Locality-Aware Software Throttling for Sparse Matrix Operation on GPUs

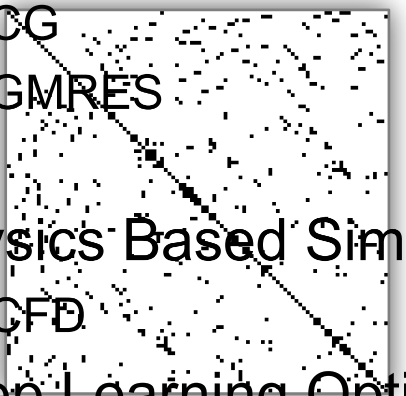
Yanhao Chen[1], Ari B. Hayes[1], Chi Zhang[2],
Timothy Salmon[1], Eddy Z. Zhang[1]

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

$$\mathbf{y} = \mathbf{A}\,\mathbf{x}$$

Output Vector

Input Vector

Sparse Matrix

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

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

Binary Operator

Sparse Matrix Vector Multiplication (SpMV)

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

$$reduce = \text{sum}$$

$$\odot = *$$

$$\Downarrow$$

$$y_i = \textbf{sum}\{A_{i,k} * x_k\}$$

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

$$\underbrace{\begin{bmatrix} A_{1,1} & A_{1,2} & A_{1,3} & A_{1,4} \\ 0 & 0 & 0 & 0 \\ A_{3,1} & 0 & A_{3,3} & 0 \\ 0 & A_{4,2} & 0 & A_{4,4} \end{bmatrix}}_{\mathbf{A}} * \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}}_{\mathbf{x}} = \underbrace{\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix}}_{\mathbf{y}}$$

$$y_3 = A_{3,1} * x_1 + A_{3,3} * x_3$$

Single Source Shortest Path (SSSP)

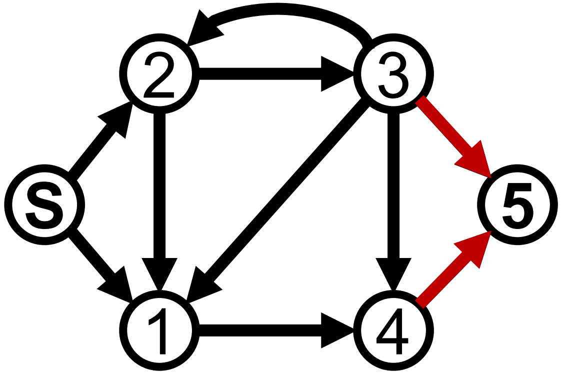
$$y_i = \color{red}{reduce}\{A_{ik} \color{blue}{\odot} x_k\}$$

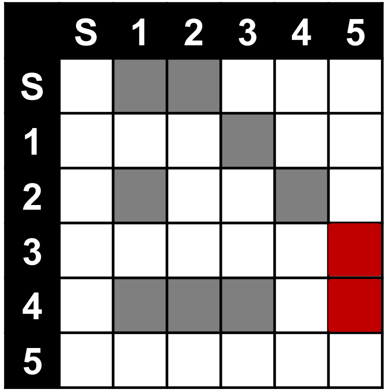
$$\color{red}{reduce} = \min$$

$$\color{blue}{\odot} = +$$

$$y_i = \color{red}{\mathbf{min}}\{A_{ki} \color{blue}{+} x_k\}$$

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





Adjacent Matrix **A**

y: distance vector of ***j*** th iteration

x : (previous) distance vector of ***j-1*** th iteration

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

Problem with Sparsity on GPUs

- Low data reuse is always a big problem

$$y_i = \mathbf{sum}\{A_{i,k} * x_k\}$$

- 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

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

$$\underset{\textbf{A}}{\begin{bmatrix} A_{1,1} & A_{1,2} & A_{1,3} & A_{1,4} \\ 0 & 0 & 0 & 0 \\ A_{3,1} & 0 & A_{3,3} & 0 \\ 0 & A_{4,2} & 0 & A_{4,4} \end{bmatrix}} \times \underset{\textbf{X}}{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}} = \underset{\textbf{Y}}{\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix}}$$

SpMV with Software Throttling

Matrix A is bypassing the cache

$$\underbrace{\begin{bmatrix} A_{1,1} & A_{1,2} & A_{1,3} & A_{1,4} \\ 0 & 0 & 0 & 0 \\ A_{3,1} & 0 & A_{3,3} & 0 \\ 0 & A_{4,2} & 0 & A_{4,4} \end{bmatrix}}_{\mathbf{A}} \times \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}}_{\mathbf{X}} = \underbrace{\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix}}_{\mathbf{Y}}$$

SpMV with Software Throttling

Original Case

Matrix A is
bypassing the cache

$$\underbrace{\begin{array}{|c|c|c|c|}\hline A_{1,1} & A_{1,2} & A_{1,3} & A_{1,4} \\ \hline 0 & 0 & 0 & 0 \\ \hline A_{3,1} & 0 & A_{3,3} & 0 \\ \hline 0 & A_{4,2} & 0 & A_{4,4} \\ \hline \end{array}}_{\mathbf{A}} \times \underbrace{\begin{array}{|c|}\hline x_1 \\ \hline x_2 \\ \hline x_3 \\ \hline x_4 \\ \hline \end{array}}_{\mathbf{X}} = \underbrace{\begin{array}{|c|}\hline y_1 \\ \hline y_2 \\ \hline y_3 \\ \hline y_4 \\ \hline \end{array}}_{\mathbf{Y}}$$

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

Running at one time

SpMV with Software Throttling

$$\underset{\mathbf{A}}{\begin{bmatrix} A_{1,1} & A_{1,2} & A_{1,3} & A_{1,4} \\ 0 & 0 & 0 & 0 \\ A_{3,1} & 0 & A_{3,3} & 0 \\ 0 & A_{4,2} & 0 & A_{4,4} \end{bmatrix}} \times \underset{\mathbf{X}}{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}} = \underset{\mathbf{Y}}{\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix}}$$

Matrix A is bypassing the cache

Original Case

Cache Capacity: 4

x_1	y_1	x_2	x_3

x_4	y_3	y_4

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

Running at one time

SpMV with Software Throttling

$$\underbrace{\begin{bmatrix} A_{1,1} & A_{1,2} & A_{1,3} & A_{1,4} \\ 0 & 0 & 0 & 0 \\ A_{3,1} & 0 & A_{3,3} & 0 \\ 0 & A_{4,2} & 0 & A_{4,4} \end{bmatrix}}_{\mathbf{A}} \times \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}}_{\mathbf{X}} = \underbrace{\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix}}_{\mathbf{Y}}$$

Throttling Phase 1

Cache Capacity: 4

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

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

Time

SpMV with Software Throttling

$$\underbrace{\begin{bmatrix} A_{1,1} & A_{1,2} & A_{1,3} & A_{1,4} \\ 0 & 0 & 0 & 0 \\ A_{3,1} & 0 & A_{3,3} & 0 \\ 0 & A_{4,2} & 0 & A_{4,4} \end{bmatrix}}_{\mathbf{A}} \times \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}}_{\mathbf{X}} = \underbrace{\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix}}_{\mathbf{Y}}$$

