# GPA: A GPU Performance Advisor Based on Instruction Sampling

**Keren Zhou**, Xiaozhu Meng, Ryuichi Sai, John Mellor-Crummey

Rice University

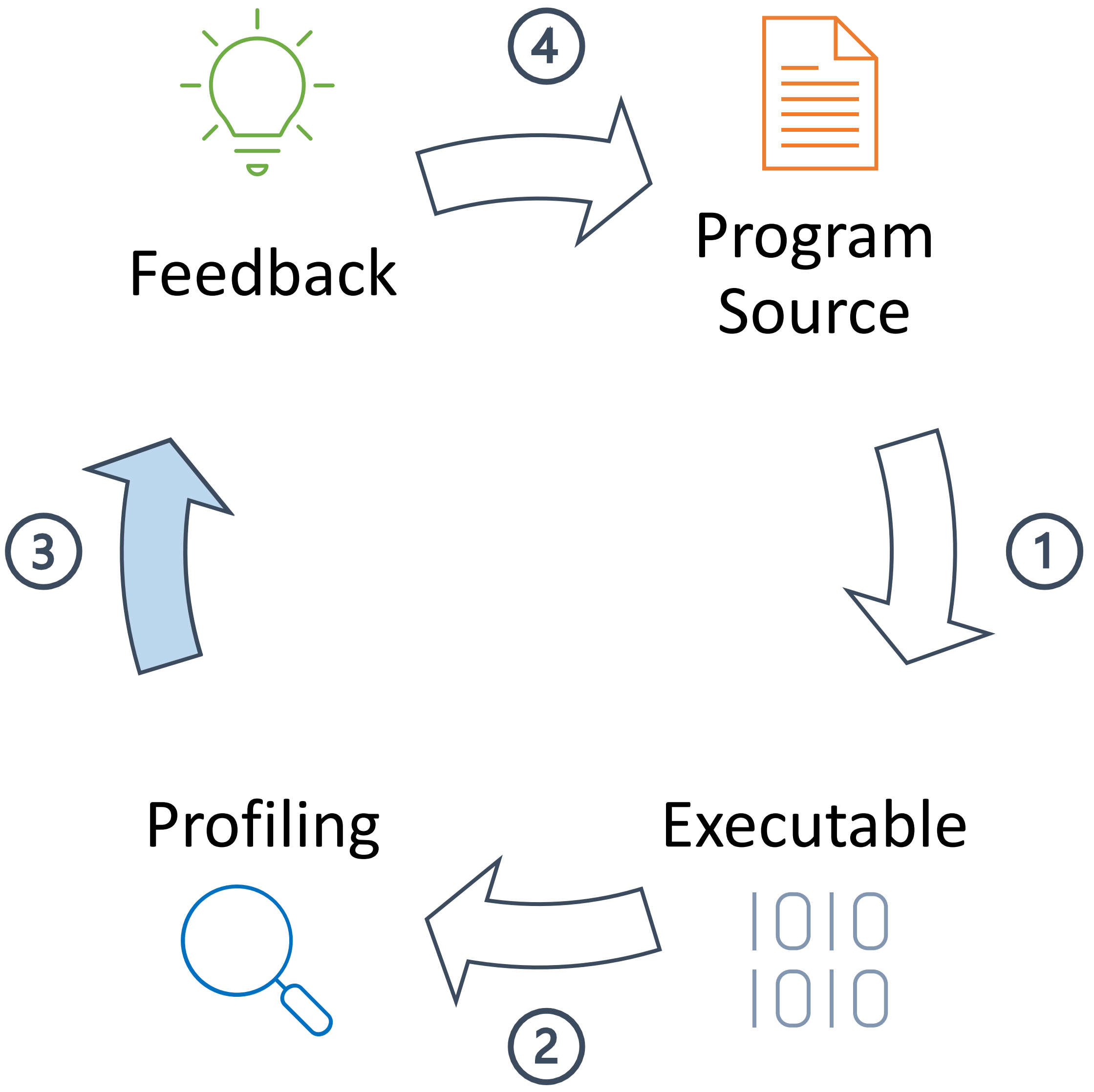
# Performance Optimization of GPU Kernels

