Finding data races in OpenCL kernels

Daniel Liew
Cristian Cadar
Alastair Donaldson

Imperial College London

The Problem

Running parallel programs written in OpenCL for GPUs has the potential for increased performance. However writing correct OpenCL programs can be difficult. In particular, data races are difficult for programmers to debug.

This kernel has several data races. Can you find them all?

```
#define tid get_global_id(0)
#define N get_global_size(0)
__kernel void dotProduct(__global int* restrict A,
                         __global int* restrict B,
                         __global int* restrict C)
{
    C[tid] = A[tid] + B[tid];
    barrier(CLK_GLOBAL_MEM_FENCE);

    for (int i=N/2; i > 0; i >>=1)
    {
        if ( tid <= i )
            C[tid] += C[tid + i];

        barrier(CLK_GLOBAL_MEM_FENCE);
    }
}
```

The Research Question

Various techniques exist for finding data races in OpenCL kernels. Two such techniques are

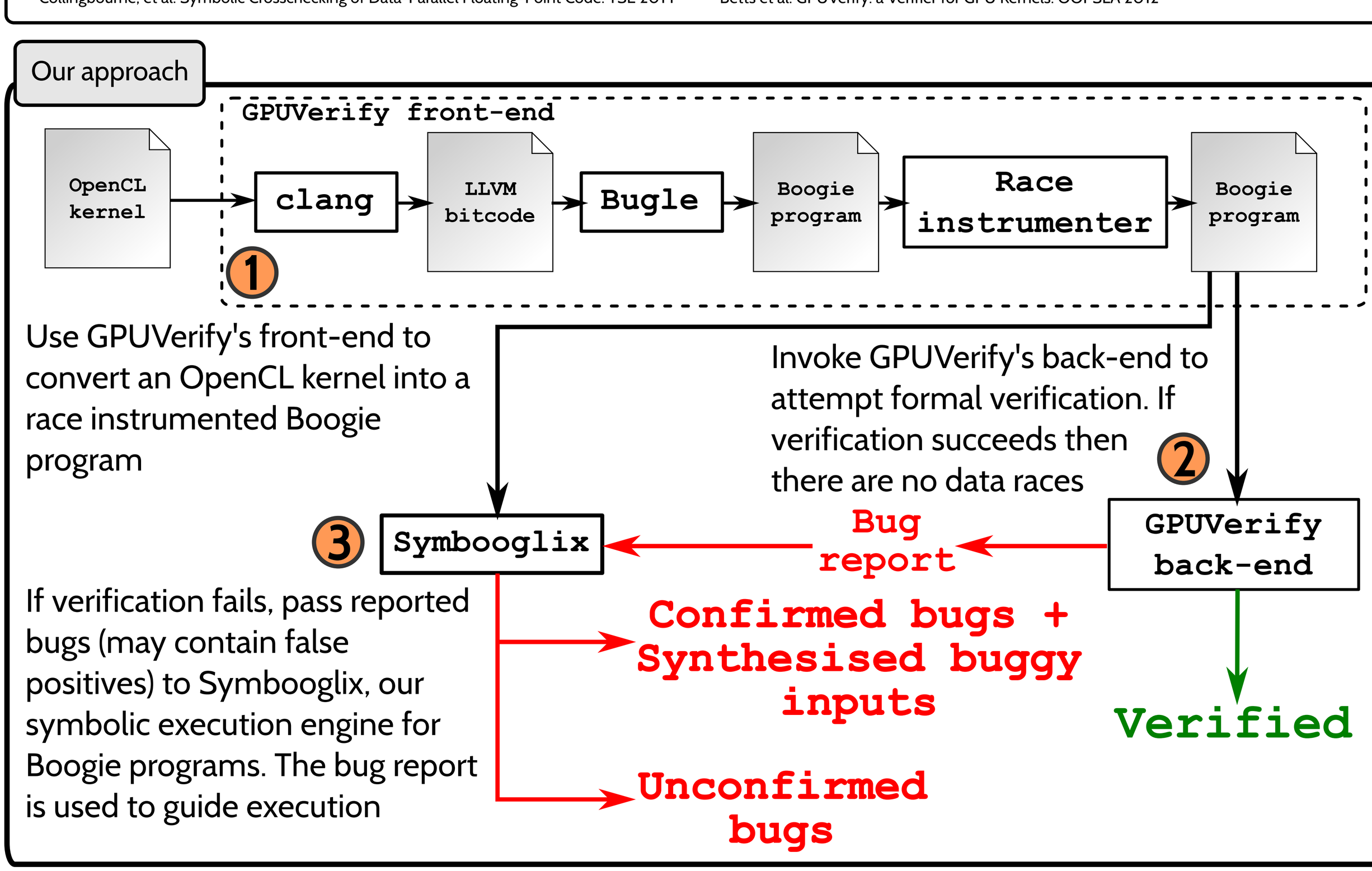
- Symbolic execution (e.g. KLEE-CL[1]) which is usually precise but scales poorly (executes all work-items explicitly)
- Formal verification (e.g. GPUVerify[2]) which is usually imprecise (uses abstractions) but scales well

Is it possible to combine these techniques in a way that can find bugs more effectively?

[1] Collingbourne, et al. Symbolic Crosschecking of Data-Parallel Floating-Point Code. TSE 2014

[2] Betts et al. GPUVerify: a Verifier for GPU Kernels. OOPSLA 2012

Our approach



This poster was produced for the Google 2014 poster competition

This work is generously funded by an EPSRC CASE studentship in collaboration with ARM