Memory Access Protocols:

Certified Data-Race Freedom for GPU Kernels

Tiago Cogumbreiro, UMass Boston

Joint work with

Julien Lange, Royal Holloway, University of London

Dennis Liew, UMass Boston

Hannah Zicarelli, UMass Boston

September 26, 2023

Boston University

Today's talk

Impact (Part 1)

- why GPUs
- what makes static analysis of GPU programs unique
- a static DRF analysis for GPU programs
- more robust, more scalable in largest comparative study of its kind

Theoretical contributions (Part 2)

- a novel analysis of data-race freedom
- a formalization of such analysis using a proof assistant

Published works

- Checking Data-Race Freedom of GPU Kernels, Compositionally **(CAV'21)**
- Memory Access Protocols: Certified Data-Race Freedom for GPU Kernels **(FMSD'23)**

Motivation

Why do GPUs matter?

GPUs are everywhere

GPUs are a computing cornerstone of scientific advancement

GPUs in High Performance Computing (HPC)

Power 8 out of 10 of the Top 10 super computers

	Name	*GPU*
1	Supercomputer Fugaku	☐
2	Summit	☑
3	Sierra	☑
4	Sunway TaihuLight	☐
5	Selene	☑
6	Tianhe-2A	☑
7	JUWELS Booster Module	☑
8	HPC5	☑
9	Frontera	☑
10	Dammam-7	☑

www.top500.org/lists/top500/2020/11/highs/

Credit: Carlos Jones/ORNL

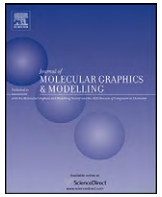
GPUs powering chemistry

Journal of Molecular Graphics and Modelling 29 (2010) 116–125

Contents lists available at ScienceDirect

Journal of Molecular Graphics and Modelling

journal homepage: www.elsevier.com/locate/JMGM



Topical perspectives

GPU-accelerated molecular modeling coming of age

John E. Stone[a], David J. Hardy[a], Ivan S. Ufimtsev[b], Klaus Schulten[c,*]

[a] Beckman Institute, University of Illinois at Urbana-Champaign, 405 N. Mathews Ave., Urbana, IL 61801, United States
[b] Department of Chemistry, Stanford University, 333 Campus Drive, Stanford, CA 94305, United States
[c] Department of Physics, University of Illinois at Urbana-Champaign, 1110 W. Green, Urbana, IL 61801, United States

ARTICLE INFO

Article history:
Received 9 February 2010
Received in revised form 24 June 2010
Accepted 30 June 2010
Available online 8 July 2010

Keywords:
GPU computing
Molecular modeling
Molecular dynamics
Quantum chemistry
Molecular graphics

ABSTRACT

Graphics processing units (GPUs) have traditionally been used in molecular modeling solely for visualization of molecular structures and animation of trajectories resulting from molecular dynamics simulations. Modern GPUs have evolved into fully programmable, massively parallel co-processors that can now be exploited to accelerate many scientific computations, typically providing about one order of magnitude speedup over CPU code and in special cases providing speedups of two orders of magnitude. This paper surveys the development of molecular modeling algorithms that leverage GPU computing, the advances already made and remaining issues to be resolved, and the continuing evolution of GPU technology that promises to become even more useful to molecular modeling. Hardware acceleration with commodity GPUs is expected to benefit the overall computational biology community by bringing teraflops performance to desktop workstations and in some cases potentially changing what were formerly batch-mode computational jobs into interactive tasks.

doi:10.1016/j.jmgm.2010.06.010

GPUs powering biology

BRIEFINGS IN BIOINFORMATICS. VOL 11. NO 3. 323–333
Advance Access published on 7 March 2010
doi:10.1093/bib/bbq006

GPU computing for systems biology

Lorenzo Dematté and Davide Prandi

Submitted: 20th November 2009; Received (in revised form): 30th January 2010

Abstract

The development of detailed, coherent, models of complex biological systems is recognized as a key requirement for integrating the increasing amount of experimental data. In addition, in-silico simulation of bio-chemical models provides an easy way to test different experimental conditions, helping in the discovery of the dynamics that regulate biological systems. However, the computational power required by these simulations often exceeds that available on common desktop computers and thus expensive high performance computing solutions are required. An emerging alternative is represented by general-purpose scientific computing on graphics processing units (GPGPU), which offers the power of a small computer cluster at a cost of ~$400. Computing with a GPU requires the development of specific algorithms, since the programming paradigm substantially differs from traditional CPU-based computing. In this paper, we review some recent efforts in exploiting the processing power of GPUs for the simulation of biological systems.

Keywords: *systems biology; simulation; agent-based modelling; cellular automata; GPGPU; CUDA*

doi:10.1093/bib/bbq006

GPUs power the AI revolution

Autoware.AI

> Autoware.AI is the world's first "All-in-One" open-source software for autonomous driving technology.

22 code results in Autoware-AI/core_perception Sort: Best match

ndt_gpu/src/MatrixDevice.cu
Cuda Last indexed on Oct 15, 2020

ndt_gpu/src/SymmetricEigenSolver.cu
Cuda Last indexed on Oct 15, 2020

vision_darknet_detect/darknet/src/dropout_layer_kernels.cu
Cuda Last indexed on Oct 15, 2020

vision_darknet_detect/darknet/src/col2im_kernels.cu
Cuda Last indexed on Oct 15, 2020

Why we should care about static verification of GPU programs?

