Highly Efficient Compensation-based Parallelism for Wavefront Loops on GPUs

Kaixi Hou, Hao Wang, Wu-chun Feng
{kaixihou, hwang121, wfeng}@vt.edu

Jeffrey S. Vetter, Seyong Lee
vetter@computer.org, lees2@ornl.gov

Virginia Tech
VIRGINIA POLYTECHNIC INSTITUTE AND STATE UNIVERSITY

OAK RIDGE National Laboratory

IEEE IPDPS 2018, Vancouver, British Columbia, Canada

Wavefront Loops

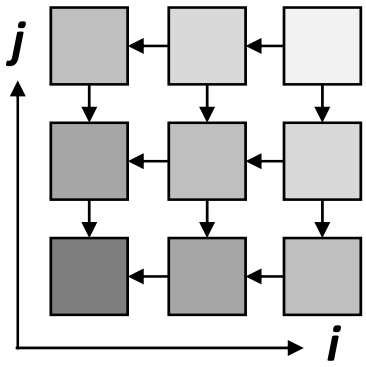
- Update each entry of a workspace grid based on the already-updated values from its neighbors
- Used in many scientific applications, e.g., PDE solver, sequence alignment tools, etc.

Example: a wavefront loop (2D matrix)

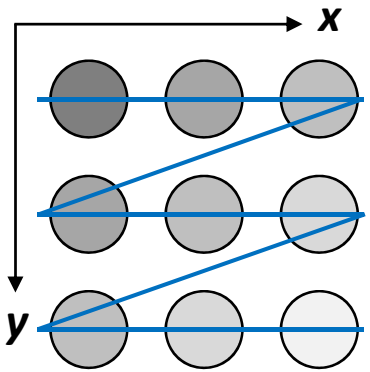
```
for(int i = 0; i < m; i++)
  for(int j = 0; j < n; j++)
    A[i][j] = A[i][j-1] * 0.5 + A[i-1][j] * 0.5;
```

Neither loop can be parallelized.

Data Dependence (Iteration Space)



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



Wavefront Loops

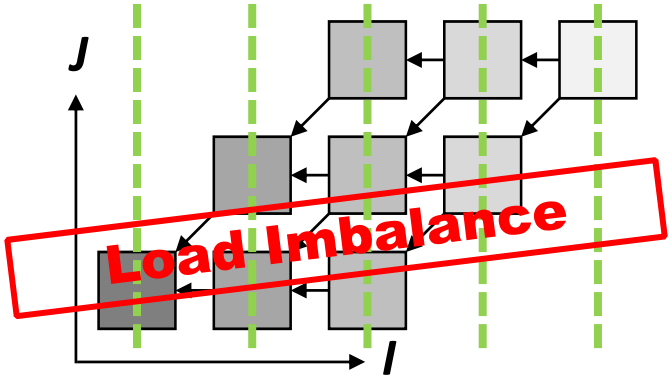
- Update each entry of a workspace grid based on the already-updated values from its neighbors
- Used in many scientific applications, e.g., PDE solver, sequence alignment tools, etc.

