Maximizing SIMD Resource Utilization in GPGPUs with SIMD Lane Permutation

Minsoo Rhu and Mattan Erez

The University of Texas at Austin
Electrical and Computer Engineering Department

Introduction

- SIMD-based GPU Architectures
  - Suffers from low SIMD efficiency when application contains highly irregular control flow
  - ***Compaction***-based architectures have recently gained high interest as a way of minimizing the waste in SIMD units
    - *Problem*: applicability of compaction limited to highly divergent applications, despite its design overhead

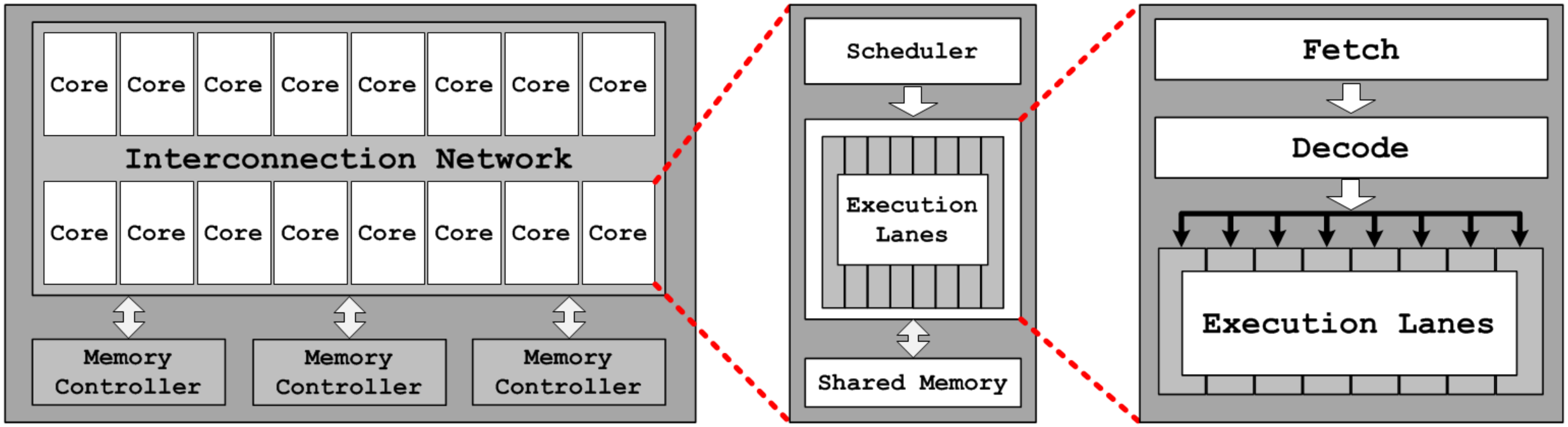
- Main contribution of this paper
  - Analysis on the cause of limited effectiveness of compaction
    - Concentration of active threads to particular SIMD lanes
  - Static/deterministic permutation of thread assignment to SIMD units to enhance the effectiveness of compaction
    - Improve SIMD utilization and performance
    - Widen the applicability of compaction schemes

Outline

- **GPU and SIMD compaction background**
- Aligned divergence and compactability
- SLP – **S**IMD **L**ane **P**ermutation
- Evaluation

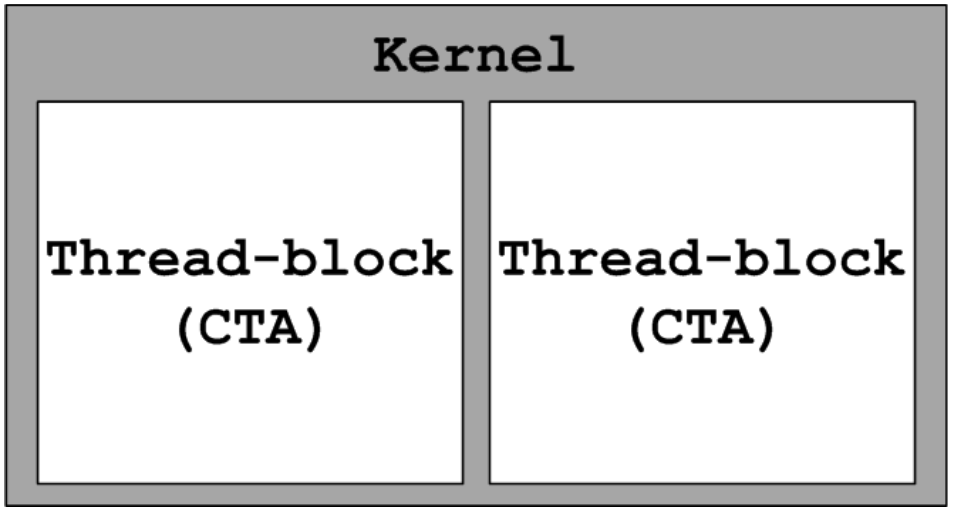
Graphic Processing Units (GPUs)

- General-purpose many-core accelerators
  - Supports non-graphics APIs (e.g. CUDA, OpenCL)
- Scalar frontend (fetch & decode) + parallel backend
  - Amortizes the cost of frontend and control



CUDA exposes hierarchy of data-parallel threads

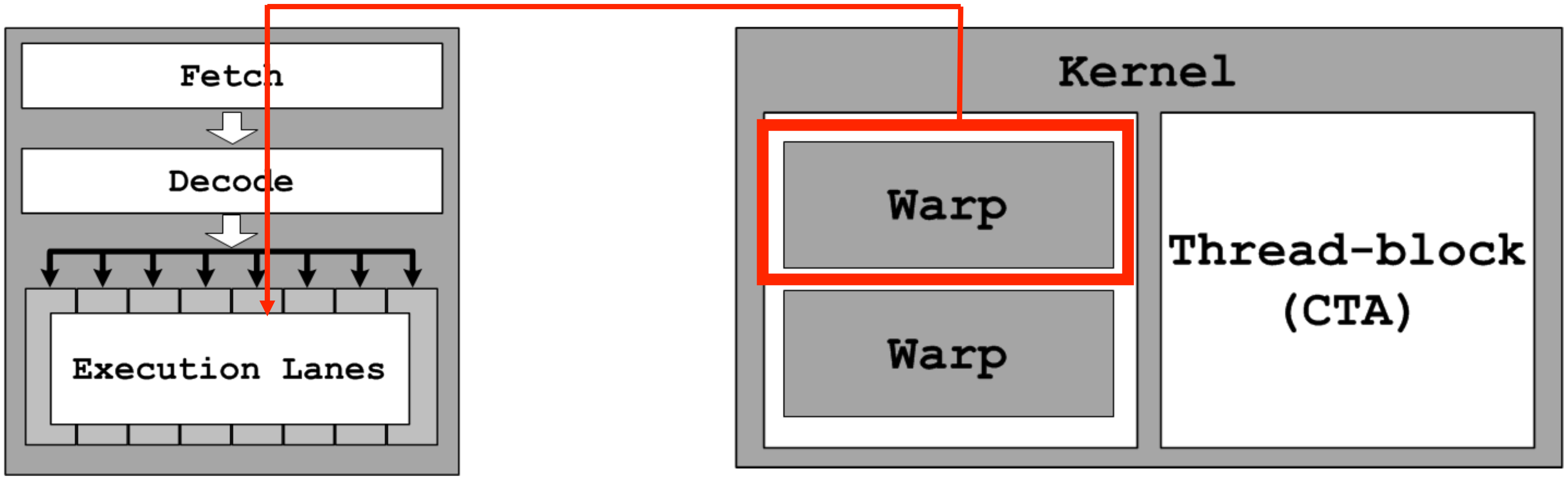
- **SPMD model:** single ***kernel*** executed by all threads
- **Kernel / Thread-block**
  - Multiple **thread-blocks** (concurrent-thread-arrays(**CTA**s)) compose a **kernel**



CUDA exposes hierarchy of data-parallel threads

- **SPMD model:** single ***kernel*** executed by all threads
- **Kernel / Thread-block / *Warp* / *Thread***
  - Multiple **warps** compose a **thread-block**
  - Multiple **threads (32)** compose a warp

A warp is scheduled as a *batch* of threads

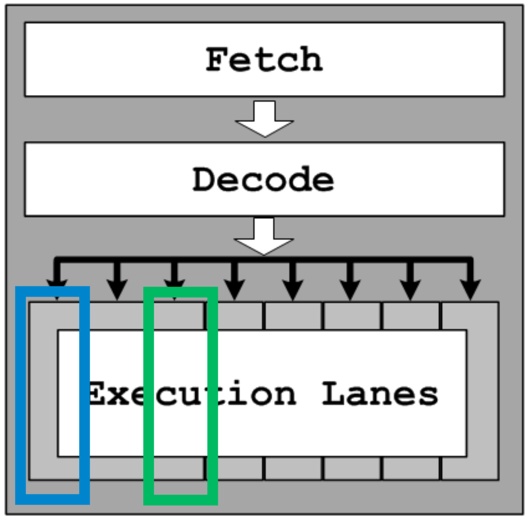


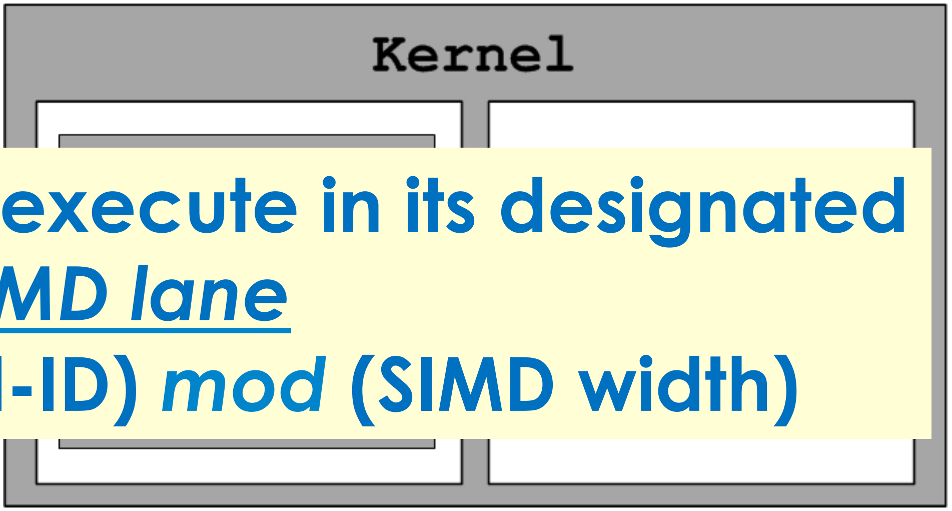
Assumption: width of SIMD lanes matches warp size (i.e. 32 SIMD lanes)

CUDA exposes hierarchy of data-parallel threads

- **SPMD model:** single ***kernel*** executed by all threads
- **Kernel / Thread-block / *Warp* / *Thread***
  - Multiple **warps** compose a **thread-block**
  - Multiple **threads (32)** compose a warp

: Thread-ID 0, 32, 64 … execute in physical lane #0
: Thread-ID 2, 34, 66 … execute in physical lane #2



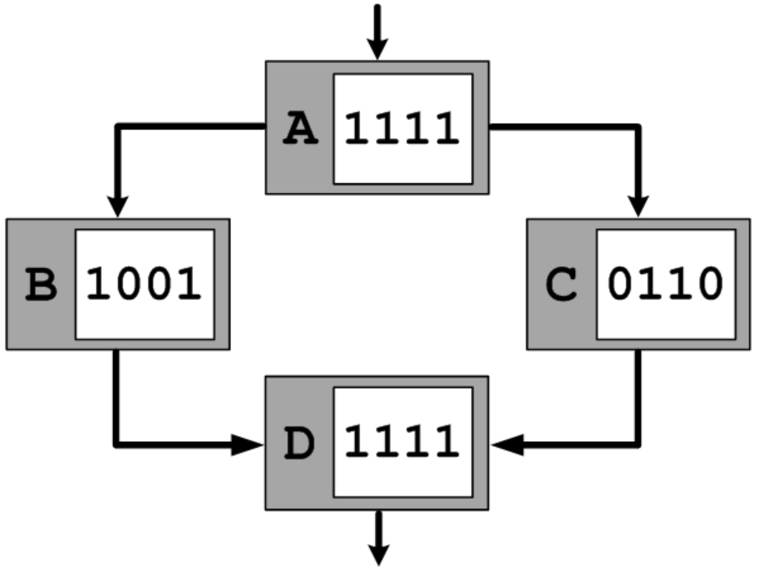


Threads execute in its designated *home SIMD lane*
: (thread-ID) *mod* (SIMD width)

