VirginiaTech College of Engineering
DEPARTMENT OF COMPUTER SCIENCE
SyNeRG synergy.cs.vt.edu

Exploring Performance Portability for Accelerators via High-level Parallel Patterns

Ph.D. Candidate: **Kaixi Hou**

Committee Members:

Wu-chun Feng (Chair), Calvin J. Ribbens, Hao Wang,
Yong Cao, Gagan Agrawal

Virginia Tech

VIRGINIA POLYTECHNIC INSTITUTE
AND STATE UNIVERSITY

Final Exam Presentation, May, 2018

Parallel Accelerators

- ***Parallel computing*** has become mainstream and very affordable in our daily life

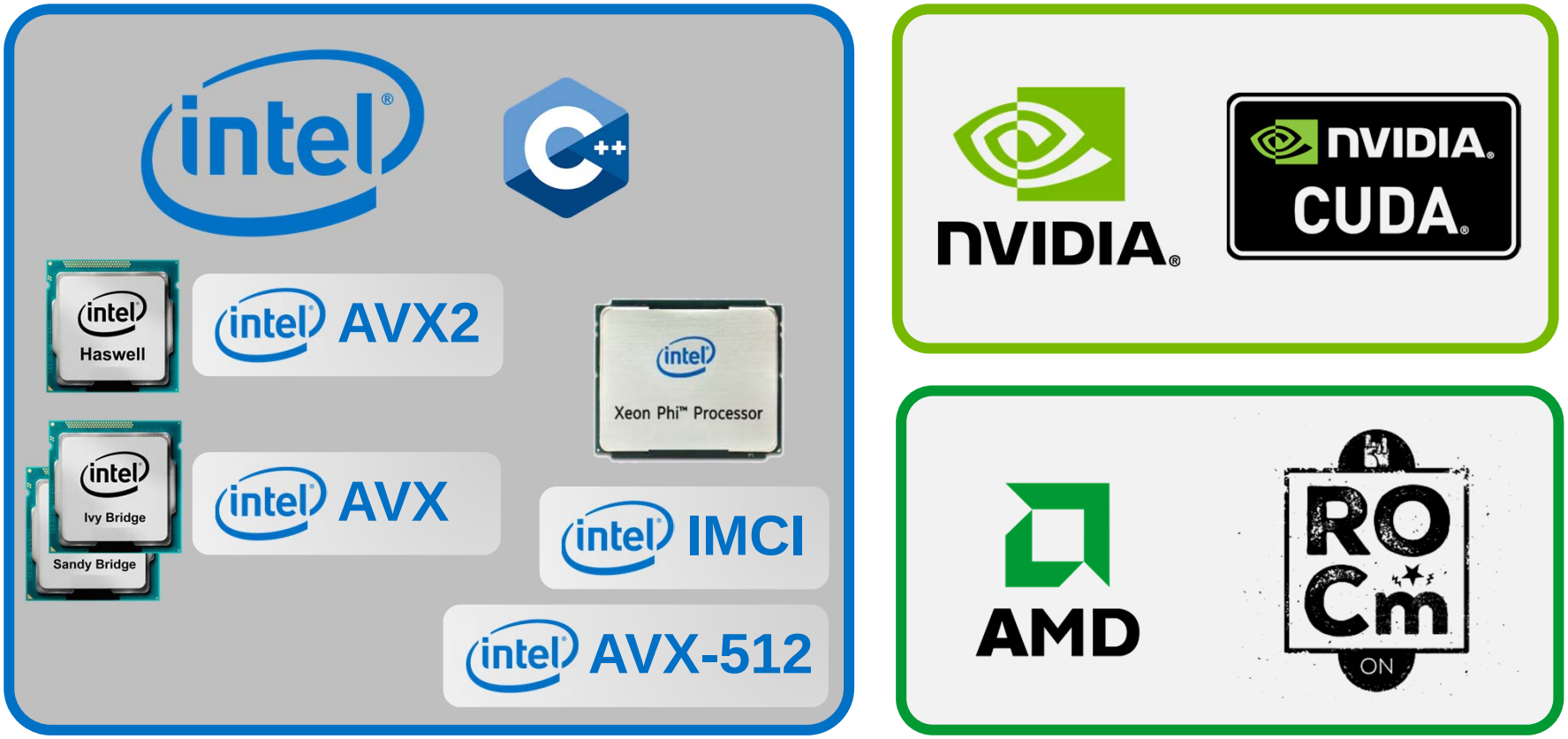
NVIDIA AMD intel

- But, how can we exploit these accelerators?
  - Highly-optimized programs require great efforts for each device

Problem: Programming different accelerators usually causes performance portability issues!

Challenges for Performance Portability

- Diversified architectures, ISAs, languages, etc.
  - Different generations with same vendor
  - Different venders

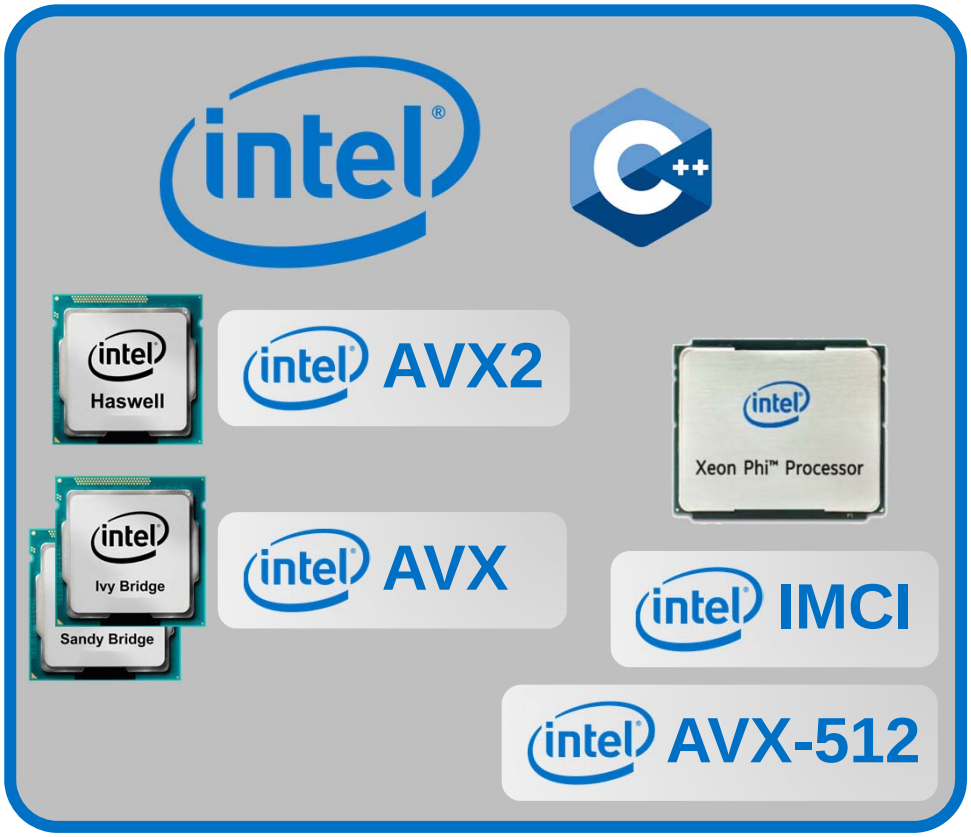


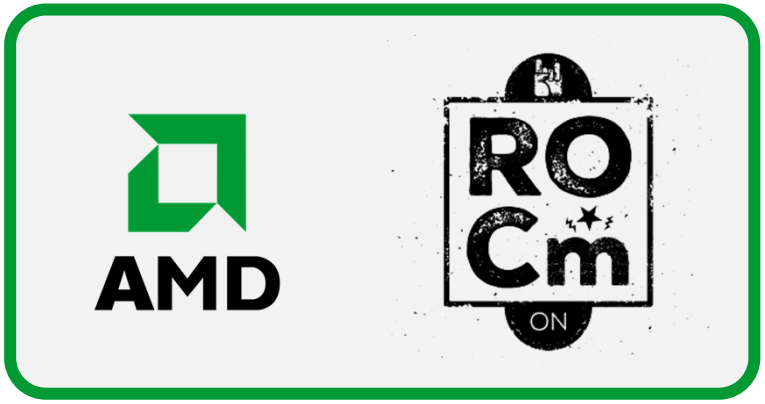
* Although ROCm is designed as an open platform for GPU computing, we focus on its superior support for AMD platforms.

Challenges for Performance Portability

- Diversified architectures, ISAs, languages, etc.
  - Different generations with same vendor
  - Different venders

We will focus on GPU systems for this talk

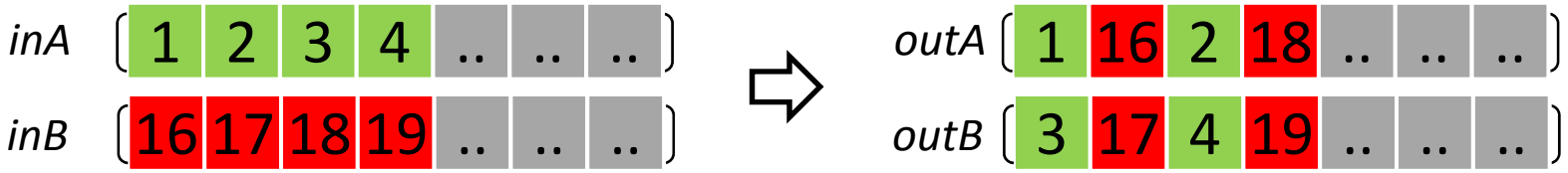




* Although ROCm is designed as an open platform for GPU computing, we focus on its superior support for AMD platforms.

Challenges for Performance Portability

- Why not simply rely on modern compilers' capability of auto-vectorization/parallelization?



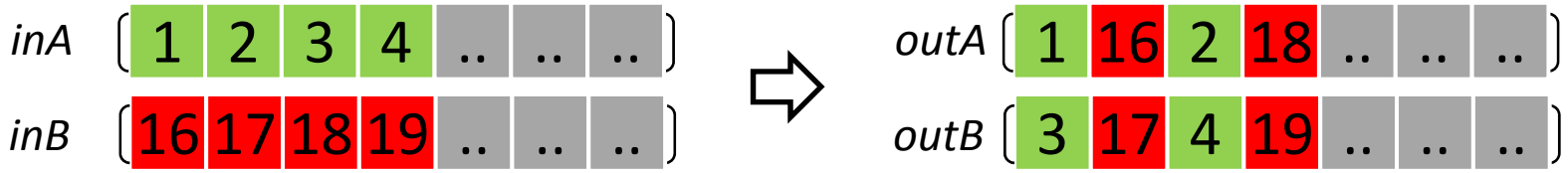
*C++ code with OpenACC for GPUs **

```
#pragma acc kernels loop gang(32), vector(32)
for(i = 0; i < n; i++) {
      if(i%2 == 0) outA[i] = inA[(i/32*32)+(i%32)/2];
      else         outA[i] = inB[(i/32*32)+(i%32)/2];     }
#pragma acc kernels loop gang(32), vector(32)
for(i = 0; i < n; i++) {
      if(i%2 == 0) outB[i] = inA[(i/32*32)+(i%32)/2+1];
      else         outB[i] = inB[(i/32*32)+(i%32)/2+1];  }
```

* The codes are based on 32 elements per group, which equals to the warp size for NVIDIA GPUs.

Challenges for Performance Portability

- Why not simply rely on modern compilers' capability of auto-vectorization/parallelization?



*C++ code with OpenACC for GPUs **

```
53, 2 loops fused
    Loop not vectorized: may not be beneficial
    if(i%2 == 0) outA[i] = inA[(i/32*32)+(i%32)/2];
    else         outA[i] = inB[(i/32*32)+(i%32)/2];     }
#pragma acc kernels loop gang(32), vector(32)
for(i = 0; i < n; i++) {
    if(i%2 == 0) outB[i] = inA[(i/32*32)+(i%32)/2+1];
    else         outB[i] = inB[(i/32*32)+(i%32)/2+1];  }
```

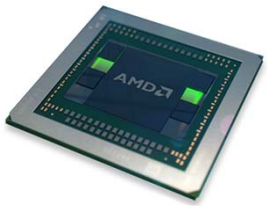
  - They might fail to apply the auto-vectorization/parallelization
    - *Data dependencies, complex memory accesses, convoluted data reordering*, etc.
  - Even if they succeed, the parallel codes might not be efficient!

* The codes are based on 32 elements per group, which equals to the warp size for NVIDIA GPUs.

Challenges for Performance Portability

- Pursuing high performance will inevitably need dedicated kernels and optimizations for every device
  - Program design (for different architectures)

AMD GPU



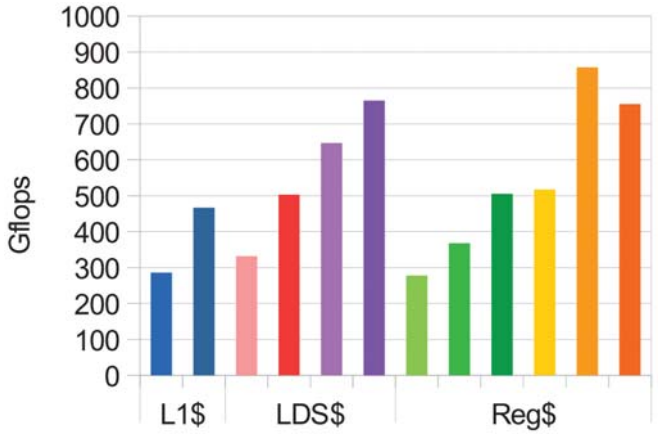
```
friend_id0 = (lane_id+11 +((lane_id>>3)<<1))&63;
tx0 = amdgcn_ds_bpermute(friend_id0<<2, reg0);
ty0 = amdgcn_ds_bpermute(friend_id0<<2, reg1);
```

```
friend_id = (lane_id+11 +((lane_id>>3)<<1))&(32-1);
tx = __shfl(reg0, friend_id);
ty = __shfl(reg1, friend_id);
```

NVIDIA GPU

  - Performance tuning

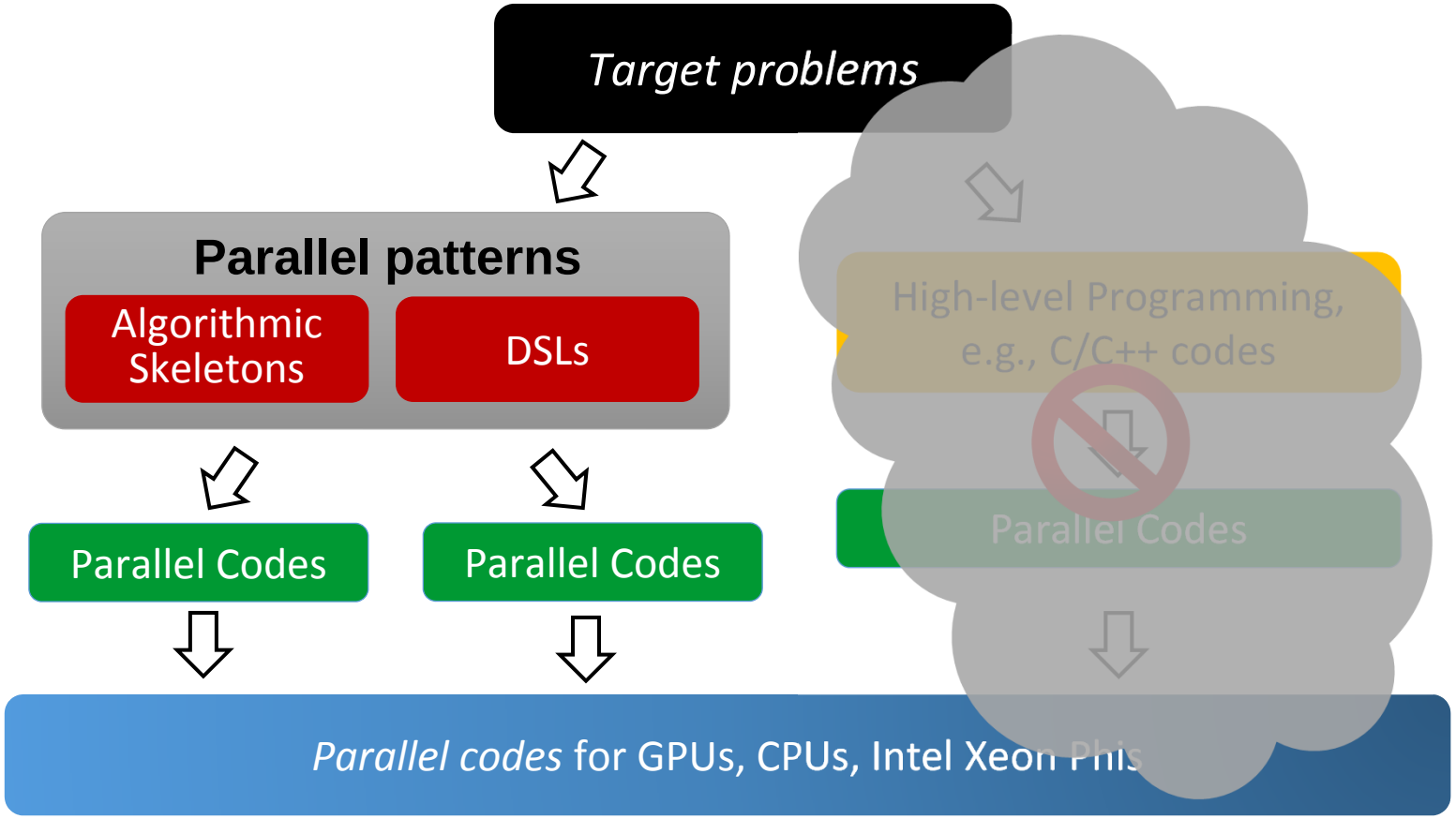
- Each color-group represents one type of GPU optimization
- Each color represents one configuration (or implementation) for that optimization



* Figure show the throughput of 3D27point stencils on AMD GPUs. Codes and figures are from our CF'17 paper.

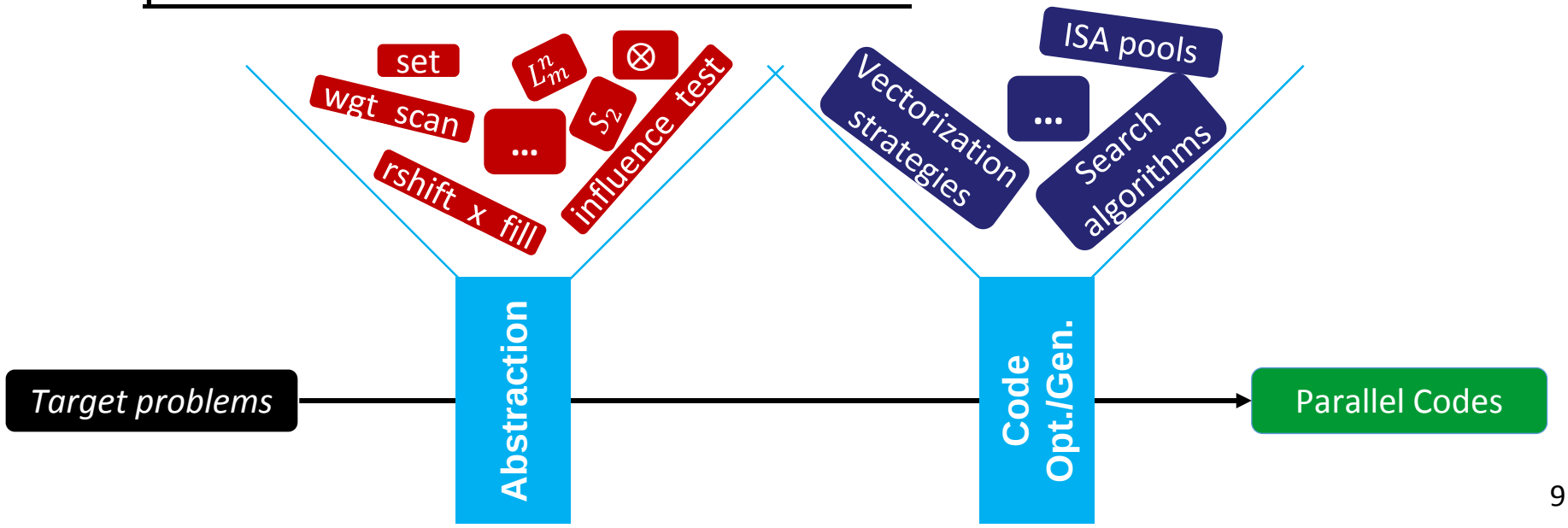
Parallel Pattern Solutions

- Parallel patterns
  - Domain Specific Languages (DSLs)
  - Algorithmic skeletons



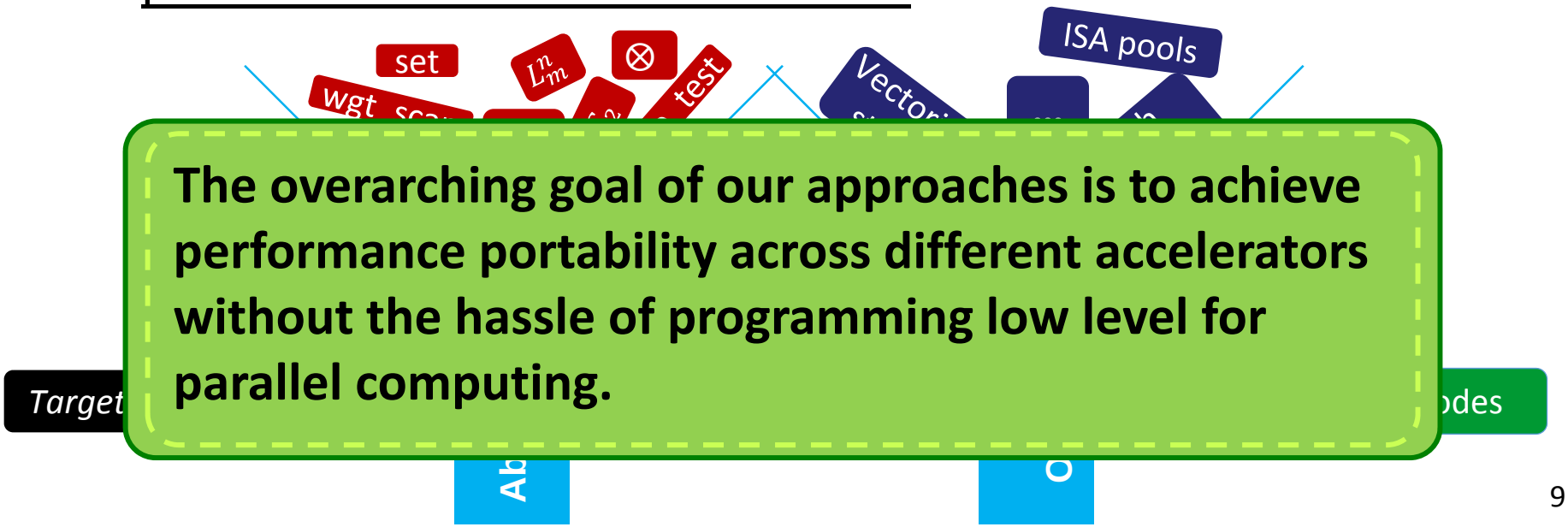
Our Contributions

- For ***abstraction***, we exploit the domain expertise to represent the core computations as parallel patterns, which can greatly facilitate subsequent code optimizations
- For ***code optimization/generation***, we propose a set of solutions to automatically search or create optimal parallel codes for different devices