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

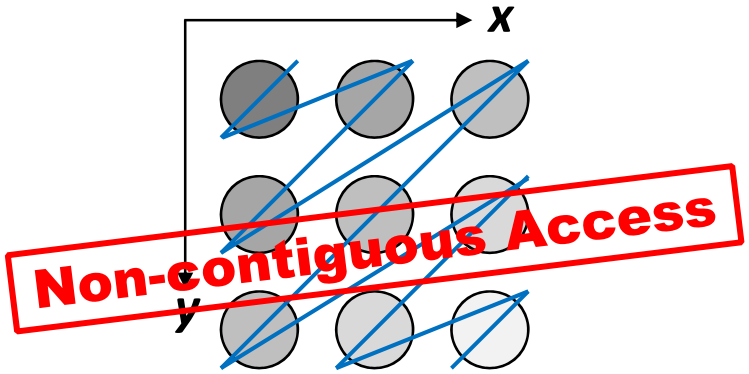
J-loop can be parallelized.

```
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)

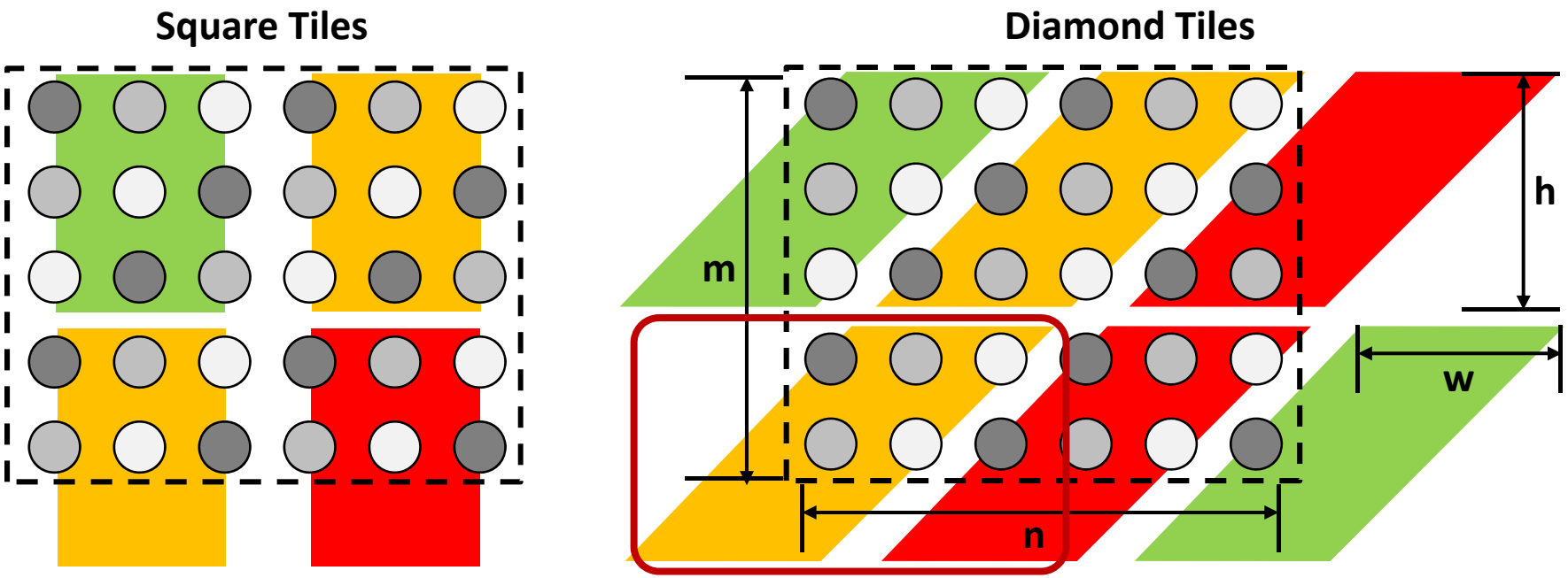


Memory Access (Memory Space A[y][x]**)**



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

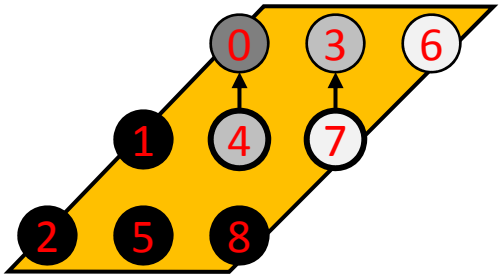
Non-contiguous memory access still exists

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

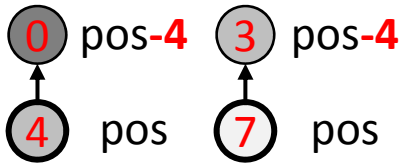
Existing Parallel Solutions

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

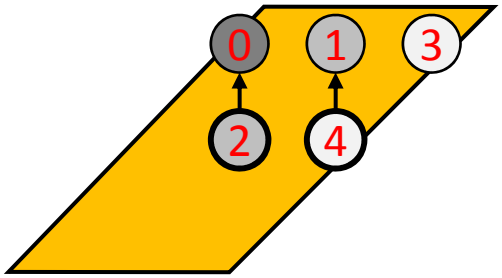
Padding