Assumption: width of SIMD lanes matches warp size (i.e. 32 SIMD lanes)

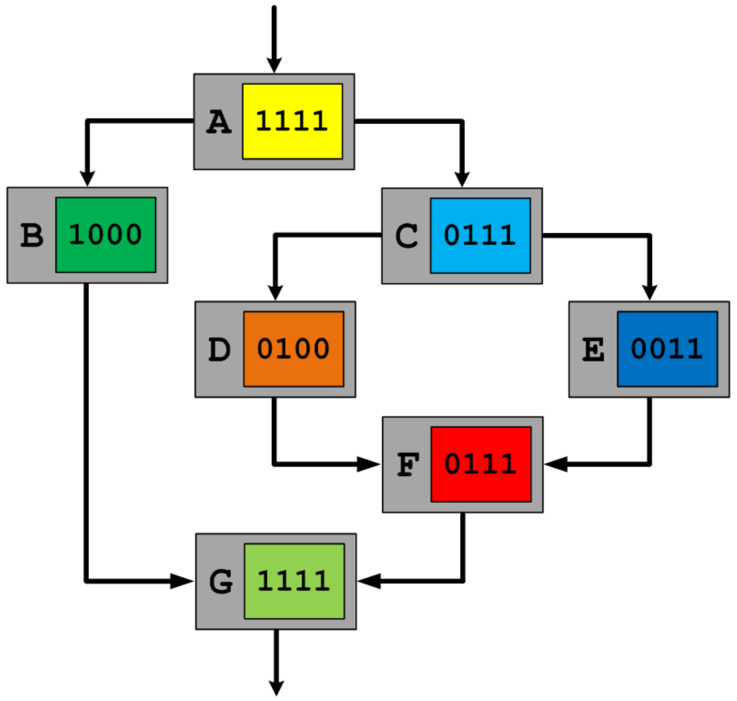
GPUs have HW support for conditional branches

- **Control divergence**: threads in warp branch differently
  - a.k.a. **Branch Divergence**
  - *Stack*-based divergence/reconvergence model
  - Active bitmasks (or predicate bitmasks):
    - Bitmasks for both true/false paths dynamically derived
    - Allows subset of a warp to commit results



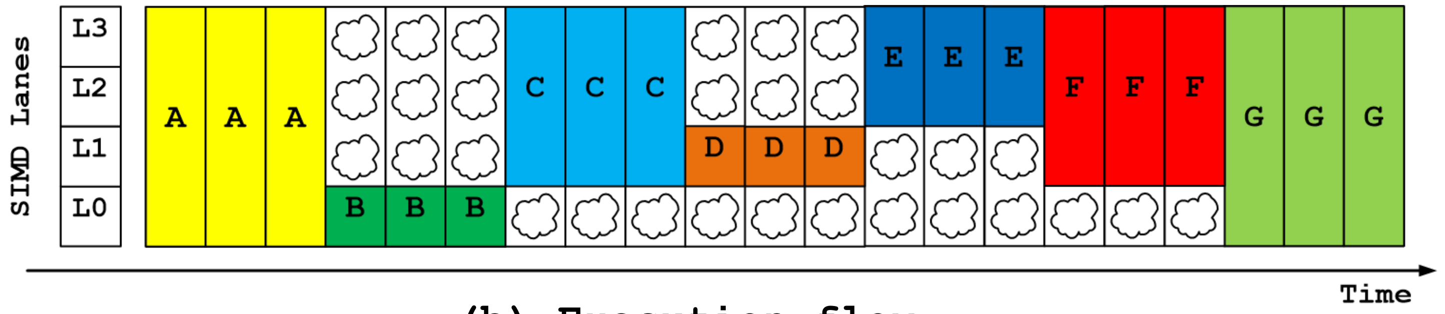
(a) Example Control Flow Graph
1: Active, 0: Inactive

Challenge: Underutilization of SIMD units



(a) Example Control Flow Graph

- Number of active threads in a warp decreases every time control diverges
- Active area of research
  - Thread block compaction [Fung'11]
  - Larger warps [Narasiman'11]



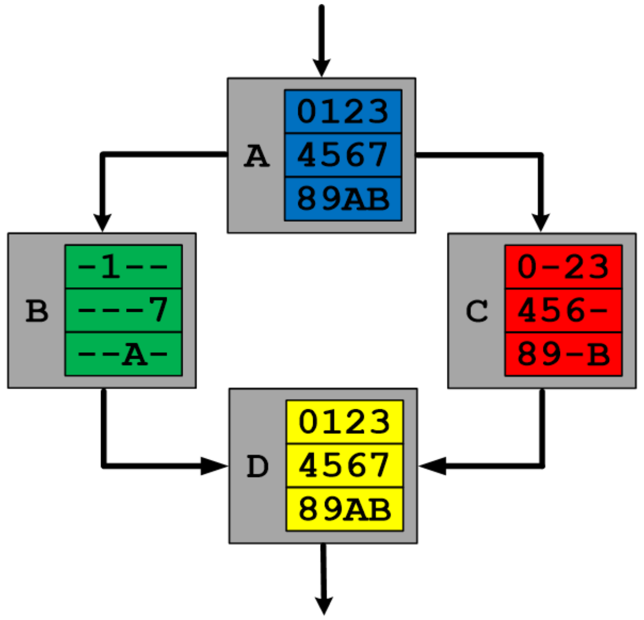
(b) Execution flow

: Threads (lanes) masked out from execution, remaining idle.

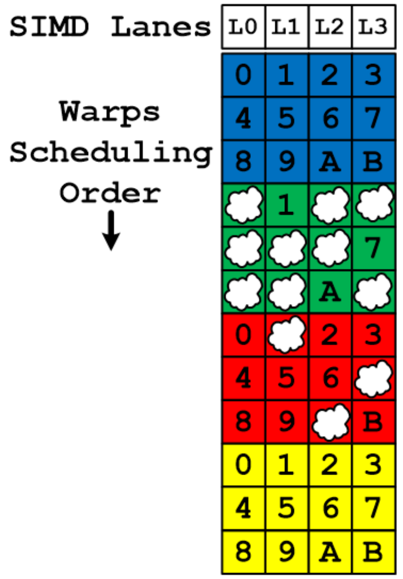
Thread-block compaction (TBC) [Fung'11]

- Dynamically ***compact*** warps within a ***thread-block***
  - ***<u>Synchronize</u>*** all warps at conditional-branches and reconvergence points
  - Threads with *<u>different</u>* home SIMD lanes compacted together

: [0,1… A,B] refers to *<u>active</u>* thread-IDs executing in that basic block
: [-] refers to *<u>inactive</u>* threads, masked out from execution



(a) Example control flow graph

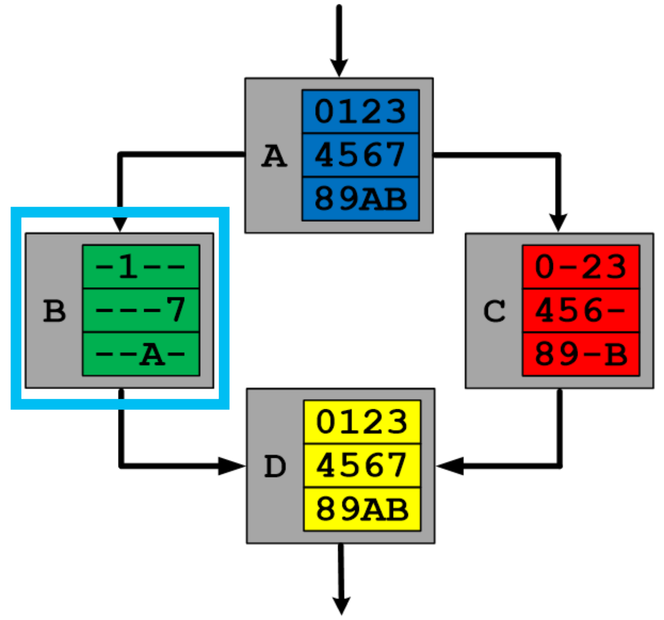


(b) Execution flow without compaction

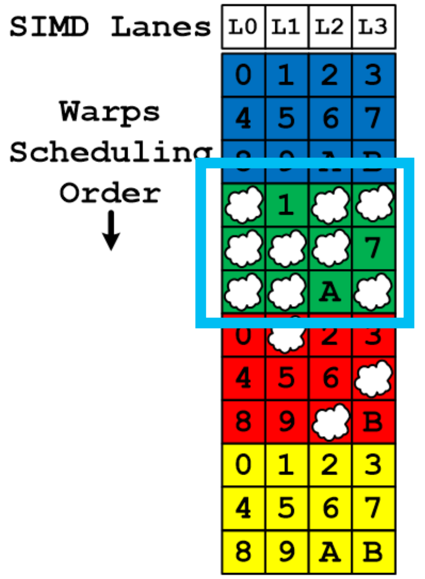
Thread-block compaction (TBC) [Fung'11]

- Dynamically ***compact*** warps within a ***thread-block***
  - ***Synchronize*** all warps at conditional-branches and reconvergence points
  - Threads with *different* home SIMD lanes compacted together

: [0,1… A,B] refers to *active* thread-IDs executing in that basic block
: [-] refers to *inactive* threads, masked out from execution



(a) Example control flow graph

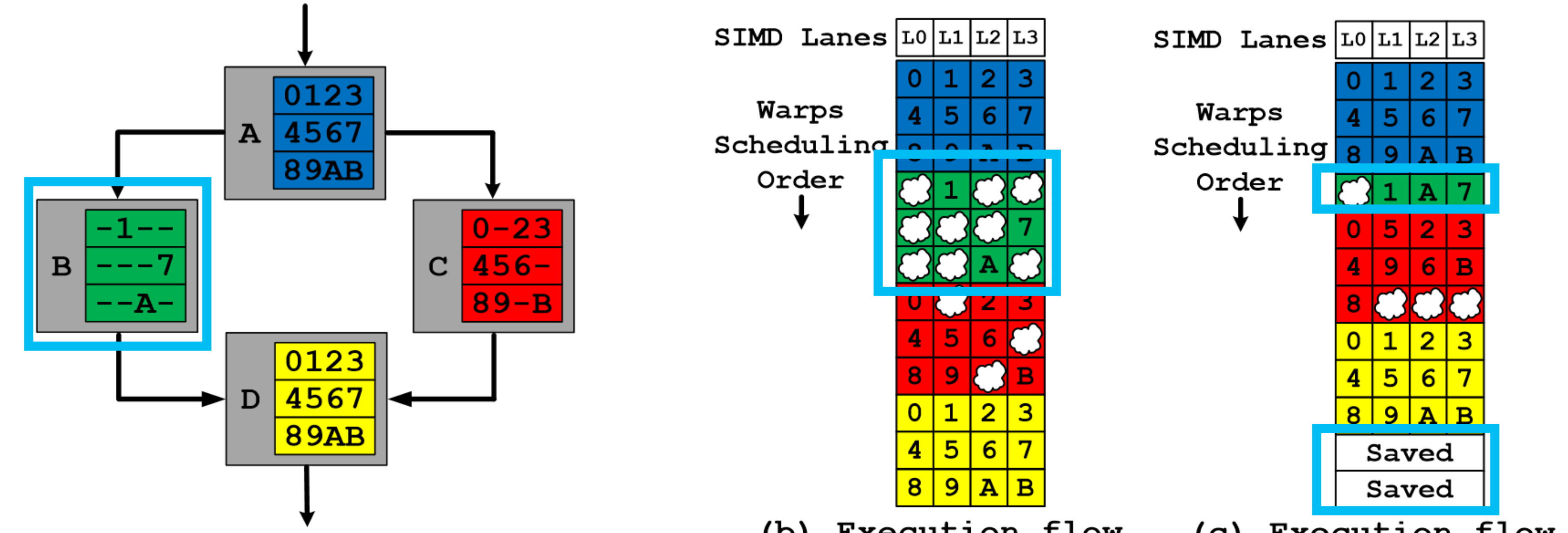


(b) Execution flow without compaction

Thread-block compaction (TBC) [Fung'11]