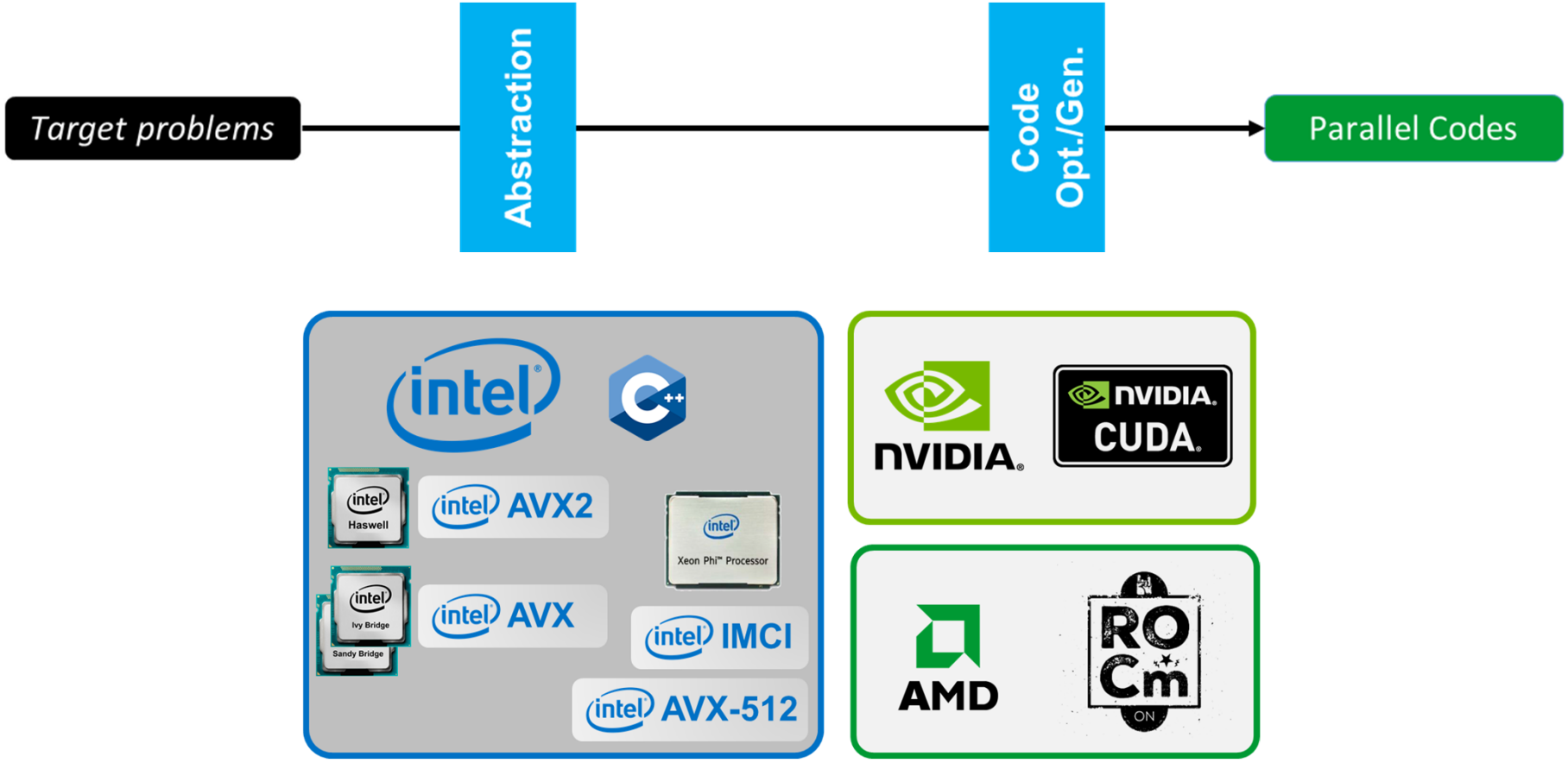
Our Contributions

- For ***abstraction***, we exploit the domain expertise to represent the core computations as parallel patterns, which can greatly facilitate subsequent code optimizations
- For ***code optimization/generation***, we propose a set of solutions to automatically search or create optimal parallel codes for different devices

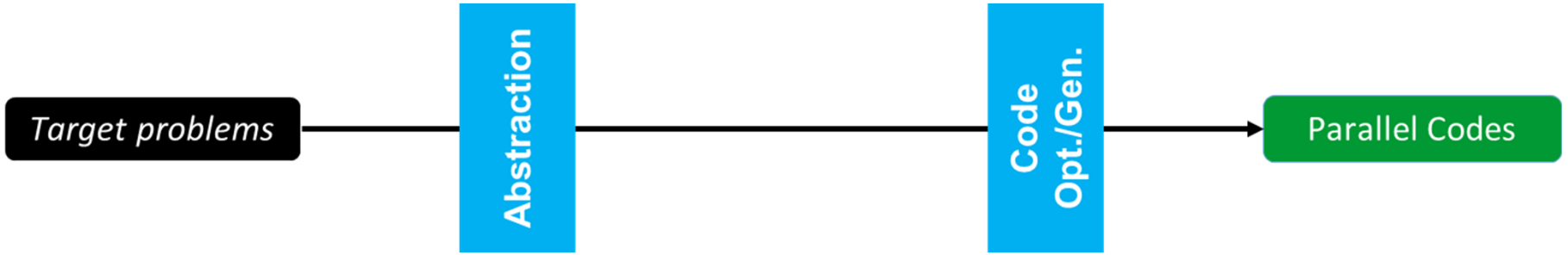


The overarching goal of our approaches is to achieve performance portability across different accelerators without the hassle of programming low level for parallel computing.

Our Contributions (papers)



Our Contributions (papers)



A Framework for the Automatic Vectorization of Parallel Sort on x86-based Processors (**TPDS'18**)

AAlign: A SIMD Framework for Pairwise Sequence Alignment on x86-based Multi- and Many-core Processors (**IPDPS'16**)

ASPaS: A Framework for Automatic SIMDization of Parallel Sorting on x86-based Many-core Processors (**ICS'15**)

Delivering Parallel Programmability to the Masses via the Intel MIC Ecosystem: A Case Study (**P2S2'14**)

Highly Efficient Compensation-based Parallelism for Wavefront Loops on GPUs (**IPDPS'18**)

Fast Segmented Sort on GPUs (**ICS'17**)

GPU-UniCache: Automatic Code Generation of Spatial Blocking for Stencils on GPUs (**CF'17**)

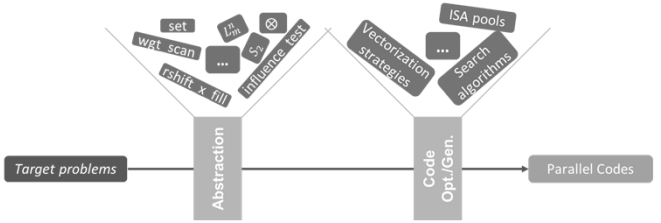
Auto-Tuning Strategies for Parallelizing Sparse Matrix-Vector (SpMV) Multiplication on Multi- and Many-Core Processors(**AsHES'17**)

My Publications

- **Papers I lead:**
  - **K. Hou**, H. Wang, W. Feng, "*A Framework for the Automatic Vectorization of Parallel Sort on x86-based Processors*", IEEE Trans. on Parallel and Distributed Systems (**TPDS**), 2018
  - **K. Hou**, H. Wang, W. Feng, J. Vetter, S. Lee, "*Highly Efficient Compensation-based Parallelism for Wavefront Loops on GPUs*", IEEE Int'l Parallel and Distributed Processing Symposium (**IPDPS**), 2018
  - **K. Hou**, W. Liu, H. Wang, W. Feng, "*Fast Segmented Sort on GPUs*", ACM Int'l Conf. on Supercomputing (**ICS**), 2017
  - **K. Hou**, H. Wang, W. Feng, "*GPU-UniCache: Automatic Code Generation of Spatial Blocking for Stencils on GPUs*", ACM Int'l Conf. on Computing Frontiers (**CF**), 2017
  - **K. Hou**, W. Feng, S. Che, "*Auto-Tuning Strategies for Parallelizing Sparse Matrix-Vector (SpMV) Multiplication on Multi- and Many-Core Processors*", Int'l Workshop on Accelerators and Hybrid Exascale Systems (**AsHES**@IPDPS), 2017
  - **K. Hou**, H. Wang, W. Feng, "*AAlign: A SIMD Framework for Pairwise Sequence Alignment on x86-based Multi- and Many-core Processors*", IEEE Int'l Parallel and Distributed Processing Symposium (**IPDPS**), 2016
  - **K. Hou**, H. Wang, W. Feng, "*ASPaS: A Framework for Automatic SIMDization of Parallel Sorting on x86-based Many-core Processors*", ACM Int'l Conf. on Supercomputing (**ICS**), 2015
  - **K. Hou**, H. Wang, W. Feng, "*Delivering Parallel Programmability to the Masses via the Intel MIC Ecosystem: A Case Study*", Int'l Workshop on Parallel Programming Models and Systems Software for High-End Computing (**P2S2**@ICPP), 2014
- **Papers I contribute:**
  - X. Yu, **K. Hou**, H. Wang, W. Feng, "Robotomata: A Framework for Approximate Pattern Matching of Big Data on an Automata Processor", IEEE Int'l Conf. on Big Data (**Big Data**), 2017
  - H. Wang, W. Liu, **K. Hou**, W. Feng, "Parallel Transposition of Sparse Data Structures", ACM Int'l Conf. on Supercomputing (**ICS**), 2016
  - D. Zhang, H. Wang, **K. Hou**, J. Zhang, W. Feng, "pDindel: Accelerating indel detection on a multicore CPU architecture with SIMD", IEEE Int'l Conf. on Computational Advances in Bio and Medical Sciences (**ICCABS**), 2015

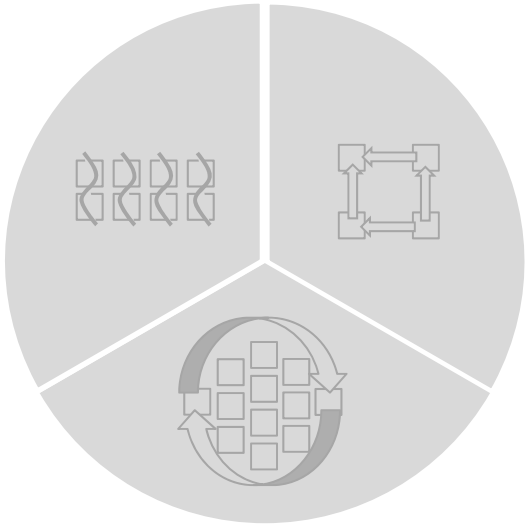
Outline of the Talk

- Motivation
- Contribution & Papers



- **Previous Work**

- Our Methods
  - *Data-reordering (covered in Prelim)*
  - *SIMD Operations (covered in Prelim)*
  - Data-thread Binding (seg_sort)
  - Data Dependencies (wavefront)
  - Data Reuse (stencils)



- Summary and Future Work

Previous Work

- Compiler-related approaches
  - Relying on compiler options/directives to get portable performance [WangICS'16]
  - Refactoring algorithms to be *auto*-parallelizable [SatishISCA'12]
  - Creating new directives or even compilers [UnatICS'11, BondhugulaPACT'14]

- Auto-parallelization for loops is still quite restricted (esp. for GPUs)
- Building a new compiler is expensive and usually specific to some patterns/applications

Previous Work

- DSL-related approaches
  - Use DSL to express the fixed or predictable comp. & comm. patterns [PetersonPADL'98,TangSPAA'11, ChafiPPoPP'11, AumagePPoPP'16, MaruyamaSC'11, KelleyPLDI'13]
- Skeleton-related approaches
  - Reusable and approachable building blocks to build other high-level applications [AndersonIPDPS'16, MalewiczSIGMOD'10, MaruyamaSC'11, HeleneHPCS'13]

We propose, design, and implement a series of automation frameworks and optimizations to determine how different levels of parallelism can be applied using DSL/Skeleton-based approaches to handle ***data-thread binding***, ***data dependencies***, ***data reuse*** problems on GPUs.

Previous Work (References)

Compiler-related approaches

[SatishISCA'12] N. Satish, et al. Can Traditional Programming Bridge the Ninja Performance Gap for Parallel Computing Applications?

[WangICS'16] H. Wang, et al. Parallel Transposition of Sparse Data Structures

[UnatICS'11] D. Unat, et al. Mint: realizing CUDA performance in 3D stencil methods with annotated C

[BondhugulaPACT'14] Bondhugula, et al. Tiling and Optimizing Time-iterated Computations on Periodic Domains

DSL-related approaches

[PetersonPADL'98] J. Peterson, et al. Lambda in Motion: Controlling Robots with Haskell

[TangSPAA'11] Y. Tang, et al. The Pochoir Stencil Compiler

[ChafiPPoPP'11] H. Chafi, et al. A Domain-specific Approach to Heterogeneous Parallelism

[AumagePPoPP'16] O. Aumage, et al. A Stencil DSEL for Single Code Accelerated Computing with SYCL

[MaruyamaSC'11] N. Maruyama, et al. Physis: An Implicitly Parallel Programming Model for Stencil Computations on Large-scale GPU-accelerated Supercomputers.

[KelleyPLDI'13] J. Ragan-Kelley, et al. Halide: A Language and Compiler for Optimizing Parallelism, Locality, and Recomputation in Image Processing Pipelines

Skeleton-related approaches

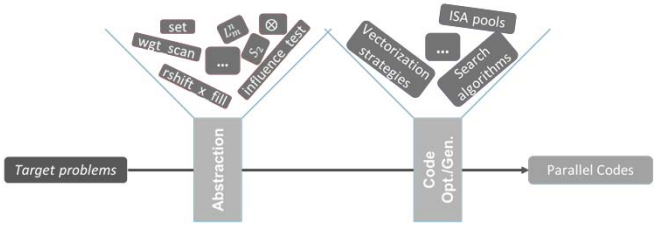
[AndersonIPDPS'16] M. Anderson, et al. GraphPad: Optimized Graph Primitives for Parallel and Distributed Platforms

[MalewiczSIGMOD'10] G. Malewicz, et al. Pregel: A System for Large-scale Graph Processing

[HeleneHPCS'13] C. Helene, et al. Algorithmic Skeleton Library for Scientific Simulations: SkelGIS

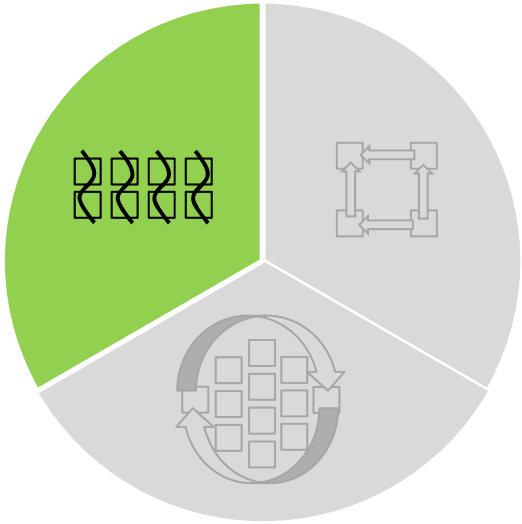
Outline of the Talk

- Motivation
- Contribution & Papers



- Previous Work

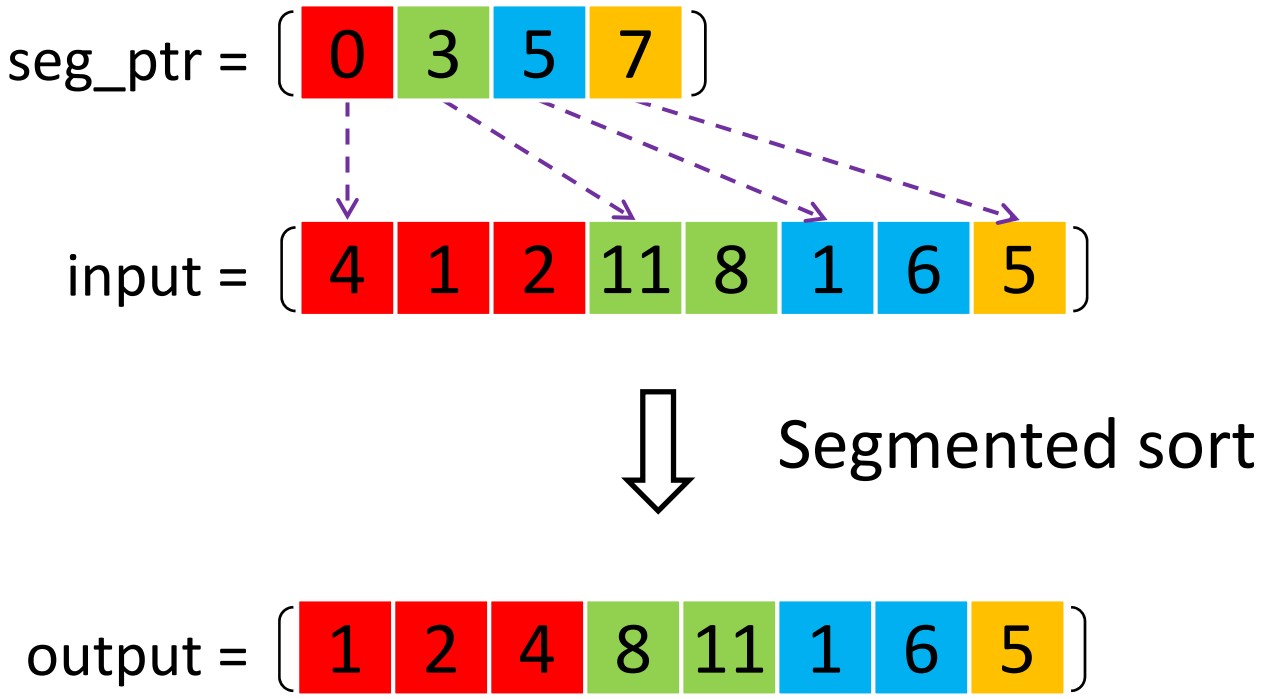
- **Our Methods**
  - *Data-reordering (covered in Prelim)*
  - *SIMD Operations (covered in Prelim)*
  - **Data-thread Binding (seg_sort)**
  - Data Dependencies (wavefront)
  - Data Reuse (stencils)



- Summary and Future Work

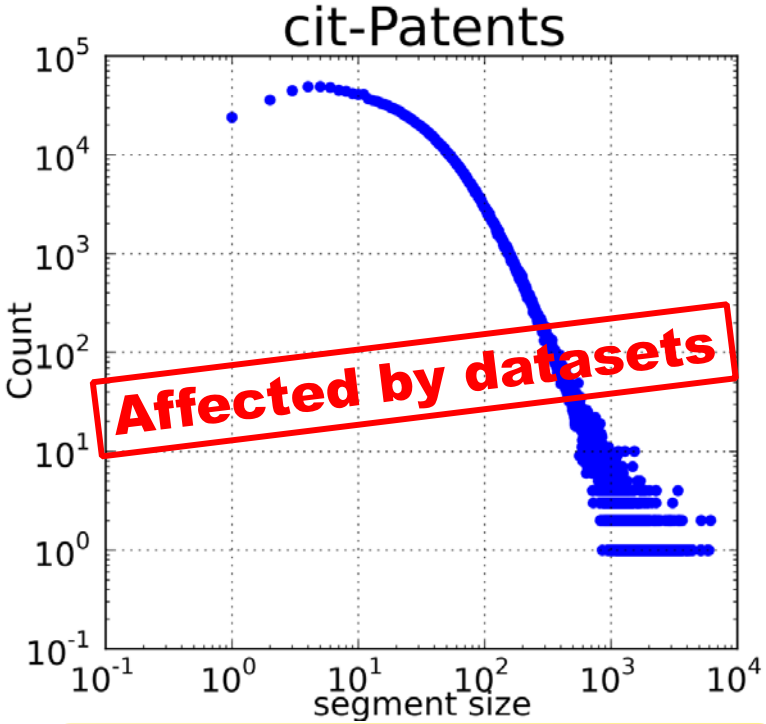
Segmented Sort (*SegSort*)

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

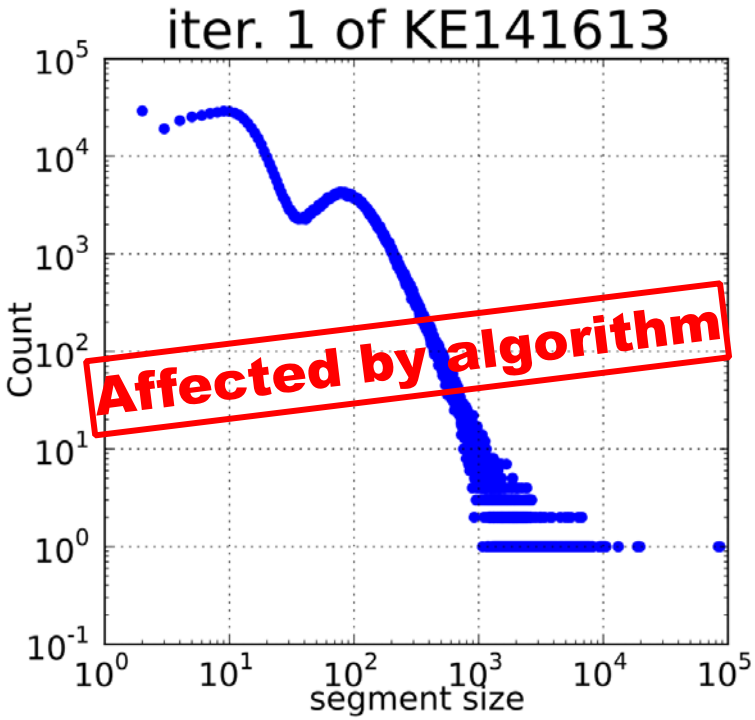


Why Segmented Sort?

