Streamlining GPU Applications On the Fly

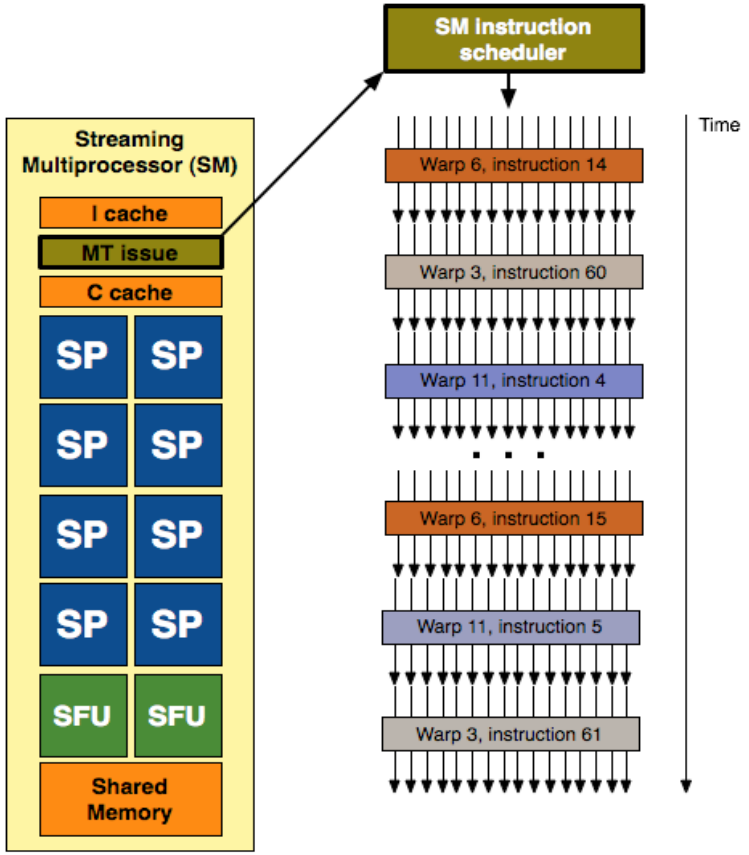
Thread Divergence Elimination through Runtime Thread-Data Remapping

Eddy Z. Zhang, Yunlian Jiang, Ziyu Guo, Xipeng Shen
Department of Computer Science, College of William and Mary

eddy@cs.wm.edu

GPU Divergence

- GPU Features
  - Streaming multiprocessors
    - SIMD
    - Single instruction issue per SM
  - Warp / Half Warp
    - SIMD execution unit
- Divergence
  - Threads in a warp take different execution paths

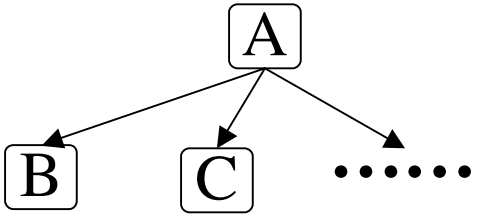


eddy@cs.wm.edu

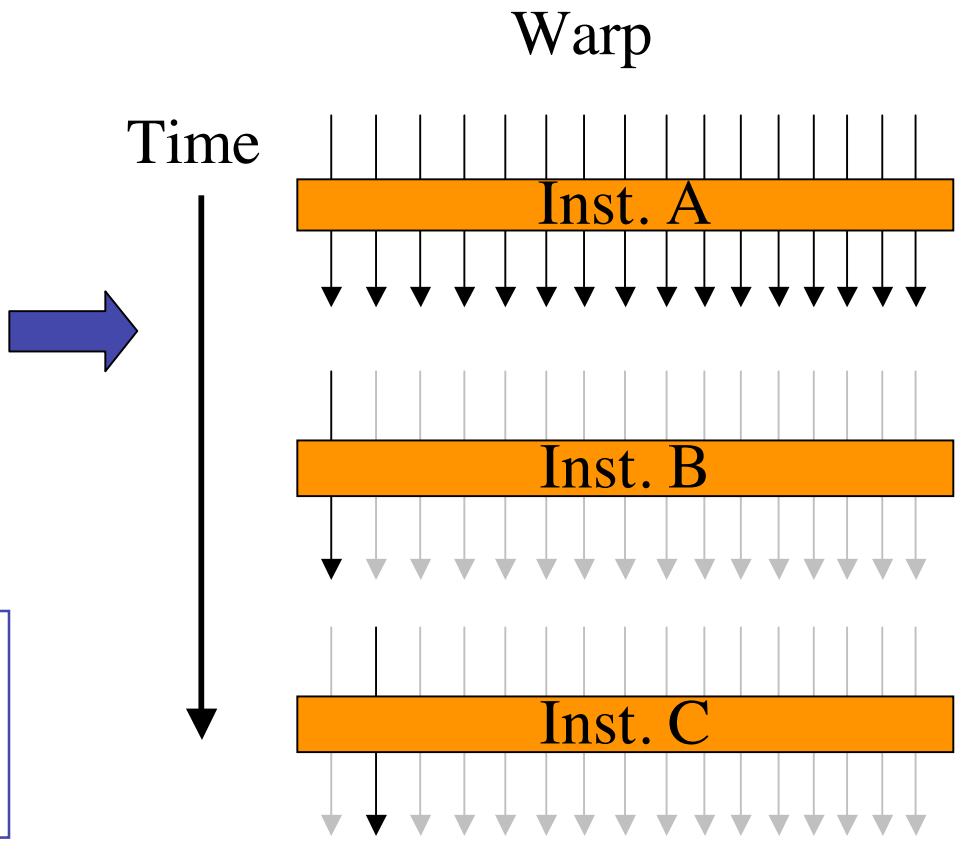
Example of GPU Divergence

Instructions: A, B, C

Control Flow



Thread Serialization Behavior



eddy@cs.wm.edu

Impact of GPU Divergence

- Degrading GPU Throughput
  - E.g., up to $\frac{15}{16}$ degradation on Tesla 1060
- Impairing GPU Usage
  - Esp. when having non-trivial condition statements

