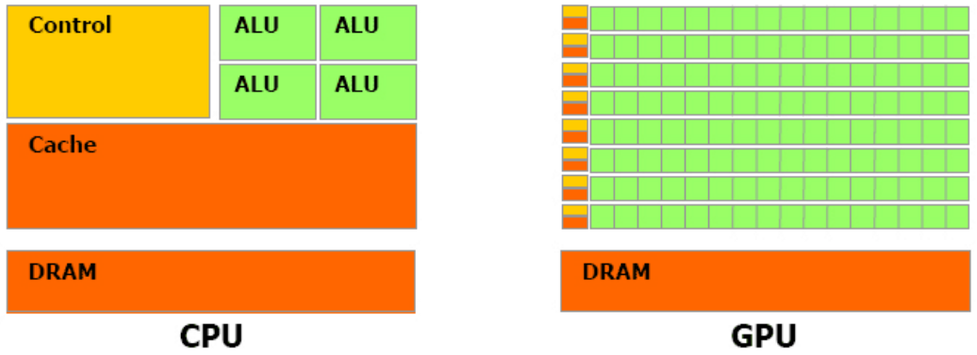
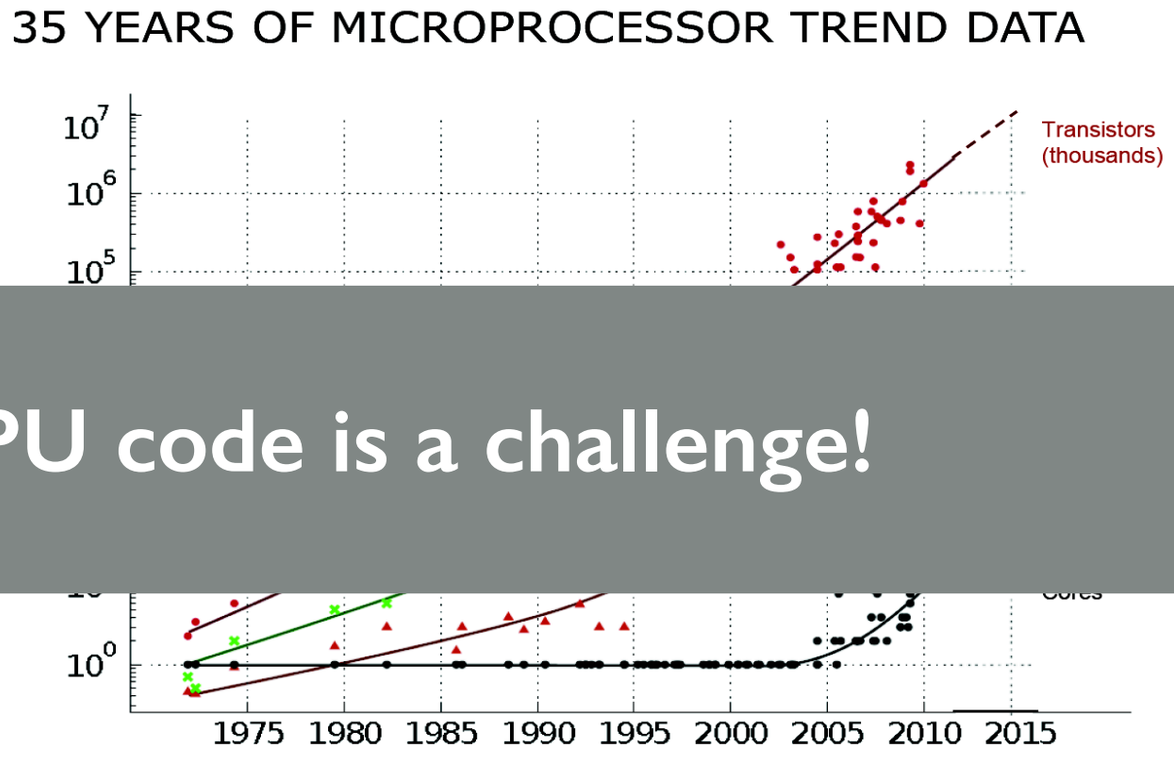
GPU Drano: Detecting Uncoalesced Accesses in GPU Programs

Rajeev Alur, Joe Devietti, Omar Navarro Leija, **Nimit Singhania**
University of Pennsylvania

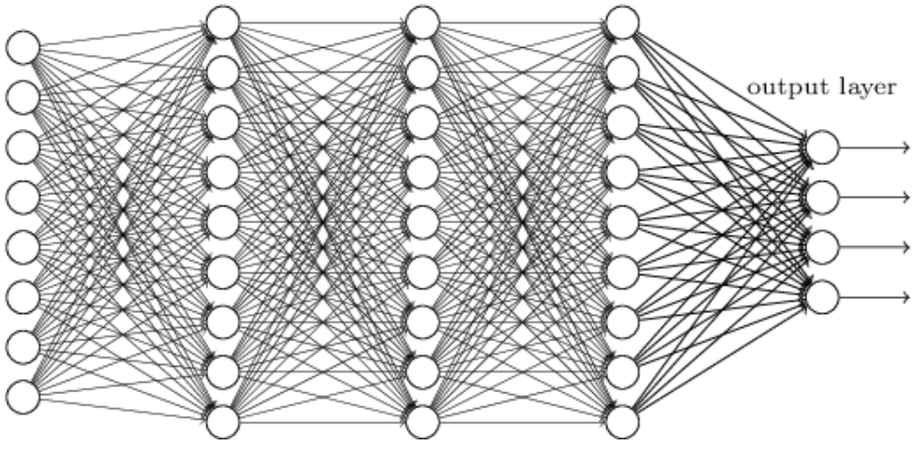
Computer Aided Verification 2017
Heidelberg, Germany

GPU: Next trend in computing





Writing efficient GPU code is a challenge!