Provide shallow insights

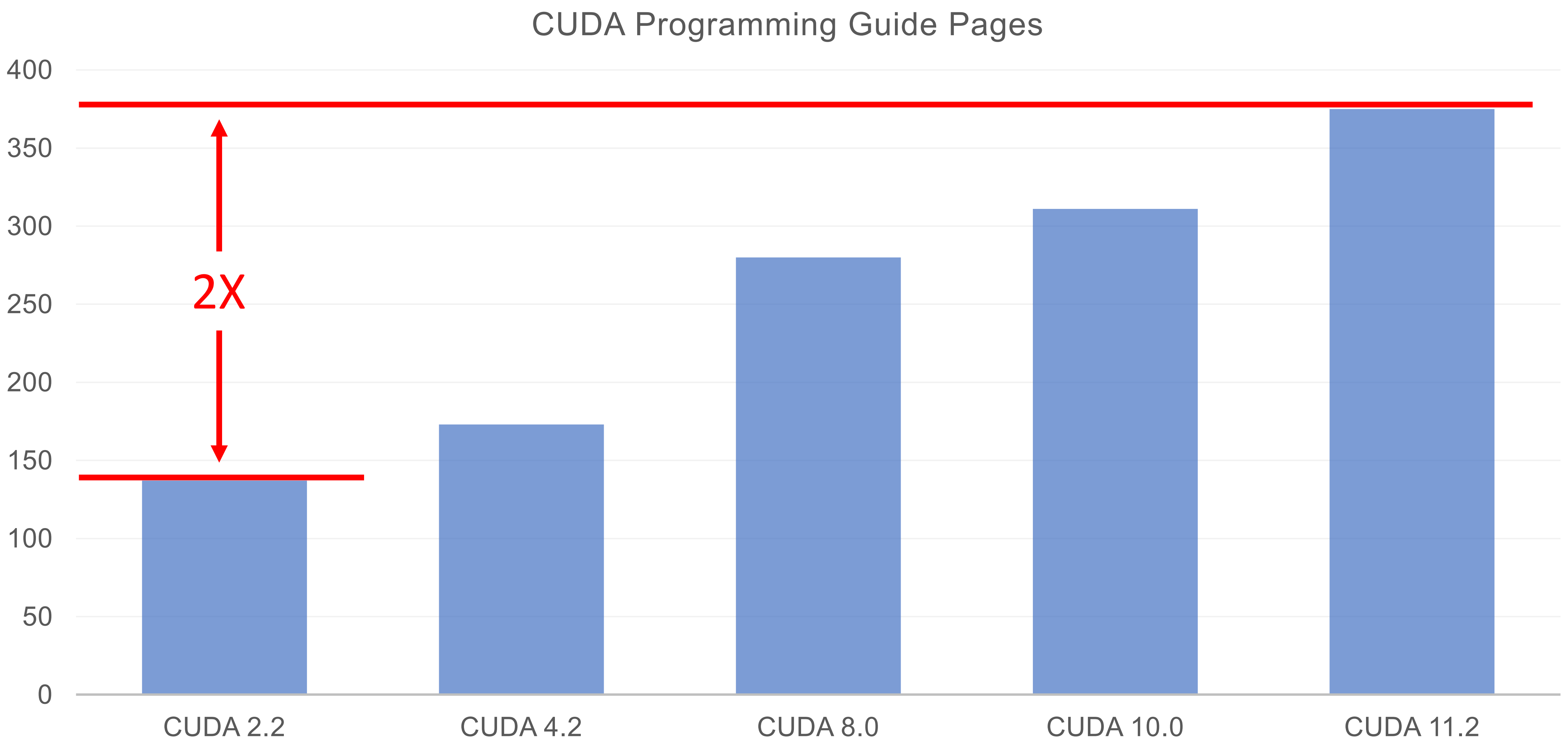
**GPU Performance Tools**

Nsight System

Nsight Compute



# Increasing Complexity of GPU Programming Model and Architecture



# Profile Rodinia - hotspot

```
1  for (int i = 0; i < iteration; i++) {
2      temp_t[ty][tx] = temp_on_cuda[ty][tx] + step_div_Cap *
3          (power_on_cuda[ty][tx] + (temp_on_cuda[S][tx] +
4           temp_on_cuda[N][tx] – 2.0 * temp_on_cuda[ty][tx]) * ...
5  }
```

- Nsight Compute
  - High *short_scoreboard* stall on **Line 4**
    - Warp was stalled waiting for a scoreboard dependency operation

What instructions cause the stall?

What optimizations are effective for eliminating the stall?

# GPA's Performance Report

```
1 for (int i = 0; i < iteration; i++) {
2     temp_t[ty][tx] = temp_on_cuda[ty][tx] + step_div_Cap *
3         (power_on_cuda[ty][tx] + (temp_on_cuda[S][tx] +
4          temp_on_cuda[N][tx] – 2.0 * temp_on_cuda[ty][tx]) * ...
5 }
```

**What instructions cause the stall?**

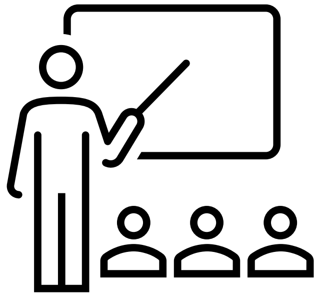
- Floating point type conversion (F2F) on Line 4 causes the stall

**What optimizations are effective for eliminating the stall?**

- Strength reduction optimization improves the performance by 9%
  - Avoid type conversion instructions that demote a 64-bit float to a 32-bit float

