# Fast Segmented Sort on GPUs

**Kaixi Hou†**, Weifeng Liu‡, Hao Wang†, Wu-chun Feng†

†{kaixihou, hwang121, wfeng}@vt.edu ‡ weifeng.liu@nbi.ku.dk

Virginia Tech
VIRGINIA POLYTECHNIC INSTITUTE AND STATE UNIVERSITY

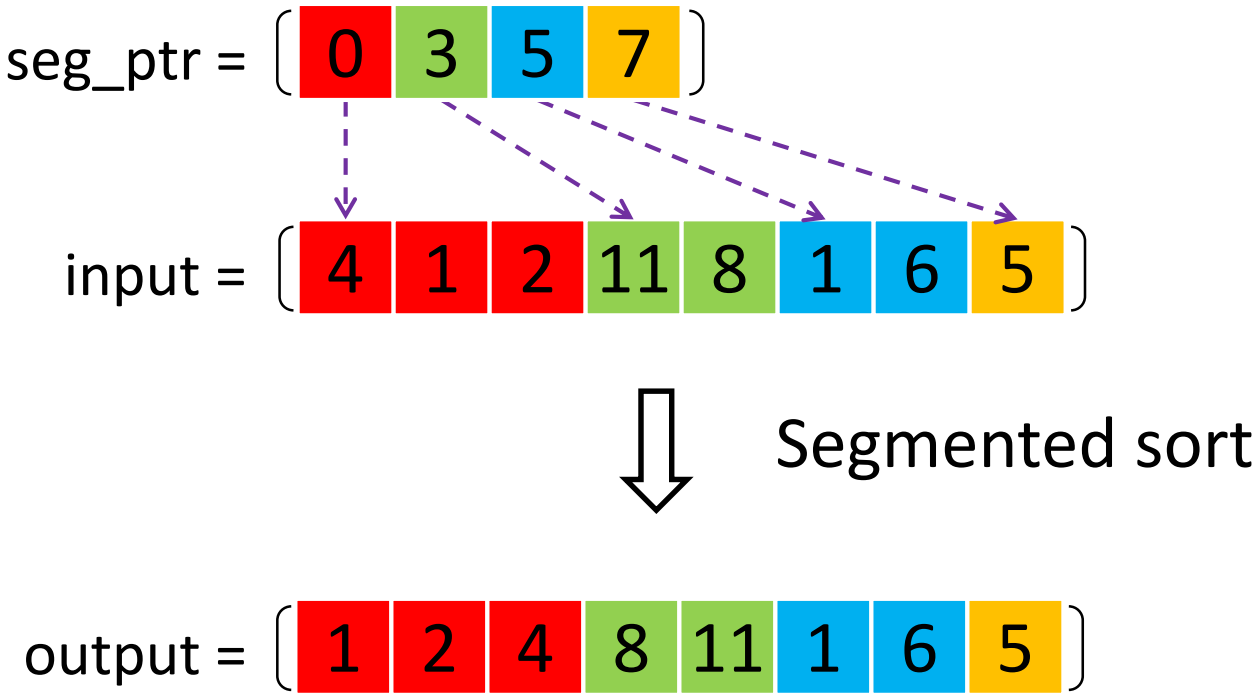
UNIVERSITY OF COPENHAGEN

NTNU
Norwegian University of Science and Technology

ACM ICS 2017, Chicago, IL, USA

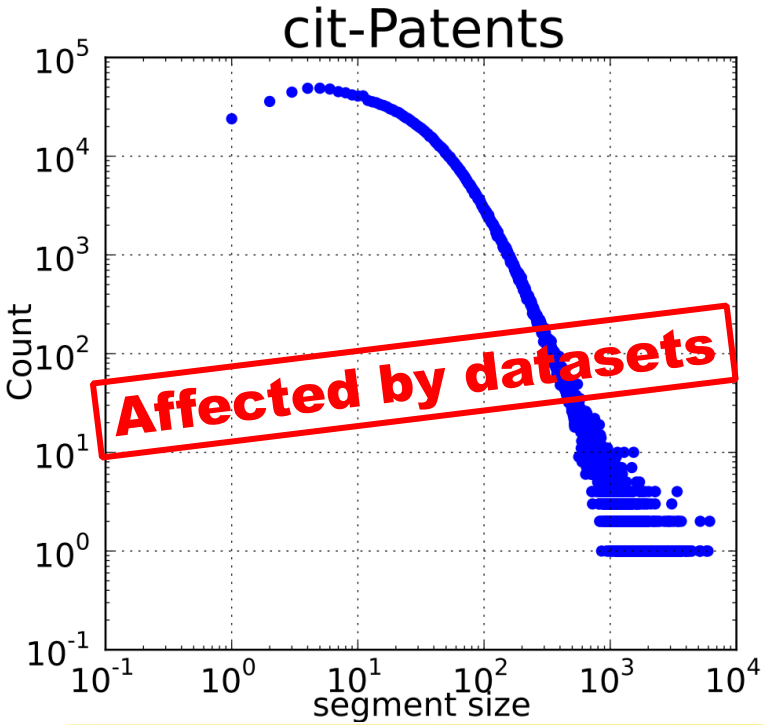
# Segmented Sort (*SegSort*)

- Perform a segment-by-segment sort on a given array composed of multiple segments

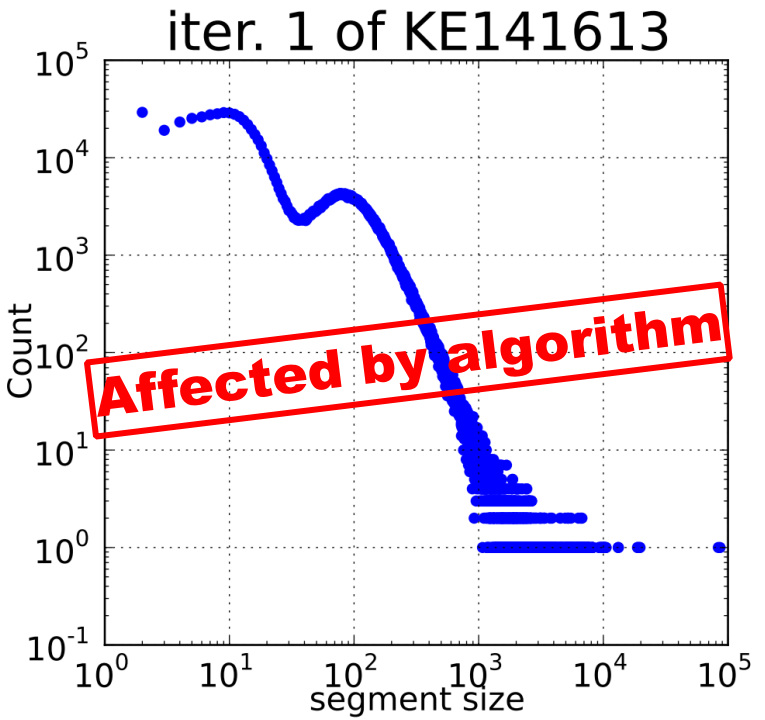


# Why Segmented Sort?

- Many applications need to process (e.g., sort) a large amount of independent arrays, due to: (1) dataset properties, (2) algorithm characteristics




Segment statistics from squaring one matrix in SpGEMM*

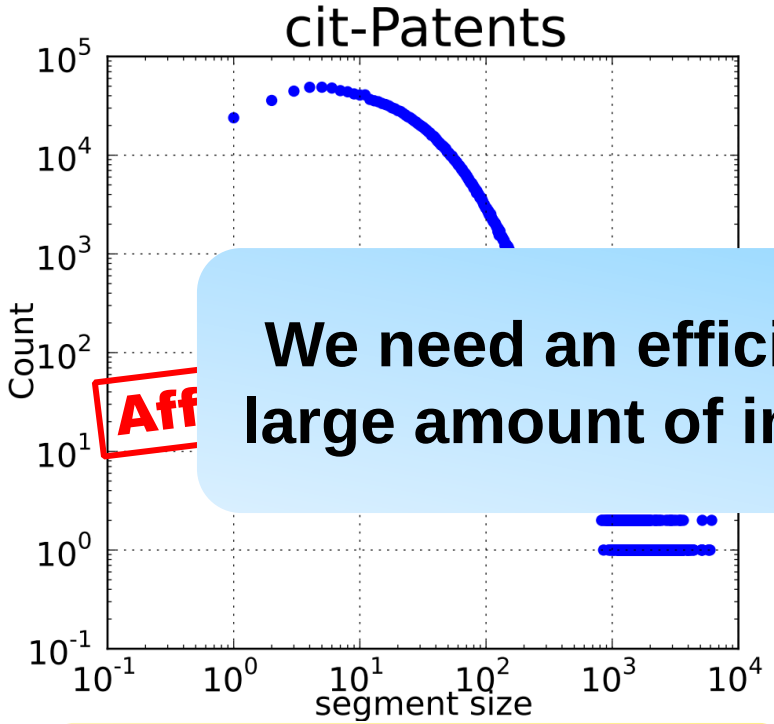


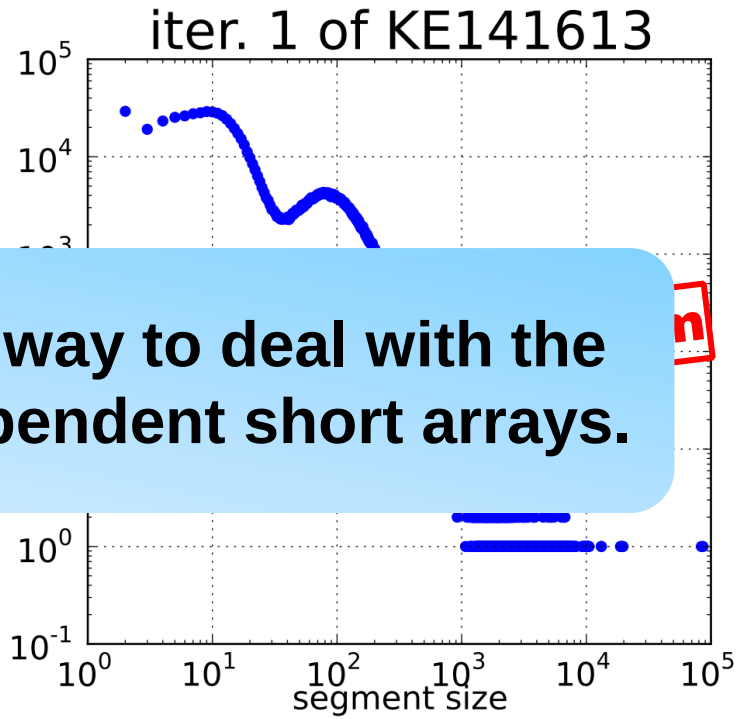

Segment statistics from 1st iteration in SAC*

* SpGEMM: Sparse General Matrix-Matrix Multiplication; SAC: Suffix Array Construction

# Why Segmented Sort?

- Many applications need to process (e.g., sort) a large amount of independent arrays, due to: (1) dataset properties, (2) algorithm characteristics




**We need an efficient way to deal with the large amount of independent short arrays.**

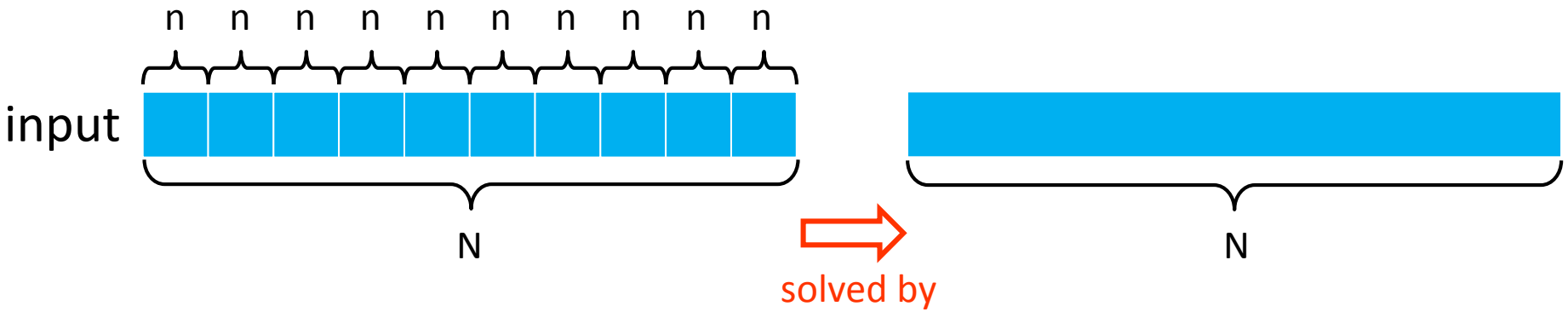
Segment statistics from squaring one matrix in SpGEMM*

Segment statistics from 1st iteration in SAC*

* SpGEMM: Sparse General Matrix-Matrix Multiplication; SAC: Suffix Array Construction

# Existing Segmented Sort

- Global sort has received much more fanfare!
- Many tools are evolved from global sort; however, there are also problems
  - Problem 1: **Time complexity**




The complexity of this segsort is $O\left(\frac{N}{n}nlogn\right) \approx O(Nlogn)^*$

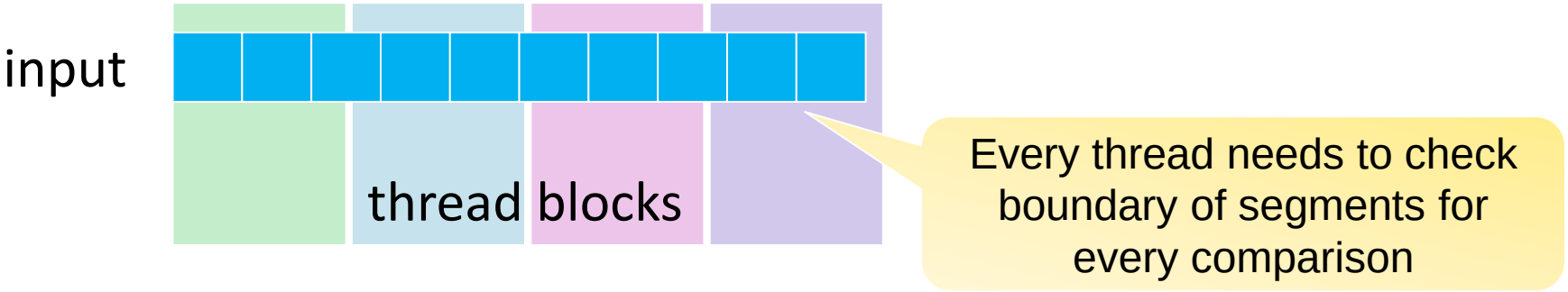
The complexity of the global sort is $O(NlogN)^*$

**SegSort, evolved from global sort, usually exhibits higher complexity, e.g., segsort from *modernGPU* and *CUSP***

* For generality, the sorting algorithms are all comparison-based.