eddy@cs.wm.edu

Related Work

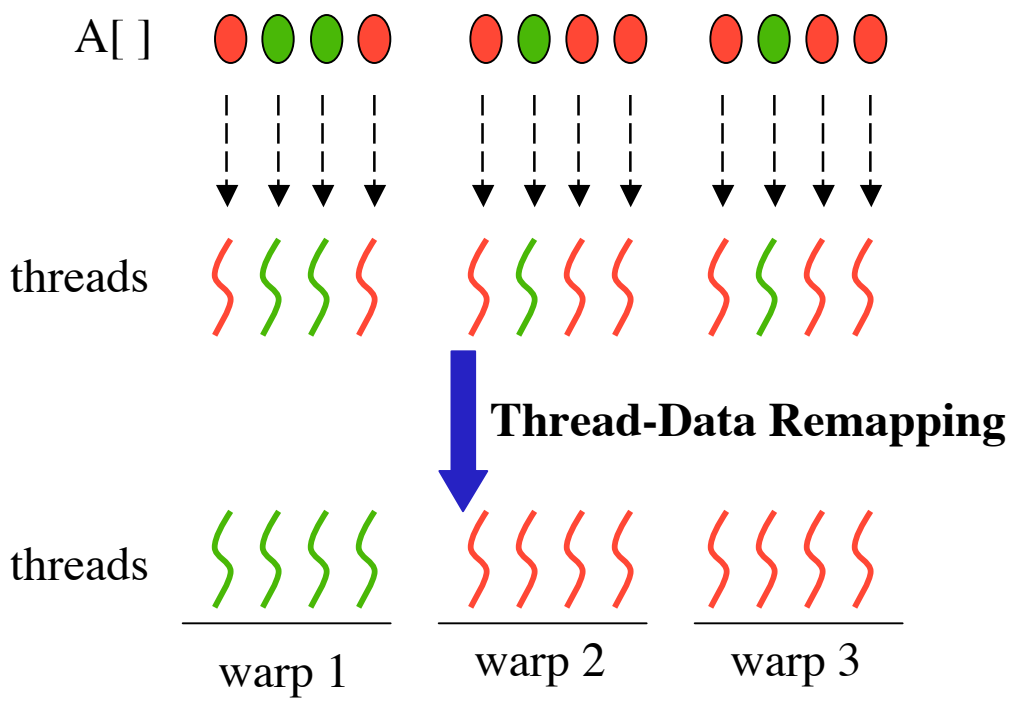
- Stream packing and unpacking
  - [Popa: U. Waterloo C.S. master thesis'04]
  - Simulate hardware packing on CPU
- Dynamic warp formation & scheduling
  - [Fung+: MICRO'07]
  - Hardware solution
- Control structure splitting
  - [Carrillo+: CF'09]
  - Reducing register pressure but removing no divergences

eddy@cs.wm.edu

Basic Idea of Our Solution

Swapping Jobs of Threads through Thread-Data Remapping

```
if ( A[tid] ){ // (green)
    C[tid] += 1;
}
else{ // (red)
   C[tid] -= 1;
}
```



eddy@cs.wm.edu

Challenges

- How to determine a desirable mapping?
  - Complexities
    - irregular accesses, complex indexing expressions, side effects on memory reference patterns, ...
- How to realize the new mapping?
  - Data movement or redirect threads' data references
    - limitations, effectiveness, and safety.
- **How to do it on the fly?**
  - Large overhead v.s. Need for runtime remapping
    - dependence on runtime data values, minimizing and hiding overhead

eddy@cs.wm.edu

Outline

- **Thread-data Remapping**
  - Concept & mechanisms
- Transformation on the Fly
  - CPU-GPU pipelining & LAM
- Evaluation
- Conclusion

eddy@cs.wm.edu

GPU Divergence Causes

- Control Flows in Code
  - E.g., ***if, do, for, while, switch***
- Input Data Dependence
  - Input data-set **-->** execution path
  - Thread-data mapping **-->** amount of thread divergence