Throttling Phase 1

Cache Capacity: 4

x_1	y_1	x_3	y_3

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

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

Time

All Data Can Fit into the Cache

SpMV with Software Throttling

$$\underbrace{\begin{bmatrix} A_{1,1} & A_{1,2} & A_{1,3} & A_{1,4} \\ 0 & 0 & 0 & 0 \\ A_{3,1} & 0 & A_{3,3} & 0 \\ 0 & A_{4,2} & 0 & A_{4,4} \end{bmatrix}}_{\mathbf{A}} \times \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}}_{\mathbf{X}} = \underbrace{\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix}}_{\mathbf{Y}}$$

Throttling
Phase 2

Cache Capacity: 4

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

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

Time

SpMV with Software Throttling

$$\underbrace{\begin{bmatrix} A_{1,1} & A_{1,2} & A_{1,3} & A_{1,4} \\ 0 & 0 & 0 & 0 \\ A_{3,1} & 0 & A_{3,3} & 0 \\ 0 & A_{4,2} & 0 & A_{4,4} \end{bmatrix}}_{\mathbf{A}} \times \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}}_{\mathbf{X}} = \underbrace{\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix}}_{\mathbf{Y}}$$

Throttling Phase 2

Cache Capacity: 4

x_2	y_1	x_4	y_4

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

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

Time

All Data Can Fit into the Cache

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

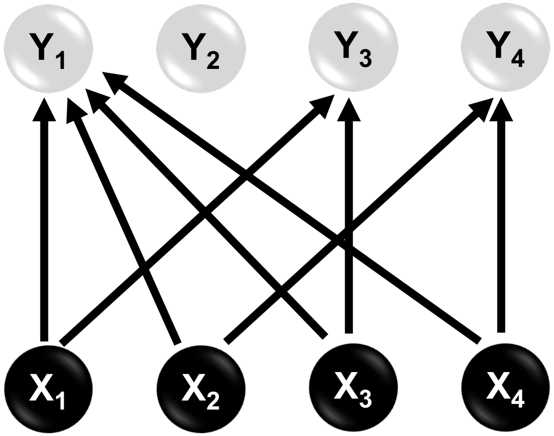
What We Need for Software Throttling

- 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 → Data object
  - Edge → Interaction between data



$$\begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & A_{1,1} & A_{1,2} & A_{1,3} & A_{1,4} \\ 2 & 0 & 0 & 0 & 0 \\ 3 & A_{3,1} & 0 & A_{3,3} & 0 \\ 4 & 0 & A_{4,2} & 0 & A_{4,4} \end{array} \times \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix}$$

A **X** **Y**

Why Edge Partition Model?

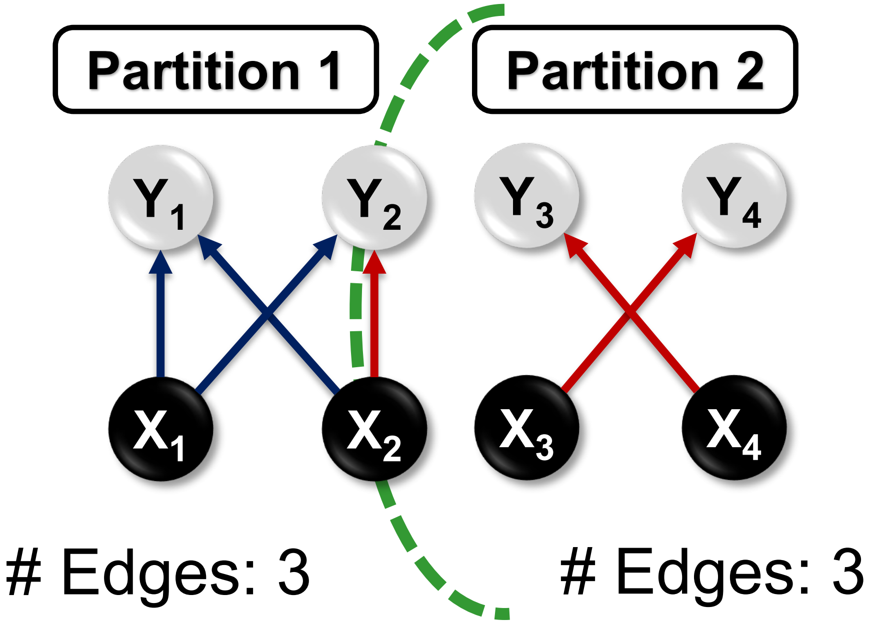
1. Better **load balancing**
   - *PowerGraph* [OSDI'12], *Streaming Edge Partition* [KDD'14], *SPAC* [SIGMETRICS'17]
     - Balanced vertex partition is sometimes **NOT equal** to balanced workload

2. **Quantifying** the communication cost

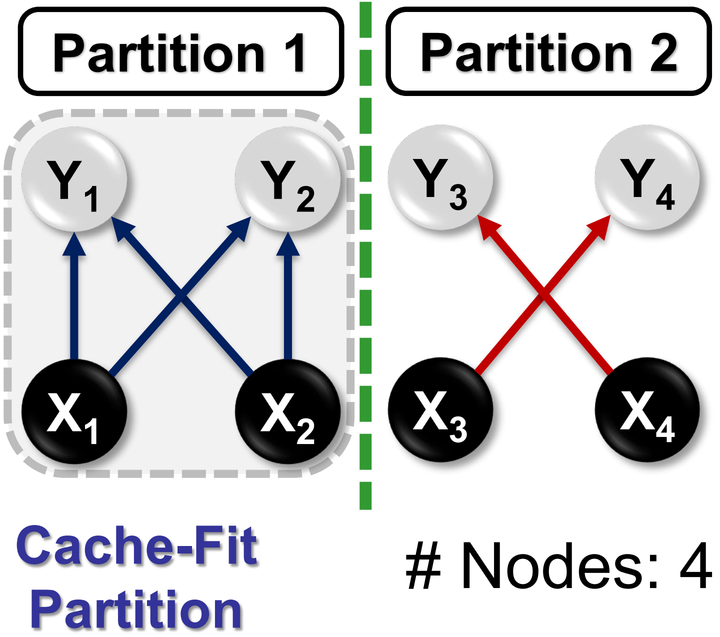
3. Applies to **a large class of** parallel applications
   - *N-body, CFD, Sparse Linear Algebra, Graph Analytics, …*

