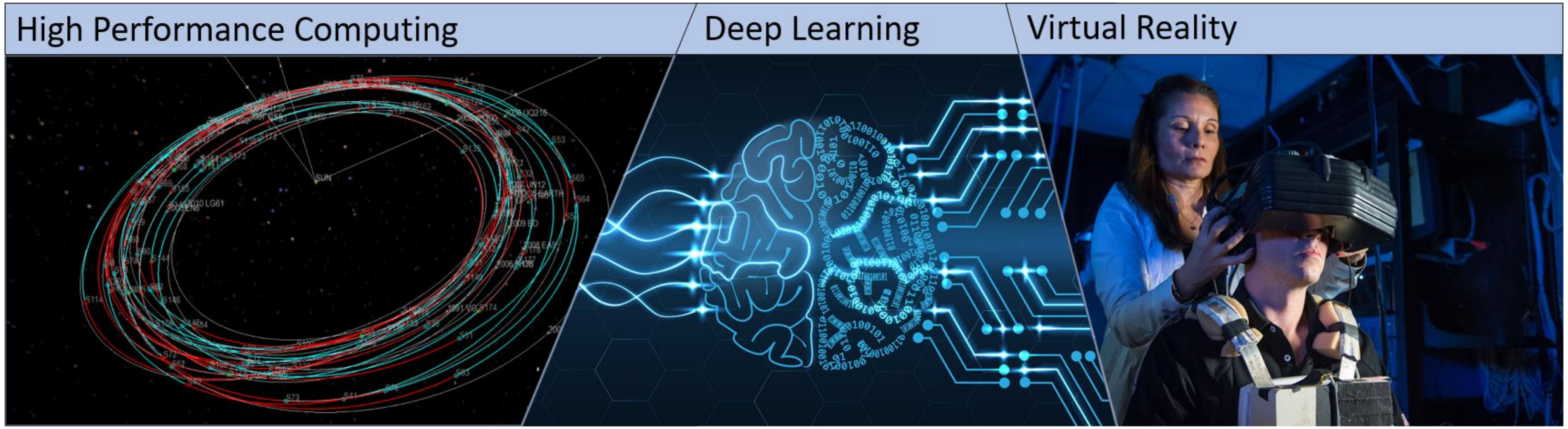
William & Mary
Chartered 1693

Fault Site Pruning for Practical Reliability Analysis of GPGPU Applications

Bin Nie, **Lishan Yang**, Adwait Jog, and Evgenia Smirni
College of William & Mary



William & Mary

Soft Errors

- Soft errors: most commonly observed errors
  - Single bit flips

- ***Masked*** output
  - Correct answer.
- ***Silent Data Corruption (SDC)*** output
  - Wrong answer.
- ***Other*** output
  - Crash, hang, …

Reliability Research: Fault Injection

- Fault injection method
  - Injecting **single-bit errors** into different locations (**fault sites**) in applications
- Ground truth: ***huge unreachable exhaustive fault sites!***

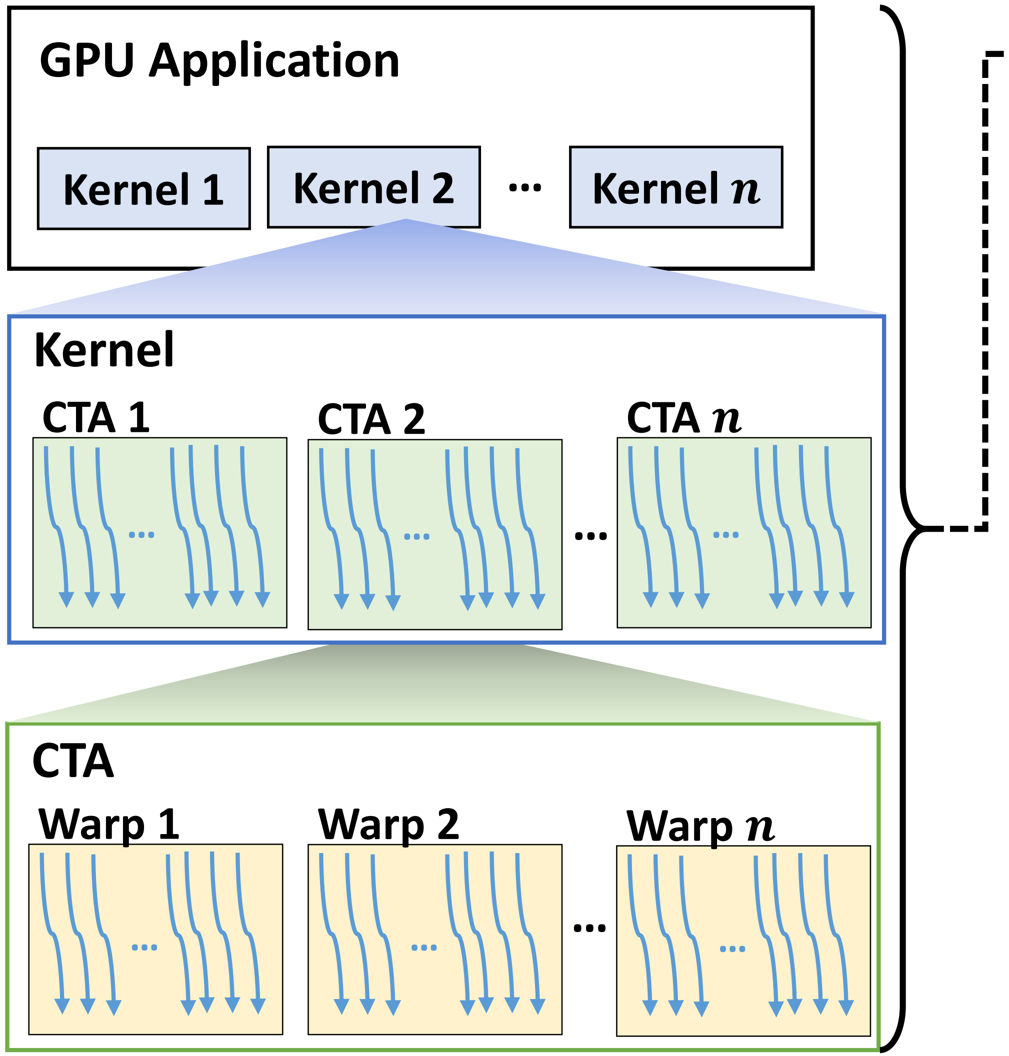
GEMM:

