A Modular Static Cost Analysis For GPU Warp-Level Parallelism

Gregory Blike[†], Hannah Zicarelli[†], Udaya Sathiyamoorthy[†]

Julien Lange[‡], Tiago Cogumbreiro[†]

[†] UMass Boston / [‡] Royal Holloway, University of London

POPL, 2026

Overview

1. Motivation: Why GPUs matter and challenges of analyzing performance bottlenecks
2. Problem statement: Cost analysis of GPU kernels
3. Theory: Relational-Cost Analysis
4. Practice: Pico, a tool to analyze performance bottlenecks of CUDA kernel & Evaluation
5. Conclusion

Motivation

- Why GPUs matter
- Challenges of analyzing performance bottlenecks

Why do GPUs matter?

GPUs are everywhere

The LLM revolution is powered by GPUs

GPT-5 was trained using 200k GPUs

Source: `www.linkedin.com/feed/update/urn:li:activity:7359279165121970176/`

Scientific advancement is powered by GPUs

Power 9 out of 10 of the Top 10 super computers

		Name	GPU
1	🇺🇸	El Capitan	☑
2	🇺🇸	Frontier	☑
3	🇺🇸	Aurora	☑
4	🇪🇺	JUPITER	☑
5	🇺🇸	Eagle	☑
6	🇪🇺	HPC6	☑
7	🇯🇵	Fugaku	☐
8	🇨🇭	Alps	☑
9	🇪🇺	LUMI	☑
10	🇪🇺	Leonardo	☑

top500.org/lists/top500/list/2025/06/

Credit: asc.llnl.gov

UMass Boston

Performance

The raison dêtre of GPU programming

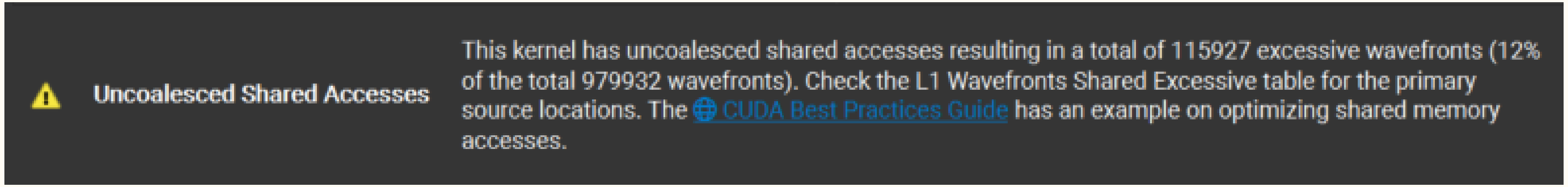
Optimizing GPU programs

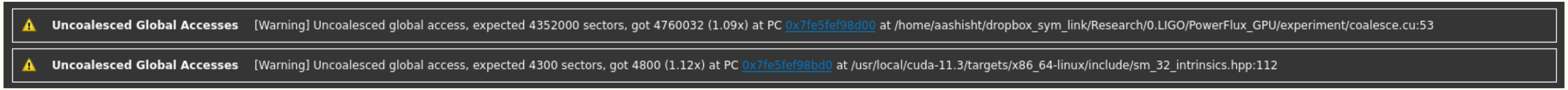
- Maximize parallelism
- Optimize memory access
- Minimize divergent behavior (more on this later)

Analyzing performance bottlenecks

Dynamic approaches

- Nvidia’s nSight (fixed run, requires input); **no worst-case analysis**
- Symbolic-execution tools only presence, not frequency/cost





UMass Boston

Analyzing performance bottlenecks

Static approaches

- PUG [FSE10,IPDPSW12] and GPUDrano[CAV17,FMSD21] only detect location, but not how frequent
- State-of-the-art **RaCUDA**[POPL21,TOPC24] has limited support:
  - GPU-specific semantics, leading to over approximations
  - cannot reason about the precision of the analysis (no exact costs)

UMass Boston

Problem statement:

Cost analysis of GPU kernels

How to analyze the cost of a performance bottleneck **statically**?

Cost analysis: statically analyzing the amount of resources needed to run a program.

Cost Analysis

Problem: count active threads per memory access

```
for (int x = 0; x <= threadIdx.x; x++) {
    read(array[x]); // How many threads are active?
}
```

Solution:

$$cost = (W^2 + W)/2 \quad \text{where } W \text{ is the number of threads}$$

Thus, when $W = 4$:

$$(4^2 + 4)/2 = 20/2 = 10$$

UMass Boston

But, why is the cost $(W^2 + W)/2$?