eddy@cs.wm.edu

Define Divergence

- Control Flow Path Vector for One Thread

Def: ***Pvector***[*tid*] = < $b_1, b_2, b_3, \ldots, b_n$ >

Condition Statements

```
if ( A[tid] % 2 ) {…};
if ( A[tid] < 10 ) {…};
```

Path Vector Example

tid	A[tid]	Pvector
0	2	<0,1>
1	11	<1,0>
2	14	<0,0>
…	…	…

eddy@cs.wm.edu

Define Divergence

- Control Flow Path Vector for One Thread

Def: ***Pvector***[*tid*] = < $b_1, b_2, b_3, \ldots, b_n$ >

Condition Statements

```
if ( A[tid] % 2 ) {…};
if ( A[tid] < 10 ) {…};
```

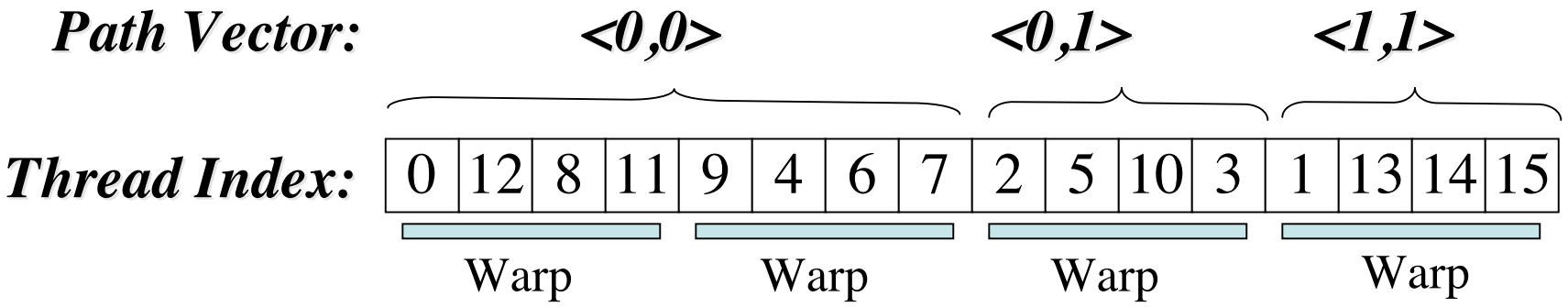
Path Vector Example

tid	A[tid]	Pvector
0	2	<0,1>
1	11	<1,0>
2	14	<0,0>
…	…	…

eddy@cs.wm.edu

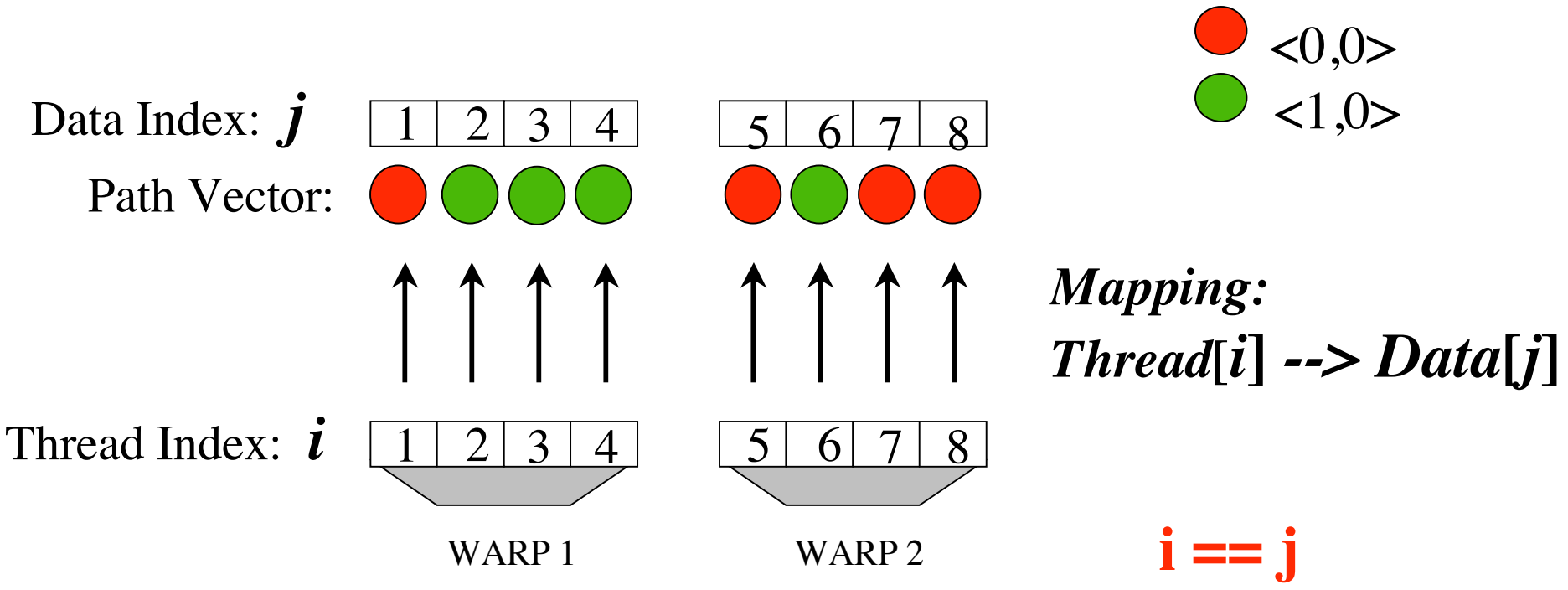
Regroup Threads

- To Satisfy Convergence Condition:
  - Sort ***Pvector***[*0*], ***Pvector***[*1*], ***Pvector***[*2*], … for all threads
  - E.g, after sorting, the grouping of threads



