OPTIMIZING SOFTWARE DIRECTED INSTRUCTION REPLICATION FOR GPU ERROR DETECTION

Abdulrahman Mahmoud (University of Illinois)
Siva Hari, Mike Sullivan, Tim Tsai, Steve Keckler (NVIDIA)

MOTIVATION

GPU Reliability

HPC and safety-critical systems are key drivers of GPU resilience

GPUs need to be resilient enough to meet HPC and autonomous safety requirements

Resilience can be improved through:

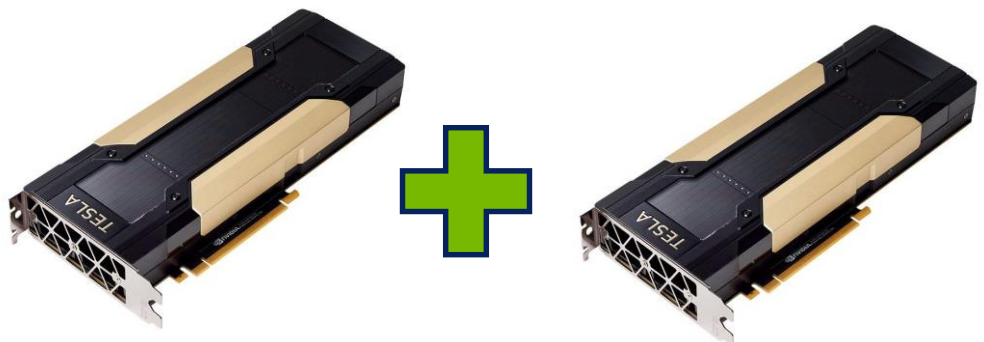
- Hardware techniques: E.g., more ECC/parity protected structures
- Software techniques: E.g., redundant execution

CURRENT SOLUTIONS

GPU Reliability

Hardware Dual Modular Redundancy

- >2x overhead
- High area and power cost
- Costly ($$)



ECC/Parity

- Only protect major storage structures

Hardware techniques either have high overhead or limited coverage

SOFTWARE ERROR DETECTION APPROACHES

Resilience through duplication at different levels of granularity

Program/algorithm level duplication

Needs programmer involvement, not applicable to non-deterministic programs

Thread/warp level duplication

Automatic, but needs spare threads/warps per block

Instruction level duplication

Automatic, applicable to all programs

Current Focus

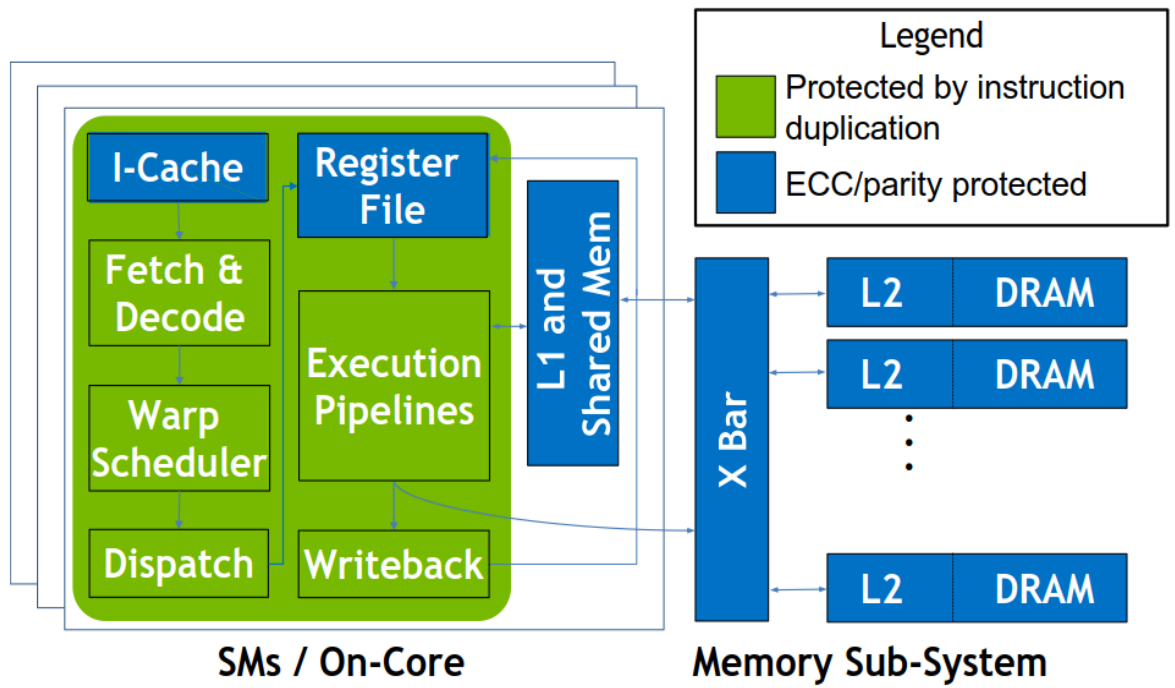
SPHERE OF REPLICATION

Instruction duplication protects datapath components

Protection from transient errors

Protects components not covered by standard ECC/parity

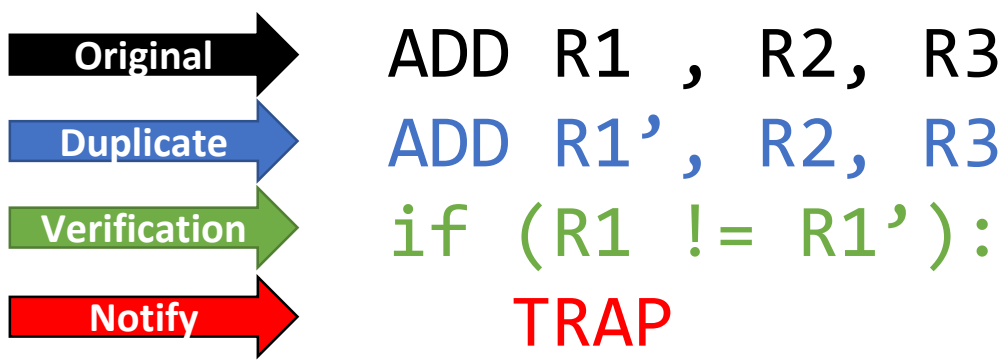
Focus on protecting the datapath



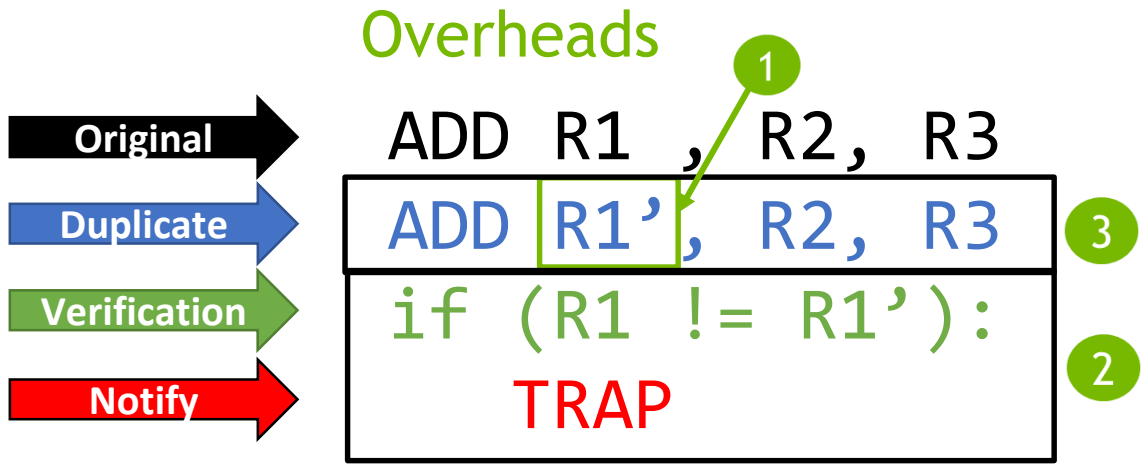
INSTRUCTION LEVEL DUPLICATION

High level overview

1. Instruction is duplicated in software (at assembly level)
2. Check for correctness (verification)
3. Trap on error (notification)