- Dynamically ***compact*** warps within a ***thread-block***
  - ***Synchronize*** all warps at conditional-branches and reconvergence points
  - Threads with *different* home SIMD lanes compacted together

: [0,1… A,B] refers to *active* thread-IDs executing in that basic block
: [-] refers to *inactive* threads, masked out from execution



(a) Example control flow graph

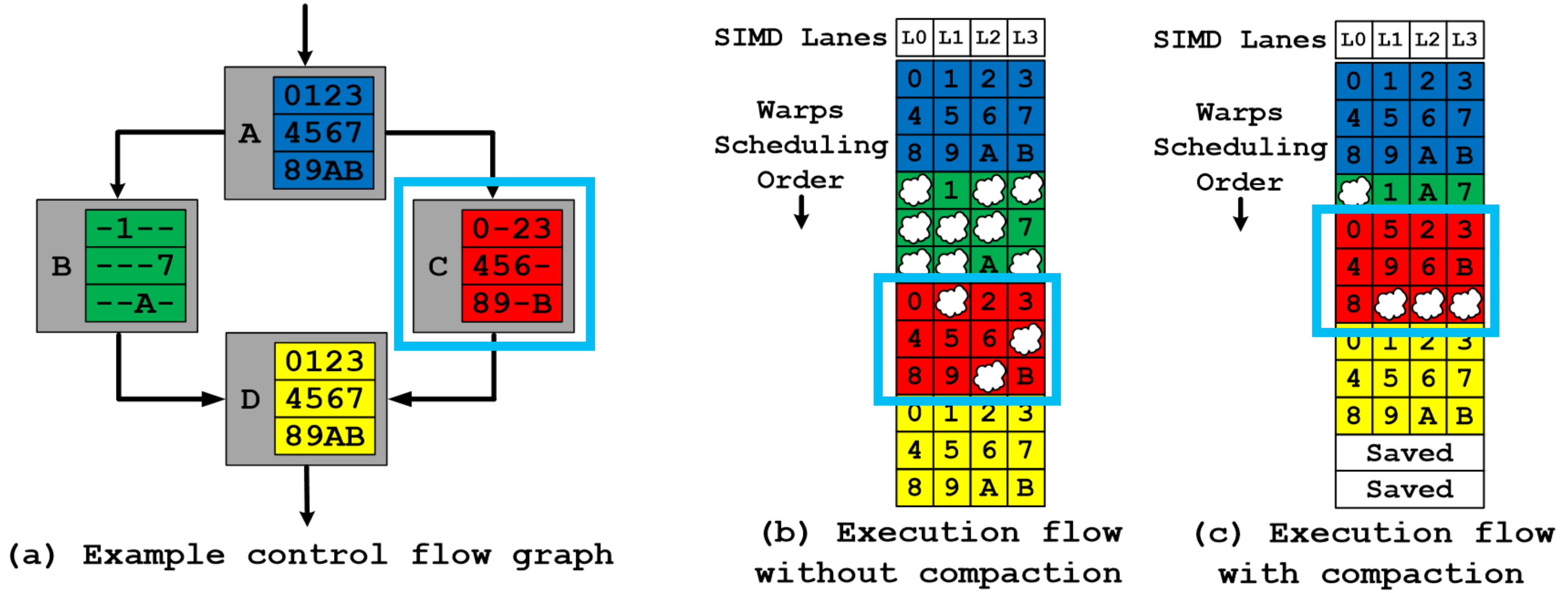
(b) Execution flow without compaction

(c) Execution flow with compaction

Thread-block compaction (TBC) [Fung'11]

- Dynamically ***compact*** warps within a ***thread-block***
  - ***Synchronize*** all warps at conditional-branches and reconvergence points
  - Threads with *different* home SIMD lanes compacted together

: [0,1… A,B] refers to *active* thread-IDs executing in that basic block
: [-] refers to *inactive* threads, masked out from execution



(a) Example control flow graph

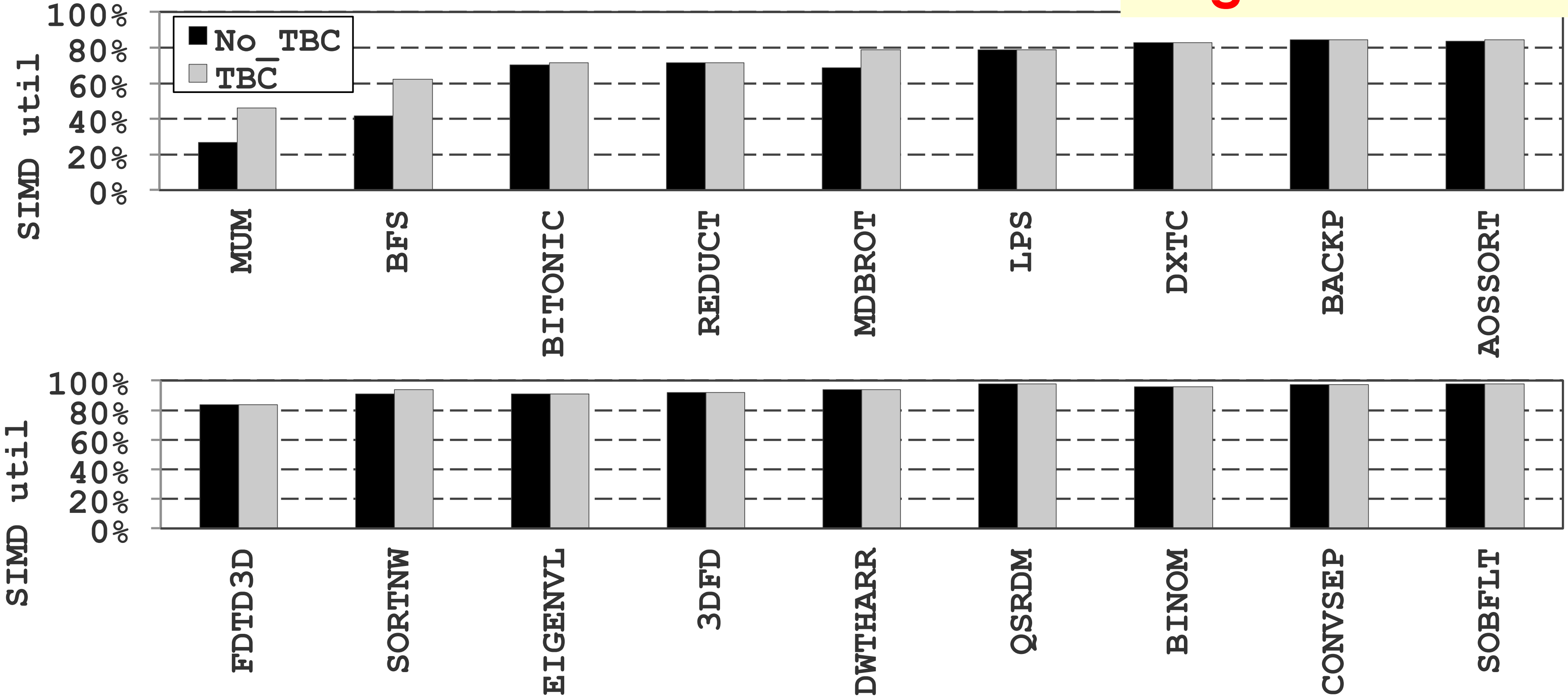
(b) Execution flow without compaction

(c) Execution flow with compaction

TBC as-is

- Average SIMD lanes occupied for execution
  - **No_TBC** : Baseline architecture ***without*** compaction
  - **TBC** : baseline compaction mechanism

Higher is better

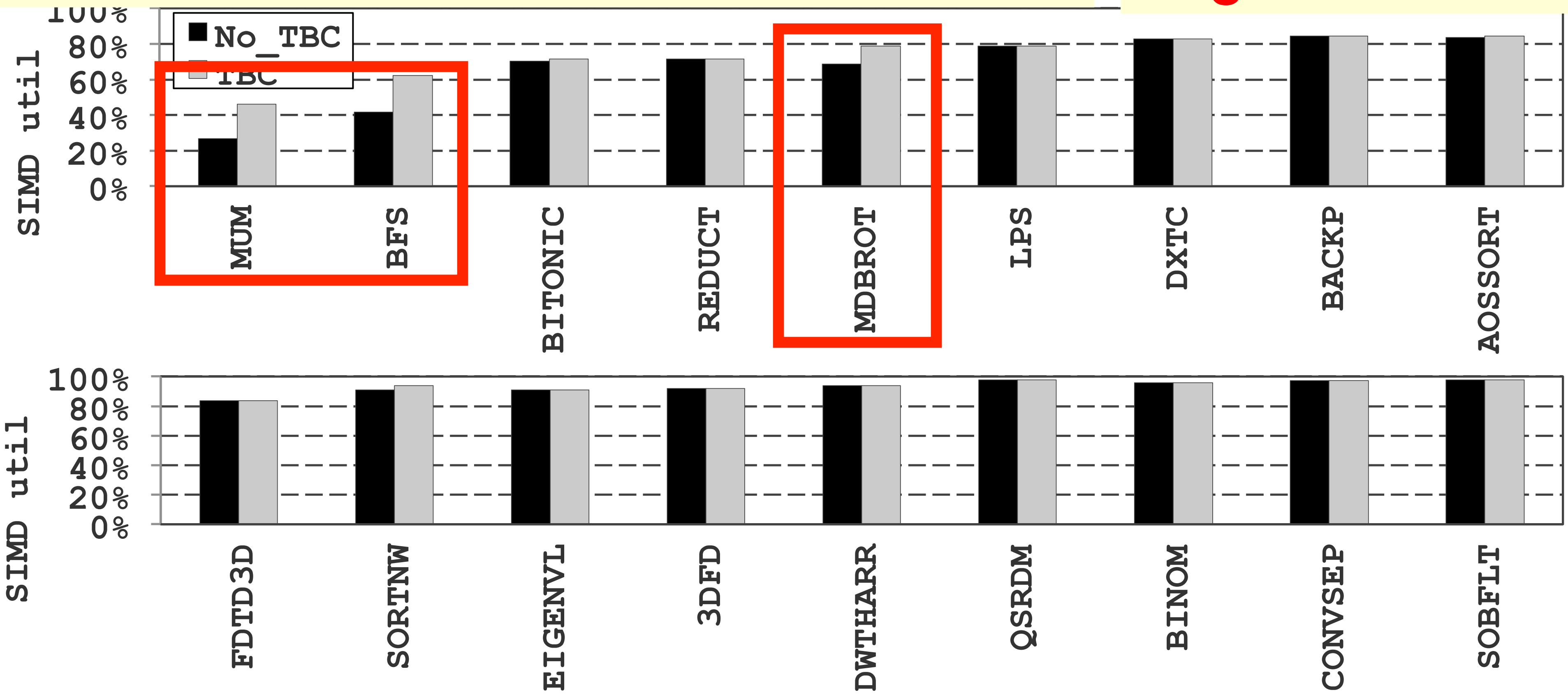


TBC as-is

- Average SIMD lanes occupied for execution
  - **No_TBC** : Baseline architecture without compaction

TBC, in general, only effective for *highly divergent* applications

Higher is better



Outline

- GPU and SIMD compaction background
- **Branch compactability and aligned divergence**
- SLP – **S**IMD **L**ane **P**ermutation
- Evaluation

When Does Compaction Fail?