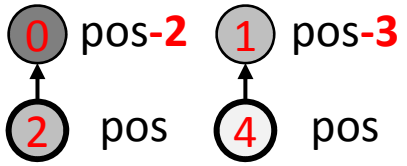
Uniform access pattern



No padding



Diverged access pattern



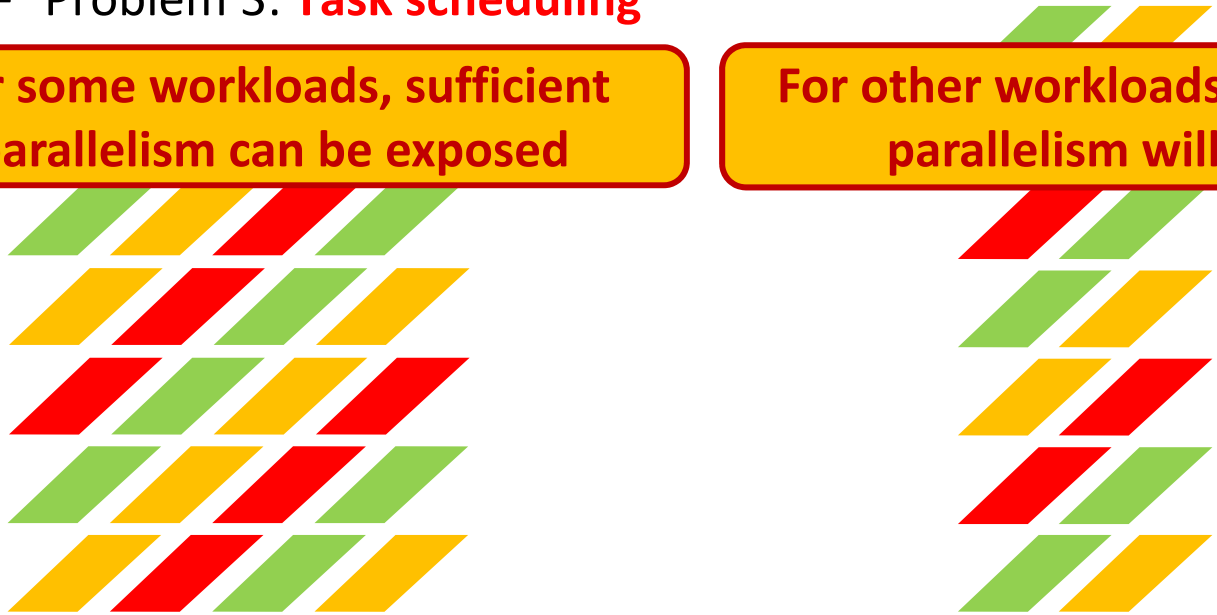
Padding-free strategy may greatly increase the complexity of indexing and lead to more branches in GPU kernels

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

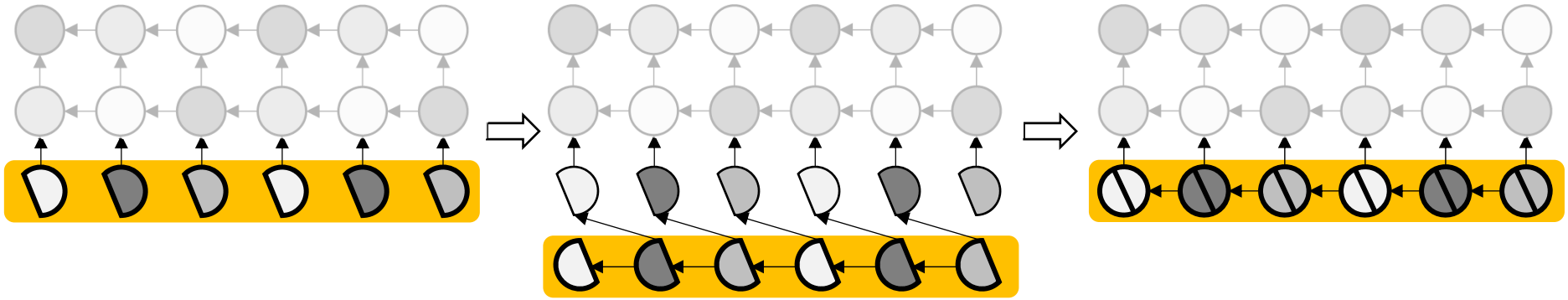
Existing Parallel Solutions

- 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

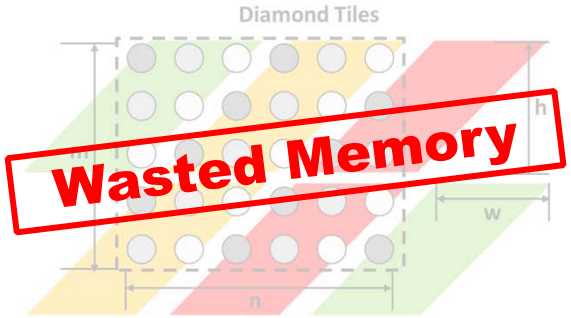
❷ Compensate the partial results

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

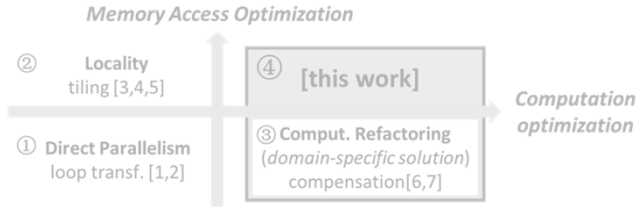
Highly Efficient Wavefront Parallelism (this work)