INSTRUCTION LEVEL DUPLICATION



Components of overheads:

1. Increase in register requirement → reduces GPU occupancy
2. Verification and notification instructions
3. Duplicate instruction overhead

SInRG: Software-managed Instruction Replication for GPUs

CONTRIBUTIONS

SInRG: Software-managed Instruction Replication for GPUs

First to analyze instruction level duplication on GPUs

Software and hardware optimizations

- Analyze GPU-specific trade-off between register usage and dynamic instructions
- Signature-based checking: reduce notification instructions
- Hardware support to remove notifications and/or verification instructions

Overhead reductions

- Software optimizations: 69% (using CPU optimizations) down to 36%, on average
- Hardware support: Average overhead reduced to 30%

SInRG: OUTLINE

Motivation, Background

Contributions

Software Optimizations and Performance Analysis

Hardware-Software Optimizations and Performance Analysis

Coverage Analysis

Conclusion

SInRG BASE TECHNIQUE 1: SRIV

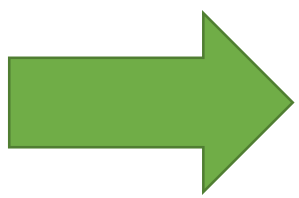
Single Register space, Immediate Verification

Simple instruction level duplication scheme

Immediate verification of original and duplicate instruction results

Example:

```
ADD R1, R2, R3
```



```
ADD R1 , R2, R3
ADD R1', R2, R3
if (R1 != R1'):
    TRAP
```

SInRG BASE TECHNIQUE 2: DRDV

Double Register space, Delayed Verification

Duplicate data flow chain, verify end of chain (a CPU-inspired technique)

Chains end at store, control, atomic, and non-deterministic operations

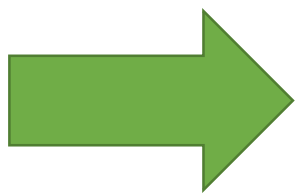
Duplicate instructions operate on shadow register space

2x virtual register usage

Verify inputs of non-duplicated instructions

Example:

```
IADD R3, R1, R2
FADD R6, R4, R5
ST   R6, [R3]
```



```
IADD R3, R1, R2
IADD R3', R1', R2'
FADD R6, R4, R5
FADD R6', R4', R5'
if  (R3 != R3')
    TRAP
if  (R6 != R6')
    TRAP
ST   R6, [R3]
```

TRADEOFFS

SRIV vs. DRDV

SRIV

More instructions

Fewer Registers

No Error Masking potential

DRDV

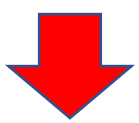
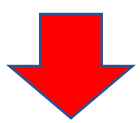
Fewer instructions

More registers

Error Masking potential

More registers can affect total warp occupancy

Registers are a limited GPU Resource

 **Regs/thread →**  **Threads in parallel →**  **Occupancy**

SOFTWARE OPTIMIZATIONS

Tackling overheads

1. Duplication may use more registers or add many verification instructions

   Study two methods that trade-off register usage for checking frequency

2. Verification and notification instructions

   FastSig: Signature-based checking

   Remove redundant checks

3. Duplicate instruction overhead

   Do not duplicate Moves (they provide inherent redundancy)

FastSig: SIGNATURE-BASED CHECKING

Reducing the checking overhead of duplication

1. Initialize an error signature
2. Update error signature register
3. Check signature register at end of function

$$S1 = (R1 \text{ XOR } R1') \text{ OR } S1$$

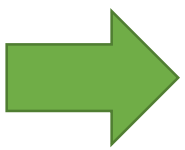
Maps to a single, high-throughput instruction on NVIDIA GPUs

LOP3 S1, S1, R1, R1’

FastSig - SRIV

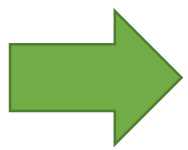
Applying signature-based checks to SRIV algorithm

```
…
ADD R3, R1, R2
…
```



SRIV

```
ADD  R3, R1, R2
ADD  R3', R1, R2
if (R3 != R3'):
    TRAP
```



FastSig – SRIV

```
MOV  S1, 0x0   #reset
…
ADD  R3, R1, R2
ADD  R3', R1, R2
LOP3 S1, S1, R3, R3'
…
if (S1 != 0):
    TRAP
```

Benefits:

- Remove control flow
- Reduces number of instructions
- Allows better reordering