# Existing Segmented Sort

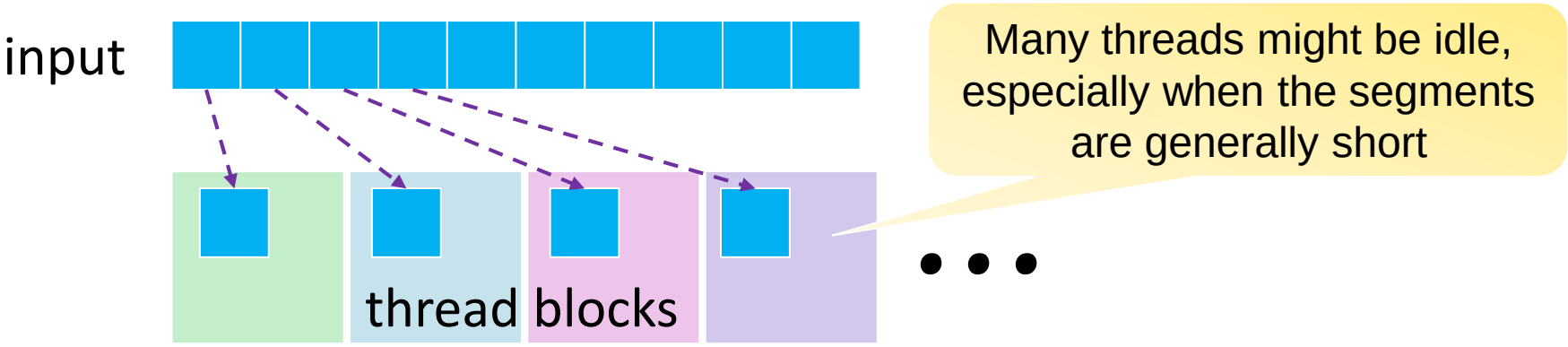
- Global sort has received much more fanfare!
- Many tools are evolved from global sort; however, there are also problems
  - Problem 1: Time complexity
  - Problem 2: **Runtime boundary checking overhead**




**Some SegSort needs to perform runtime boundary checking, causing additional overhead, e.g., segsort from *modernGPU***

# Existing Segmented Sort

- Global sort has received much more fanfare!
- Many tools are evolved from global sort; however, there are also problems
  - Problem 1: Time complexity
  - Problem 2: Runtime boundary checking overhead
  - Problem 3: **Underutilized resources**

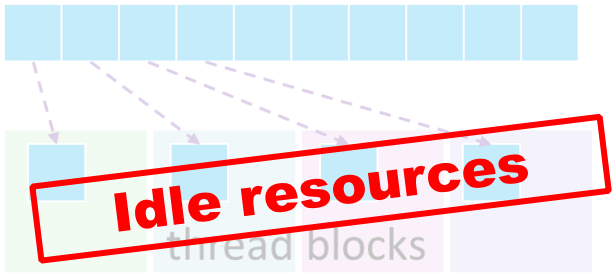



**Some SegSort simply assigns each segment to each thread block, leading to idle resources, e.g., segsort from *CUB***

# Fast Segmented Sort (this work)


**High complexity**


**Checking overhead**

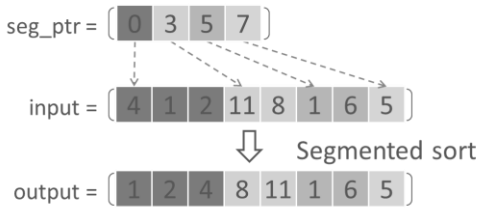


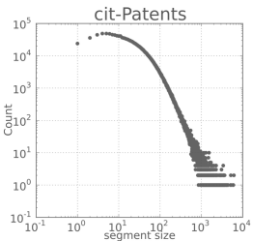

**Idle resources**

**We propose an adaptive segmented sort mechanism for GPUs: (1) differentiated methods for different segments, (2) an algorithm supporting variable data-thread binding and thread communication.**

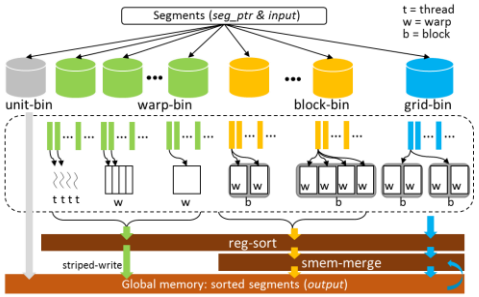
# Outline

- Introduction
- Motivation

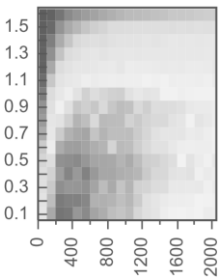


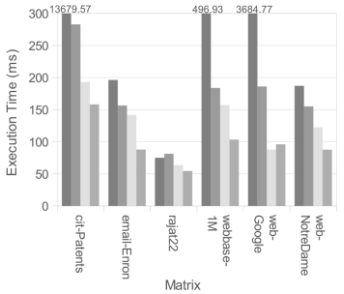




- Our Method
  - GPU SegSort Mechanism
  - GPU Register-based Sort
  - Other Techniques & Opt.




- Evaluation
  - Kernel Performance
  - Kernel in Real Applications

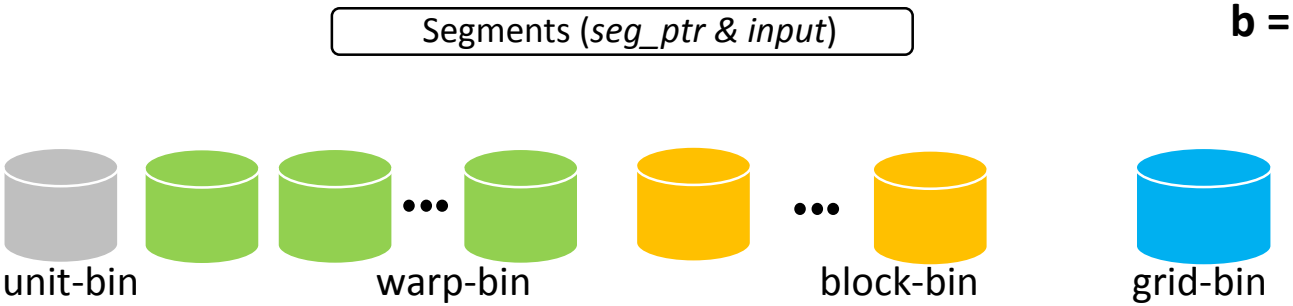




# Adaptive GPU SegSort Mechanism

- Overview of our proposed GPU SegSort design

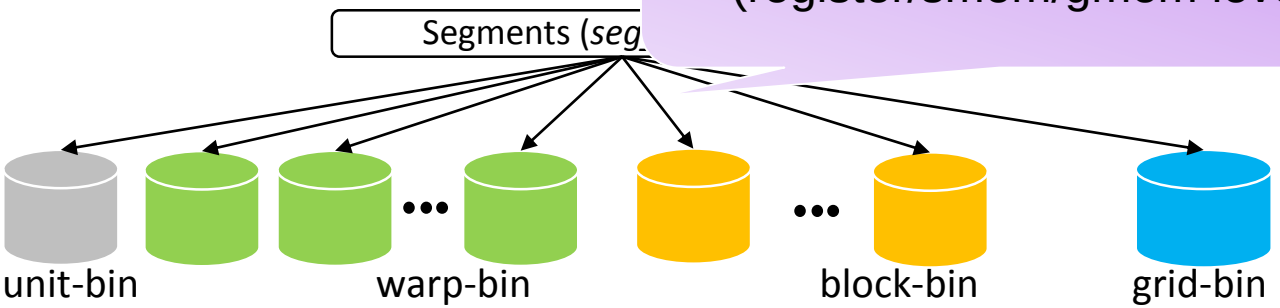
**t = thread**
**w = warp**
**b = block**



# Adaptive GPU SegSort Mechanism

- Overview of our proposed (

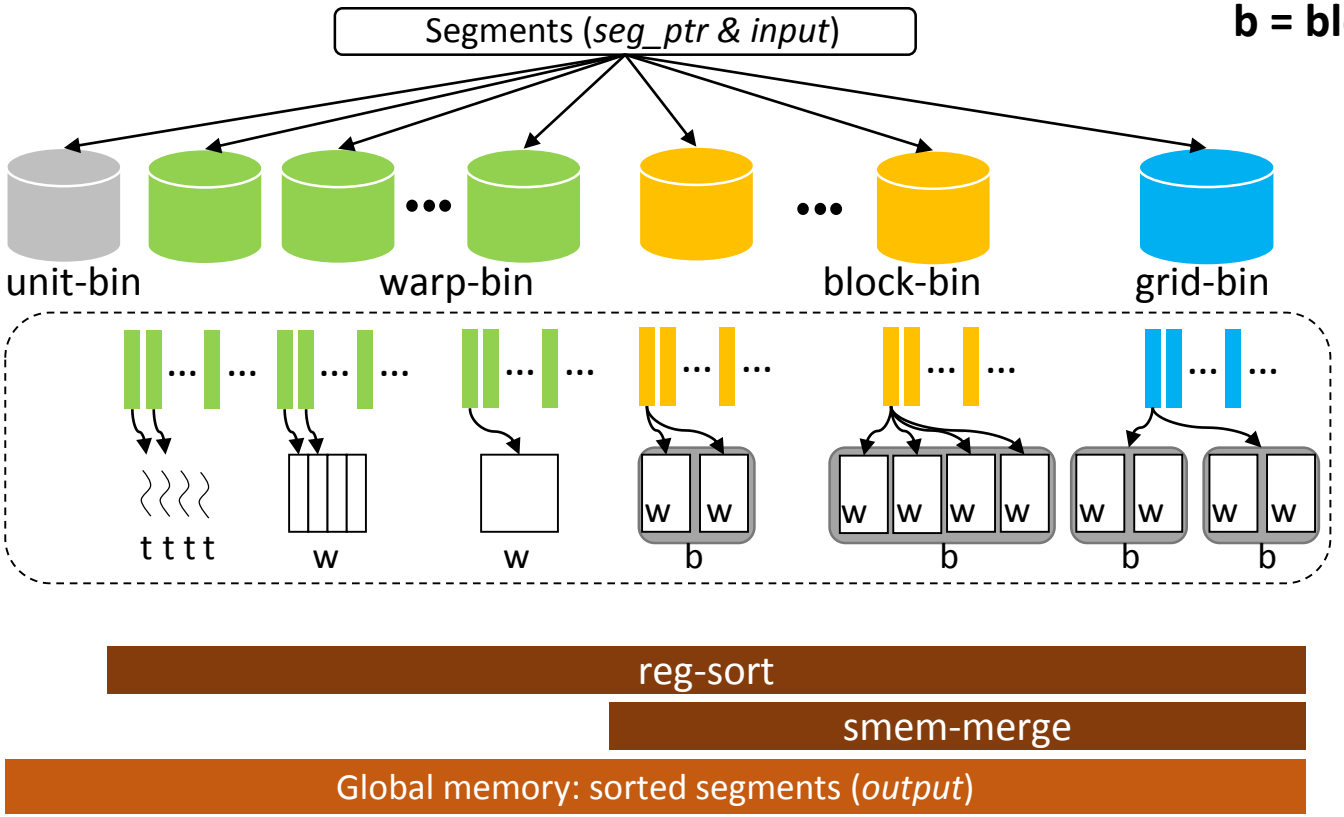
- Hierarchical binning (register/smem/gmem levels)



# Adaptive GPU SegSort Mechanism

- Overview of our proposed GPU SegSort design

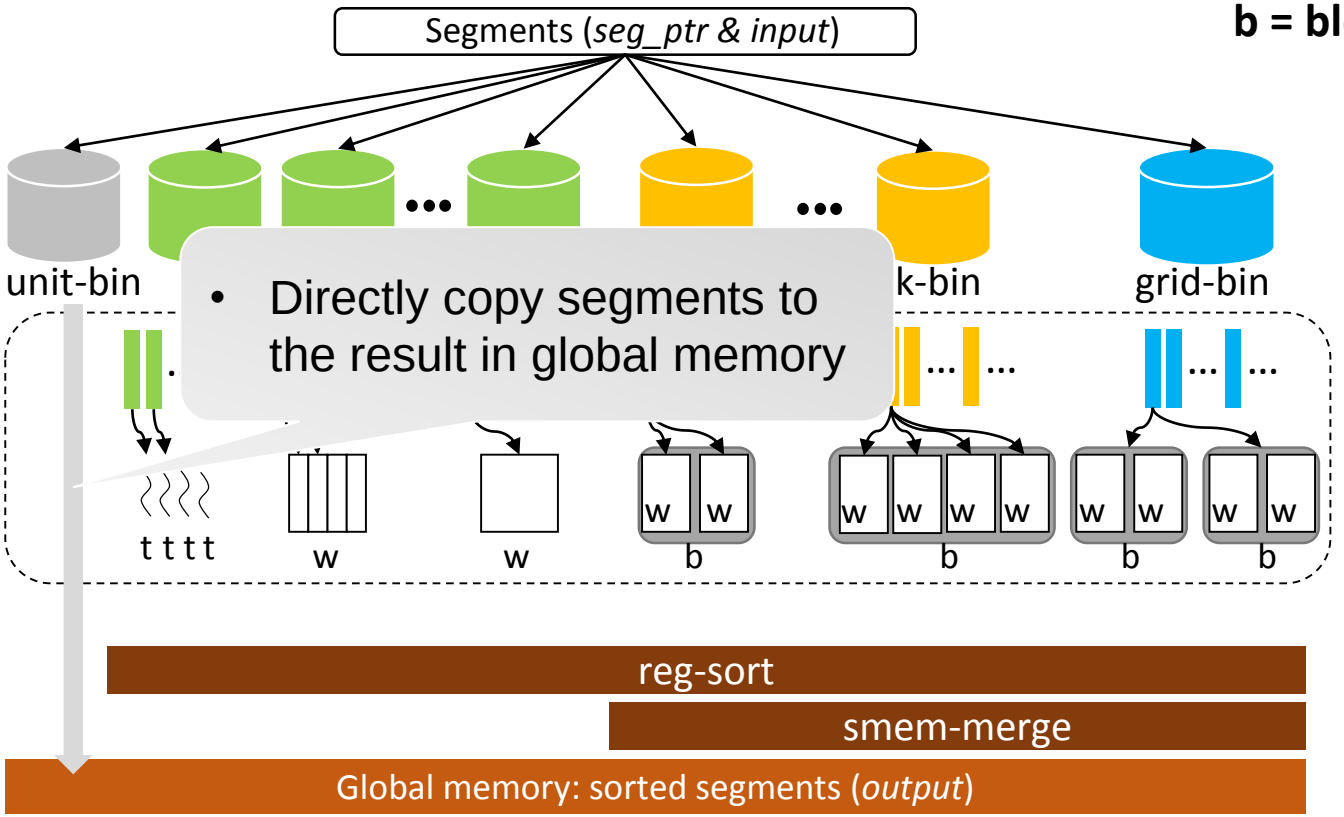
**t = thread**
**w = warp**
**b = block**