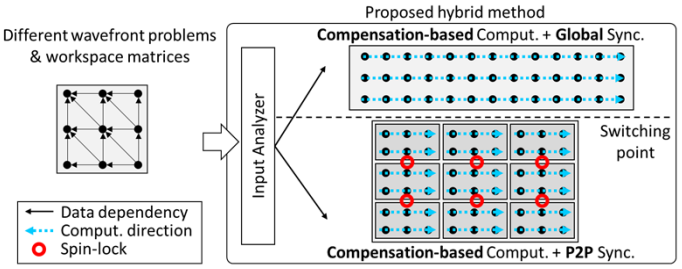
(1) Can the compensation-based method be used to optimize general wavefront loops?
(2) Is the compensation-based method sufficient for any types of workloads?

Outline

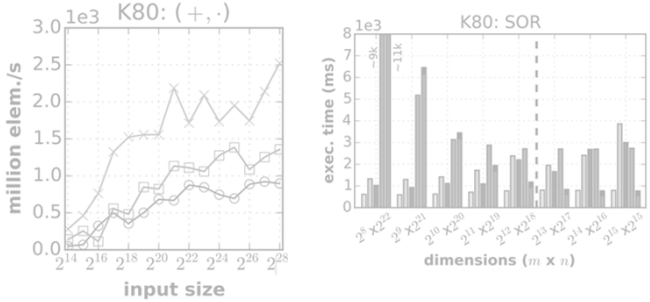
- Introduction
- Motivation



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



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

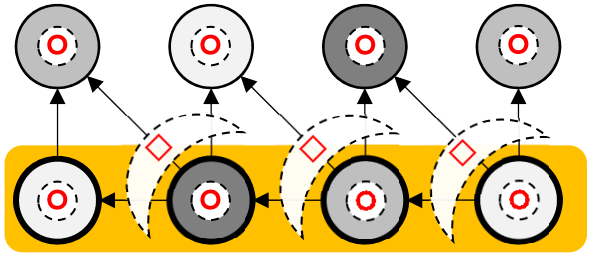


Compensation-based Method

- Wavefront Patten

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

- $\circ$ *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 $\diamond$. *

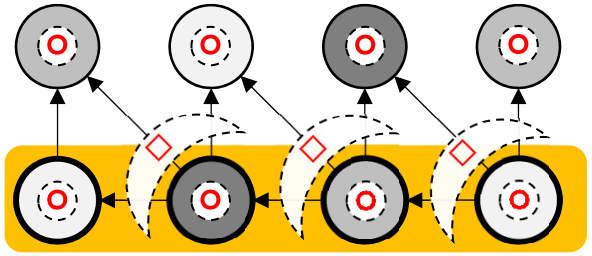