- 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*

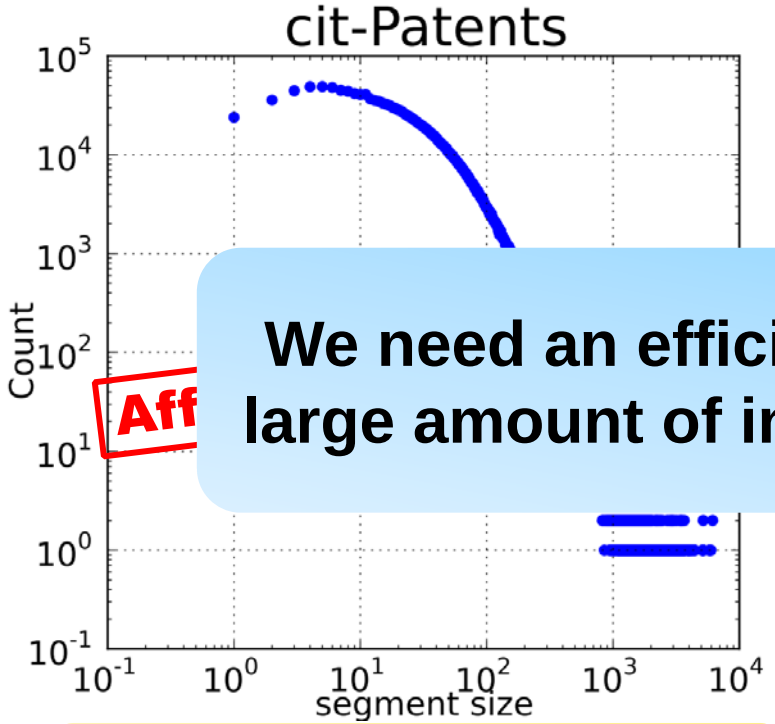


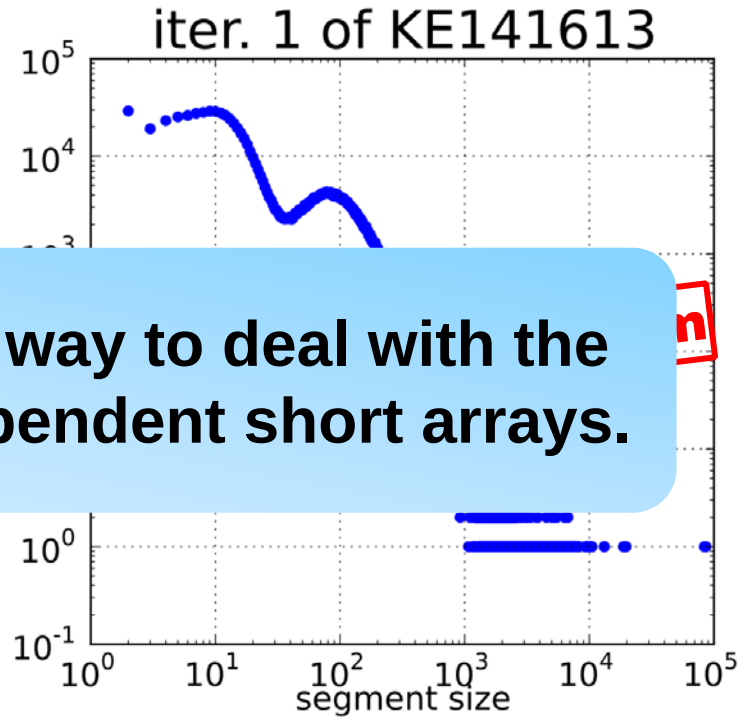
Segment statistics from 1st iteration in SAC*

* 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.

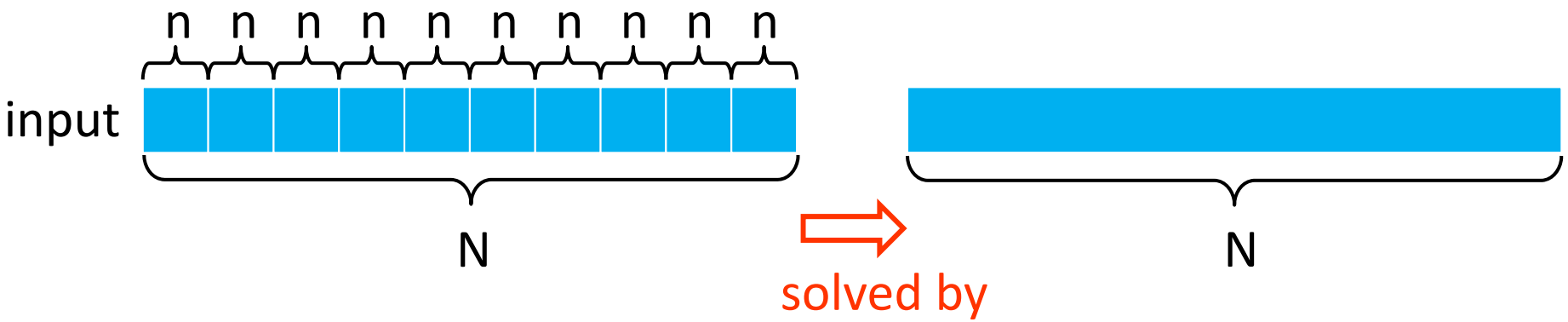
Segment statistics from squaring one matrix in SpGEMM*

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 $\boldsymbol{O\left(\frac{N}{n}nlogn\right) \approx O(Nlogn)}$*

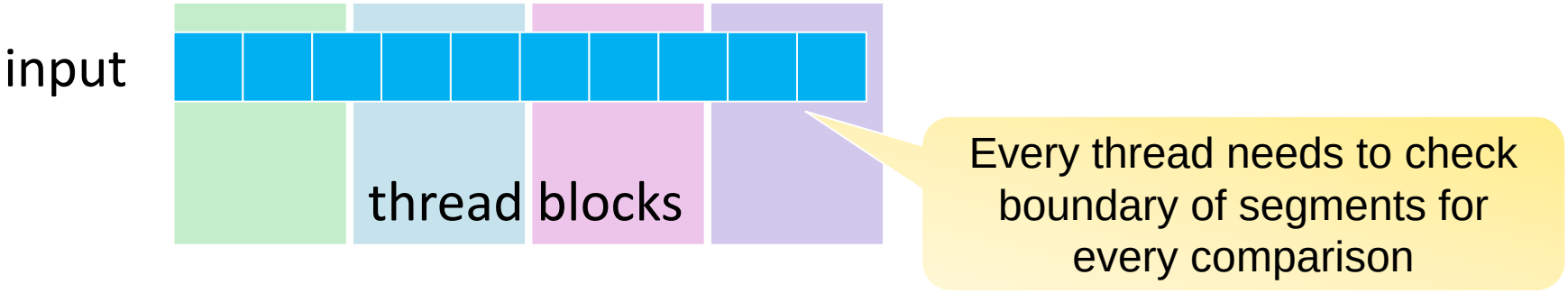
The complexity of the global sort is $\boldsymbol{O(NlogN)}$*

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

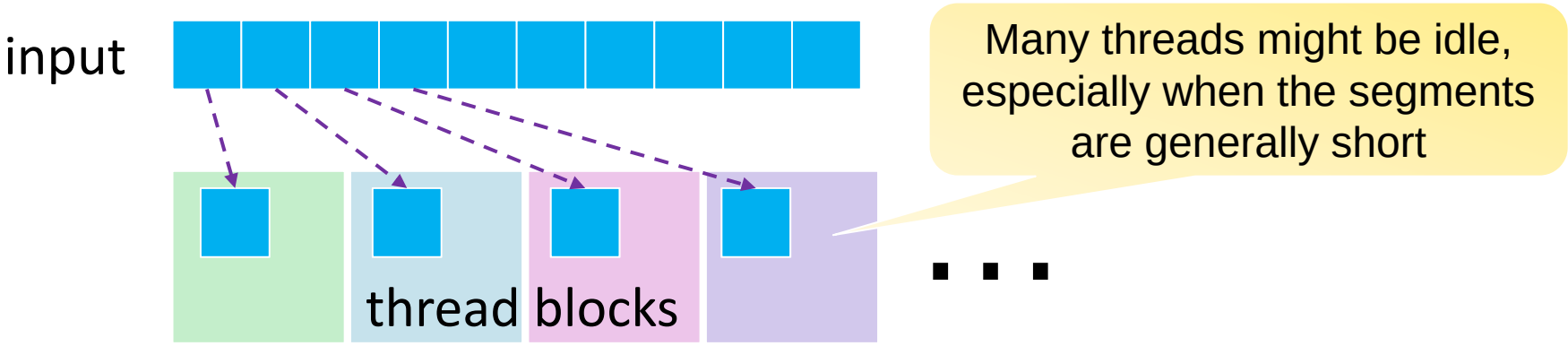
- Global sort has received much more fanfare!
- Many tools are evolved from global sort; however, there are also problems
  - Problem 1: Time complexity
  - Problem 2: **Runtime boundary checking overhead**



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**



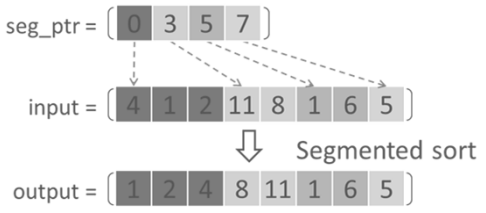
Some SegSort simply assigns each segment to each thread block, leading to idle resources, e.g., segsort from *CUB*

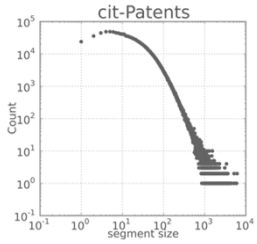
Fast Segmented Sort (this work)

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.

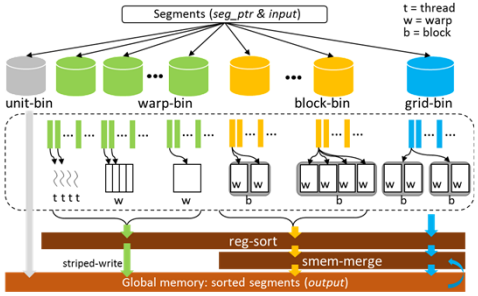
Outline (Data-Thread Binding)

- Introduction
- Motivation

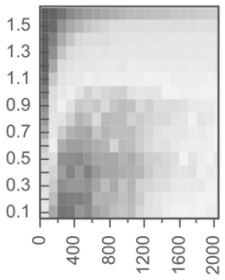


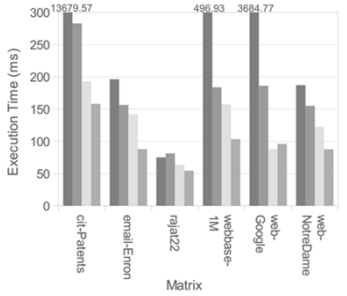


- Our Method
  - GPU SegSort Mechanism
  - GPU Register-based Sort
  - Other Techniques & Opt.



- Evaluation
  - Kernel Performance
  - Kernel in Real Applications

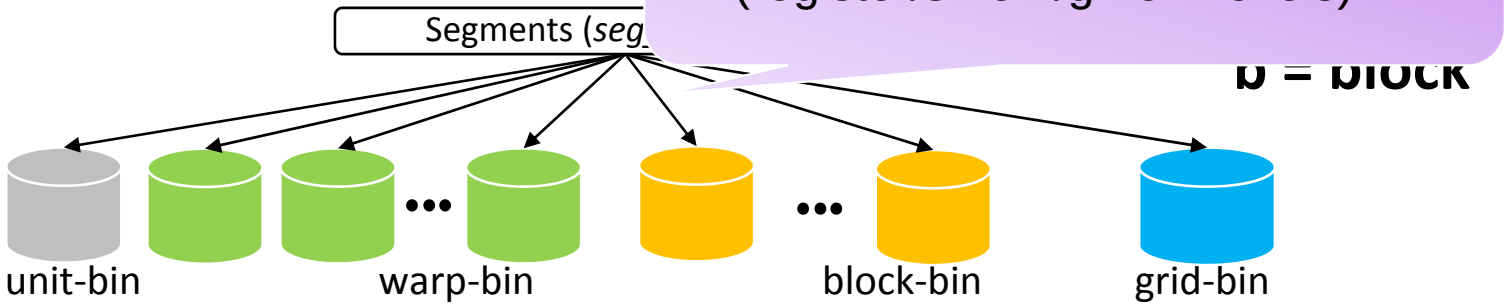




Adaptive GPU SegSort Mechanism

- Overview of our proposed (

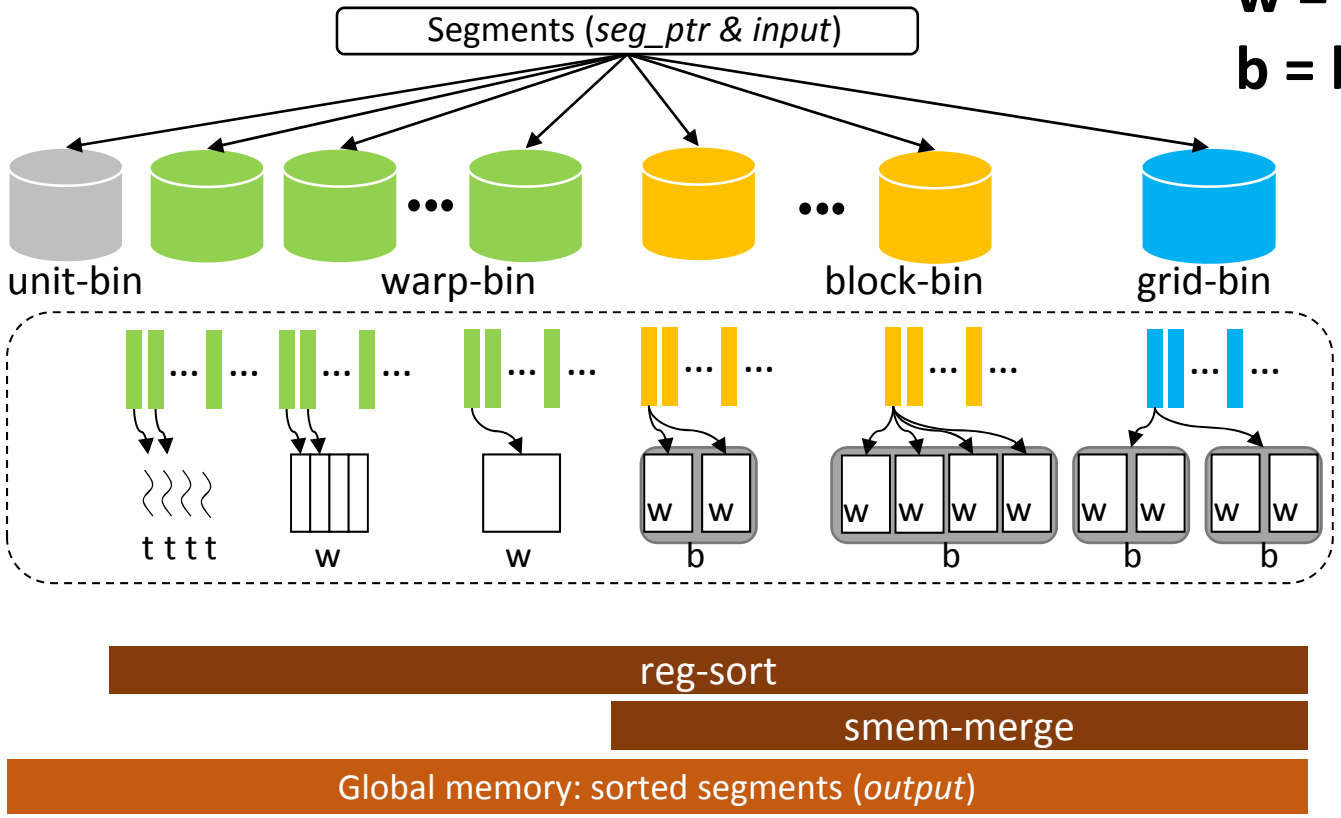
- Hierarchical binning (register/smem/gmem levels)



