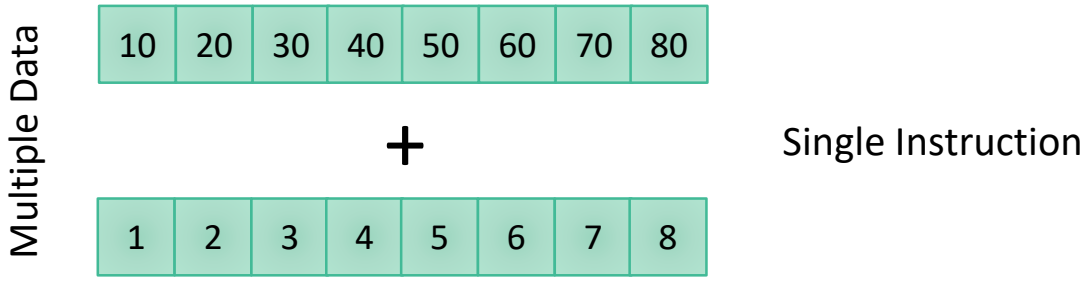
GPURepair: Automated Repair of GPU Kernels

SAT+SMT 2020

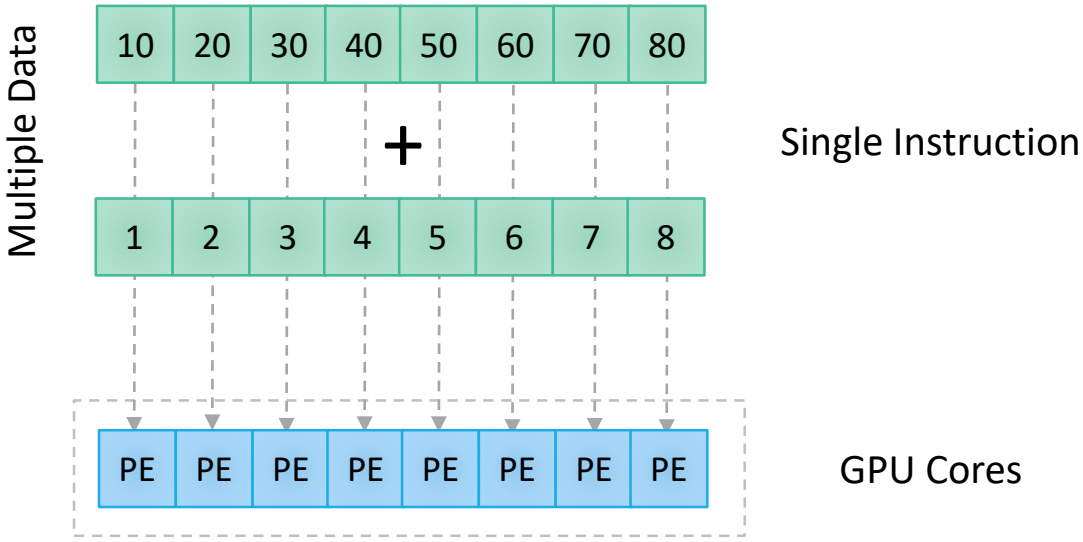
Presented By

Saurabh Joshi (sbjoshi@iith.ac.in)
Gautam Muduganti (cs17resch01003@iith.ac.in)