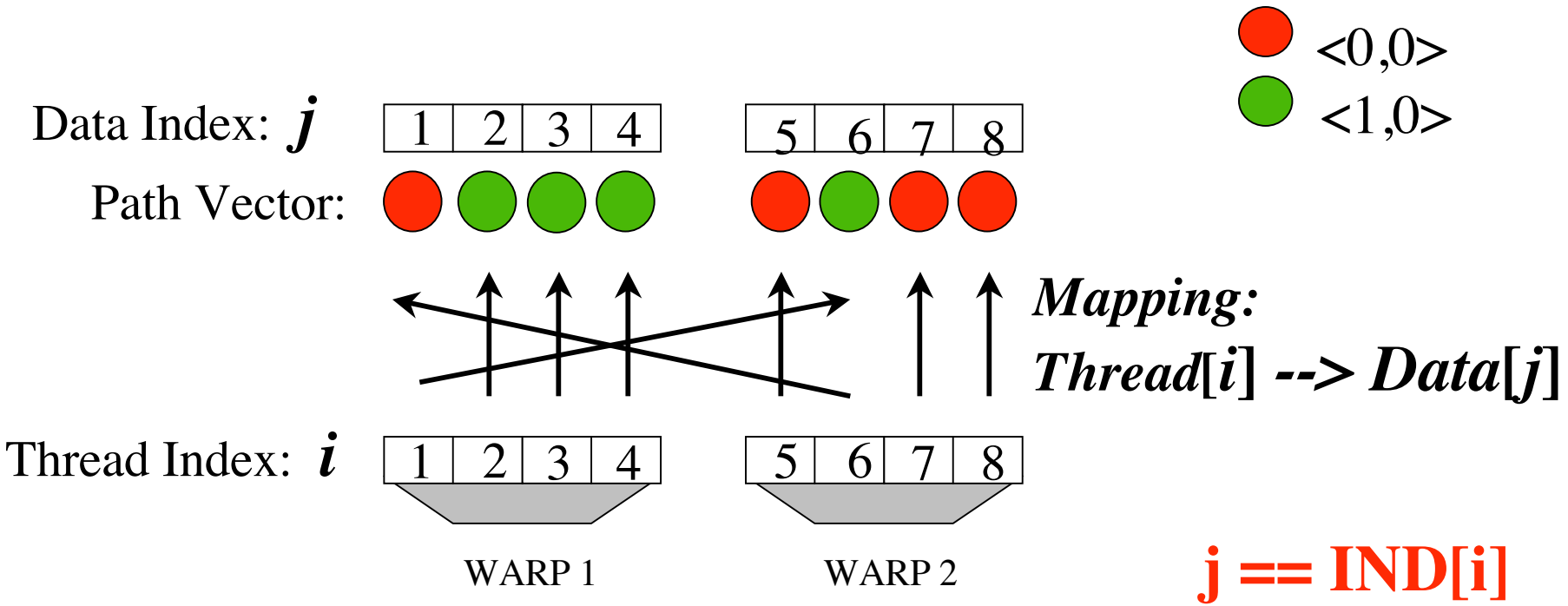
eddy@cs.wm.edu

Example of GPU Thread Divergence



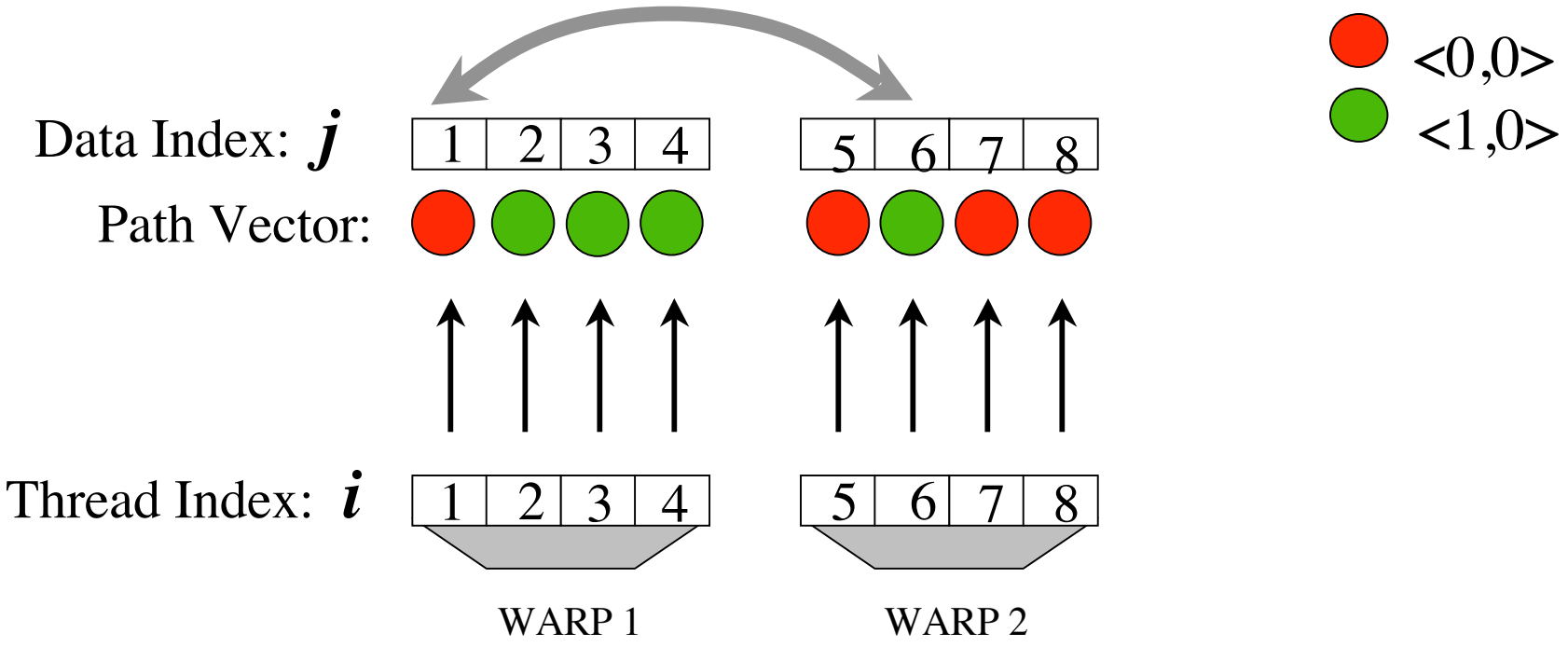
eddy@cs.wm.edu

Remapping by Reference Redirection



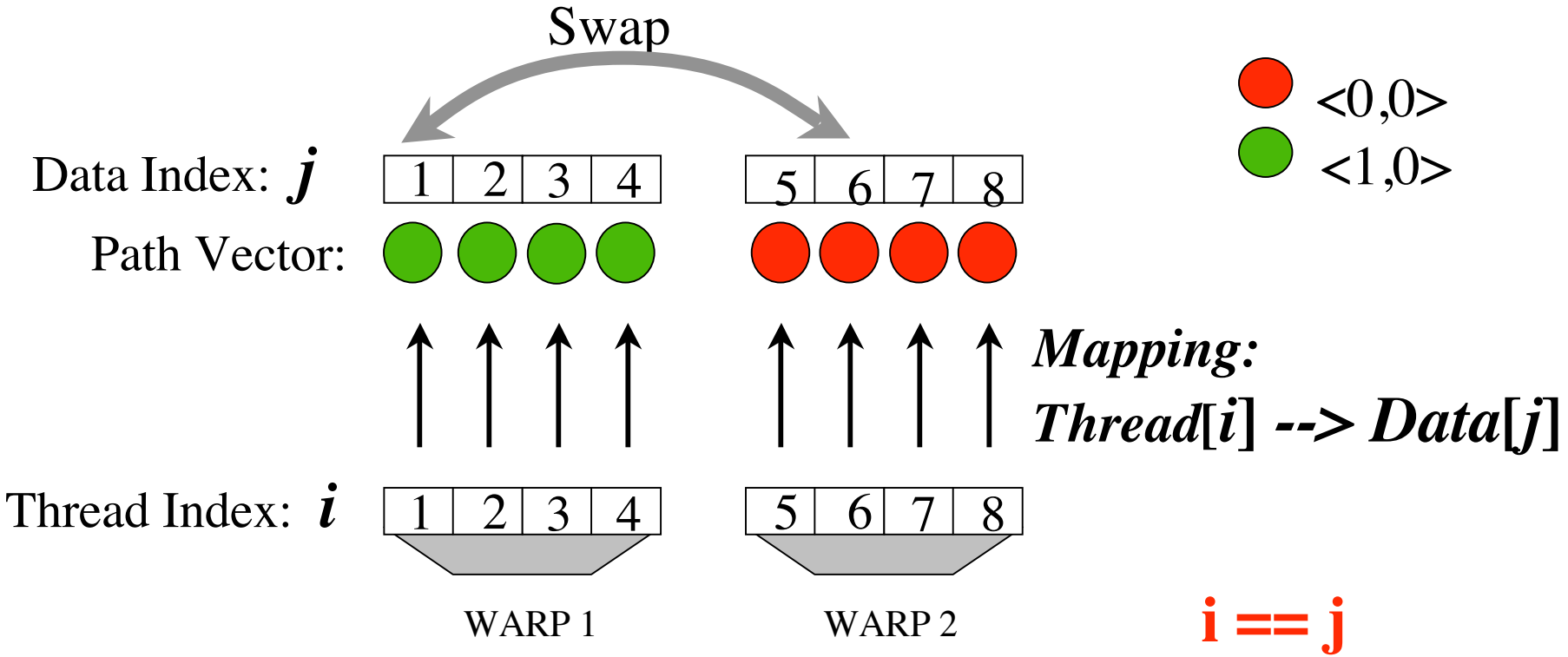
eddy@cs.wm.edu

Remapping by Data Transformation