GPU programming, a primer

① High-level of parallelism at a reduced cost

(faster processing, lower cost, reduced power consumption)

② Techniques designed for CPUs do not work for GPUs

(hardware assumptions differ: memory available, execution model)

③ GPUs are difficult to program and debug

GPU programming is difficult

- high degree of parallelism (up to tens of thousand of threads)
- high degree of concurrency (up to 1,024 threads accessing the same array)
- **unconstrained** access to a shared memory (no locks)
- thousands of threads indexing disjoint portions of arrays
- devices are memory constrained (affects debugging techniques)

GPU program example

```
for (int r = 0; r < N; r++) {
  for (int i = 0; i<TILE_DIM; i+=BLOCK_ROWS)
   { tile [tid.y+i][tid.x] = idata [index_in+i*width];}
  __syncthreads();
  for (int j = 0; j<TILE_DIM; j+=BLOCK_ROWS)
   { odata[index_out+j*height] = tile [tid.x][tid.y+j];}}
```

Source:

- Optimizing matrix transpose in CUDA. NVIDIA CUDA SDK Application Note 18 (2009)

Also in:

- Padding free bank conflict resolution for CUDA-based matrix transpose algorithm. DOI: 10.1109/SNPD.2014.6888709

GPU program example

```
for (int r = 0; r < N; r++) {
  for (int i = 0; i<TILE_DIM; i+=BLOCK_ROWS)
   { tile [tid.y+i][tid.x] = idata[index_in+i*width];}
  __syncthreads();
  for (int j = 0; j<TILE_DIM; j+=BLOCK_ROWS)
   { odata[index_out+j*height] = tile [tid.x][tid.y+j];}}
```

GPU program example

thread (0,1)

```
for (int r = 0; r < N; r++) {
  for (int i = 0; i<TILE_DIM; i+=BLOCK_ROWS)
   { tile [ 0 +i ][ 1 ] = idata [index_in+i*width ]; }
  __syncthreads();
  for (int j = 0; j<TILE_DIM; j+=BLOCK_ROWS)
   { odata[index_out+j*height] = tile [ 0 ][ 1 +j ];}}
```

thread (1,0)

```
for (int r = 0; r < N; r++) {
  for (int i = 0; i<TILE_DIM; i+=BLOCK_ROWS)
   { tile [ 1 +i ][ 0 ] = idata [index_in+i*width ]; }
  __syncthreads();
  for (int j = 0; j<TILE_DIM; j+=BLOCK_ROWS)
   { odata[index_out+j*height] = tile [ 1 ][ 0 +j ];}}
```

GPU data-races

Data-race

- Two threads accessing the same array index concurrently
- At least one thread writing

Data-Race Freedom (DRF) analysis

Show that for all possible inputs and executions a program is absent of data-races.

A trivial data-race example (every thread writes to position 0)

```
A[0] = 1;
```

GPU program example

```
for (int r = 0; r < N; r++) {
  for (int i = 0; i<TILE_DIM; i+=BLOCK_ROWS)
   { tile [tid.y+i][tid.x] = idata [index_in+i*width];}
  __syncthreads();
  for (int j = 0; j<TILE_DIM; j+=BLOCK_ROWS)
   { odata[index_out+j*height] = tile [tid.x][tid.y+j];}}
```

> Exhibits a data-race: the code after `__syncthreads()` of iteration `i + 1` runs concurrently with the code before `__syncthreads()` of iteration `i`.

- Outer loops is used to measure the benefit of an optimization
- Data-race **corrupts** the data in the array and affects the time measurements

Contributions

Contributions

Impact (Part 1)

- a static DRF analysis for GPU programs
- more robust, more scalable in largest comparative study of its kind

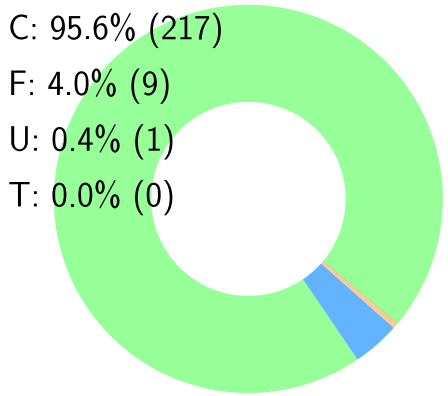
Theoretical contributions (Part 2)

- a novel analysis of data-race freedom
- a formalization of such analysis using a proof assistant

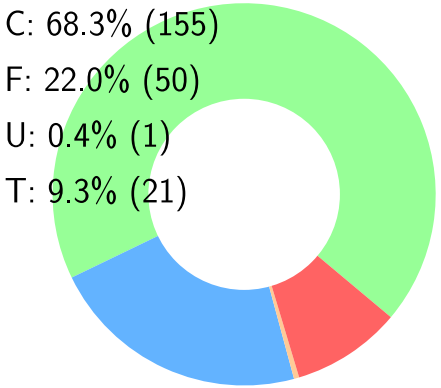
Lowest false-positive rate

- Dataset of 227 data-race free real-world kernels
- Can verify 41% more kernels than others

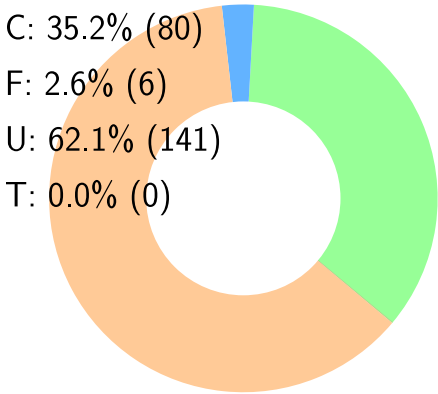
Faial (our tool)