Different Edge Partition Models

Load Balanced Partition

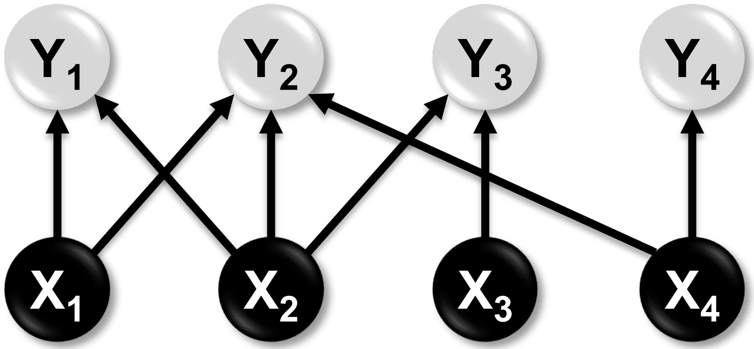


Data Balanced Partition



Data Balanced Partition

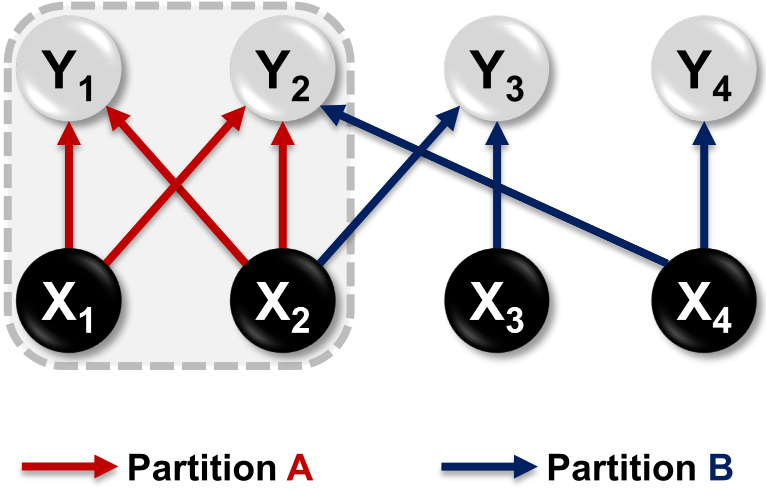
- Recursive **Bisection** Framework
  - 2-way Load Balanced Edge Partition
    - ***SPAC*** [Li+,SIGMETRICS'17]
  - Minimize vertex replica (data copy)



Cache Capacity: 4

Data Balanced Partition

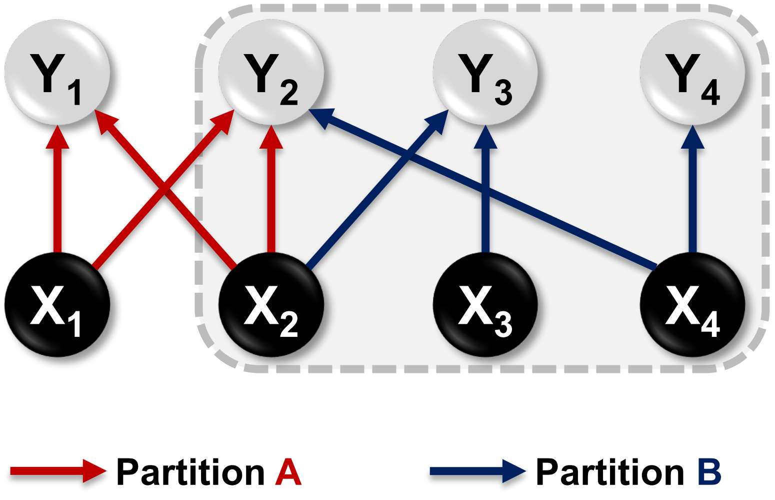
- Recursive **Bisection** Framework
  - 2-way Load Balanced Edge Partition
    - *SPAC* [Li+,SIGMETRICS'17]
  - Minimize vertex replica (data copy)



Cache Capacity: 4

Data Balanced Partition

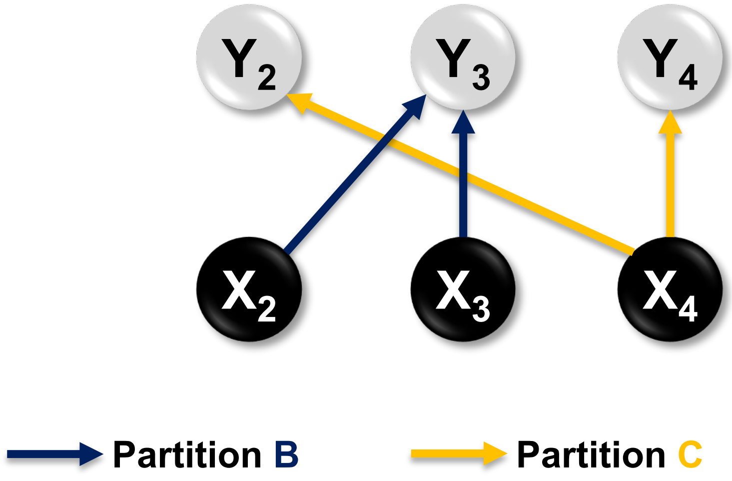
- Recursive **Bisection** Framework
  - 2-way Load Balanced Edge Partition
    - *SPAC* [Li+,SIGMETRICS'17]
  - Minimize vertex replica (data copy)



Cache Capacity: 4

Data Balanced Partition

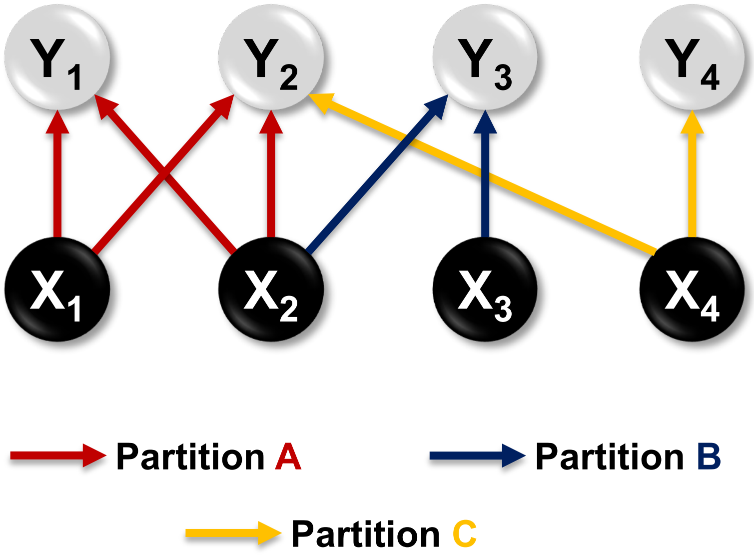
- Recursive **Bisection** Framework
  - 2-way Load Balanced Edge Partition
    - ***SPAC*** [Li+,SIGMETRICS'17]
  - Minimize vertex replica (data copy)



Cache Capacity: 4

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: Kernel<<<blocknum, blockdim>>(**TL**, **N**);