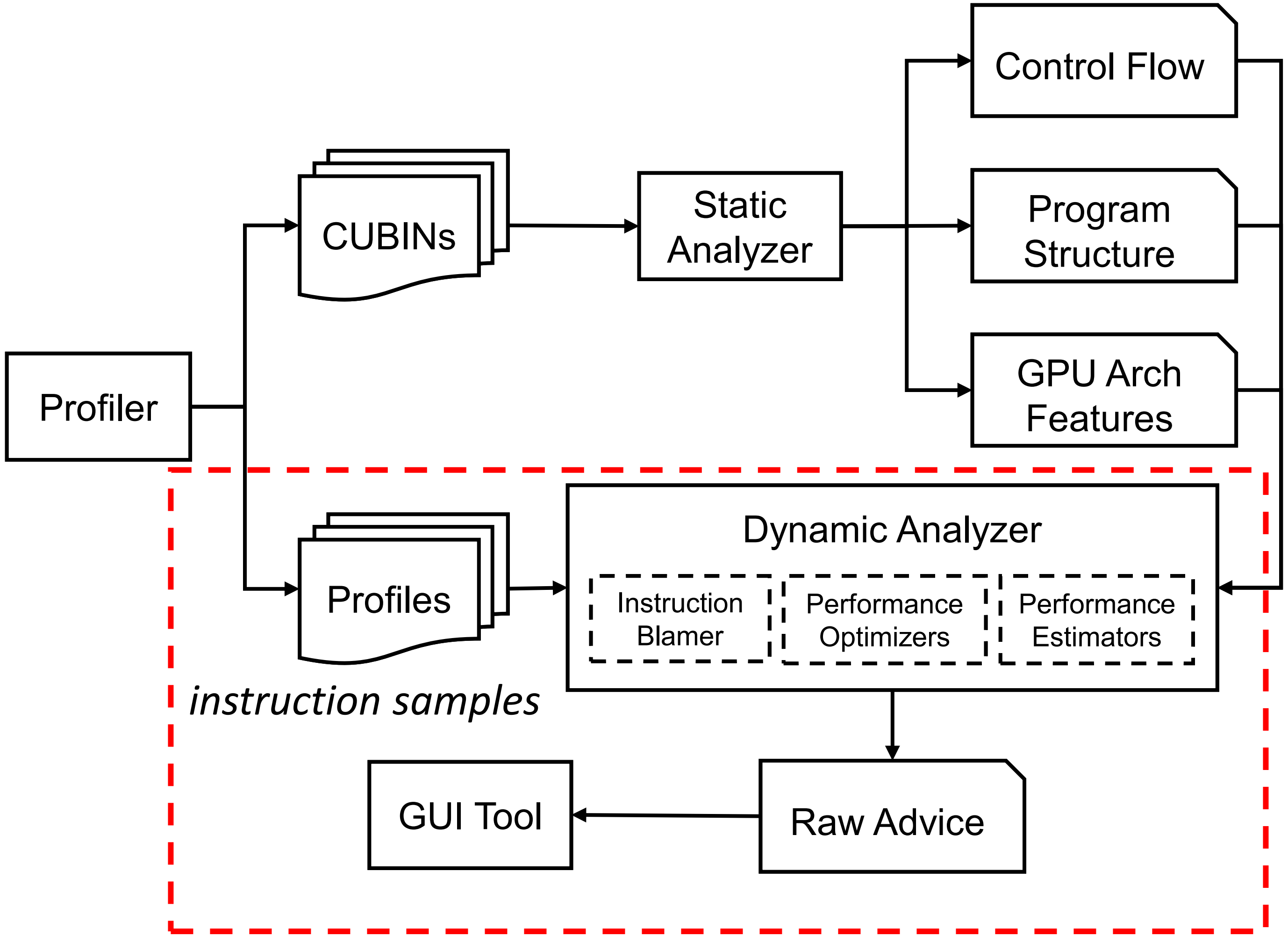
# Overview

- GPA is a GPU performance analysis tool that
  - identifies hot GPU kernels
  - reasons about the causes of slowness
  - suggests the most effective optimization strategies
  - estimates speedups for each optimization

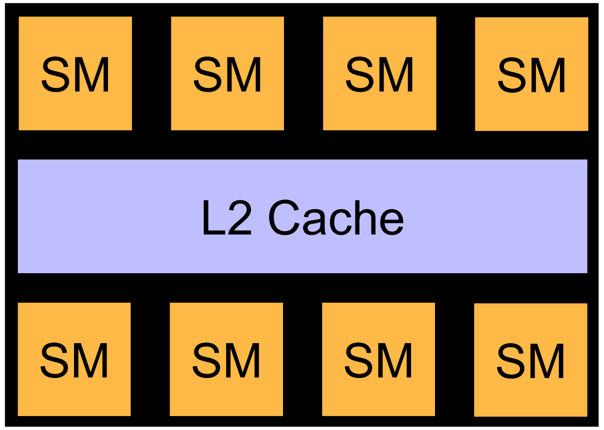


- GPA doesn't
  - modify source code
  - guarantee 100% accuracy of speedup estimate
- GPA works like an academic advisor who suggests approaches to achieve high scores but doesn't complete homework and exams on your behalf