eddy@cs.wm.edu

Remapping by Data Transformation



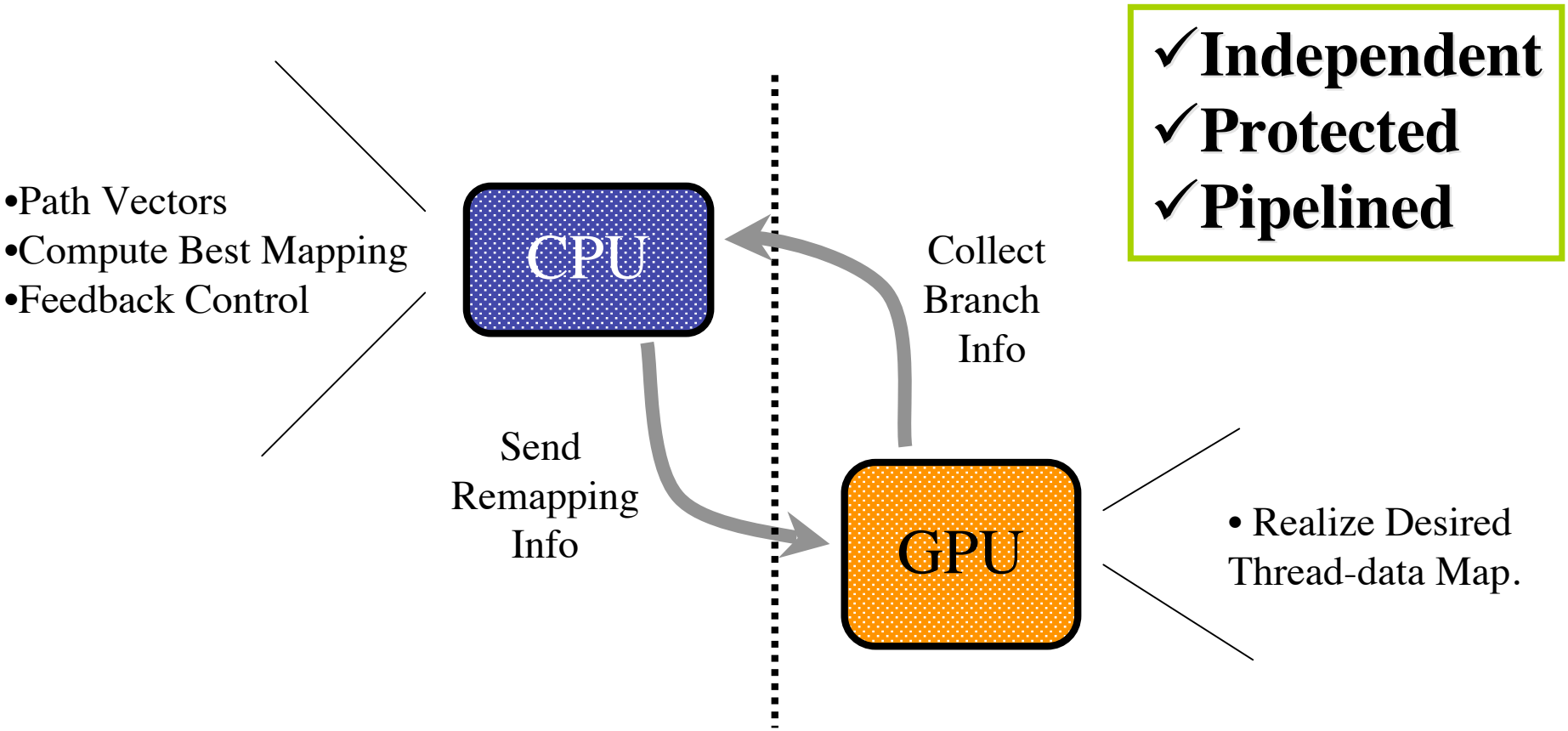
eddy@cs.wm.edu

Outline

- Thread-data Remapping
  - Concept & mechanisms
- **Transformation on the Fly**
  - CPU-GPU pipelining & LAM
- Evaluation
- Conclusion