LOP3: S1 = (R3 XOR R3') OR S1

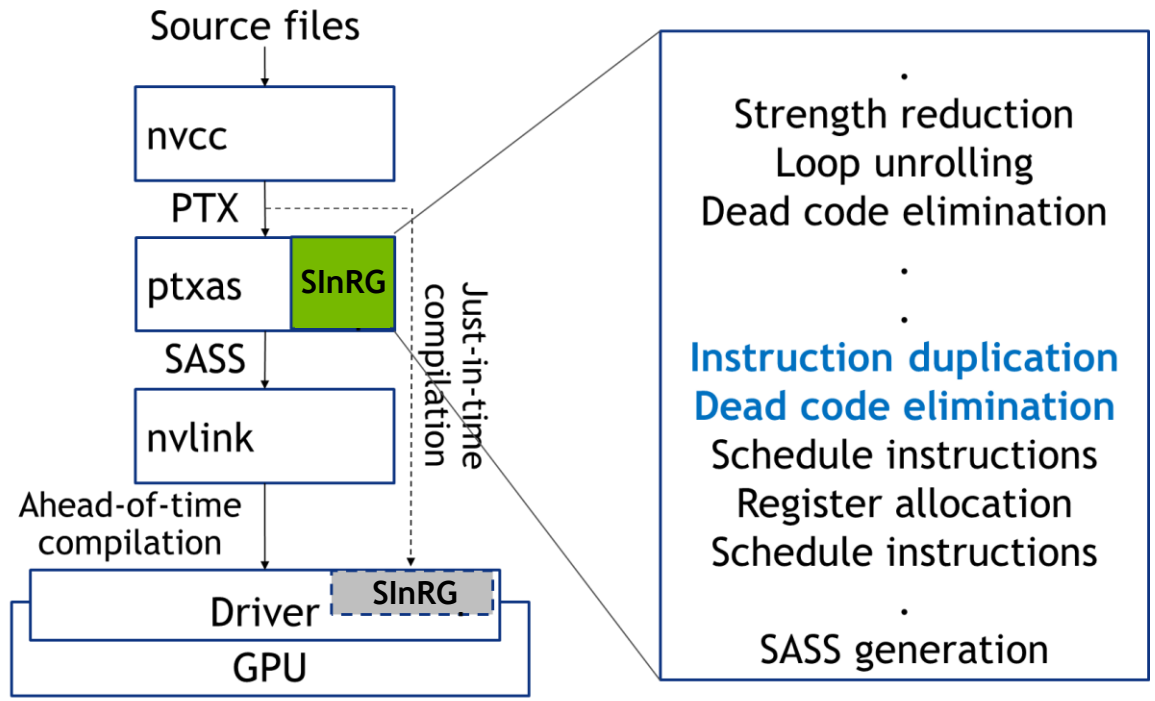
IMPLEMENTATION

Backend compiler

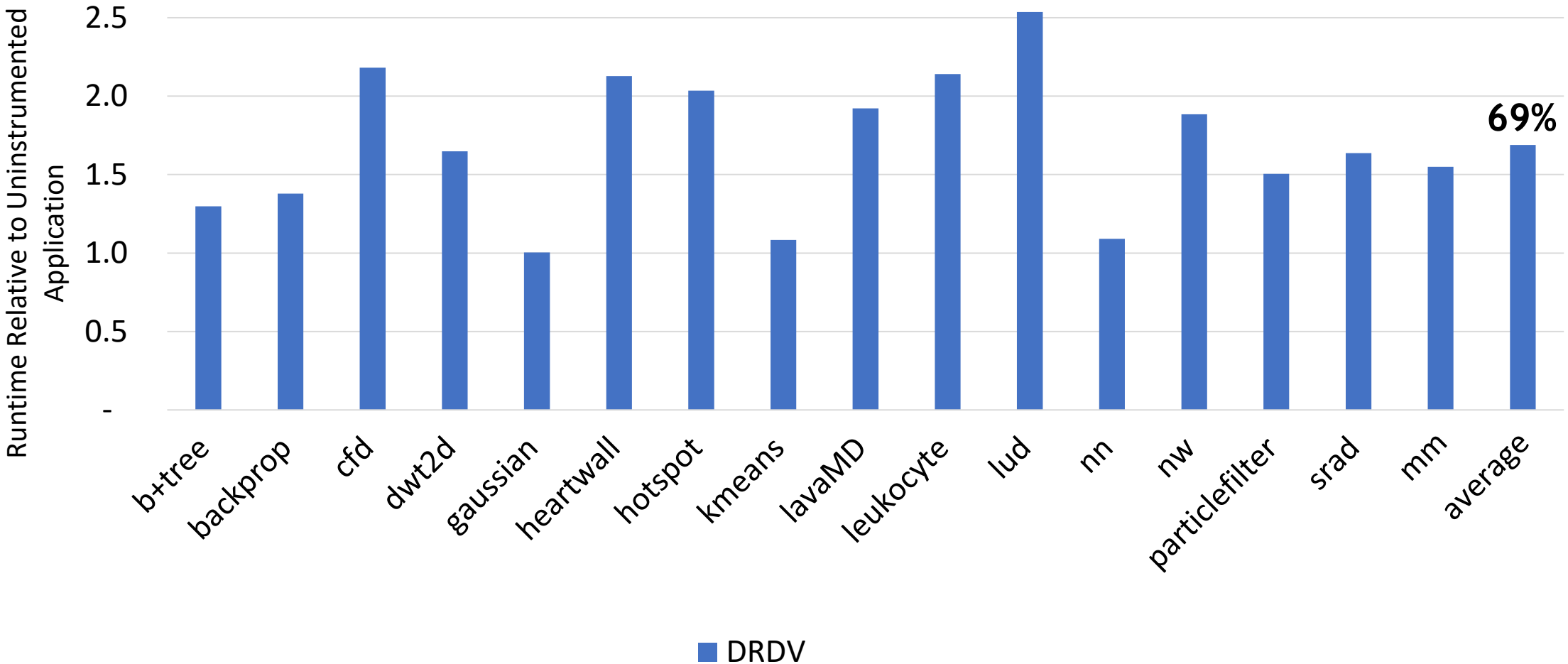
NVIDIA’s production compiler backend

Implemented in compiler IR

Leverages backend optimizations

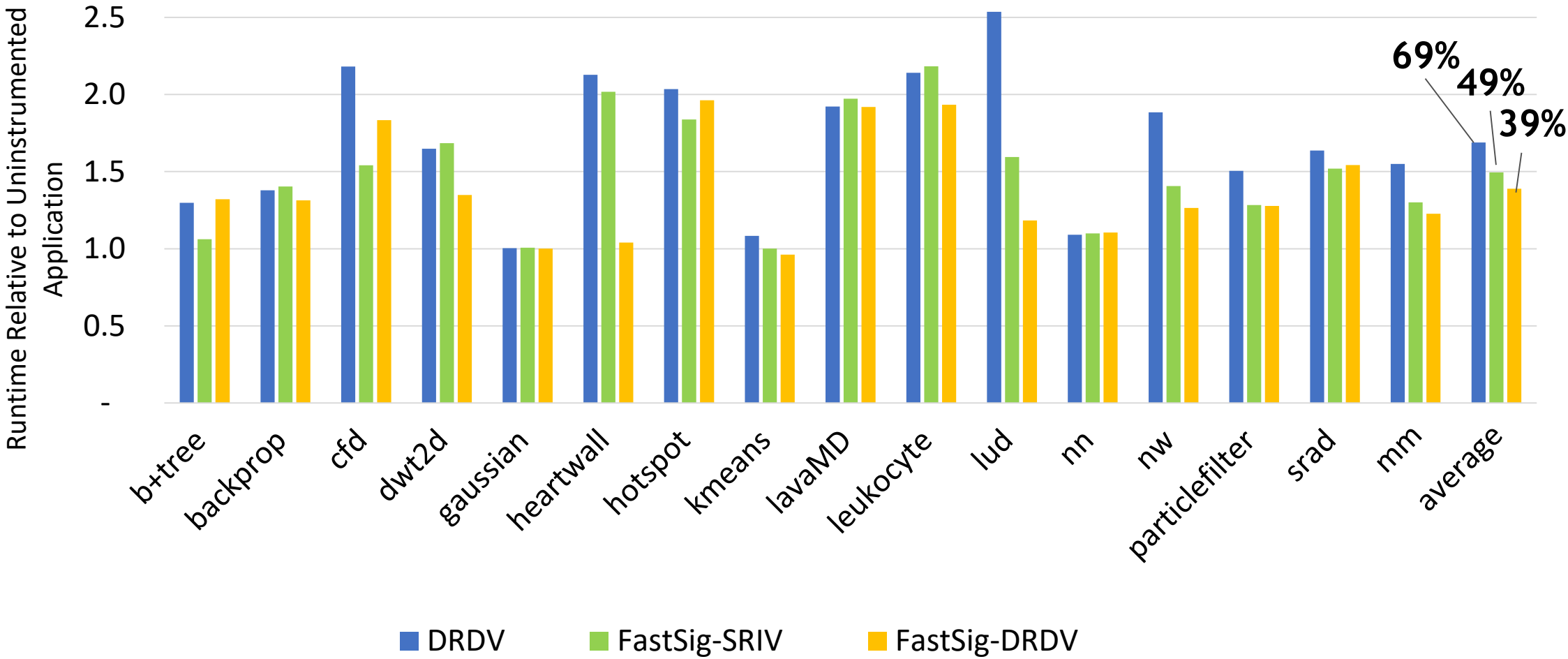


SW PERFORMANCE OVERHEAD (TITAN XP)