# GPA's Framework



# Instruction Sampling on NVIDIA GPUs

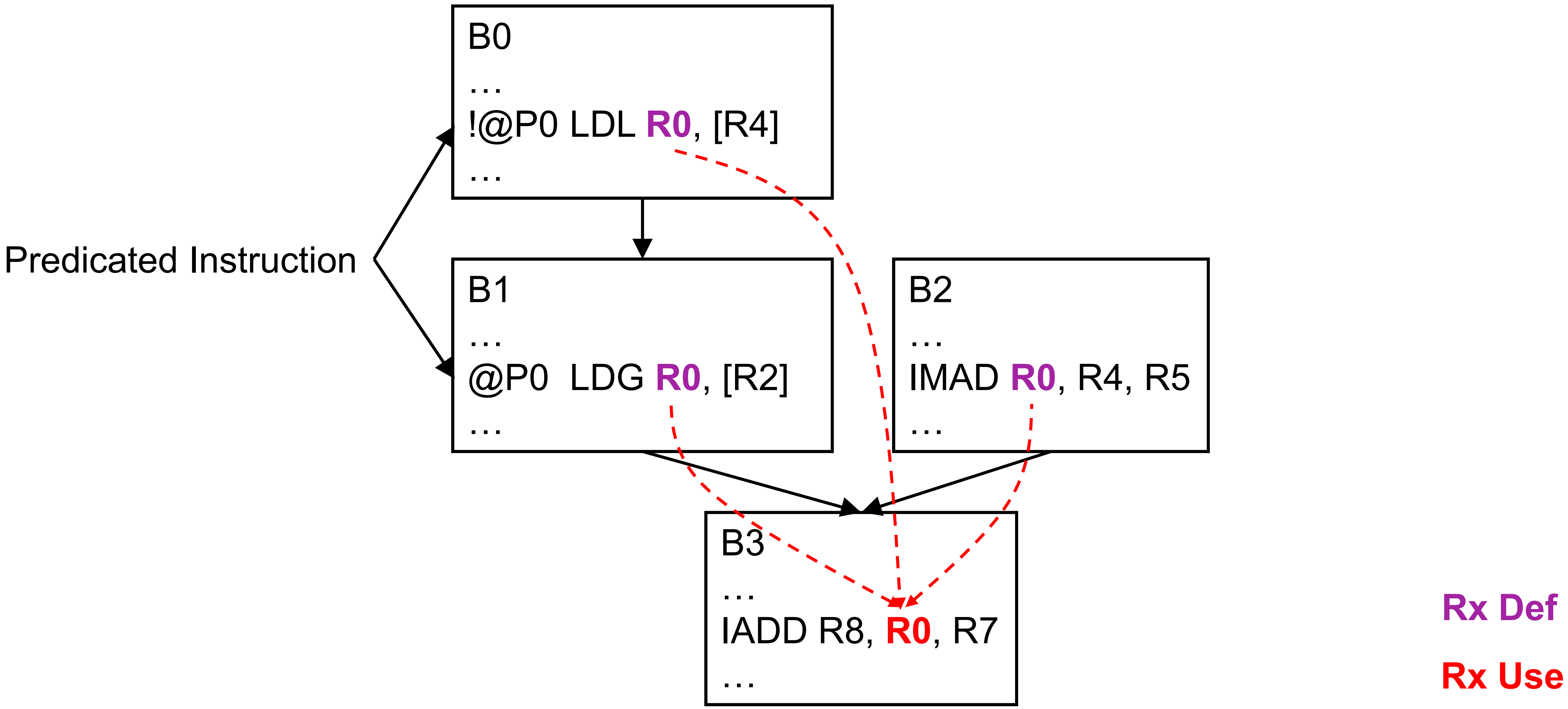



SM (Streaming Multiprocessor)

| PC | Stall Reason | Samples |
|---|---|---|
| 0xa0 | Memory dependency | 16 |
| 0xb0 | Memory throttling | 20 |
| 0xc0 | Instruction fetch | 3 |

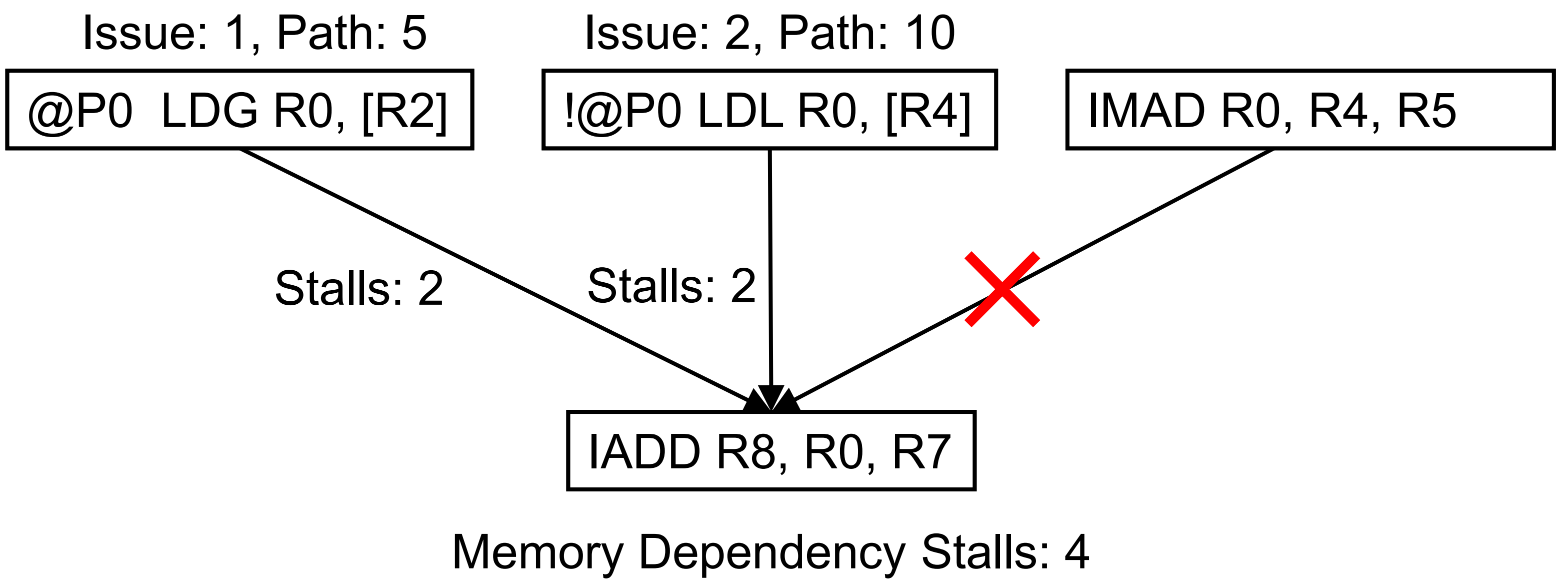
Which instructions cause memory dependencies?