# Adaptive GPU SegSort Mechanism

- Overview of our proposed GPU SegSort design

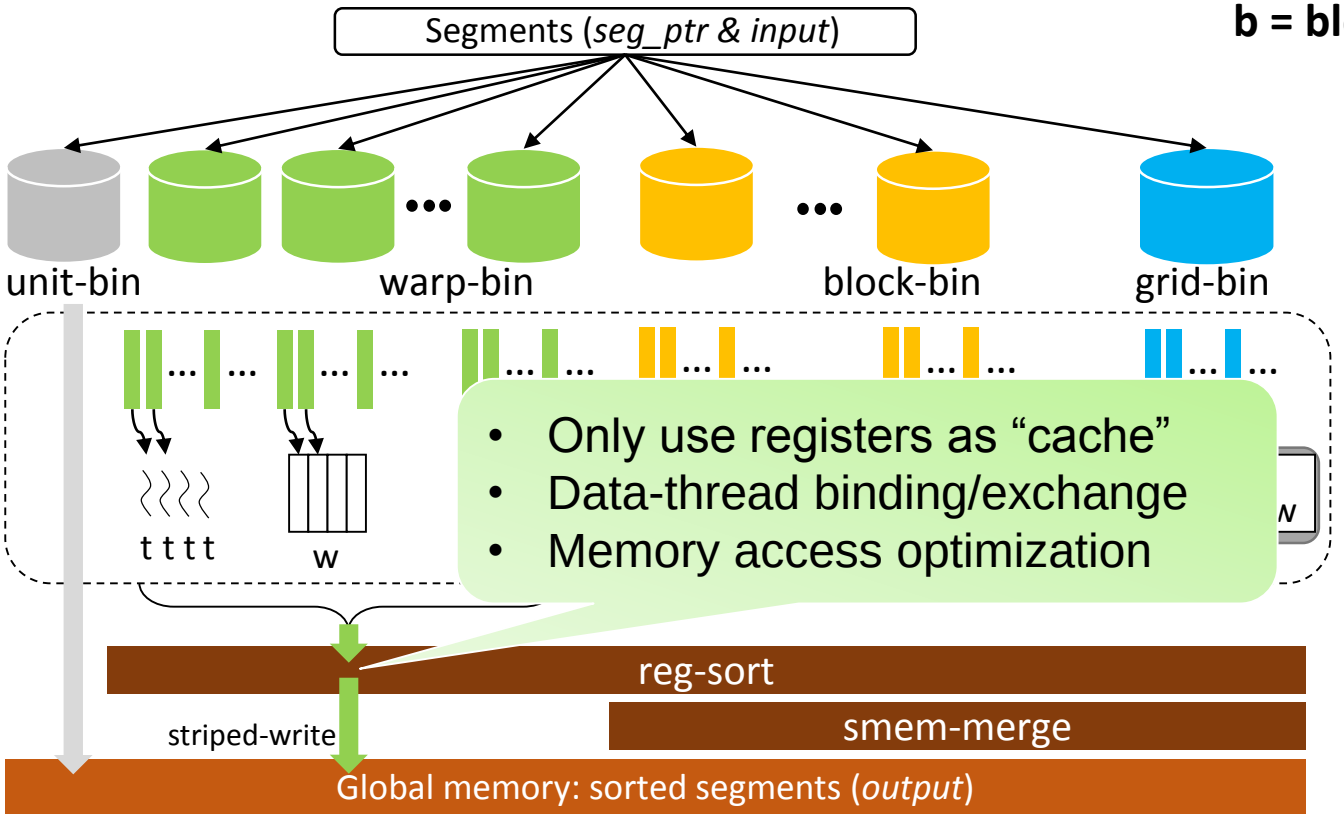
**t = thread**
**w = warp**
**b = block**



# Adaptive GPU SegSort Mechanism

- Overview of our proposed GPU SegSort design

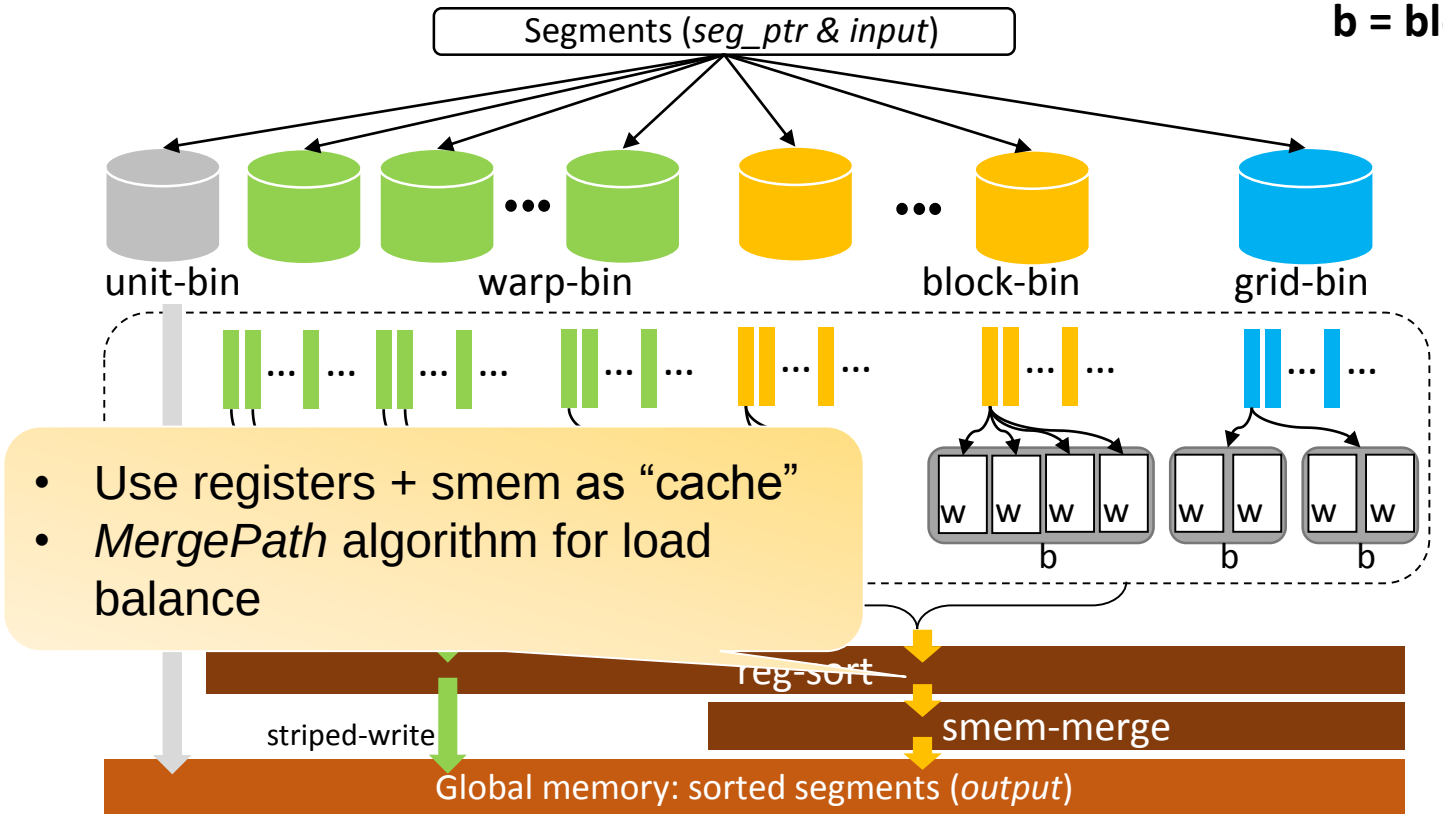
**t = thread**
**w = warp**
**b = block**



# Adaptive GPU SegSort Mechanism

- Overview of our proposed GPU SegSort design

**t = thread**
**w = warp**
**b = block**

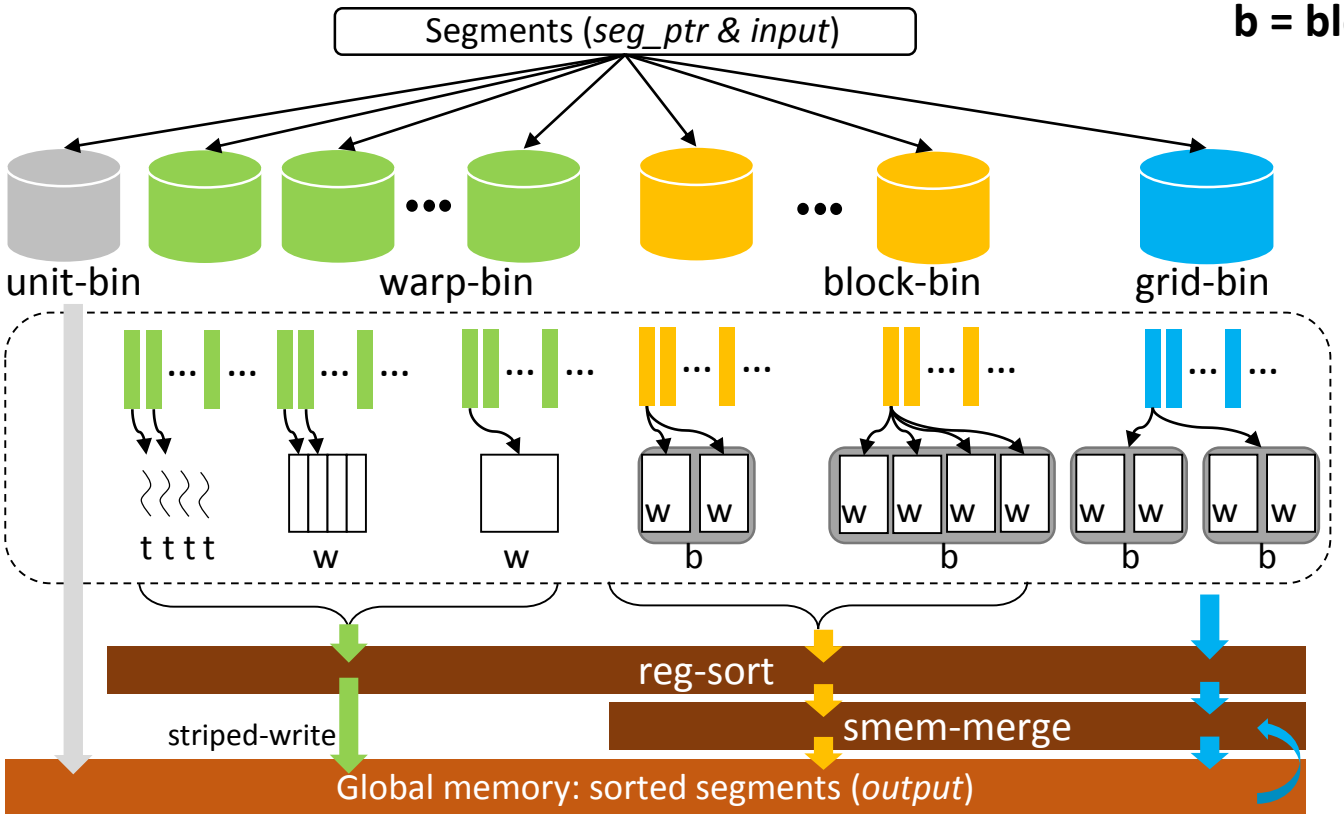



- Use registers + smem as "cache"
- *MergePath* algorithm for load balance

# Adaptive GPU SegSort Mechanism

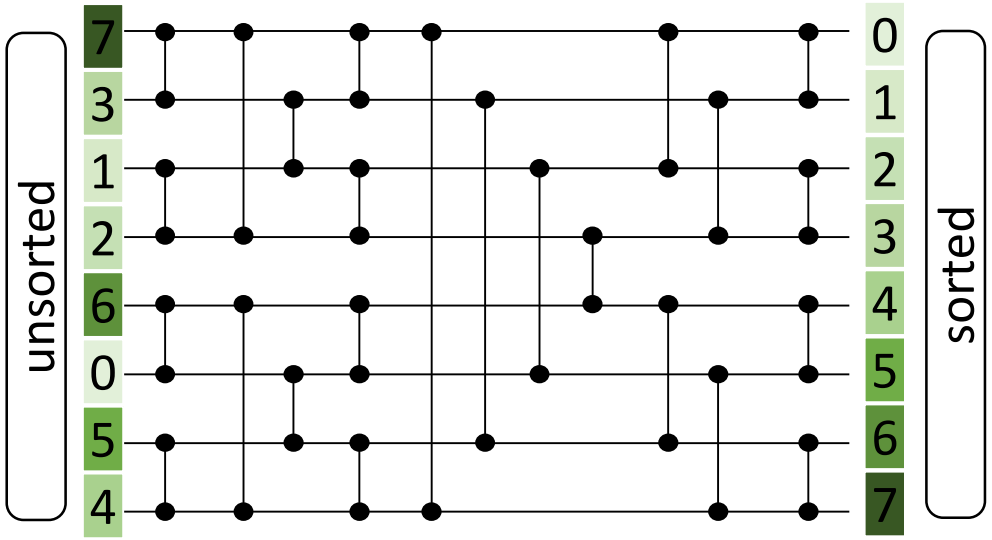
- Overview of our proposed GPU SegSort design

**t = thread**
**w = warp**
**b = block**



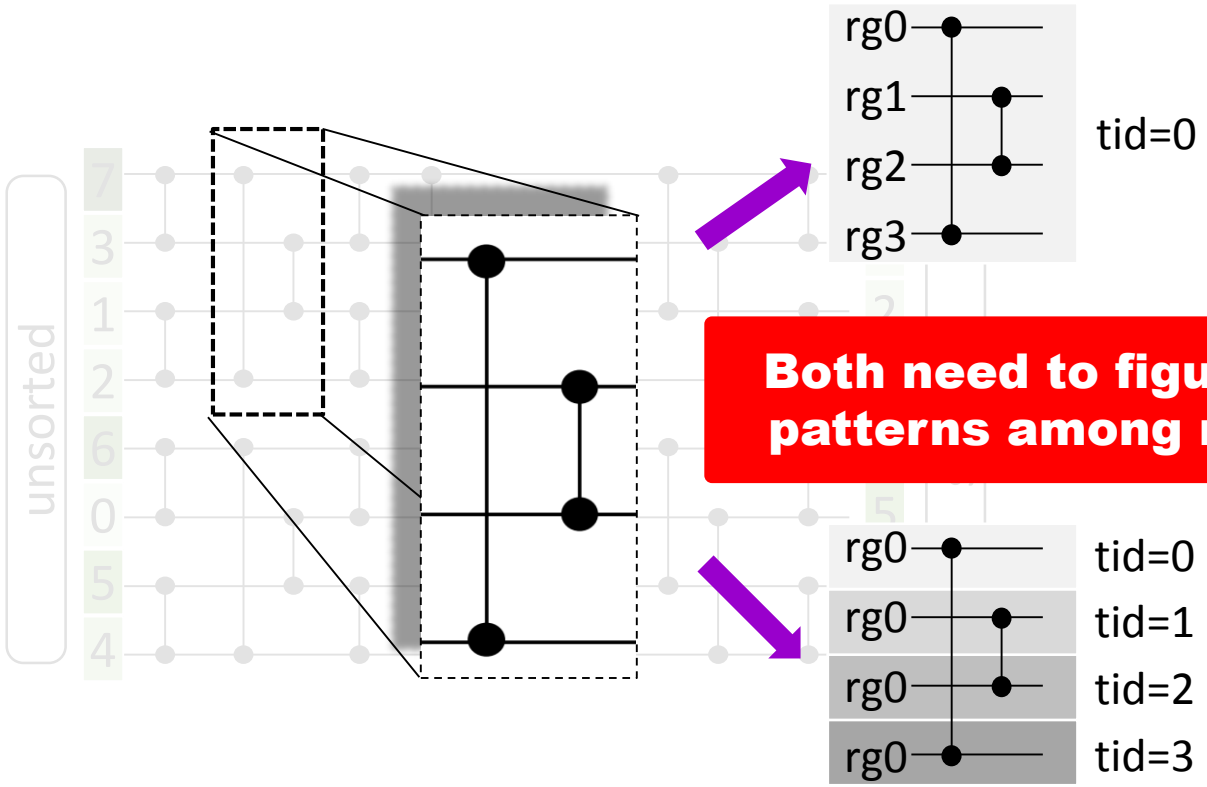
# GPU Register-based Sort

- **Sorting networks** usually serve as building blocks of efficient parallel sort
- How to bind the data items (operands) to different threads?