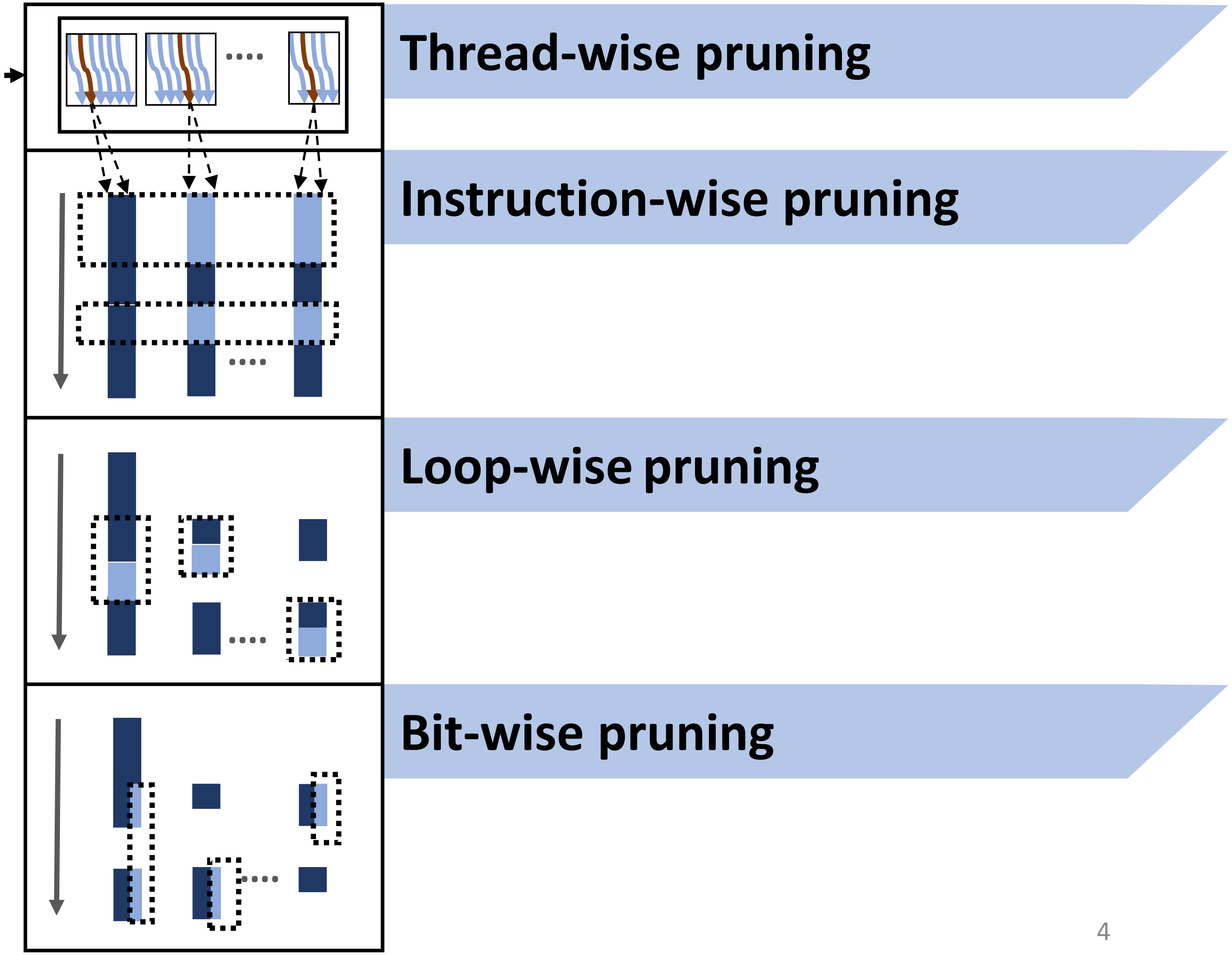
$16384 \text{ threads} \times 1305 \text{ insns/thd} \times 29.6 \text{bits/insn} = 6.23 \times 10^8$ fault sites!

- Closest to ground truth (Baseline):
  - Random sampling based on statistics

Confidence Interval: 99.8%, Error Margin: 1.26%
→ 60K fault sites

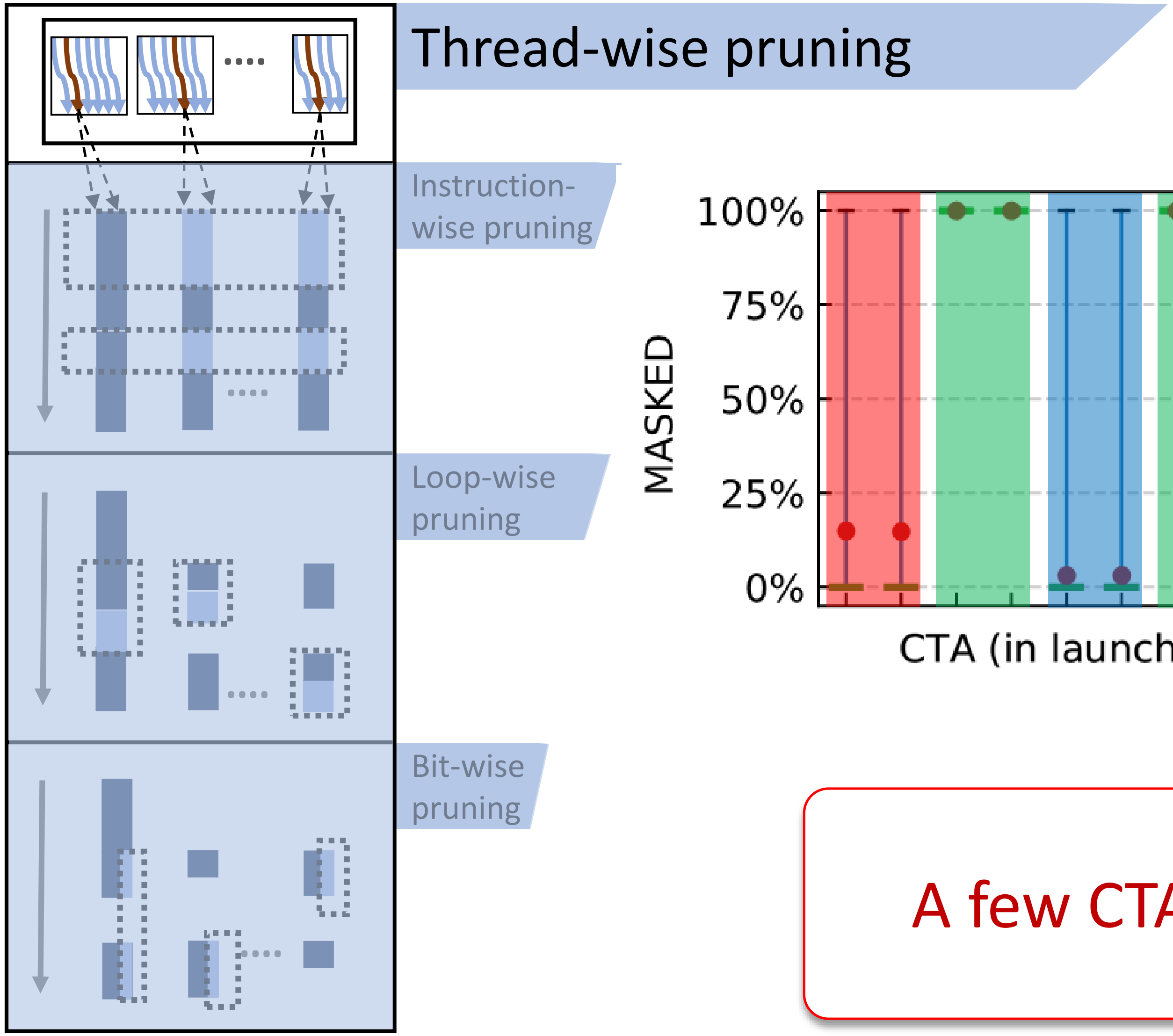
GPU Architecture → Progressive Fault Sites Pruning