↓

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

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

……

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

Strict Barriers

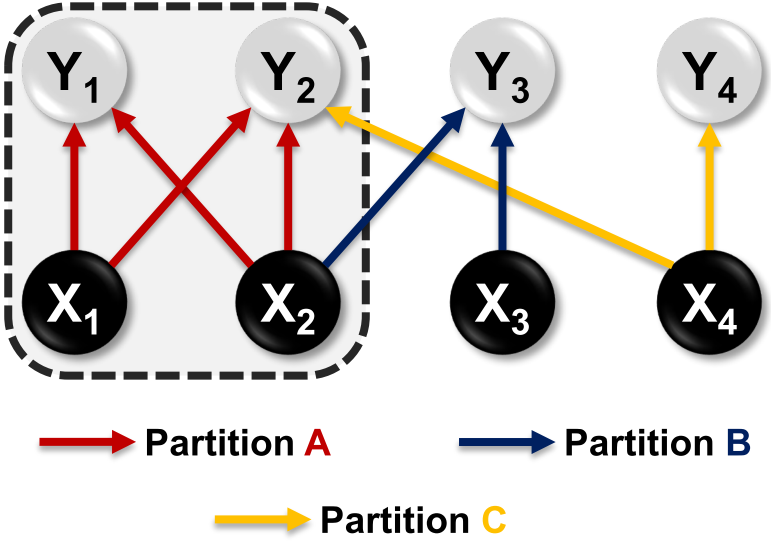
TL: tuple list

N: # of tuples

TL': new tuple list

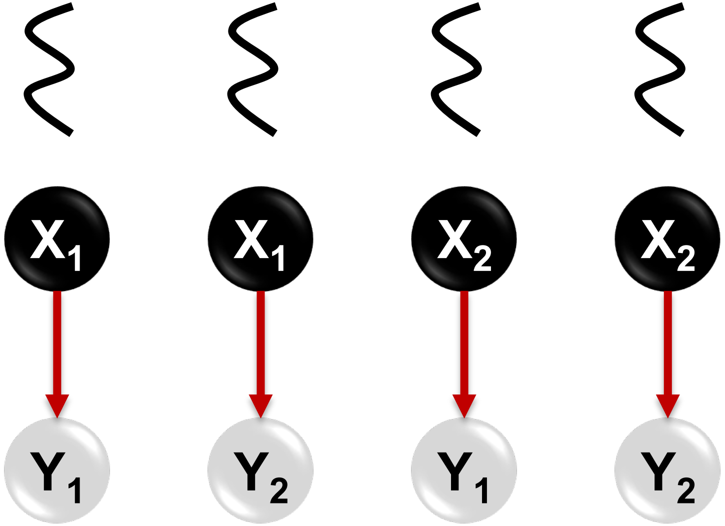
P_i: # of tuples in TL[i]