Understanding Warp-Semantics

Warp semantics in GPU programming

```
for (int x = 0; x <= threadIdx.x; x++) {
    foo(A[x]);
}
```

- ***Warp:*** a group of W-threads that execute the kernel lockstep
- `threadIdx.x` is a unique thread-identifier

UMass Boston

At runtime: per-thread view for $W = 4$

```
for(x=0;x<=0;x++){
    foo(A[x]);
}
```

```
for(x=0;x<=1;x++){
    foo(A[x]);
}
```

```
for(x=0;x<=2;x++){
    foo(A[x]);
}
```

```
for(x=0;x<=3;x++){
    foo(A[x]);
}
```

Warp semantics example

x = 0

```
for(x=0;x<=0;x++)
```

```
for(x=0;x<=1;x++)
```

```
for(x=0;x<=2;x++)
```

```
for(x=0;x<=3;x++)
```

Warp semantics example

x = 0

`for(x=0;x<=0;x++)`	`for(x=0;x<=1;x++)`	`for(x=0;x<=2;x++)`	`for(x=0;x<=3;x++)`
`  foo(A[0])`	`  foo(A[0])`	`  foo(A[0])`	`  foo(A[0])`

Warp semantics example

x = 0

`for(x=0;x<=0;x++)`	`for(x=0;x<=1;x++)`	`for(x=0;x<=2;x++)`	`for(x=0;x<=3;x++)`
`  foo(A[0])`	`  foo(A[0])`	`  foo(A[0])`	`  foo(A[0])`

x = 1

`// x <= 0 is FALSE`	`for(x=0;x<=1;x++)`	`for(x=0;x<=2;x++)`	`for(x=0;x<=3;x++)`

Warp semantics example

x = 0

`for(x=0;x<=0;x++)`	`for(x=0;x<=1;x++)`	`for(x=0;x<=2;x++)`	`for(x=0;x<=3;x++)`
`  foo(A[0])`	`  foo(A[0])`	`  foo(A[0])`	`  foo(A[0])`

x = 1

`// x <= 0 is FALSE`	`for(x=0;x<=1;x++)`	`for(x=0;x<=2;x++)`	`for(x=0;x<=3;x++)`
`// inactive`	`  foo(A[1])`	`  foo(A[1])`	`  foo(A[1])`

Warp semantics example

x = 0

`for(x=0;x<=0;x++)`	`for(x=0;x<=1;x++)`	`for(x=0;x<=2;x++)`	`for(x=0;x<=3;x++)`
`  foo(A[0])`	`  foo(A[0])`	`  foo(A[0])`	`  foo(A[0])`

x = 1

`// x <= 0 is FALSE`	`for(x=0;x<=1;x++)`	`for(x=0;x<=2;x++)`	`for(x=0;x<=3;x++)`
`// inactive`	`  foo(A[1])`	`  foo(A[1])`	`  foo(A[1])`

x = 2

`// x <= 0 is FALSE`	`// x <= 0 is FALSE`	`for(x=0;x<=2;x++)`	`for(x=0;x<=3;x++)`

Warp semantics example

x = 0

`for(x=0;x<=0;x++)` `  foo(A[0])`	`for(x=0;x<=1;x++)` `  foo(A[0])`	`for(x=0;x<=2;x++)` `  foo(A[0])`	`for(x=0;x<=3;x++)` `  foo(A[0])`

x = 1

`// x <= 0 is FALSE` `// inactive`	`for(x=0;x<=1;x++)` `  foo(A[1])`	`for(x=0;x<=2;x++)` `  foo(A[1])`	`for(x=0;x<=3;x++)` `  foo(A[1])`

x = 2

`// x <= 0 is FALSE` `// inactive`	`// x <= 0 is FALSE` `// inactive`	`for(x=0;x<=2;x++)` `  foo(A[2])`	`for(x=0;x<=3;x++)` `  foo(A[2])`

Counting active threads per memory access

```
for (int x = 0; x <= threadIdx.x; x++) {
    foo(A[x]); // How many threads are active?
}
```

Iteration (x)	*Thread 0*	*Thread 1*	*Thread 2*	*Thread 3*	*Active Count*
0	☑	☑	☑	☑	4
1	☐	☑	☑	☑	3
2	☐	☐	☑	☑	2
3	☐	☐	☐	☑	1
4	☐	☐	☐	☐	0

$$cost = (W^2 + W)/2 = 4 + 3 + 2 + 1 = 10$$

UMass Boston

Why is counting threads important?