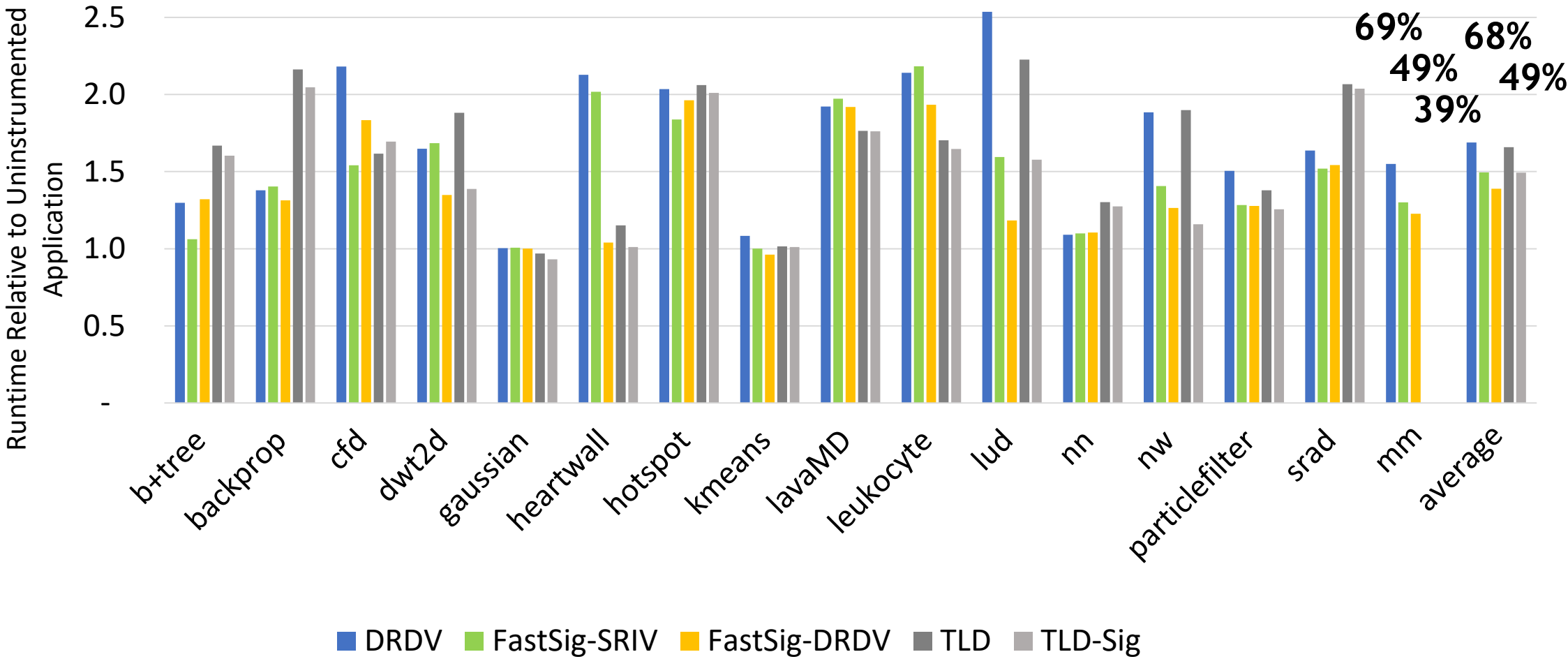


*SRIV overheads are high. Not shown here.

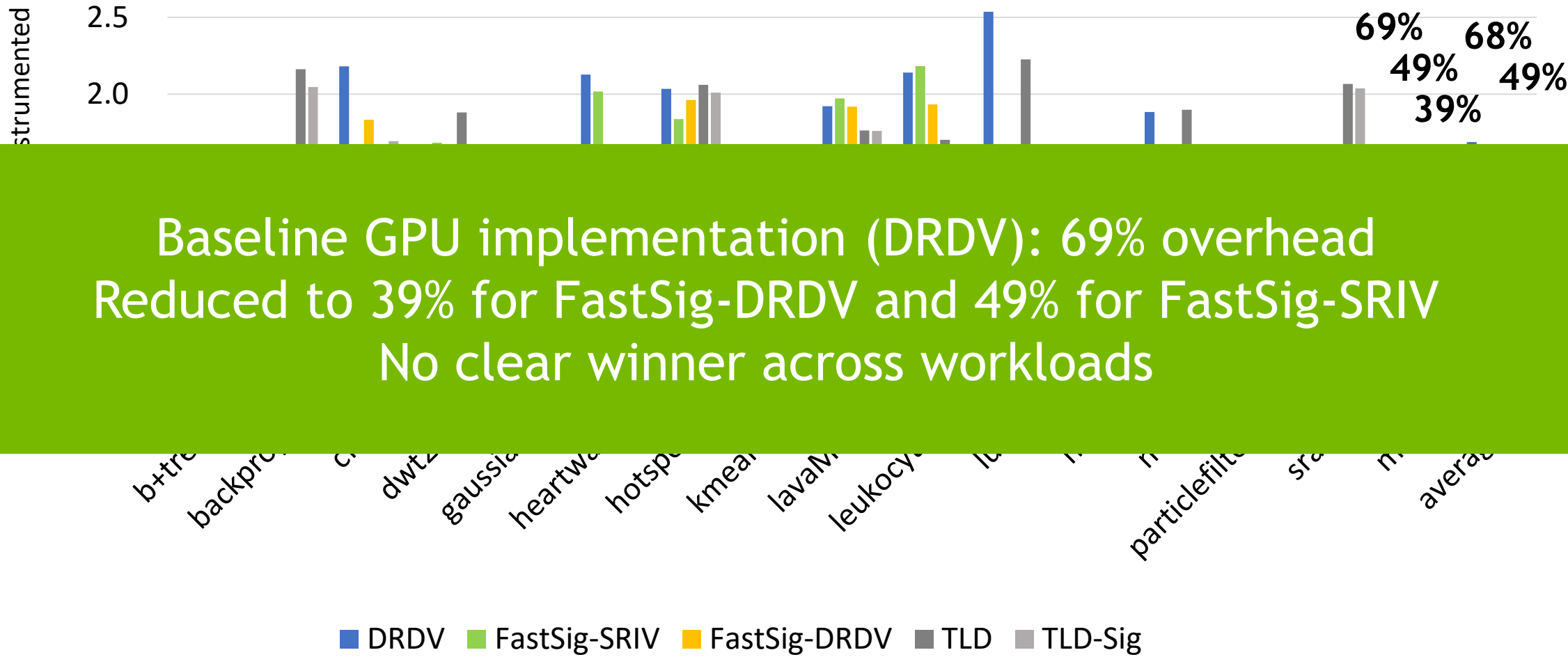
SW PERFORMANCE OVERHEAD (TITAN XP)



SW PERFORMANCE OVERHEAD (TITAN XP)



SW PERFORMANCE OVERHEAD (TITAN XP)



Baseline GPU implementation (DRDV): 69% overhead
Reduced to 39% for FastSig-DRDV and 49% for FastSig-SRIV
No clear winner across workloads