Introduction to GPUs

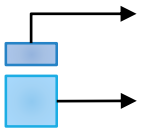


Introduction to GPUs



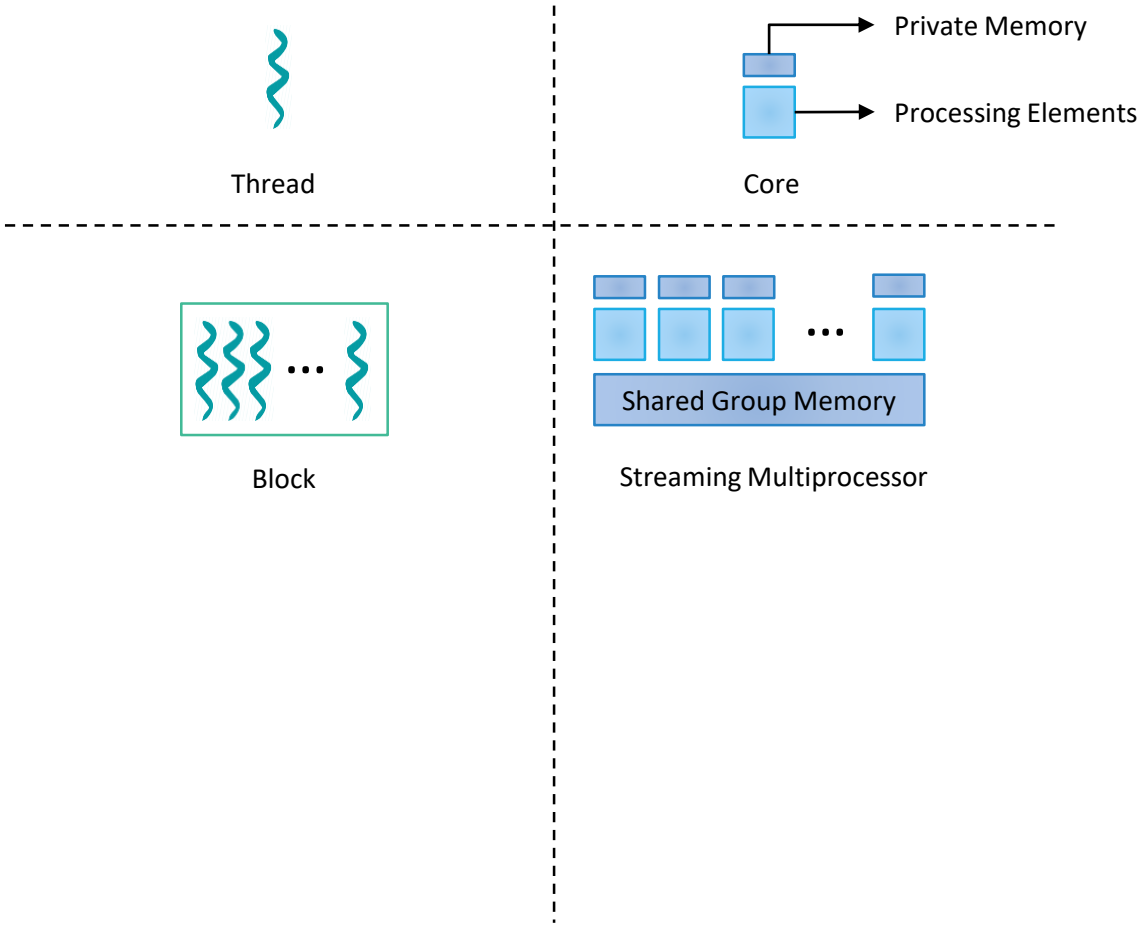
Introduction to GPUs

Thread

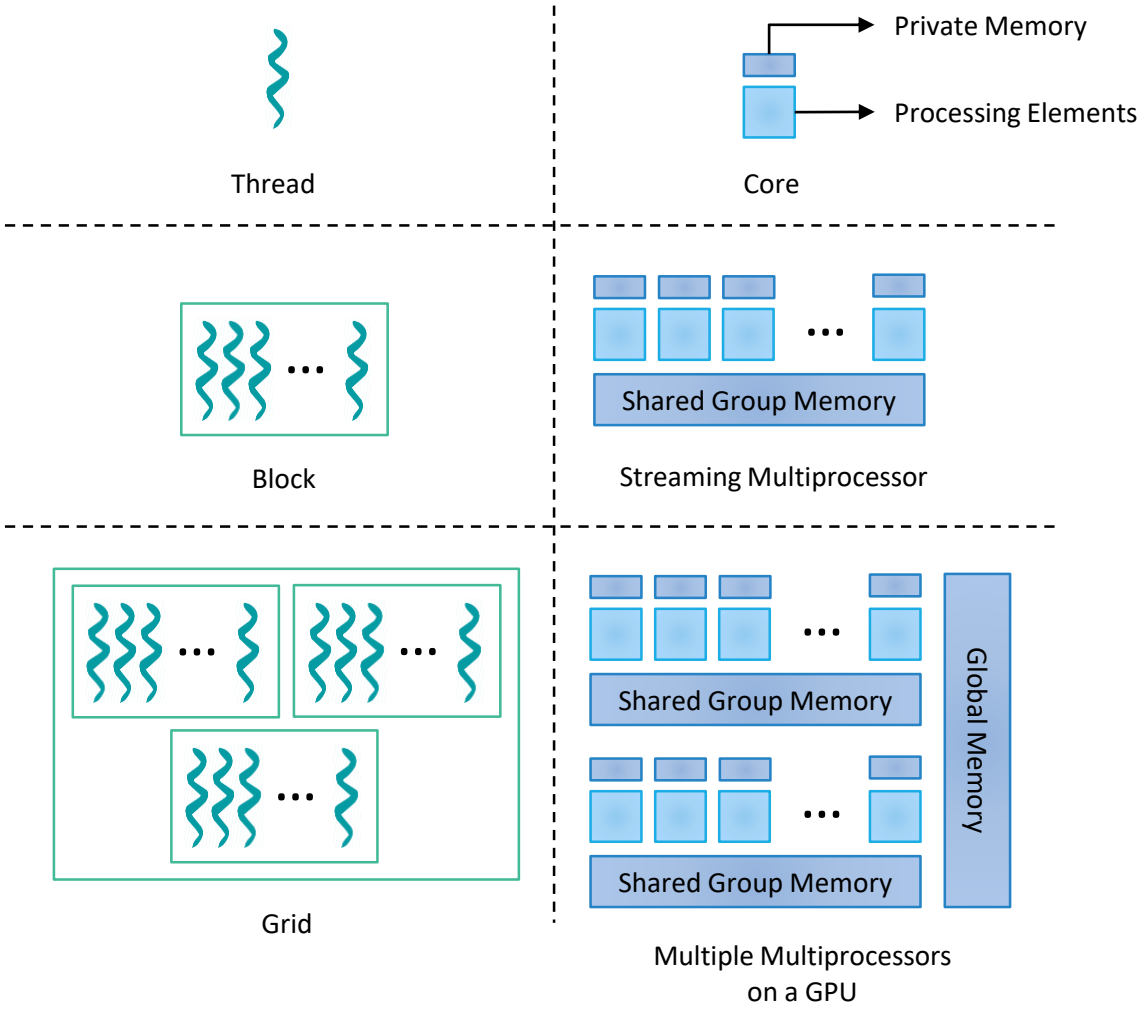


Core

Introduction to GPUs



Introduction to GPUs

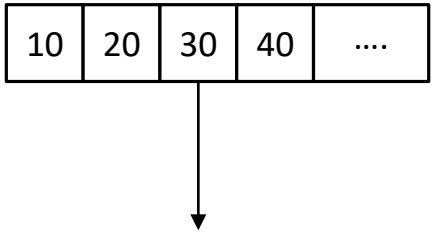


Challenges of GPU Programming: Data Races

Challenges of GPU Programming: Data Races

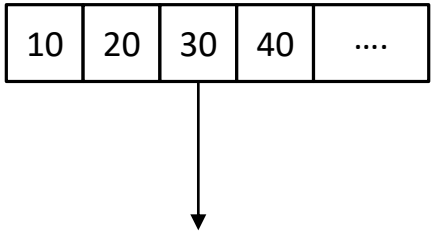
```
__global__ void race(int* A) {
    int temp = A[threadIdx.x + 1];
    A[threadIdx.x] = temp;
}
```

Challenges of GPU Programming: Data Races



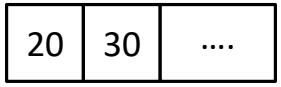

```
__global__ void race(int* A) {
    int temp = A[threadIdx.x + 1];
    A[threadIdx.x] = temp;
}
```

Challenges of GPU Programming: Data Races

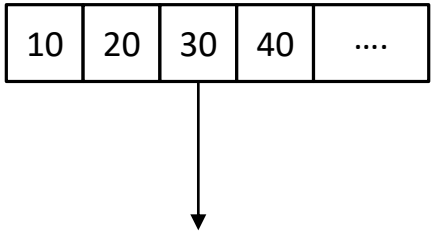



```
__global__ void race(int* A) {
    int temp = A[threadIdx.x + 1];
    A[threadIdx.x] = temp;
}
```

Thread-0	Thread-1
temp = 20;	temp = 30;
A[0] = 20;	A[1] = 30;

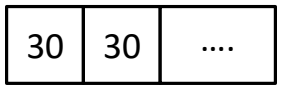


Challenges of GPU Programming: Data Races

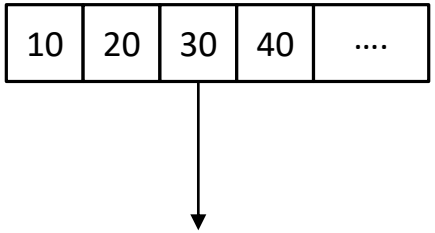



```
__global__ void race(int* A) {
    int temp = A[threadIdx.x + 1];
    A[threadIdx.x] = temp;
}
```

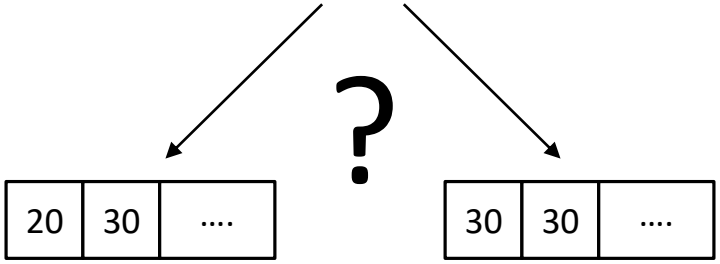
Thread-0	Thread-1
	temp = 30; A[1] = 30;
temp = 30; A[0] = 30;	



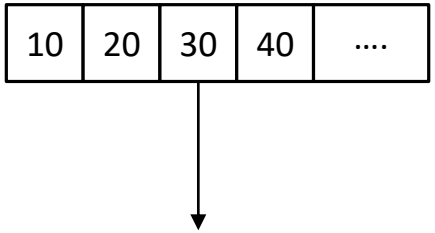
Challenges of GPU Programming: Data Races