Low Pipeline Utilization



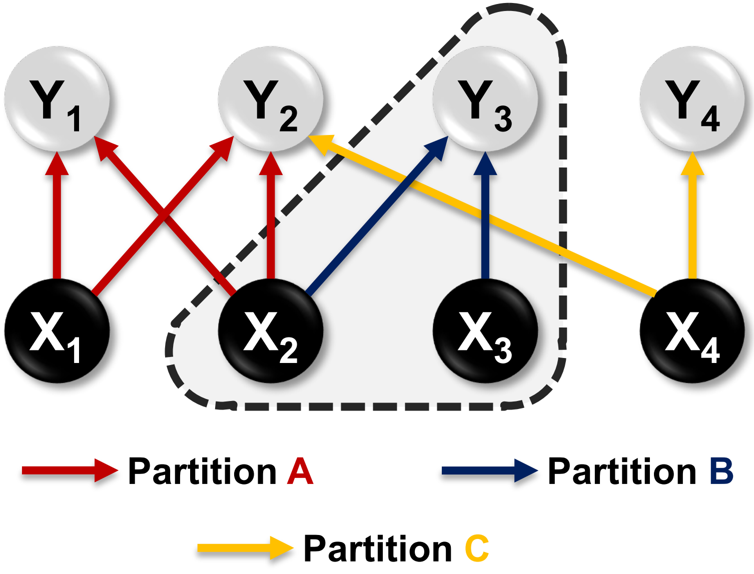
Cache Capacity: 4

4 Working Threads



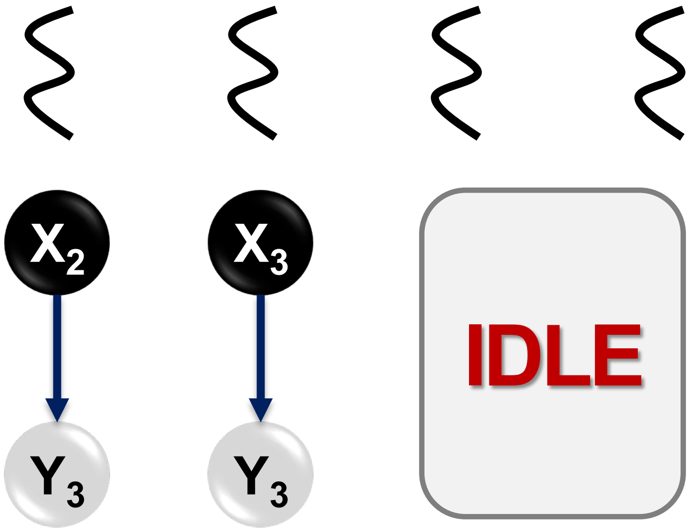
Kernel 1 -- Partition A

Low Throughput



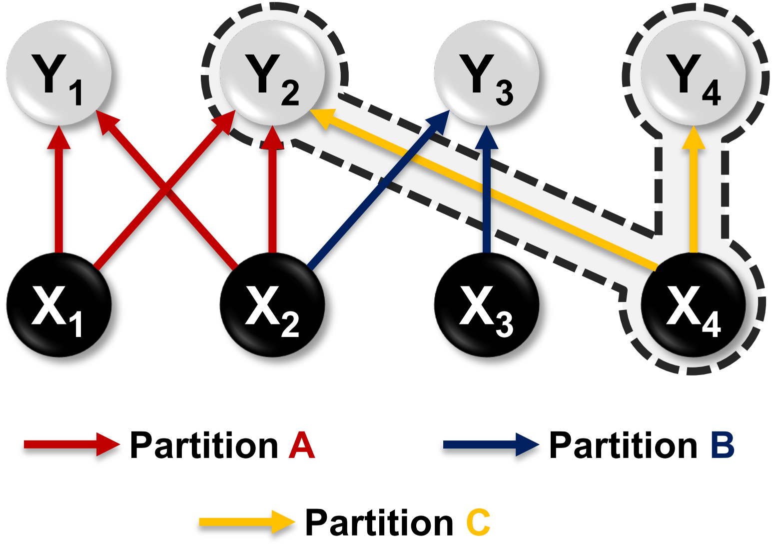
Cache Capacity: 4

4 Working Threads



Kernel 2 -- Partition B

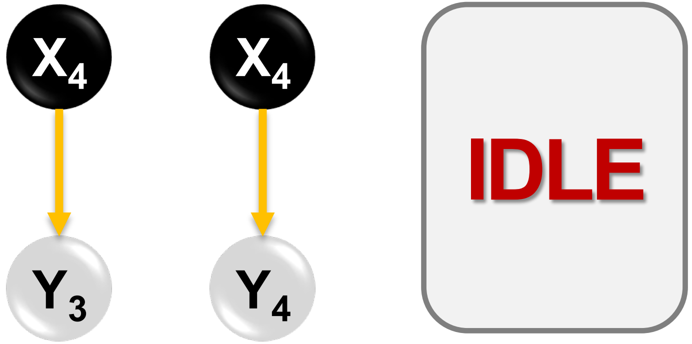
Low Pipeline Utilization



Cache Capacity: 4

4 Working Threads

low pipeline utilization



Kernel 3 -- Partition C

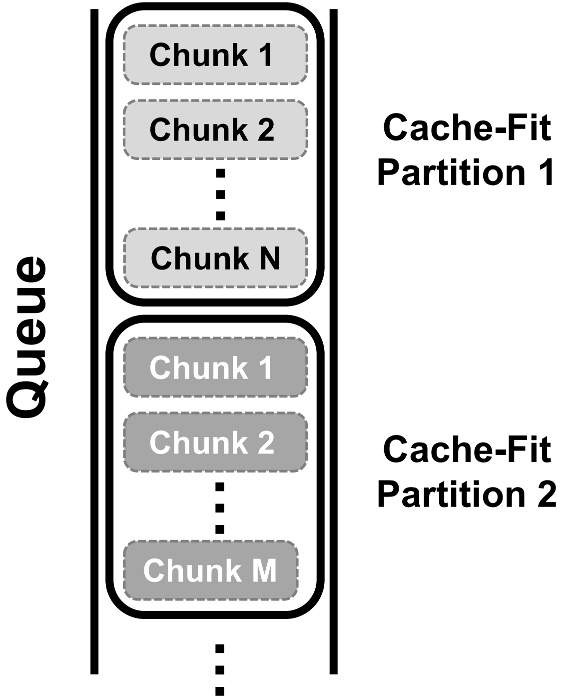
Effective scheduling methods

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

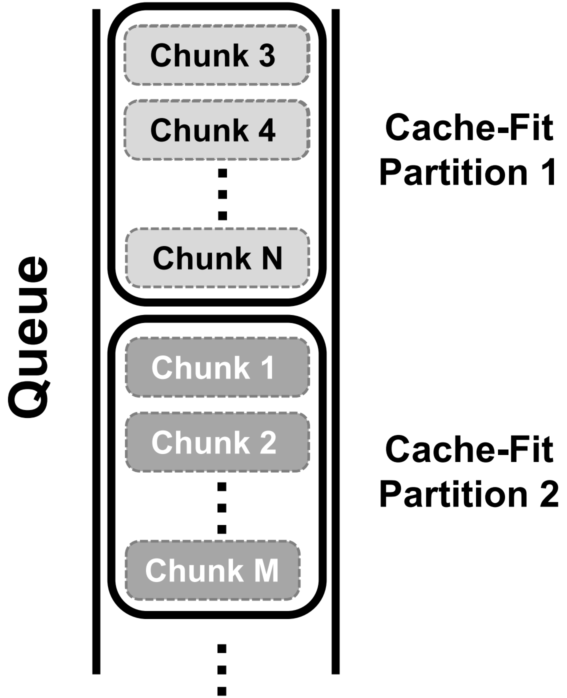
Cache-Fit Queue (CF-Q) Scheduling

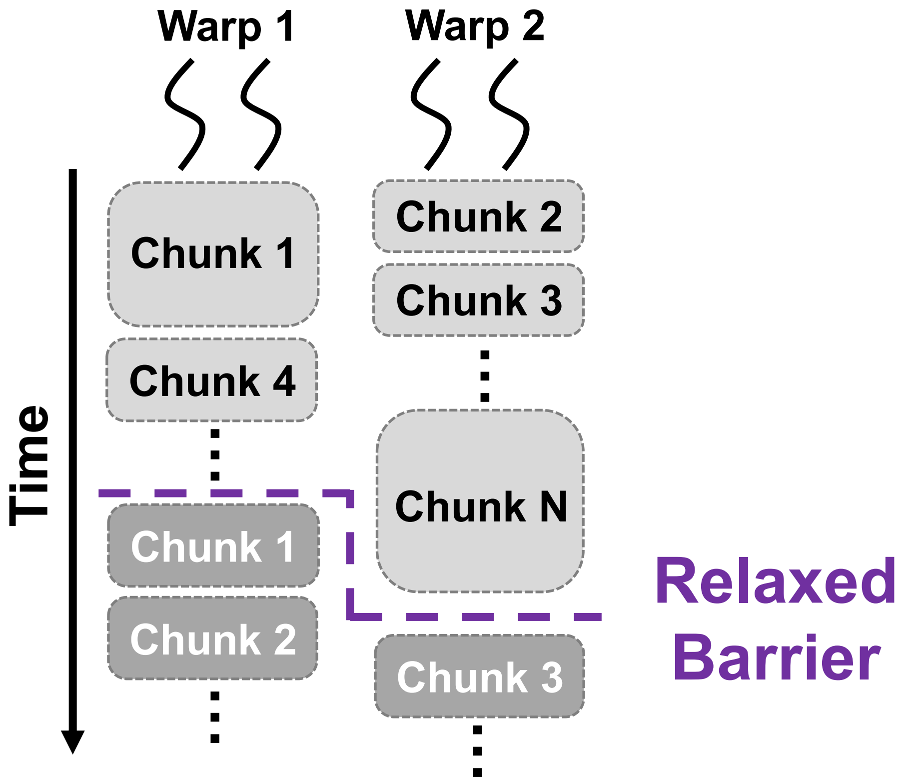
- Invoke a **single** kernel call but still enable **throttling**

- 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.





