utilization

Cache Capacity: 4

4 Working Threads

low pipeline utilization

Kernel 3 -- Partition C

IDLE
Effective scheduling methods

• Four different scheduling methods
  – Cache-Fit (CF)
  – Cache-Fit Queue (CF-Q)
  – Split-Join (SJ)
  – Split-Join Queue (SJ-Q)
Cache-Fit Queue (CF-Q) Scheduling

• Invoke a **single** kernel call but still enable **throttling**

• Set up a FIFO queue

• Each entry corresponds to a **chunk**
  – A chunk is part of a cache-fit partition
  – Adjacent **chunks** are from the same **Cache-Fit Partition**

• Each warp fetches a **chunk** from the queue and processes it
Cache-Fit Queue (CF-Q) Scheduling cont.

Cache-Fit Queue (CF-Q) Scheduling cont.

Cache-Fit Queue (CF-Q) Scheduling cont.

Cache-Fit Queue (CF-Q) Scheduling cont.

Cache-Fit Queue (CF-Q) Scheduling cont.

Cache-Fit Queue (CF-Q) Scheduling cont.

Cache-Fit Queue (CF-Q) Scheduling cont.
Effective scheduling methods

- Four different scheduling methods
  - Cache-Fit (CF)
  - Cache-Fit Queue (CF-Q)
  - **Split-Join (SJ)**
  - Split-Join Queue (SJ-Q)
Split-Join (SJ) Scheduling

- Dynamically merge Cache-Fit Partitions

- Perform an Online Profiling to decide which partitions should be merged

- Use the Tree Representation of the data balanced partition to help the Online Profiling
Split-Join (SJ) Scheduling

Cache Capacity: 4
Split-Join (SJ) Scheduling

**stime**: measured standalone running time

**btime**: optimal running time on this node

---

**Partition A**

- **A.stime**: 0.5
- **A.btime**: 0.5

**Partition B**

- **B.stime**: 0.2
- **B.btime**: 0.2

**Partition C**

- **C.stime**: 0.2
- **C.btime**: 0.2
Split-Join (SJ) Scheduling

stime: measured standalone running time
btime: optimal running time on this node

All.stime > A.btime + BC.btime

All.stime: 1.2
All.btime: 0.8

BC.stime < B.btime + C.btime

BC.stime: 0.3
BC.btime: 0.3

A.stime: 0.5
A.btime: 0.5

B.stime: 0.2
B.btime: 0.2

C.stime: 0.2
C.btime: 0.2
Effective scheduling methods

- Four different scheduling methods
  - Cache-Fit (CF)
  - Cache-Fit Queue (CF-Q)
  - Split-Join (SJ)
  - Split-Join Queue (SJ-Q)
Split-Join Queue (SJ-Q) Scheduling

- Provide **strict barriers** between different merged partitions

- No **barriers** inside a merged partition of **SJ**
  - No guarantee of the execution order

- Set up one FIFO queue for each merged partition
  - Provide **relaxed barriers** between cache-fit partitions from the same merged partition
## Four Scheduling Methods Summary

<table>
<thead>
<tr>
<th>Methods</th>
<th>Pipeline Utilization</th>
<th>Profiling</th>
<th>Barriers</th>
<th>Queue</th>
<th>Code Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>CF</td>
<td>Low</td>
<td>No</td>
<td>Strict</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>CF-Q</td>
<td>High</td>
<td>No</td>
<td>Relaxed</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>SJ</td>
<td>High</td>
<td>Yes</td>
<td>Strict</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>SJ-Q</td>
<td>High</td>
<td>Yes</td>
<td>Strict / Relaxed</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Software Throttling Performance

• Experiment Settings

<table>
<thead>
<tr>
<th>GPU Model</th>
<th>Titan X</th>
<th>GTX 745</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>Pascal</td>
<td>Maxwell</td>
</tr>
<tr>
<td>Core #</td>
<td>5376</td>
<td>576</td>
</tr>
<tr>
<td>L2 Cache</td>
<td>3MB</td>
<td>2MB</td>
</tr>
<tr>
<td>CUDA Version</td>
<td>CUDA 8.0</td>
<td>CUDA 8.0</td>
</tr>
<tr>
<td>CPU</td>
<td>Intel Xeon E5-2620</td>
<td>Intel Core i7-4790</td>
</tr>
</tbody>
</table>
Benchmarks

• Sparse Linear Algebra Workloads
  – Sparse Matrix Vector Multiplication (SPMV)

• Graph Processing Workloads
  – Bellman-Ford (BMF)
  – Page Rank (PR)

• Neural Network Optimization
  – Deep Compression: Pruned AlexNet
Sparse Matrix Vector Multiplication

• Baseline: CUSP

• Matrices: Florida Sparse Matrix Collection
  – Focus on large matrices: working set cannot fit into L2 cache
  – 228 large matrices on GTX 745
  – 192 large matrices on Titan X
Overall SPMV Performance

Average Speedup For SPMV

Normalized Speedup

- Org + R
- CF
- SJ
- CF-Q
- SJ-Q

GTX 745
Titan X

2.018
Effect of Cache Hit Rate

Normalized Speedup

Titan X

Org + R CF SJ CF-Q SJ-Q

4.16 4.24

0% - 10% 10% - 20% 20% - 30% 30% - 40% 40% - 100%

2018 USENIX Annual Technical Conference
Effect of Working Set Size

Normalized Speedup

Titan X

- Org + R
- CF
- SJ
- CF-Q
- SJ-Q

1X - 2X (cache size)
2X - 4X
4X - 8X
8X ~ INF

3.02
Graph Application Performance

BMF & PR Performance using SJ w/ overhead

Normalized Speedup

Pokec  |  WebGoogle |  Wikipedia* |  WikiTalk |  IMDB |  RoadCentral |  RoadCal

*full name: Wikipedia-051105

RoadCal is a small Graph
Graph Application Performance cont.

BMF Cache Hit Rate

PR Cache Hit Rate

RoadCal is a small Graph
Deep Learning Benchmark

- Deep Compression [Han+, ICLR’16]
  - Prune AlexNet to remove low weight elements in fully connected layers
  - Deep Compression provide us sparse matrices

![Speedup for AlexNet using Deep Compression](image)

- fc7 and fc8 has small vector size
Conclusion

• We proposed the first locality-aware **Software Throttling** framework for GPUs

• Our framework can increase data reuse by improving **Temporal Locality**

• We exploited the **Trade-off** between cache performance and pipeline utilization
Questions?