* The mathematic proof is included in our paper.

Compensation-based Method

- Wavefront Patten

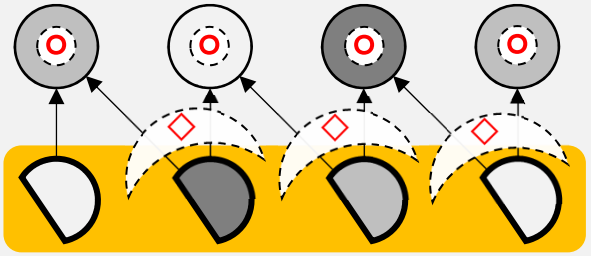
$$A_{i,j} = (A_{i,j-1} \circ b_0) \diamond (A_{i-1,j} \circ b_1) \diamond (A_{i-1,j-1} \circ b_2)$$

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

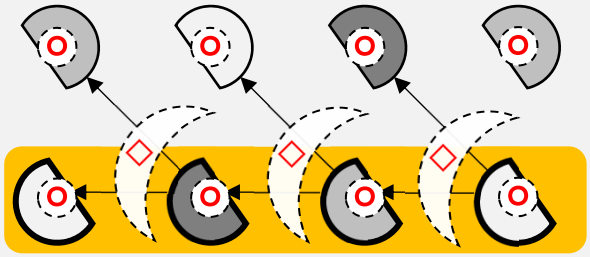


- Compensation-based Method

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, +)

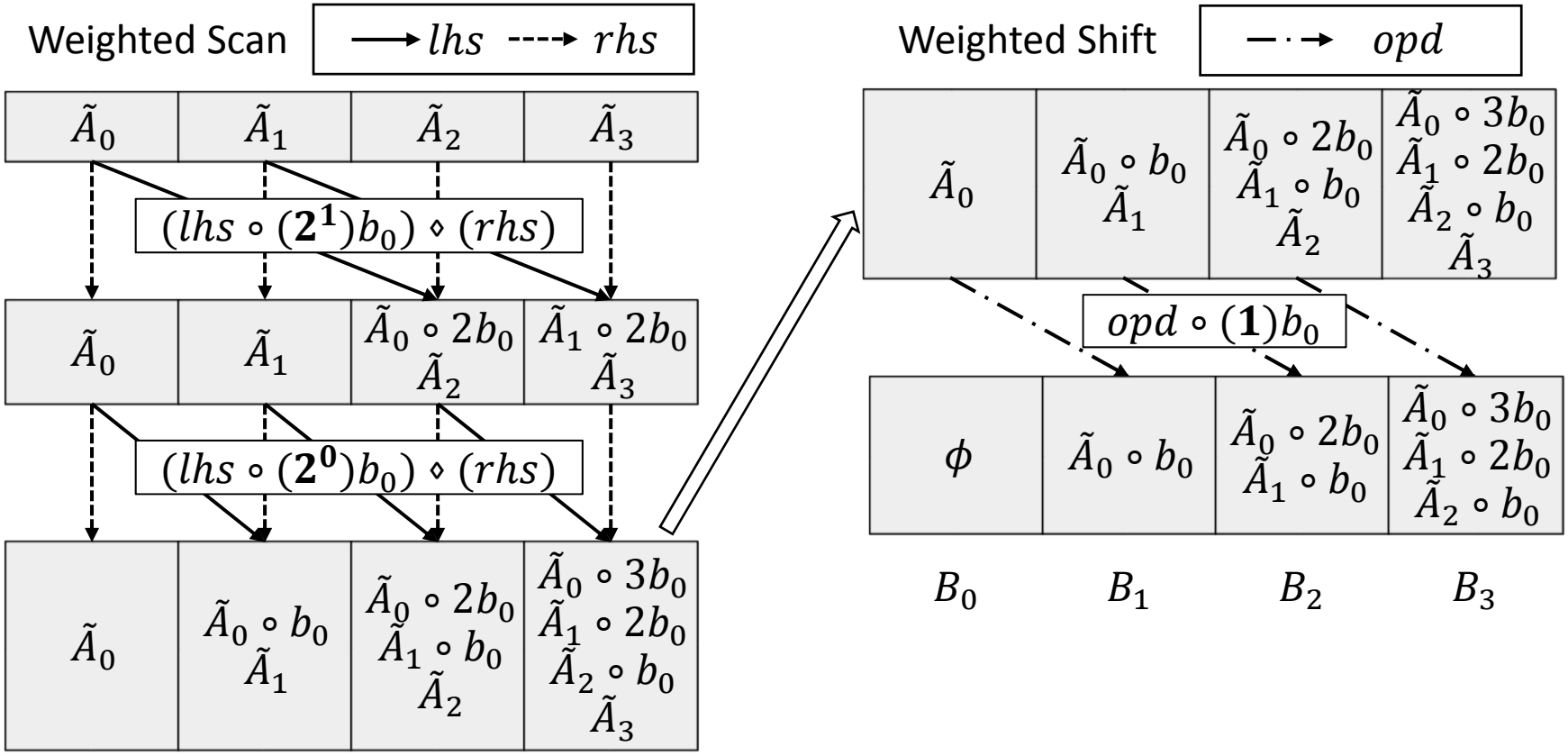
```
A[i][j] = max(A[i][j-1] - 2, A[i-1][j] - 2,
              A[i-1][j-1] + s(i,j), 0);
```

- 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

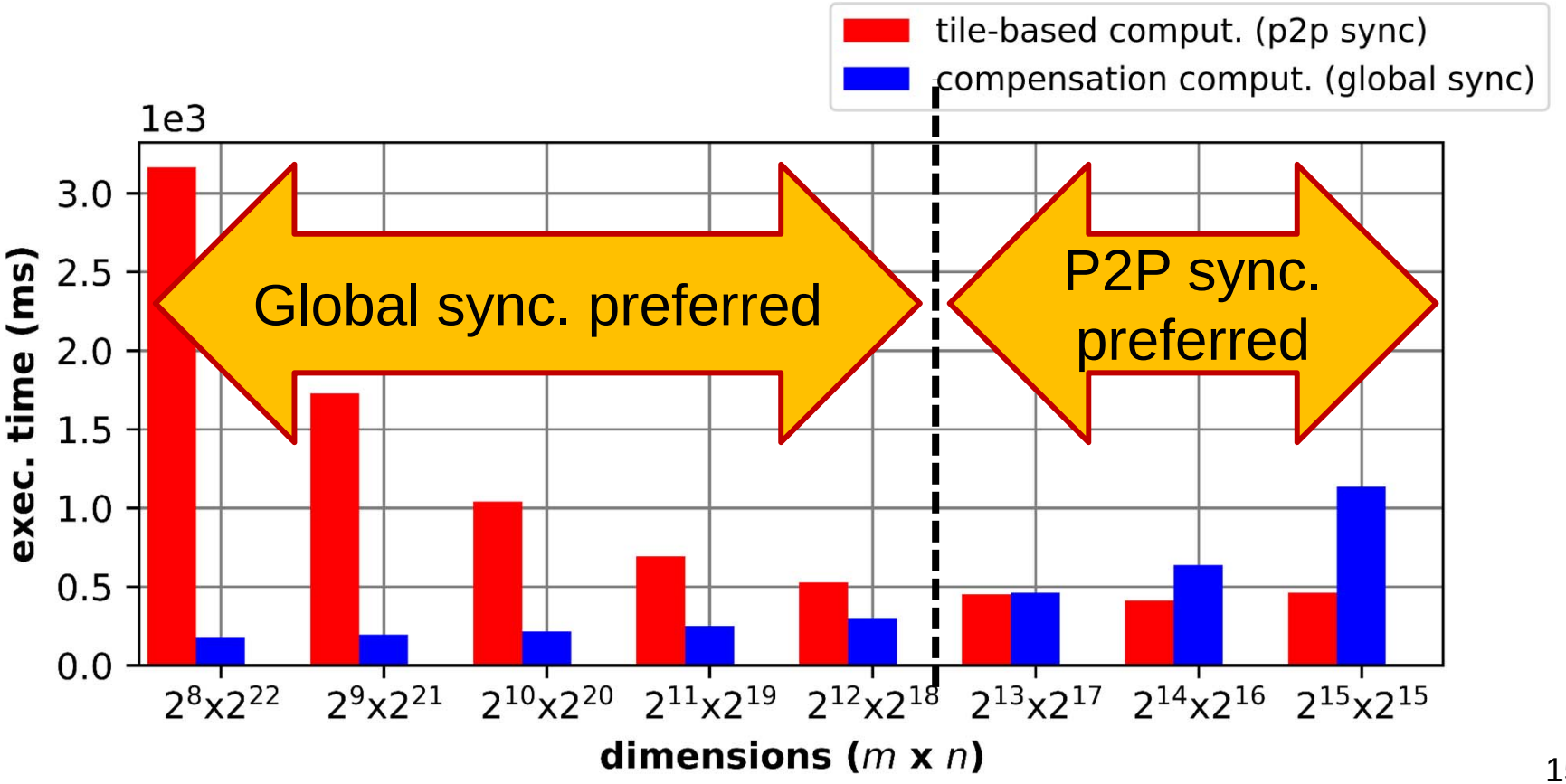