# GPU Register-based Sort

- **Sorting networks** usually serve as building blocks of efficient parallel sort
- How to bind the data items (operands) to different threads?




Each thread uses too many registers (**high register pressure**)

**Both need to figure out data exchange patterns among registers and threads**

Each comparison is performed twice (**wasted computing resources**)

# GPU Register-based Sort

- Propose **a general way** to solve the data-thread binding problem at GPU register level
- **Primitive pattern**

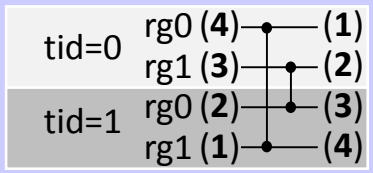
Two data items are bound to each thread

Tells which thread swaps registers

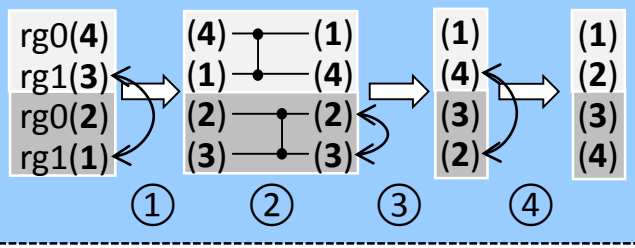
**_exch_primitive(rg0, rg1, tmask, swbit)**

Tells how threads communicate

*_exch_primtive(rg0,rg1,**0x1**,**0**)*




*Implementation Details*




① _shuf_xor(*rg1*, **0x1**); // Shuffle data in *rg1*
② cmp_swp(*rg0*, *rg1*); // Compare data of *rg0* & *rg1* locally
③ if( bfe(tid,**0**) ) swp(*rg0*, *rg1*); // Swap data of *rg0* & *rg1* if **0** bit of tid is set
④ _shuf_xor(*rg1*, **0x1**); // Shuffle data in *rg1*

# GPU Register-based Sort

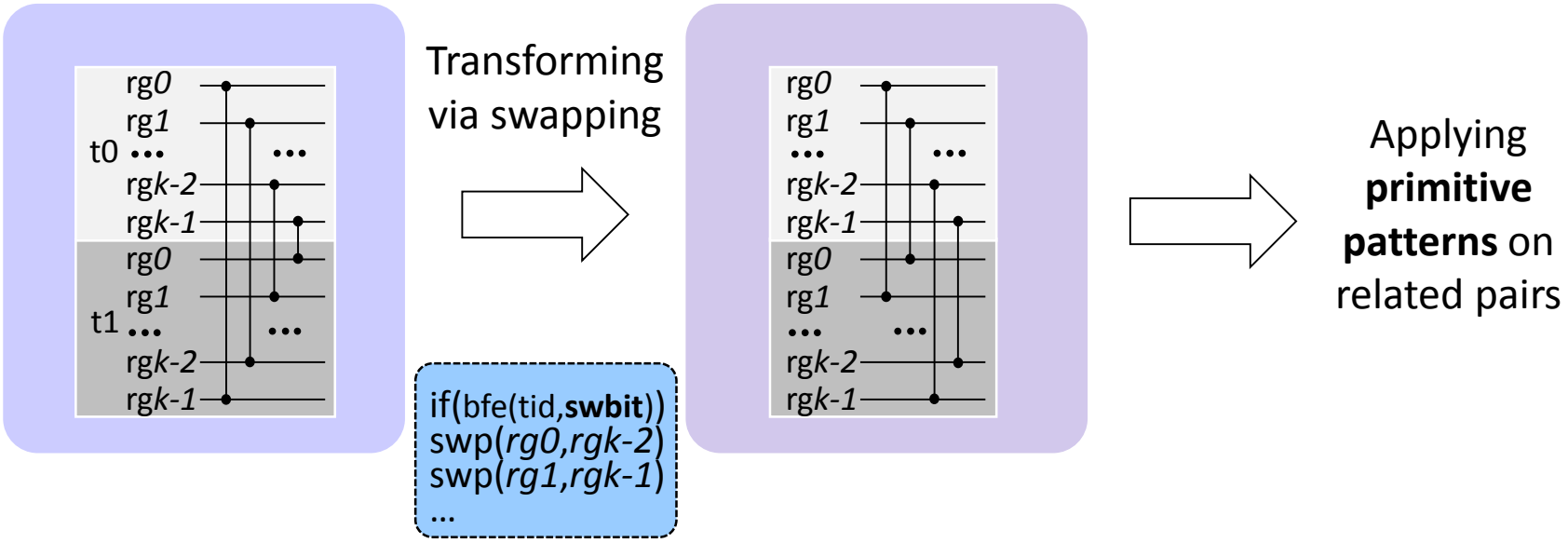
- Other patterns, then, can be solved by transformation and the primitive patterns
- **Intersecting Pattern**

Any number of data items are bound to each thread

Tells which thread swaps registers

**_exch_intxn(rg0, rg1, …, rgk-1, tmask, swbit)**

Tells how threads communicate



# GPU Register-based Sort

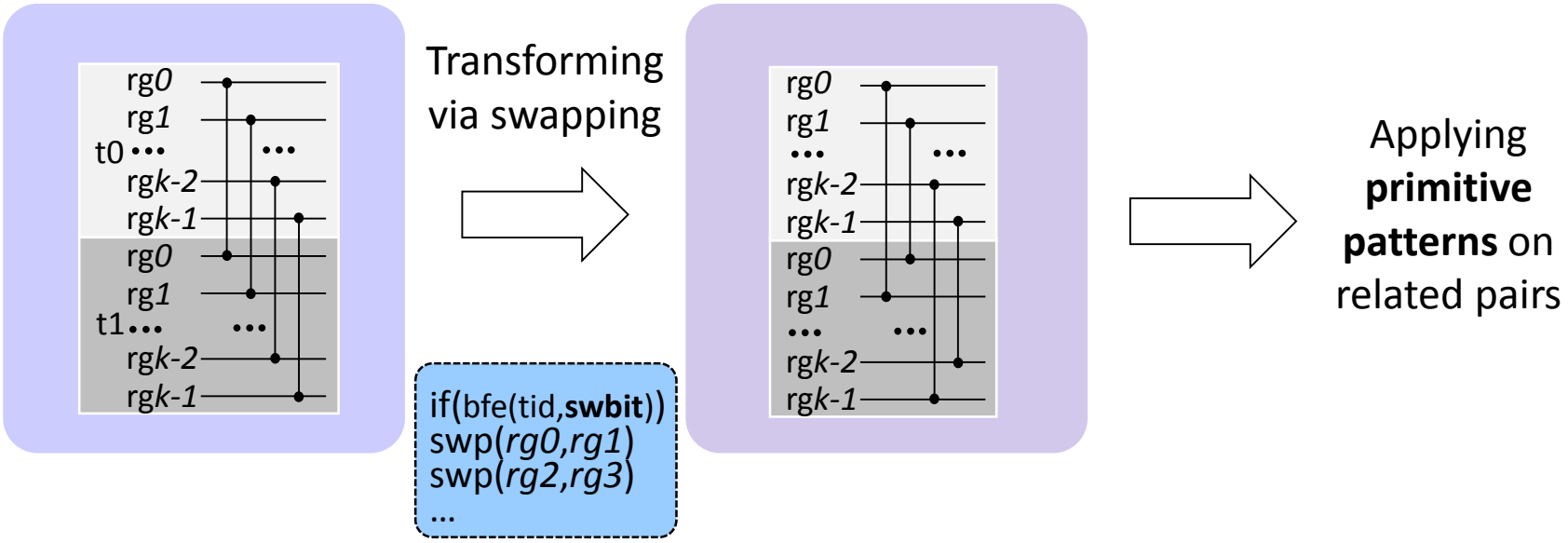
- Other patterns, then, can be solved by transformation and the primitive patterns
- **Parallel Pattern**

Any number of data items are bound to each thread

Tells which thread swaps registers

**_exch_paral(rg0, rg1, …, rgk-1, tmask, swbit)**

Tells how threads communicate



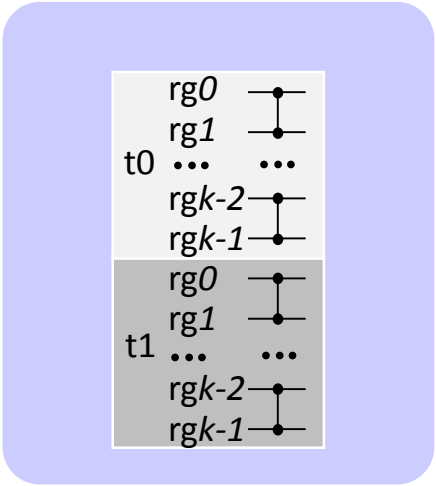
# GPU Register-based Sort

- Also, we can solve patterns without thread communication
  - "Communication" only occurs between registers
- **Local Pattern**

Any number of data items are bound to each thread

**_exch_local(rg0, rg1, …, rgk-1, rmask)**

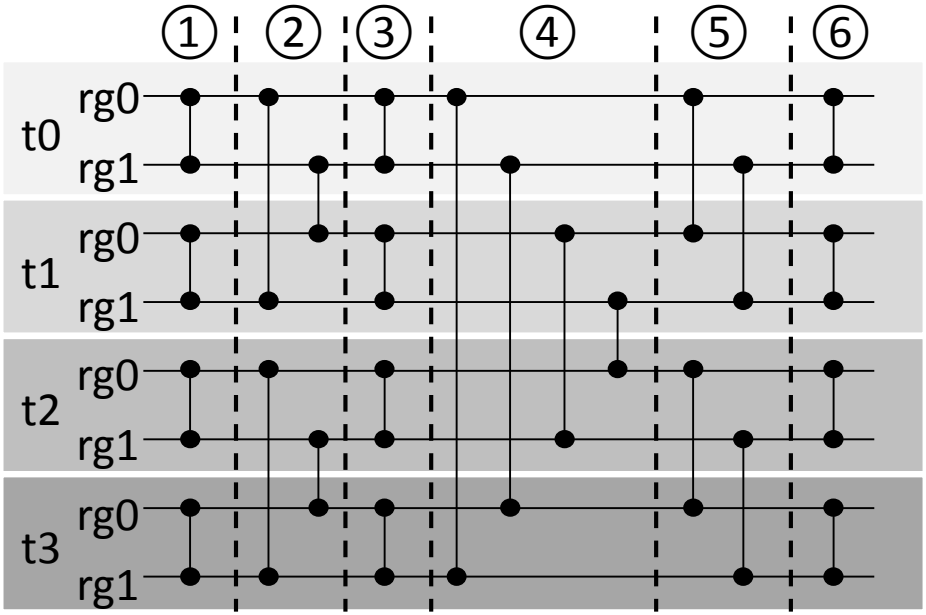
Tells how registers compare with each other



# GPU Register-based Sort: An Example

- Represent the sorting network by using our generalized patterns

**reg_sort(data items=8, thread num=4)**




①_exch_local(*rg0*,*rg1*);
②_exch_intxn(*rg0*,*rg1*,**0x1**,**0**);
③_exch_local(*rg0*,*rg1*);
④_exch_intxn(*rg0*,*rg1*,**0x3**,**1**);
⑤_exch_paral(*rg0*,*rg1*,**0x1**,**0**);
⑥_exch_local(*rg0*, *rg1*);

**Read our paper and see more details of (1) how to automatically decide which patterns to use, (2) how to order the patterns, (3) how to compute the parameters (e.g., tmask)**

# Other Techniques & Optimizations

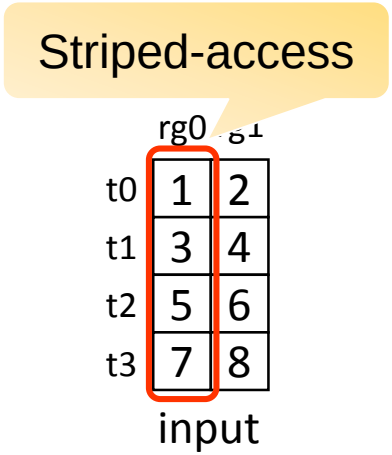
- A hierarchical binning
  - Using warp vote function __balloc() and __popc() at warp level
  - Using shared memory at thread-block level
- Better locality by optimizing access pattern
  - Transforming striped write to coalesced memory access

| | rg0 | rg1 |
|---|---|---|
| t0 | 1 | 2 |
| t1 | 3 | 4 |
| t2 | 5 | 6 |
| t3 | 7 | 8 |

input

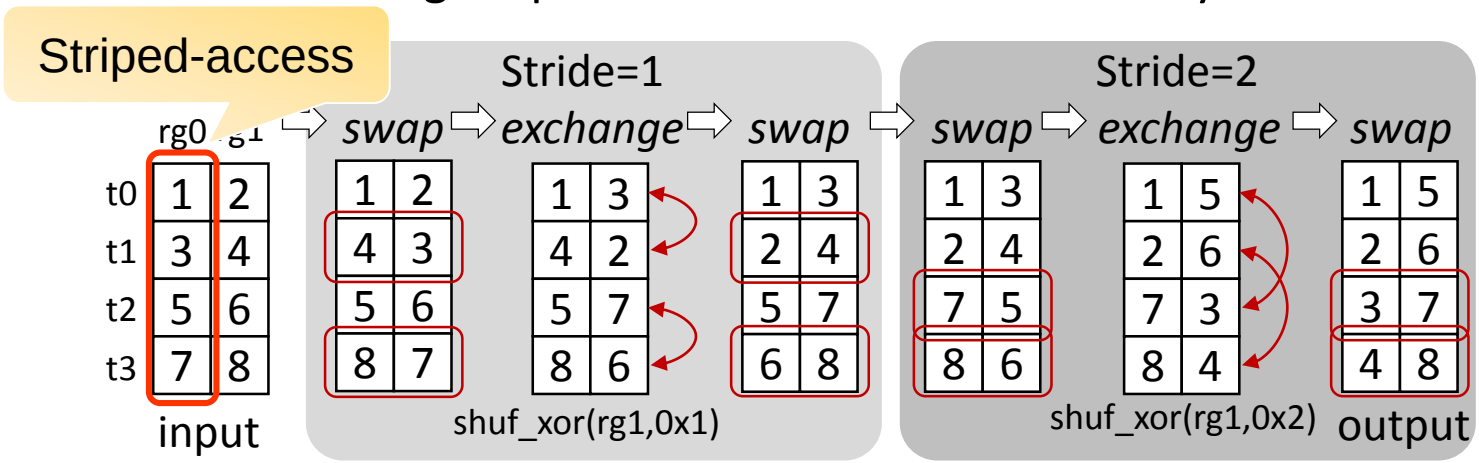
# Other Techniques & Optimizations

- A hierarchical binning
  - Using warp vote function __balloc() and __popc() at warp level
  - Using shared memory at thread-block level
- Better locality by optimizing access pattern
  - Transforming striped write to coalesced memory access