# Analyze Instruction Dependencies

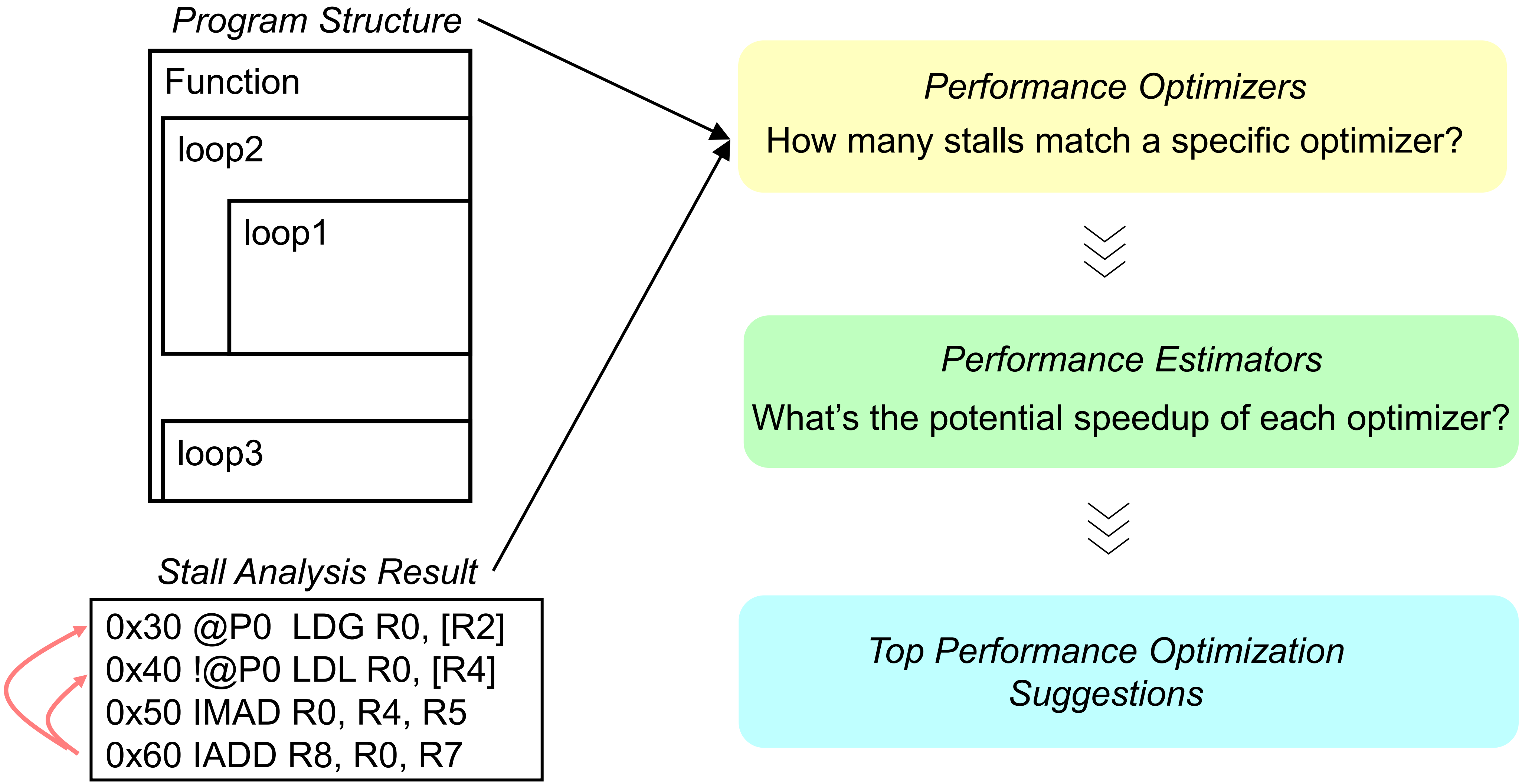


# Blame Stalls on Instructions

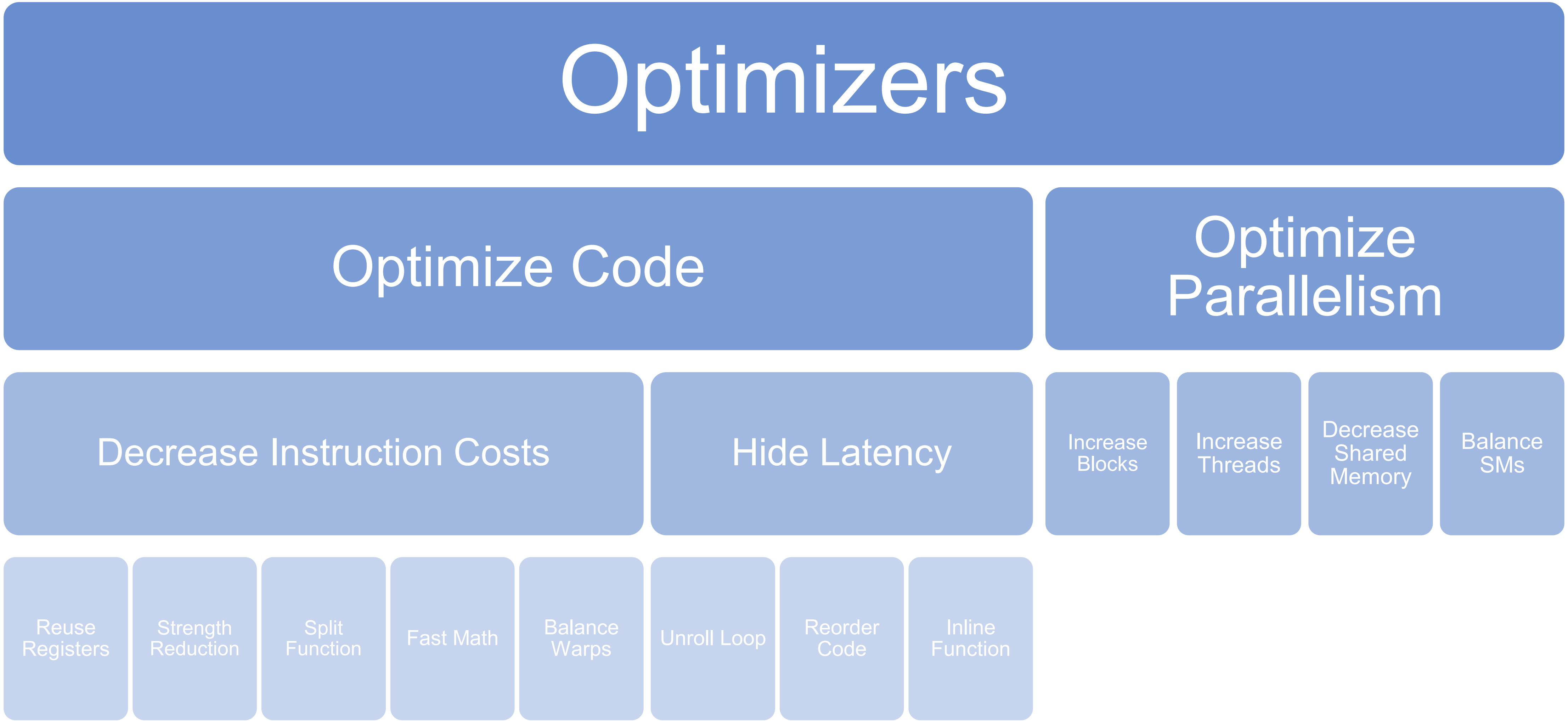
- Arithmetic instructions do not cause memory dependency stalls
- With more issued samples, blame more stalls on the register def
- For a longer path, blame fewer stalls on the register def