GPU Implementation

- 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

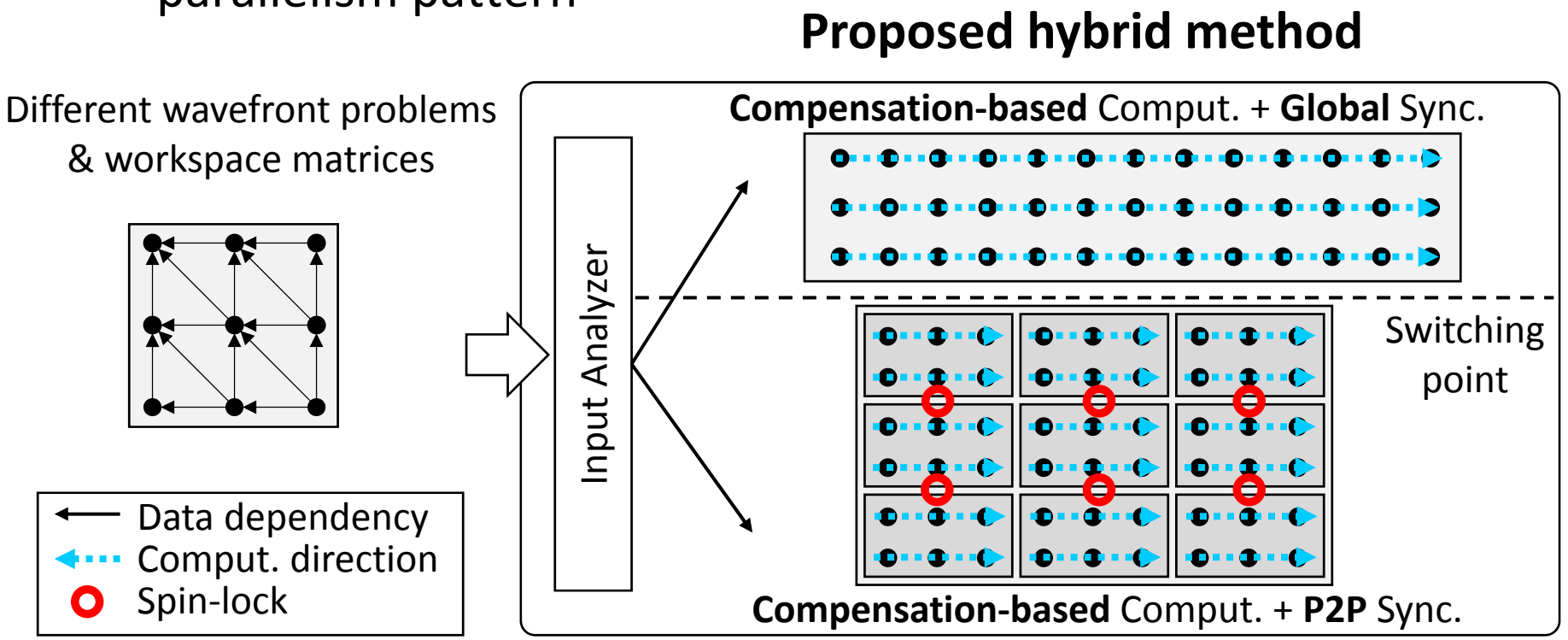
Hybrid Parallel Strategy

- *Is the compensation-based method sufficient for any types of workloads?*
- Observations (using the wavefront shown in the 1st slide)



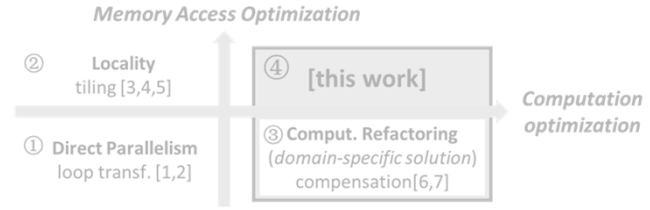
Hybrid Parallel Strategy

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

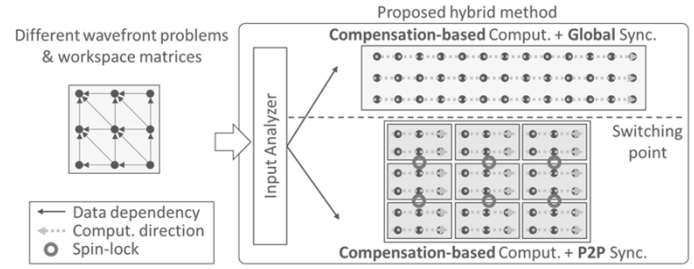


Outline

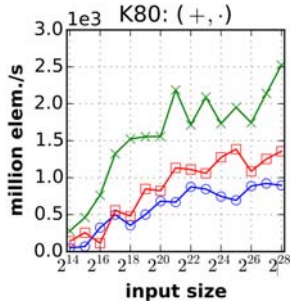
- Introduction
- Motivation

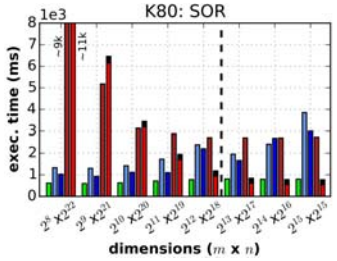


- Our Method
  - Compensation-based Method
  - GPU Implementation
  - Hybrid Parallel Strategy