SInRG: OUTLINE

Motivation, Background

Contributions

Software Optimizations and Performance Analysis

Hardware-Software Optimizations and Performance Analysis

Coverage

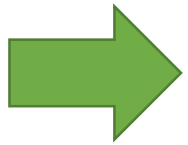
Conclusion

HARDWARE EXTENSIONS

Low-cost ISA extensions

HW-Notify: Hardware instruction to compare-then-exception

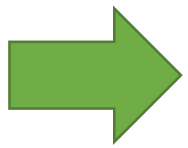
```
ADD R3, R1, R2
```



```
ADD R4, R1, R2
ADD R3, R1, R2
LOP.xor.ex R4, R3
```

HW-Sig: Signature-based checking in hardware

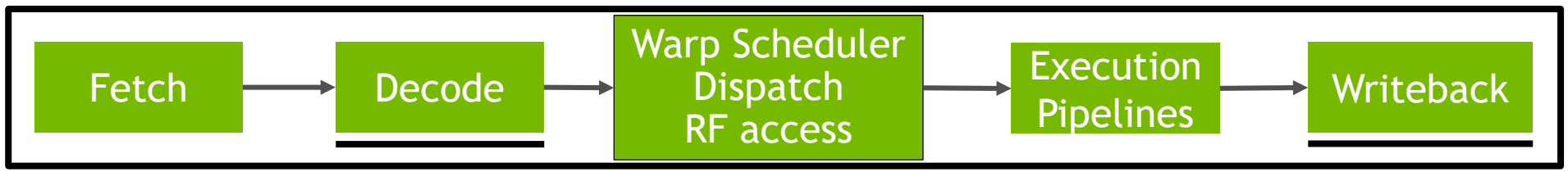
```
ADD R3, R1, R2
```



```
ADD.sig RZ, R1, R2
ADD.sig R3, R1, R2
```

AREA OVERHEAD

Hardware Implementation



Decoder changes:

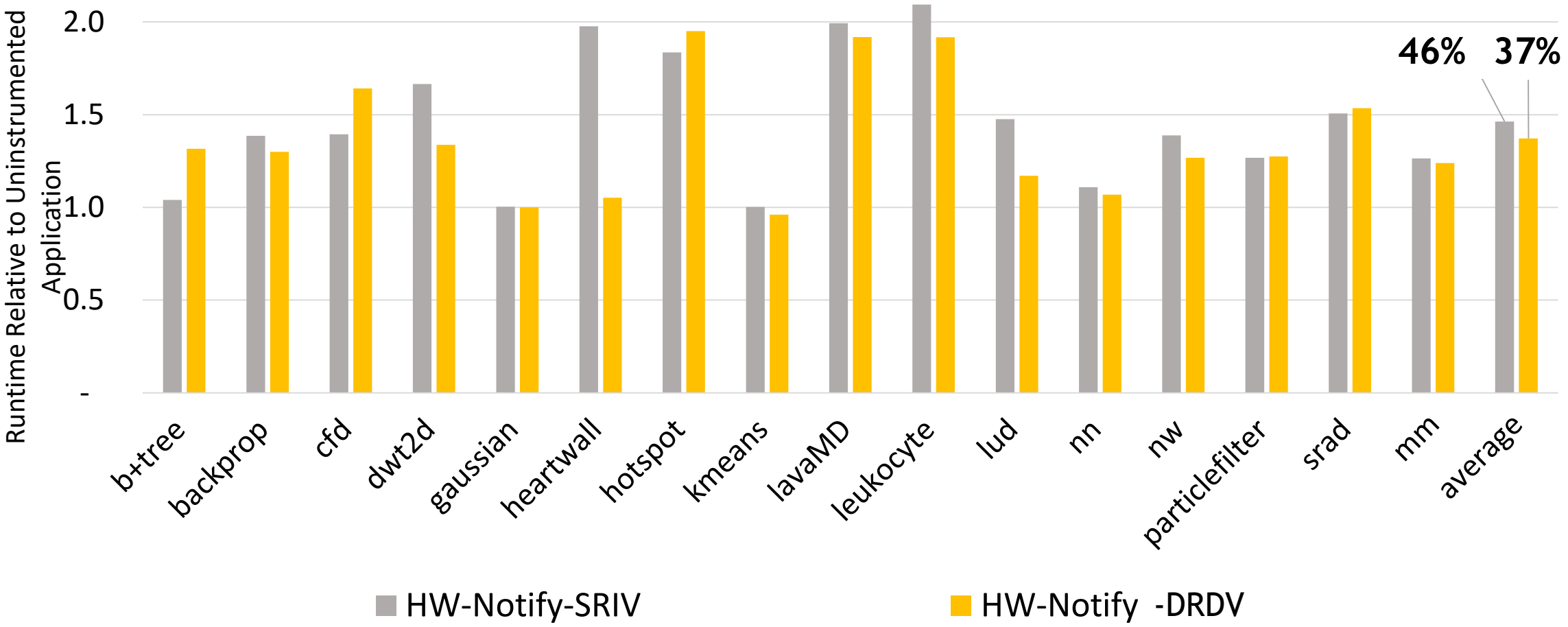
- New opcodes for HW-Notify and HW-Sig
- Additional decode bit for HW-Sig to identify original from duplicate

Writeback:

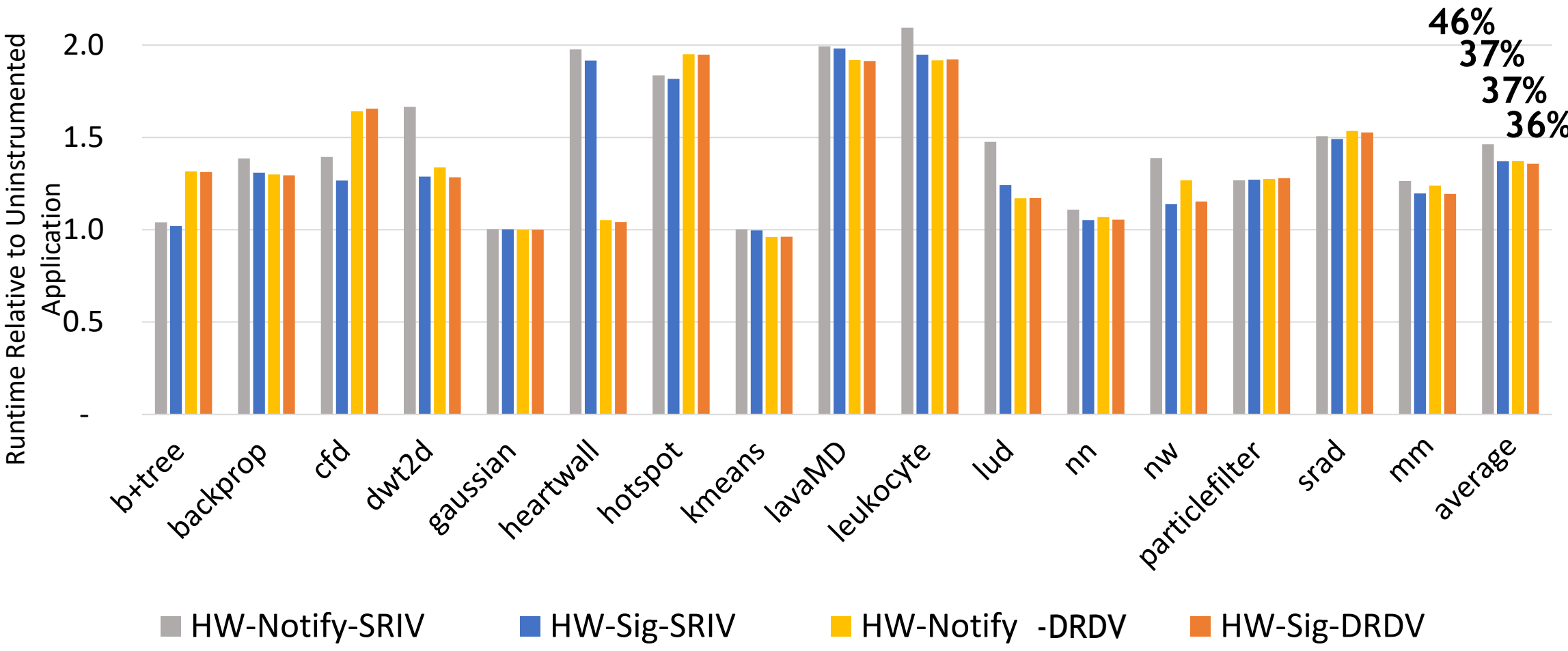
- Signature register per lane
- Logical operation for signature update