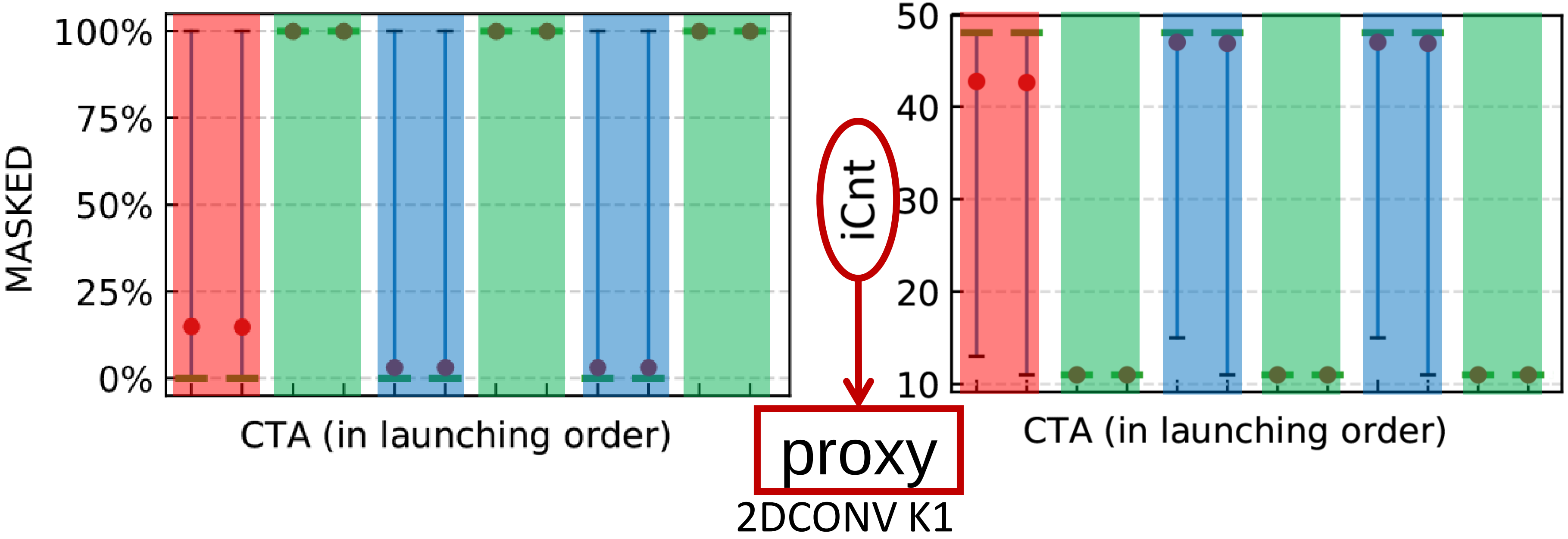


William & Mary

Progressive Fault Sites Pruning



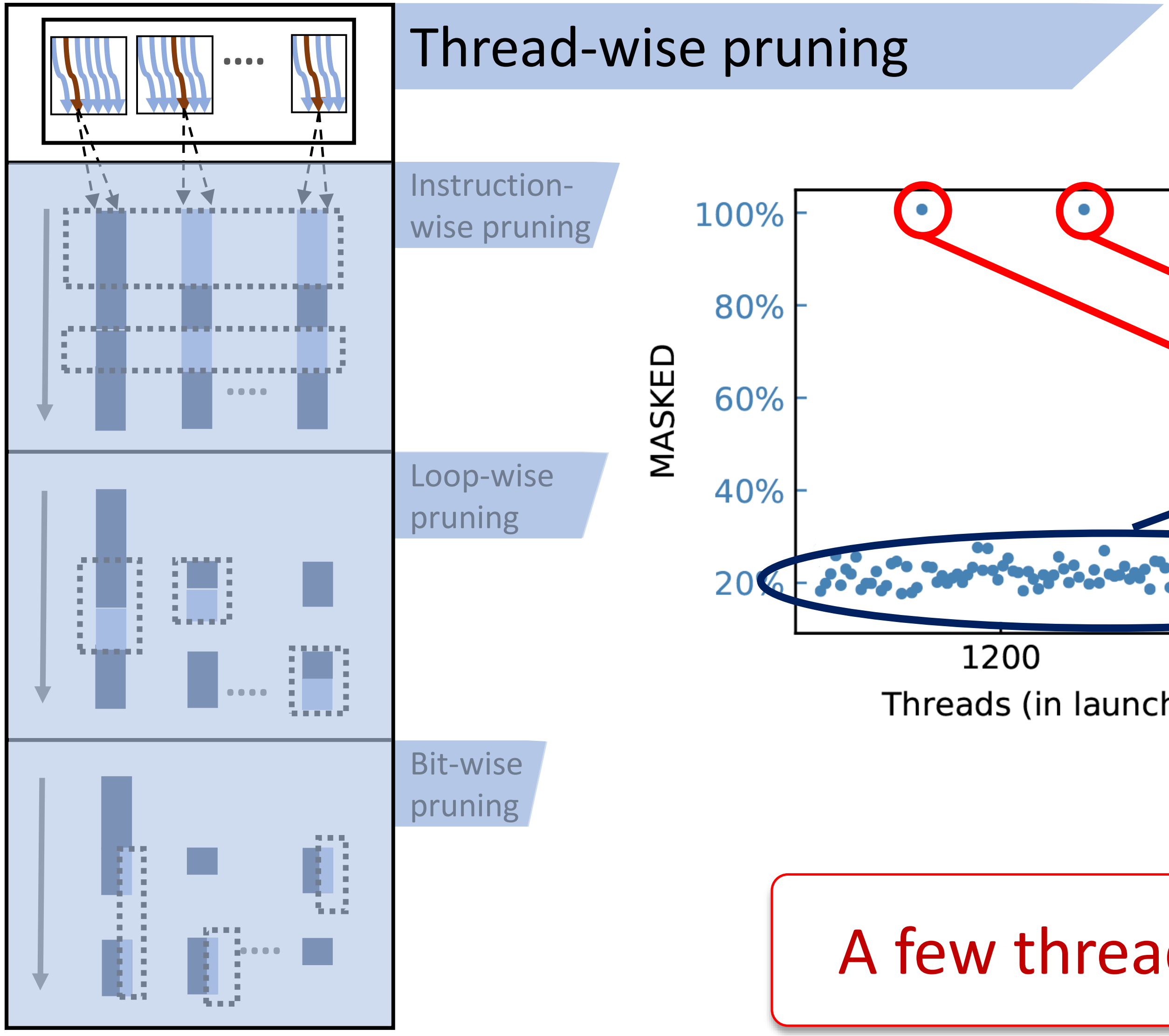
- CTA-level



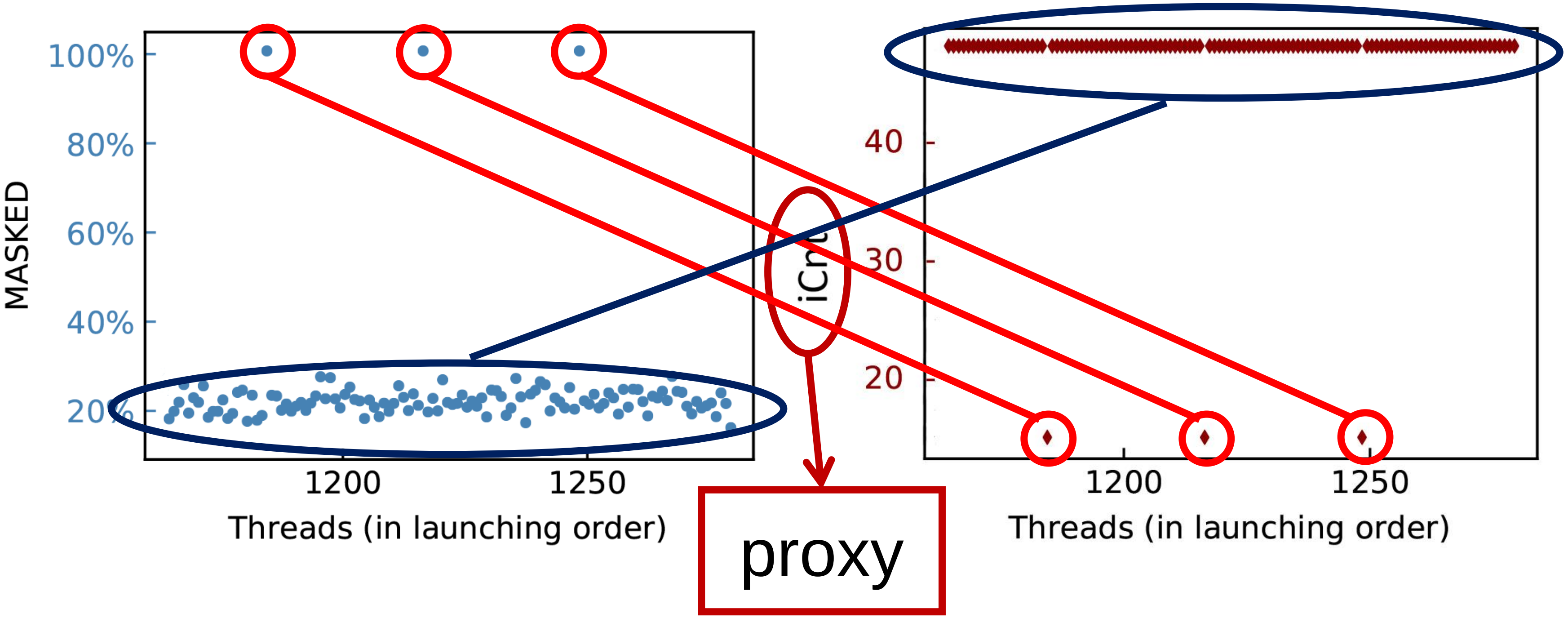