eddy@cs.wm.edu

Overview of CPU-GPU Synergy



eddy@cs.wm.edu

CPU-GPU Pipeline Scheme

- Without Pipelining Scheme

$$\frac{T_{div}}{(T_{no\text{-}div} + T_{remap})} : \quad >=1 \text{ or } < 1$$

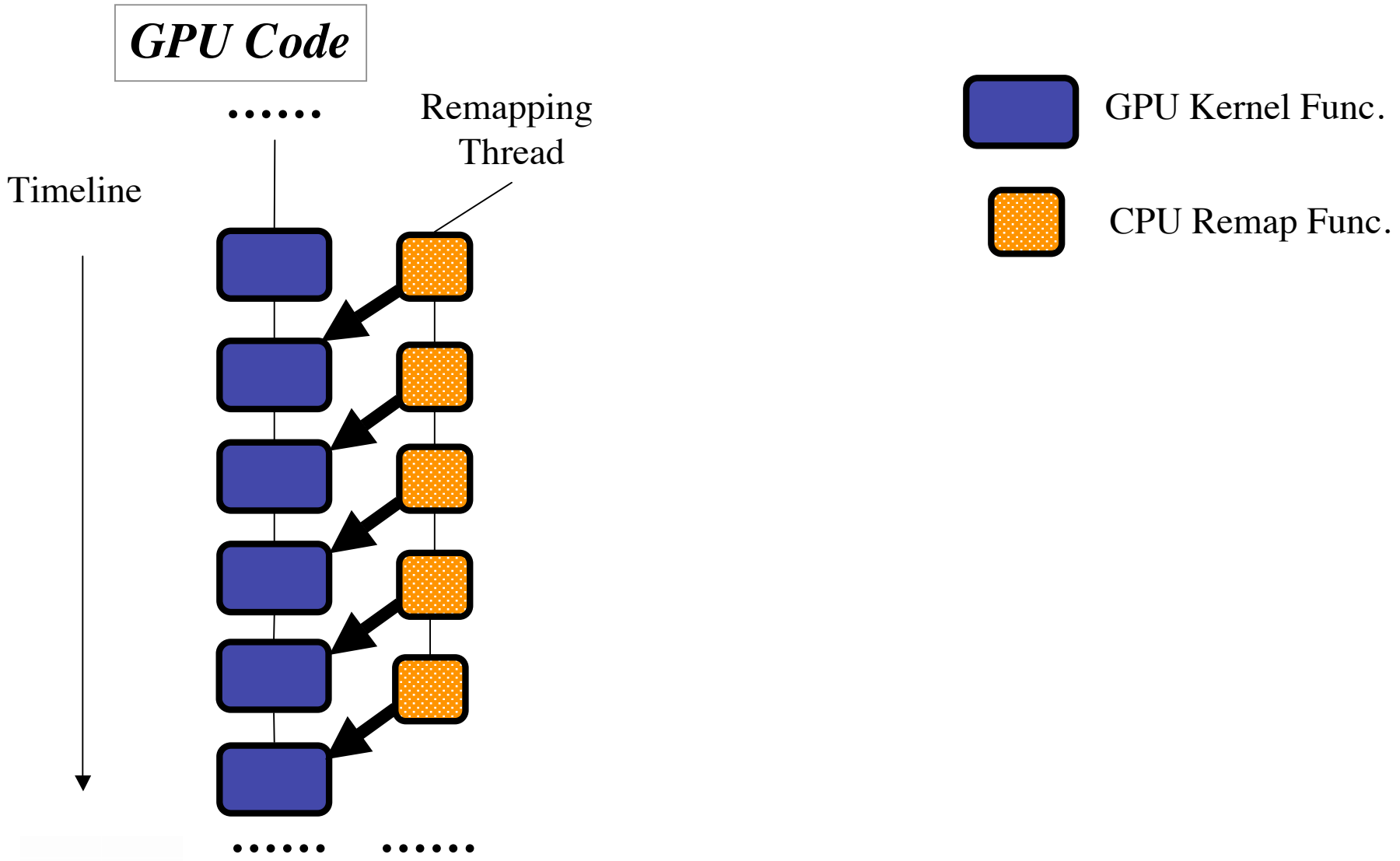
- With Pipelining Scheme

$$\frac{T_{div}}{(T_{no\text{-}div} + T_{remap})} : \quad >= 1$$

No Slow-down!