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





Experiment Platforms

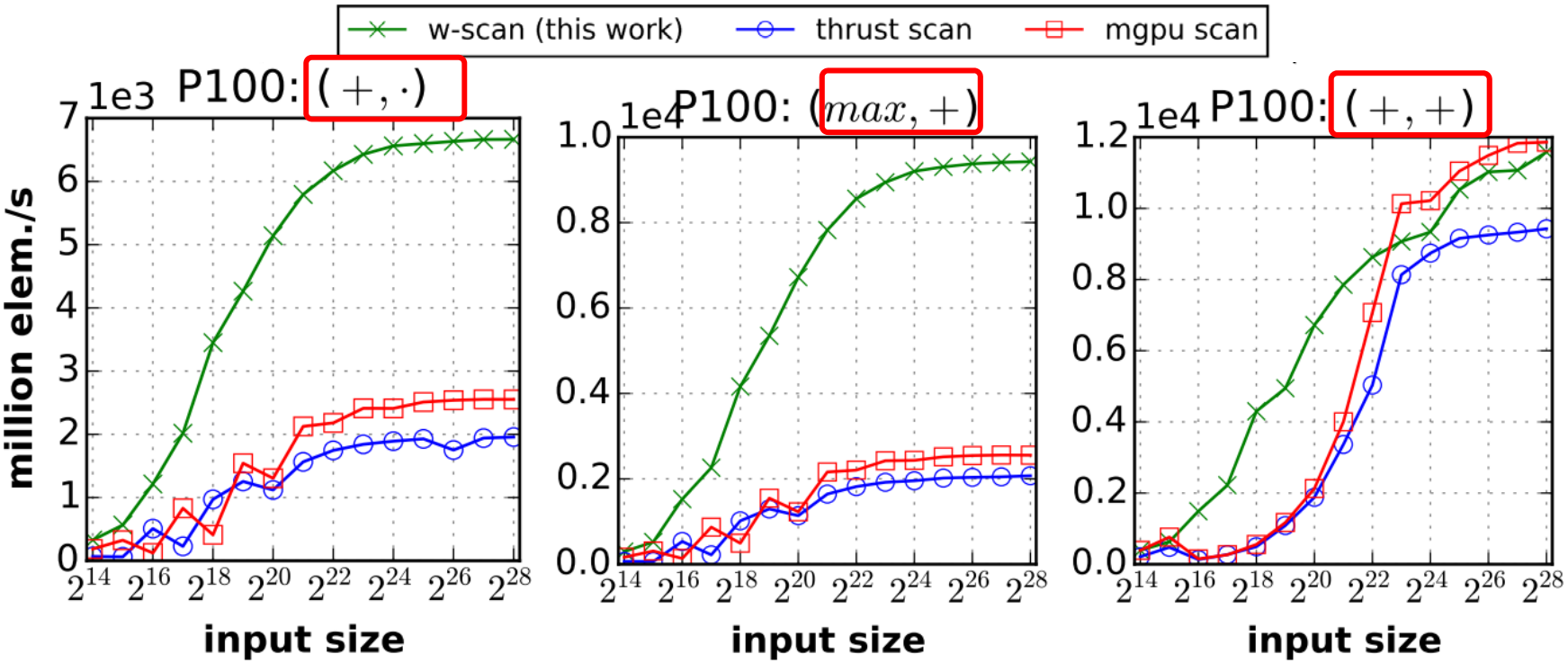
- **nVidia Tesla K80 (Kepler-K80)**, 2496 CUDA cores @ 824 MHz, 240 GB/s bandwidth
- **nVidia Pascal P100 (Pascal-P100)**, 3584 CUDA cores @ 405 MHz, 720 GB/s bandwidth *
- We compare our **Weighted Scan** to other tools of
  a. Thrust v.1.8.1 (thrust::exclusive_scan w/ custom comparator)
  b. ModernGPU 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

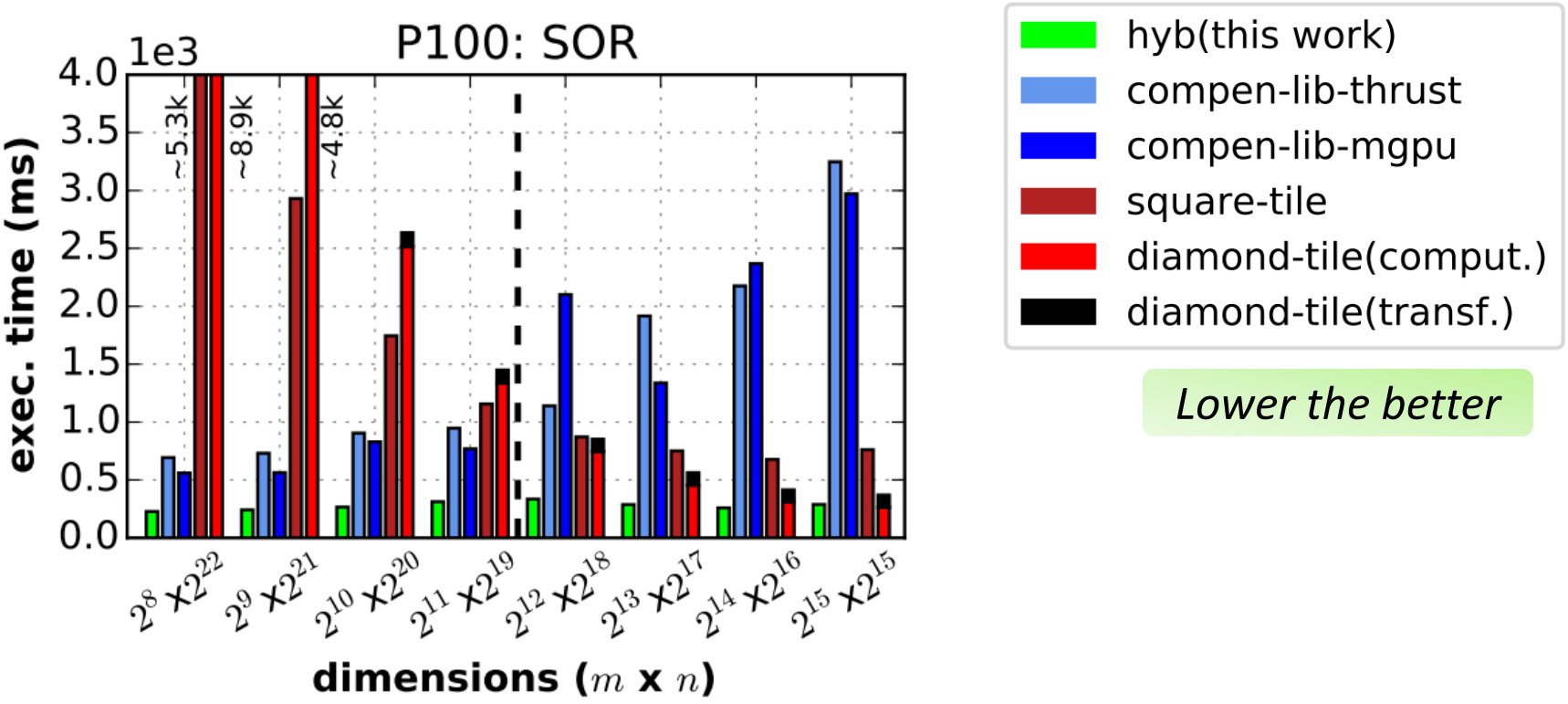
Higher the better



- For $\circ \neq \diamond$, 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 $\circ = \diamond$, 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