# Performance Advice Workflow



# Performance Optimizers



# Optimizer Examples

- Strength reduction
  - **Match if** long latency arithmetic instructions cause execution latency
    - Type conversion (e.g., $I2F, F2I$)
    - Costly math operations such as mod and division (e.g., $MUFU$)
  - **Suggest** improvements that are mathematically equivalent
- Loop unrolling
  - **Match if** significant memory/execution/synchronization dependency stalls exist in a loop
    - Both stall *source* and *dest* instructions are in the loop
  - **Suggest** unrolling the loop aggressively

# Code Optimization Estimators

- Decrease instruction costs
  - Total samples: $T$
  - Matching samples: $M$
  - Speedup: $\frac{T}{T-M}$
  - Strength reduction
- Hide latency
  - Total samples: $T$
  - Matching stalled samples: $M_s$
  - Active samples: $A$
  - Speedup: $\frac{T}{T-Min(A,M_s)}$
  - Reorder code

LDG R0, [R2]
STALL
STALL
~~I2F R5, R0~~
IADD R6, R6, R6
IADD R7, R7, R7

→

LDG R5, [R2]
STALL
STALL
IADD R6, R6, R6
IADD R7, R7, R7

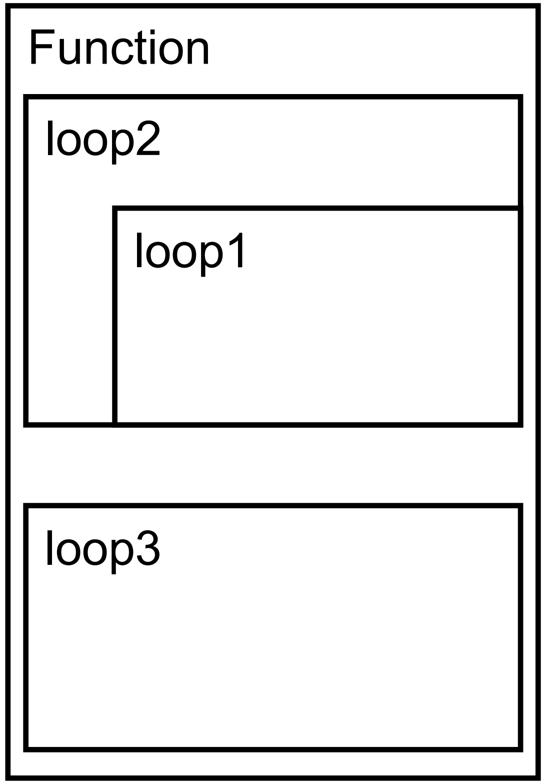
LDG R0, [R2]
STALL
STALL
I2F R5, R0
IADD R6, R6, R6
IADD R7, R7, R7