Adaptive GPU SegSort Mechanism

- Overview of our proposed GPU SegSort design

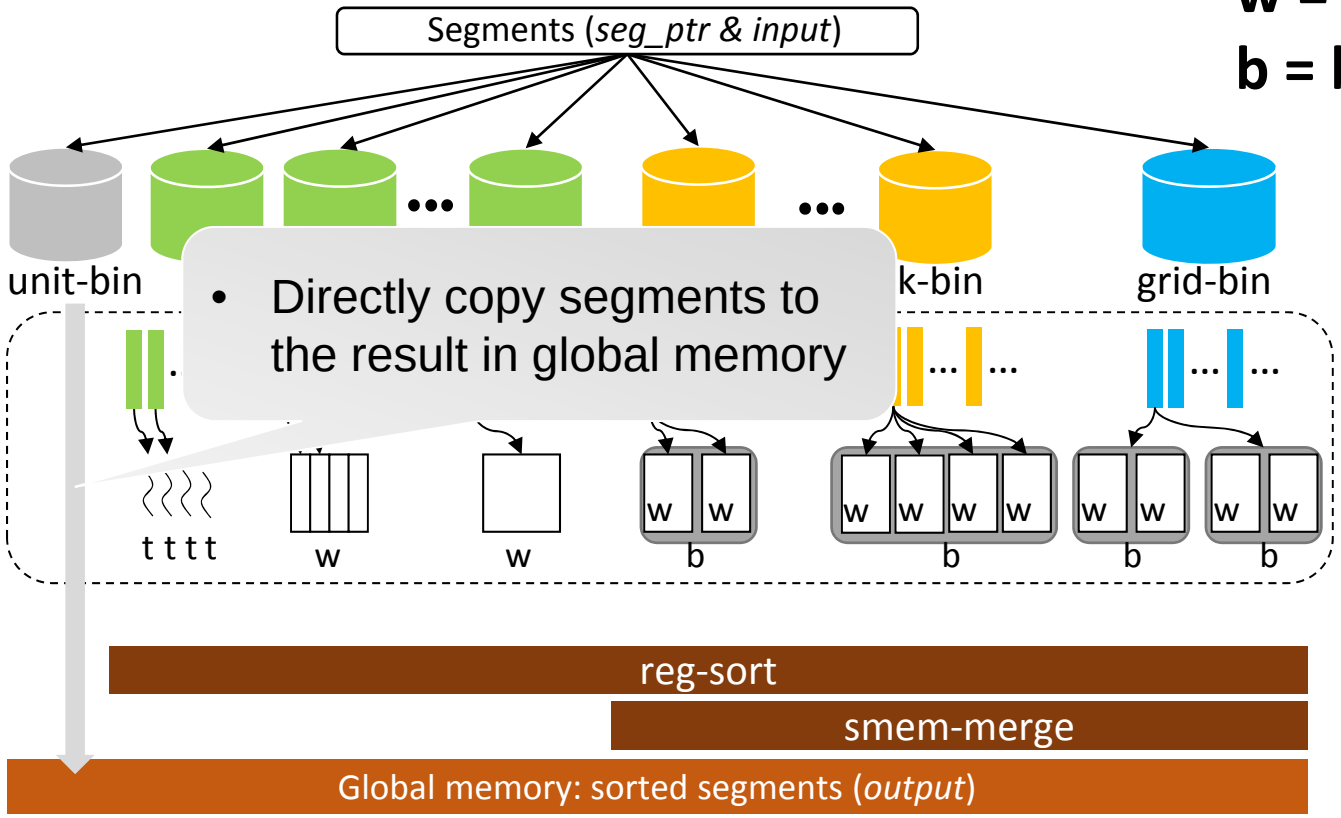
t = thread
w = warp
b = block



Adaptive GPU SegSort Mechanism

- Overview of our proposed GPU SegSort design

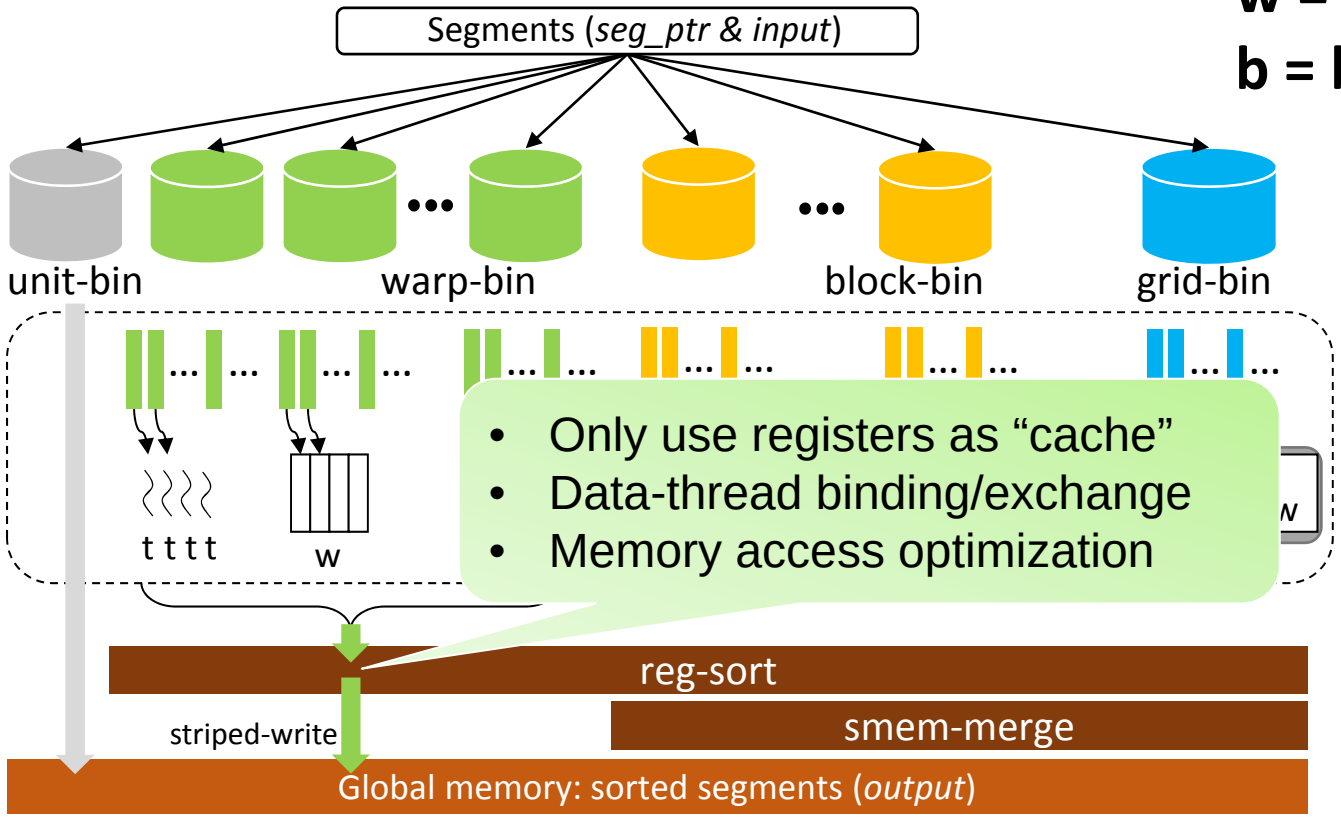
t = thread
w = warp
b = block



Adaptive GPU SegSort Mechanism

- Overview of our proposed GPU SegSort design

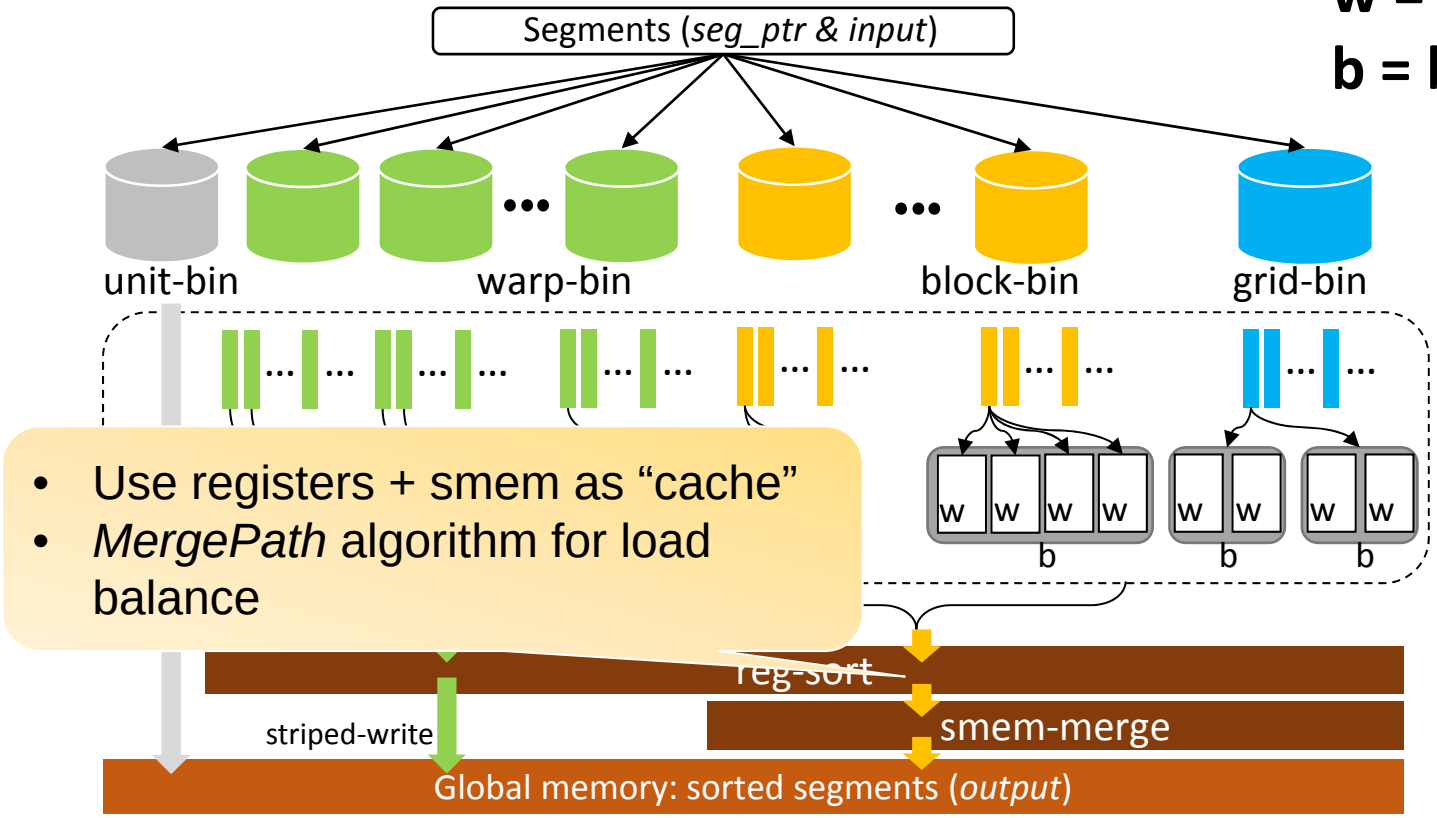
t = thread
w = warp
b = block



Adaptive GPU SegSort Mechanism

- Overview of our proposed GPU SegSort design

t = thread
w = warp
b = block

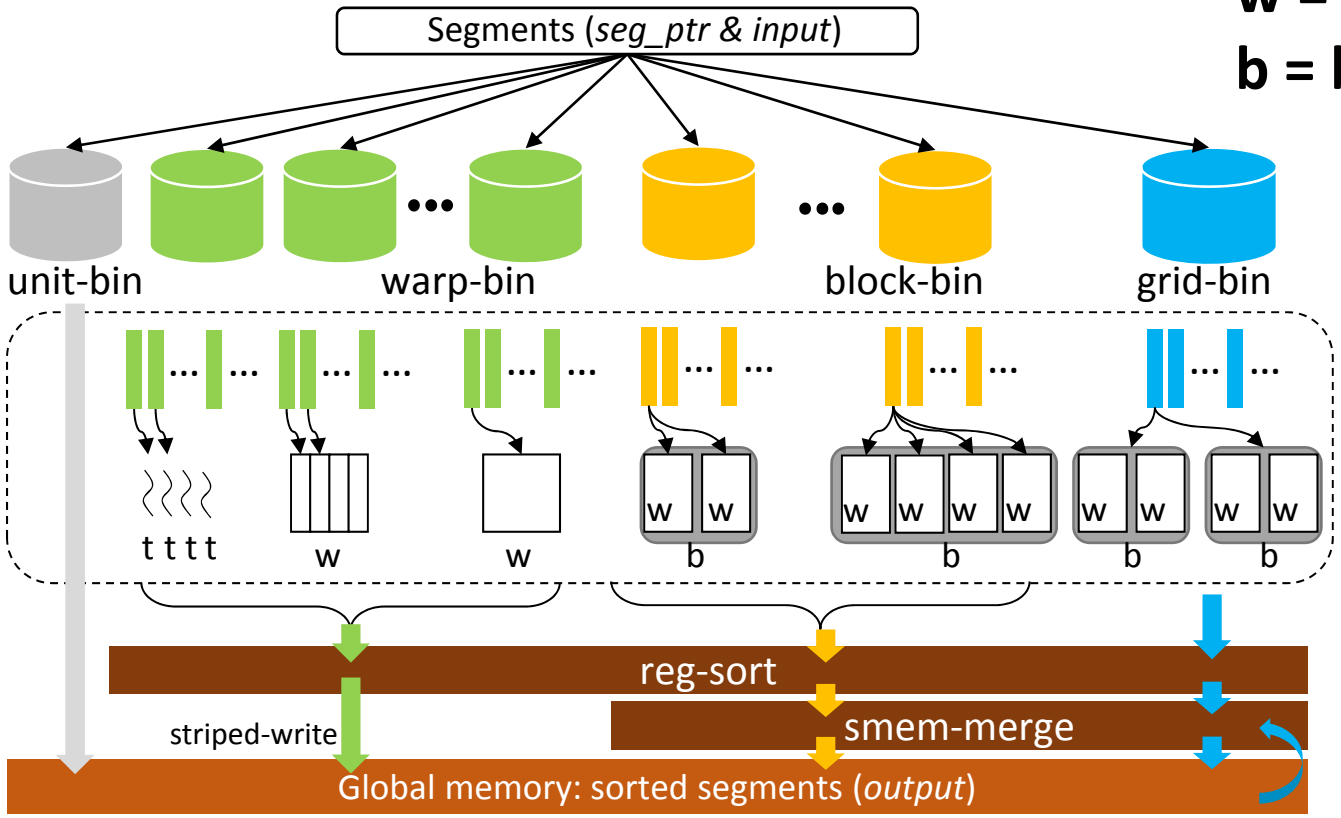


- Use registers + smem as "cache"
- *MergePath* algorithm for load balance

Adaptive GPU SegSort Mechanism

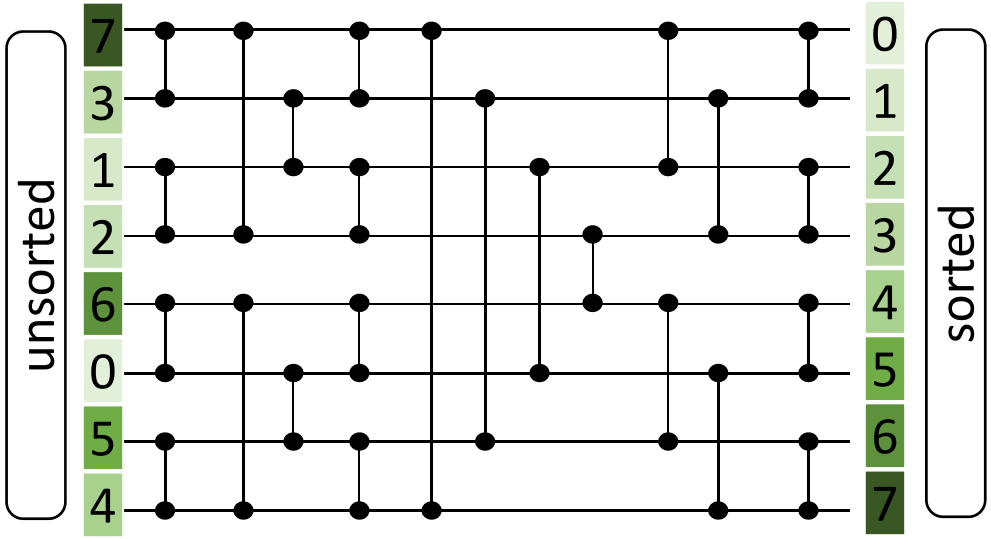
- Overview of our proposed GPU SegSort design

t = thread
w = warp
b = block



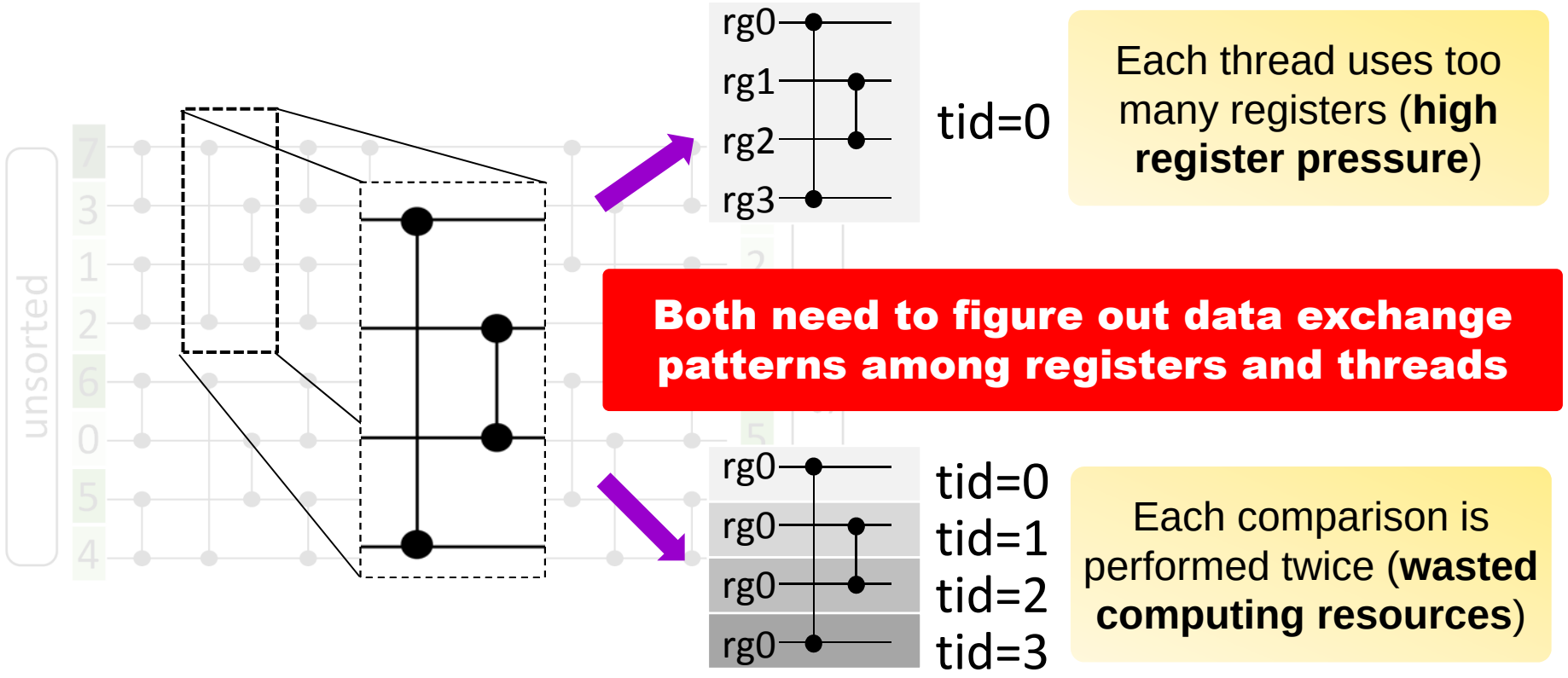
GPU Register-based Sort

- **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

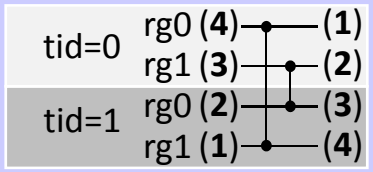
- **Sorting networks** usually serve as building blocks of efficient parallel sort
- How to bind the data items (operands) to different threads?



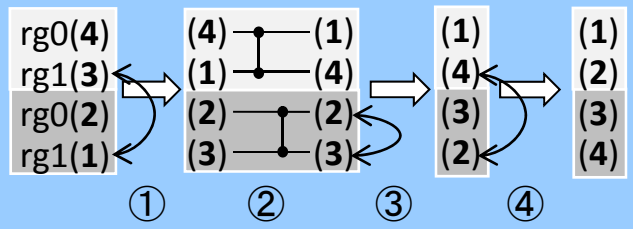
GPU Register-based Sort

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

*_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*

Two data items are bound to each thread

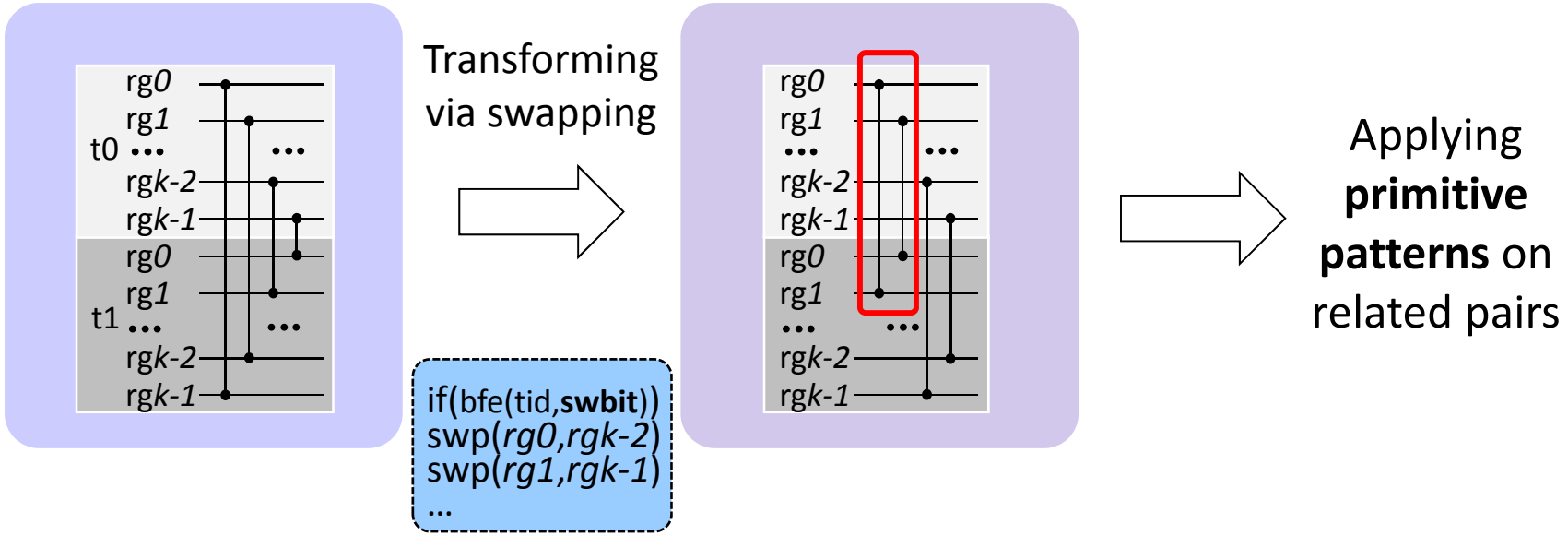
Tells which thread swaps registers

_exch_primitive(rg0, rg1, tmask, swbit)

Tells how threads communicate

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

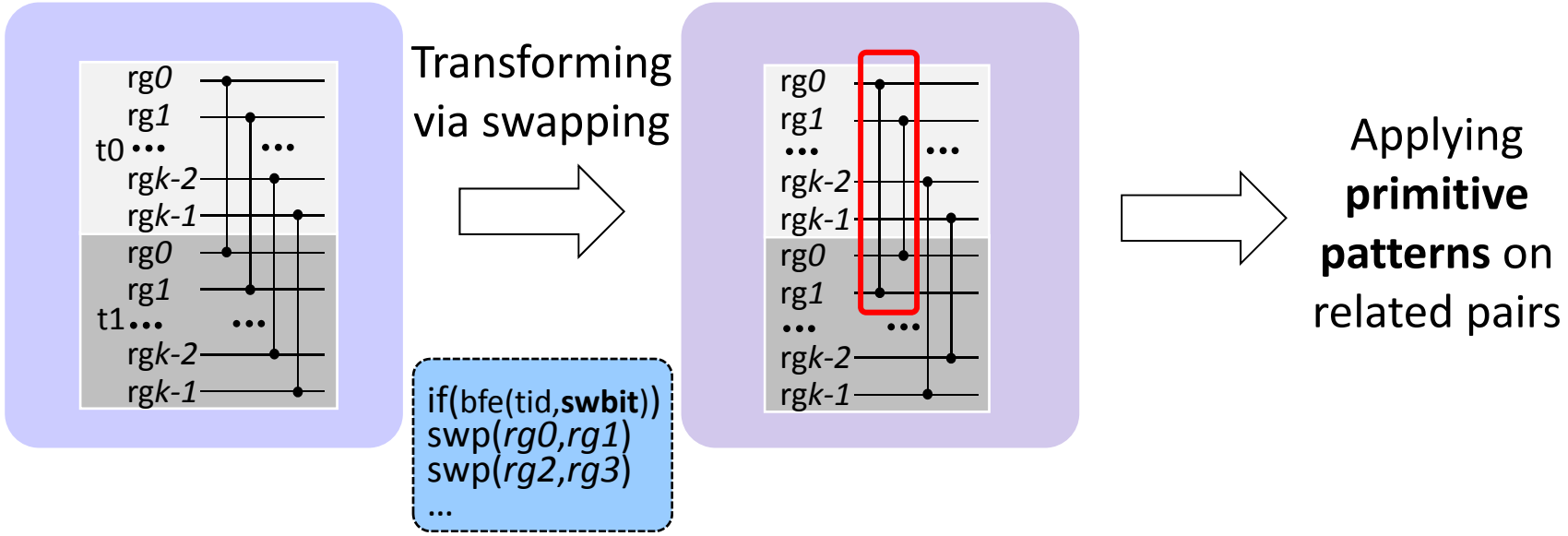
Tells which thread swaps registers

_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

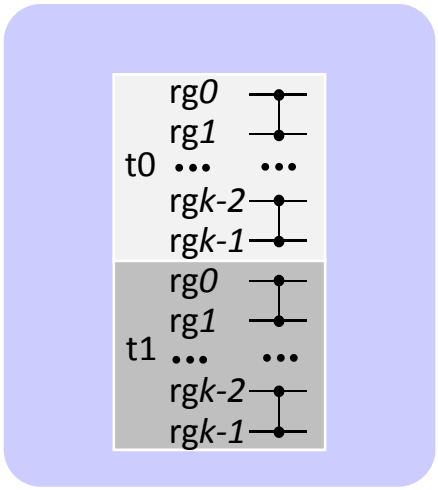
Tells which thread swaps registers

_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

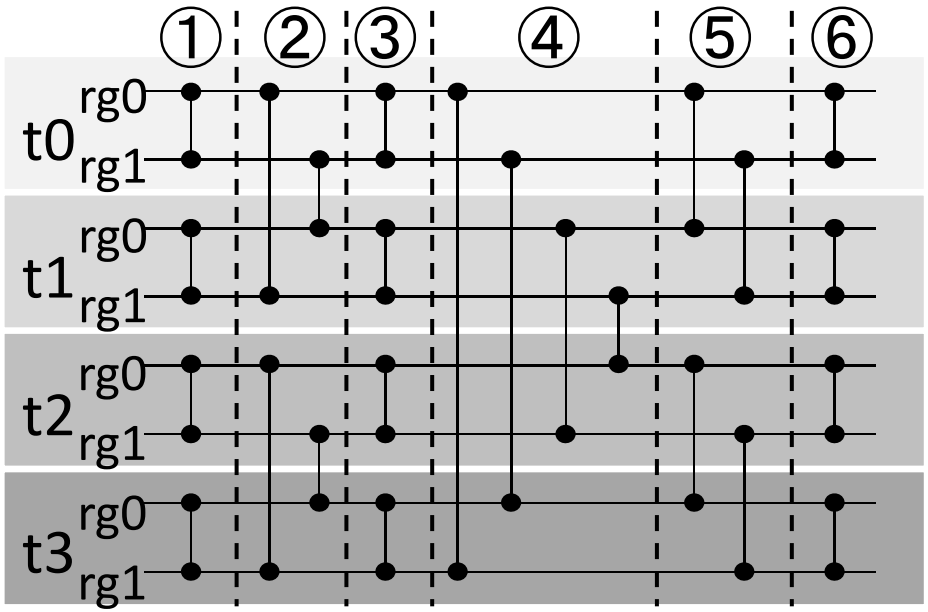
_exch_local(rg0, rg1, …, rgk-1, rmask)

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)