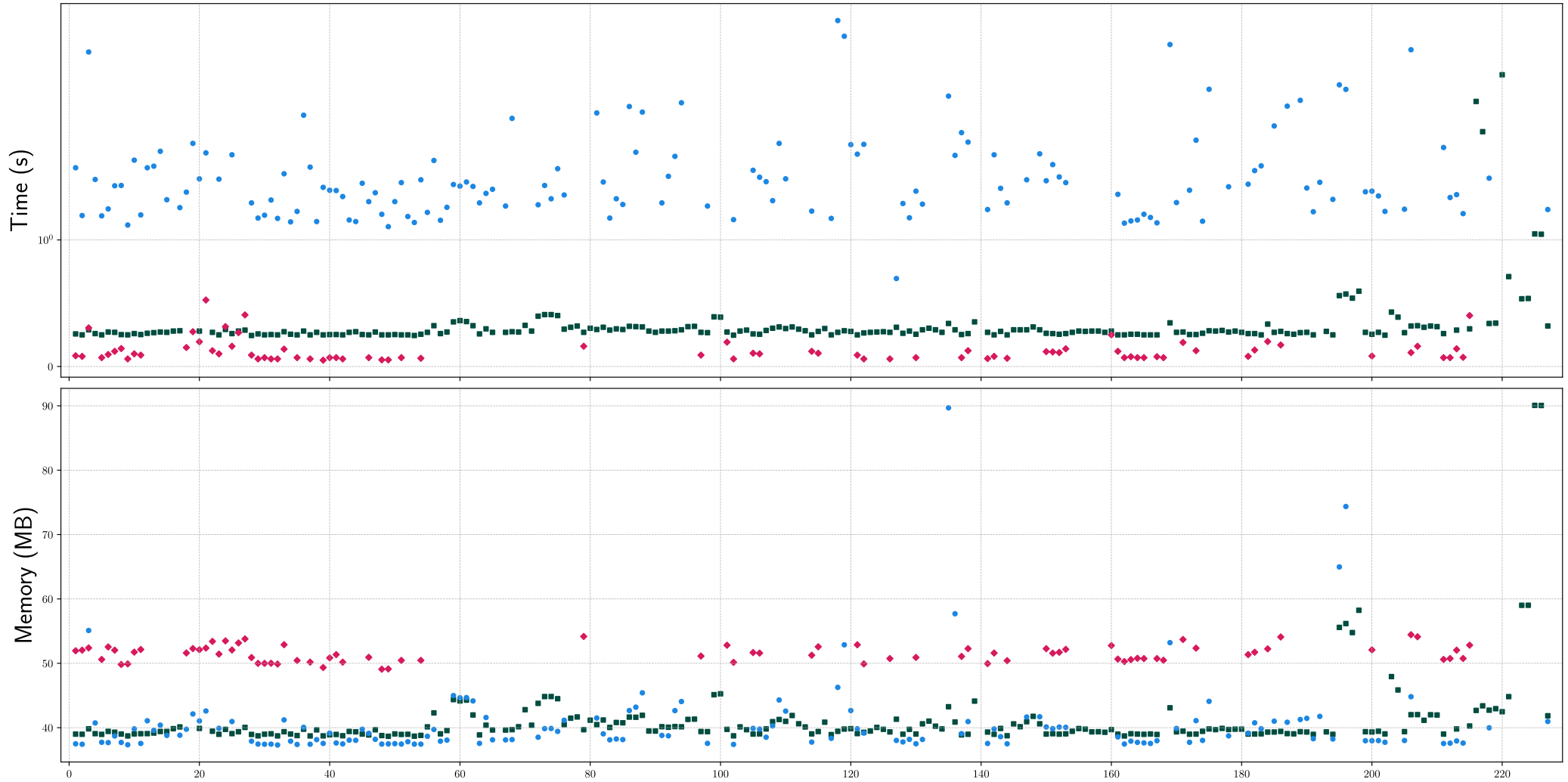
GPUVerify



PUG

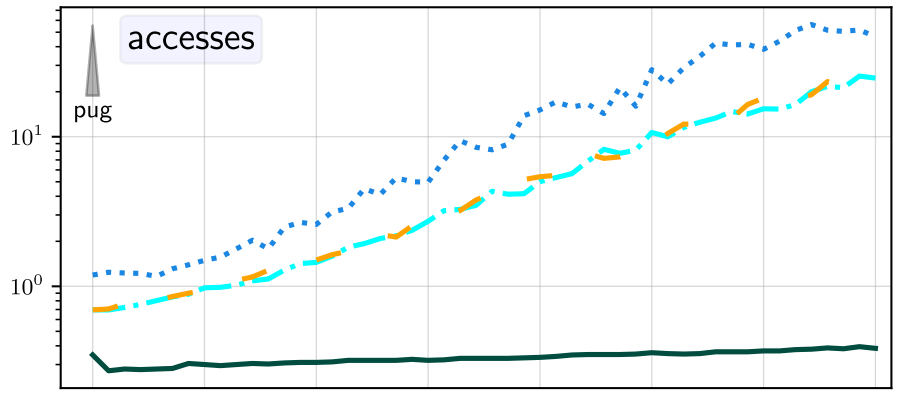


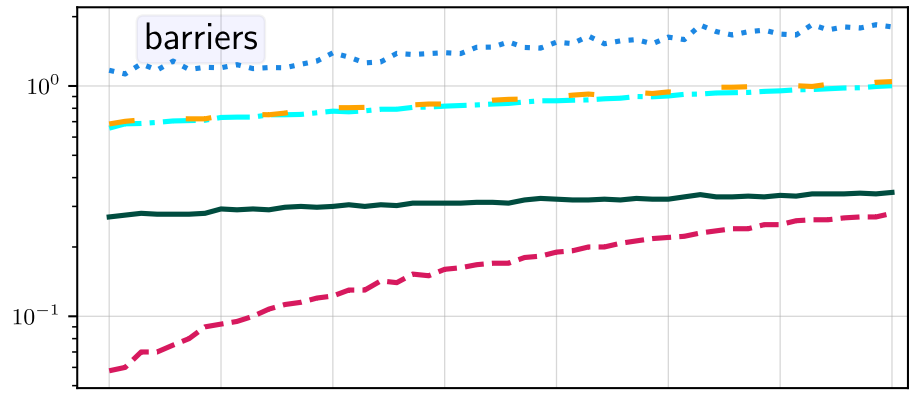
Best compromise time/memory

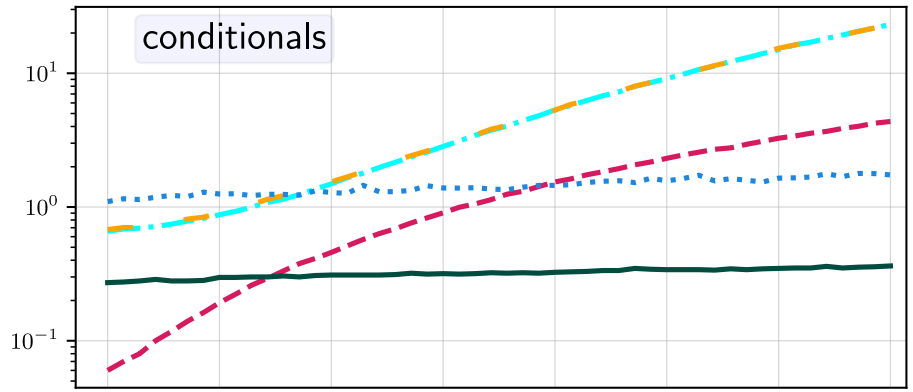


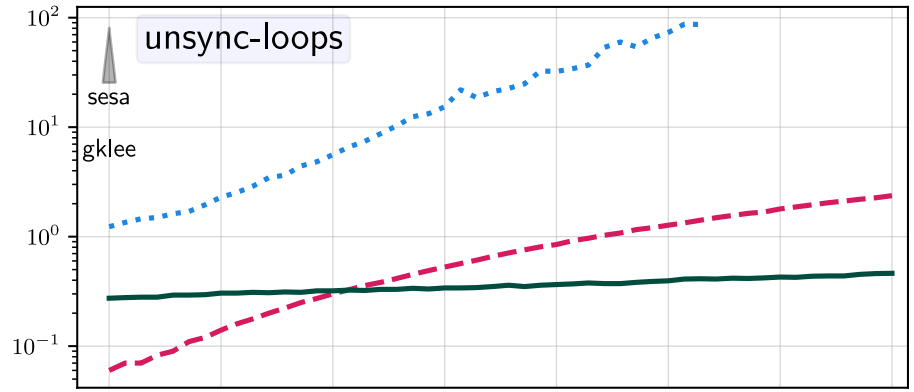
Highest scalability

- Vary the number of constructs from 1 to 50 (250 kernels in total)
- Out of 5 tools, the only that **scales linearly** (time) (PUG, GPUVerify, GKlee, SESA)



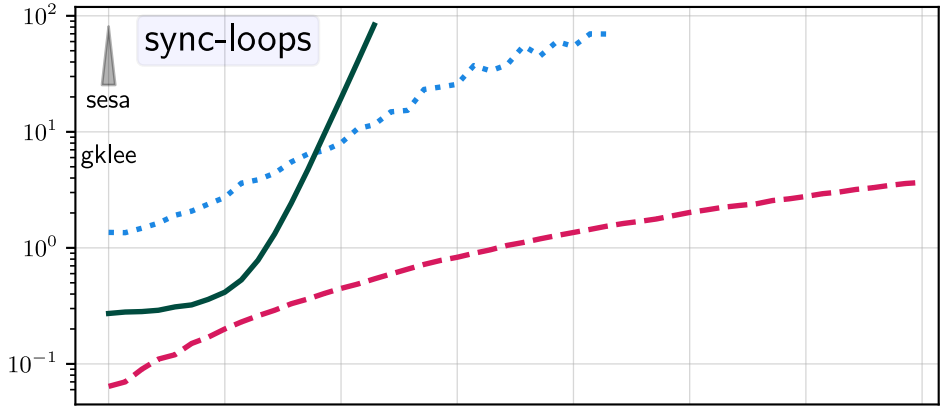






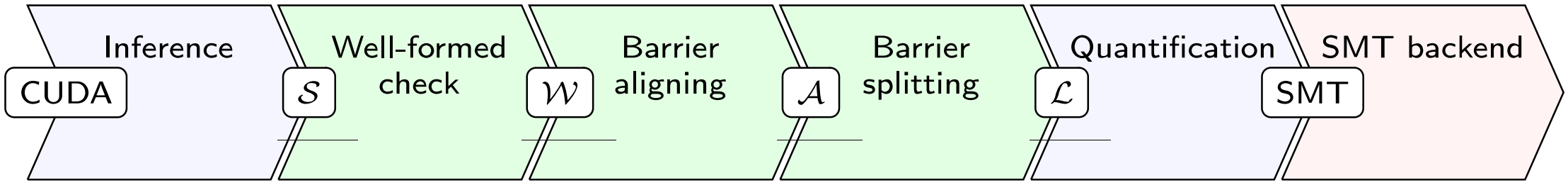
Limitations of our analysis

- Cannot handle more than 13 nested synchronized loops
- 3rd out of 5 tools
- We found a maximum nesting level of 3 in our experiments