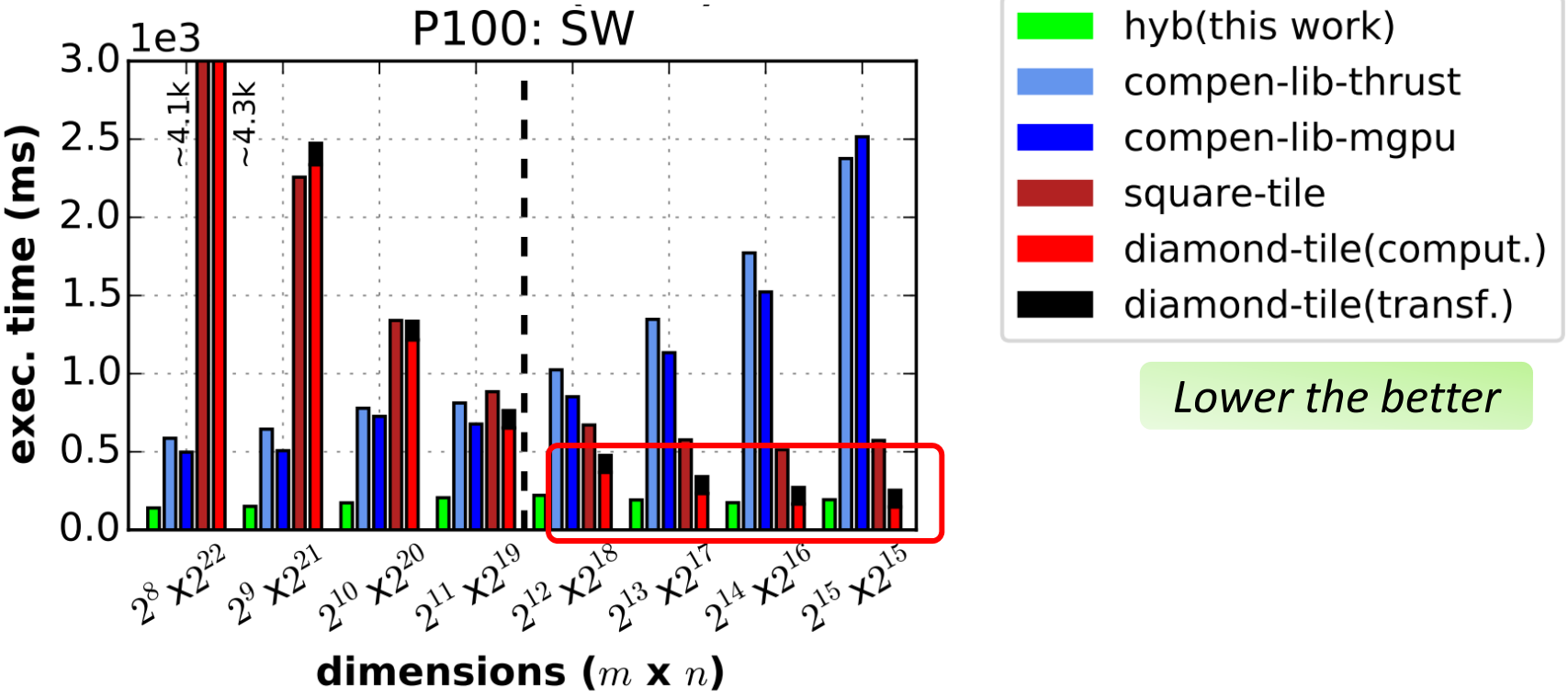


Lower the better

  - Our method (green bars) 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

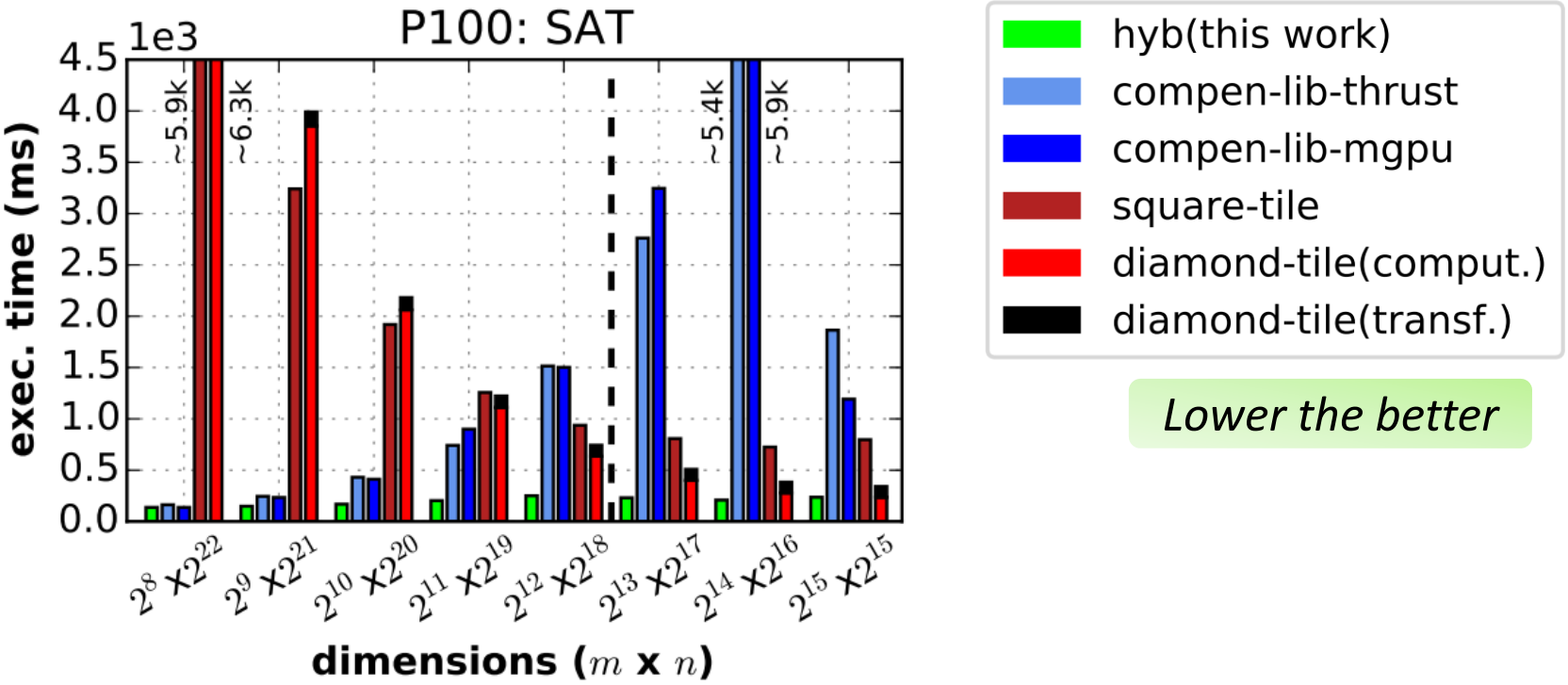


Lower the better

  - 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

- We target the compensation-based parallelism for wavefront loops on GPUs
- We prove the validity of the compensation-based method when the accumulation operator is associative and commutative
- We also 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

Thank you!

Email to **kaixihou@vt.edu**
More from **http://people.cs.vt.edu/~kaixihou/**