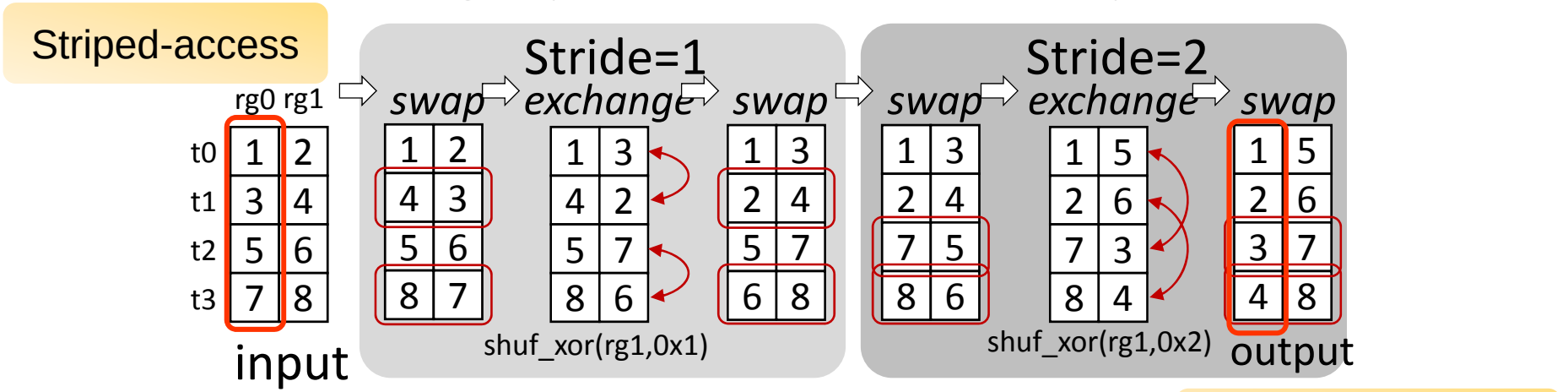


①**_exch_local**(*rg0*,*rg1*);
②**_exch_intxn**(*rg0*,*rg1*,**0x1**,**0**);
③**_exch_local**(*rg0*,*rg1*);
④**_exch_intxn**(*rg0*,*rg1*,**0x3**,**1**);
⑤**_exch_paral**(*rg0*,*rg1*,**0x1**,**0**);
⑥**_exch_local**(*rg0*, *rg1*);

More details in our paper show (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

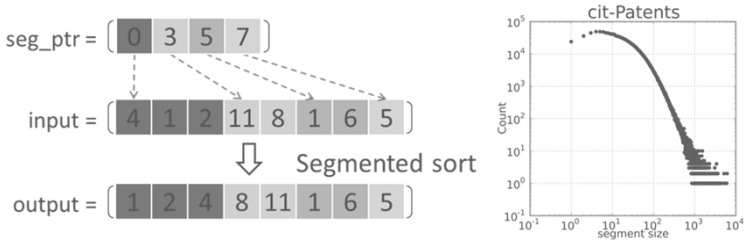
- A hierarchical binning
  - Using warp vote function __balloc() and __popc() at warp level
  - Using shared memory at thread-block level
- Better locality by optimizing access pattern
  - Transforming striped write to coalesced memory access



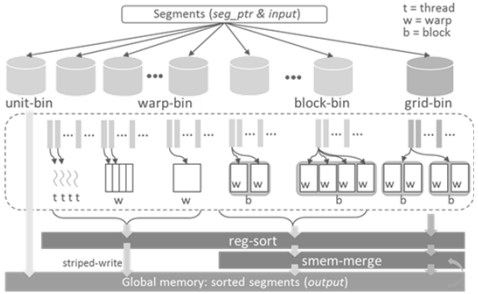
- Shared memory based merge solution
  - *MergePath algorithm [`12]* for load balance

Outline (Data-Thread Binding)

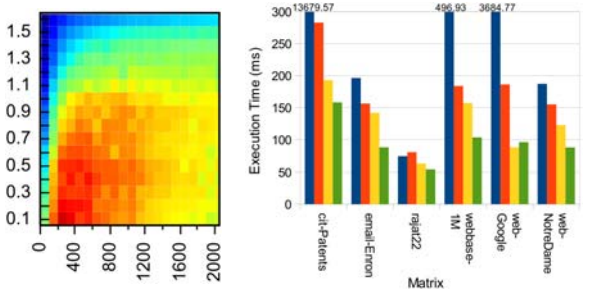
- Introduction
- Motivation



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



- Evaluation
  - Kernel Performance
  - Kernel in Real Applications



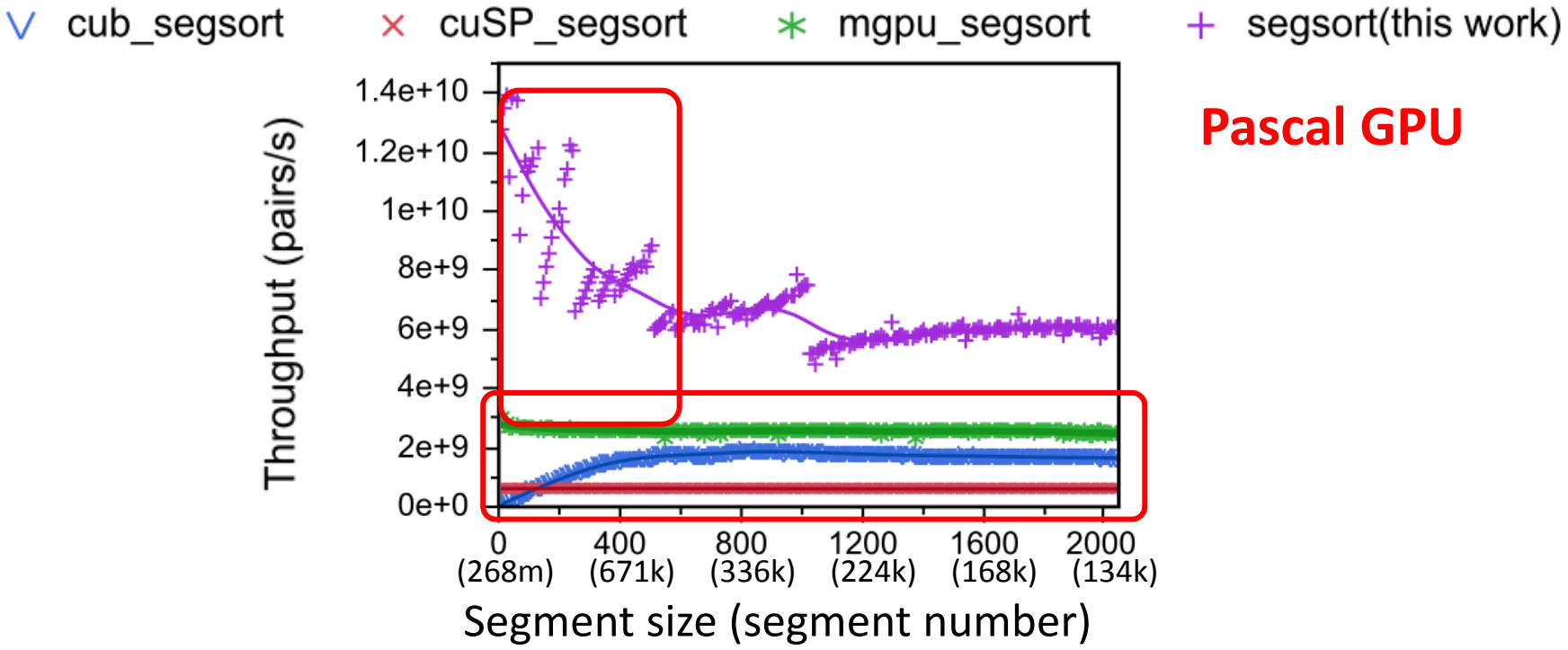
Experiment Platforms

- **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

* We only show the performance results of TitanX GPU in the presentation.
* CUSP performs segmented sort by using THRUST sort twice. We extract this as a stand-alone function.

SegSort Performance

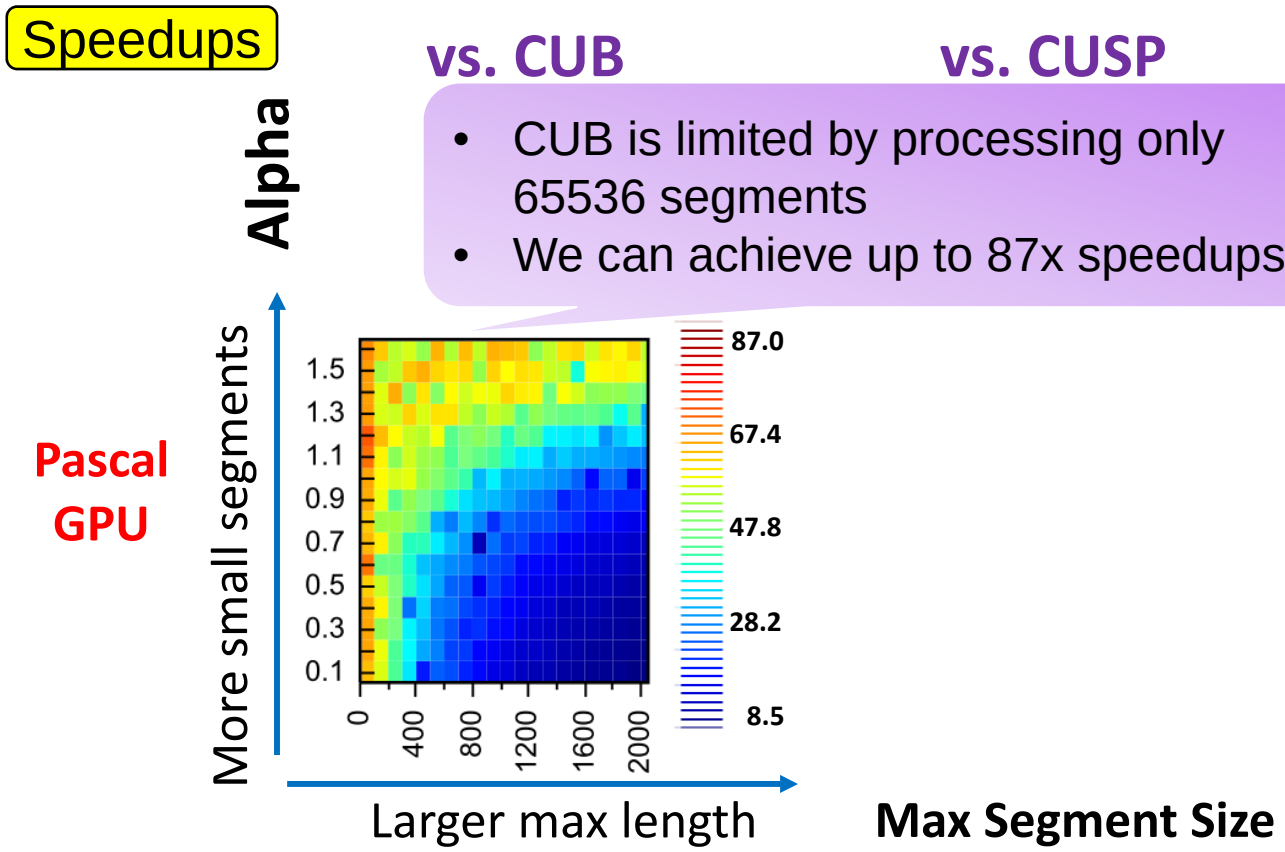
- Fixing total data size w/ variable segment number and size



  - 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.



- CUB is limited by processing only 65536 segments
- We can achieve up to 87x speedups

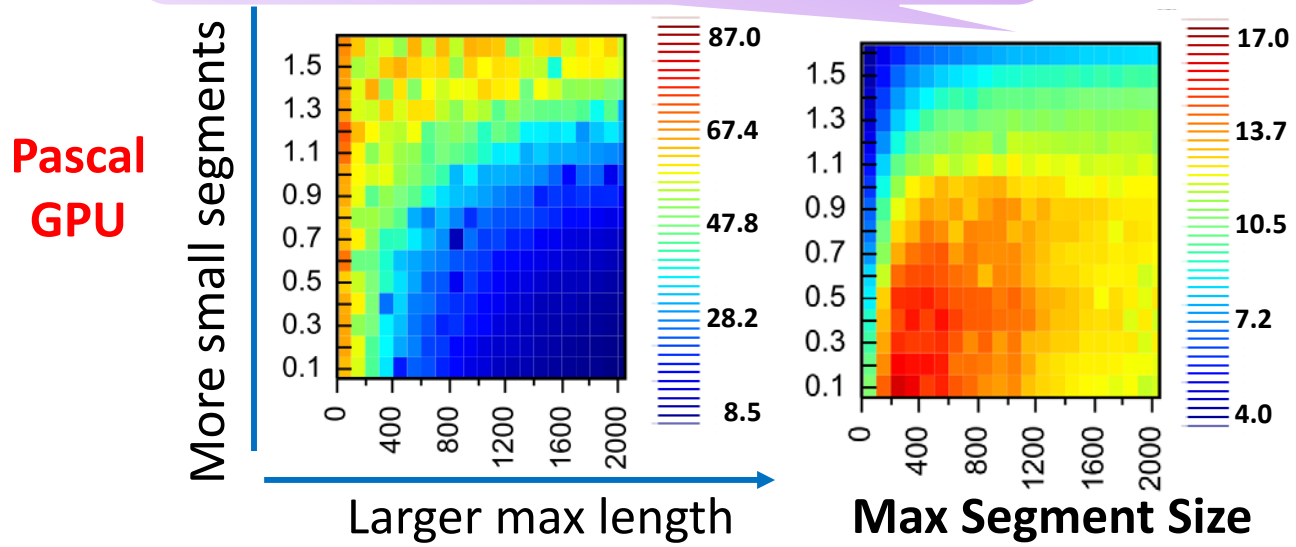
SegSort Performance

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

Speedups

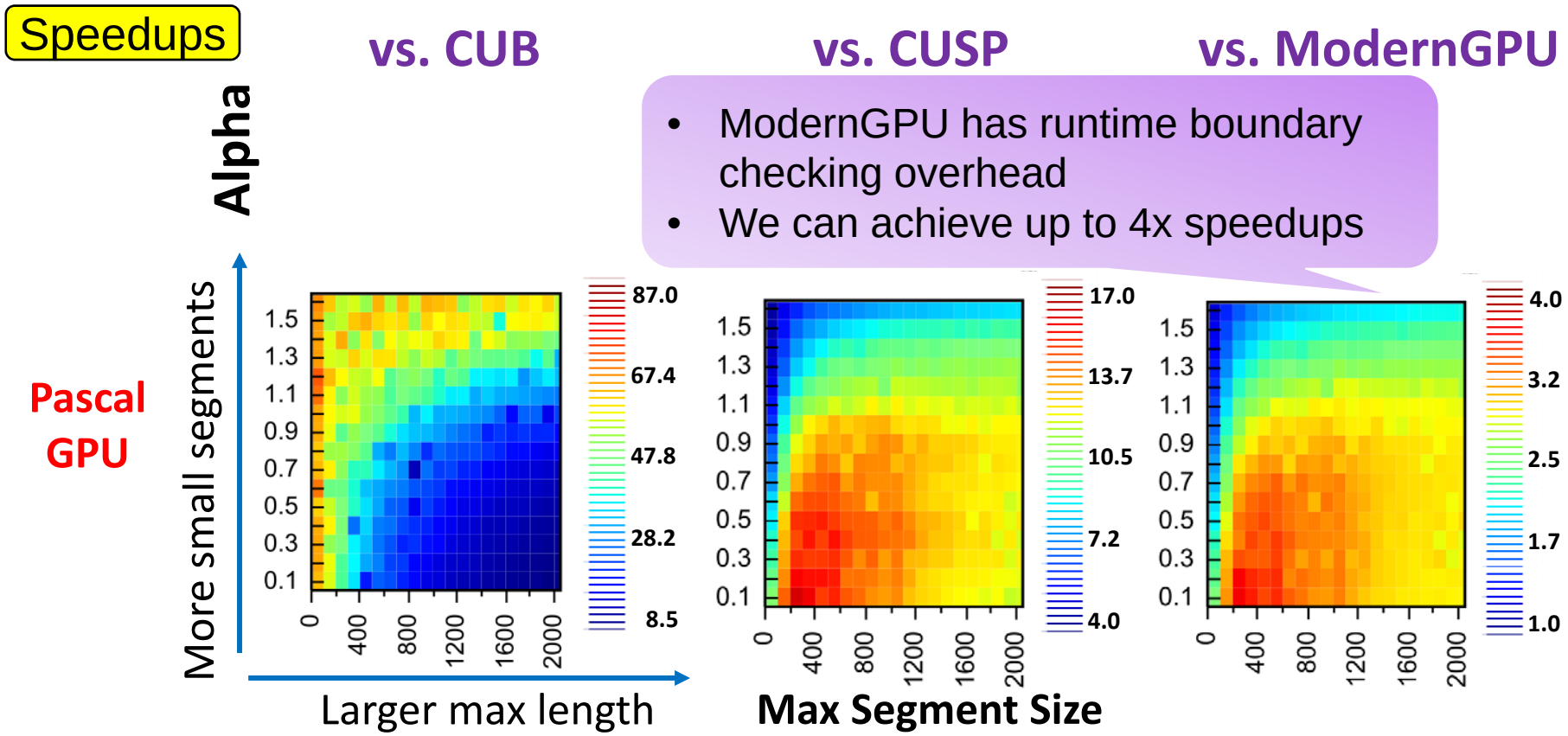
vs. CUB **vs. CUSP** **vs. ModernGPU**

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



SegSort Performance

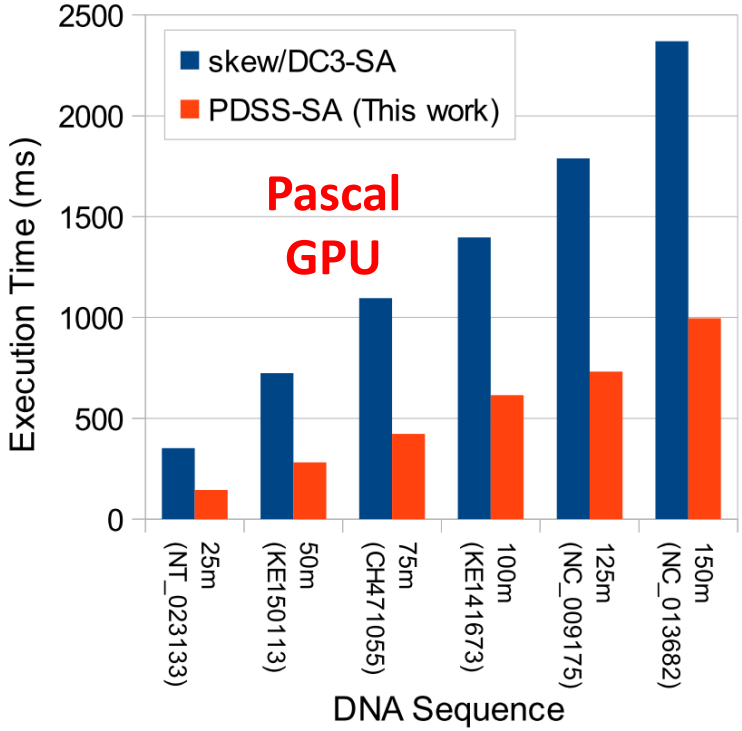
- Fixing total data size w/ segments of power-law distribut.



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

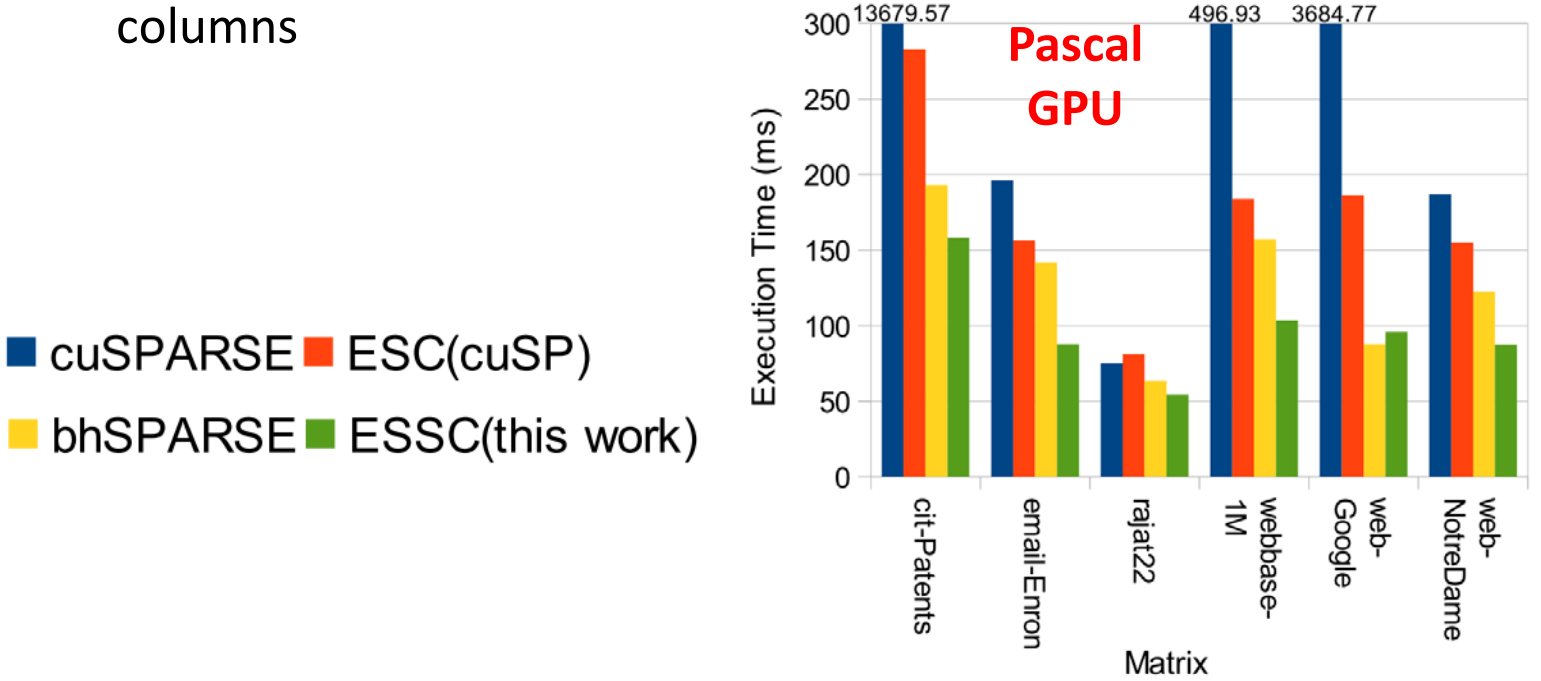
SegSort in Real-world Applications

- **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 (SpGEMM): multiply a sparse matrix *A* with another sparse matrix *B* and obtains a resulting sparse matrix *C*
- Our method is using the *Expansion*, ***Seg-Sorting*** and *Compression* (ESSC) algorithm [`12]
  - Sorting an intermediate sparse matrix $\hat{C}$ by its indices of rows and columns

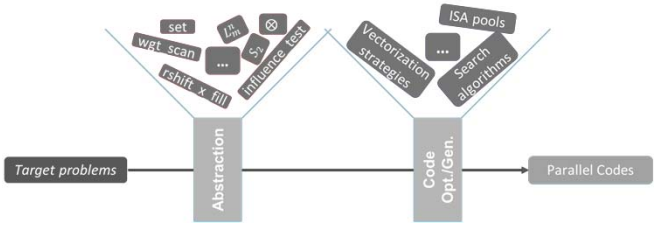


Summary of this Work

- Propose a general solution to handle different data-thread binding cases.
- Identify 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

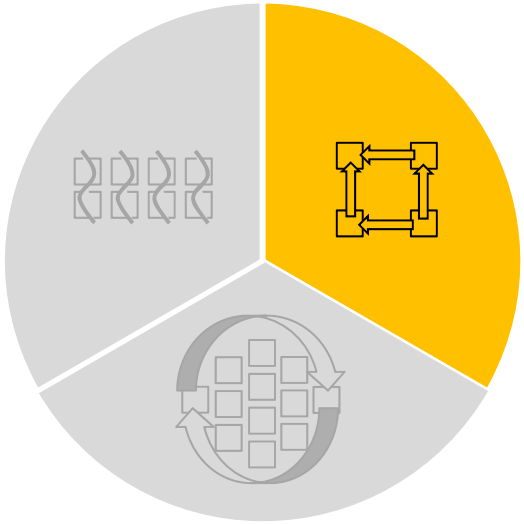
Outline of the Talk

- Motivation
- Contribution & Papers



- Previous Work

- **Our Methods**
  - *Data-reordering (covered in Prelim)*
  - *SIMD Operations (covered in Prelim)*
  - Data-thread Binding (seg_sort)
  - **Data Dependencies (wavefront)**
  - Data Reuse (stencils)



- Summary and Future Work

Wavefront Loops

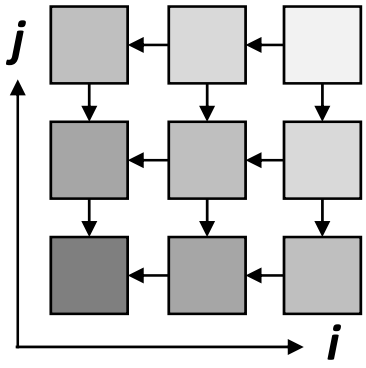
- Update each entry of a workspace grid based on the already-updated values from its neighbors
- Used in many scientific applications, e.g., PDE solver, sequence alignment tools, etc.