Thread-divergence is a core aspect of 2 major performance bottlenecks:

- Bank conflicts
- Uncoalesced accesses

> **Key takeaway:** Precisely characterizing thread-divergence $\implies$ precisely characterizing performance bottlenecks

Relational-Cost Analysis

Our approach

Reduce problem to analyzing cost of sequential program

```
for (x=0; x<=threadIdx.x; x++) {   ➡   for x ∈ 0 .. W-1 {
 foo(A[x]);                         ➡     tick (W - x);
}                                   ➡   }
```

A sequential program captures the control-flow of the ***warp*** (group) of threads.

- `tick` consumes `W - x` resources
- Off-the-shelf tools can reduce sequential program to symbolic expression

UMass Boston

Our approach

> Translate a concurrent-warp semantics into a sequential program, and build on the vast literature of cost analysis of ***sequential*** programs!

Our approach

> Translate a concurrent-warp semantics into a sequential program, and build on the vast literature of cost analysis of ***sequential*** programs!

- **Step 1:** Translate (A) CUDA kernel into (B) **concurrent** Intermediate Representation (IR)
- **Step 2:** Translate (B) concurrent IR into (C) **sequential cost program**
- **Step 3:** Given (C) sequential program calculate cost (D)

Our approach

> Translate a concurrent-warp semantics into a sequential program, and build on the vast literature of cost analysis of ***sequential*** programs!

- **Step 1:** Translate (A) CUDA kernel into (B) **concurrent** Intermediate Representation (IR)
- **Step 2:** Translate (B) concurrent IR into (C) **sequential cost program**
- **Step 3:** Given (C) sequential program calculate cost (D)

```
(A)
for(x=0;x<=threadIdx.x;x++){
    foo(A[x]);
}
```

➡

```
(B)
for x ∈ 0..tid{
 rd A[x];
}
```

➡

```
(C)
for x ∈ 0..W-1{
  tick (W - x);
}
```

➡ (D)

$$\sum_{x=0}^{W-1} (W - x) = (W^2 + W)/2$$

Our approach

> Translate a concurrent-warp semantics into a sequential program, and build on the vast literature of cost analysis of ***sequential*** programs!

- **Step 1:** Translate (A) CUDA kernel into (B) **concurrent** Intermediate Representation (IR)
- **Step 2:** Translate (B) concurrent IR into (C) **sequential cost program**
- **Step 3:** Given (C) sequential program calculate cost (D)

```
(A)
for(x=0;x<=threadIdx.x;x++){
    foo(A[x]);
}
```

➡

```
(B)
for x ∈ 0..tid{
  rd A[x];
}
```

➡

```
(C)
for x ∈ 0..W-1{
  tick (W - x);
}
```

➡ (D)

$$\sum_{x=0}^{W-1} (W - x) = (W^2 + W)/2$$

Major Contribution: Prove correctness of Step 2 with a new **Relational-Cost Analysis**

UMass Boston

Key Technical Contributions

- Lower, upper, exact bounds can all be handled with the same formalism
  Prior work: only upper-bounds bounds
- CUDA semantics with support for thread-divergence
  Prior work: ommission from the SOTA
- A soundness result that shows that static costs are correct for all types of bounds (fully mechanized in ROCQ)

UMass Boston

Concurrent IR to **Sequential Program**

- ***Q: How can we guarantee that the sequential program captures the cost of the warp-semantics?***
- **Answer:** A theoretical framework to relate the cost of the concurrent and sequential programs, eg, show that two programs have the ***same*** cost.

$$\Gamma; \Phi \vdash p \sim s$$

UMass Boston

Relational-cost analysis

$$\Gamma; \Phi \vdash p \sim s$$

- A proof system to derive the cost of a protocol p in terms of a sequential program s under relation $\sim$
- $\sim \in \{=, \leq, \geq\}$ among others

- Γ - typing environment
- Φ - active threads (b-exp)
- p - warp protocol
- $\sim$ - cost relation
- s - sequential cost