Synthesized total area cost: less than an adder

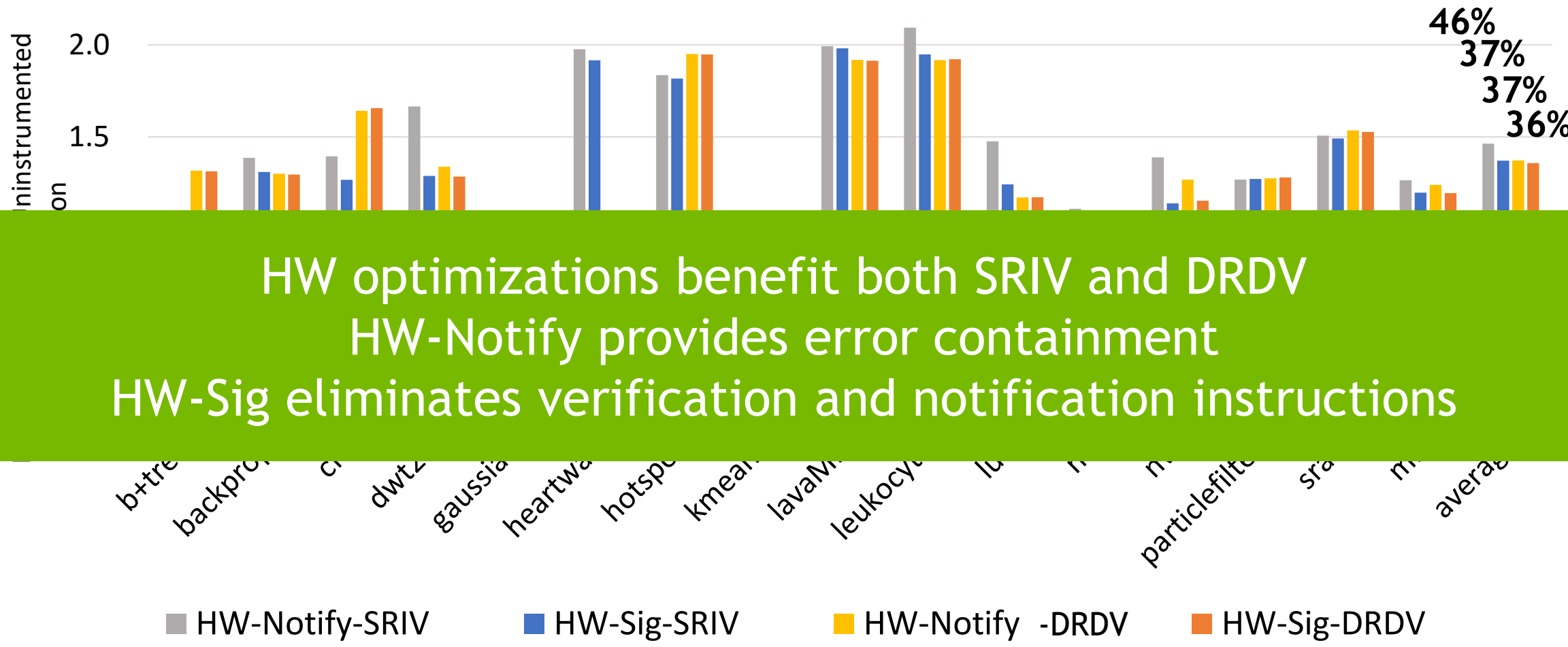
HARDWARE PERFORMANCE OVERHEAD



HARDWARE PERFORMANCE OVERHEAD



HARDWARE PERFORMANCE OVERHEAD



HW optimizations benefit both SRIV and DRDV
HW-Notify provides error containment
HW-Sig eliminates verification and notification instructions

AUTOMATED DUPLICATION SELECTION

Workload-specific duplication technique

DRDV and SRIV perform best for specific GPU kernel, depending on resources

Designed an automated supervised learning model to select best SInRG technique

Based on static information provided by compiler

Reduces average runtime for FastSig-*, HW-Notify, and HW-Sig to **36%, 34%, 30%,** respectively