A few CTAs are enough; the rest are pruned

WILLIAM & MARY

Progressive Fault Sites Pruning



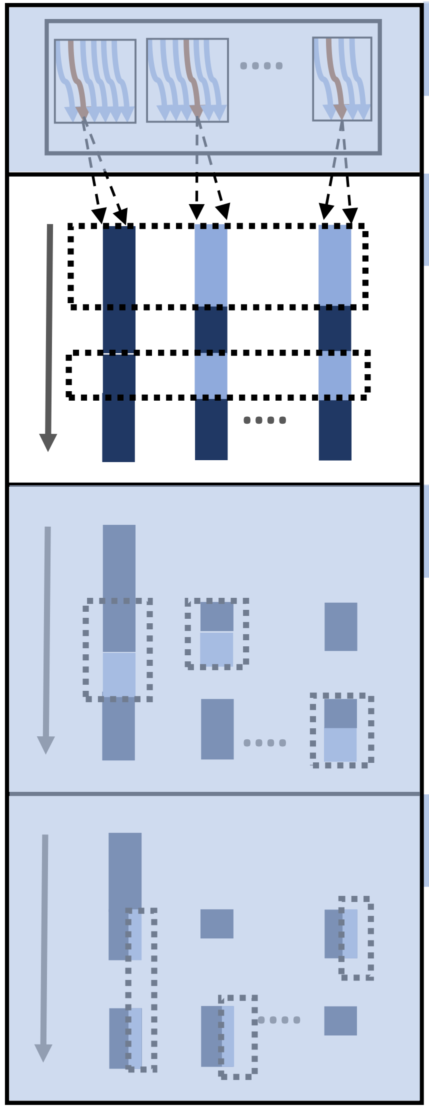
- CTA-level → Thread-level



2DCONV K1, CTA(0-1-0)