- Most (or all) of SIMD lanes _already_ occupied
  - Compaction inherently impossible

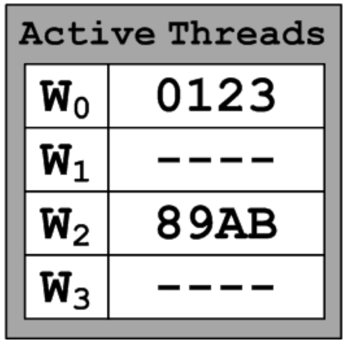
Active Threads	
W_0	0123
W_1	----
W_2	89AB
W_3	----

Active Threads	
W_0	012-
W_1	456-
W_2	89A-
W_3	CDE-

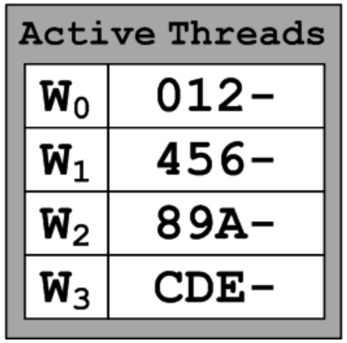
Active Threads	
W_0	0123
W_1	4567
W_2	89AB
W_3	CD--

When Does Compaction Fail?

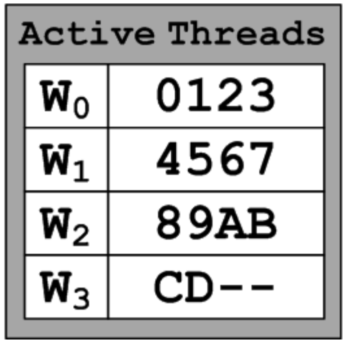
- Most (or all) of SIMD lanes *already* occupied
  - Compaction inherently impossible



Active Threads	
W_0	0123
W_1	----
W_2	89AB
W_3	----



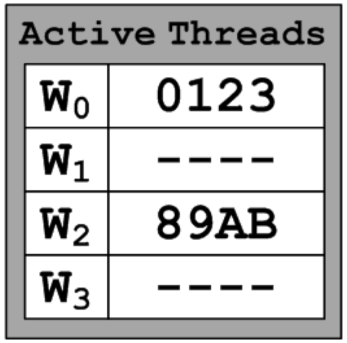
Active Threads	
W_0	012-
W_1	456-
W_2	89A-
W_3	CDE-



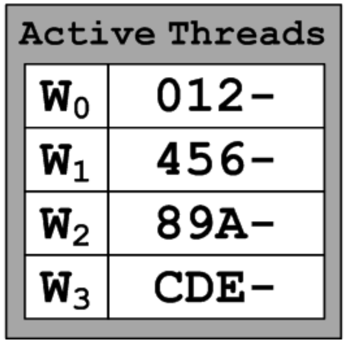
Active Threads	
W_0	0123
W_1	4567
W_2	89AB
W_3	CD--

When Does Compaction Fail?

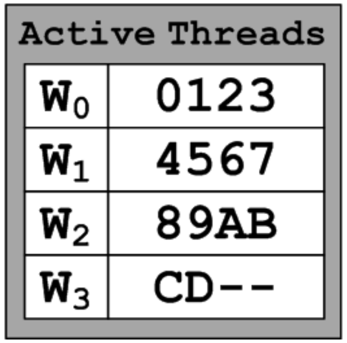
- Most (or all) of SIMD lanes *<u>already</u>* occupied
  - Compaction inherently impossible



Active Threads	
W_0	0123
W_1	----
W_2	89AB
W_3	----

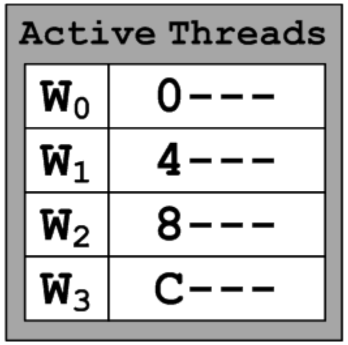


Active Threads	
W_0	012-
W_1	456-
W_2	89A-
W_3	CDE-

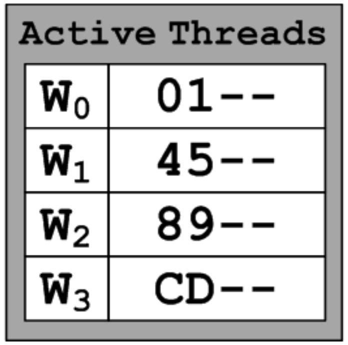


Active Threads	
W_0	0123
W_1	4567
W_2	89AB
W_3	CD--

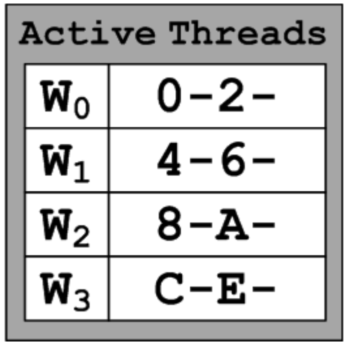
- Active threads *<u>aligned (clustered)</u>* on certain lanes
  - Compaction theoretically feasible, if crossbars can distribute operands across different SIMD lanes



Active Threads	
W_0	0---
W_1	4---
W_2	8---
W_3	C---



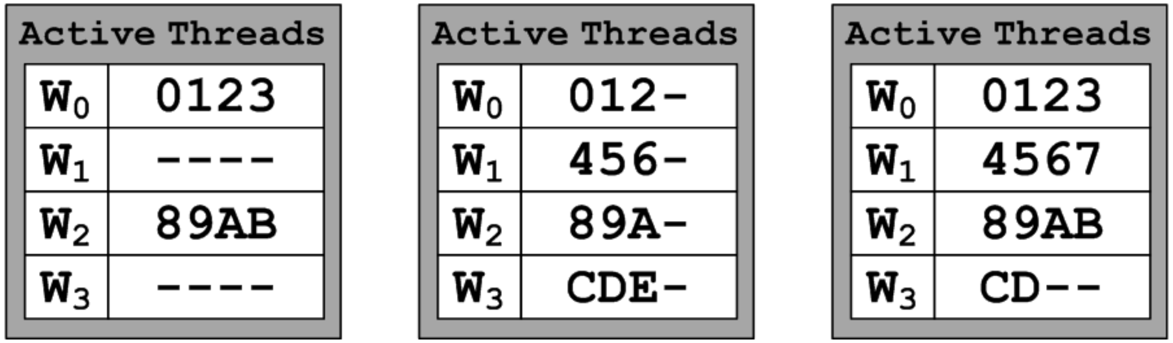
Active Threads	
W_0	01--
W_1	45--
W_2	89--
W_3	CD--



Active Threads	
W_0	0-2-
W_1	4-6-
W_2	8-A-
W_3	C-E-

When Does Compaction Fail?

- Most (or all) of SIMD lanes *already* occupied
  - Compaction inherently impossible



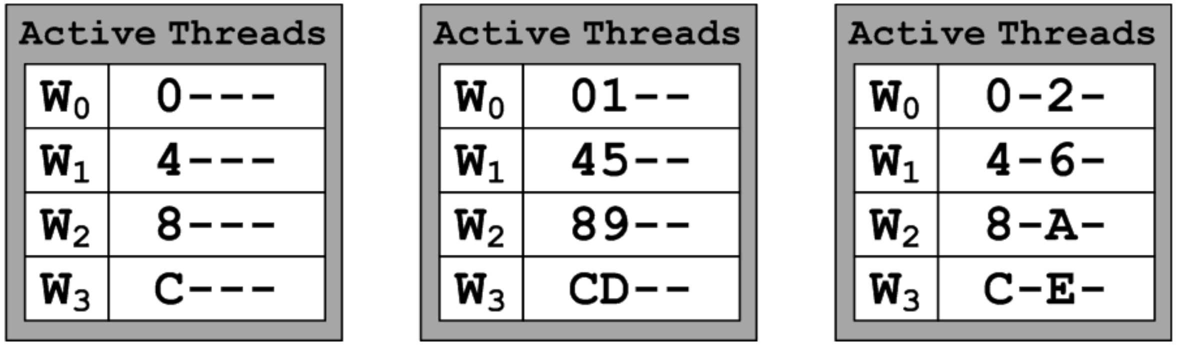
Active Threads	
W_0	0123
W_1	----
W_2	89AB
W_3	----

Active Threads	
W_0	012-
W_1	456-
W_2	89A-
W_3	CDE-

Active Threads	
W_0	0123
W_1	4567
W_2	89AB
W_3	CD--

- Active threads *aligned (clustered)* on certain lanes
  - Compaction the ute operands across different SIMD lanes

Aligned Divergence!



Active Threads	
W_0	0---
W_1	4---
W_2	8---
W_3	C---

Active Threads	
W_0	01--
W_1	45--
W_2	89--
W_3	CD--

Active Threads	
W_0	0-2-
W_1	4-6-
W_2	8-A-
W_3	C-E-

Aligned divergence – (1)

- Case 1: Branch condition depends on ***data*** array
  - Each thread references ***different*** element of the array
  - Unlikely to cause aligned divergence

Code #1) Branch depending on data arrays

```
// Code snippet from the kernel of BFS benchmark
// g_graph_visited and g_graph_edges are data array parameters.

int tid = blockIdx.x*MAX_THREADS_PER_BLOCK + threadIdx.x;

…
int id = g_graph_edges[…];
if( !g_graph_visited[id] )
{
      …
}
```

Aligned divergence – (1)

- Case 1: Branch condition depends on ***data*** array
  - Each thread references ***different*** element of the array
  - Unlikely to cause aligned divergence

Code #1) Branch depending on data arrays

```
// Code snippet from the kernel of BFS benchmark
// g_graph_visited and g_graph_edges are data array parameters.

int tid = blockIdx.x*MAX_THREADS_PER_BLOCK + threadIdx.x;

…
int id = g_graph_edges[…];
if( !g_graph_visited[id] )
{
      …
}
```

D-Branches

Aligned divergence – (2)

- Case 2: Branch cond. depends on ***programmatic*** value
  - *Indices* of thread-ID, warp-ID, CTA-ID, *width/height* of CTA
  - *Scalar* input parameters to the kernel, *constants*
  - Threads sharing home SIMD lane likely to reference ***same*** value

Code #2) Programmatic branch causing only the 1st half of the warp active

```
// Code snippet from the kernel of BACKP benchmark
// CTA is a (8 × 16) 2-D array of threads.

int tx = threadIdx.x;
int ty = threadIdx.y;
…
for (int i=1; i<=__log2f(HEIGHT); i++){
   int power_two = __powf(2,i);

   if( ty % power_two == 0 ) {…}
   …
}
```

Aligned divergence – (2)

- Case 2: Branch cond. depends on ***programmatic*** value
  - *Indices* of thread-ID, warp-ID, CTA-ID, *width/height* of CTA
  - *Scalar* input parameters to the kernel, *constants*
  - Threads sharing home SIMD lane likely to reference ***same*** value