Deep Learning

Scalable

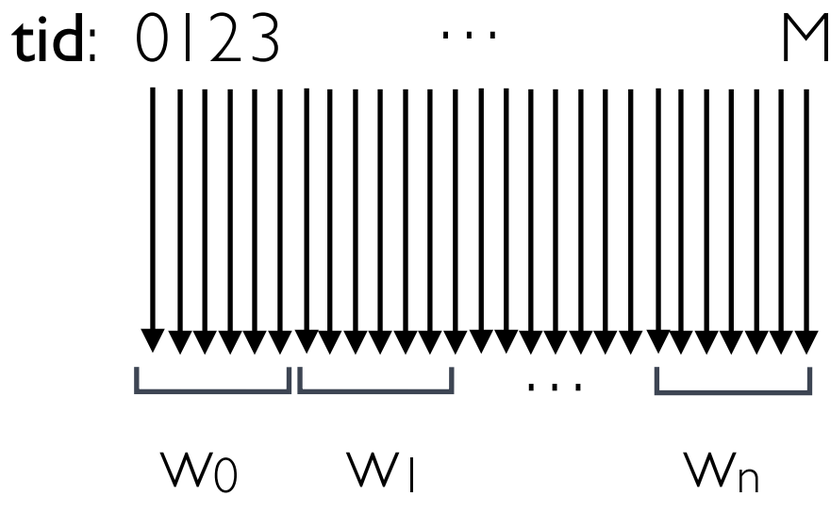
GPU Drano

light-weight
compile-time

A *static analysis* to detect
uncoalesced accesses in GPU programs

Performance bug

GPU Programming Model (simplified)




```
x = tid;

for(y=0; y<N; y++){
  idx = N*x + y;
  PSum[x] += A[idx];
}

if(x == 0){
  for(y=0; y<N; y++){
    Sum += PSum[y];
  }
}
```

local

global

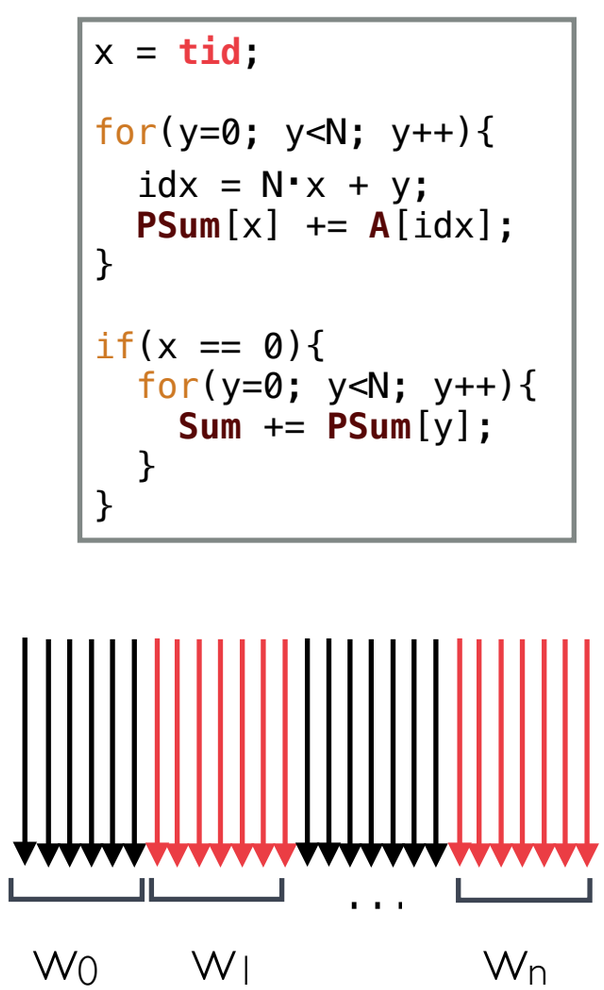
Threads

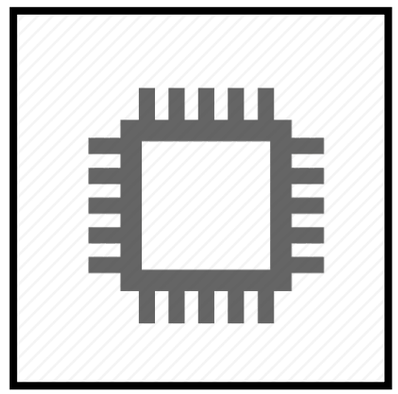
Kernel

Single Instruction Multiple Threads Model

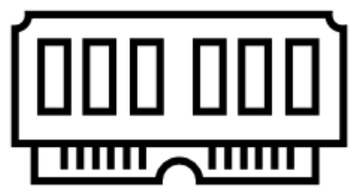
GPU Execution Model (simplified)

```
x = tid;

for(y=0; y<N; y++){
  idx = N*x + y;
  PSum[x] += A[idx];
}

if(x == 0){
  for(y=0; y<N; y++){
    Sum += PSum[y];
  }
}
```