→

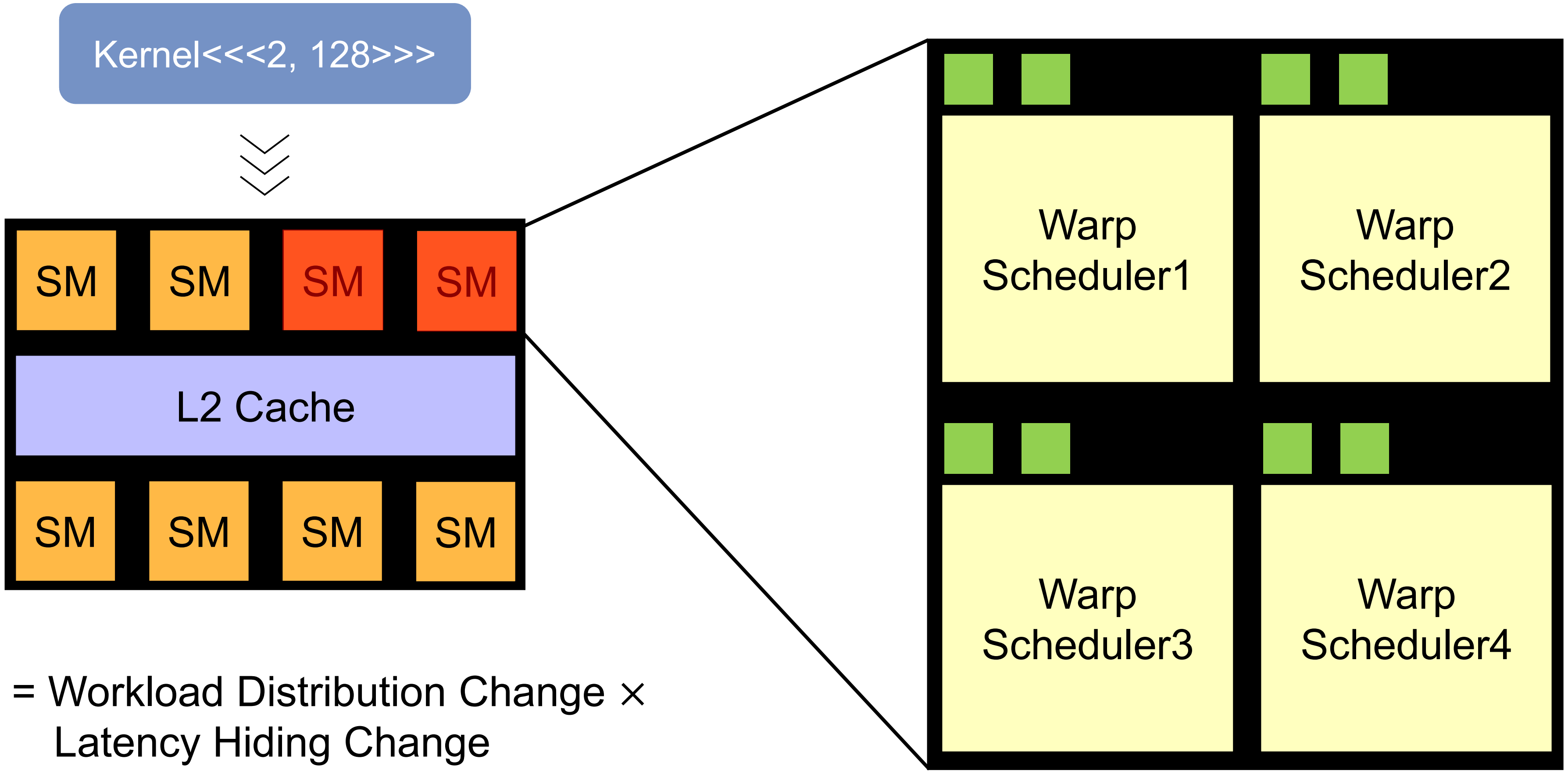
LDG R5, [R2]
IADD R6, R6, R6
IADD R7, R7, R7
I2F R5, R0

# Scope Analysis

- Observation
  - Optimizations such as loop unrolling only rearrange code in a specific *scope*
  - Only instructions (active samples) within the scope can be used to hide latency (latency samples)
- Active samples of a scope
  - Sum of active samples of all its nested scopes
- Matching samples of a scope
  - Matching samples within the scope itself
- $speedup(loop2) = \frac{T}{T - Min\left(A(loop2 + loop1), M_S(loop2)\right)}$



# Parallelism Optimization Estimator




Speedup = Workload Distribution Change $\times$ Latency Hiding Change

# Case Study - ExaTensor

Apply GPUStrengthReductionOptimizer optimization, importance 5.805%, estimate speedup 1.062x

Optimization

Long latency non-memory instructions are used. Look for improvements that are mathematically equivalent, but the compiler is not intelligent to do so.

1. Avoid integer division. Integer division requires using a special function unit to perform floating point transformations. One can use multiplication by a reciprocal instead.
2. Avoid conversion. If the float constant is multiplied by a 32-bit float value, the compiler might transform the 32-bit value to a 64-bit value first.

Hints

1. Hot BLAME GINS:LAT_IDEP_DEP code, importance 0.444%, speedup 1.004x, distance 1

Hotspot

From tensor_transpose at /home/kz21/Codes/GPA-Benchmark/ExaTENSOR/cuda2.cu:16
0x1620 at Line 34 in Loop at Line 30
To tensor_transpose at /home/kz21/Codes/GPA-Benchmark/ExaTENSOR/cuda2.cu:16
0x1630 at Line 34 in Loop at Line 30