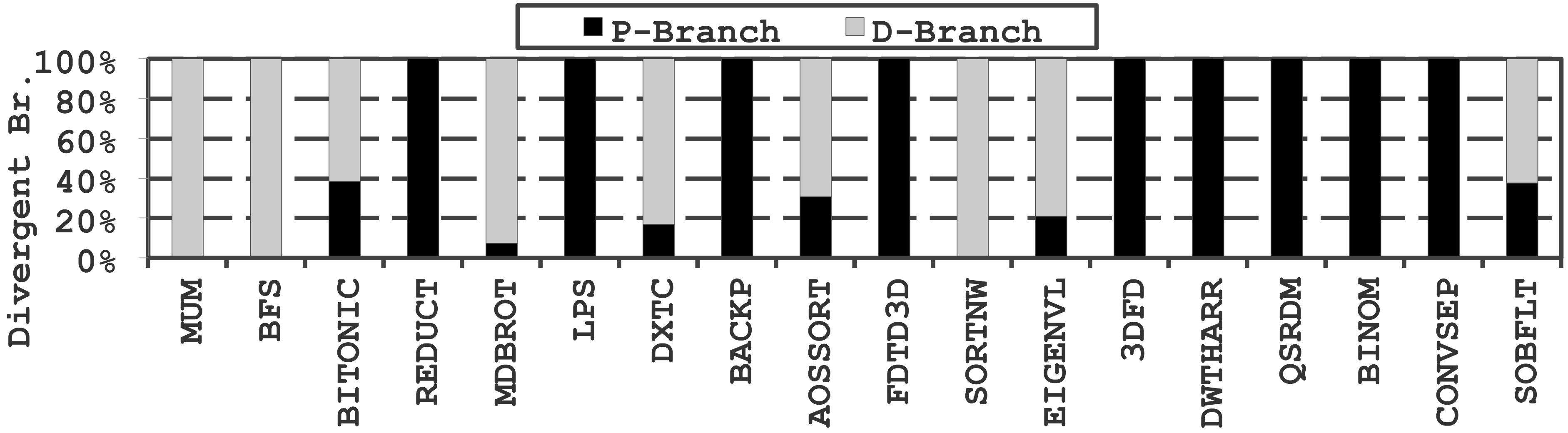
Code #2) Programmatic branch causing only the 1st half of the warp active

```
// Code snippet from the kernel of BACKP benchmark
// CTA is a (8 × 16) 2-D array of threads.

int tx = threadIdx.x;
int ty = threadIdx.y;
…
for (int i=1; i<=__log2f(HEIGHT); i++){
  int power_two = __powf(2,i);

  if( ty % power_two == 0 ) {…}
  …
}
```

P-Branches

Compaction Rate of P-/D-Branches (with TBC)

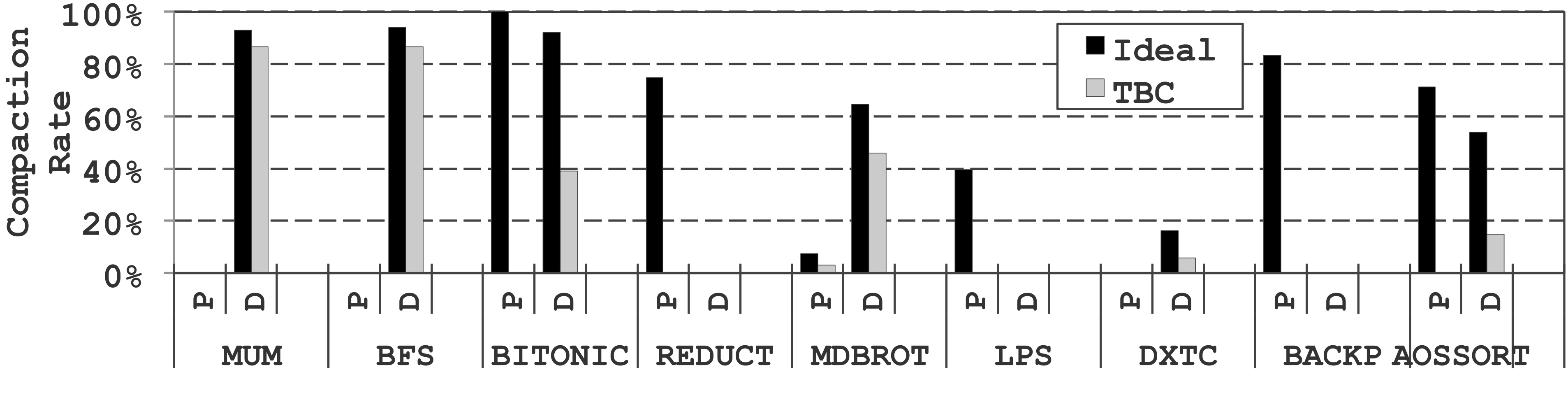
- **Definition**: Fraction of *compactable* paths among *all* paths generated by divergent branches

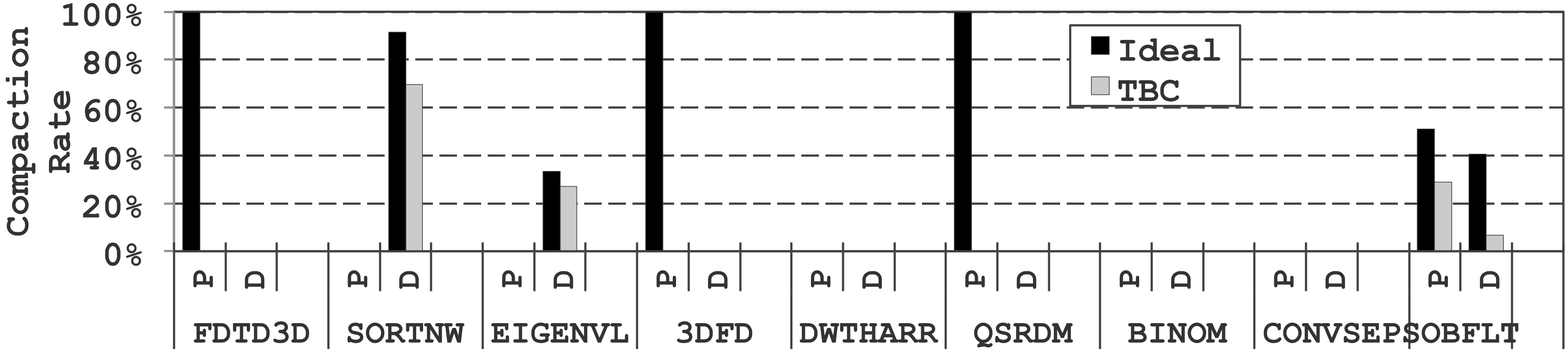


Branch categorization done with GPUOcelot using Taint-analysis (detailed in paper)

Compaction Rate of P-/D-Branches (with TBC)

- **Definition**: Fraction of _compactable_ paths among *all* paths generated by divergent branches



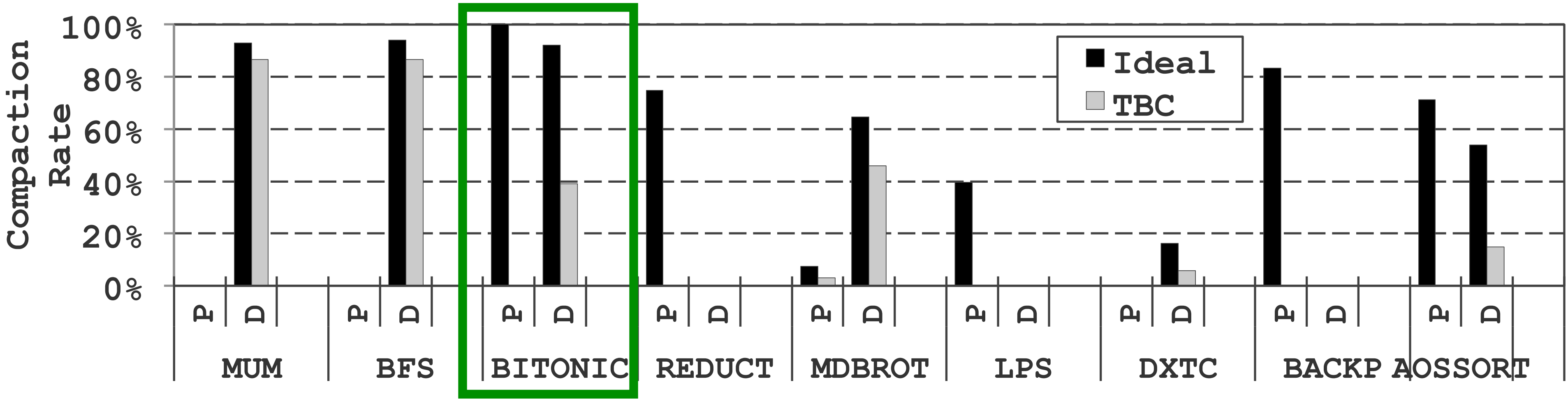


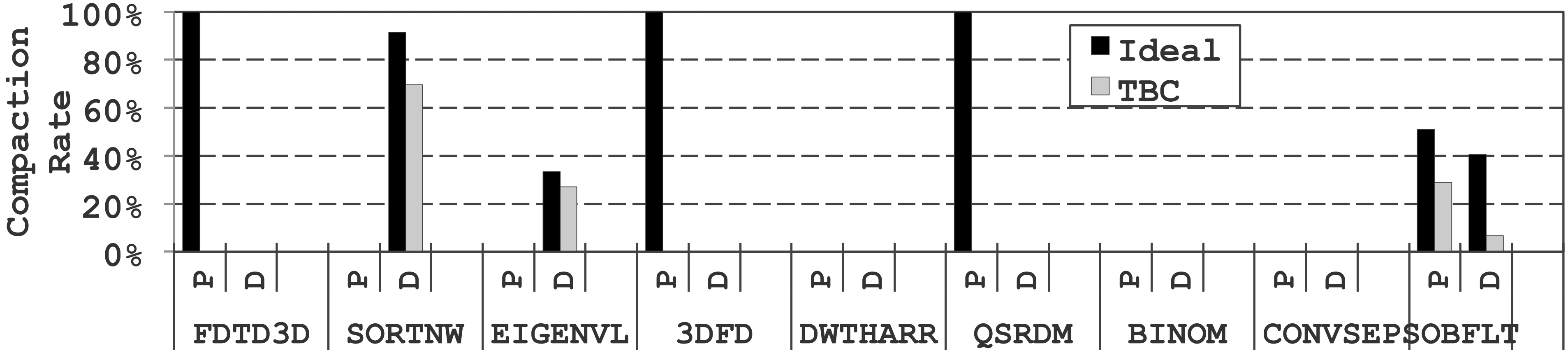
Ideal: TBC with crossbar

Higher is better

Compaction Rate of P-/D-Branches (with TBC)