GPU

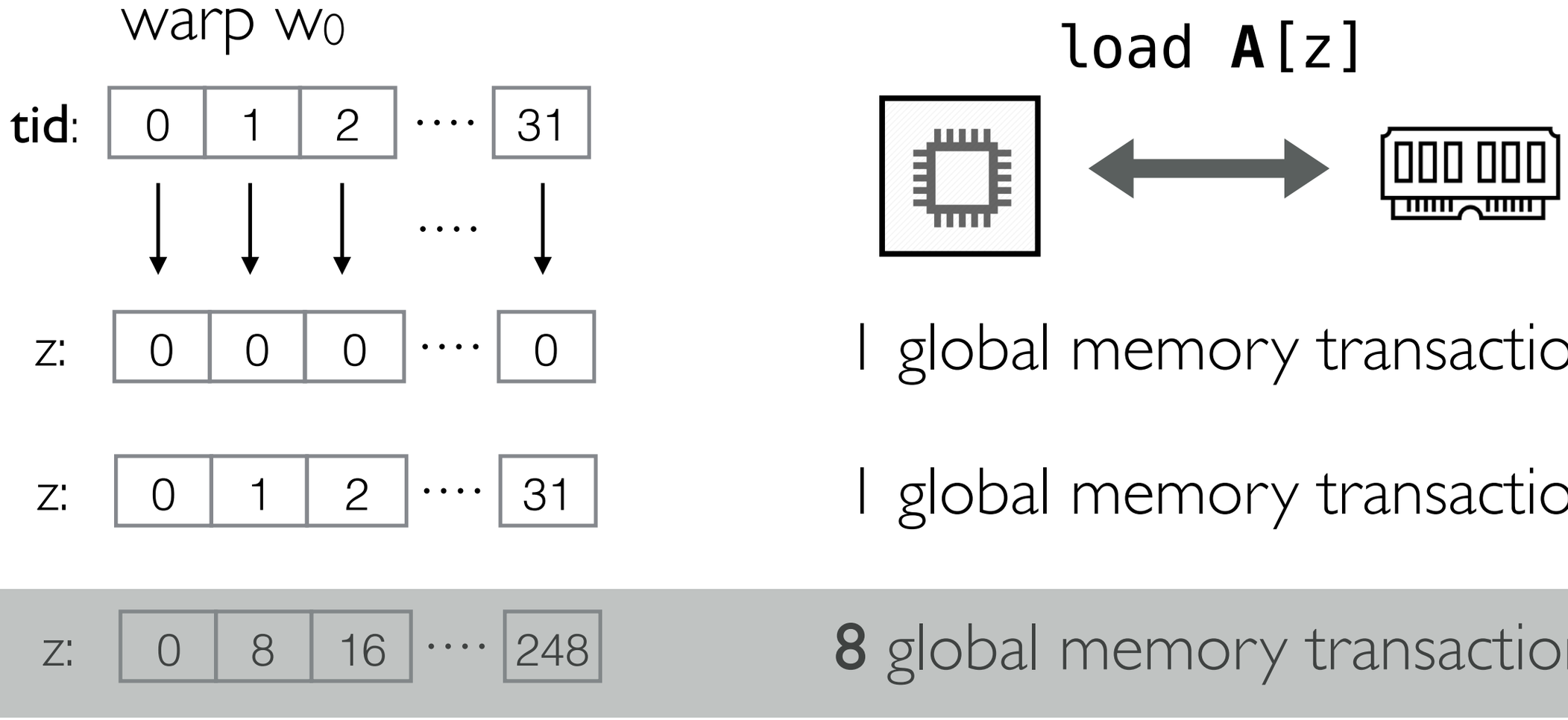


Global Memory

high latency,
high bandwidth

Threads in a warp execute in lock-step

Uncoalesced Accesses



Uncoalesced

Property of the execution of a warp

Example

```
x = tid;

for(y=0; y<N; y++){
   z = N*x + y;
   PSum[x] += A[z];
}

if(x == 0){
   for(y=0; y<N; y++){
      Sum += PSum[y];
   }
}
```

warp w_0

tid:	0	1	2	….	31
N:	32	32	32	….	32
x:	0	1	2	….	31
y:	0	0	0	….	0
z:	0	32	64	….	992