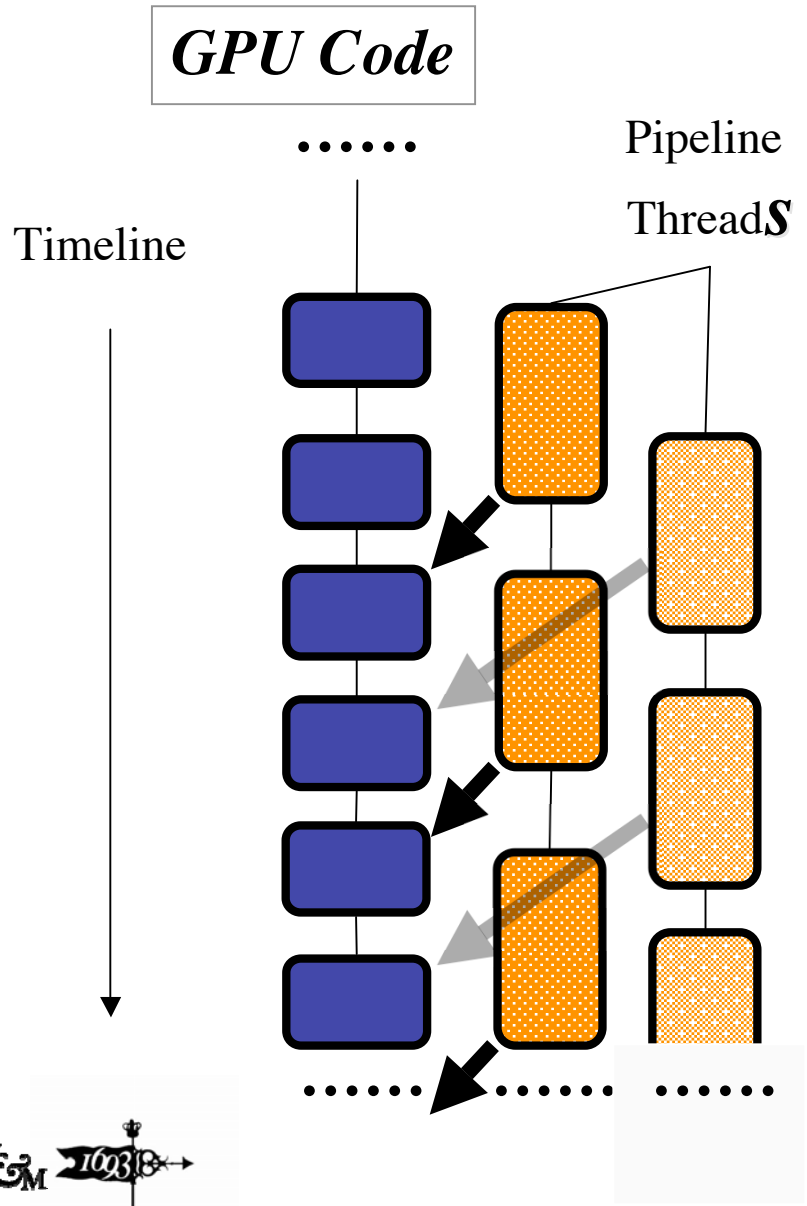
eddy@cs.wm.edu

CPU-GPU Pipeline Example I



eddy@cs.wm.edu

CPU-GPU Pipeline Example II




GPU Kernel Func.


CPU Remap Func.

- Controllable Threading
- Adaptive to Avail. Resource

eddy@cs.wm.edu

Applicable Scenarios

- Loops
  - Multiple invocation of same kernel function
- Input Data Partition for A Kernel Function
  - Create multiple iterations
- Across Different Kernels
  - With idle CPU processing system resources

eddy@cs.wm.edu

LAM: Reduce Data Movements

- For Data Layout Transformation - LAM Scheme
  - 3 Steps: Label, Assign & Move (LAM)

Tunable # of Classes

  - Label -- Classify path vectors into multiple classes
    - Based on similarity
  - Assign -- Assign warps to different classes
    - Based on occupation ratio

Increase # of no-need-moves

  - Move -- Determine the destination

eddy@cs.wm.edu

Outline

- Thread-data Remapping
  - Concept & mechanisms
- Transformation on the fly
  - CPU-GPU pipelining & LAM
- **Evaluation**
- Conclusion

eddy@cs.wm.edu

Experiment Settings

- Host Machine
  - Dual-socket quad-core Intel Xeon E5540
- GPU Device
  - NVIDIA Tesla 1060
- Runtime Library
  - Reference redirection & data transformation
  - Pipeline threads management

eddy@cs.wm.edu

Benchmarks

Program	Comments	Potential Div. Source	Percent of Div. Warps
3D-LBM	LBM based PDE solver	if statement	50-100%
GAFORT	genetic algorithm	if statement & loop	100%
Marching Cubes	graphics algorithm	if statement	100%
Reduction	parallel sum	if statement	100%
Blackscholes	option pricing	if statement	0%

eddy@cs.wm.edu

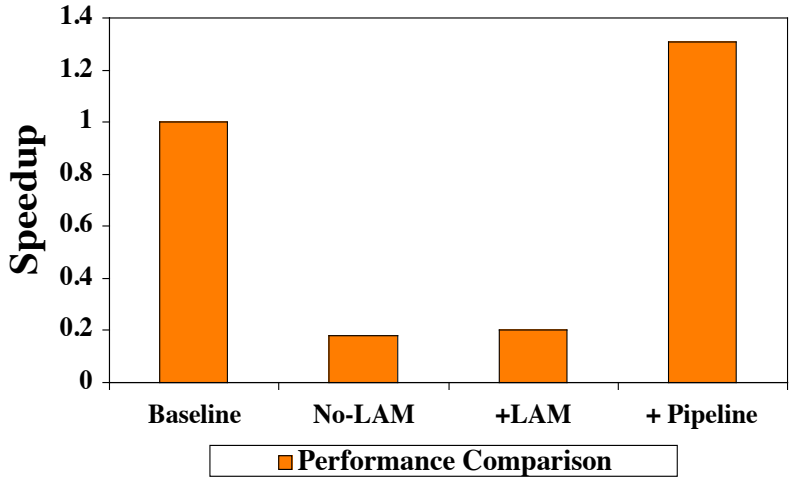
Benchmarks

Program	Comments	Potential Div. Source	Percent of Div. Warps
3D-LBM	LBM based PDE solver	if statement	50-100%
GAFORT	genetic algorithm	if statement & loop	100%
Marching Cubes	graphics algorithm	if statement	100%
Reduction	parallel sum	if statement	100%
Blackscholes	option pricing	if statement	0%

eddy@cs.wm.edu

Evaluation: Data Transformation

- GAFORT
  - Mutation probabilities
  - Regular mem. access
  - ***select_cld*** kernel
  - Remap scheme
    - Data layout transform.
  - Efficiency control
    - LAM & Pipeline



Perform.	Before	After
Div. Ratio	100%	56%
Time	67225	51325

44% Reduced

1.31 Speedup

eddy@cs.wm.edu

Evaluation: Reference Redirect.