UMass Boston

Example

$$\emptyset; t \vdash \ \mathsf{for}\ x \in 0..\mathsf{tid}\ \{\mathtt{A}[0]\} \ = \ \mathsf{for}\ x \in 0..W-1\ \{\mathsf{tick}(W-x)\}$$

Example

$$\boxed{\emptyset; t} \vdash \texttt{for } x \in 0..\mathrm{tid}\ \{A[0]\} \;=\; \texttt{for } x \in 0..W-1\ \{\mathrm{tick}(W-x)\}$$

- $\emptyset$ - closed environment
- t - all threads active (t = true boolean)

Example

$$\emptyset; t \vdash \boxed{\text{for } x \in 0..\text{tid } \{\mathtt{A}[0]\}} = \text{for } x \in 0..W - 1 \; \{\text{tick}(W - x)\}$$

- $\emptyset$ - closed environment
- t - all threads active (t = true boolean)
- LHS: warp protocol

Theory overview

$$\emptyset; \mathsf{t} \vdash \ \mathsf{for}\ x \in 0..\mathsf{tid}\ \{\mathsf{A}[0]\} \ = \boxed{\mathsf{for}\ x \in 0..W-1\ \{\mathsf{tick}(W-x)\}}$$

- $\emptyset$ - closed environment
- t - all threads active (t = true boolean)
- LHS: warp protocol
- RHS: sequential cost

Theory overview

$$\emptyset; t \vdash \ \mathsf{for}\ x \in 0..\mathsf{tid}\ \{A[0]\}\ \boxed{=}\ \mathsf{for}\ x \in 0..W-1\ \{\mathsf{tick}(W-x)\}$$

- $\emptyset$ - closed environment
- t - all threads active (t = true boolean)
- LHS: warp protocol
- RHS: sequential cost
- $=$ - same cost

Analysis at a glance: conditionals

Uniform conditionals

IF-U

$$\frac{\Gamma \vdash c : \mathsf{U} \qquad \Gamma ; \Phi \vdash p \sim s}{\Gamma ; \Phi \vdash \mathsf{if}\ (c)\ \{p\} \sim \mathsf{if}\ (c)\ \{s\}}$$

> When a conditional has the same value for **all** threads (U), then we preserve that conditonal sequentially

Analysis at a glance: conditionals

Uniform conditionals

IF-U

$$\frac{\Gamma \vdash c : \mathsf{U} \qquad \Gamma ; \Phi \vdash p \sim s}{\Gamma ; \Phi \vdash \textsf{if}\ (c)\ \{p\} \sim \textsf{if}\ (c)\ \{s\}}$$

> When a conditional has the same value for **all** threads (U), then we preserve that conditonal sequentially

Divergent conditionals

IF-D

$$\frac{\Gamma \vdash c : \mathsf{D} \qquad \Gamma ; \Phi \land c \vdash p \sim s}{\Gamma ; \Phi \vdash \textsf{if}\ (c)\ \{p\} \sim s}$$

> Otherwise (D), we capture the condition in the expression of active threads $\Psi \land c$.

UMass Boston

Analysis at a glance: conditionals

FOR

$$\frac{\Gamma;\Phi \vdash_{\sim} r\ \mathcal{I}\ r' : \tau \qquad \Gamma, x : \tau_{(r,r')};\Phi \vdash p \sim s}{\Gamma;\Phi \vdash \mathsf{for}\ x \in r\ \{p\} \sim \mathsf{for}\ x \in r'\ \{s\}}$$

- **Loop analysis:** Relates a range r for warp with a range r' for sequential
  Lemma 5.4: If r is uniform, use r!
- Compare the loop body, while recording the original ranges of x

Key contributions

$$\emptyset; \mathsf{t} \vdash \ \mathsf{for}\ \mathsf{x} \in 0..\mathsf{tid}\ \{\mathsf{A}[0]\} \ = \ \mathsf{for}\ \mathsf{x} \in 0..\mathsf{W}-1\ \{\mathsf{tick}(\mathsf{W}-\mathsf{x})\}$$

1. A **thread-divergent** range $0..\mathrm{tid}$ captured by a sequential range $0..W-1$
   SOTA limitation: thread-divergent loops are unsupported
2. Analysis states exact costs ($=$)
   SOTA limitaton: only upper-bounds (cost over approximations)
3. Symbolic costs per iteration $W-x$
   SOTA limitation:only supports fixed costs, say 4