PSum

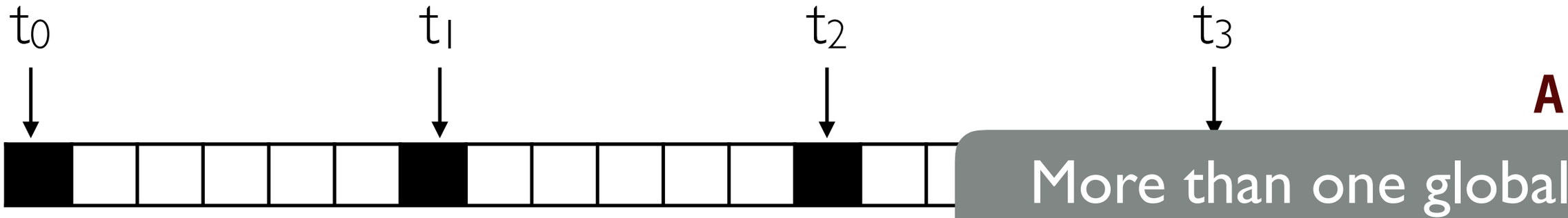
One global memory transaction

Access **A**[z] is uncoalesced

```
x = tid;

for(y=0; y<N; y++){
   z = N*x + y;
   PSum[x] += A[z];
}

if(x == 0){
   for(y=0; y<N; y++){
      Sum += PSum[y];
   }
}
```

warp w_0

tid:	0	1	2	….	31
N:	32	32	32	….	32
x:	0	1	2	….	31
y:	0	0	0	….	0
z:	0	32	64	….	992



More than one global memory transaction

Example: Fixing the bug

```
x = tid;

for(y=0; y<N; y++){
  z = N*x + y;
  PSum[x] += A[z];
}

if(x == 0){
  for(y=0; y<N; y++){
    Sum += PSum[y];
  }
}
```

Original program

```
y = tid;

for(x=0; x<N; x++){
  z = N*x + y;
  PSum[y] += A[z];
}

if(y == 0){
  for(x=0; x<N; x++){
    Sum += PSum[x];
  }
}
```

Fixed program

Improves performance by orders of magnitude

GPU Drano

- Static Analysis to identify Uncoalesced Accesses
- Property of execution of a warp
- Amenable to static analysis
  - Regular patterns exhibited by index variables

Dependence on **tid**

- load **A**[z] — Abstraction
  - z = constant — 

 — **0**
  - z = **tid** + constant — 

 — **I**
  - z = -**tid** + constant — 

 — **-I**
  - z = f(**tid**) (say k.**tid** + constant) — T

Uncoalesced

Abstract Execution

- Abstract Semantics defined for the abstraction
- Intra-procedural analysis
- Variables initialized to **0**
- Program executed with abstract semantics
- Accesses with index **T** labelled as *uncoalesced*

Example: Static Analysis

```
x = tid;

for(y=0; y<N; y++){
  z = N*x + y;
  PSum[x] += A[z];
}

if(x == 0){
  for(y=0; y<N; y++){
    Sum += PSum[y];
  }
}
```

N → 0

x → 1

y → 0

z → 0*1 + 0 = T + 0 = T

*0*1 = (c)*(**tid** + c) = T*

Access **A**[**z**] labelled uncoalesced!

Boolean Abstraction

```
x = tid
if(x == 0){
    sum += A[N·x]
}
```

x → 1

N.x → 0*1 = T

Executed only for **tid** = 0

Dependence of boolean predicates on **tid**

(**tid**-independent), (**tid** == c), (**tid** != c), and **T**

Accesses executed by a single thread ?