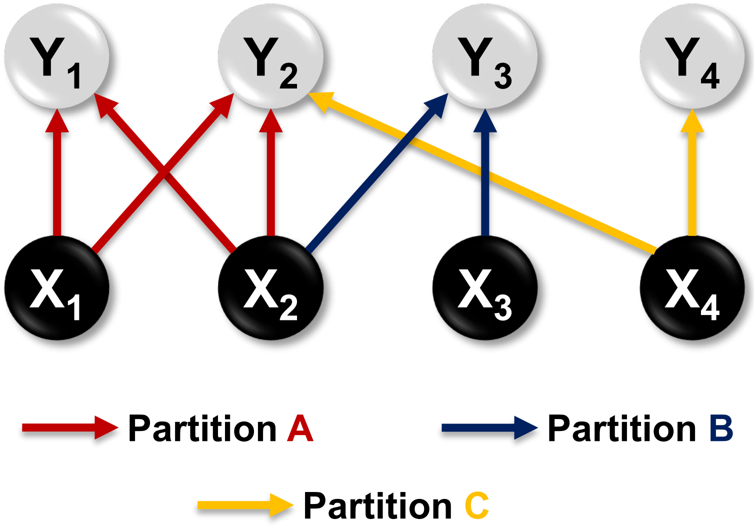
Effective scheduling methods

- 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**

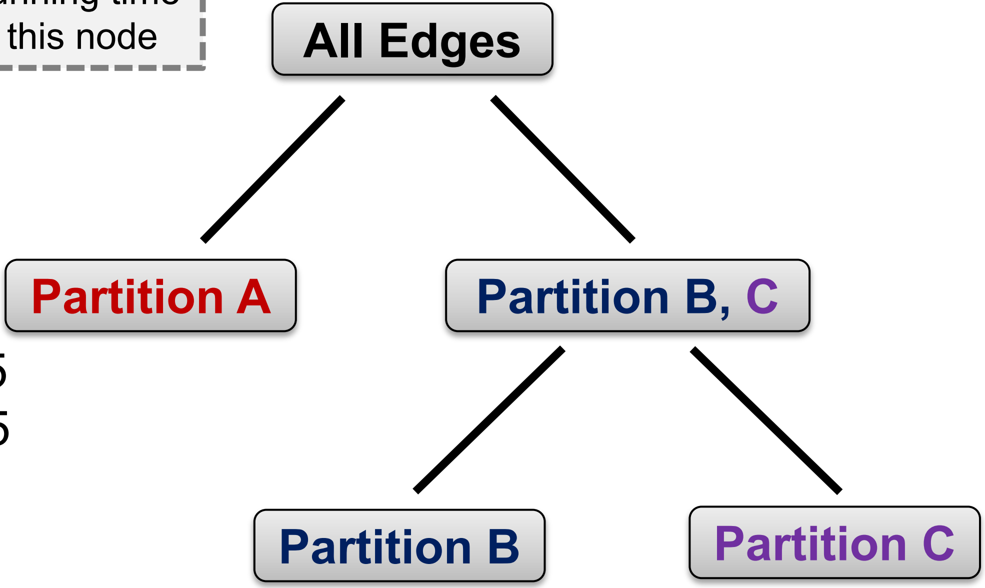
Split-Join (SJ) Scheduling



Cache Capacity: 4

Split-Join (SJ) Scheduling

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



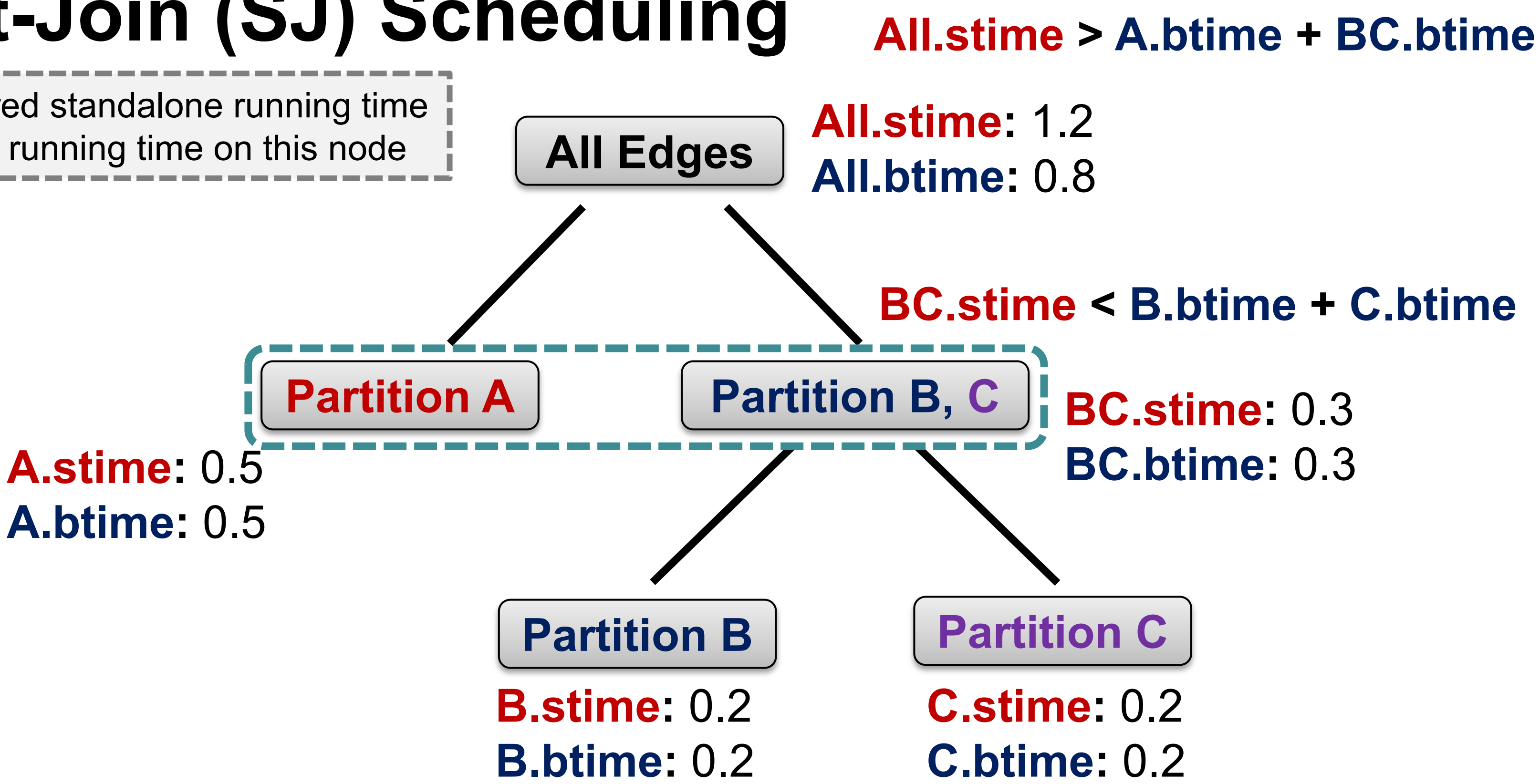
A.stime: 0.5
A.btime: 0.5

B.stime: 0.2
B.btime: 0.2

C.stime: 0.2
C.btime: 0.2

Split-Join (SJ) Scheduling

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



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

Methods	Pipeline Utilization	Profiling	Barriers	Queue	Code Change
CF	Low	No	Strict	No	No
CF-Q	High	No	Relaxed	Yes	Yes
SJ	High	Yes	Strict	No	No
SJ-Q	High	Yes	Strict / Relaxed	Yes	Yes

Software Throttling Performance

- Experiment Settings

GPU Model	Titan X	GTX 745
Architecture	Pascal	Maxwell
Core #	5376	576
L2 Cache	3MB	2MB
CUDA Version	CUDA 8.0	CUDA 8.0
CPU	Intel Xeon E5-2620	Intel Core i7-4790