UMass Boston

Proving loops have same cost

$$\frac{\emptyset; t \vdash_{=} 0..\mathtt{tid} \approx 0..W-1 : \mathtt{D} \qquad \emptyset, x : D_{(0..\mathtt{tid}, 0..\mathtt{W}-1)}; t \vdash A[0] = \mathtt{tick}(W - x)}{\emptyset; t \vdash \ \mathtt{for}\ x \in 0..\mathtt{tid}\ \{\mathtt{A}[0]\} \ = \ \mathtt{for}\ x \in 0..W-1\ \{\mathtt{tick}(W-x)\}}\ \textsc{for}$$

Proving loops have same cost

$$\frac{\boxed{\emptyset; \mathtt{t} \vdash_{=} 0..\mathtt{tid} \approx 0..W - 1 : \mathtt{D}} \qquad \emptyset, x : \mathtt{D}_{(0..\mathtt{tid}, 0..W-1)}; \mathtt{t} \vdash \mathtt{A}[0] = \mathtt{tick}(W - x)}{\emptyset; \mathtt{t} \vdash \mathtt{for}\ x \in 0..\mathtt{tid}\ \{\mathtt{A}[0]\} = \mathtt{for}\ x \in 0..W - 1\ \{\mathtt{tick}(W - x)\}}\ \text{FOR}$$

1. **Prove that the iteration spaces have same number of iterations (≈)**

Proving loops have same cost

$$\frac{\emptyset; \mathtt{t} \vdash_{=} 0..\mathtt{tid} \approx 0..W-1 : \mathtt{D} \qquad \boxed{\emptyset, \mathtt{x} : \mathtt{D}_{(0..\mathtt{tid},0..W-1)}; \mathtt{t} \vdash \mathtt{A}[0] = \mathtt{tick}(W - \mathtt{x})}}{\emptyset; \mathtt{t} \vdash \mathtt{for}\ \mathtt{x} \in 0..\mathtt{tid}\ \{\mathtt{A}[0]\} = \mathtt{for}\ \mathtt{x} \in 0..W-1\ \{\mathtt{tick}(W-\mathtt{x})\}}\ \text{FOR}$$

1. Prove that the iteration spaces have same number of iterations (≈)
2. **Prove that the loop bodies have the same cost**

scalar	*active*	*vector*
0	[t,t,t,t]	[0,0,0,0]
1	[f,t,t,t]	[1,1,1,1]
2	[f,f,t,t]	[2,2,2,2]
3	[f,f,f,t]	[3,3,3,3]

Typing context: $\emptyset, x : D_{(0..\mathrm{tid},0..W-1)}$

- The range of x in the scalar/vector contexts
- The active threads

Key takeaways

$$\frac{\emptyset; t \vdash_{=} 0..\mathsf{tid} \approx 0..W-1 : D \qquad \emptyset, x : D_{(0..\mathsf{tid}, 0..W-1)}; t \vdash A[0] = \mathsf{tick}(W-x)}{\emptyset; t \vdash \ \mathbf{for}\ x \in 0..\mathsf{tid}\ \{A[0]\} \ = \ \mathbf{for}\ x \in 0..W-1\ \{\mathsf{tick}(W-x)\}}\ \text{FOR}$$

Our theory is driven by two core ideas:

1. **Loop analysis:** relate iteration spaces (≈ same numer of iterations, ≤ fewer iterations, ≥ more iterations)
2. **Metric analysis:** relate a metric with a symbolic cost (a parameter of the theory)

UMass Boston

Main result

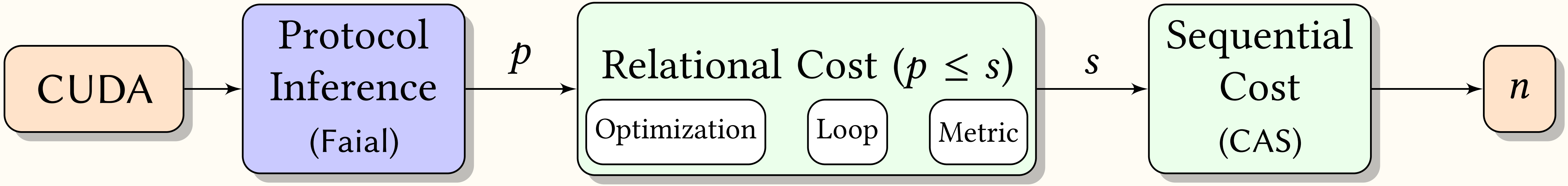
Corollary 4.3 (Soundness for closed terms). Let M be a metric and $\sim$ a cost relation. If $\emptyset; t \vdash p \sim s$, then $(t^W, \emptyset, \emptyset) \vDash p \sim s$.