Demo

Theoretical contributions



- Property: **DRF analysis (in green) is proved sound & complete** (Theorem 1)
- Technique: A behavioral type (syntax+semantics)
- Artifact: Mechanized proofs using the Coq proof assistant (18,000 LOC)

Remainder of the talk

- Present our main result
- Introduce a motivating example
- Detail our analysis
  1. Align protocols
  2. Split protocols
  3. Sequentialize protocols

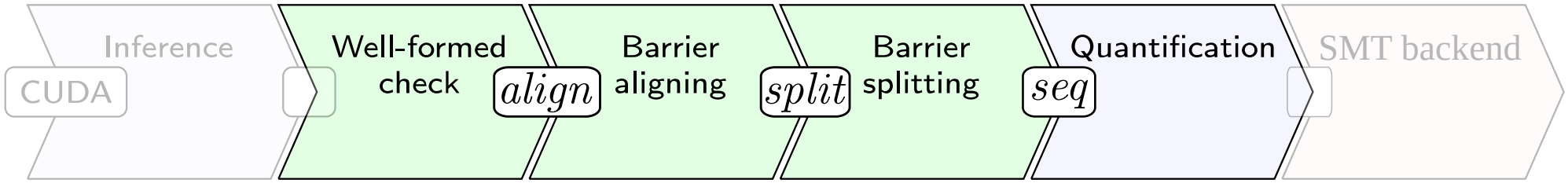
Main result

Let `safe(H)` mean that `H` is data-race free.

Theorem 1. *If* $p \downarrow H_1$ *and* $seq(split(align(p))) \Downarrow H_2$, *then* $safe(H_1)$ *if and only if* $safe(H_2)$.