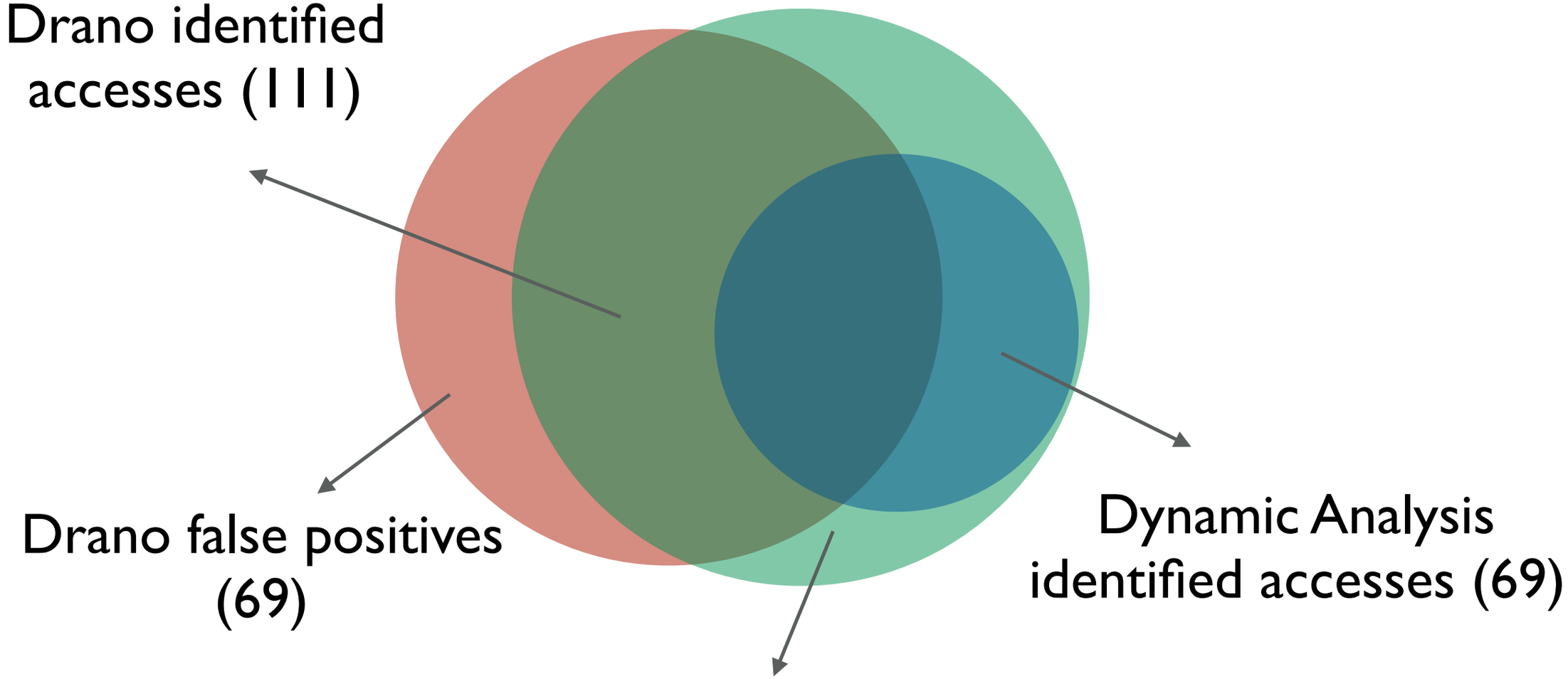
Evaluation

- Implemented in gpucc, an LLVM-based CUDA compiler
- Comparison with Dynamic Analysis implementation
- Evaluated on Rodinia benchmark suite
  - 22 GPU programs from diverse domains

Dynamic Analysis

- Executes programs and records number of transactions required for each global memory access
- Reports *average* number of transactions per access
- Gives precise results; no false positives
- Misses bugs along unexecuted paths and uncalled kernels

Evaluation Results



False Positives

```
row =  tid / N;
col =  tid % N;

p = N*row + col; // tid
y = A[p];
```

row → 1

col → 1

p → 0*1 + 1 = T

Imprecision in tracking constants (Heartwall)

```
if (x == N)
  neighbor[x] = N;
else
  neighbor[x] = x + 1;
```

```
p = neighbor[tid];
y = A[p];
```

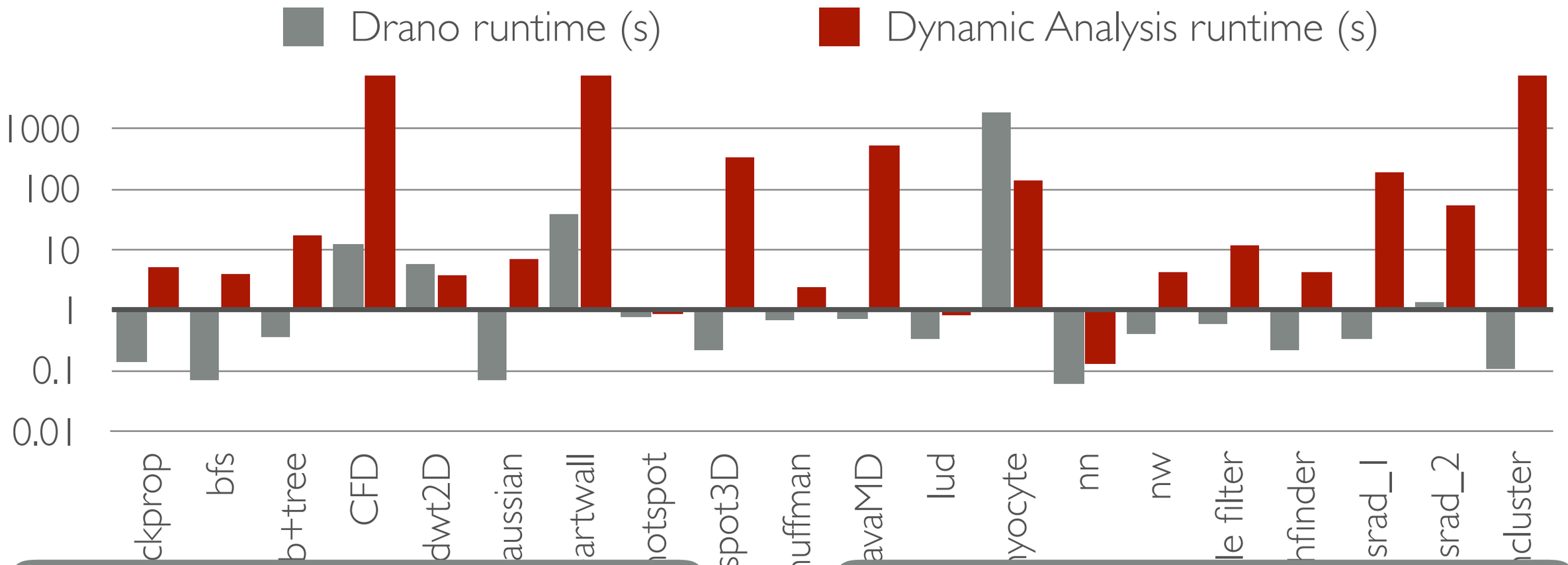
p → T

Imprecision in tracking arrays (CFD)

Evaluation: Runtime

GPU Drano takes less than one seconds for most programs.