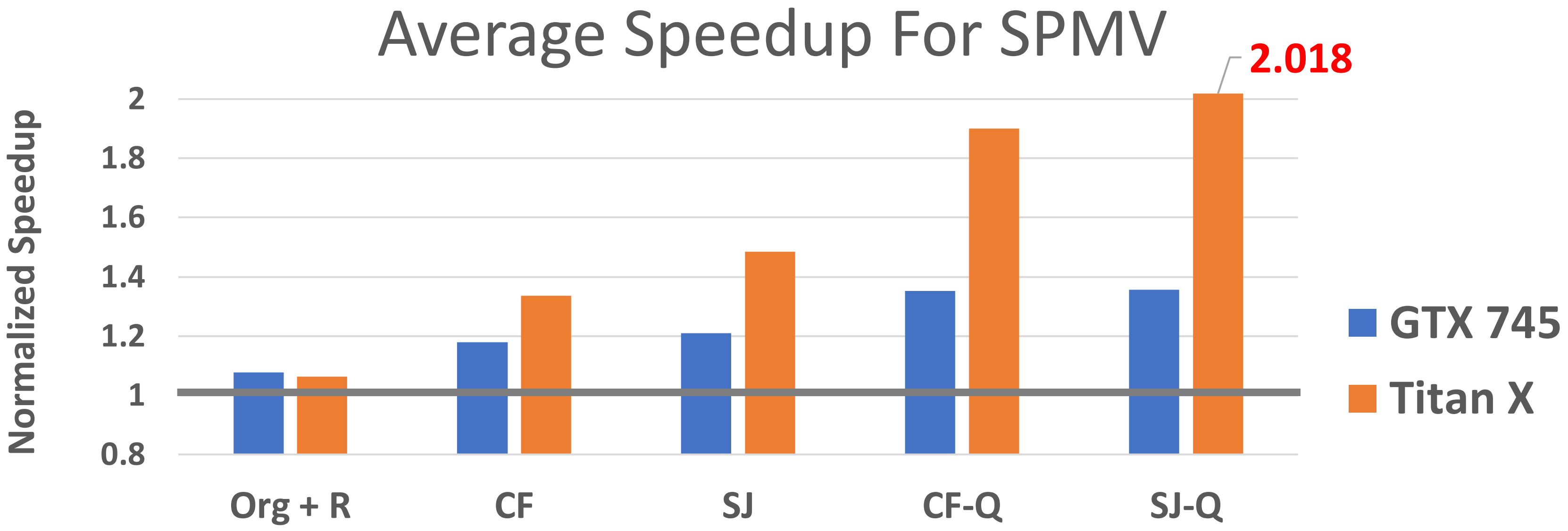
Benchmarks

- 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

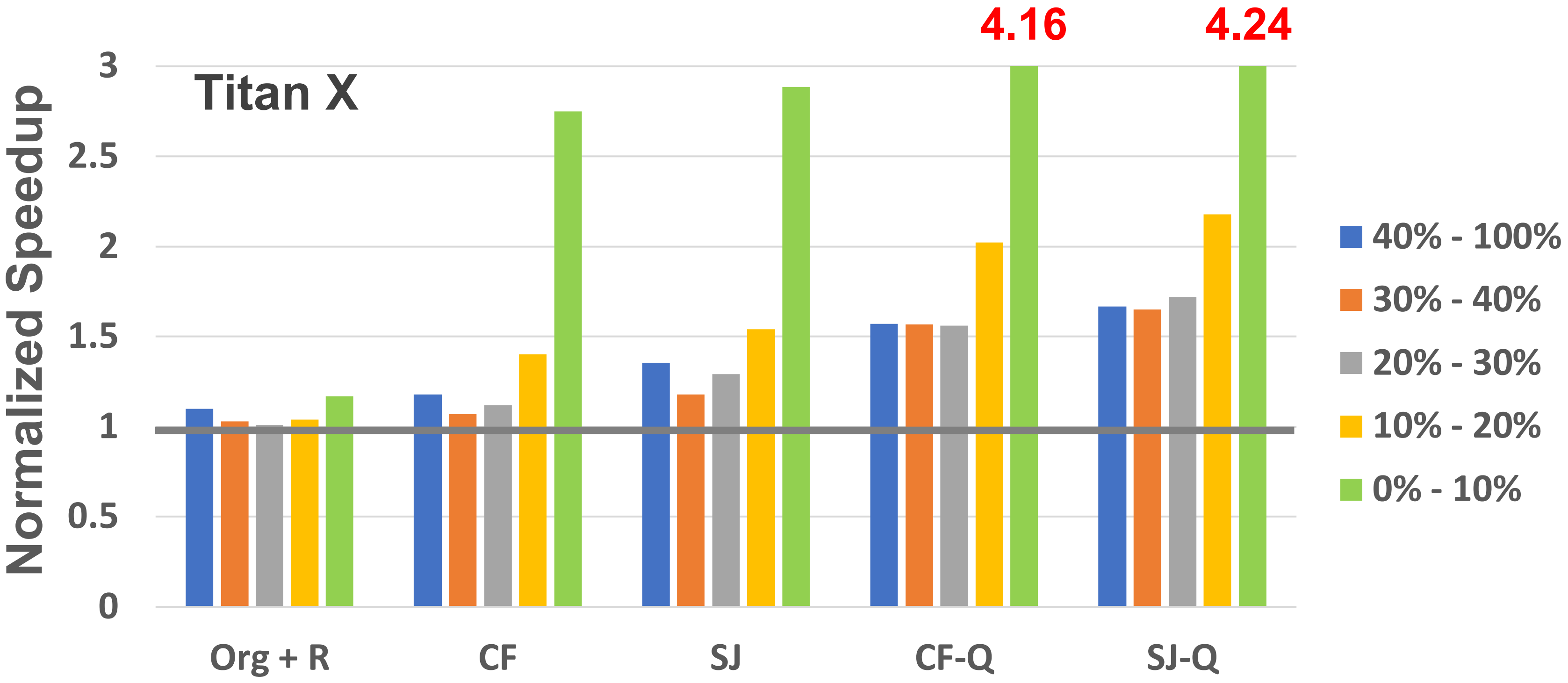
Sparse Matrix Vector Multiplication

- 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

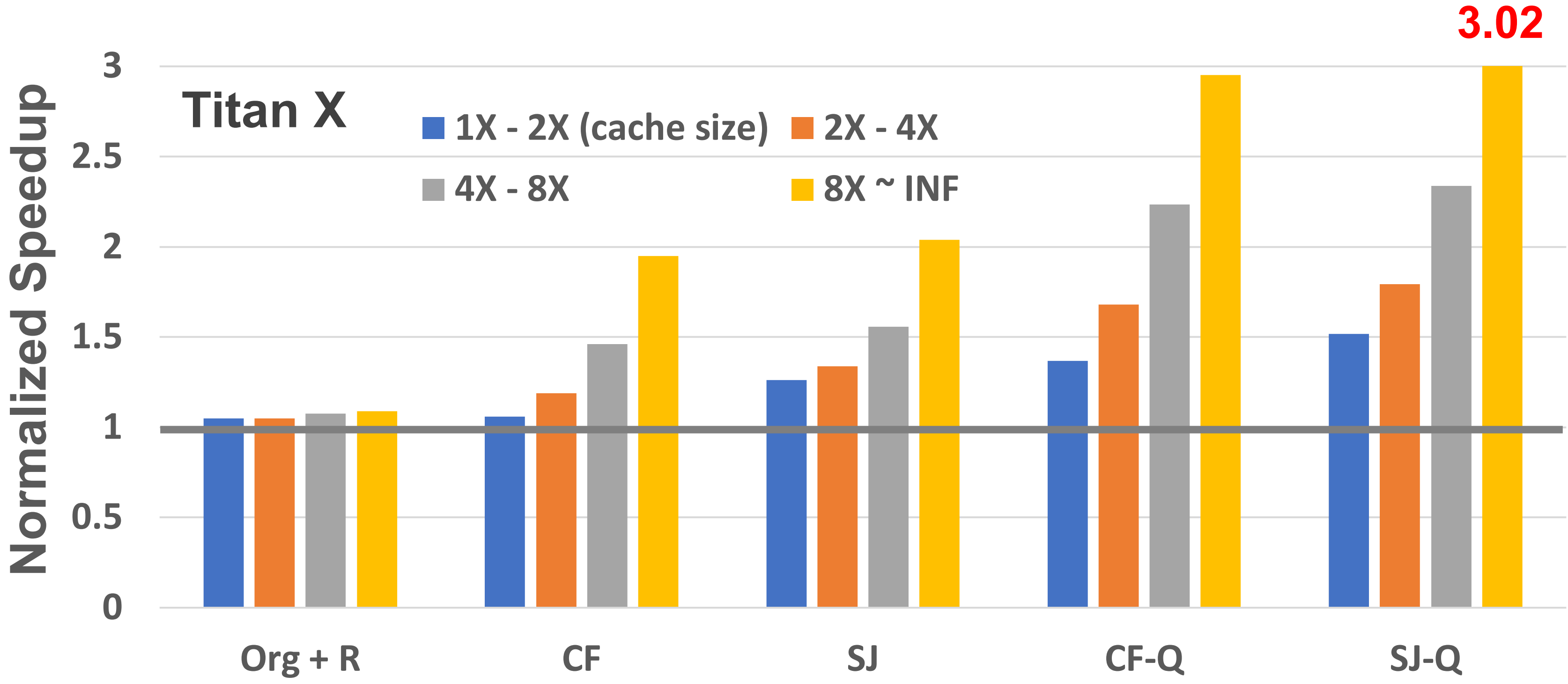
Overall SPMV Performance



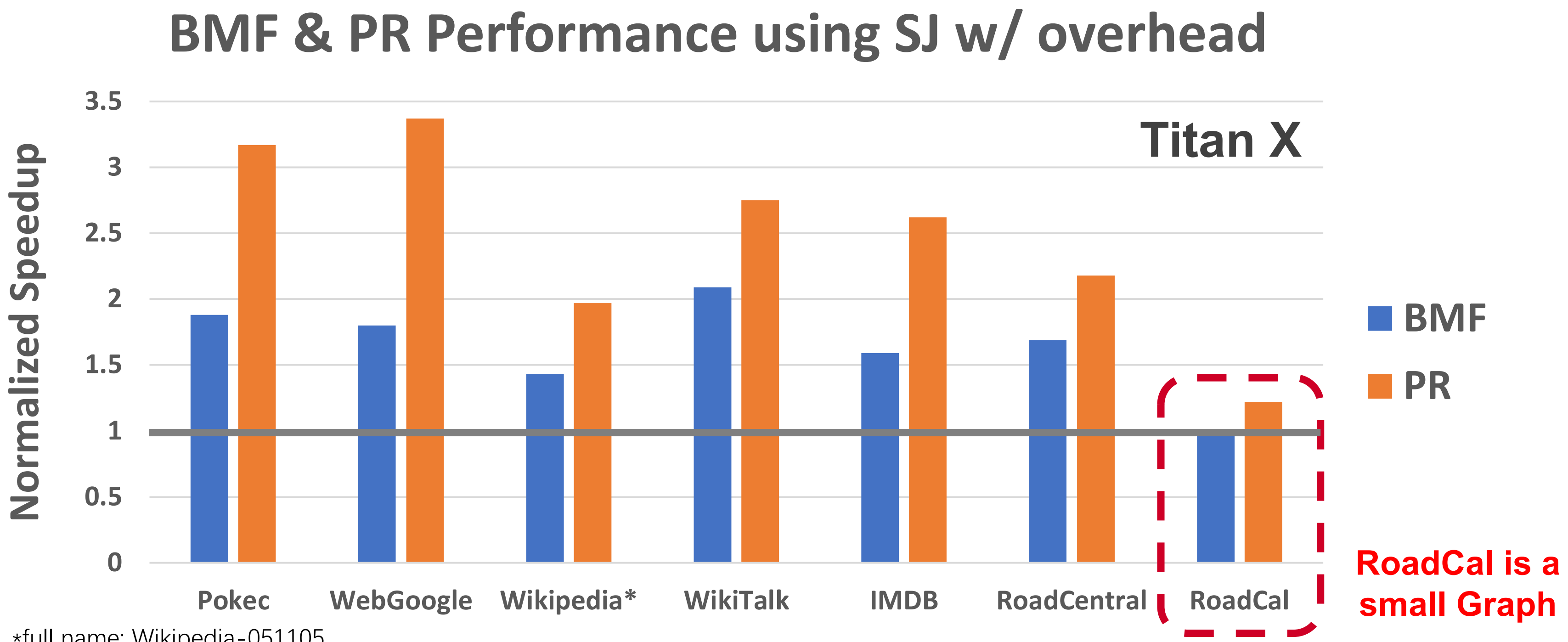
Effect of Cache Hit Rate