*def* and *use* locations

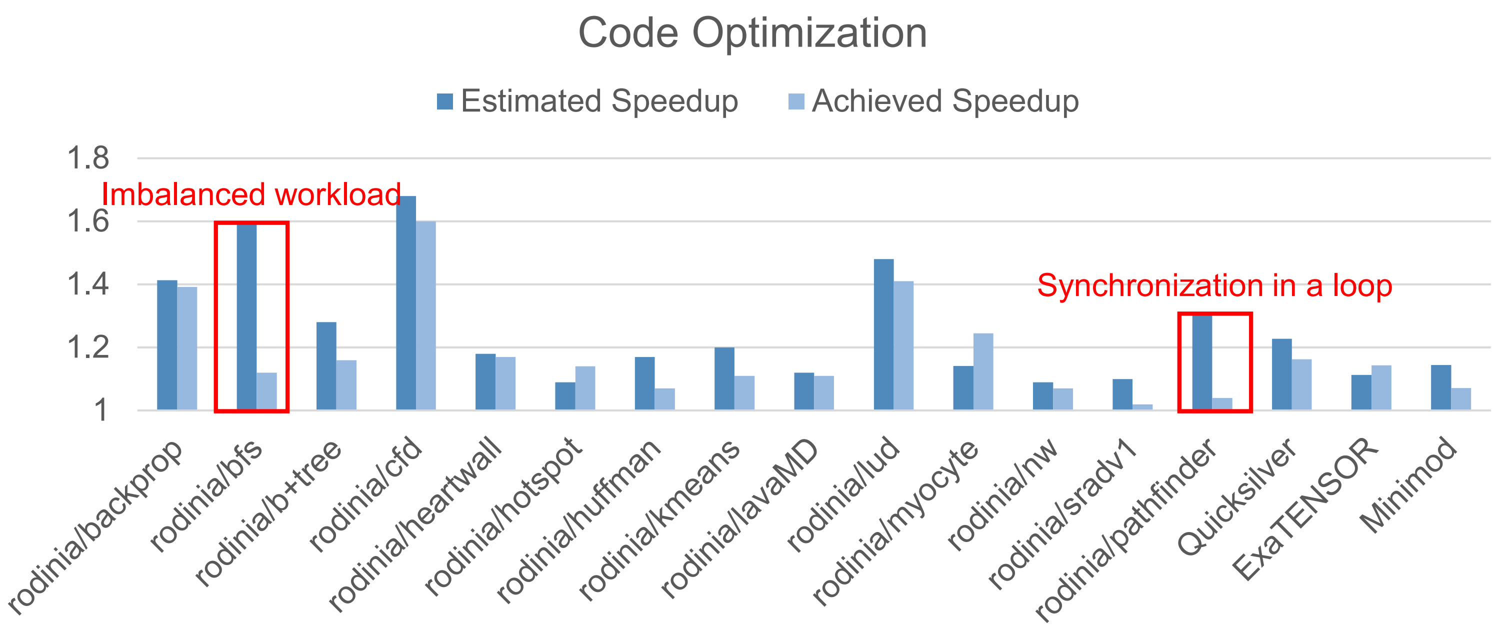
# Case Study - ExaTensor

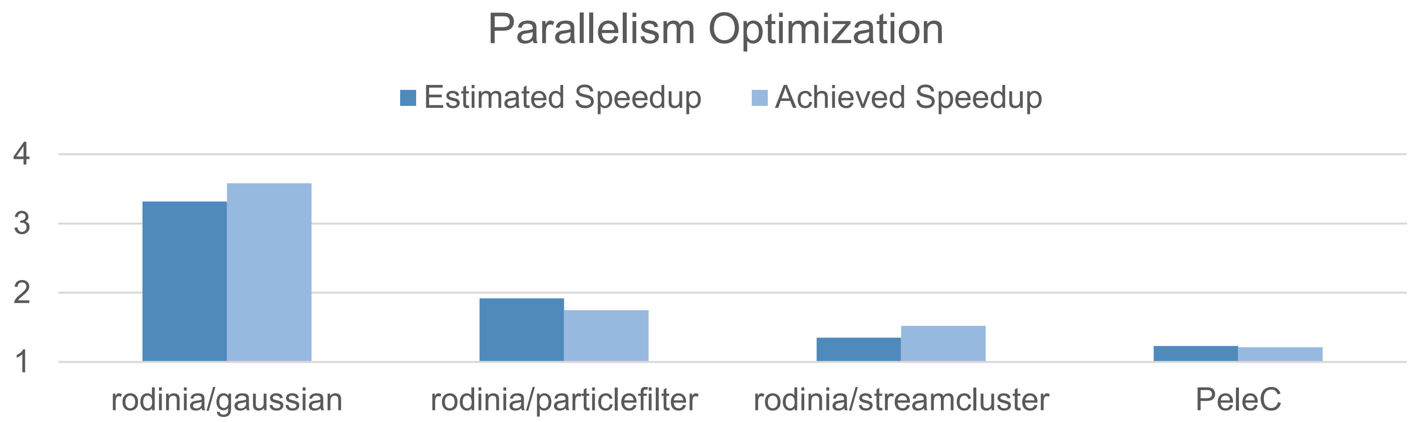
```
for (int i = threadIdx.x; i < tile_size; i += blockDim.x) {
    for (int j = 0; j < dim_output; j++) {
        im = it / shape_output[j];
        …
    }
}
```

- Strength reduction
  - Hint: Replace integer division with a multiplication by its reciprocal
  - Speedup: estimate 1.06×, achieved 1.11×
- Global memory transaction reduction
  - Hint: Replace global memory reads by constant memory reads if elements are shared between threads
  - Speedup: estimate 1.05×, achieved 1.03×

# Evaluation

- CPU
  - E5-2695v4
- GPU
  - NVIDIA V100 16GB
  - CUDA 11.0
- Benchmarks
  - ExaTENSOR
  - Rodinia
  - Quicksilver
  - Minimod
  - PeleC



# Status and Ongoing Work

- GPA provides an insightful performance report to guide performance optimization for GPU Kernels on NVIDIA GPUs
- New capabilities since the paper
  - Use instrumentation to collect more performance metrics
    - Improve stall attribution accuracy
    - Construct new advisors
  - Analyze new instructions on Turing and Ampere GPUs
    - Uniform data path instructions
    - Indirect memory access instructions
    - Asynchronous memory copy instructions

# Contact

- Tool
  - Jokeren/GPA: GPU Performance Advisor (github.com)
- Email
  - keren.zhou@rice.edu
  - xiaozhu.meng@rice.edu
  - ryuichi.sai@rice.edu
  - johnmc@rice.edu (PI)
- This presentation and recording belong to the authors. No distribution is allowed without the authors' permission