- Analysis steps `seq`, `split`, `align` are sound ***and*** complete, wrt the DRF property
- The over-approximations happen ***before*** MAP is created (protocol inference)
- We further show that the set of concurrent accesses is preserved (more general)

Theorem 1. *If* $p \downarrow H_1$ *and* $seq(split(align(p))) \Downarrow H_2$, *then* $safe(H_1)$ *if and only if* $safe(H_2)$.



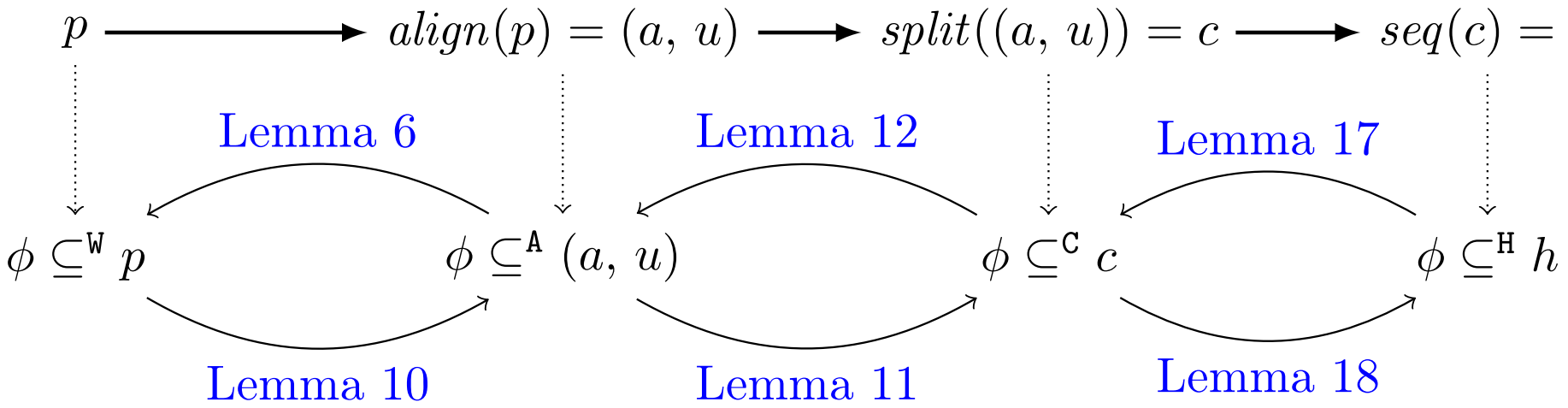
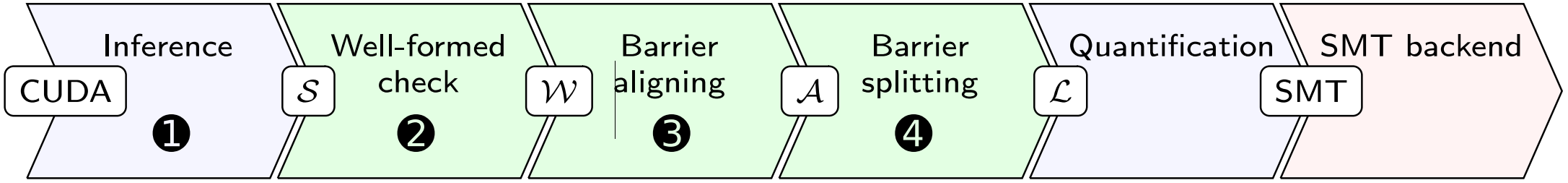


Fig. 9: Overview of the key lemmas used to establish Theorem 1.

Simplified running example

- A CUDA example, which simplifies our initial example
- Exhibits the same root cause (data-race)

```
1  for (int r = 0; r < N; r++) {
2    for (int i = 0; i<M; i++)
3      { tile [tid] = ...;}
4    __syncthreads();
5    for (int j = 0; j<M; j++)
6      {... = tile [tid+j];}}
```




```
for (int r = 0; r < N; r++) {
  for (int i = 0; i<M; i++)
    { tile[tid] = ...;}
  __syncthreads();
  for (int j = 0; j<M; j++)
    {... = tile[tid+j];}}
```
❶

```
for^S r in 0..N {
  for^U i in 0..M { wr[tid] }
  sync;
  for^U j in 0..M { rd[tid + j] }
}
```
❷

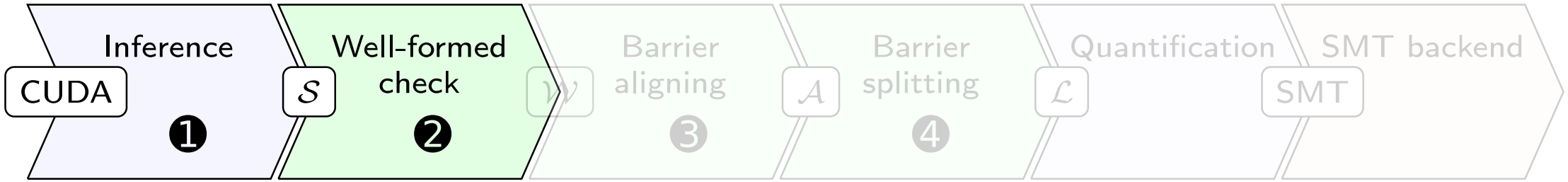
```
for^U i in 0..M { wr[tid] }
sync;
for^S r in 1..N {
  for^U j in 0..M { rd[tid + j] }
  for^U i in 0..M { wr[tid] }
  sync; }
for^U j in 0..M { rd[tid + j] }
```
❸

$$\forall r, j_1, i_1, j_2, i_2 \colon\ 1 \le r < N \ \wedge 0 \le j_1 < M \wedge 0 \le i_1 < M \wedge 0 \le j_2 < M \wedge 0 \le i_2 < M$$
$$\implies \{\mathsf{rd}[t_1 + j_1]\} \cup \{\mathsf{wr}[t_1]\} \quad \textit{DRF with?} \quad \{\mathsf{rd}[t_2 + j_2]\} \cup \{\mathsf{wr}[t_2]\}$$
❹

Memory access protocols

- Behavioral types for SIMT/SPMD that capture memory accesses
- One type per array. Capture: accesses, synchronization, structured loops
- Distinguish between synchronized/unsynchronized loops