Example: a wavefront loop (2D matrix)

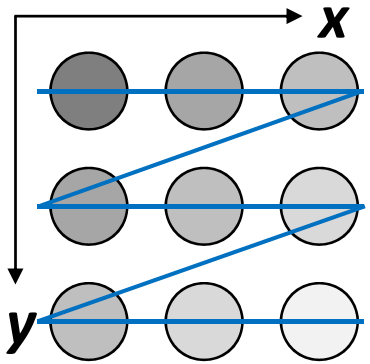
```
for(int i = 0; i < m; i++)
  for(int j = 0; j < n; j++)
    A[i][j] = A[i][j-1] * 0.5 + A[i-1][j] * 0.5;
```

Neither loop can be parallelized.

Data Dependence (Iteration Space)



Memory Access (Memory Space A[y][x]**)**



Wavefront Loops

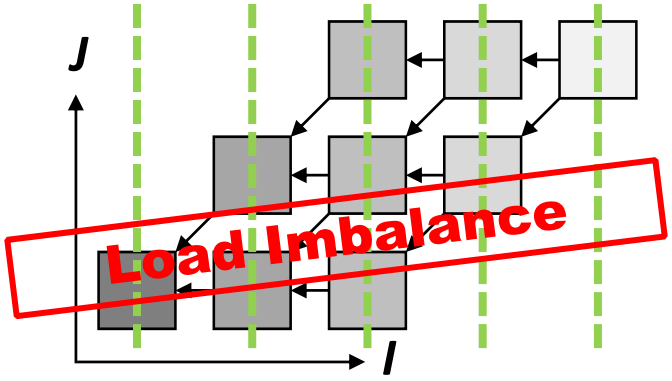
- Update each entry of a workspace grid based on the already-updated values from its neighbors
- Used in many scientific applications, e.g., PDE solver, sequence alignment tools, etc.

Example: a wavefront loop (2D matrix) -- Tra

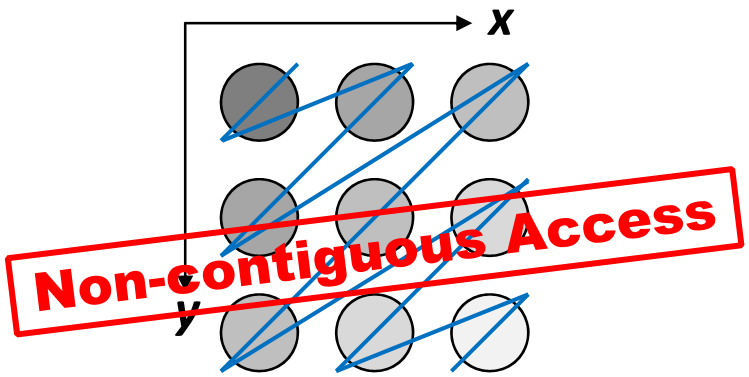
J-loop can be parallelized.

```
for(int I = 0; I < m+n-1; I++)
  for(int J = max(0, I-n+1); J < min(m, I+1); J++)
    A[J][I-J] = A[J][I-J-1] * 0.5 + A[J-1][I-J] * 0.5;
```

Data Dependence (Iteration Space)

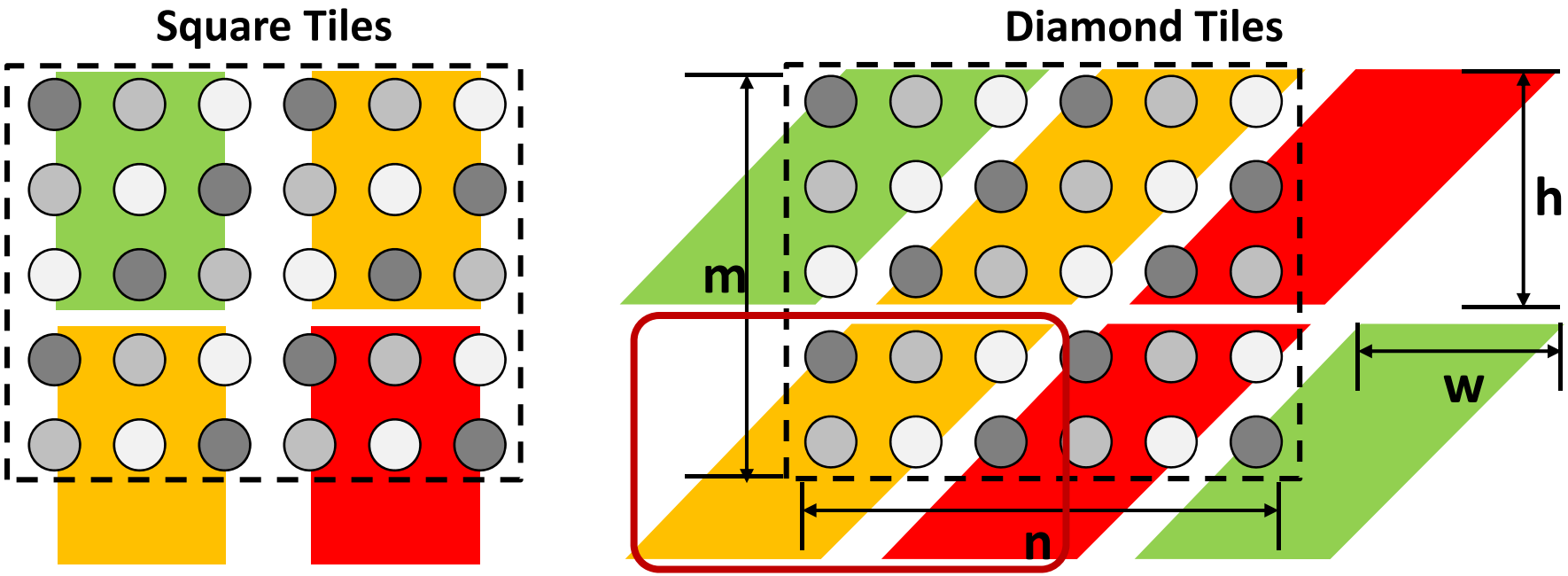


Memory Access (Memory Space A[y][x]**)**



Existing Parallel Solutions

- Tiling-based solutions and their limitations
  - Problem 1: **Wasted memory and computing resources**



Tiles with same color can be executed in parallel

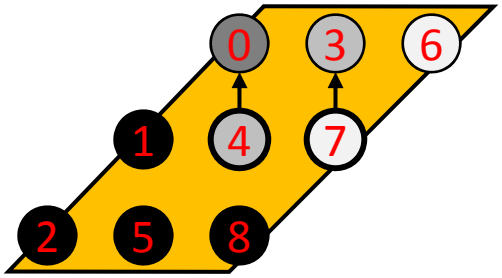
Non-contiguous memory access still exists

Much memory space will be wasted (The rate of effective memory usage ≈$nl(n+h)$)

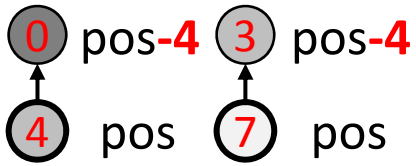
Existing Parallel Solutions

- Tiling-based solutions and their limitations
  - Problem 1: **Wasted memory and computing resources**

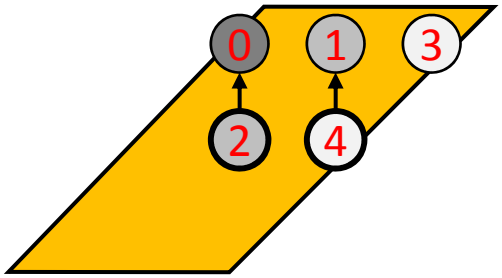
Padding



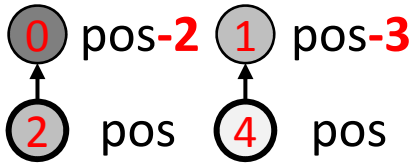
Uniform access pattern



No padding



Diverged access pattern



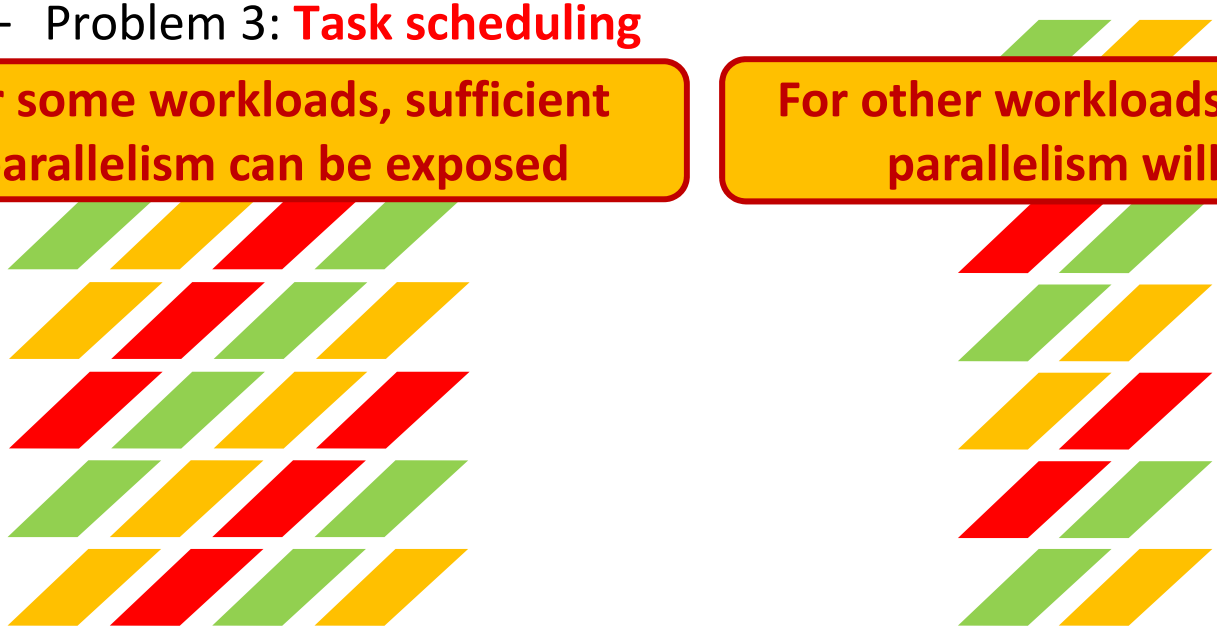
Padding-free strategy may greatly increase the complexity of indexing and lead to more branches in GPU kernels

Existing Parallel Solutions

- Tiling-based solutions and their limitations
  - Problem 1: Wasted memory and computing resources
  - Problem 2: **Layout transformation overhead**
  - Problem 3: **Task scheduling**

For some workloads, sufficient parallelism can be exposed

For other workloads, insufficient parallelism will be met



Tiles with same color can be executed in parallel

For some workloads, tiling-based solution may lose efficiency, because of the small amount of tiles along anti-diagonals

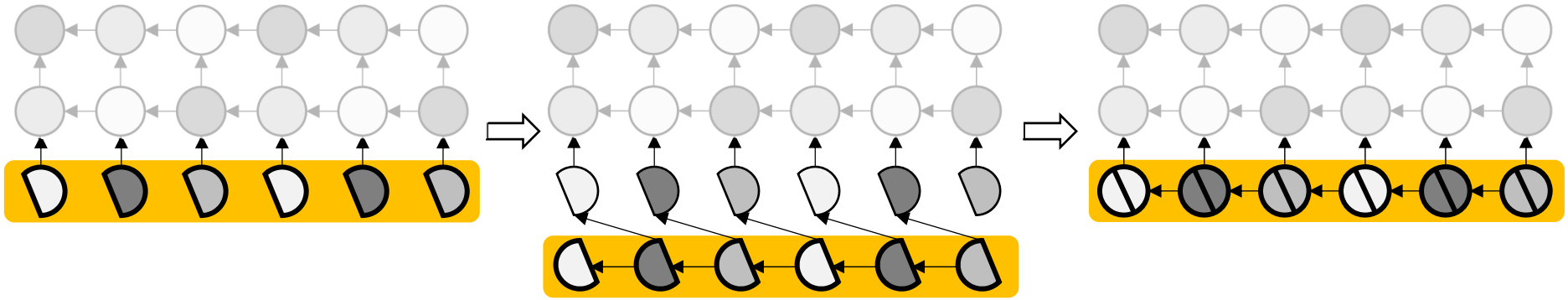
Existing Parallel Solutions

- Compensation-based solutions and their limitations
  - Problem 1: **Global synchronizations**
  - Problem 2: **Limited usage in sequence alignment algorithms**

❶ Compute partial results by ignoring horizontal dependency

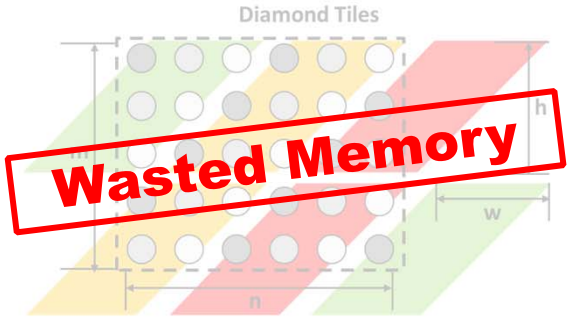
❷ Compensate the partial results

❸ Combine the results from ❶ and ❷



Multiple expensive global synchronizations are required for processing each row; the compensation-based solution works well for string matching operations

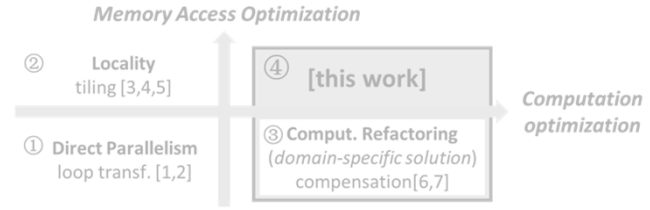
Highly Efficient Wavefront Parallelism (this work)



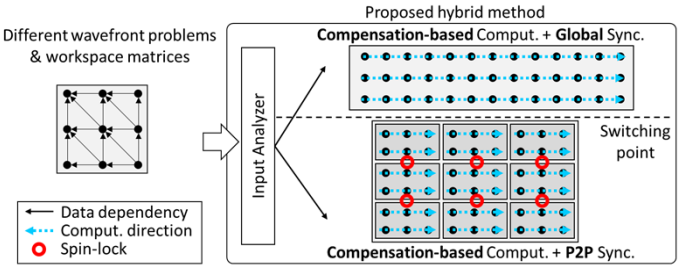
(1) Can the compensation-based method be used to optimize general wavefront loops?
(2) Is the compensation-based method sufficient for any types of workloads?

Outline (Data Dependencies)

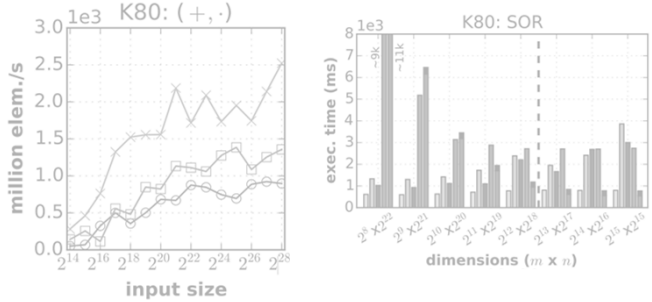
- Introduction
- Motivation



- Our Method
  - Compensation-based Method
  - GPU Implementation
  - Hybrid Parallel Strategy



- Evaluation
  - Weighted-scan Kernel Performance
  - Wavefront Kernel Performance

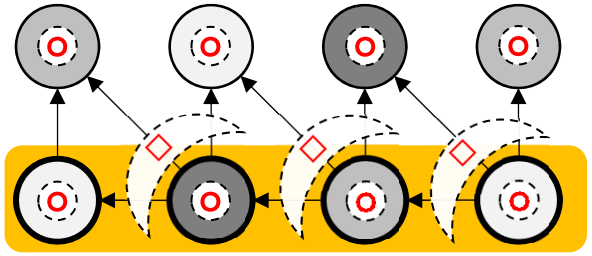


Compensation-based Method

- Wavefront Pattern

$$A_{i,j} = (A_{i,j-1} \circ b_0) \diamond (A_{i-1,j} \circ b_1) \diamond (A_{i-1,j-1} \circ b_2)$$

- $\circ$ *generic distribution operator (for adding weights)*
- $\diamond$ *generic accumulation operator (for adding neighbors)*



- Compensation-based Method

Step 1: $\tilde{A}_{i,j} = (A_{i-1,j} \circ b_1) \diamond (A_{i-1,j-1} \circ b_2)$

Step 2: $B_{i,j} = \begin{cases} \sum_{u=0}^{j-1} (\tilde{A}_{i,u} \circ \prod_{v=u}^{j-1} b_0) & \text{when } \circ \neq \diamond \\ \sum_{u=0}^{j-1} (\tilde{A}_{i,u} \diamond b_0) & \text{when } \circ = \diamond \end{cases}$

Step 3: $A_{i,j} = \tilde{A}_{i,j} \diamond B_{i,j}$

This is valid when (1) $\circ$ has the distributive property over $\diamond$; (2) $\circ$ is same with $\diamond$. *

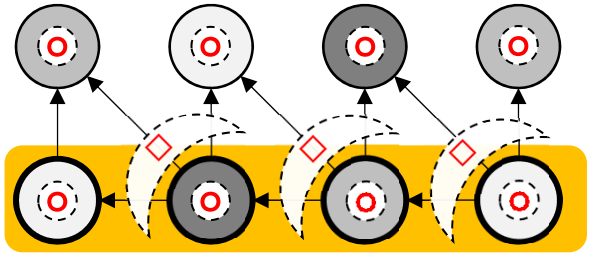
* The mathematic proof is included in our IPDPS'18 paper.

Compensation-based Method

- Wavefront Pattern

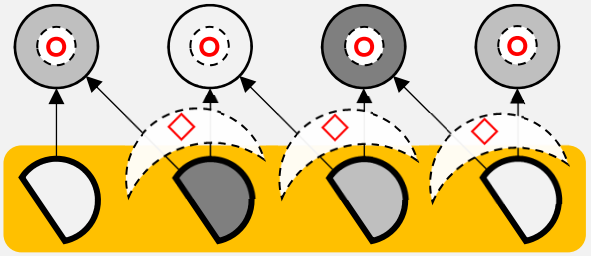
$$A_{i,j} = (A_{i,j-1} \circ b_0) \diamond (A_{i-1,j} \circ b_1) \diamond (A_{i-1,j-1} \circ b_2)$$

- $\circ$ *generic distribution operator (for adding weights)*
- $\diamond$ *generic accumulation operator (for adding neighbors)*



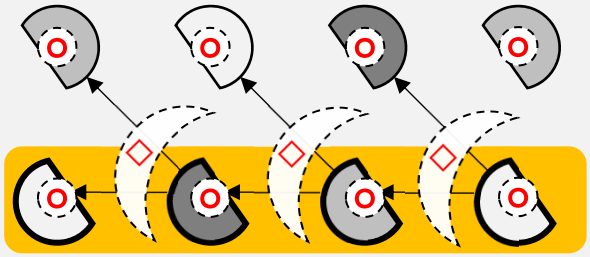
- Compensation-based Method

Step 1: $\tilde{A}_{i,j} = (A_{i-1,j} \circ b_1) \diamond (A_{i-1,j-1} \circ b_2)$



Step 2:

$$B_{i,j} = \begin{cases} \sum_{u=0}^{j-1} (\tilde{A}_{i,u} \circ \prod_{v=u}^{j-1} b_0) & \text{when } \circ \neq \diamond \\ \sum_{u=0}^{j-1} (\tilde{A}_{i,u} \diamond b_0) & \text{when } \circ = \diamond \end{cases}$$



Step 3: $A_{i,j} = \tilde{A}_{i,j} \diamond B_{i,j}$

Compensation-based Method

- The real-world wavefront loops can be expressed in the compensation-based parallelism patterns
- SOR (*Successive Over-relaxation*) Solver:
  - (◇ , ○) maps to (+, ·)

```
A[i][j] = (A[i][j] + A[i][j-1] + A[i-1][j] +
           A[i+1][j] + A[i][j+1]) / 5;
```

- SW (Smith-Waterman):
  - (◇ , ○) maps to (max, +)