COVERAGE

1. Instruction coverage

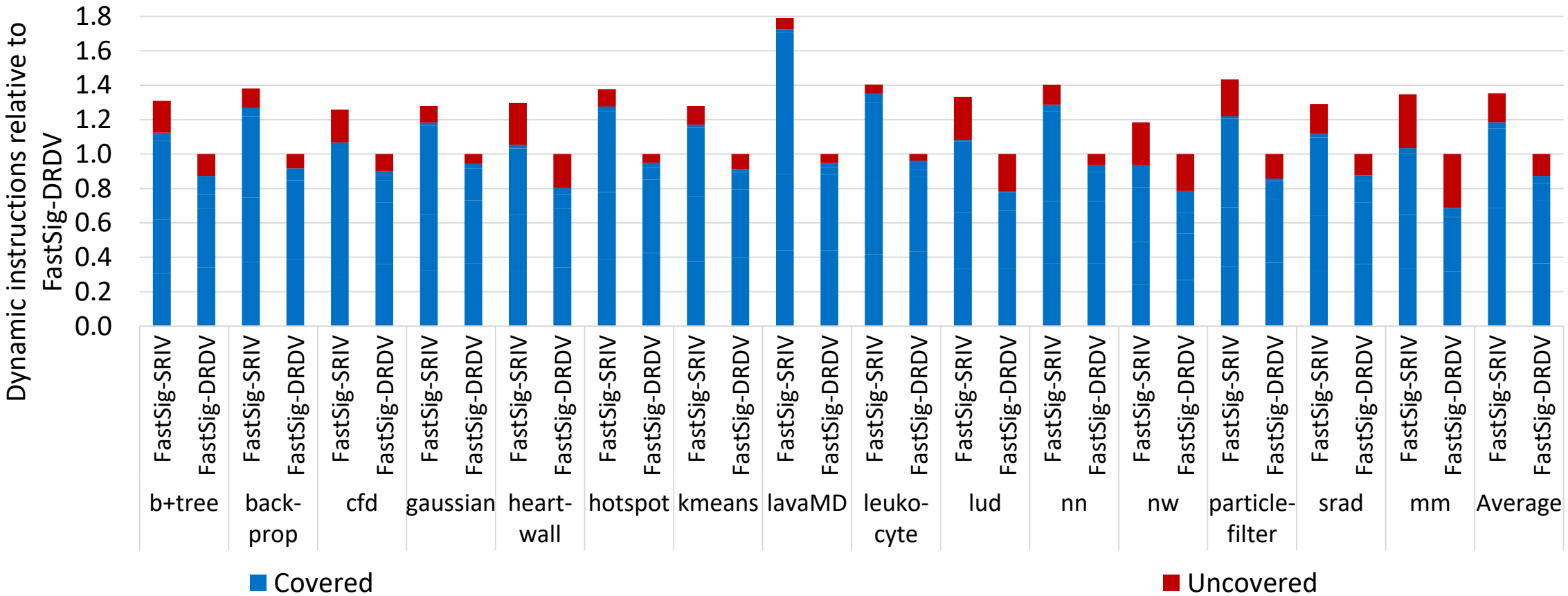
2. Error Injections

3. High particle beam experiment

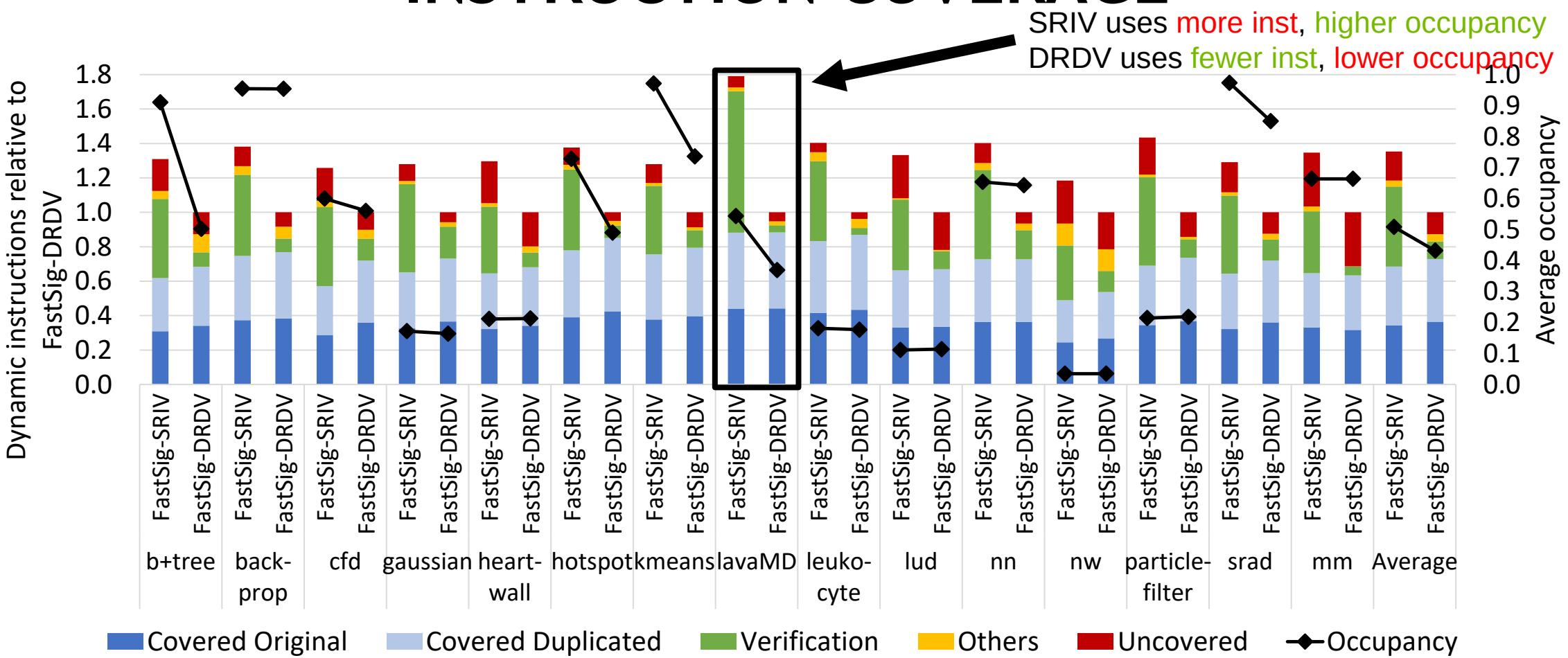
COVERAGE

1. Instruction coverage
2. Error Injections
3. High particle beam experiment

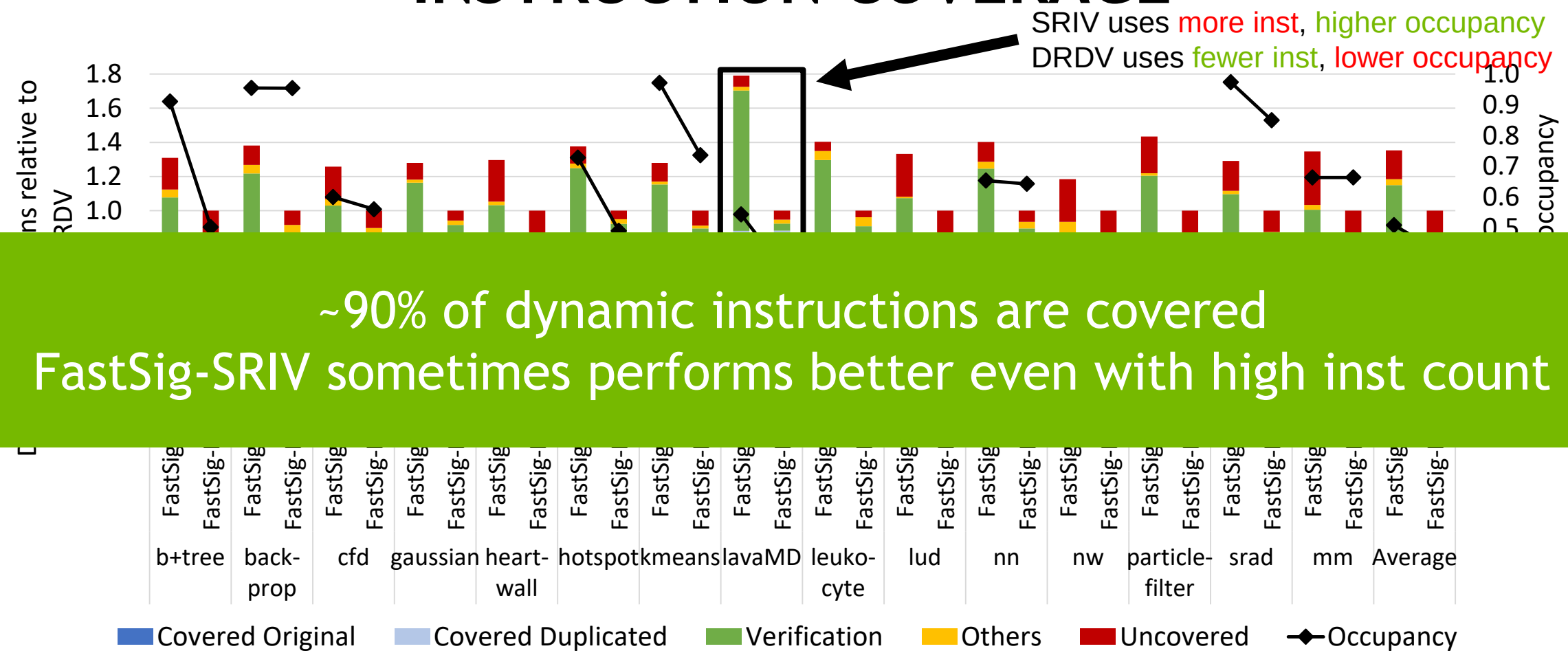
INSTRUCTION COVERAGE



INSTRUCTION COVERAGE

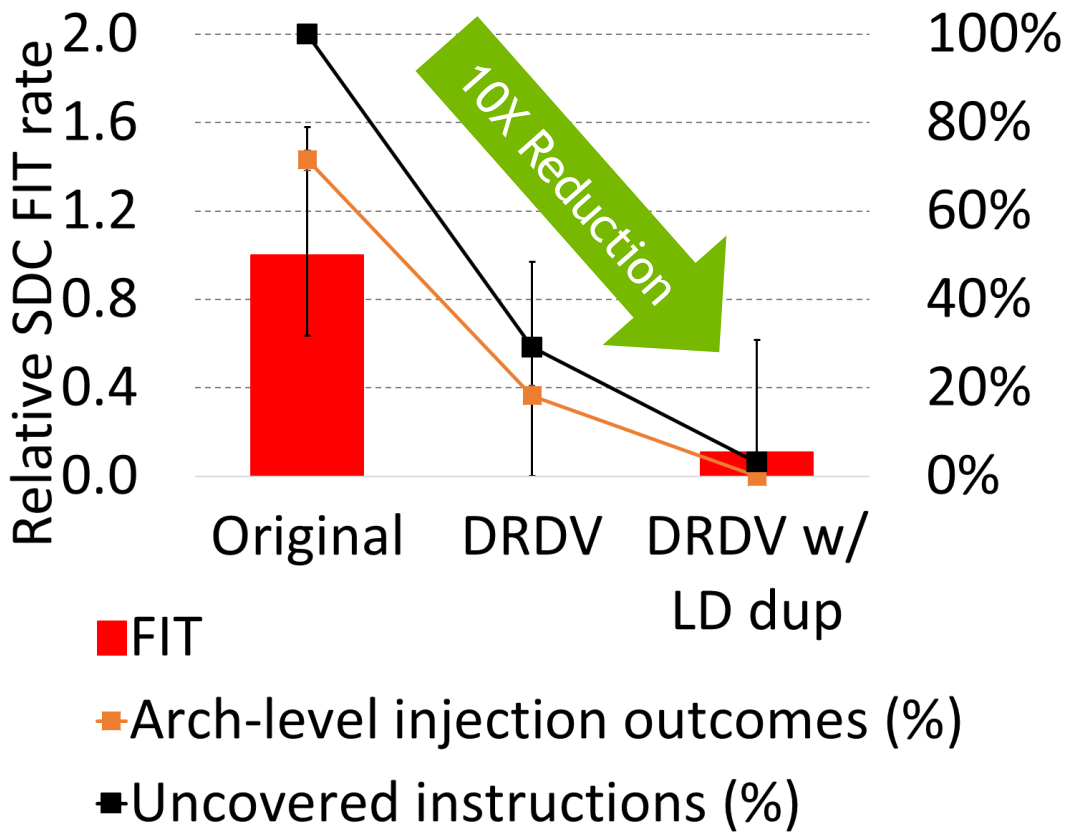


INSTRUCTION COVERAGE

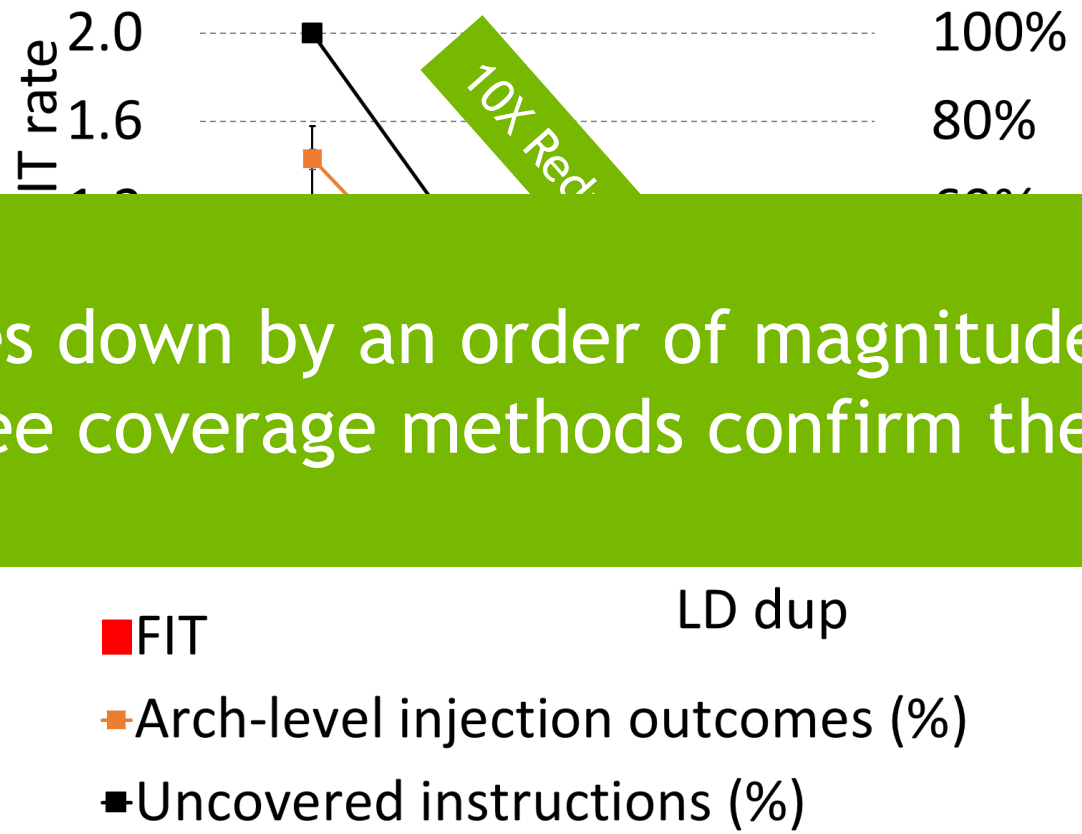


~90% of dynamic instructions are covered
FastSig-SRIV sometimes performs better even with high inst count

ACCELERATED BEAM EXPERIMENT



ACCELERATED BEAM EXPERIMENT



SDC rate goes down by an order of magnitude with SInRG
All three coverage methods confirm the trend

CONCLUSION

SInRG: a family of SW-managed GPU instruction duplication schemes

First analysis of instruction level duplication on GPUs

Software and hardware optimizations to reduce performance overheads

Performance overhead reduction:

- Baseline GPU implementation shows 69% overhead → reduced to just 30% overhead
- Very high coverage (observed ~10X FIT rate improvement)

Future research:

- Selective duplication of instructions
- Workload-specific insights