D-branch compacts well with TBC
P-branch not so much compared to *Ideal*





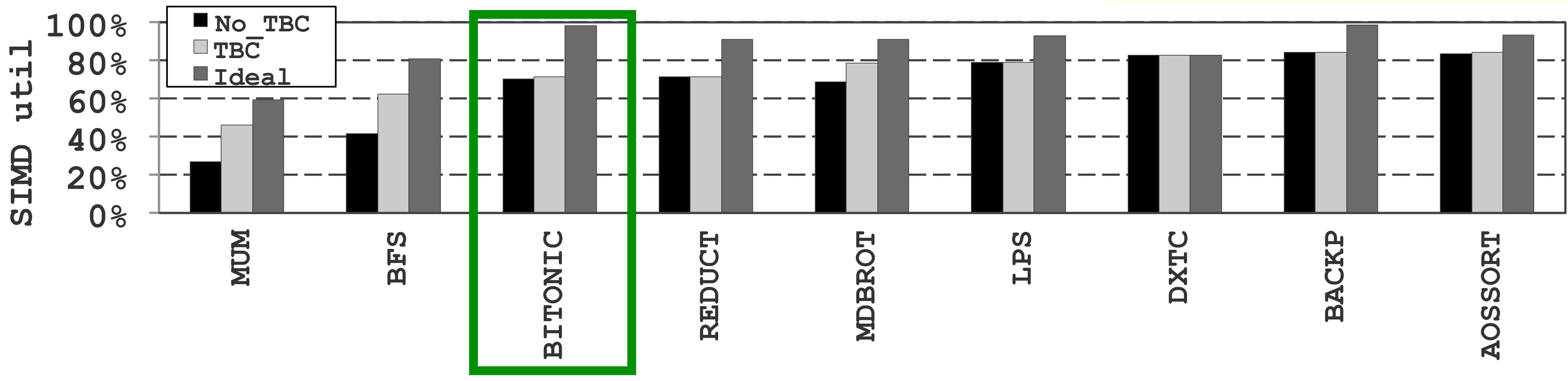
Ideal: TBC with crossbar

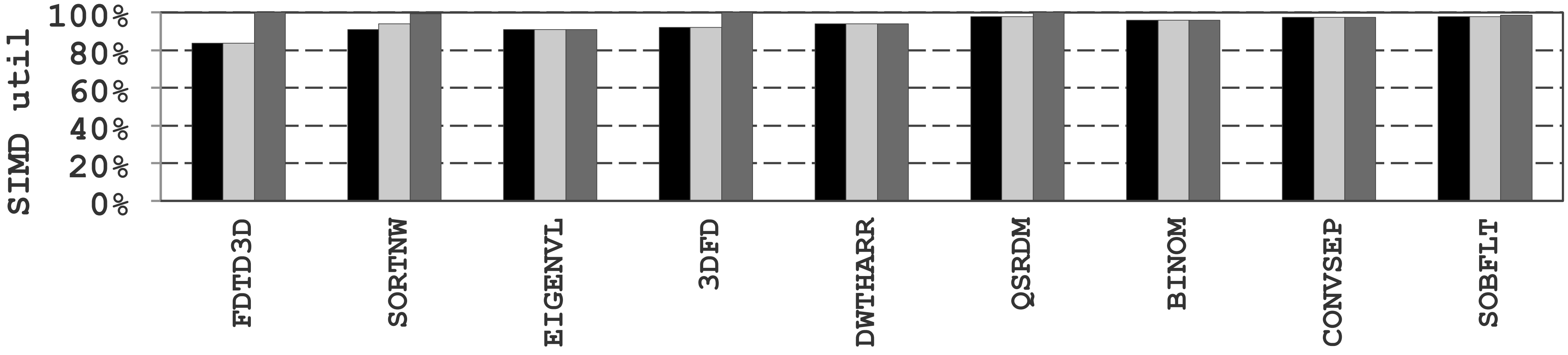
Higher is better

TBC with Crossbar (*Ideal* Compaction)

- Average SIMD lanes occupied for execution

Higher is better





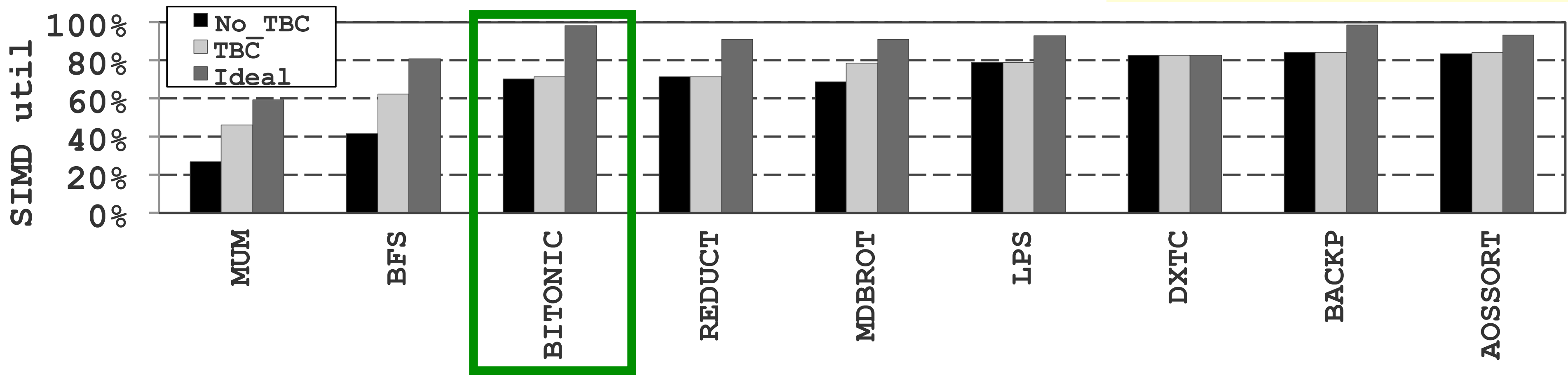
tion)

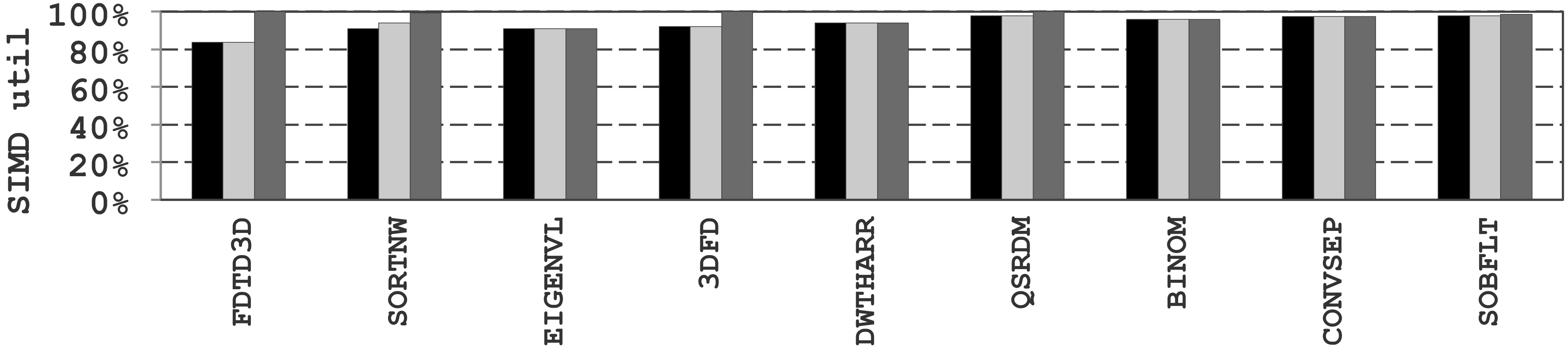
ecution

Significant *opportunities* to enhance SIMD efficiency with P-branches.

But crossbars are *expensive!*

Higher is better





Outline

- GPU and SIMD compaction background
- Branch compactability and aligned divergence
- **SLP – <u>S</u>IMD <u>L</u>ane <u>P</u>ermutation**
- Evaluation

SIMD Lane Permutation (SLP)

- **Motivation**: Permute home SIMD lanes from their sequentially assigned location to alleviate aligned divergence

* WID: Warp-ID

(TBC)

	Active Threads		Active Threads
W_0	0---	W_0	0---
W_1	4---	W_1	4---
W_2	8---	W_2	8---
W_3	C---	W_3	C---
	(Permuted)	→	(Compacted)

(a) Compaction *as-is*,

(Odd_Even)

	Active Threads		Active Threads
W_0	0---	W_0	04--
W_1	-4--	W_1	8C--
W_2	8---	W_2	----
W_3	-C--	W_3	----
	(Permuted)	→	(Compacted)

(b) XOR warps with *odd* WID by 1

(FLIP_odd)

	Active Threads		Active Threads
W_0	0---	W_0	0--4
W_1	---4	W_1	8--C
W_2	8---	W_2	----
W_3	---C	W_3	----
	(Permuted)	→	(Compacted)

(c) Flip warps with *odd* WID

(ROTATE_all)

	Active Threads		Active Threads
W_0	0---	W_0	048C
W_1	-4--	W_1	----
W_2	--8-	W_2	----
W_3	---C	W_3	----
	(Permuted)	→	(Compacted)

(d) Rotate *all* warps by WID

(ROTATE_odd_1)

	Active Threads		Active Threads
W_0	0---	W_0	04--
W_1	-4--	W_1	8C--
W_2	8---	W_2	----
W_3	-C--	W_3	----
	(Permuted)	→	(Compacted)

(e) Rotate warps with *odd* WID by 1

(Rev_WID)

	Active Threads		Active Threads
W_0	0---	W_0	084C
W_1	--4-	W_1	----
W_2	-8--	W_2	----
W_3	---C	W_3	----
	(Permuted)	→	(Compacted)

(f) XOR all warps by bit-reverse of WID

- ***Odd_Even*: proposed by Fung et al. [MICRO'07] for tuning/optimizing BITONIC sorting application**

Pitfalls of *Randomly* Permuting Lanes

- **Observation**: Many P-branches invoke *up to half of lanes active* after divergence
- Compaction fails unless active threads permuted *exactly to the other half* of (vacant) lanes

Active Threads	
W_0	0123----
W_1	89AB----
W_2	GHIJ----
W_3	OPQR----

Active Threads	
W_0	01--45--
W_1	89--CD--
W_2	GH--KL--
W_3	OP--ST--

Active Threads	
W_0	0-2-4-6-
W_1	8-A-C-E-
W_2	G-I-K-M-
W_3	O-Q-S-U-

Balanced Permutation

- **Observation**: P-branches frequently exhibit highly skewed, predictable concentration of active lanes
- **Intuition** : Distribute active threads across SIMD lanes *evenly* in a *balanced* manner

Lane-ID	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*
	000	001	010	011	100	101	110	111
For W0	0	1	2	3	4	5	6	7
For W1	0	1	2	3	4	5	6	7
For W2	0	1	2	3	4	5	6	7
For W3	0	1	2	3	4	5	6	7
For W4	0	1	2	3	4	5	6	7
For W5	0	1	2	3	4	5	6	7
For W6	0	1	2	3	4	5	6	7
For W7	0	1	2	3	4	5	6	7

Balanced Permutation

- **Observation**: P-branches frequently exhibit highly <u>skewed</u>, <u>predictable</u> concentration of active lanes
- **Intuition** : Distribute active threads across SIMD lanes <u>*evenly*</u> in a <u>*balanced*</u> manner

Lane-ID	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*
	000	001	010	011	100	101	110	111
For W0	0	1	2	3	4	5	6	7
For W1	0	1	2	3	4	5	6	7
For W2	0	1	2	3	4	5	6	7
For W3	0	1	2	3	4	5	6	7
For W4	0							
For W5	0							
For W6	0	1	2	3	4	5	6	7
For W7	0	1	2	3	4	5	6	7

Baseline: assigned lane-IDs based on <u>*sequential*</u> assignment

Balanced Permutation

- **Observation**: P-branches frequently exhibit highly skewed, predictable concentration of active lanes
- **Intuition** : Distribute active threads across SIMD lanes *evenly* in a *balanced* manner

Lane-ID	0	1	2	3	4	5	6	7
	000	001	010	011	100	101	110	111
For W0	0	1	2	3	4	5	6	7
For W1	0	1	2	3	4	5	6	7
For W2	0	1	2	3	4	5	6	7
For W3	0	1	2	3	4	5	6	7
For W4	0	1	2	3	4	5	6	7
For W5	0	1	2	3	4	5	6	7
For W6	0	1	2	3	4	5	6	7
For W7	0	1	2	3	4	5	6	7

Balanced Permutation

- **Observation:** P-branches frequently exhibit highly skewed,
- **Intuiti**

< *Balanced* permutation algorithm >

- Even-ID warps: *XOR* lane-ID by (Warp-ID >> 1)

- Odd-ID warps: *XOR* lane-ID by ~(Warp-ID >> 1)

Lane-ID	0	1	2	3	4	5	6	7
	000	001	010	011	100	101	110	111
For W0	0	1	2	3	4	5	6	7
For W1	0	1	2	3	4	5	6	7
For W2	0	1	2	3	4	5	6	7
For W3	0	1	2	3	4	5	6	7
For W4	0	1	2	3	4	5	6	7
For W5	0	1	2	3	4	5	6	7
For W6	0	1	2	3	4	5	6	7
For W7	0	1	2	3	4	5	6	7

Balanced Permutation

- **Observation:** P-branches frequently exhibit highly skewed,
- **Intuiti**

< *Balanced* permutation algorithm >

- Even-ID warps: *XOR* lane-ID by (Warp-ID >> 1)

- Odd-ID warps: *XOR* lane-ID by ~(Warp-ID >> 1)

	Lane-ID	0	1	2	3	4	5	6	7
		000	001	010	011	100	101	110	111
For W0	XOR-000	0	1	2	3	4	5	6	7
For W1	XOR-111	7	6	5	4	3	2	1	0
For W2	XOR-001	1	0	3	2	5	4	7	6
For W3	XOR-110	6	7	4	5	2	3	0	1
For W4	XOR-010	2	3	0	1	6	7	4	5
For W5	XOR-101	5	4	7	6	1	0	3	2
For W6	XOR-011	3	2	1	0	7	6	5	4
For W7	XOR-100	4	5	6	7	0	1	2	3