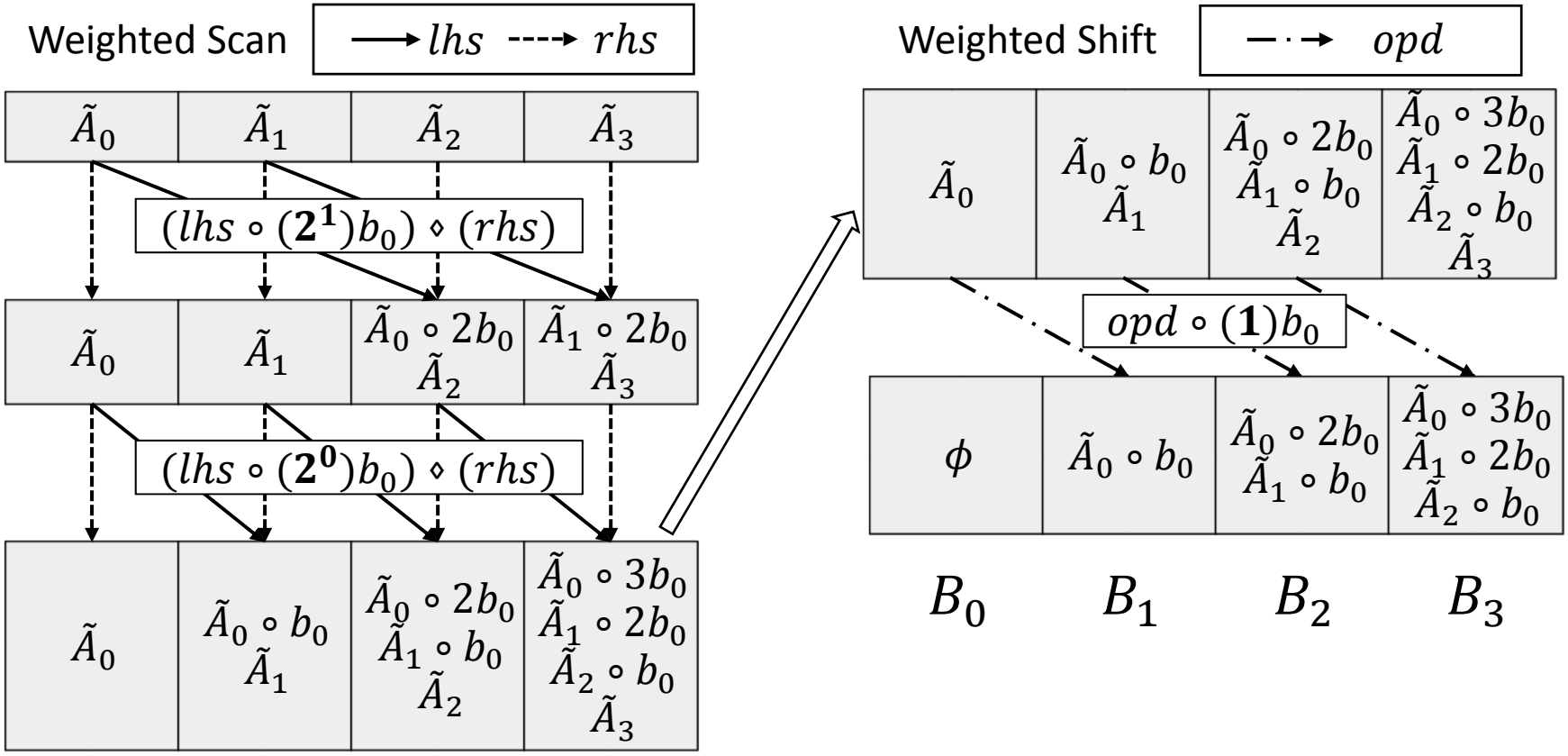
```
A[i][j] = max(A[i][j-1] - 2, A[i-1][j] - 2,
              A[i-1][j-1] + s(i,j), 0);
```

- SAT (Summed-area Table):
  - (◇ , ○) maps to (+, +)

```
A[i][j] = p[i][j] + A[i][j-1] + A[i-1][j] - A[i-1][j-1];
```

GPU Implementation

- Step 2 of the compensation-based method is the critical part: "***Weighted Scan***"*



* which also includes a weighted shift operation

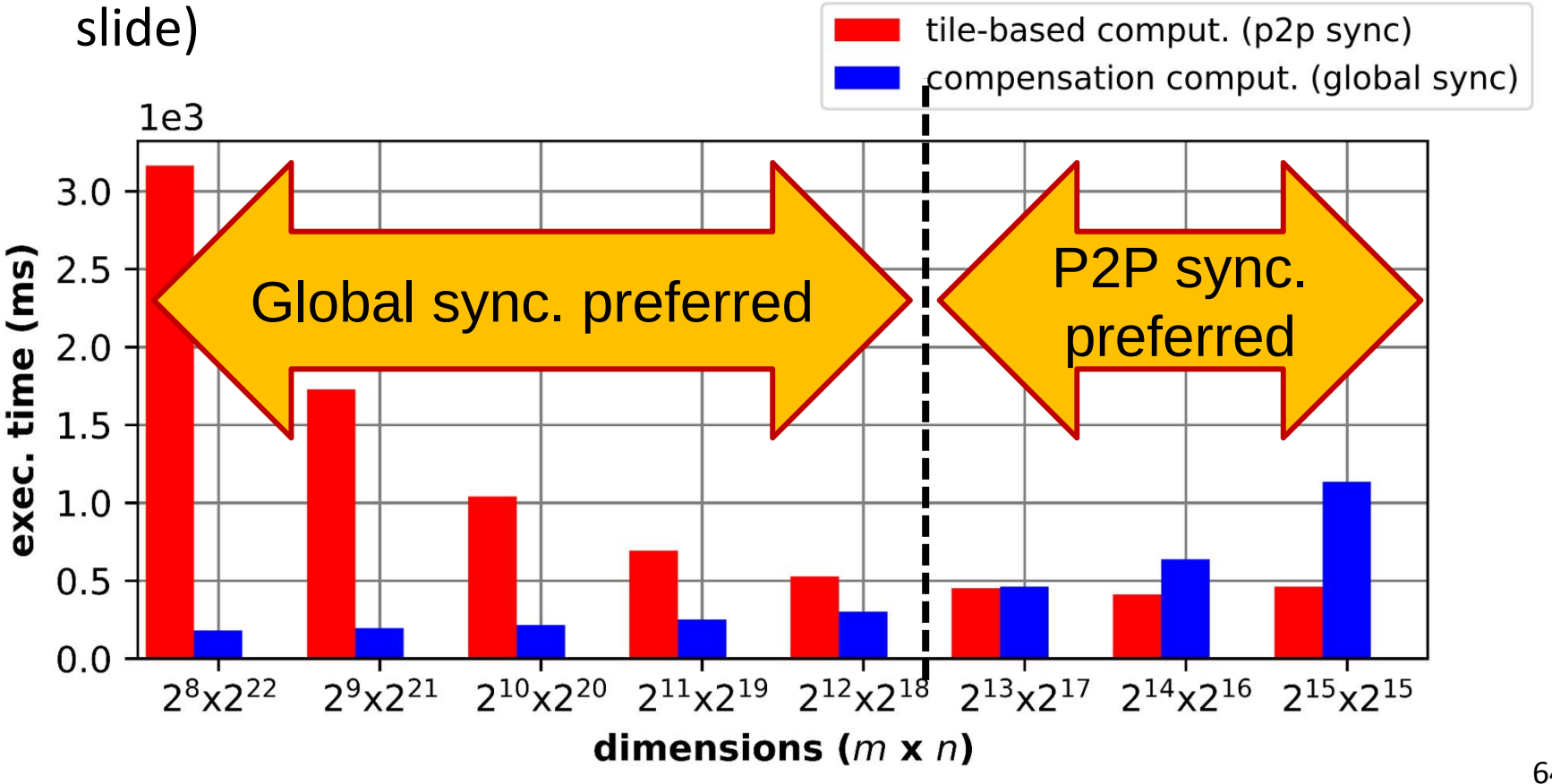
GPU Implementation

- Step 2 of the compensation-based method is the critical part: "***Weighted Scan***"
- Our algorithm handles the changing weights during each stages of the operations
- A hierarchical design is used for GPUs
  - *Register level*: compute how the preceding neighbor affects the current one via *data shuffle instructions*
  - *Shared memory level*: compute how the preceding **"warp"*** of neighbors affect the current one via *shared memory access*
  - *Global memory level*: compute how the preceding **"block"*** of neighbors affect the current one via *global memory access*

* which are thread organization units in NVIDIA GPU terminology

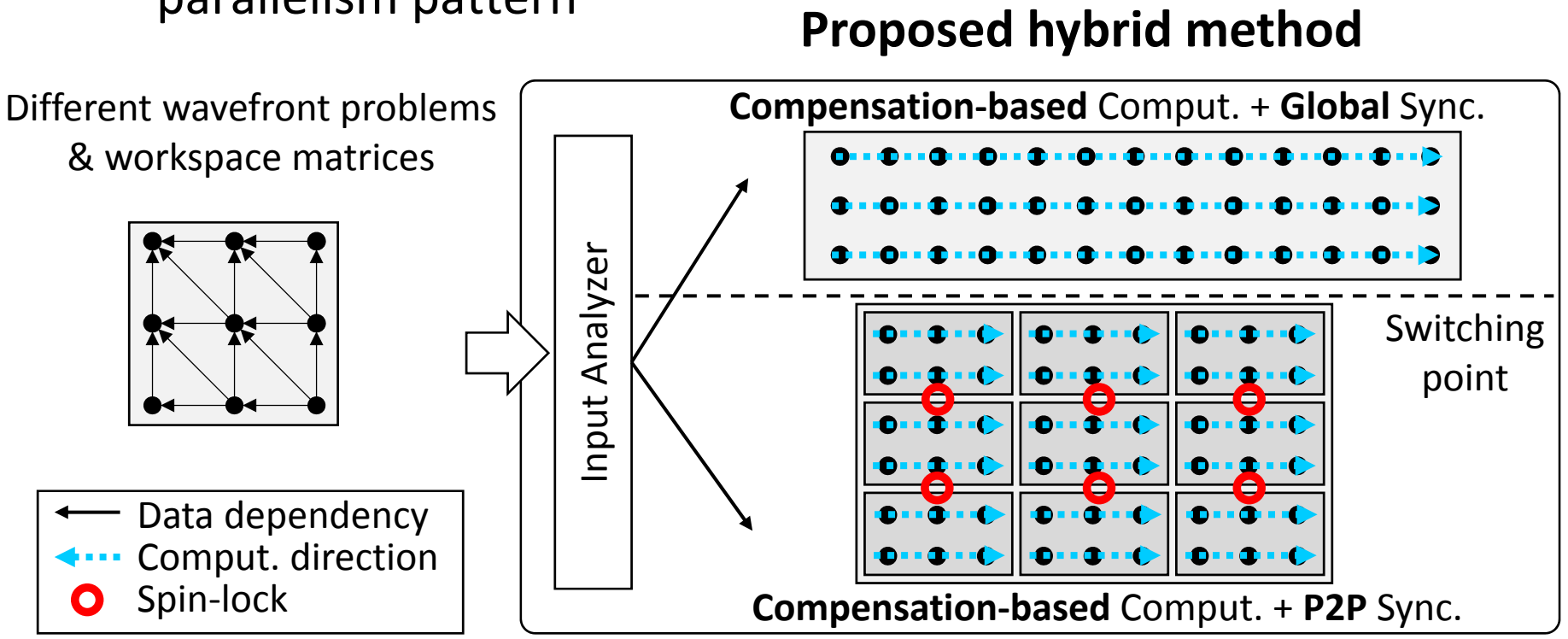
Hybrid Parallel Strategy

- *Is the compensation-based method sufficient for any types of workloads?*
- Observations (using the wavefront shown in the beginning slide)



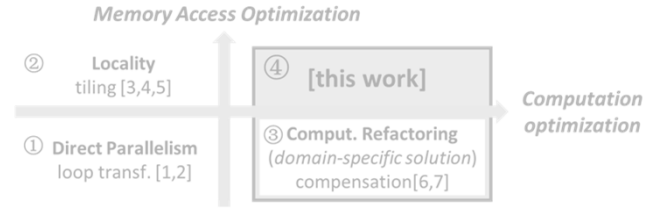
Hybrid Parallel Strategy

- Our hybrid design can switch to the appropriate parallel method according to the input workloads
- All the computation follows the compensation-based parallelism pattern

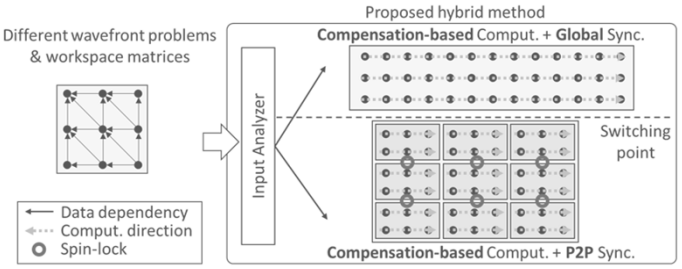


Outline (Data Dependencies)

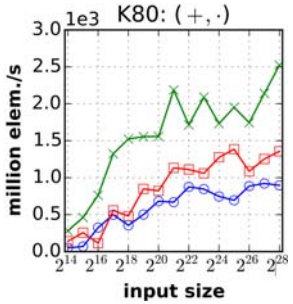
- Introduction
- Motivation

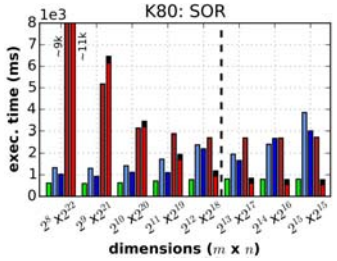


- Our Method
  - Compensation-based Method
  - GPU Implementation
  - Hybrid Parallel Strategy



- Evaluation
  - Weighted-scan Kernel Performance
  - Wavefront Kernel Performance





Experiment Platforms

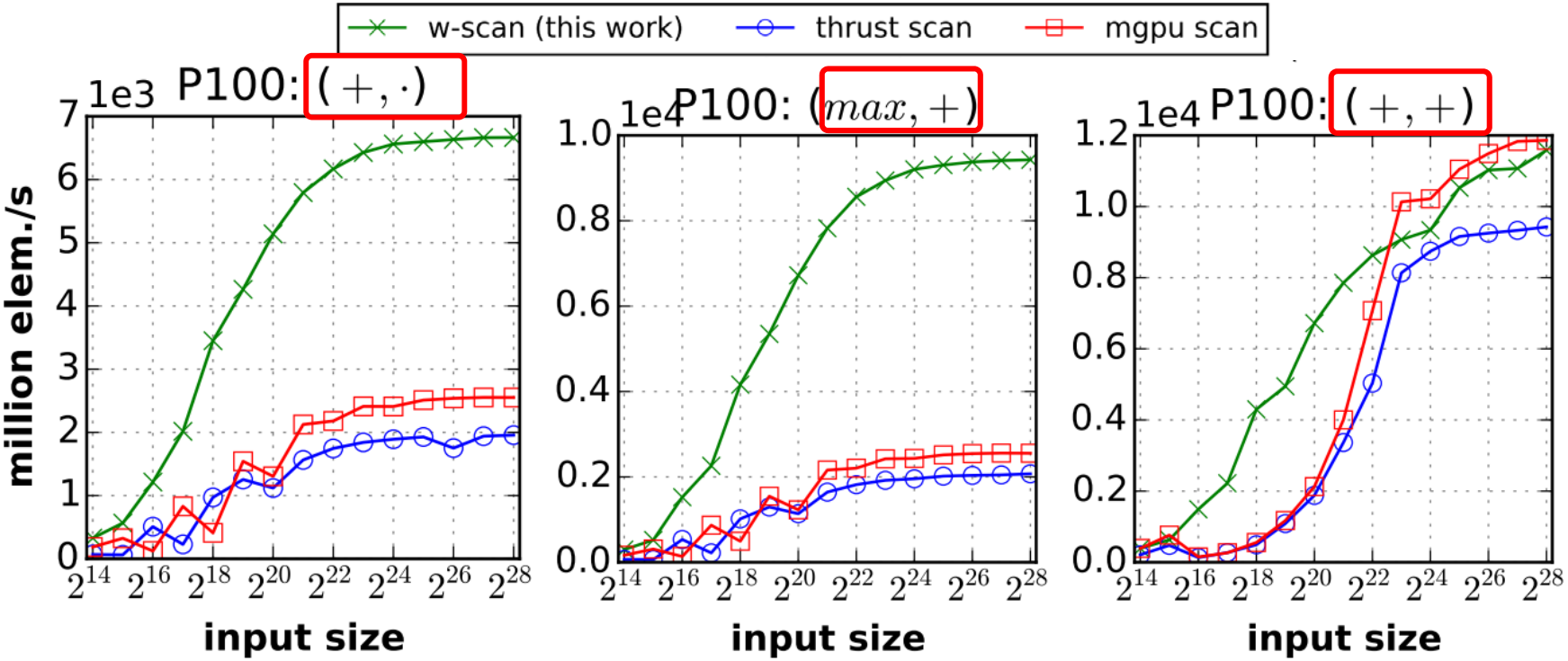
- **nVidia Tesla K80 (Kepler-K80)**, 2496 CUDA cores @ 824 MHz, 240 GB/s bandwidth
- **nVidia Pascal P100 (Pascal-P100)**, 3584 CUDA cores @ 405 MHz, 720 GB/s bandwidth *
- We compare our **Weighted Scan** to other tools of
  a. Thrust v.1.8.1 (thrust::exclusive_scan w/ custom comparator)
  b. ModernGPU v.2.0 (mgpu::scan w/ custom comparator)
  – Using 1D array of data to mimic different rows
- We compare our **Hybrid Wavefront** kernel with
  a. Tile-based methods [`15] (incl. square & diamond tiles)
  b. Compensation-based methods [`12, `16, this work]
  – Using 2D array of data to mimic different workloads

* We only show the performance results of P100 GPU in the presentation.

Weighted-scan Kernel Performance

- Processing a row of data with variable sizes

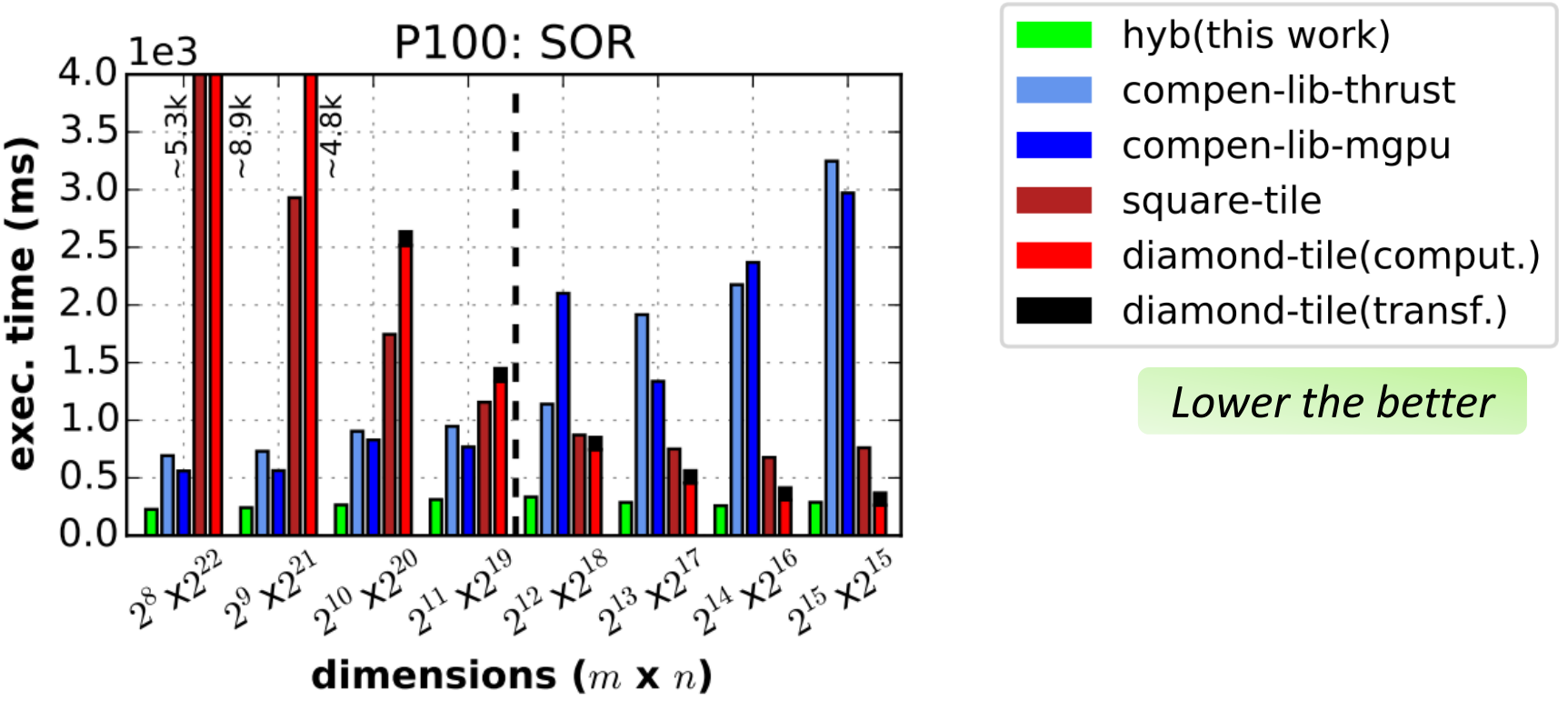
Higher the better



- For ○≠◇, the significant performance benefits of our method can be obtained (*mainly because we can calculate the distance-related weights more efficiently in the kernel*)
- For ○=◇, our method reduces to an ordinary scan kernel

Wavefront Kernel Performance

- Using SOR, SW, and SAT as representative wavefront kernels
- Processing 2D matrices of data with variable dimensions

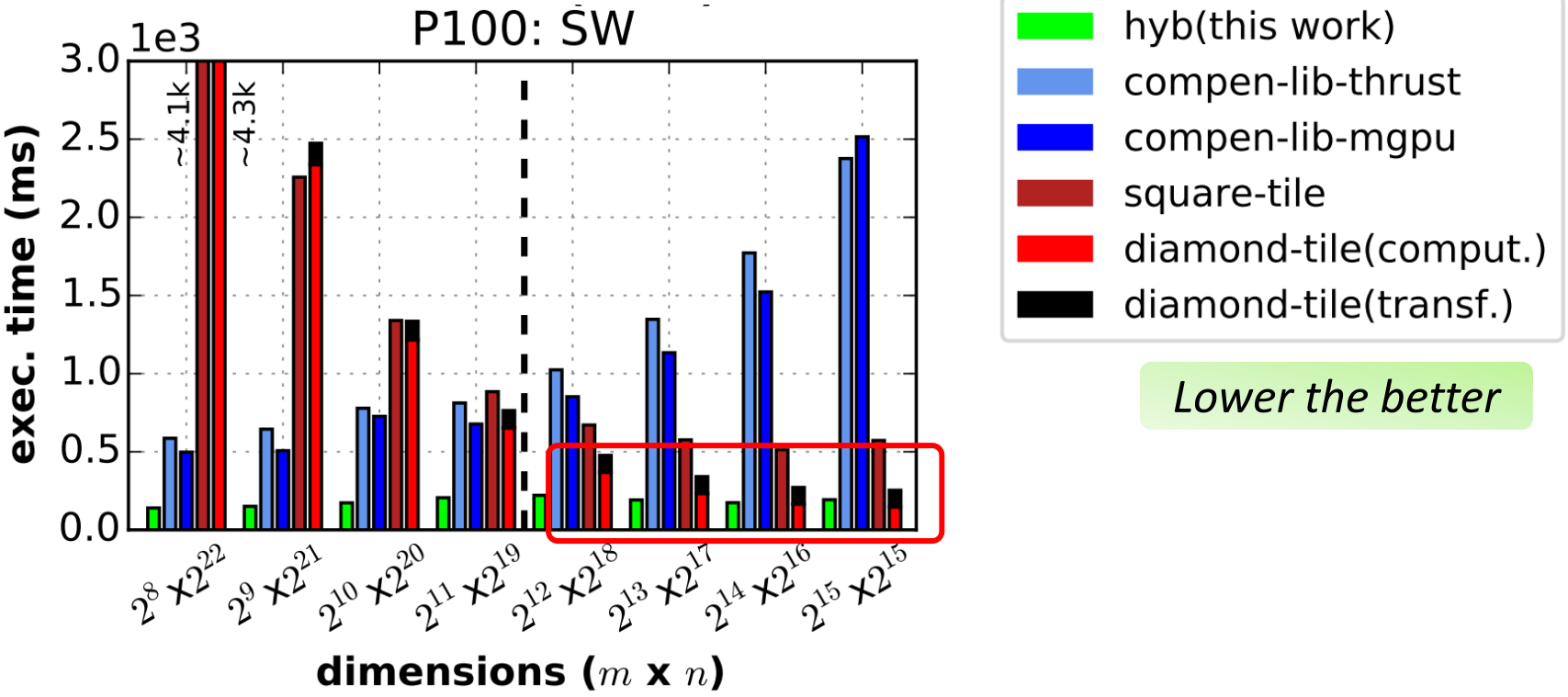


Lower the better

  - Our method (green) can always achieve better performance than previous solutions