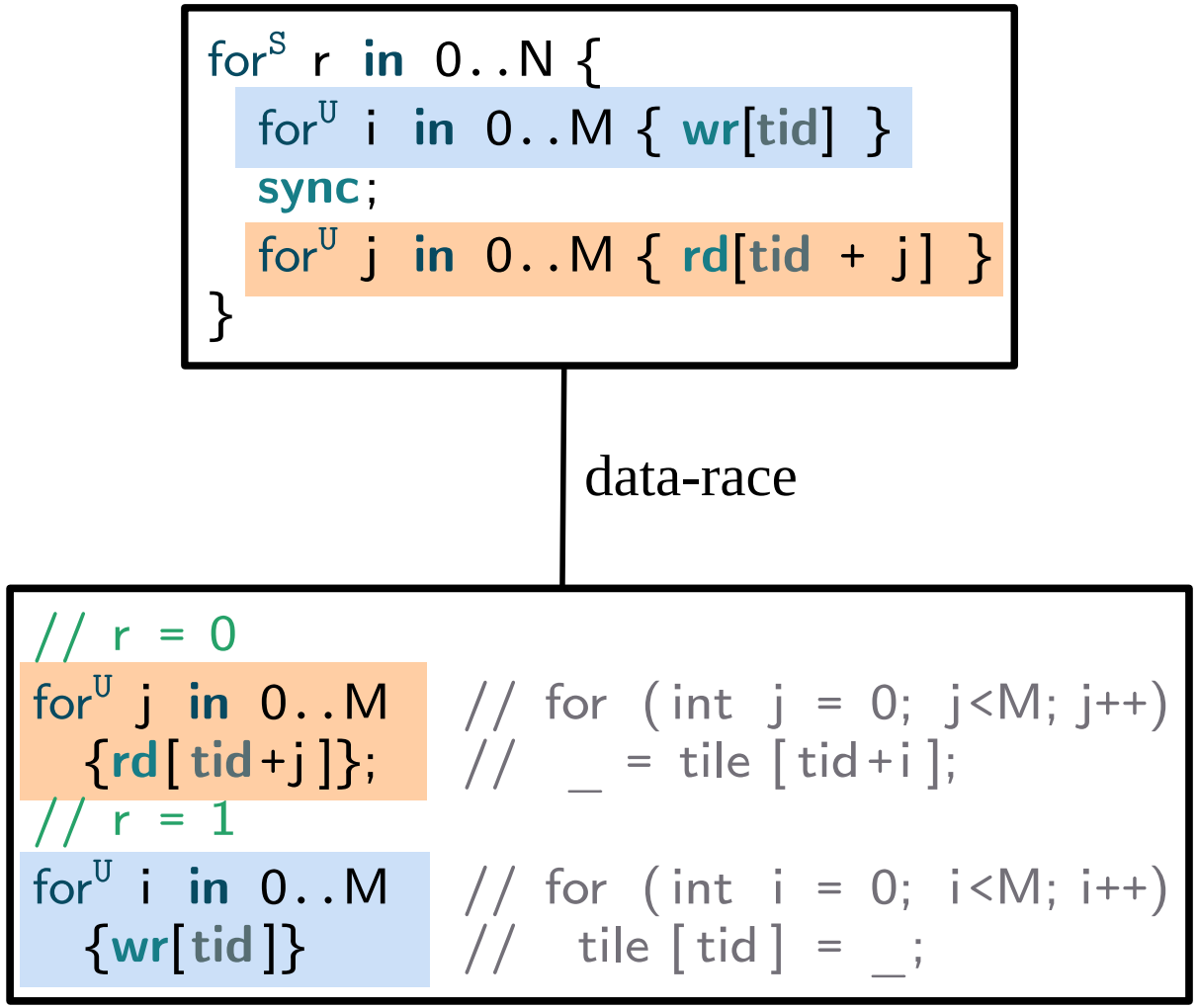
Ongoing work (inference): Provable GPU Data-Races in Static Race Detection [PLACES'22]




```
for (int r = 0; r < N; r++) {
  for (int i = 0; i<M; i++)
    { tile[tid] = ...;}
  __syncthreads();
  for (int j = 0; j<M; j++)
    {... = tile[tid+j];}}
```
1

```
for^S r in 0..N {
  for^U i in 0..M { wr[tid] }
  sync;
  for^U j in 0..M { rd[tid + j] }
}
```
2

A racy protocol



How do we solve the core problem?

Proving data-race-freedom in the unsynchronized fragment

A data-race in the unsynchronized fragment

Data-race: $t_1 = 0, t_2 = 1, j_1 = 1, M > 1$: rd[1] and wr[1].

```
// r = 0
forU j in 0..M      // for (int j = 0; j<M; j++)
  {rd[tid+j]};      //   _ = tile[tid+i];
// r = 1
forU i in 0..M      // for (int i = 0; i<M; i++)
  {wr[tid]}         //   tile[tid] = _;
```

Assumptions

- No synchronizations possible
- ***Data-race:*** given $t_1 \neq t_2$, index of t_1 equals index of t_2 and at least one is a write.

$$\forall j_1, i_1, j_2, i_2 : 0 \leq j_1 < M \land 0 \leq i_1 < M \land 0 \leq j_2 < M \land 0 \leq i_2 < M \implies$$
$$\{rd[t_1 + j_1]\} \cup \{wr[t_1]\} \quad DRF\ with?\quad \{rd[t_2 + j_2]\} \cup \{wr[t_2]\}$$

$\Leftrightarrow$ index(a_1)=index(a_4) ∨ index(a_2) = index(a_3) ∨ index(a_2) = index(a_4)

$\Leftrightarrow$ $t_1 + j_1 = t_2 \lor t_1 = t_2 + j_2 \lor t_1 = t_2$

Sequentializing protocols (step 3)

- Idea: represent the interleaving of any two threads (instead of all available threads) (FSE'10)
  - Each variable is duplicated per thread (thread-local view)
  - Interpret loops as ∀-binders (FSE'10)
- Key insight: Data-races are isolated on pairs of threads (not a transitive)

(FSE'10) Scalable SMT-Based Verification of GPU Kernel Functions.Guodong Li and Ganesh Gopalakrishnan.