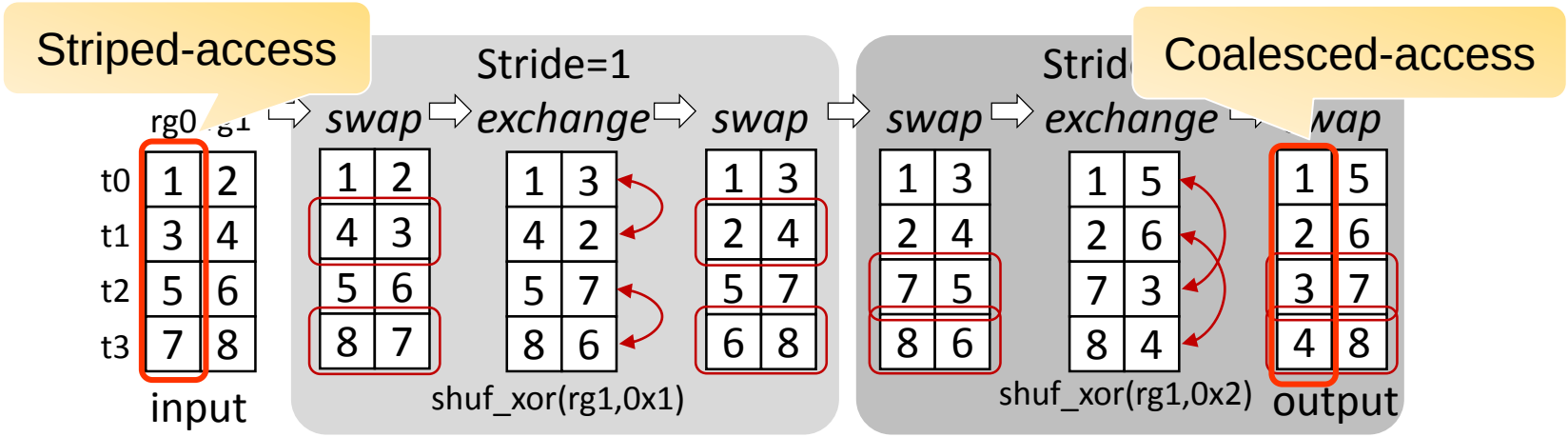
# Other Techniques & Optimizations

- A hierarchical binning
  - Using warp vote function __balloc() and __popc() at warp level
  - Using shared memory at thread-block level
- Better locality by optimizing access pattern
  - Transforming striped write to coalesced memory access



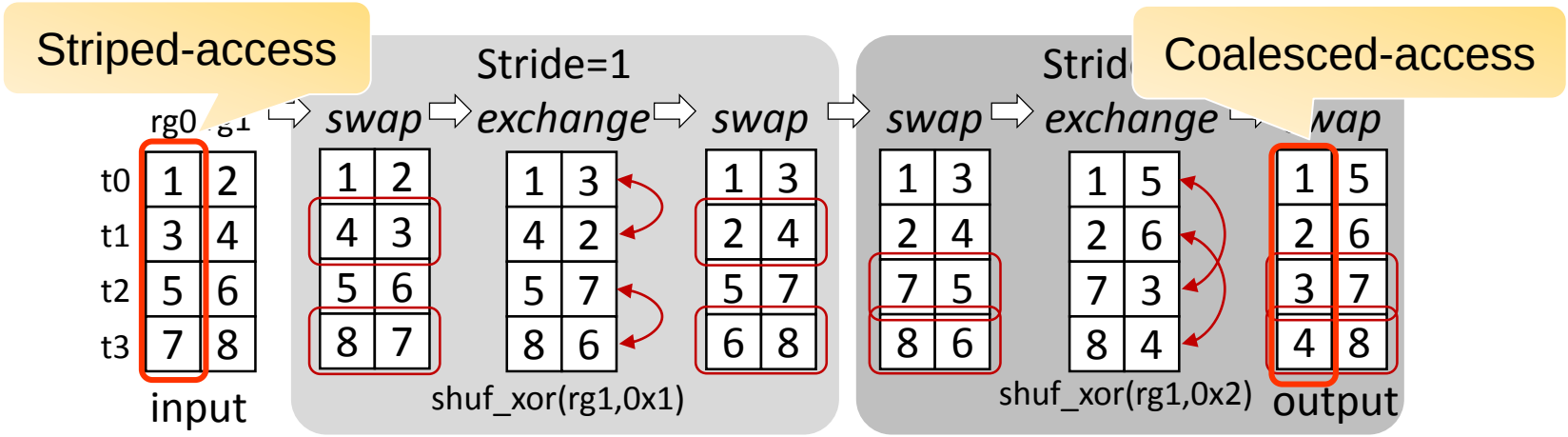
# Other Techniques & Optimizations

- A hierarchical binning
  - Using warp vote function __balloc() and __popc() at warp level
  - Using shared memory at thread-block level
- Better locality by optimizing access pattern
  - Transforming striped write to coalesced memory access



# Other Techniques & Optimizations

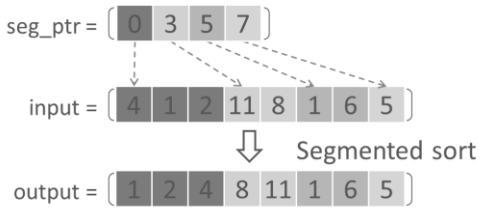
- A hierarchical binning
  - Using warp vote function __balloc() and __popc() at warp level
  - Using shared memory at thread-block level
- Better locality by optimizing access pattern
  - Transforming striped write to coalesced memory access

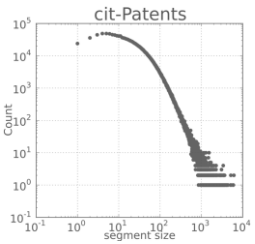



- Shared memory based merge solution
  - *MergePath algorithm [`12]* for load balance

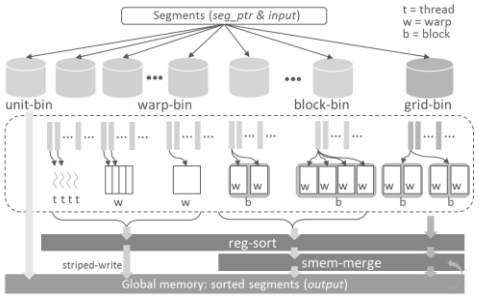
# Outline

- Introduction
- Motivation

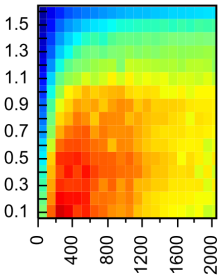


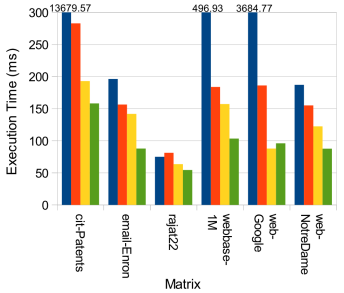




- Our Method
  - GPU SegSort Mechanism
  - GPU Register-based Sort
  - Other Techniques & Opt.




- Evaluation
  - Kernel Performance
  - Kernel in Real Applications



# Experiment Platforms

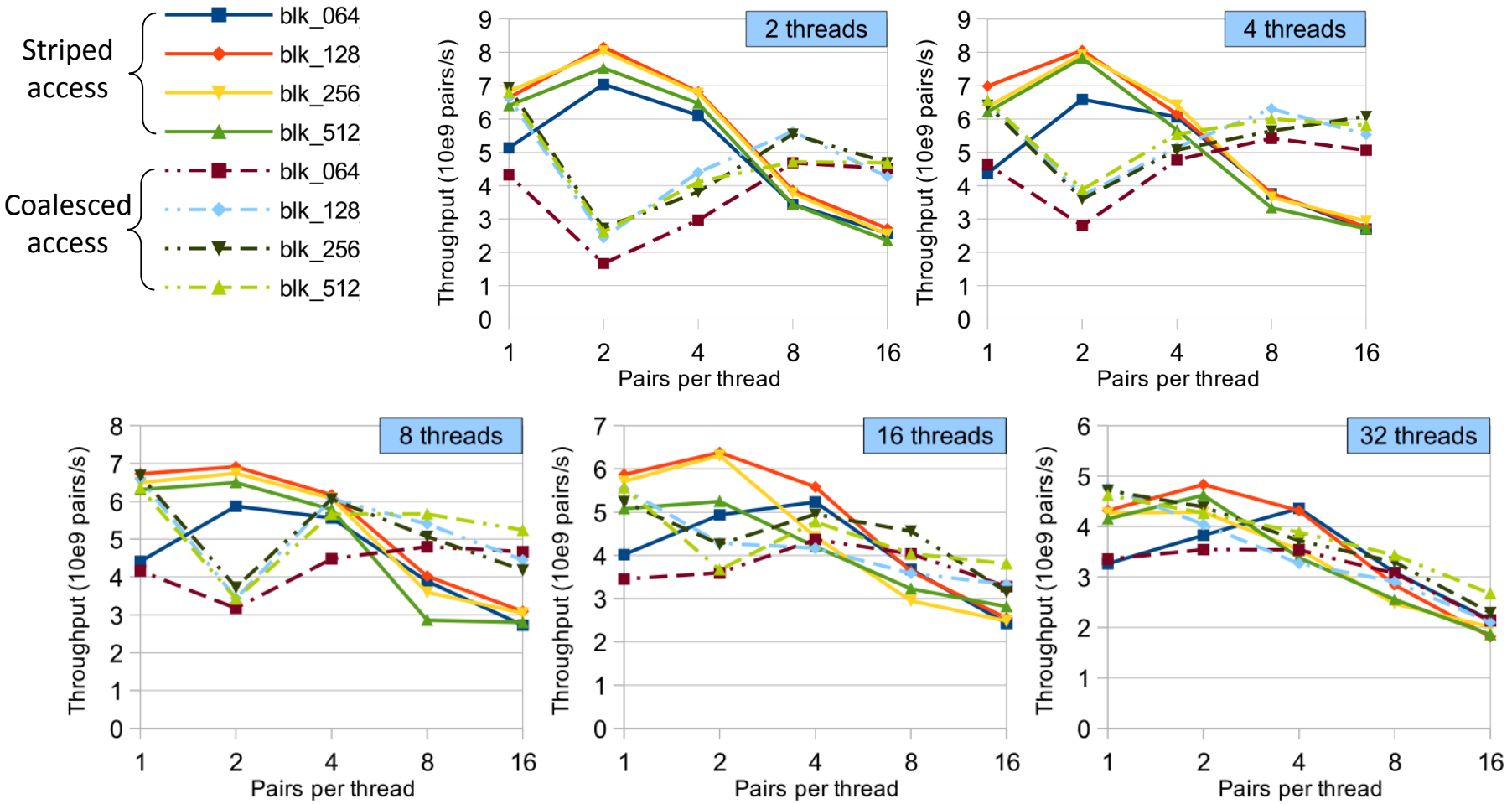
- **nVidia Tesla K80 (Kepler-GK210)**, 2496 CUDA cores @ 824 MHz, 240 GB/s bandwidth
- **nVidia TitanX (Pascal-GP102)**, 3584 CUDA cores @ 1531 MHz, 480 GB/s bandwidth
- We compare our **SegSort** to other tools from libraries of
  a. ModernGPU v.2.0 (boundary checking, global sort based)
  b. CUSP* v.0.5.0 (global sort based)
  c. CUB v.1.6.4 (segment per block)
  – Generating datasets to mimic different segment distributions
- We compare SAC and SpGEMM optimized by our **SegSort** to
  a. cuDPP v.2.3 for SAC
  b. cuSPARSE [`16], CUSP [`14], bhSPARSE [`14] for SpGEMM
  – Using real input datasets from NCBI library and UF matrix collection

* CUSP performs segmented sort by using THRUST sort twice. We extract this as a stand-alone function.

# Kernel Performance Tuning

- Binding different number of data items to threads (**reg_sort**)

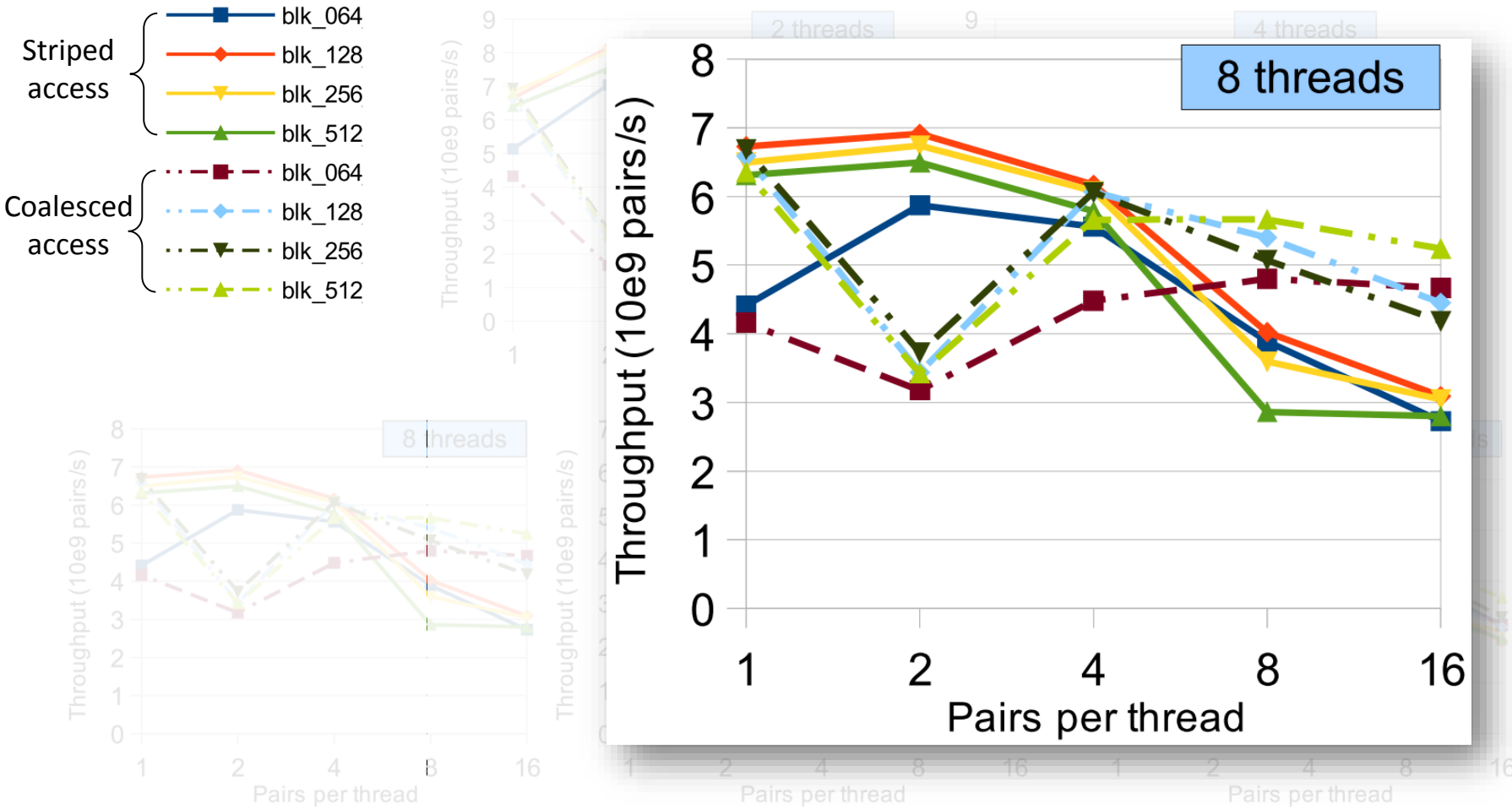
**Kepler GPU**



# Kernel Performance Tuning

- Binding different number of data items to threads (**reg_sort**)