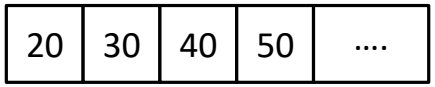

```
__global__ void race(int* A) {
    int temp = A[threadIdx.x + 1];
    A[threadIdx.x] = temp;
}
```



Challenges of GPU Programming: Data Races




```
__global__ void race(int* A) {
    int temp = A[threadIdx.x + 1];
    __syncthreads();
    A[threadIdx.x] = temp;
}
```

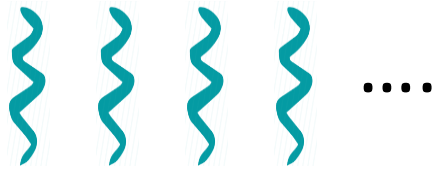


Challenges of GPU Programming: Barrier Divergence

```
__global__ void race(int* A) {
    if (threadIdx.x % 2 == 0) {
        int temp = A[threadIdx.x] + 10;
        __syncthreads();
        A[threadIdx.x] = temp;
    }
}
```

Challenges of GPU Programming: Barrier Divergence

```
__global__ void race(int* A) {
    if (threadIdx.x % 2 == 0) {
        int temp = A[threadIdx.x] + 10;
        __syncthreads();
        A[threadIdx.x] = temp;
    }
}
```



Challenges of GPU Programming: Barrier Divergence

```
__global__ void race(int* A) {
    if (threadIdx.x % 2 == 0) {
        int temp = A[threadIdx.x] + 10;
        __syncthreads();
        A[threadIdx.x] = temp;
    }
}
```

Challenges of GPU Programming: Barrier Divergence

```
__global__ void race(int* A) {
    if (threadIdx.x % 2 == 0) {
        int temp = A[threadIdx.x] + 10;
        __syncthreads();
        A[threadIdx.x] = temp;
    }
}
```

Challenges of GPU Programming: Barrier Divergence

```
__global__ void race(int* A) {
    if (threadIdx.x % 2 == 0) {
        int temp = A[threadIdx.x] + 10;
        A[threadIdx.x] = temp;
    }
}
```

GPURepair: Problem Statement

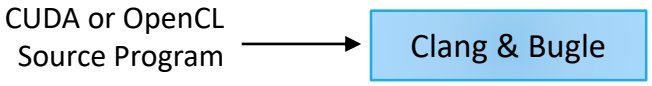
Given a GPU kernel P containing data races and barrier divergence errors, GPURepair returns a repaired kernel P' with the least number of barriers enabled in P'