Sequentializing protocols

Source syntax

$$\mathcal{U} \ni u \ ::= \ \mathsf{skip} \ | \ o[n] \ | \ u \,;\ u \ | \ \mathsf{if}\ b\ \{u\}\ \mathsf{else}\ \{u\} \ | \ \mathsf{for}^{\mathsf{U}}\ x \in n..m\ \{u\}$$

Target syntax

$$\mathcal{L} \ni h \ ::= \ \mathsf{skip} \ | \ n{:}o[m] \ | \ h \,;\ h \ | \ \mathsf{if}\ b\ \{h\}\ \mathsf{else}\ \{h\} \ | \ \mathsf{var}\ x\ \mathsf{in}\ n..m;\ h$$

Tracing

$$trace(\mathsf{skip}) = \mathsf{skip} \qquad trace(o[n]) = \mathsf{tid}{:}o[n]$$

$$trace(u_1 \,;\ u_2) = trace(u_1) \,;\ trace(u_2)$$

$$trace(\mathsf{if}\ b\ \{u_1\}\ \mathsf{else}\ \{u_2\}) = \mathsf{if}\ b\ \{trace(u_1)\}\ \mathsf{else}\ \{trace(u_2)\}$$

$$trace(\mathsf{for}^{\mathsf{U}}\ x \in n..m\ \{u\}) = \mathsf{var}\ x\ \mathsf{in}\ n..m;\ trace(u)$$

Sequentializing protocols

Sequencing

$$\frac{t_1, t_2 \text{ fresh} \qquad h_1 = trace(u)[\mathsf{tid} := t_1] \qquad h_2 = trace(u)[\mathsf{tid} := t_2]}{seq(u) = \mathsf{var}\ t_1\ \mathsf{in}\ 1..|\mathbb{T}|;\ \mathsf{var}\ t_2\ \mathsf{in}\ 0..t_1;\ h_1\ ;\ h_2}$$

$$seq(\mathsf{var}\ x\ \mathsf{in}\ n..m;\ c) = \mathsf{var}\ x\ \mathsf{in}\ n..m;\ seq(c)$$

$$seq([c_1, \ldots, c_n]) = [seq(c_1), \ldots, seq(c_n)]$$

Results

Lemma 17. *If* $\phi \subseteq^{\mathtt{H}} seq(c)$, *then* $\phi \subseteq^{\mathtt{C}} c$.

Lemma 18. *Let* $\phi = (\alpha_1, \alpha_2)$. *If* $owner(\alpha_1) \neq owner(\alpha_2)$ *and* $\phi \subseteq^{\mathtt{C}} c$, *then* $\phi \subseteq^{\mathtt{H}} seq(c)$.

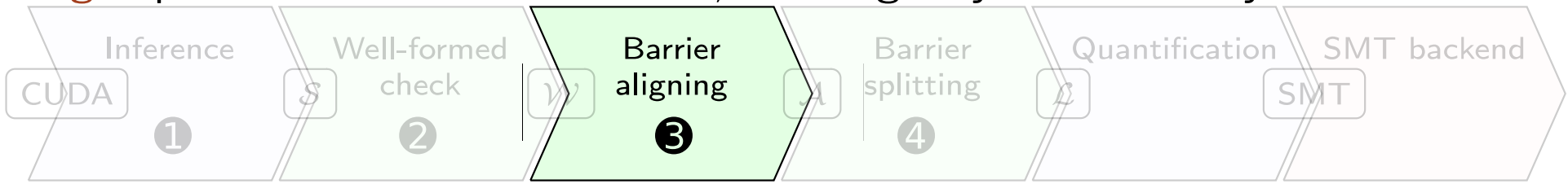
How do we get to unsychronized protocols?

Aligning protocols

(step 1)

Aligning protocols (step 1)

- We define a notion of **aligned** protocols, where accesses do not "leak" across iterations
- We show that all protocols can be aligned (modulo notion of well-formedness)
- **Intuition:** unfold loop and rearrange accesses
- Insight: protocols have no local state; ordering only matter wrt synch




```
forS r in 0..N {
  forU i in 0..M { wr[tid] }
  sync;
  forU j in 0..M { rd[tid + j] }
}
```
❷

```
forU i in 0..M { wr[tid] }
sync;
forS r in 1..N {
  forU j in 0..M { rd[tid + j] }
  forU i in 0..M { wr[tid] }
  sync; }
forU j in 0..M { rd[tid + j] }
```
❸

Aligning protocols

Source: well-formed protocols

$$\mathcal{W} \ni p \ ::= \ u\,;\ \mathsf{sync} \ \mid \ p\,;\ p \ \mid \ u\,;\ \mathsf{for}^{\mathsf{S}}\ x \in n..m\ \{p\,;\ u\}$$

Target: aligned protocols

$$\mathcal{A} \ni a \ ::= \ u\,;\ \mathsf{sync} \ \mid \ a\,;\ a \ \mid \ a\,;\ \mathsf{for}^{\mathsf{S}}\ x \in n..m\ \{a\}$$

Aligning protocols

$$\frac{\phi \subseteq^{\mathtt{U}} u}{\phi \subseteq^{\mathtt{W}} u\,;\ \mathsf{sync}}\ \mathcal{W}\text{-ACC}$$

$$\frac{\phi \subseteq^{\mathtt{W}} p}{\phi \subseteq^{\mathtt{W}} p\,;\ q}\ \mathcal{W}\text{-SEQ-L}$$

$$\frac{\phi \subseteq^{\mathtt{W}} q}{\phi \subseteq^{\mathtt{W}} p\,;\ q}\ \mathcal{W}\text{-SEQ-R}$$

$$\frac{\alpha_1 \in^{\mathtt{W}} lst(p) \qquad \alpha_2 \in^{\mathtt{W}} fst(q)}{(\alpha_1, \alpha_2) \subseteq^{\mathtt{W}} p\,;\ q}\ \mathcal{W}\text{-SEQ-B}$$

$$\frac{\phi \subseteq^{\mathtt{U}} u_1}{\phi \subseteq^{\mathtt{W}} u_1\,;\ \mathsf{for}^{\mathtt{S}}\ x \in n..m\ \{p\,;\ u_2\}}\ \subseteq^{\mathtt{W}}\text{-FOR-1}$$

$$\frac{\alpha_1 \in^{\mathtt{U}} u_1 \qquad \alpha_2 \in^{\mathtt{W}} fst(p[x := n])}{(\alpha_1, \alpha_2) \subseteq^{\mathtt{W}} u_1\,;\ \mathsf{for}^{\mathtt{S}}\ x \in n..m\ \{p\,;\ u_2\}}\ \subseteq^{\mathtt{W}}\text{-FOR-2}$$