A few threads are enough; the rest are pruned

Progressive Fault Sites Pruning



Thread-wise pruning

Instruction-wise pruning

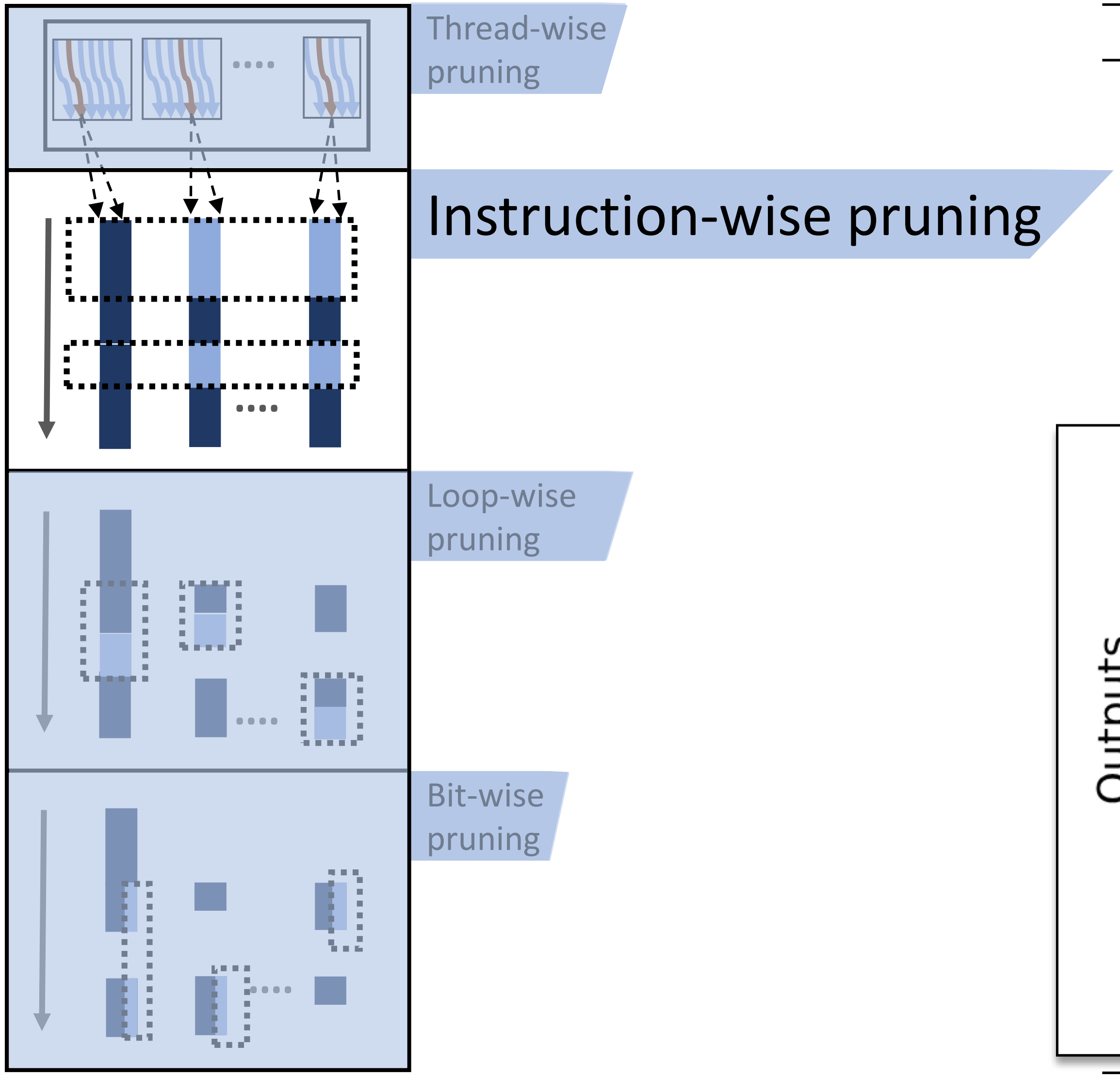
Loop-wise pruning

Bit-wise pruning

PathFinder K1

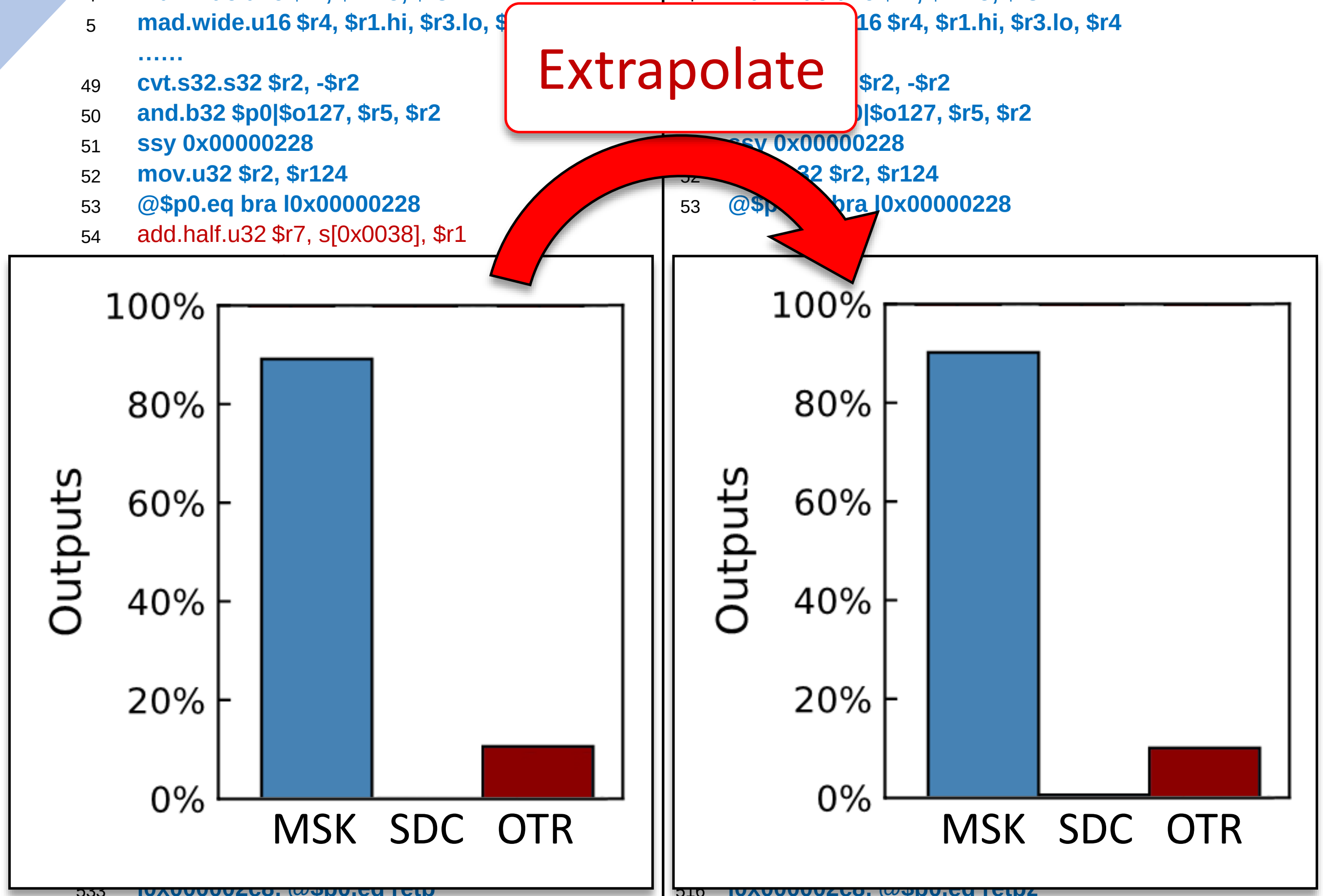
	Thread “a” ($iCnt = 533$)		Thread “b” ($iCnt = 516$)
1	`shl.u32 $r3, s[0x0010], 0x00000001`	1	`shl.u32 $r3, s[0x0010], 0x00000001`
2	`cvt.u32.u16 $r1, %ctaid.x`	2	`cvt.u32.u16 $r1, %ctaid.x`
3	`add.u32 $r3, -$r3, 0x00000100`	3	`add.u32 $r3, -$r3, 0x00000100`
4	`mul.wide.u16 $r4, $r1.lo, $r3.hi`	4	`mul.wide.u16 $r4, $r1.lo, $r3.hi`
5	`mad.wide.u16 $r4, $r1.hi, $r3.lo, $r4`	5	`mad.wide.u16 $r4, $r1.hi, $r3.lo, $r4`
	……		……
49	`cvt.s32.s32 $r2, -$r2`	49	`cvt.s32.s32 $r2, -$r2`
50	`and.b32 $p0\|$o127, $r5, $r2`	50	`and.b32 $p0\|$o127, $r5, $r2`
51	`ssy 0x00000228`	51	`ssy 0x00000228`
52	`mov.u32 $r2, $r124`	52	`mov.u32 $r2, $r124`
53	`@$p0.eq bra l0x00000228`	53	`@$p0.eq bra l0x00000228`
54	`add.half.u32 $r7, s[0x0038], $r1`		
55	`mov.half.u32 $r2, s[0x0030]`		
56	`mul.wide.u16 $r8, $r2.lo, $r7.hi`		
57	`mad.wide.u16 $r8, $r2.hi, $r7.lo, $r8`		
58	`shl.u32 $r8, $r8, 0x00000010`		
	……		
66	`min.s32 $r7, s[$ofs2+0x0040], $r8`		
67	`ld.global.u32 $r2, [$r2]`		
68	`add.u32 $r2, $r2, $r7`		
69	`mov.u32 s[$ofs3+0x0440], $r2`		
70	`mov.u32 $r2, 0x00000001`		
71	`l0x00000228: nop`	54	`l0x00000228: nop`
72	`bar.sync 0x00000000`	55	`bar.sync 0x00000000`
73	`set.eq.s32.s32 $p0/$o127, $r6, $r1`	56	`set.eq.s32.s32 $p0/$o127, $r6, $r1`
74	`@$p0.ne bra l0x000002b8`	57	`@$p0.ne bra l0x000002b8`
75	`set.ne.s32.s32 $p1/$r1, $r2, $r124`	58	`set.ne.s32.s32 $p1/$r1, $r2, $r124`
	……		……
529	`set.eq.s32.s32 $p0/$o127, $r6, $r1`	512	`set.eq.s32.s32 $p0/$o127, $r6, $r1`
530	`@$p0.ne bra l0x000002b8`	513	`@$p0.ne bra l0x000002b8`
531	`l0x000002b8: set.ne.s32.s32 $p0/$o127, $r2, $r124`	514	`l0x000002b8: set.ne.s32.s32 $p0/$o127, $r2, $r124`
532	`bra l0x000002c8`	515	`bra l0x000002c8`
533	`l0x000002c8: @$p0.eq retp`	516	`l0x000002c8: @$p0.eq retpz`