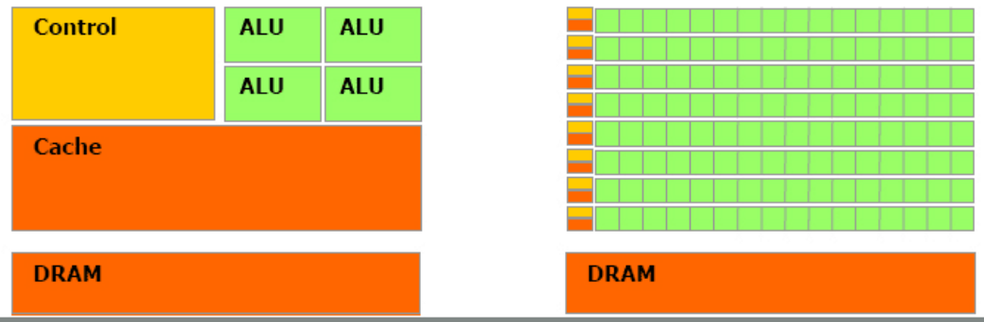
Scales to 1000s of lines of code.

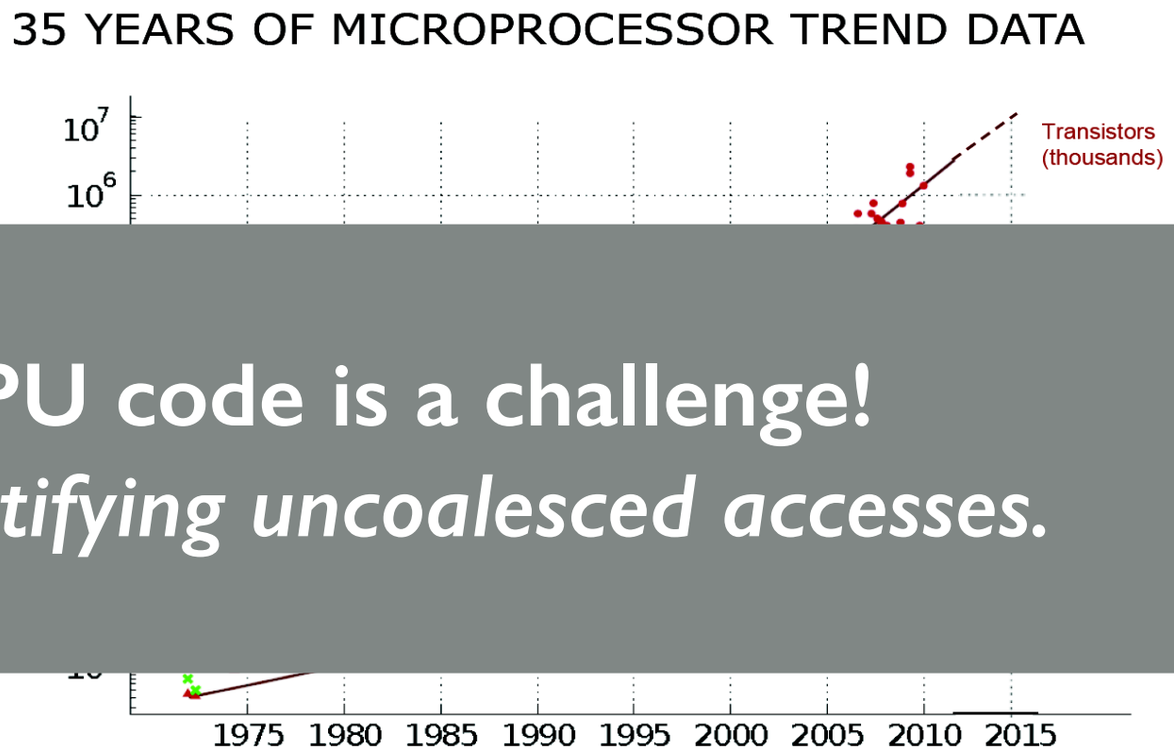


Takes 1/2 hr for Myocyte due to large number of nested loops.

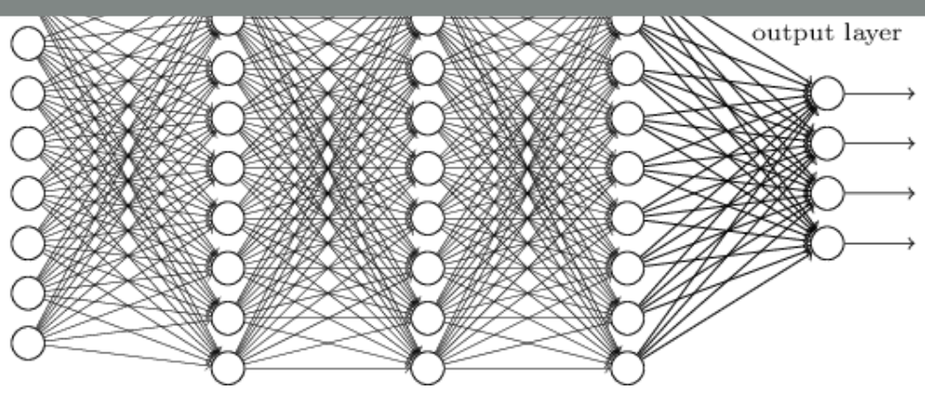
Dynamic Analysis takes 100s of seconds.