GPURepair: Architecture

GPURepair: Architecture

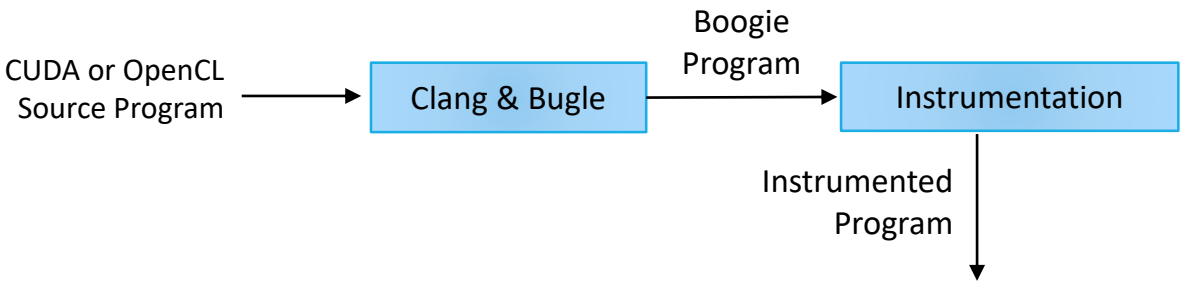
CUDA or OpenCL
Source Program

GPURepair: Architecture

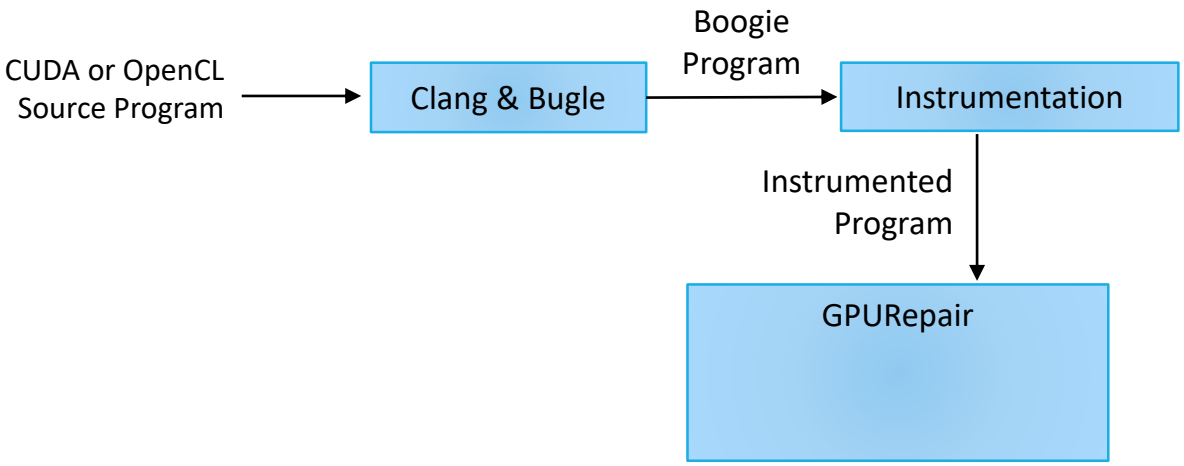


GPURepair: Architecture

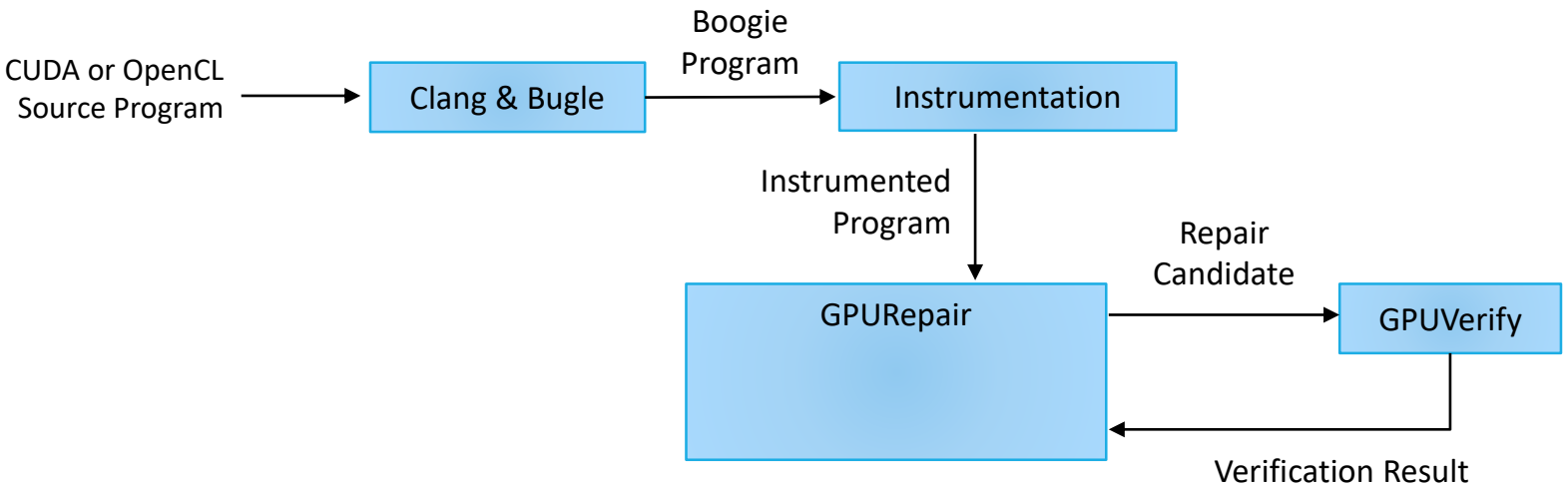
GPURepair: Architecture



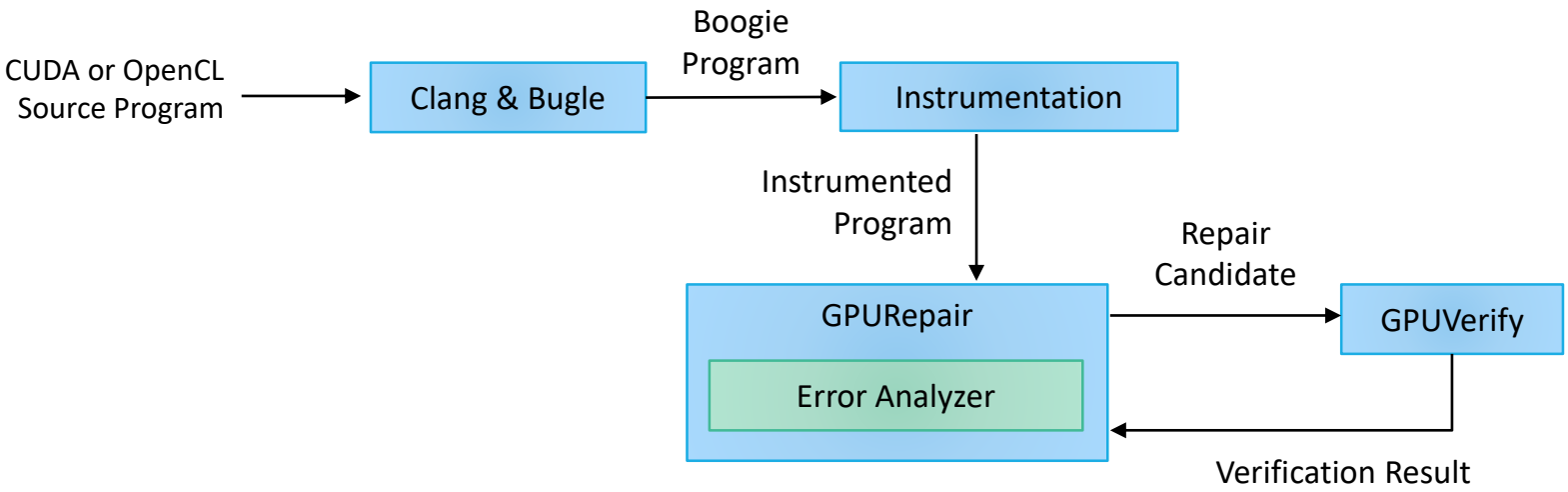
GPURepair: Architecture



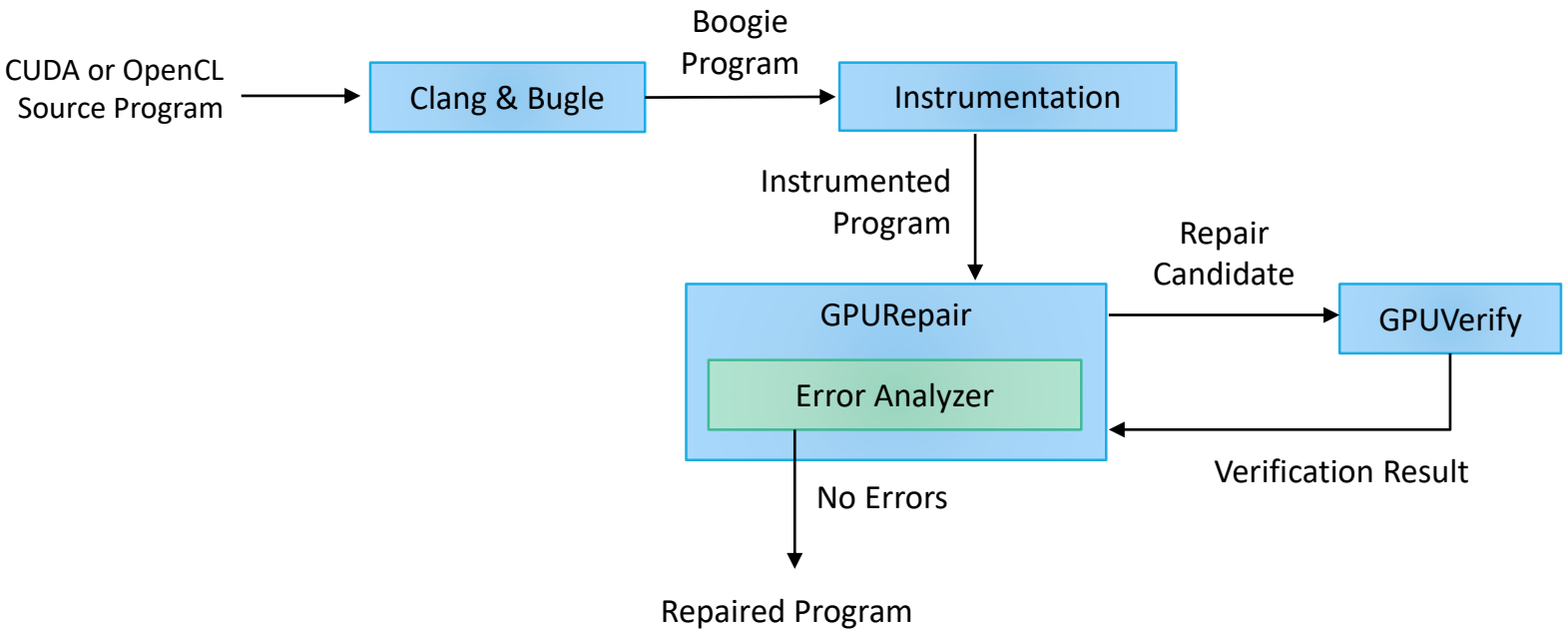