Wavefront Kernel Performance

- Using SOR, SW, and SAT as representative wavefront kernels
- Processing 2D matrices of data with variable dimensions

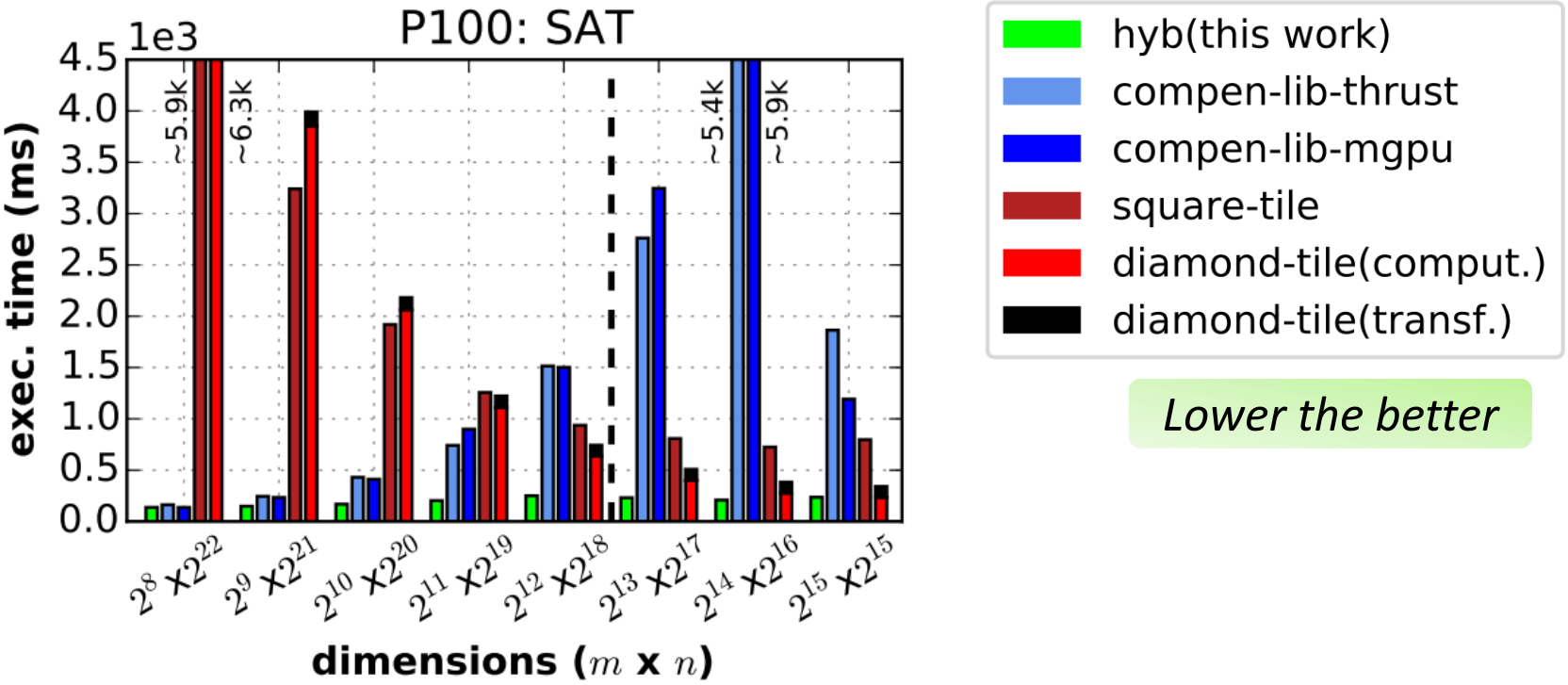


Lower the better

  - The transformation overhead becomes non-negligible for the diamond-tile method

Wavefront Kernel Performance

- Using SOR, SW, and SAT as representative wavefront kernels
- Processing 2D matrices of data with variable dimensions



Lower the better

  - Still, our hybrid method exhibits superior performance regardless the workloads and wavefront types

Performance Discussion

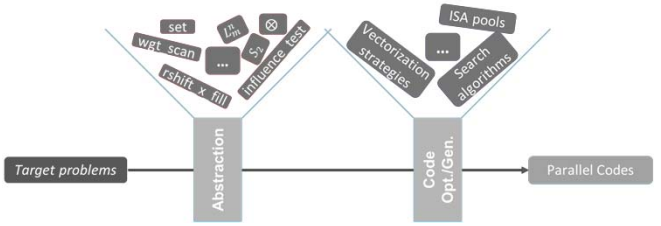
- Tile size selection
  - For some workloads, we need to use tiles and we vary tile dimensions to select the optimal one for our experiments
- Precision
  - For integers, the compensation-based method can obtain exactly same results with the original methods
  - For floats, we observe ~10^{-6} relative errors; for doubles, they are ~10^{-8}
- Generality
  - Applications in a more general data dependency (e.g., FSM) could benefit from our methods, if they fulfil the operator requirements

Conclusion

- Prove the generality and validity of the compensation-based parallelism for wavefront loops on GPUs under which operator properties
- Propose a highly efficient hybrid design of the compensation-based parallelism on GPUs
- Experiments demonstrate that our work can achieve significant performance improvements for different wavefront problems on various inputs

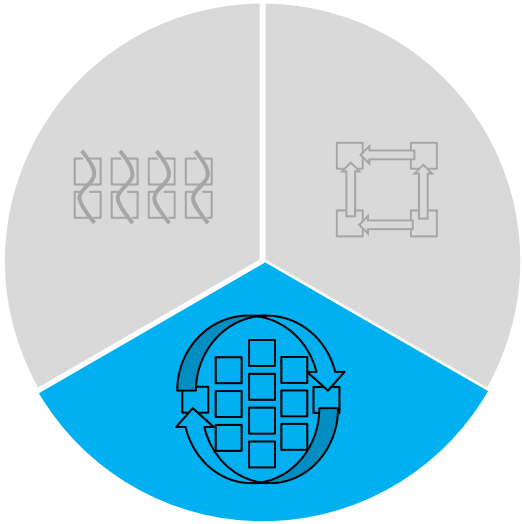
Outline of the Talk

- Motivation
- Contribution & Papers



- Previous Work

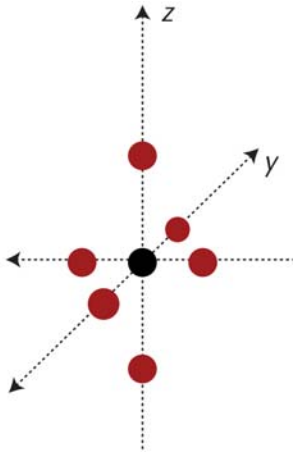
- **Our Methods**
  - *Data-reordering (covered in Prelim)*
  - *SIMD Operations (covered in Prelim)*
  - Data-thread Binding (seg_sort)
  - Data Dependencies (wavefront)
  - **Data Reuse (stencils)**



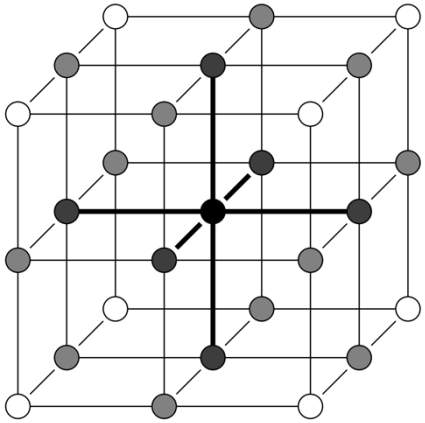
- Summary and Future Work

Stencil Computations

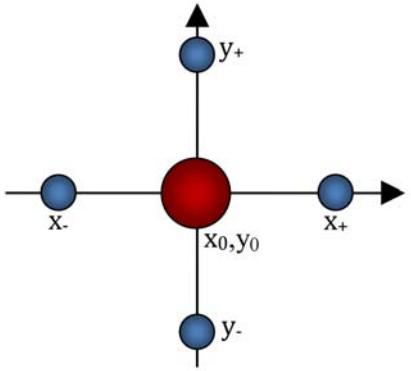
- Nearest neighbor computations
  - Update every grid cell using its surrounding neighbors
  - Sweep over a structured grid (**spatial dimension**)
  - Iterate many times (time dimension)



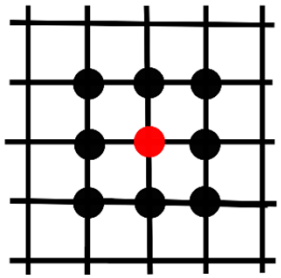
3D Stencil (7 Points)[1]



3D Stencil (27 Points)[2]



2D Stencil (5 Points)[3]



2D Stencil (9 Points)[4]

[1] X. Cai, *et al.* "Accelerating a 3D Finite-Difference Earthquake Simulation with a C-to-CUDA Translator", IEEE CS&E (2012).

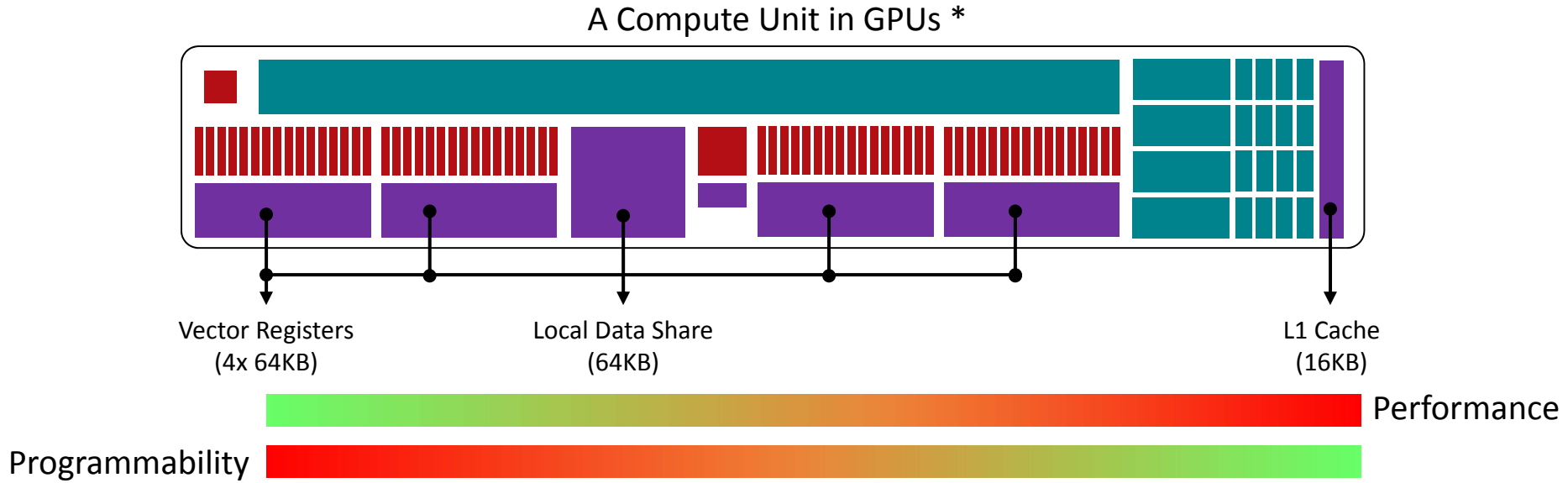
[2] F. Mueller, *et al.* "Autogeneration and Autotuning of 3D Stencil Codes on Homogeneous and Heterogeneous GPU Clusters", IEEE TPDS (2013).

[3] M. J. Gourlay, "Fluid Simulation for Video Games (part 6)". Intel online articles (2012).

[4] Compact stencil, https://en.wikipedia.org/wiki/Compact_stencil

Spatial Blocking for Stencil Computations

- High memory traffic + low arithmetic intensity
  - Memory bound computation
- Blocking optimizations are critical to achieve optimal performance
  - Different blocking strategies, e.g., 3D-blocking and 2.5D-blocking
  - Different GPU cache levels for the reusable data

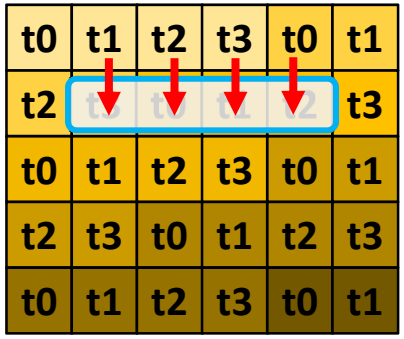


* This figure serves as a logical diagram rather than a physical diagram

GPU Register Data Exchange

- Using registers as cache in stencils is non-trivial
  - What are the communication patterns?

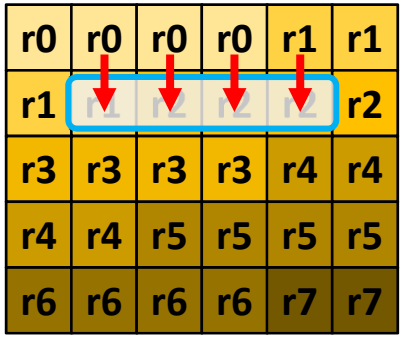
Data distribution (thread view)



To access north (N) neighbors, first figure out ***the correct destination threads***

  - Which registers are of interest for exchange?

Data distribution (register view)

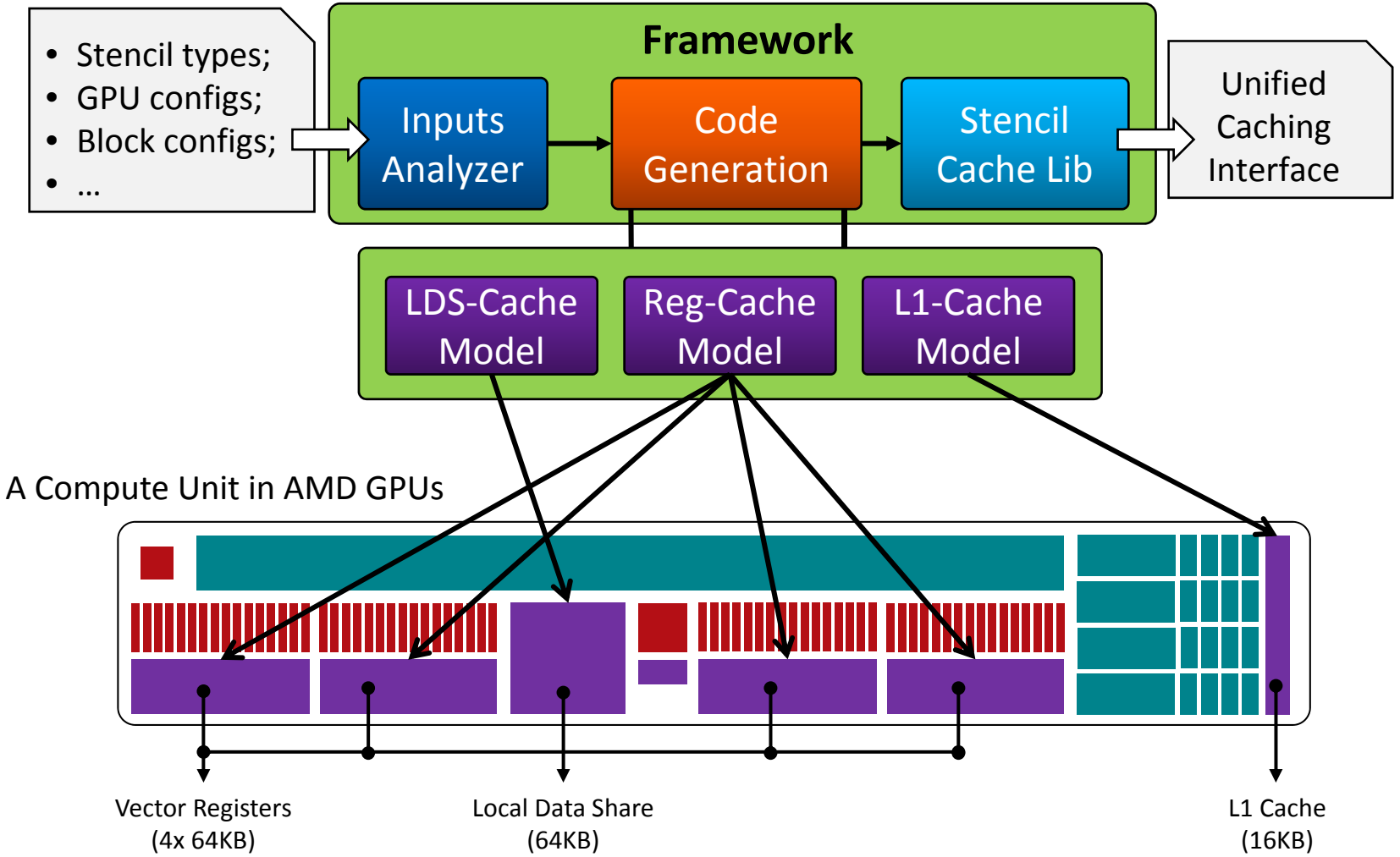


Then, figure out ***the correct registers*** in the correct destination threads

  - The answers are determined by specific stencils, execution unit sizes and dimensionalities (platforms).

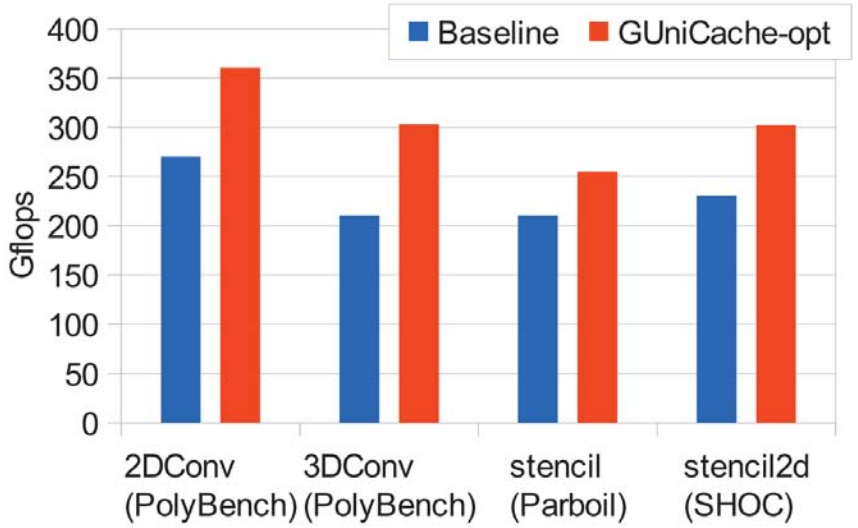
GPU-UNICACHE Framework

- Overview of the structure



Comparison with Open-Source Benchmarks

- **AMD Fiji XT (Radeon-R9)**, 4096 cores @ 1000 MHz, 512GB/s bandwidth, L1/LDS/Rg 16/64/256KB
- **nVidia Maxwell (GTX-980)**, 2048 CUDA cores @ 1126 MHz, 224 GB/s bandwidth, L1/LDS/Rg 16/96/64KB *
- Choose our best performant GPU-UNICACHE codes (in most cases, using registers as cache)



Compare to the third-party stencils with spatial blocking optimizations

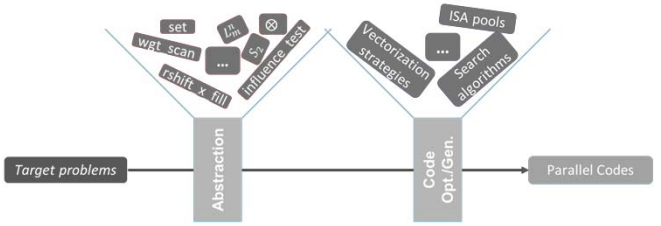
  - By simply using the GPU-UNICACHE functions, we can outperform the existing benchmarks by up to 1.5x

Conclusion

- Propose a framework to automatically generate cell-based library codes to use different cache levels for stencils computations.
  - Support different stencils
  - Support different blocking strategies
  - Support different platforms
- Performance:
  - Evaluate relationships between different stencils and cache levels
  - Up to 1.5x performance benefit over existing stencil benchmarks

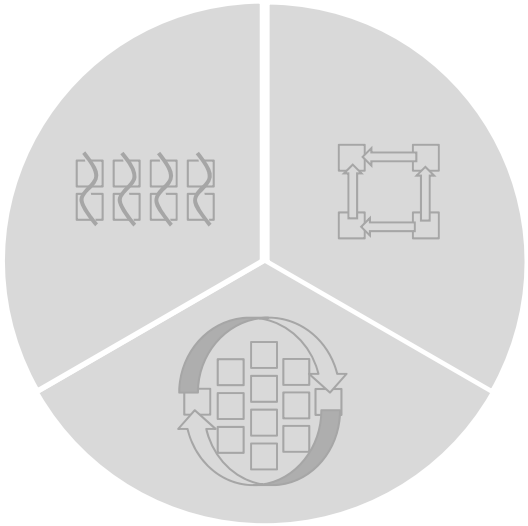
Outline of the Talk

- Motivation
- Contribution & Papers



- Previous Work

- Our Methods
  - *Data-reordering (covered in Prelim)*
  - *SIMD Operations (covered in Prelim)*
  - Data-thread Binding (seg_sort)
  - Data Dependencies (wavefront)
  - Data Reuse (stencils)



- **Summary**

Summary

Data-Thread Binding in SegSort

- Explored how the data items are stored in the registers for each thread to enable efficient processing on GPUs

Data Dependencies in Wavefront

- Proposed a new parallel pattern to loosen the data dependencies and then compensate the loss caused by that

Data Reuse in Stencils

- Focused on the data reuse in difference levels of memory hierarchies on GPUs

Ph. D. Timeline