Programming Tools for GPUs





Writing efficient GPU code is a challenge!
GPU Drano helps by identifying uncoalesced accesses.



Deep Learning

Scalable

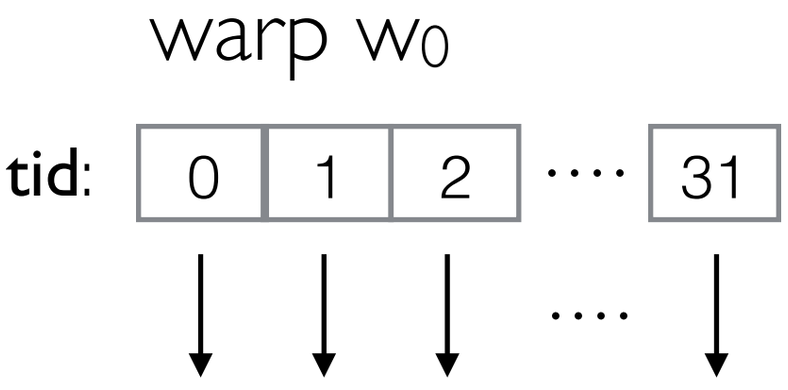
Thank you!

Backup Slides

Existing Tools

- Analysis tools
  - Dynamic analysis: heavy instrumentation necessary
  - Static analysis: slow, focused on correctness
- Optimization tools
  - Semi-automatic: complex loop transformations, hints from programmer necessary