GPURepair: Architecture



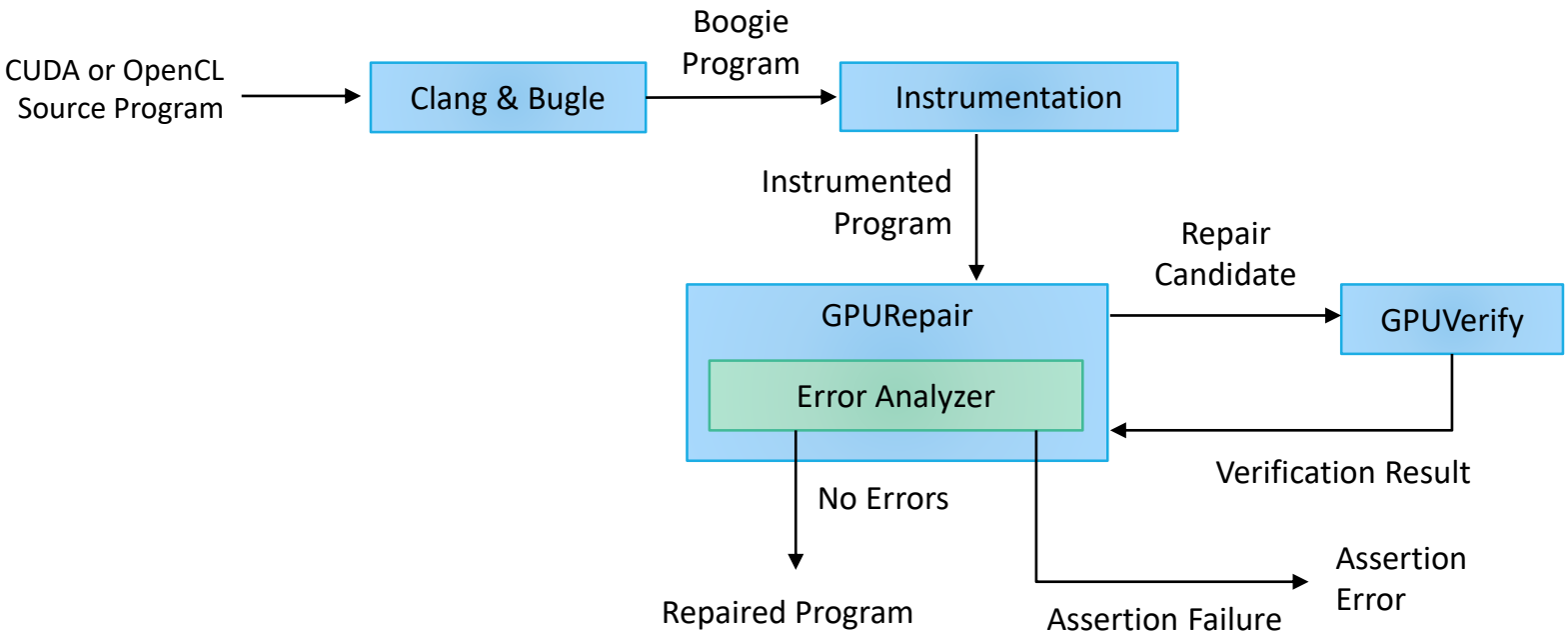
GPURepair: Architecture



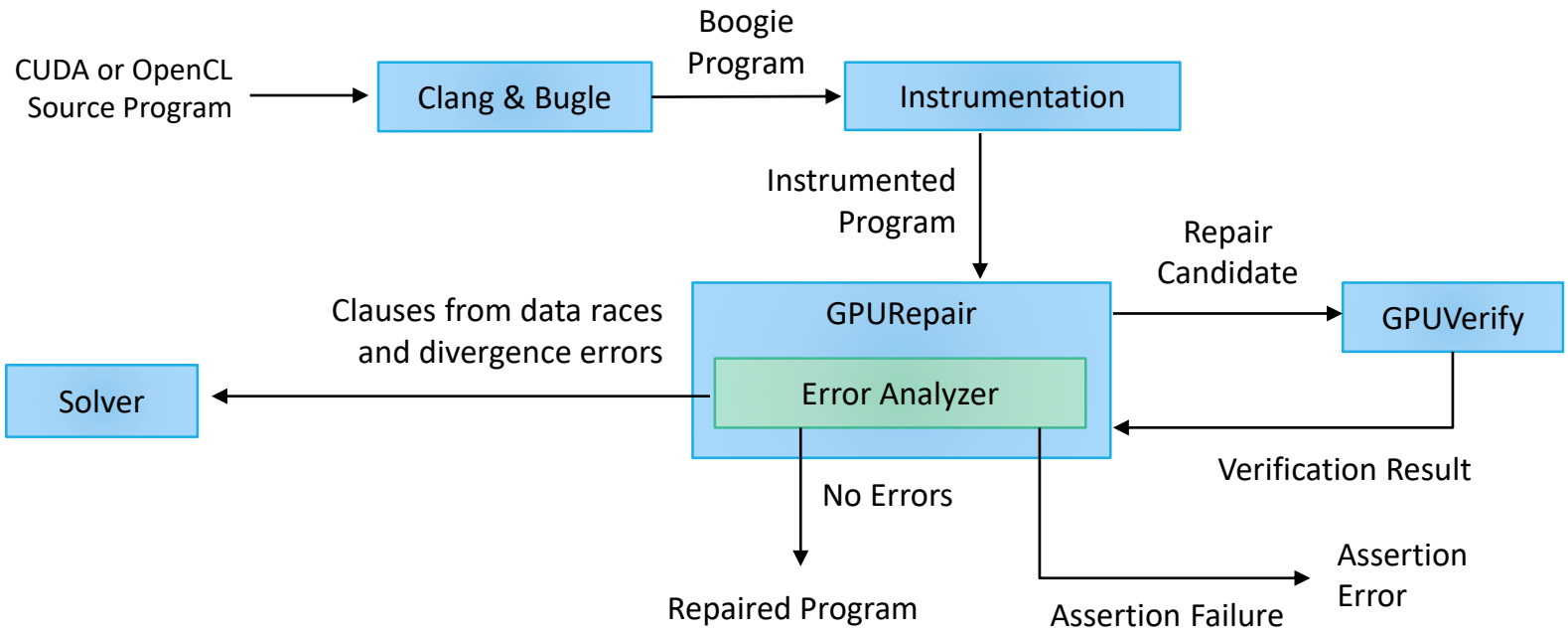
GPURepair: Architecture



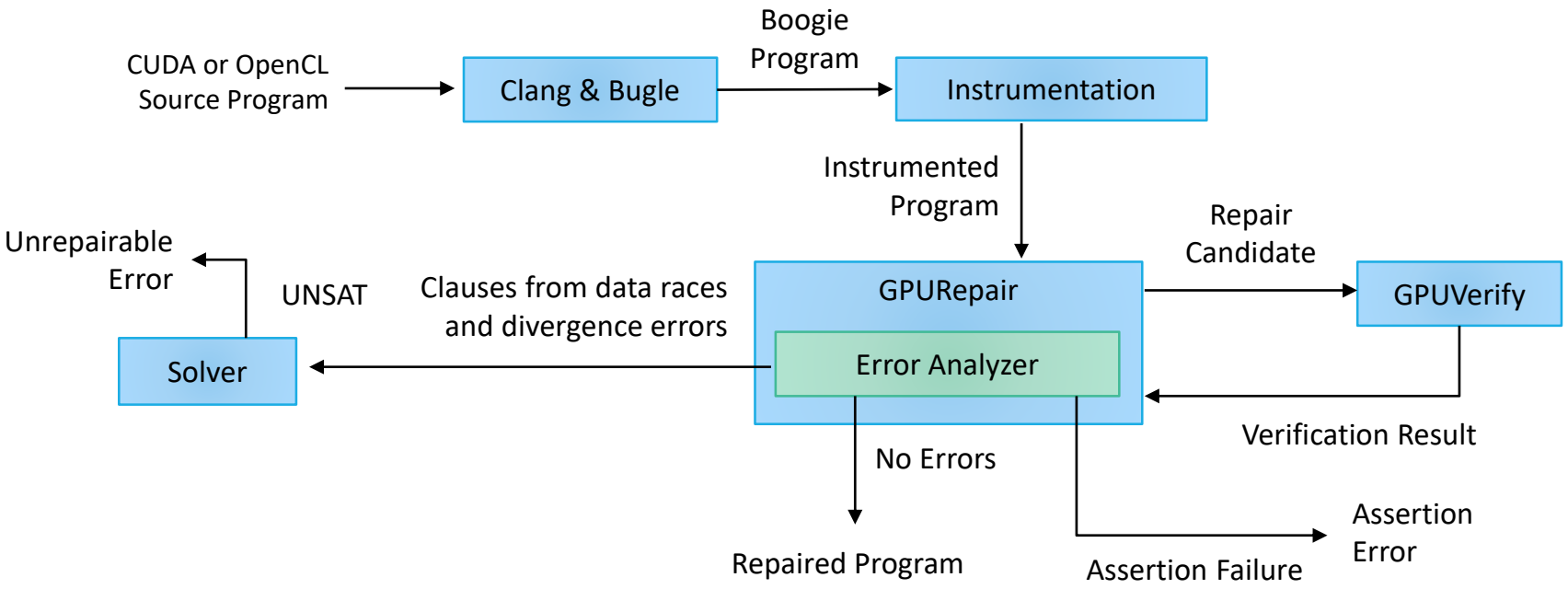
GPURepair: Architecture



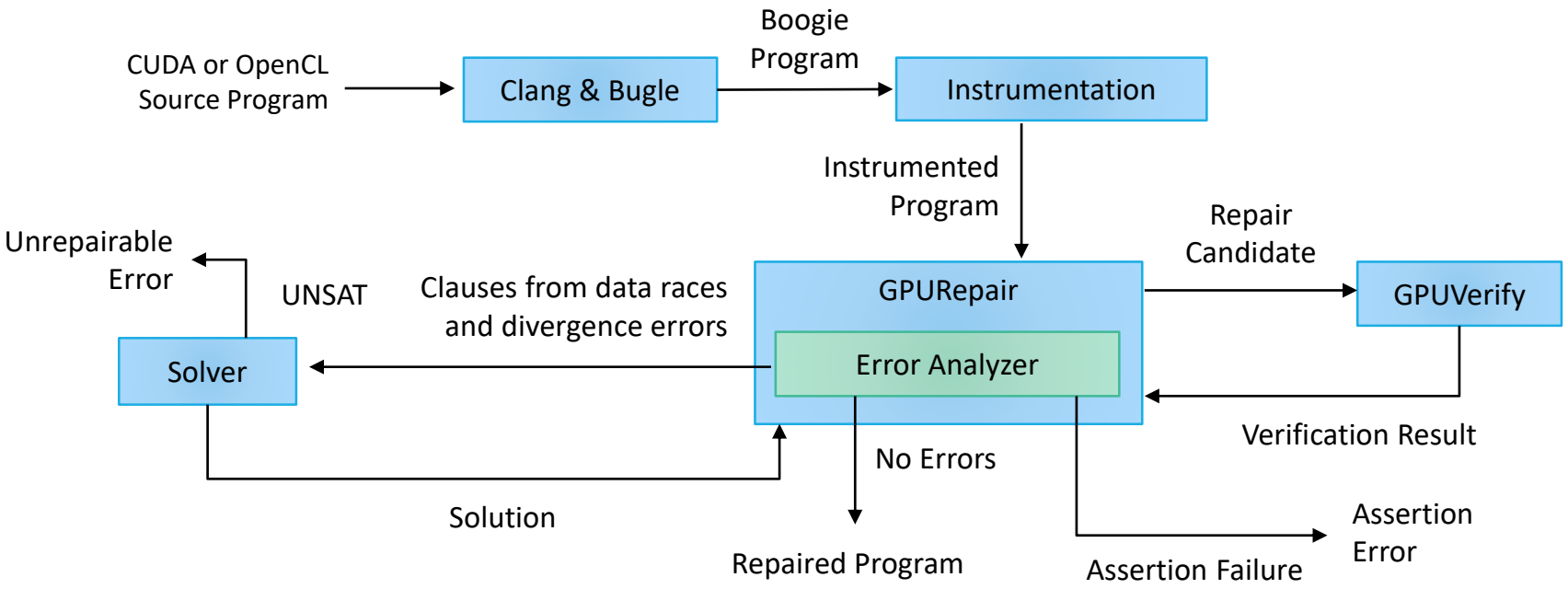
GPURepair: Architecture



GPURepair: Architecture

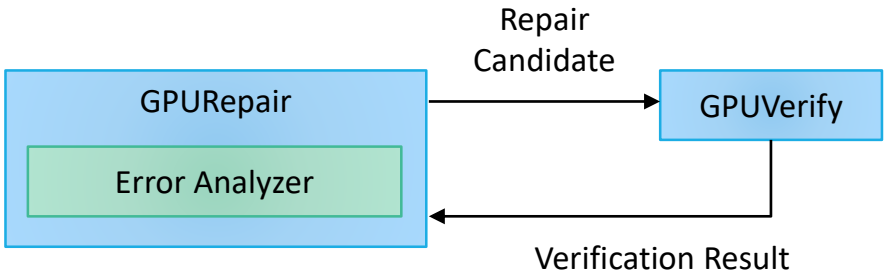


GPURepair: Architecture



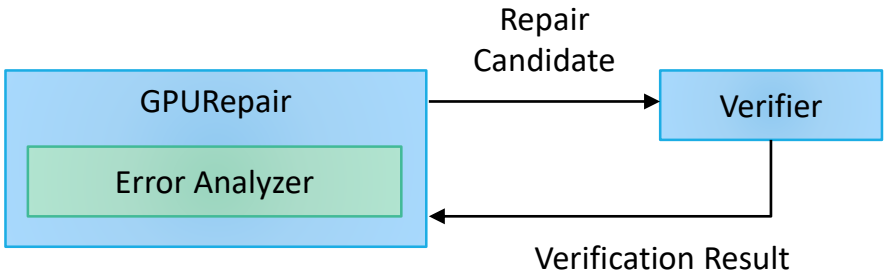
GPURepair: Architecture

Counter-example guided repair synthesis approach



GPURepair: Architecture

In principle, any verifier can be used as an oracle



Instrumentation: Shared Variables

Instrumentation: Shared Variables

```
__global__ void race(int* A) {

    int temp = A[threadIdx.x + 1];


    A[threadIdx.x] = temp;
}
```