> If we can derive a **cost statically** for protocol p with program s, then that **cost holds dynamically**.

Pico

Cost-analysis of CUDA kernels

Pipeline



1. **Extract concurrent IR:**
   CUDA ➡ 【 Protocol Inference 】 ➡ p
2. **Extract sequential IR:**
   p ➡ 【 Relational Cost Analysis 】 ➡ s
3. **Extract numeric expression:**
   s ➡ 【 Sequential Cost Analysis 】 ➡ n

Step 1: Extract concurrent IR

Background: Faial static analysis framework

- Concurrent IR is Memory Access Protocol
- Preserves control flow constructs (includes analysis to find loop bounds)
- Supports inter-procedural calls, C++ templates, CUDA memory spaces, atomics, array aliasing, and barrier synchronization
- OOPSLA24 discusses sources and impact of over-approximation in translation

```
(A)
for(x=0;x<=threadIdx.x;x++){
    foo(A[x]);                ➡
}
```

```
(B)
for x ∈ 0..tid{
 rd A[x];
}
```

Step 2: Extract sequential IR

- **Loop analysis:** thread-divergent ranges converted as an optimizer (Z3):
  $0..\text{tid} \Rightarrow 0..W-1$
- **Thread-divergent conditionals:** capture thread-divergent constraints:
  $\text{tid}\%2{=}0$ are added to Φ (active threads)
- **Metric analysis:** support for uncoalesced accesses and bank conflicts
- We develop a **exactness check** to track when over-approximations are introduced (more on this in the evaluation).

```
(B)
for x ∈ 0..tid{
  rd A[x];          ➡
}
```

```
(C)
for x ∈ 0..W-1{ // <- range analysis
  tick (W - x); // <- metric analysis
}
```

Step 3: Extract numeric expression

- **Support for off-the-shelf analysis:** Absynth, KoAT, CoFloCo
- **Direct analysis with a Computer Algebra System (CAS):** we use Maxima; guaranteed correct cost

```
(C)
for x ∈ 0..W-1{
  tick (W - x);
}
```

(D)

$$\Rightarrow \sum_{x=0}^{W-1} (W - x) = (W^2 + W)/2$$

Evaluation

- **RQ1:** How Does Pico Compare to the State of the Art, RaCUDA?
  Reproduce tevaluation of [POPL21] to compare the state-of-the-art tool RaCUDA
- **RQ2:** How Frequently Does Control Flow Affect the Accuracy of Pico?
  Find ratio of precisely analyzable loops and conditons in 226 kernels [CAV14]

RQ1: How Does Pico Compare to the State of the Art, RaCUDA?

Expressiveness: Pico supports / RaCUDA lacks: thread-divergent loops, arbitrary loop steps, bitwise operators, C+++ features (templates, array aliasing, inter-proc analysis)

Reproducibility: Costs of 25 programs (across 2 metrics):

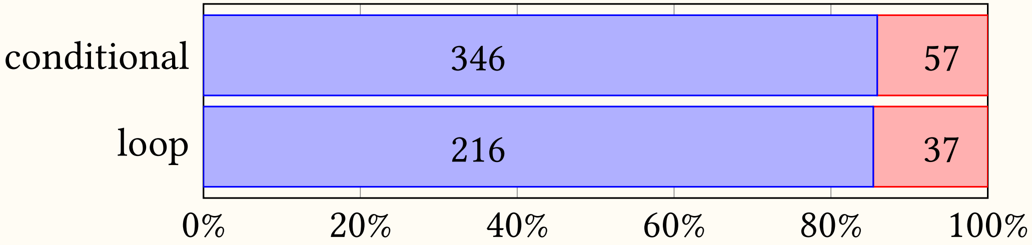
- Pico gave **tighter bound in all 25 examples** (vs 15 of RaCUDA)
- We documented 4 unsound bounds (lower than should be) in RaCUDA