Alignment Bugs



x: 0 1 2 ···· 31 — 1 global memory transaction

x: **4** **5** **6** ···· **35** — **2** global memory transactions

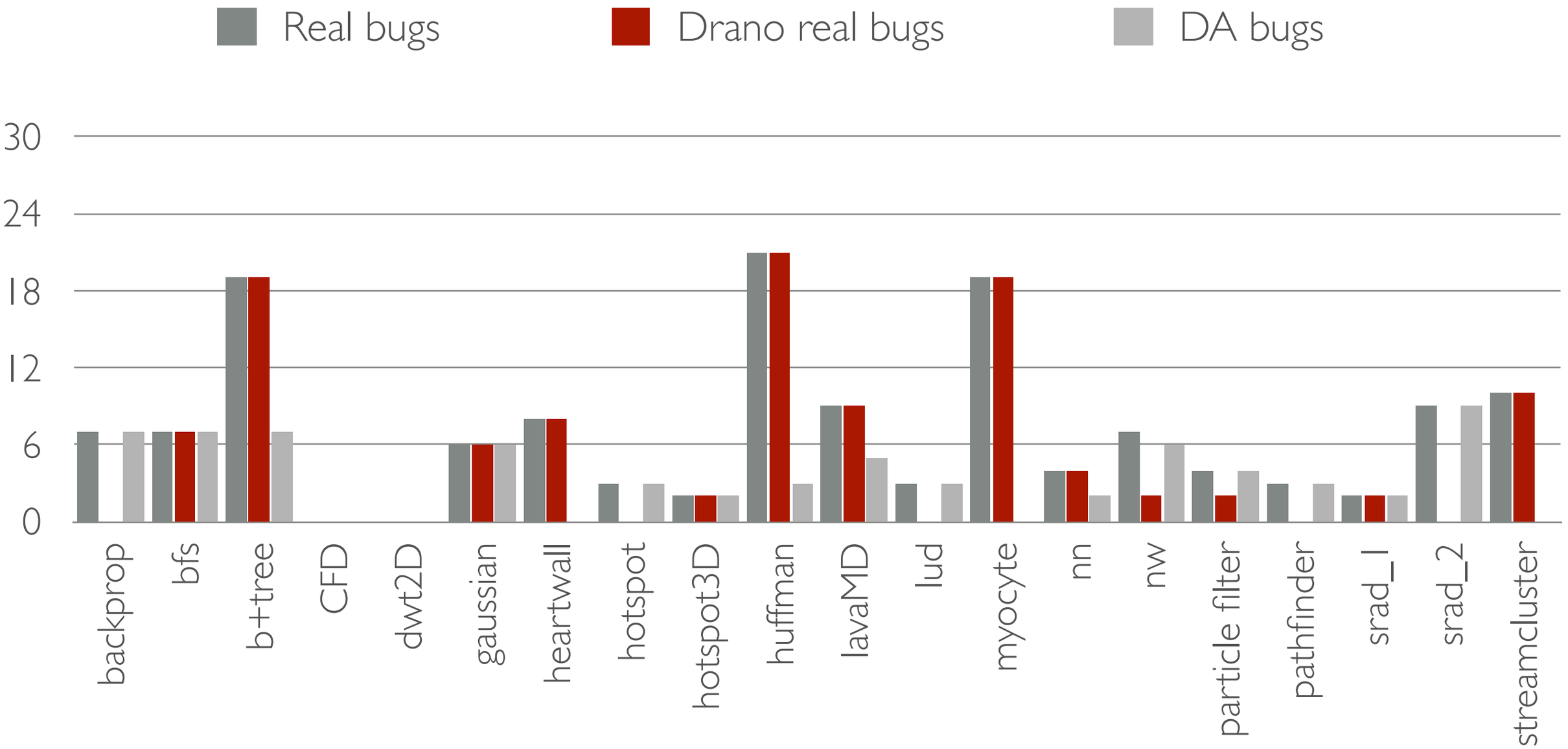
Uncoalesced

Alignment with memory-block boundary
necessary for coalesced access

Formalization

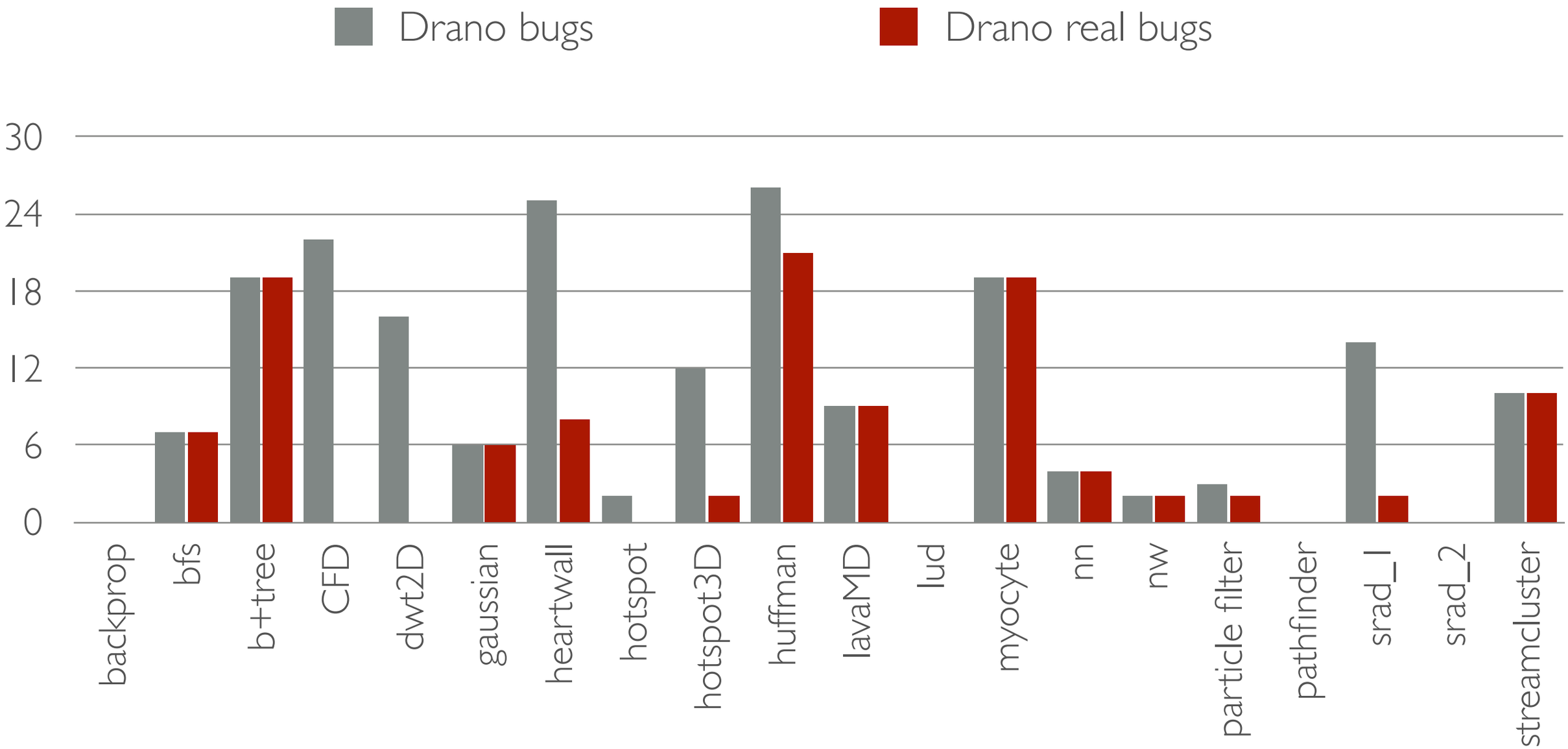
- Formalization of Uncoalesced Accesses
  - Necessary to formally define the analysis
- Correctness guarantees for the analysis
  - **All non-alignment uncoalesced accesses are captured by GPU Drano**

Evaluation: Recall



111 out of 143 real bugs caught by GPU Drano

Evaluation: Precision



111 out of 180 reported bugs by GPU Drano are real