Effect of Working Set Size

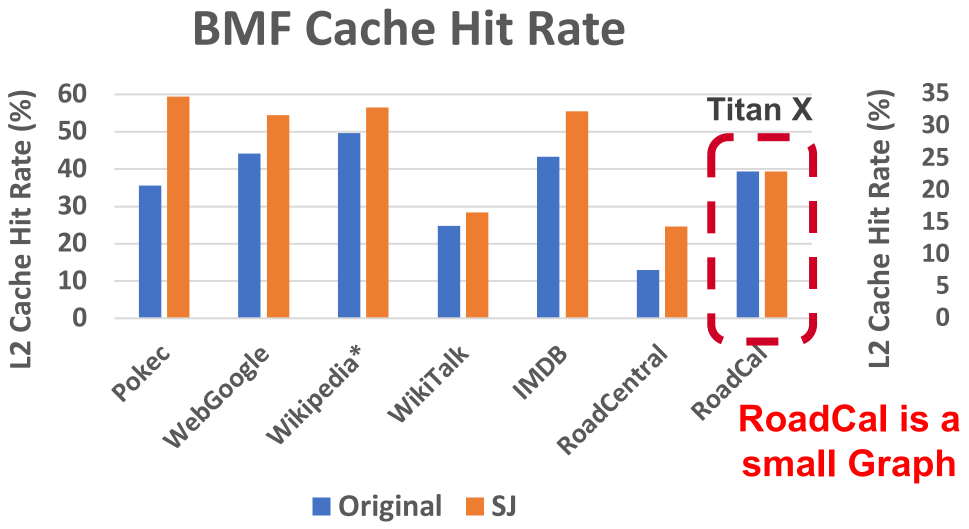


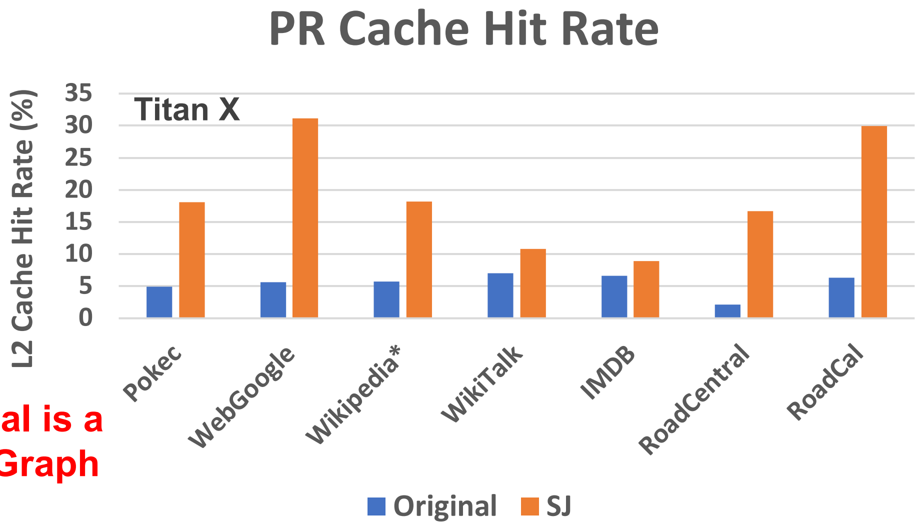
Graph Application Performance



*full name: Wikipedia-051105

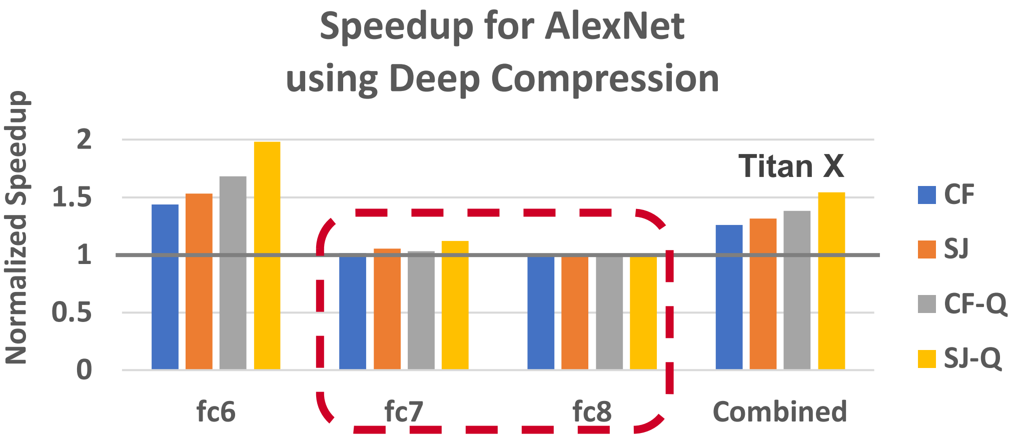
Graph Application Performance cont.





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



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?