Tool	*Tightest bound*	*Unsound*
RaCUDA	15	4
Pico	**25**	0

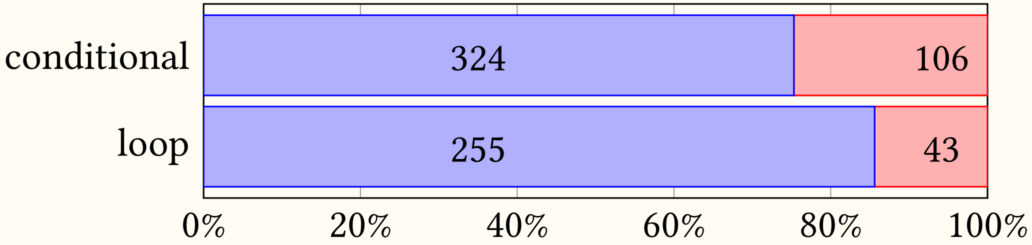
UMass Boston

RQ2: How Frequently Does Control Flow Affect the Accuracy of Pico?

At a glance: 1812 array accesses, 471 structured loops, and 670 conditional expressions



(a) Uncoalesced Access



(b) Bank Conflict

- At least 75.3% of conditionals can be precisely captured
- At least 85.4% loops can be precisely captured

Conclusions

- Relational-cost analysis for warp-parallelism
  support for thread-divergence
- Correctness for any cost relation
  exact (=), over-, and under-approximations (≥).
- Mechanized proofs of all results in Rocq
- Pico achieves the lowest bound
  outperforms RaCUDA in 10 kernels (1.7× better)

Conclusions

- Relational-cost analysis for warp-parallelism
  support for thread-divergence
- Correctness for any cost relation
  exact (=), over-, and under-approximations (≥).
- Mechanized proofs of all results in Rocq
- Pico achieves the lowest bound
  outperforms RaCUDA in 10 kernels (1.7× better)

Future work

- Correctness of metrics
  (tradeoff performance vs precision)
- How to report **helpful** performance bottlenecks?
  (average costs, differential analysis)
- Relational-cost between protocols
  (eg, for program repair)

Extra slides

Memory Access Protocol + Warp Semantics

Contribution 1: Concurrent IR

Syntax

A protocol p is:

- skip no-op
- $A[n]$ access index n and an array A
- $p; p$ sequential composition
- $\text{if}(c)\ p$ branching
- $\text{for}\ x \in r\ p$ looping

- Our contribution is the semantics for Warps
- We reuse the same syntax of Memory Access Protocols

Warp Semantics

A semantics of vectors

Rules

$$\text{Judgment:} \quad \langle n, \sigma \rangle \Downarrow I$$

$$\langle i, \sigma \rangle \Downarrow i^{\mathrm{W}} \quad \text{(E-num)}$$

$$\langle \mathrm{tid}, \sigma \rangle \Downarrow (0, 1, \ldots, \mathrm{W} - 1) \quad \text{(E-tid)}$$

Syntax

- Vector of naturals: I
- Vector store: σ

- Naturals: $i ::= 0 \mid 1 \mid \ldots$
- Numeric expressions: $n ::= i \mid x \mid \mathrm{tid} \mid n \star n$
- Booleans: $b ::= \mathrm{t} \mid \mathrm{f}$
- Boolean expressions: $c ::= b \mid n \diamond n \mid b \circ b$

UMass Boston

Examples of vectorized expressions

Example 1

$$\langle 2 \times \mathrm{tid}, \sigma \rangle \Downarrow (0, 2, 4, 6)$$

$$\text{since } (2, 2, 2, 2) \times (0, 1, 2, 3) = (2 \times 0, 2 \times 1, 2 \times 2, 2 \times 3) = (0, 2, 4, 6)$$

Example 2

$$\langle 1 \leq \mathrm{tid}, \sigma \rangle \Downarrow (\mathrm{f}, \mathrm{t}, \mathrm{t}, \mathrm{t})$$

$$\text{since } (1, 1, 1, 1) \leq (0, 1, 2, 3) = (1 \leq 0, 1 \leq 1, 1 \leq 2, 1 \leq 3) = \mathrm{fttt}$$

Protocol semantics

$$\langle p, B, \sigma \rangle \Downarrow_M i$$

1. A protocol `p` to run
2. A vector of active threads B
3. A store of vectors σ
4. A cost i

$$\frac{\langle c, \sigma \rangle \Downarrow C \quad C \neq \mathrm{f}^{\mathrm{W}} \quad \langle p, B \wedge C, \sigma \rangle \Downarrow i}{\langle \mathrm{if}\ (c)\ p, \sigma \rangle \Downarrow i}$$