Instrumentation: Shared Variables

```
__global__ void race(int* A) {

    int temp = A[threadIdx.x + 1];


    A[threadIdx.x] = temp;
}
```

Instrumentation: Shared Variables

```
bool b1, b2;
__global__ void race(int* A) {
     if (b1) { __syncthreads(); }
     int temp = A[threadIdx.x + 1];

     if (b2) { __syncthreads(); }
     A[threadIdx.x] = temp;
}
```

Instrumentation: Scope Boundaries

Instrumentation: Scope Boundaries

```
__global__ void race(int* A) {

    int temp = A[threadIdx.x + 1];


    if (threadIdx.x % 2 == 0) {

        A[threadIdx.x] = temp;
    }
}
```

Instrumentation: Scope Boundaries

```
bool b1, b2;
__global__ void race(int* A) {
     if (b1) { __syncthreads(); }
     int temp = A[threadIdx.x + 1];


     if (threadIdx.x % 2 == 0) {
          if (b2) { __syncthreads(); }
          A[threadIdx.x] = temp;
     }
}
```

Instrumentation: Scope Boundaries

```
bool b1, b2;
__global__ void race(int* A) {
     if (b1) { __syncthreads(); }
     int temp = A[threadIdx.x + 1];


     if (threadIdx.x % 2 == 0) {
          if (b2) { __syncthreads(); }
          A[threadIdx.x] = temp;
     }
}
```

Instrumentation: Scope Boundaries

```
bool b1, b2, b3;
__global__ void race(int* A) {
     if (b1) { __syncthreads(); }
     int temp = A[threadIdx.x + 1];

     if (b3) { __syncthreads(); }
     if (threadIdx.x % 2 == 0) {
          if (b2) { __syncthreads(); }
          A[threadIdx.x] = temp;
     }
}
```

Instrumentation: Inter-block Synchronization

```
bool b1, b2;
__global__ void race(int* A) {
    if (b1) { __syncthreads(); }
    int temp = A[threadIdx.x + 1];

    if (b2) { __syncthreads(); }
    A[threadIdx.x] = temp;
}
```

Block-level Synchronization

Instrumentation: Inter-block Synchronization

```
bool b1, b2, b3, b4;
__global__ void race(int* A) {
    auto g = this_grid();

    if (b1) { __syncthreads(); }
    if (b3) { g.sync(); }
    int temp = A[threadIdx.x + 1];

    if (b2) { __syncthreads(); }
    if (b4) { g.sync(); }
    A[threadIdx.x] = temp;
}
```

Block-level Synchronization

Grid-level Synchronization

Repair

```
bool b1, b2, b3;
__global__ void race(int* A) {
     if (b1) { __syncthreads(); }
     int temp = A[threadIdx.x + 1];

     if (b3) { __syncthreads(); }
     if (threadIdx.x % 2 == 0) {
          if (b2) { __syncthreads(); }
          A[threadIdx.x] = temp;
     }
}
```

Repair

```
bool b1, b2, b3;
__global__ void race(int* A) {
    if (b1) { __syncthreads(); }
    int temp = A[threadIdx.x + 1];

    if (b3) { __syncthreads(); }
    if (threadIdx.x % 2 == 0) {
        if (b2) { __syncthreads(); }
        A[threadIdx.x] = temp;
    }
}
```

Iteration 1

Error Type: Data Race

GPUVerify Assignments:

$$b1 = false$$
$$b2 = false$$
$$b3 = false$$

Clauses Generated:

$$b1 \vee b2 \vee b3$$

Solver Solution:

$$b2 \Leftrightarrow true$$

Repair

```
bool b1, b2, b3;
__global__ void race(int* A) {
     if (b1) { __syncthreads(); }
     int temp = A[threadIdx.x + 1];

     if (b3) { __syncthreads(); }
     if (threadIdx.x % 2 == 0) {
          __syncthreads();
          A[threadIdx.x] = temp;
     }
}
```

Iteration 1

Error Type: Data Race

GPUVerify Assignments:

$$b1 = false$$
$$b2 = false$$
$$b3 = false$$

Clauses Generated:

$$b1 \vee b2 \vee b3$$

Solver Solution:

$$b2 \Leftrightarrow true$$

Repair

```
bool b1, b2, b3;
__global__ void race(int* A) {
     if (b1) { __syncthreads(); }
     int temp = A[threadIdx.x + 1];

     if (b3) { __syncthreads(); }
     if (threadIdx.x % 2 == 0) {
          __syncthreads();
          A[threadIdx.x] = temp;
     }
}
```

Iteration 2

Error Type: Divergence

GPUVerify Assignments:

$$b2 = true$$

Clauses Generated:

$$b1 \vee b2 \vee b3$$
$$\neg b2$$

Solver Solution:

$$b2 \Leftrightarrow false$$
$$b3 \Leftrightarrow true$$

Repair

```
bool b1, b2, b3;
__global__ void race(int* A) {
    if (b1) { __syncthreads(); }
    int temp = A[threadIdx.x + 1];

    __syncthreads();
    if (threadIdx.x % 2 == 0) {

        A[threadIdx.x] = temp;
    }
}
```

Iteration 2

Error Type: Divergence

GPUVerify Assignments:

$$b2 = true$$

Clauses Generated:

$$b1 \vee b2 \vee b3$$
$$\neg b2$$

Solver Solution:

$$b2 \Leftrightarrow false$$
$$b3 \Leftrightarrow true$$

Repair

```
bool b1, b2, b3;
__global__ void race(int* A) {
     if (b1) { __syncthreads(); }
     int temp = A[threadIdx.x + 1];

     __syncthreads();
     if (threadIdx.x % 2 == 0) {

          A[threadIdx.x] = temp;
     }
}
```

Iteration 3

Error Type: No Errors

Repair

```
bool b1, b2, b3;
__global__ void race(int* A) {

    int temp = A[threadIdx.x + 1];

    __syncthreads();
    if (threadIdx.x % 2 == 0) {

        A[threadIdx.x] = temp;
    }
}
```

Iteration 3

Error Type: No Errors

Solver Strategies

1. MaxSAT Strategy

Solver Strategies

1. MaxSAT Strategy

Hard Clauses

$b1 \vee b2 \vee b3$

$\neg b2$

Soft Clauses

$\neg b1$

$\neg b2$

$\neg b3$

Solver Strategies

2. Minimal Hitting Set (mhs) Strategy
   - Polynomial-time greedy algorithm + Unit Literal Propagation

Solver Strategies

2. Minimal Hitting Set (mhs) Strategy
   - Polynomial-time greedy algorithm + Unit Literal Propagation

Example Clauses

$$b1 \vee b2 \vee b4$$
$$b2 \vee b4$$
$$b2 \vee b3 \vee b5$$
$$b3 \vee b5 \vee b6$$
$$b1 \vee b6$$
$$\neg b2 \vee \neg b5$$