Outline

- GPU and SIMD compaction background
- Branch compactability and aligned divergence
- SLP – SIMD Lane Permutation
- **Evaluation**

Simulation Environment

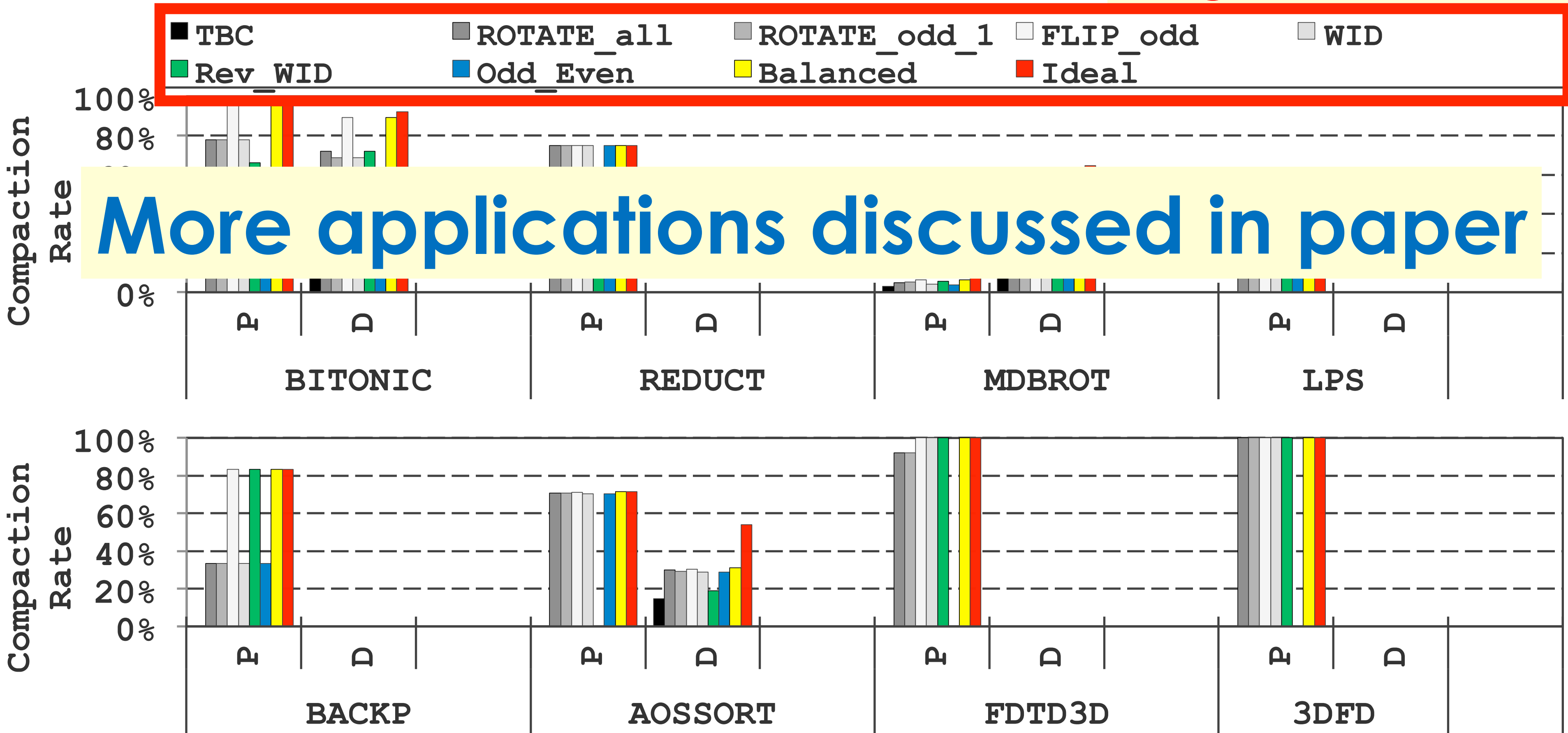
- GPUOcelot (r1865) – based on TBC
  - Branch categorization
  - Compaction rate
  - SIMD lane utilization
- GPGPU-Sim (v2.1) – based on TBC + CAPRI*
  - Performance study
  - Similar to QuadroFX-5800
  - uArch configurations detailed in paper
- Workloads
  - Chosen from CUDA-SDK(v3.0), Rodinia, Parboil, etc

*** "CAPRI: Prediction of Compaction Adequacy for Handling Control-Divergence in GPGPU Architectures", Rhu et al., ISCA'12**

Compaction Rate

- **Definition**: Fraction of *compactable* paths among *all* paths generated by divergent branches
- **SLP**: significantly improves *P*-branch compaction rate

Higher is better

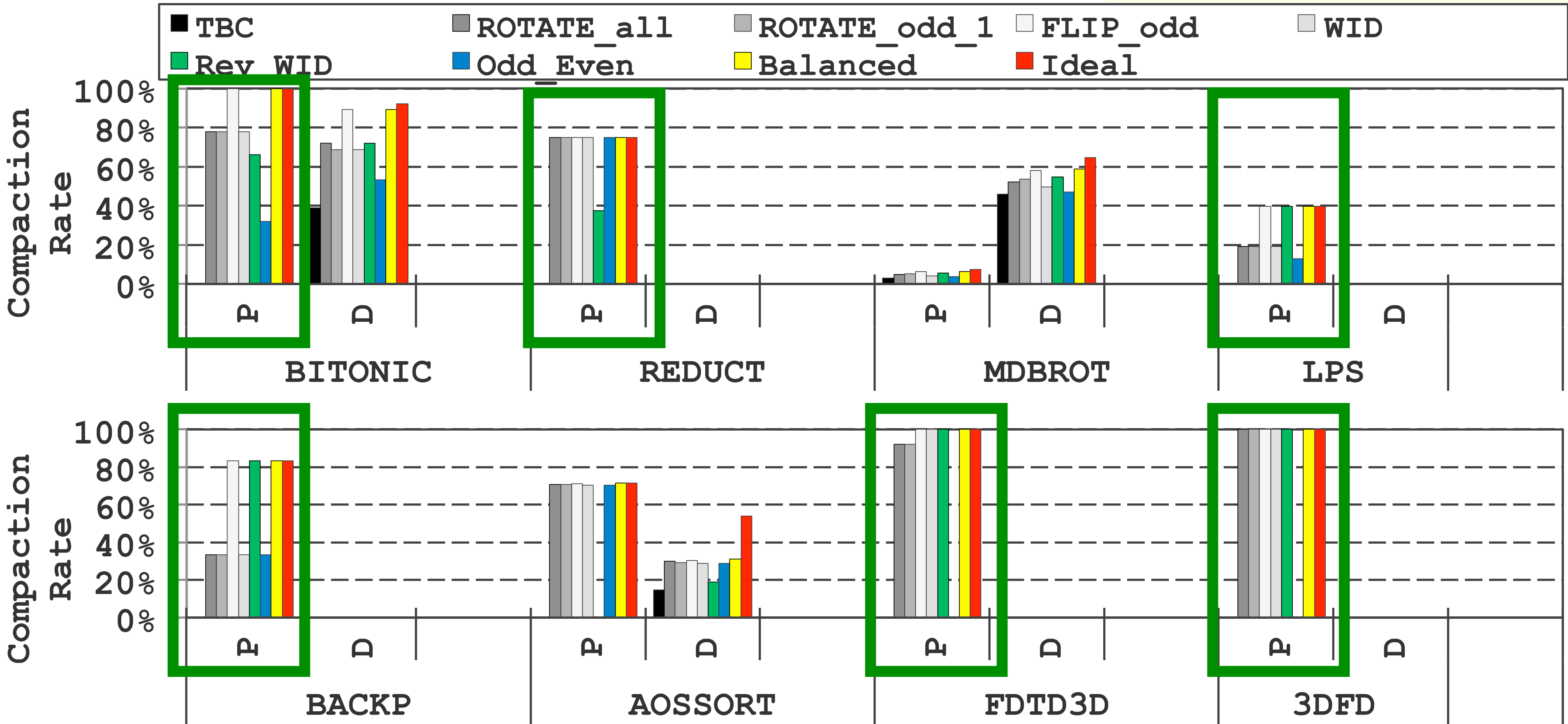


More applications discussed in paper

Compaction Rate

- **Definition**: Fraction of *compactable* paths among *all* paths generated by divergent branches
- **SLP**: significantly improves *P*-branch compaction rate

Higher is better

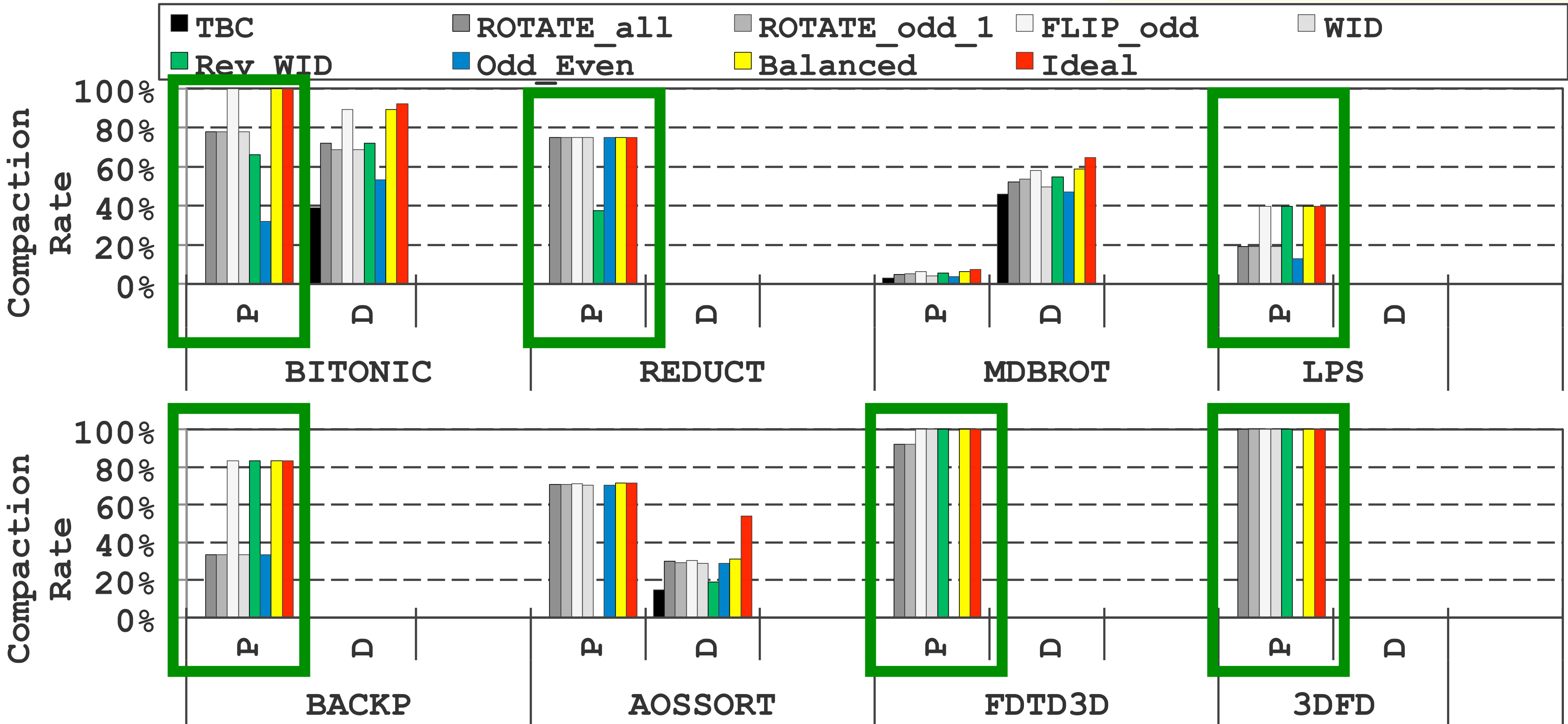


Compaction Rate

- De
- SL

- **P-branches**
  - **TBC**: average 3.2%
  - **Odd_Even**: average 28.9%
  - **Balanced**: average 71.5%
  - **Ideal**: average 72.7%