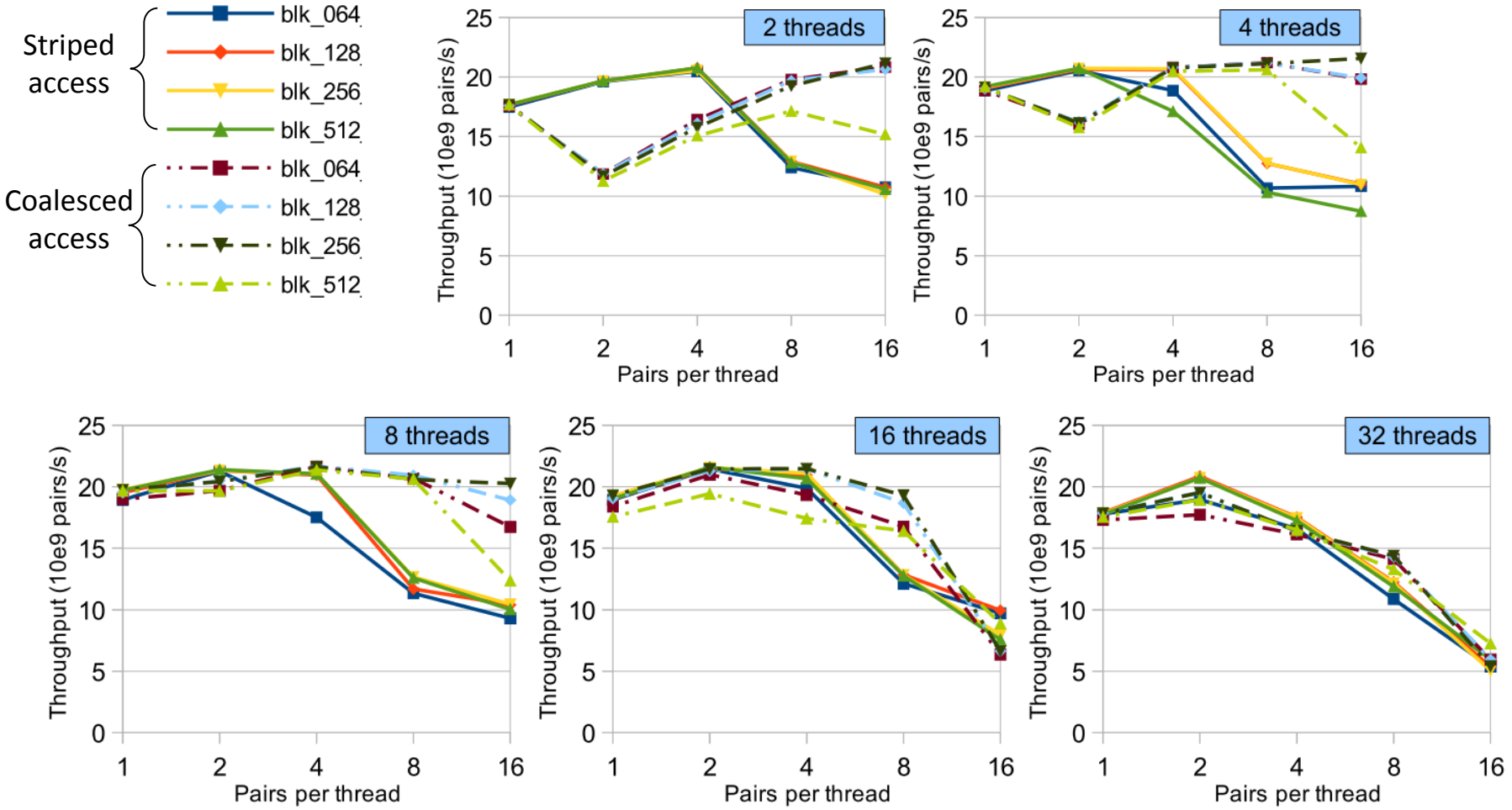
**Kepler GPU**



# Kernel Performance Tuning

- Binding different number of data items to threads (**reg_sort**)

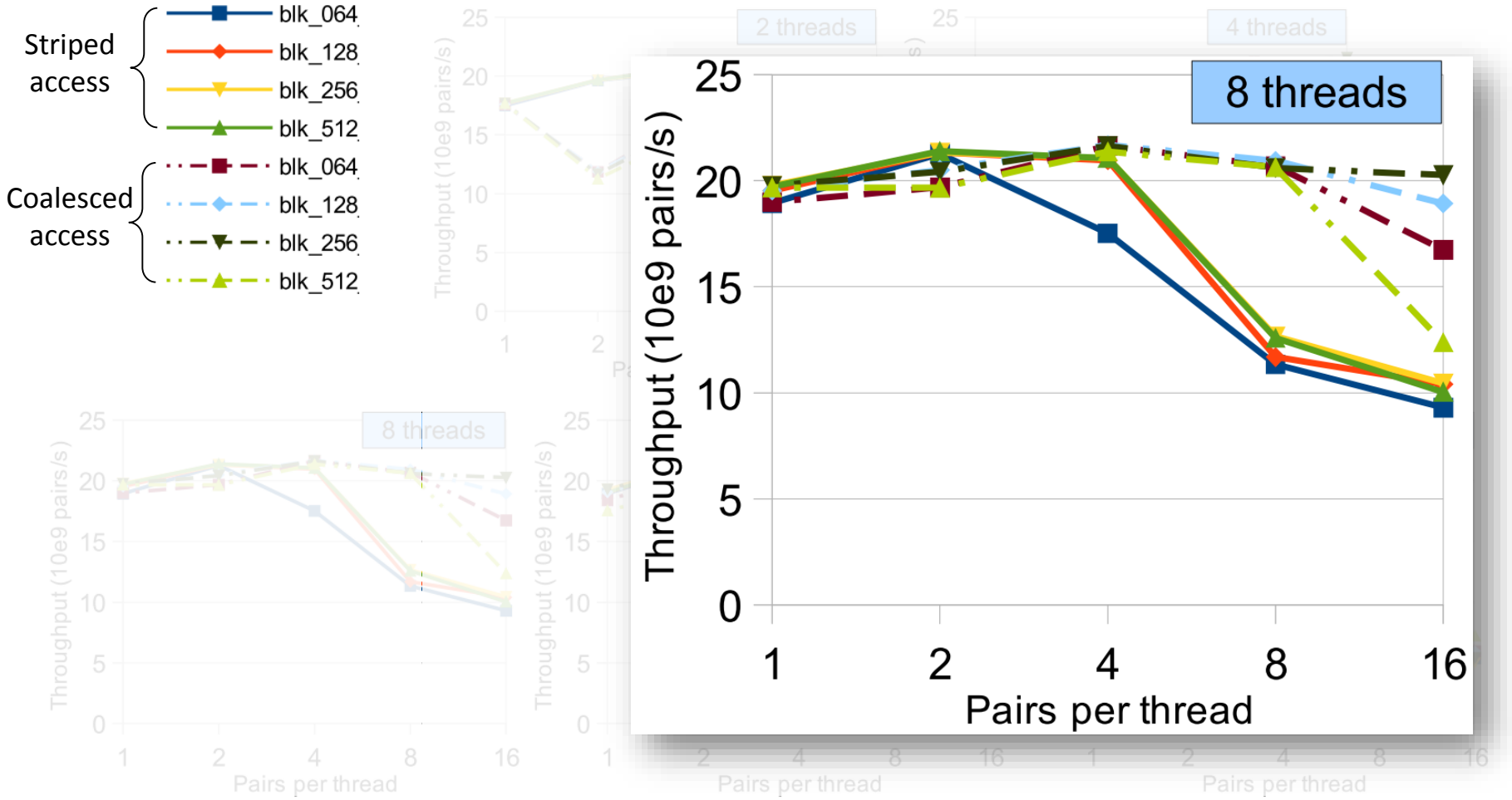
**Pascal GPU**



# Kernel Performance Tuning

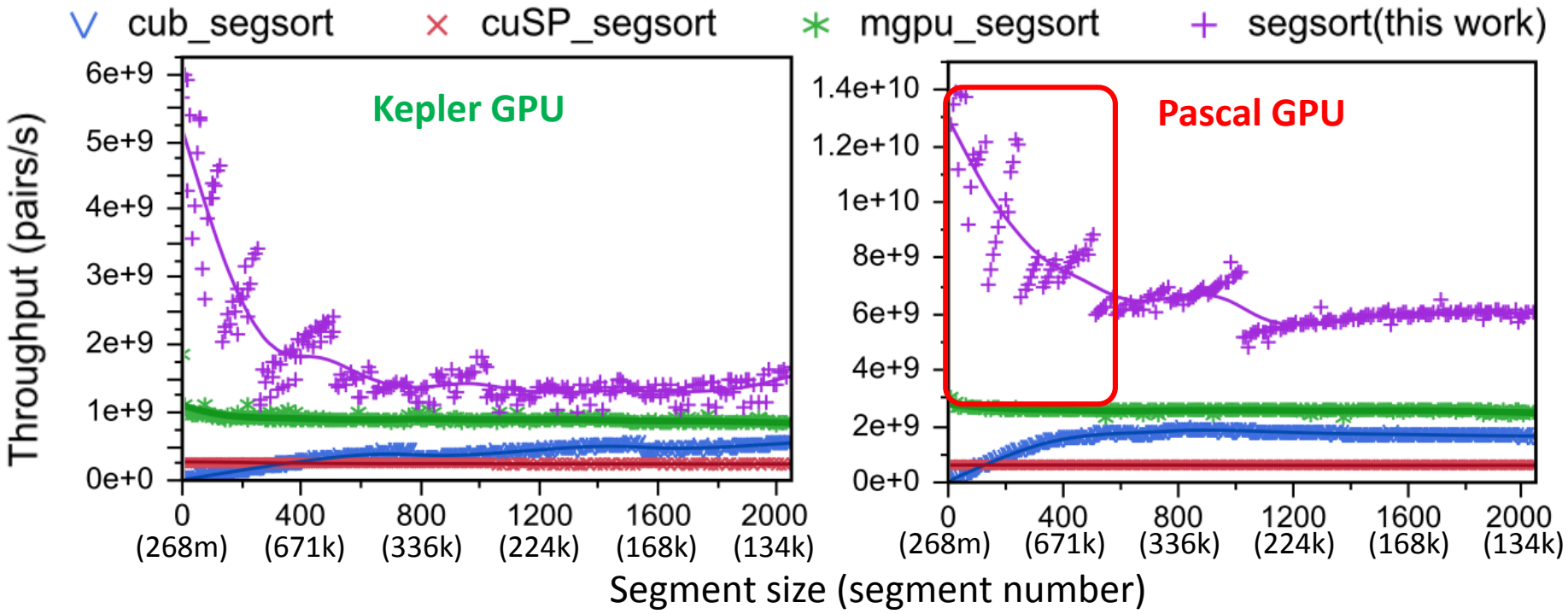
- Binding different number of data items to threads (**reg_sort**)

**Pascal GPU**



# SegSort Performance

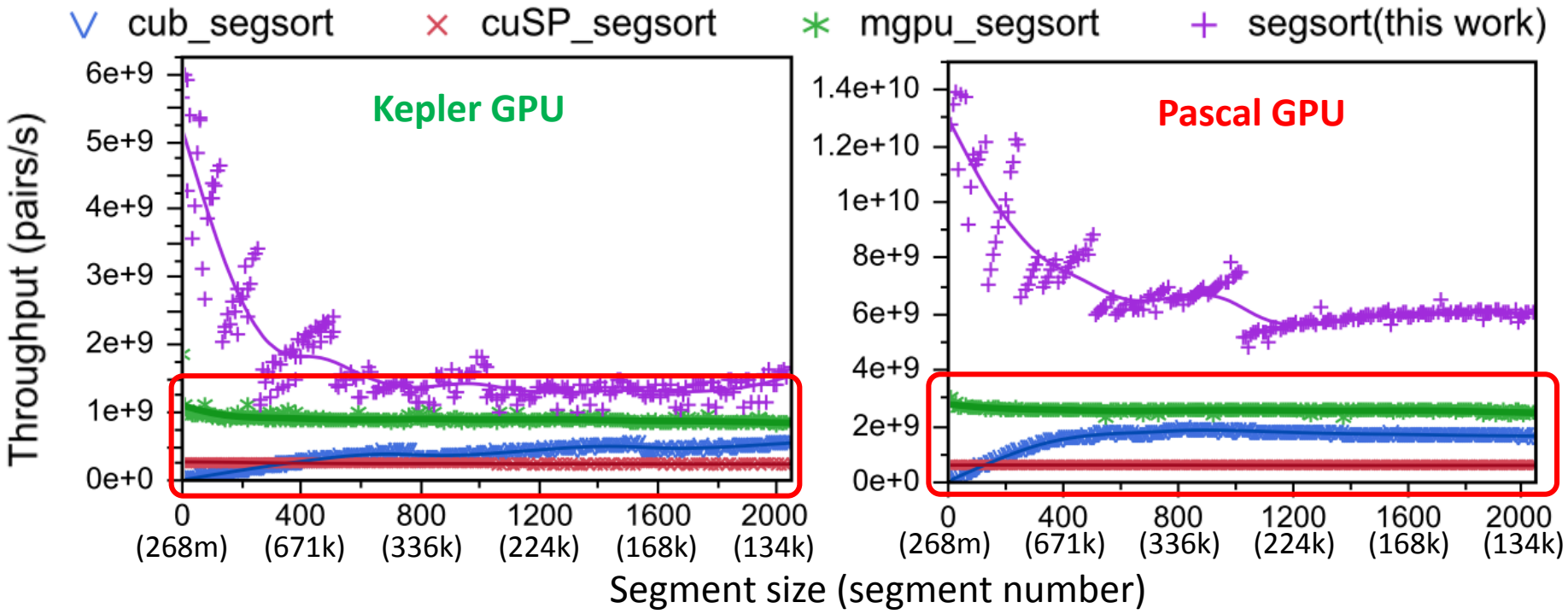
- Fixing total data size w/ variable segment number and size




  - Our SegSort is proficient in solving a large amount of segments, achieving an average of 3.2x speedups over the better performed baseline mgpu-segsort on Pascal