Solver Strategies

2. Minimal Hitting Set (mhs) Strategy
   - Polynomial-time greedy algorithm + Unit Literal Propagation

Example Clauses

$$b1 \vee b2 \vee b4$$
$$b2 \vee b4$$
$$b2 \vee b3 \vee b5$$
$$b3 \vee b5 \vee b6$$
$$b1 \vee b6$$
$$\neg b2 \vee \neg b5$$

Solution

Solver Strategies

2. Minimal Hitting Set (mhs) Strategy
   - Polynomial-time greedy algorithm + Unit Literal Propagation

Example Clauses	Solution
$b1 \vee b2 \vee b4$	$b2 \Leftrightarrow true$
$b2 \vee b4$	
$b2 \vee b3 \vee b5$	
$b3 \vee b5 \vee b6$	
$b1 \vee b6$	
$\neg b2 \vee \neg b5$	

Solver Strategies

2. Minimal Hitting Set (mhs) Strategy
   - Polynomial-time greedy algorithm + Unit Literal Propagation

Example Clauses	**Solution**
$b1 \vee b2 \vee b4$	$b2 \Leftrightarrow true$
$b2 \vee b4$	$b5 \Leftrightarrow false$
$b2 \vee b3 \vee b5$	
$b3 \vee b5 \vee b6$	
$b1 \vee b6$	
$\neg b2 \vee \neg b5$	

Solver Strategies

2. Minimal Hitting Set (mhs) Strategy
   - Polynomial-time greedy algorithm + Unit Literal Propagation

Example Clauses

$b1 \vee b2 \vee b4$
$b2 \vee b4$
$b2 \vee b3 \vee b5$
$b3 \vee b5 \vee b6$
$b1 \vee b6$
$\neg b2 \vee \neg b5$

Solution

$b2 \Leftrightarrow true$
$b5 \Leftrightarrow false$
$b6 \Leftrightarrow true$

Benchmark Summary

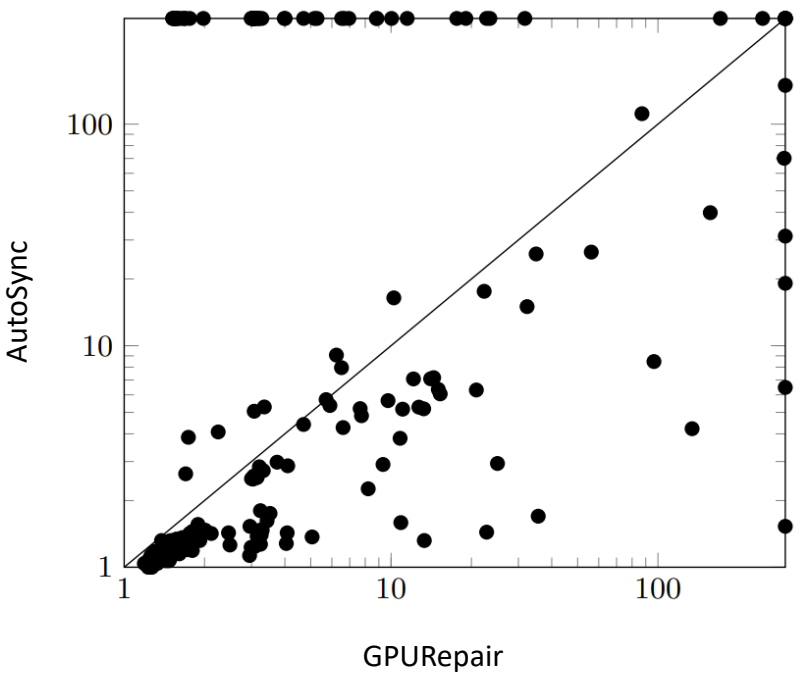
AutoSync	GPURepair
GPUVerify Test Suite (Inc. **482** OpenCL Kernels) [1]	**658**
NVIDIA GPU Computing SDK v5.0 [2]	**56**
AutoSync Micro Benchmarks	**8**
GPURepair Test Suite (Inc. **16** examples for CUDA Cooperative Groups)	**26**
Total	**748**

[1] https://github.com/mc-imperial/gpuverify/tree/master/testsuite [2] https://developer.nvidia.com/cuda-toolkit-50-archive

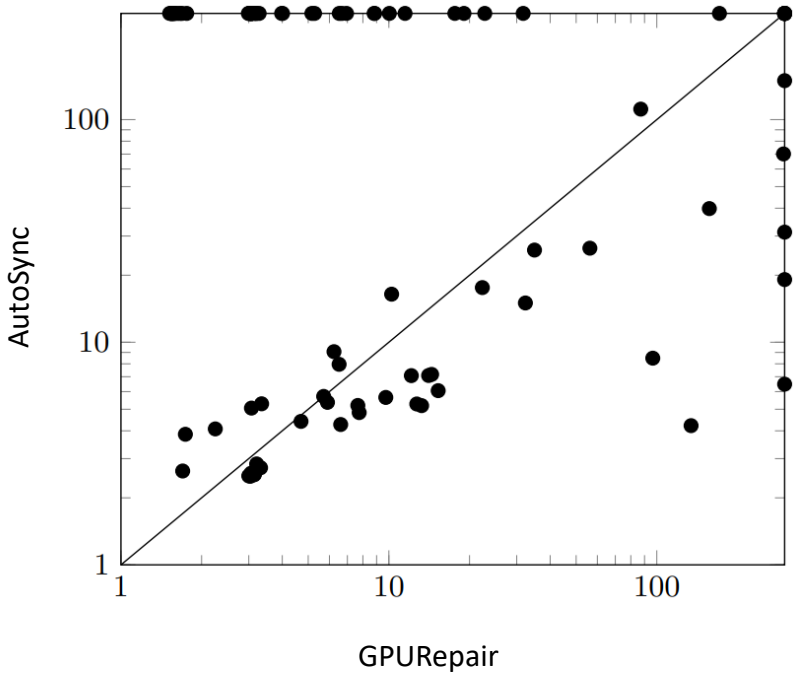
Results

Category	AutoSync	GPURepair	
	CUDA	CUDA	OpenCL
Total benchmarks	266	266	482
Verified by GPUVerify	152	152	331
No changes made by the tool	146	138	293
Changes recommended by the tool	0	**13**	25
Errors	6	**0**	10
Timeouts (300 seconds)	0	1	3
Data Race/Barrier Divergence Errors identified by GPUVerify	89	89	69
Repaired by the tool	31	**43**	33
Repaired using grid-level barriers	0	**15**	0
Could not be repaired by the tool	10	20	34
Errors	14	**0**	0
Timeouts (300 seconds)	34	**11**	2
Unrepairable errors identified by GPUVerify	25	25	82
Handled gracefully by the tool	0	**25**	82
False Positives	24	**0**	0
Errors	1	**0**	0

Runtime



Runtime in seconds (Log Scale)



Runtime in seconds - Repair Candidate (Log Scale)

Try GPURepair

Paper: https://arxiv.org/abs/2011.08373

Source Code: https://github.com/cs17resch01003/gpurepair

VMCAI 2021 Artifacts: https://zenodo.org/record/4276526