Progressive Fault Sites Pruning

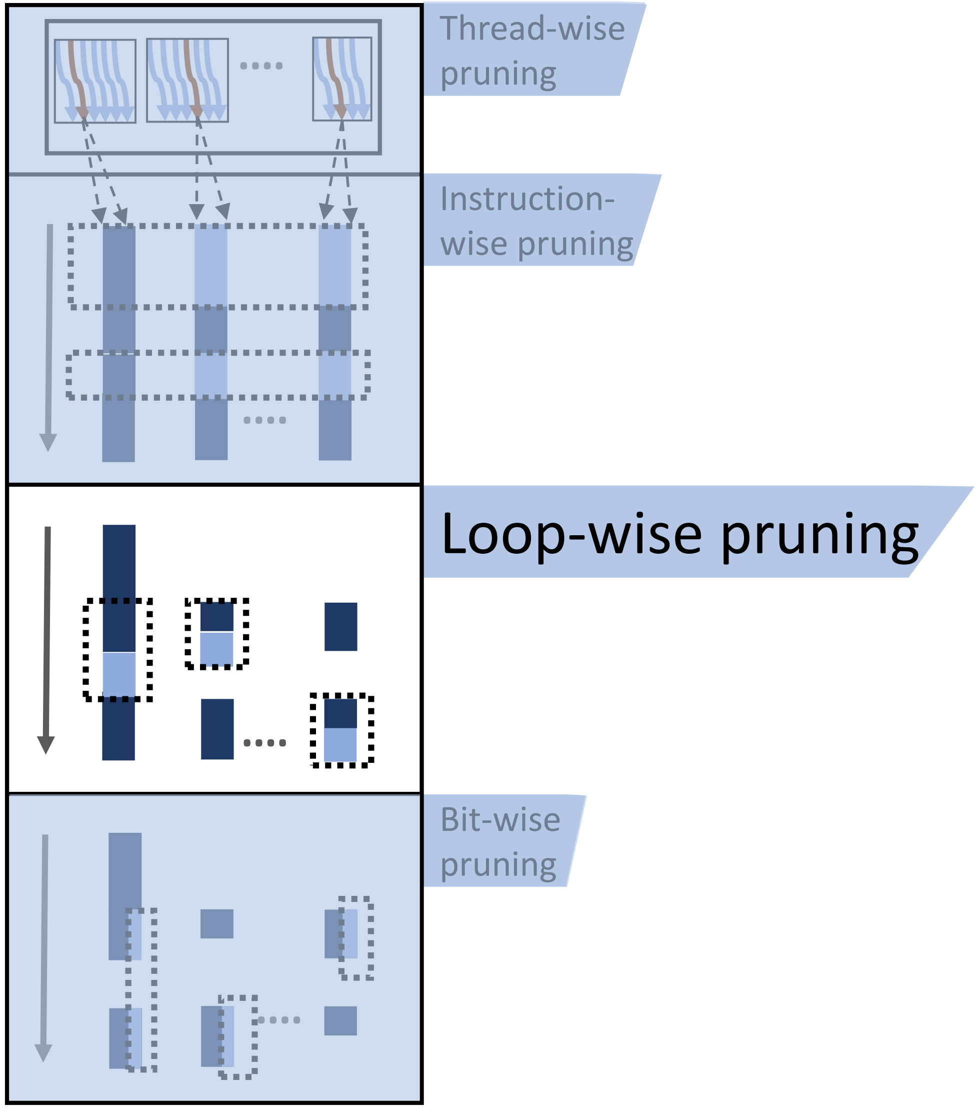


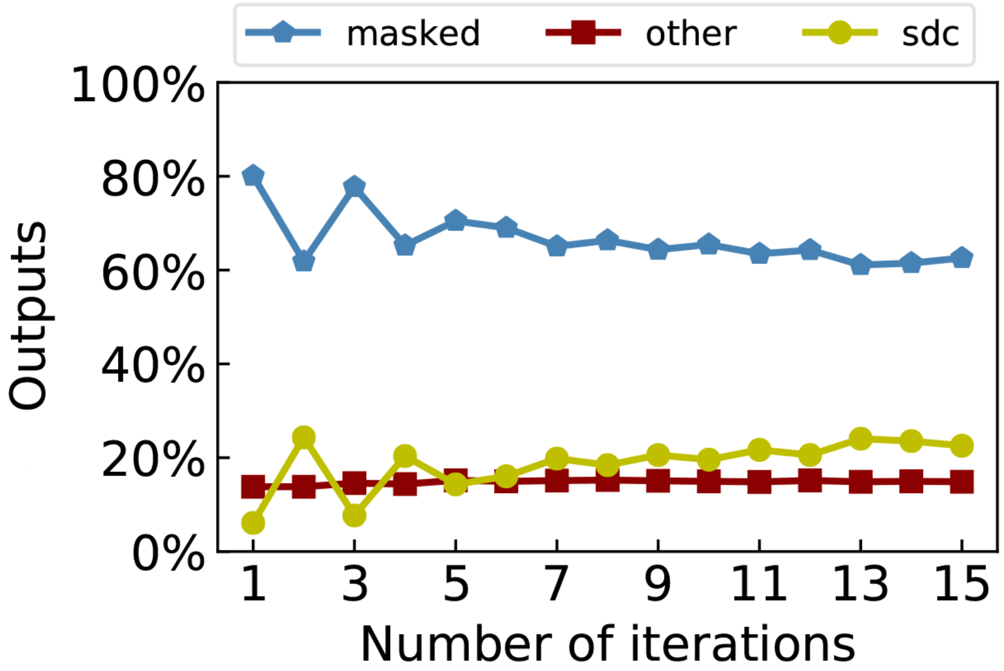
PathFinder K1

<table>
<tr><th>Thread “a” (iCnt = 533)</th><th>Thread “b” (iCnt = 516)</th></tr>
<tr><td>shl.u32 $r3, s[0x0010], 0x00000001</td><td>shl.u32 $r3, s[0x0010], 0x00000001</td></tr>
<tr><td>cvt.u32.u16 $r1, %ctaid.x</td><td>cvt.u32.u16 $r1, %ctaid.x</td></tr>
<tr><td>add.u32 $r3, -$r3, 0x00000100</td><td>add.u32 $r3, -$r3, 0x00000100</td></tr>
<tr><td>mul.wide.u16 $r4, $r1.lo, $r3.hi</td><td>mul.wide.u16 $r4, $r1.lo, $r3.hi</td></tr>
<tr><td>mad.wide.u16 $r4, $r1.hi, $r3.lo, $</td><td>16 $r4, $r1.hi, $r3.lo, $r4</td></tr>
<tr><td>……</td><td></td></tr>
<tr><td>cvt.s32.s32 $r2, -$r2</td><td>$r2, -$r2</td></tr>
<tr><td>and.b32 $p0|$o127, $r5, $r2</td><td>)|$o127, $r5, $r2</td></tr>
<tr><td>ssy 0x00000228</td><td>ssy 0x00000228</td></tr>
<tr><td>mov.u32 $r2, $r124</td><td>32 $r2, $r124</td></tr>
<tr><td>@$p0.eq bra l0x00000228</td><td>@$p bra l0x00000228</td></tr>
<tr><td>add.half.u32 $r7, s[0x0038], $r1</td><td></td></tr>
</table>