- MarchingCubes
  - Div: number of vertices that intersect isosurface
  - Random memory access
  - ***generateTriangles2*** kernel
  - Remap scheme
    - Reference redirection
  - Efficiency control
    - Pipeline

Time (micro-sec)

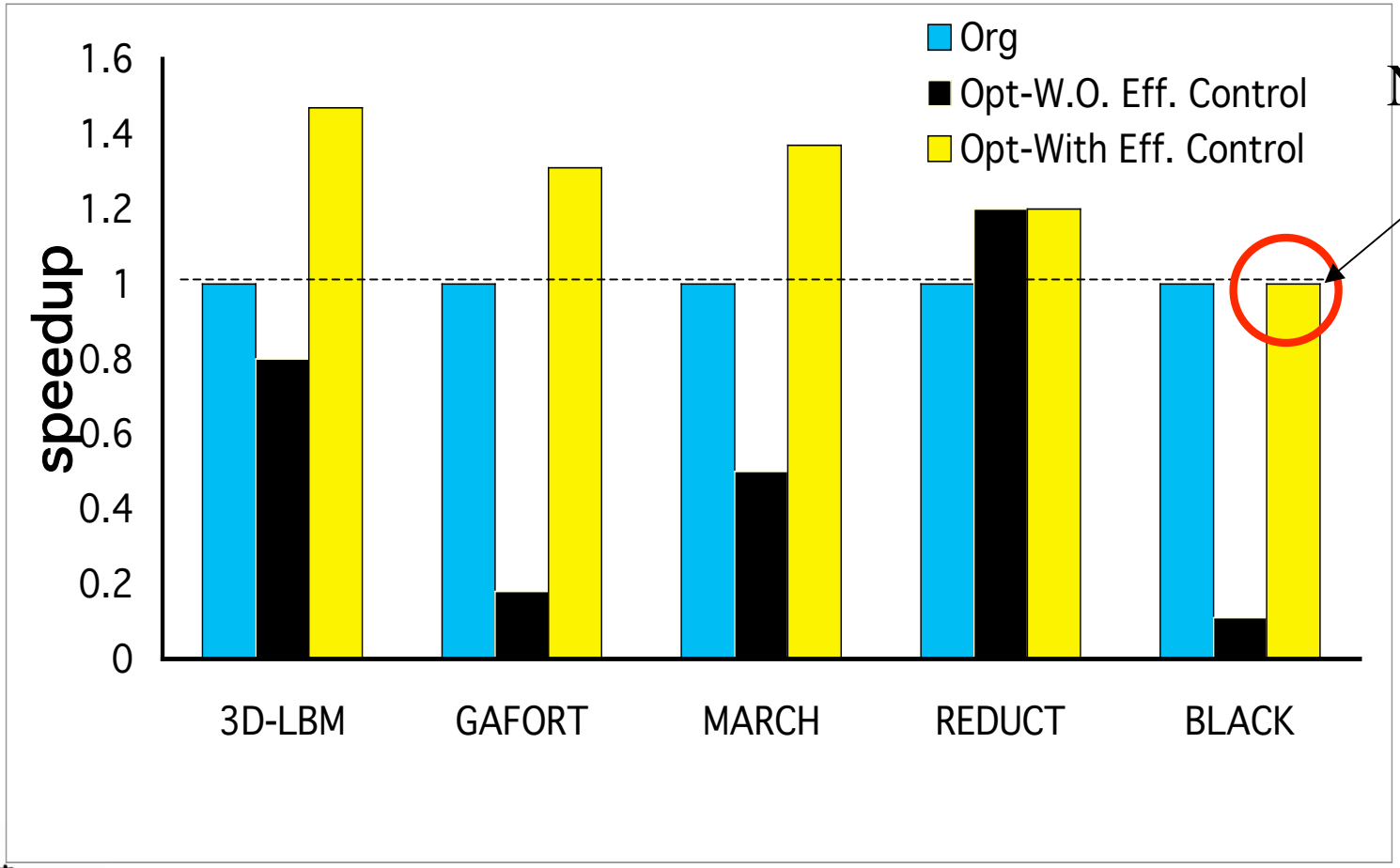
BlockSize	32	64	256
Org.Time	17414	16707	16673
Opt.Time	12666	12371	12425

Performance

BlockSize	32	64	256
Div.Reduct	99%	99%	99%
SpeedUp	1.37	1.35	1.34

eddy@cs.wm.edu

Evaluation: All Benchmarks



eddy@cs.wm.edu

Conclusion

- An Efficient Software Solution for GPU Div.
  - Mechanism
    - On-the-fly thread-data remapping
  - Overhead control
    - CPU-GPU pipelining: whole-system synergy
    - LAM scheme: balance between benefit & overhead
  - Effectiveness
    - Up to 1.4x speedup
    - Efficiency protection: no slowdown

eddy@cs.wm.edu

Acknowledgement

- Ye Zhao
- Xiaoming Li
- NVIDIA
  - Donation of GPU Device
- NSF Funds & IBM CAS Fellowship
- Anonymous Reviewers

eddy@cs.wm.edu

Questions?

eddy@cs.wm.edu