# SegSort Performance

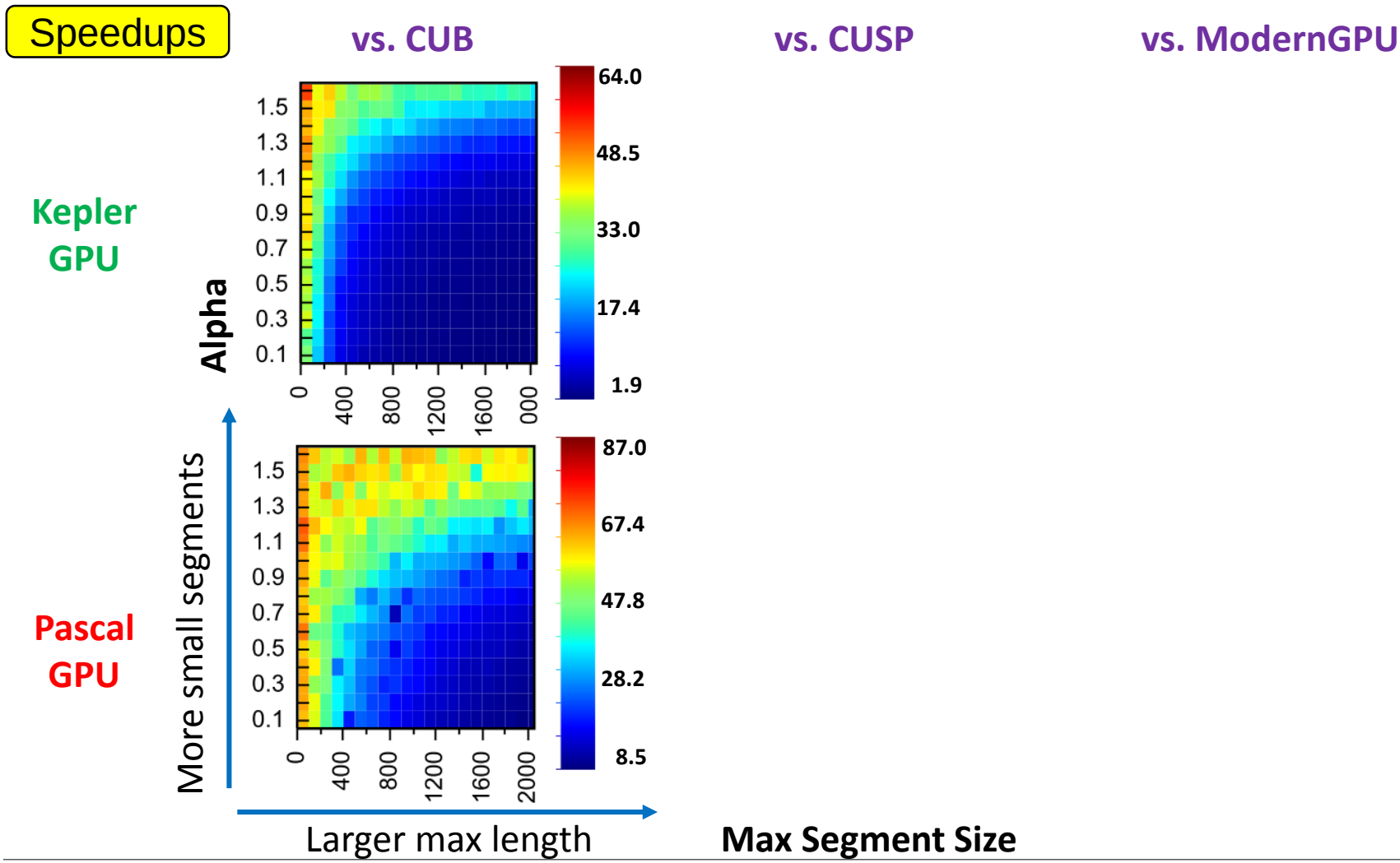
- Fixing total data size w/ variable segment number and size




  - Our SegSort is proficient in solving a large amount of segments, achieving an average of 3.2x speedups over the better performed baseline mgpu-segsort on Pascal
  - The performance of SegSorts, evolved from global sort, is more affected by the total array size

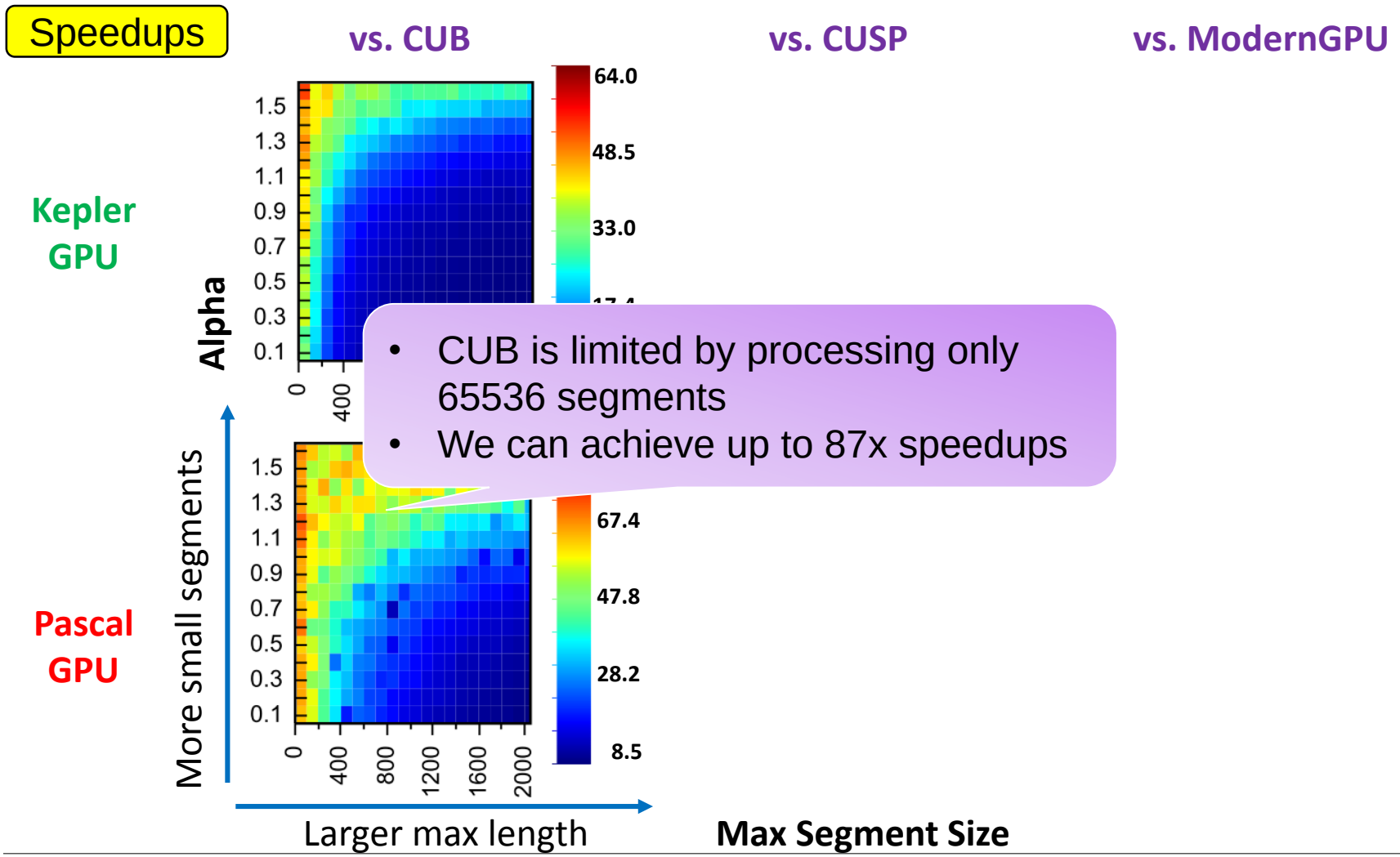
# SegSort Performance

- Fixing total data size w/ segments of power-law distribut.



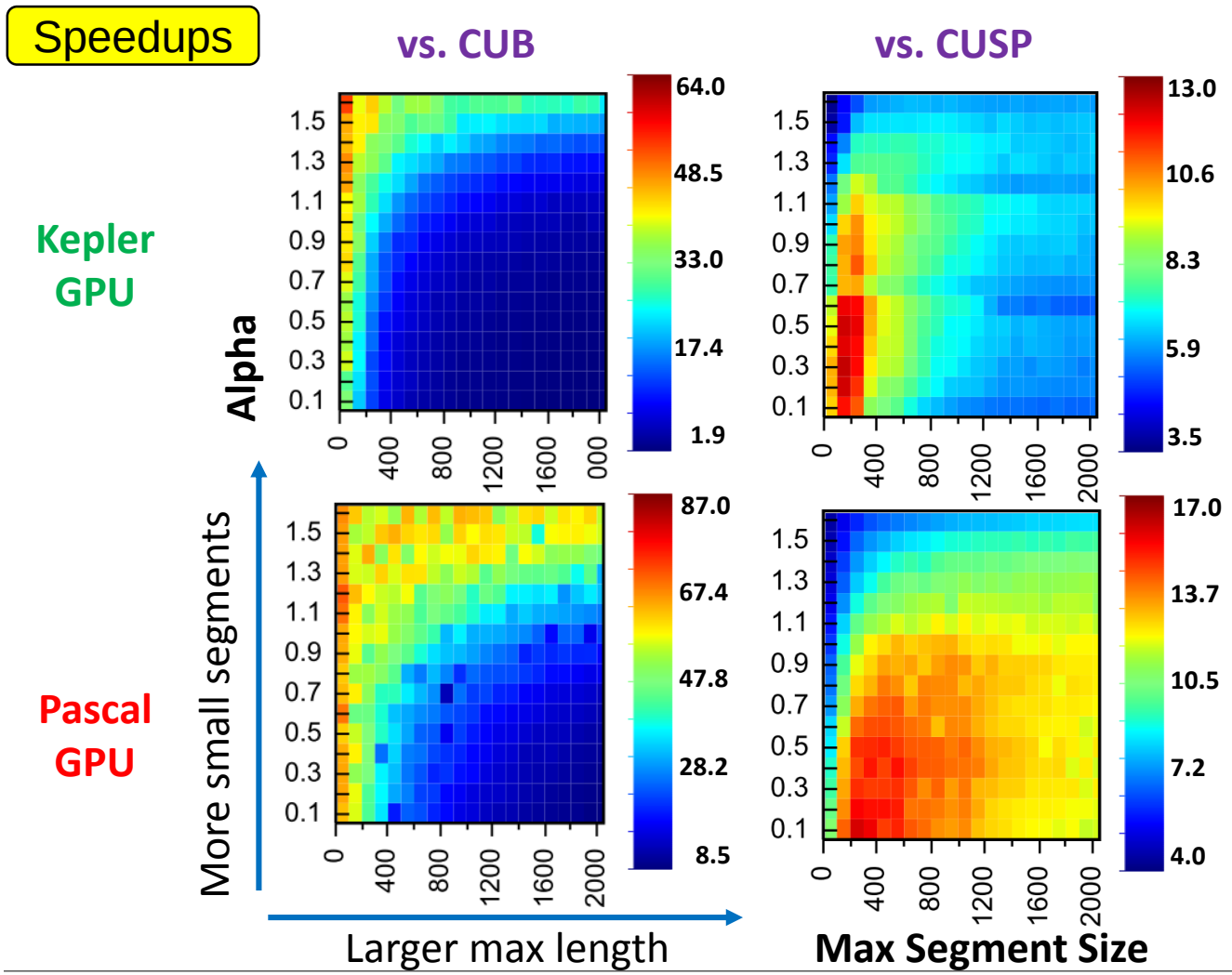
# SegSort Performance

- Fixing total data size w/ segments of power-law distribut.



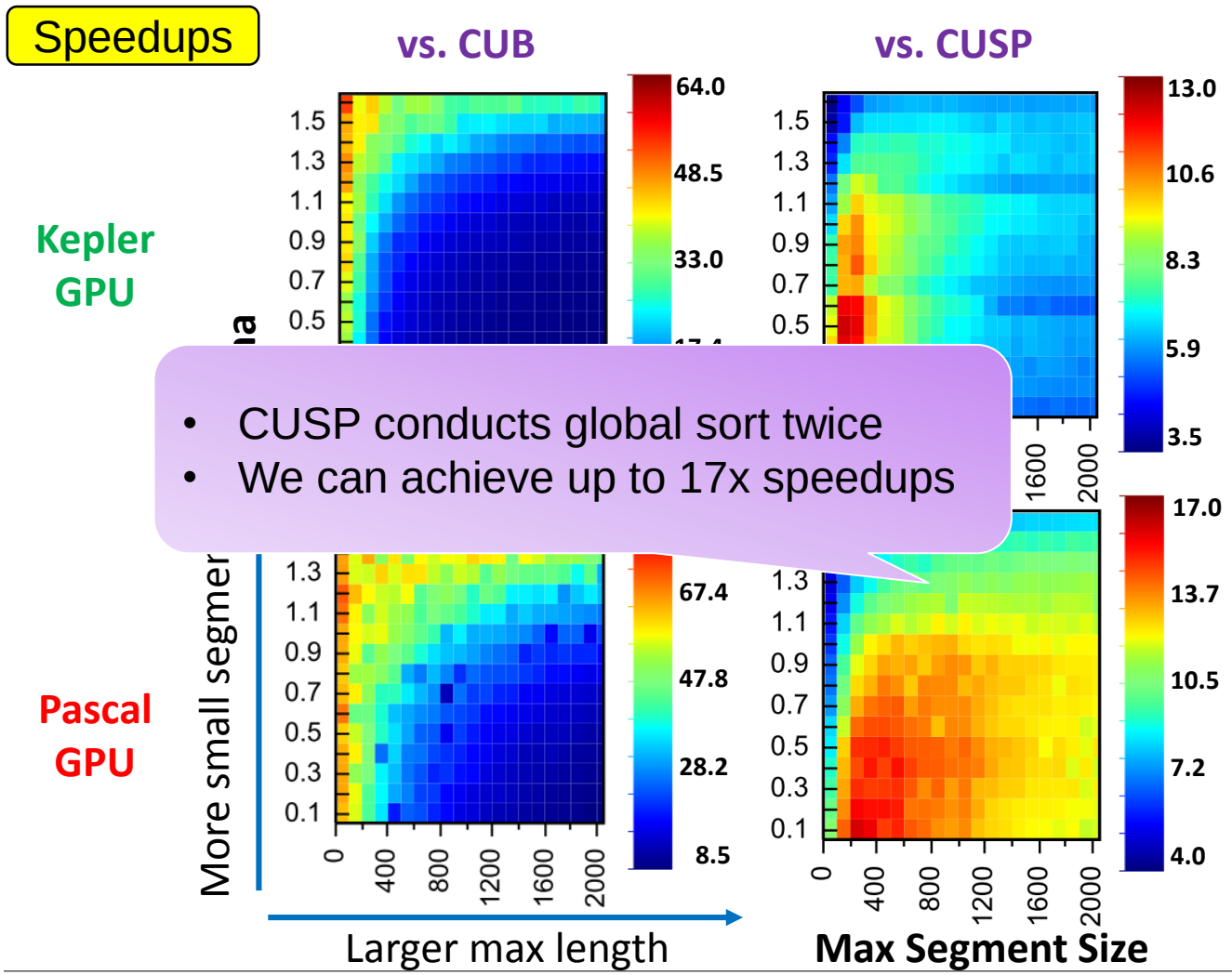
# SegSort Performance

- Fixing total data size w/ segments of power-law distribut.