Aligning protocols

$$\frac{n \downarrow j \qquad m \downarrow k \qquad \exists i \colon j \leq i < k \qquad \phi \subseteq^{\mathtt{W}} p[x := i]}{\phi \subseteq^{\mathtt{W}} u_1 \,;\, \mathsf{for}^{\mathtt{S}}\ x \in n..m\ \{p \,;\, u_2\}}\ \subseteq^{\mathtt{W}}\text{-FOR-3}$$

$$\frac{n \downarrow j \qquad m \downarrow k \qquad \exists i \colon j \leq i < k \qquad \phi \subseteq^{\mathtt{W}} u_2[x := i]}{\phi \subseteq^{\mathtt{W}} u_1 \,;\, \mathsf{for}^{\mathtt{S}}\ x \in n..m\ \{p \,;\, u_2\}}\ \subseteq^{\mathtt{W}}\text{-FOR-4}$$

$$\frac{n \downarrow j \qquad m \downarrow k \qquad \exists i \colon j \leq i < k \qquad \alpha_1 \in^{\mathtt{W}} lst(p[x := i]) \qquad \alpha_2 \in^{\mathtt{U}} u_2[x := i]}{(\alpha_1, \alpha_2) \subseteq^{\mathtt{W}} u_1 \,;\, \mathsf{for}^{\mathtt{S}}\ x \in n..m\ \{p \,;\, u_2\}}\ \subseteq^{\mathtt{W}}\text{-FOR-5}$$

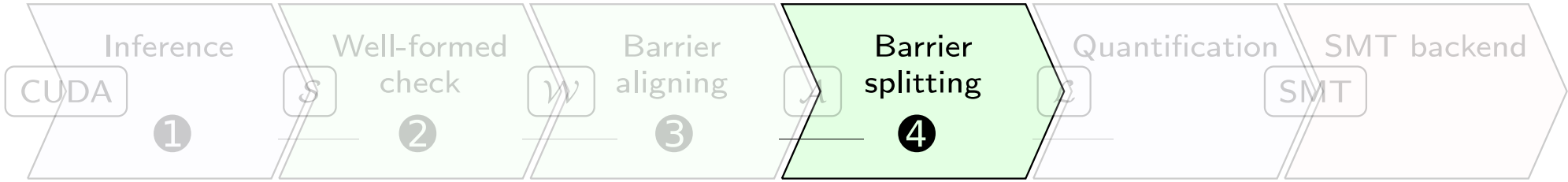
$$\frac{n \downarrow j \qquad m \downarrow k \qquad \exists i \colon j \leq i < k \qquad \alpha_1 \in^{\mathtt{U}} u_2[x := i] \qquad \alpha_2 \in^{\mathtt{W}} fst(q[x := i+1])}{(\alpha_1, \alpha_2) \subseteq^{\mathtt{W}} u_1 \,;\, \mathsf{for}^{\mathtt{S}}\ x \in n..m\ \{p \,;\, u_2\}}\ \subseteq^{\mathtt{W}}\text{-FOR-6}$$

$$\frac{n \downarrow j \qquad m \downarrow k \qquad \exists i \colon j \leq i < k \qquad \alpha_1 \in^{\mathtt{W}} lst(q[x := i]) \qquad \alpha_2 \in^{\mathtt{W}} fst(q[x := i+1])}{(\alpha_1, \alpha_2) \subseteq^{\mathtt{W}} u_1 \,;\, \mathsf{for}^{\mathtt{S}}\ x \in n..m\ \{p \,;\, u_2\}}\ \subseteq^{\mathtt{W}}\text{-FOR-7}$$

Splitting protocols

(step 3)

Splitting protocols




```
forU i in 0..M { wr[tid] }
sync;
forS r in 1..N {
  forU j in 0..M { rd[tid + j] }
  forU i in 0..M { wr[tid] }
  sync; }
forU j in 0..M { rd[tid + j] }
```
❸

$$\forall r, j_1, i_1, j_2, i_2 : 1 \leq r < N \land 0 \leq j_1 < M \land 0 \leq i_1 < M \land 0 \leq j_2 < M \land 0 \leq i_2 < M$$
$$\implies \{\mathsf{rd}[t_1 + j_1]\} \cup \{\mathsf{wr}[t_1]\} \quad \textit{DRF with?} \quad \{\mathsf{rd}[t_2 + j_2]\} \cup \{\mathsf{wr}[t_2]\}$$ ❹

Splitting protocols

- Syntax-oriented extraction of unsynchronized fragments
- **Compositional** analysis (no data-races between fragments)
- Synchronized loop variables can also be interpret as a forall-binder
- However, the binder must be shared by both threads (ie, only one r variable shared by both threads)

Conclusion

- Behavioral types being used to enforce data-race freedom
- A compositional analysis, formally proved
- Large experimental evaluation (229 real-world + 258 synthetic = 487 kernels)
- Used our tool to confirm data-races found in the wild
- Our approach is more scalable and more precise (fewer false-positives) than related work
- Source code and proofs available in a free software license

Future directions

- Proving alarms are true (rather than proving false alarms)
- Resource analysis of memory access protocols
- Incompleteness logic to showcase performance bottlenecks

https://gitlab.com/umb-svl/faial
https://gitlab.com/umb-svl/faial-coq

UMASS BOSTON