Progressive Fault Sites Pruning

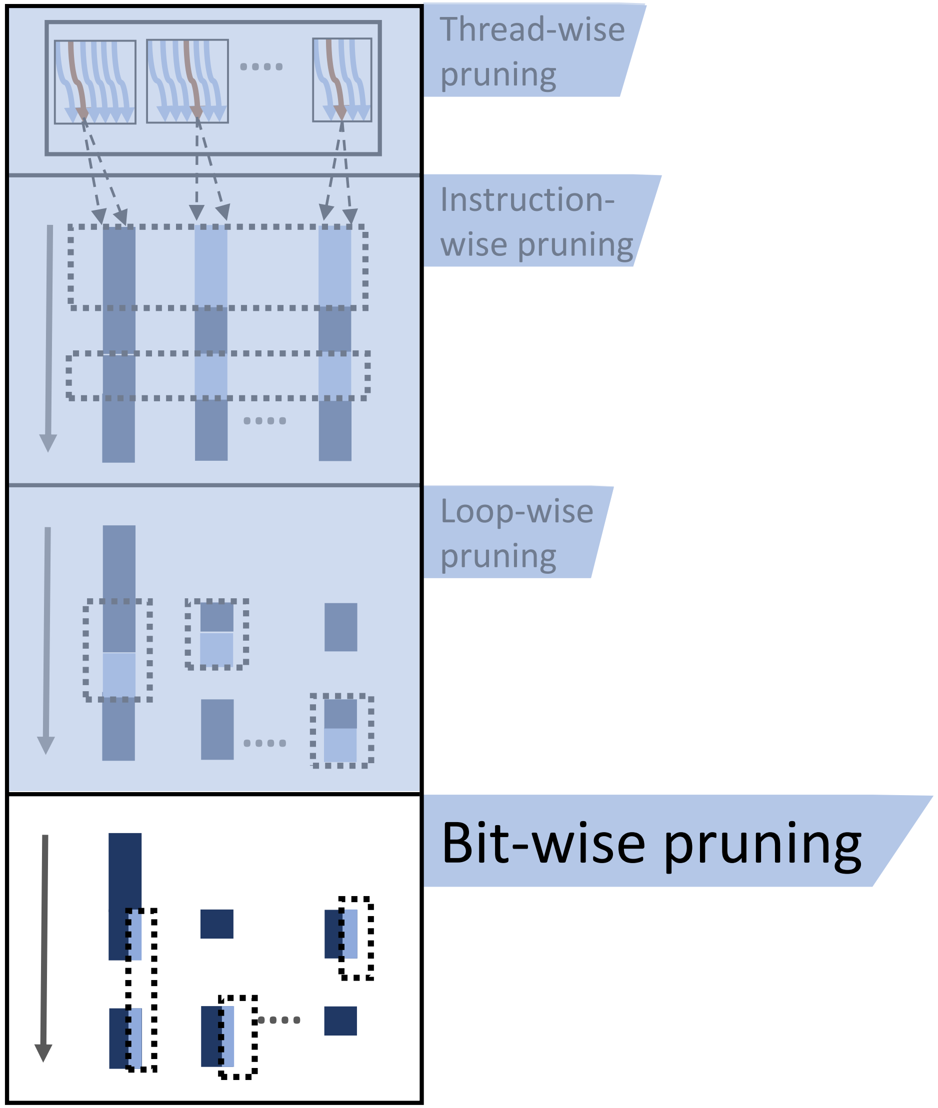


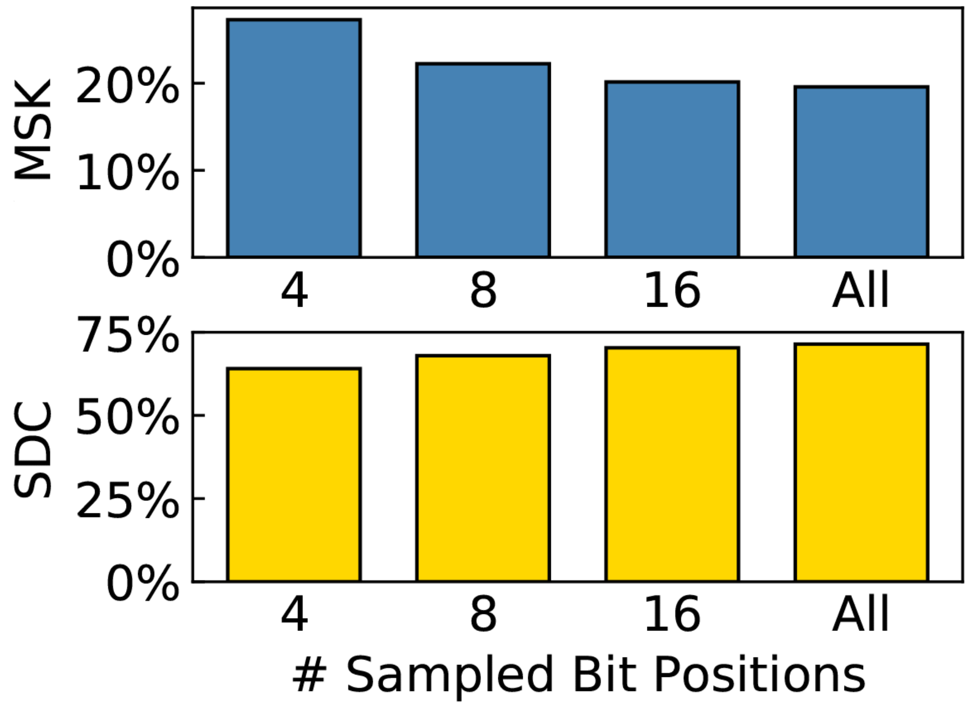


K-Means K1
34 loop iterations

WILLIAM & MARY

Progressive Fault Sites Pruning

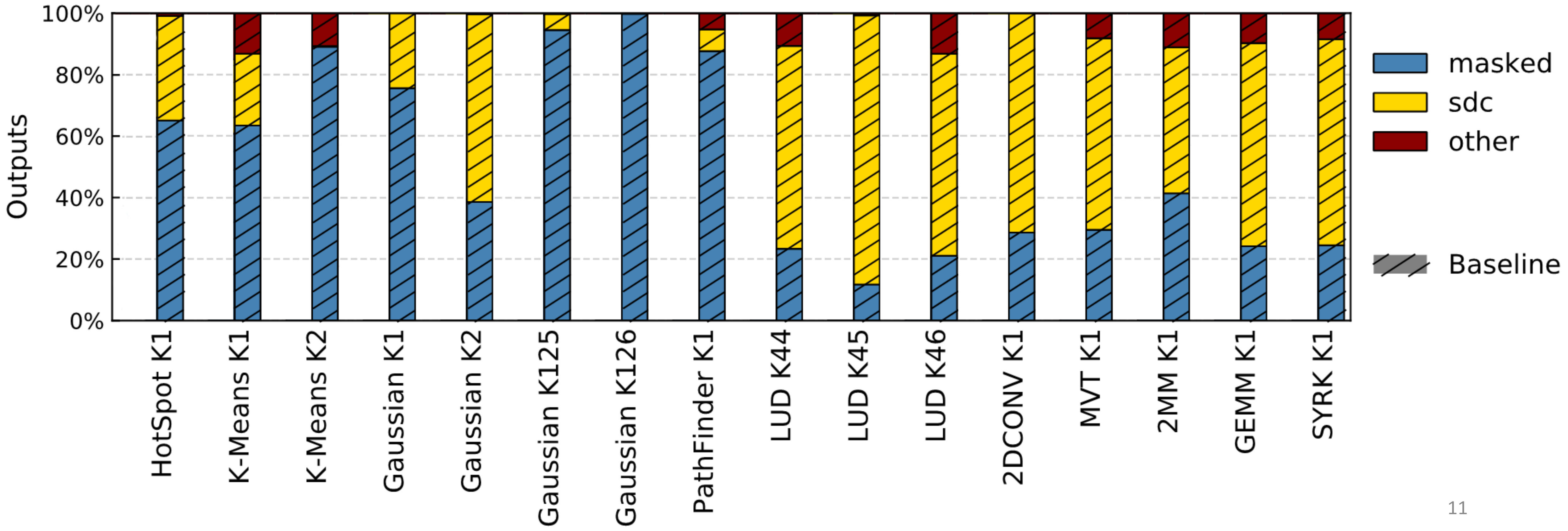




MVT K1

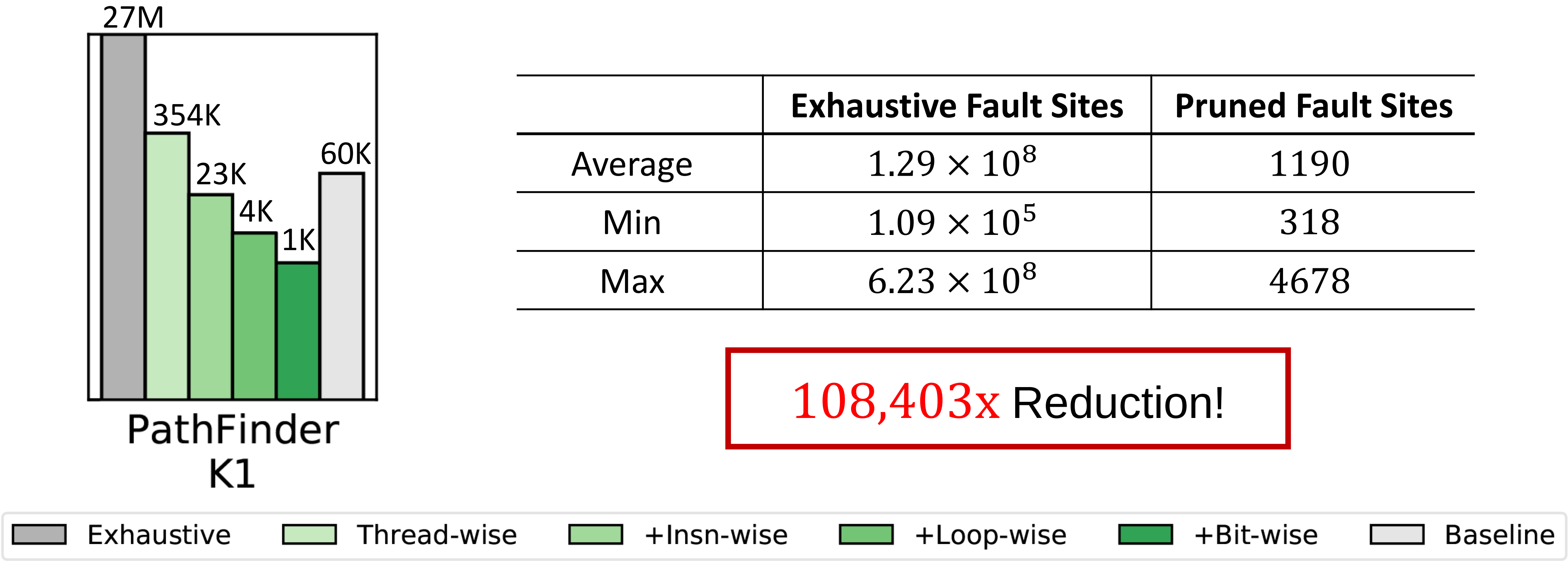
Evaluation

- ***Fault Site Pruning*** vs. ***Baseline***
  - *Baseline*: 60K fault sites (confidence interval=99.8%, error margin=1.26%)
    - Closest to ground truth



Evaluation

- Effectiveness



	Exhaustive Fault Sites	Pruned Fault Sites
Average	1.29×10^8	1190
Min	1.09×10^5	318
Max	6.23×10^8	4678

108,403x Reduction!

Conclusion

- GPGPU applications: huge unreachable exhaustive fault sites
- Progressive Fault Site Pruning
  - GPGPU-specific features
- Accurate GPU reliability assessment
- Significant reduction:
  - Up to 7 orders of magnitude
  - 108,403x on average

WILLIAM & MARY

CHARTERED 1693

Thank you :)

Fault Site Pruning for Practical Reliability Analysis of GPGPU Applications

Bin Nie, **Lishan Yang**, Adwait Jog, and Evgenia Smirni
College of William & Mary

This work is supported by NSF grants CCF-17175332 (CRII) and CCF-1750667 (CORE Small)