# SegSort Performance

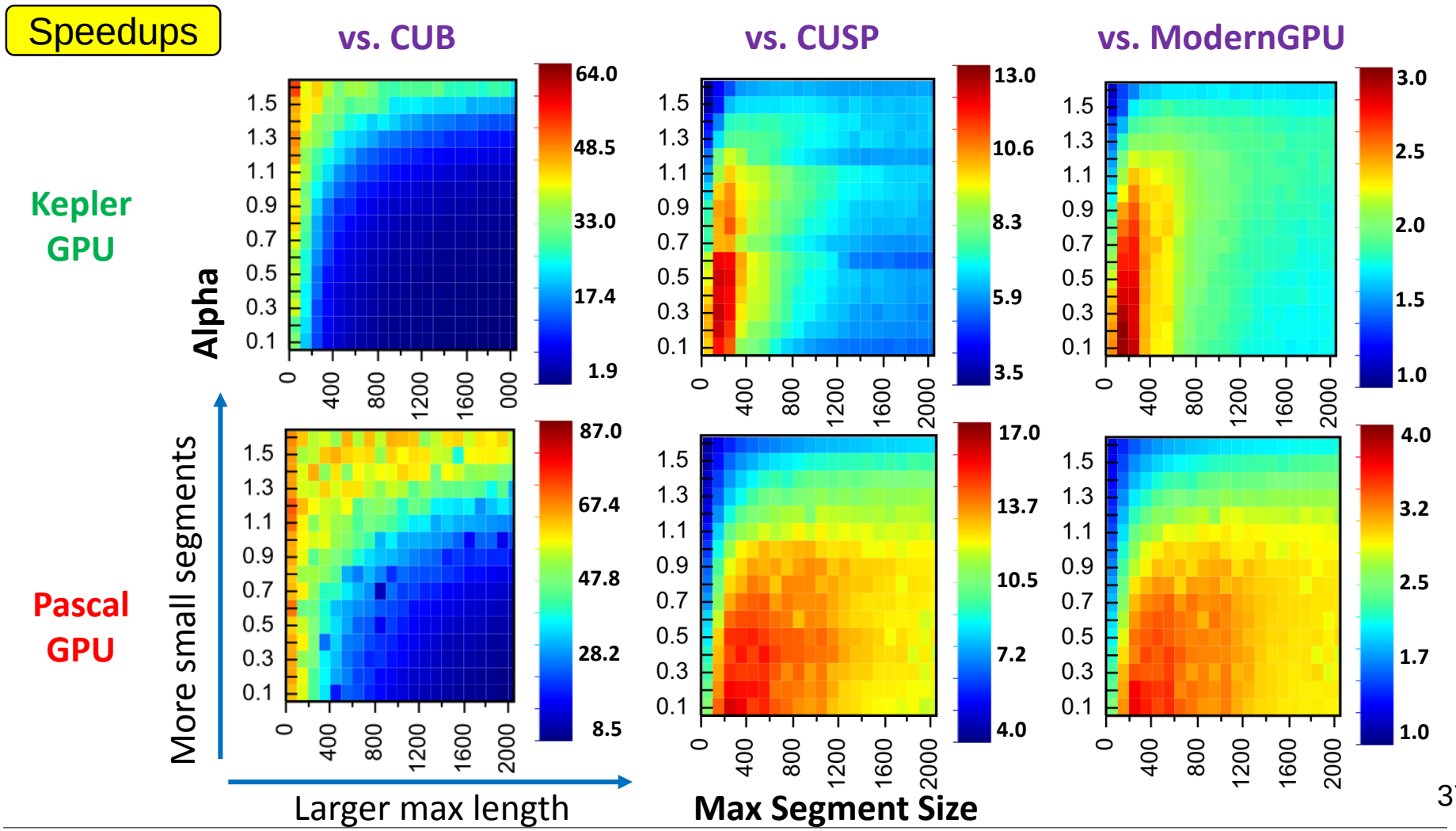
- Fixing total data size w/ segments of power-law distribut.




- CUSP conducts global sort twice
- We can achieve up to 17x speedups

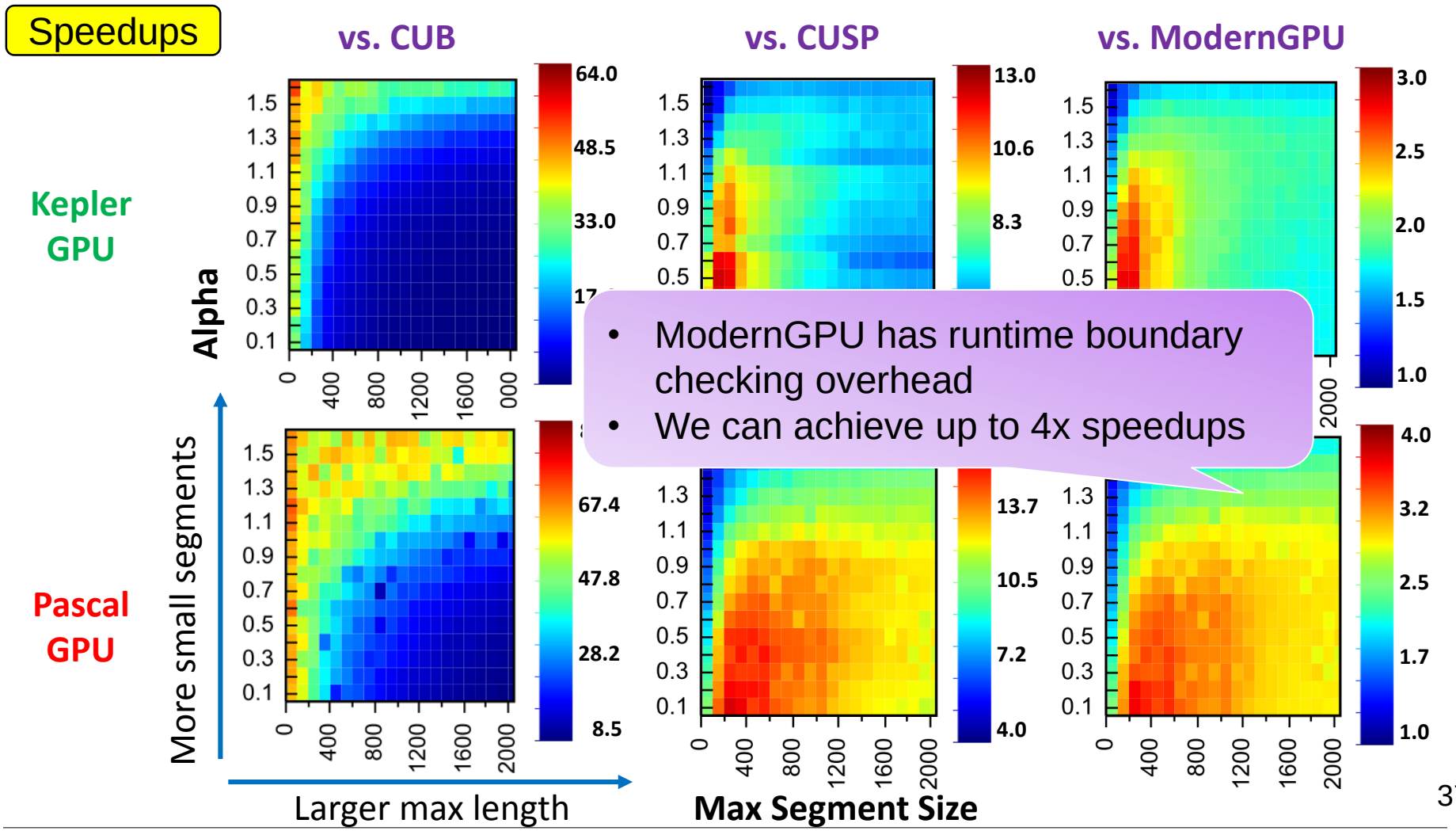
# SegSort Performance

- Fixing total data size w/ segments of power-law distribut.



# SegSort Performance

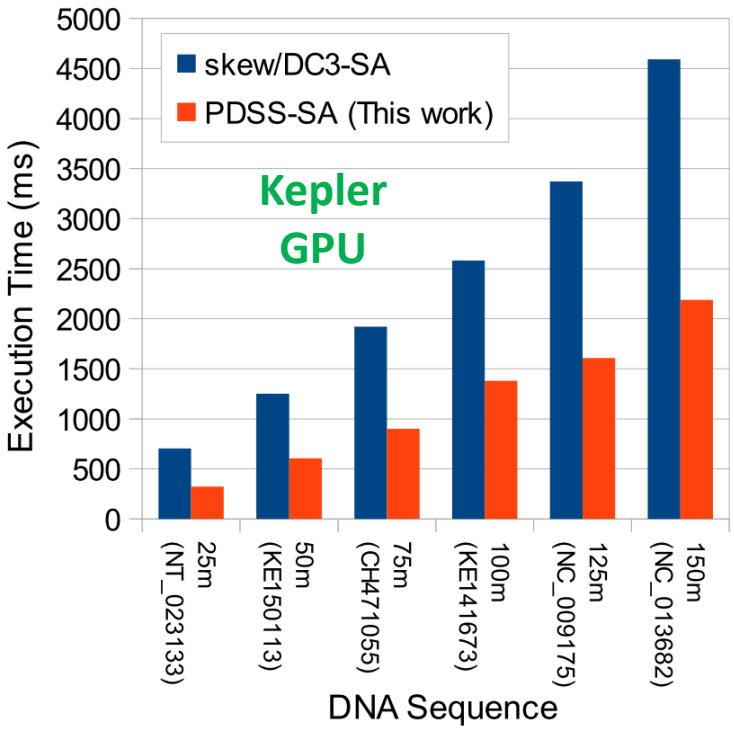
- Fixing total data size w/ segments of power-law distribut.

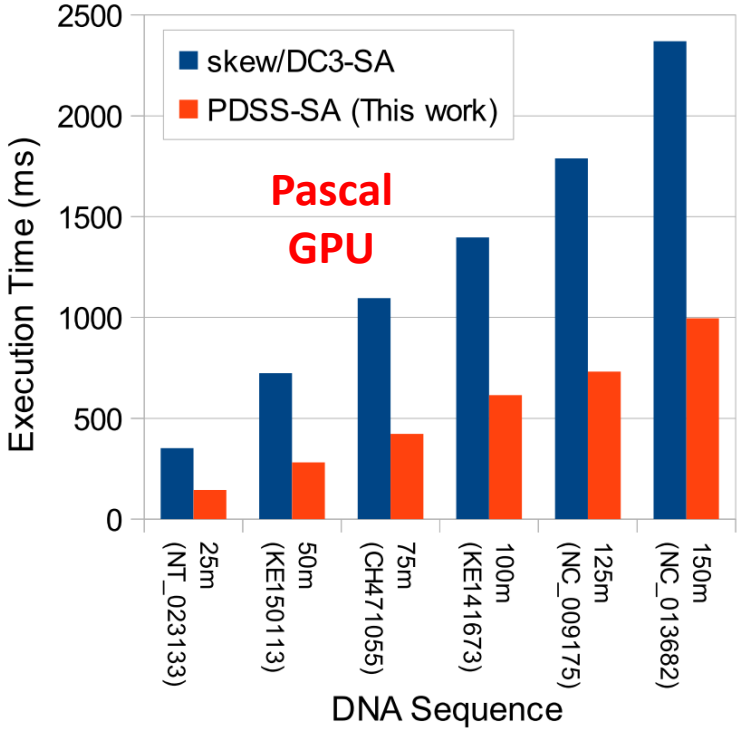



- ModernGPU has runtime boundary checking overhead
- We can achieve up to 4x speedups

# SegSort in Real-world Applications

- **Suffix Array**: store lexicographically sorted indices of all suffixes of a given sequence
- Our method is based on the *prefix doubling* algorithm [`93]
  - Deducing the orders of ***2h*** strings from the calculated orders of ***h*** strings



# SegSort in Real-world Applications

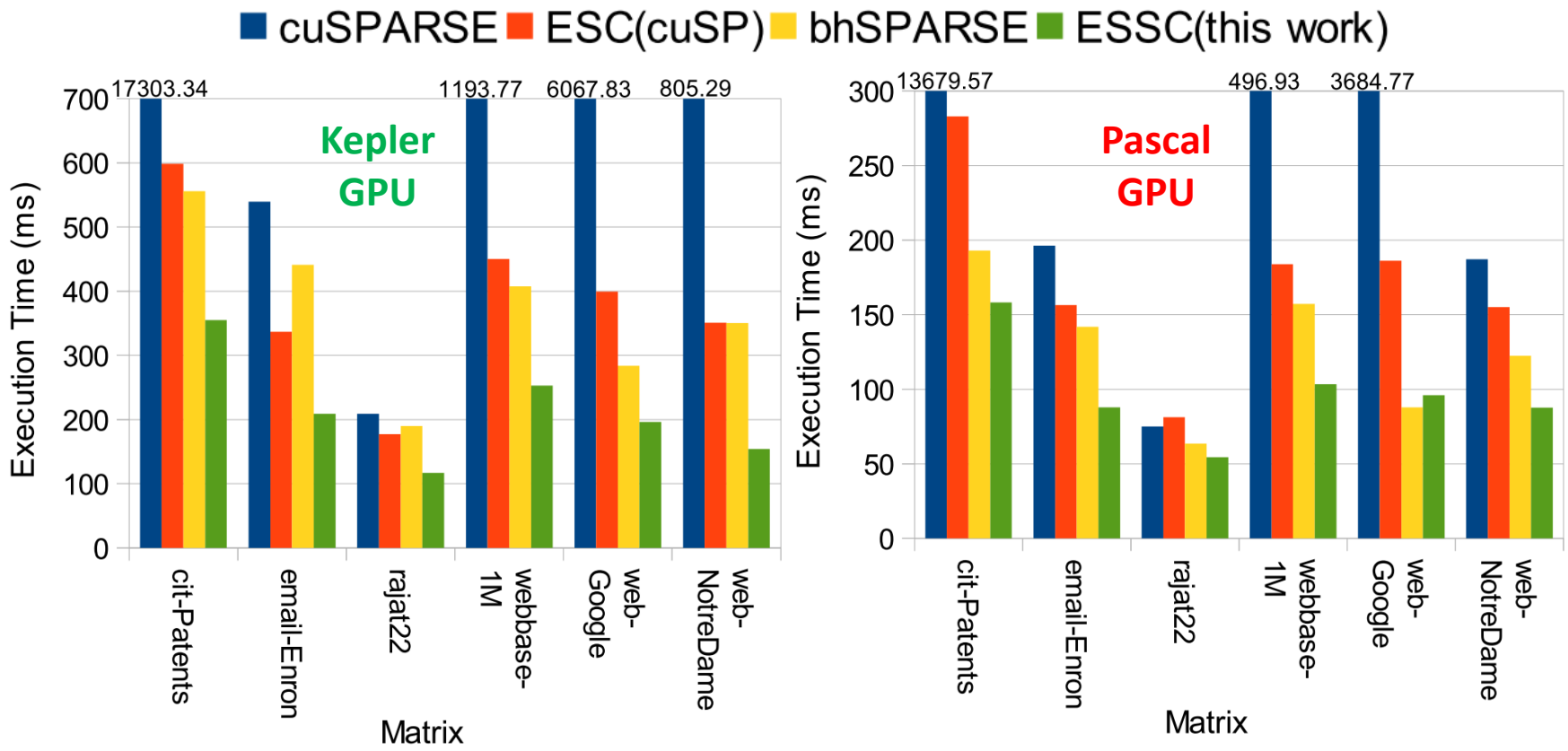
Sparse Matrix-Matrix Multiplication (Sp by its indices of rows and columns

- Our method is using the *Expansion*, ***Seg-Sorting*** and *Compression* (ESSC) algorithm [`12]
  - Sorting an intermediate sparse matrix $C$ by its indices of rows and columns

# SegSort in Real-world Applications

Sparse Matrix-Matrix Multiplication (Sp by its indices of rows and columns

- Our method is using the *Expansion, Seg-Sorting* and



# Conclusion

- We identified the importance of segmented sort on various applications, and proposed efficient approaches on GPUs
- Our GPU segmented sort method outperforms other state-of-the-art approaches in libraries of CUB, CUSP, ModernGPU
- We can see that the capacity of registers is important for segmented sort in modern GPUs
- Please visit our GIT repo **https://github.com/vtsynergy**

**Thank you!**

Email to **kaixihou@vt.edu**
More from **synergy.cs.vt.edu**