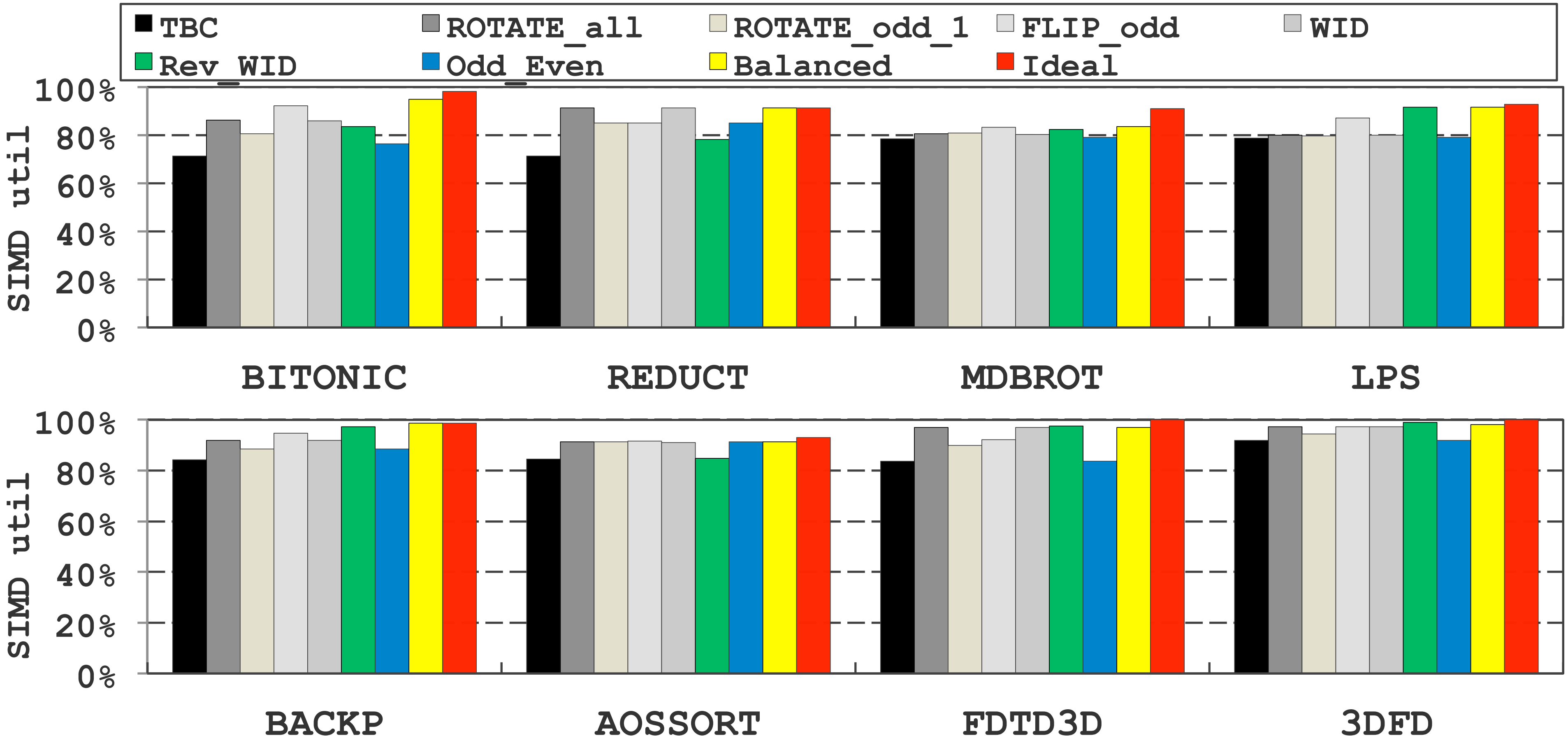
- **D-branches**
  - **TBC**: average 42.5%
  - **Odd_Even**: average 45.2%
  - **Balanced**: average 59.3%
  - **Ideal**: average 68.5%



Average SIMD Lane Utilization

- **Definition**: average number of SIMD lanes *actually* executing and committing results
- Compaction rate doesn't tell '*how effectively*' warps are reduced

Higher is better



Average SIMD Lane Utilization

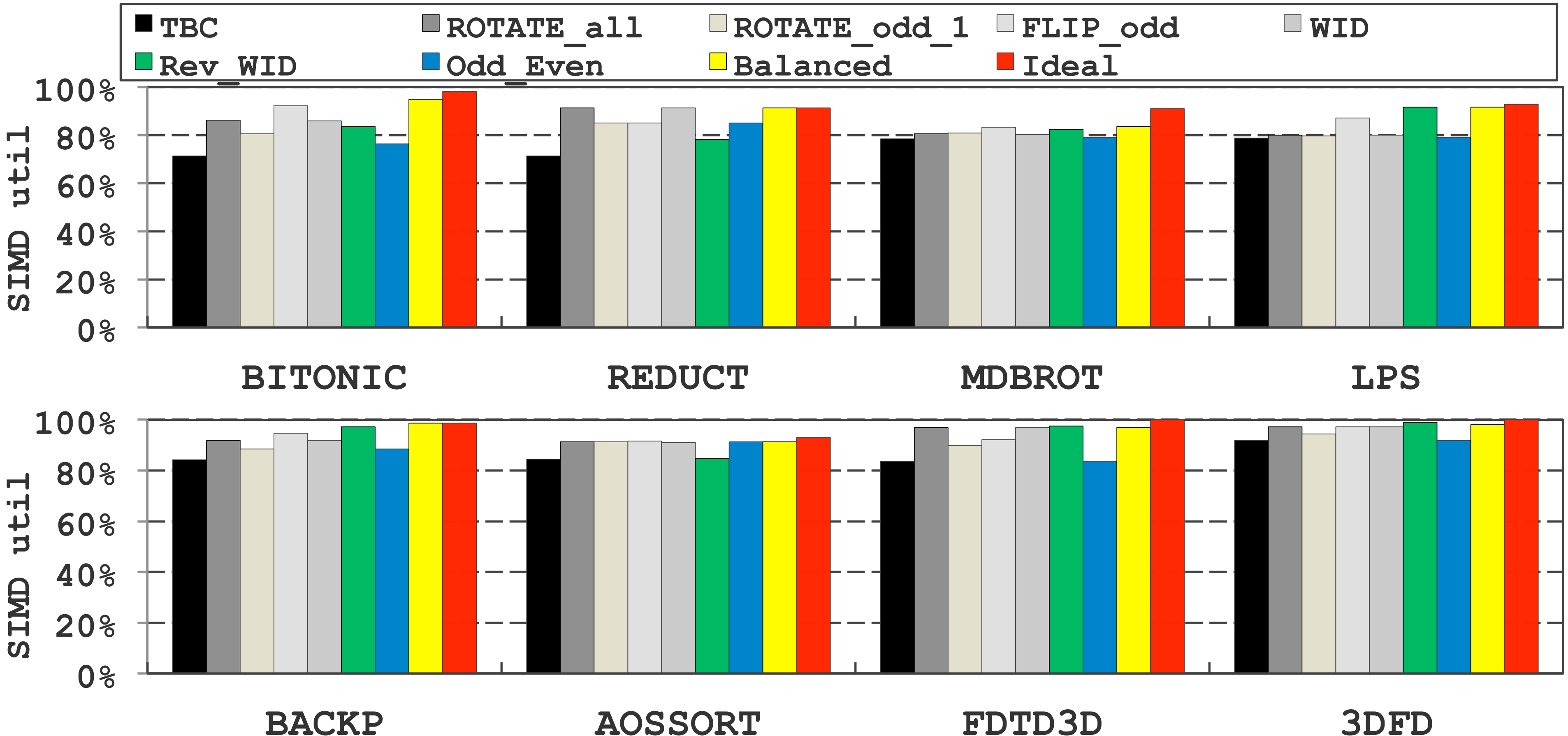
- **Definition**: average number of SIMD lanes *actually* executing
-

- TBC: lowest utilization due to un-compacted P-branches

- SLP:

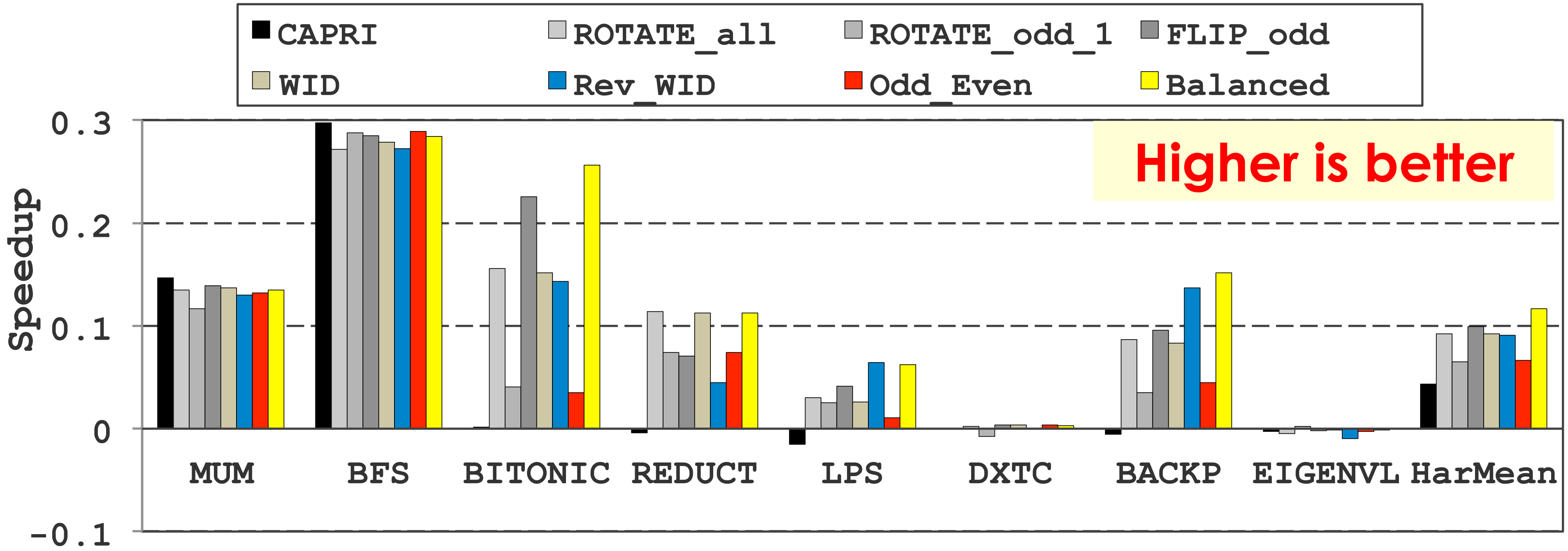
Odd_Even: average 2.1% (max 18%) increase over TBC

Balanced: average 7.1% (max 34%) increase over TBC



Speedup

- **Baseline**: No compaction
- TBC+CAPRI only effective for 'irregular' applications
- SLP widens the range of applications that benefit from compaction techniques
  - **7.1**% (max 34%) improvements on top of TBC+CAPRI



Conclusions

- Effectiveness of previous compactions limited to highly irregular applications
  - Only effective for D-branches
  - Non-compactable P-branches
- SLP enables instrumenting such lost opportunities
  - Requires simple re-mapping of thread-IDs to Lane-ID
  - Enhance compaction rate and SIMD utilization
- Throughput improvements
  - **7.1%** (max 34%) improvements on top of TBC+CAPRI