Oclgrind: An Open Source SPIR Interpreter and OpenCL Device Simulator

James Price and Simon McIntosh-Smith

Department of Computer Science, University of Bristol

Introduction

Oclgrind is an open source OpenCL device simulator, built around an interpreter for SPIR (Standard Portable Intermediate Representation). The project aims to provide a platform with which a variety of useful tools can be created to aid OpenCL application development and research efforts. Oclgrind is available via GitHub and is offered under a BSD license.

Implementation

The execution of kernels is simulated on an abstract OpenCL device. This device exhibits the high-level characteristics of the OpenCL execution and memory models, but does not model any architecture specific features such as SIMD execution or cache hierarchies. The simulation understands the concepts of work-items, work-groups and the different memory address spaces and synchronisation constructs available in OpenCL, and therefore kernel execution is semantically correct with respect to the OpenCL standard.

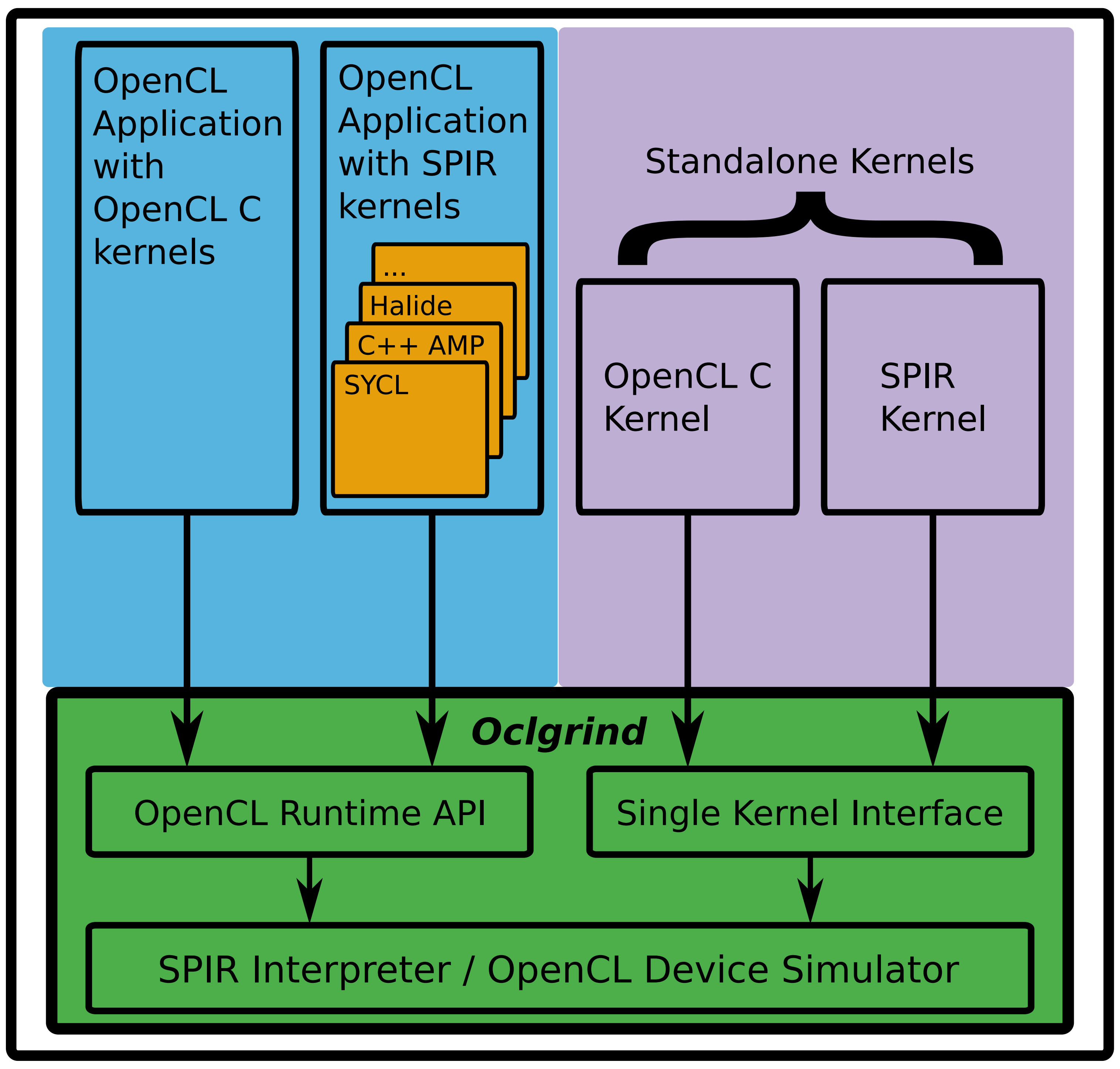
OpenCL Application with OpenCL C kernels
OpenCL Application with SPIR kernels
...
Halide
C++ AMP
SYCL
Standalone Kernels
OpenCL C Kernel
SPIR Kernel
Oclgrind
OpenCL Runtime API
Single Kernel Interface
SPIR Interpreter / OpenCL Device Simulator

Figure 1: High-level description of different Oclgrind workflows

There are two mechanisms for simulating kernel execution with Oclgrind. A kernel can be executed in isolation, by describing the NDRange information and kernel argument data in a simple configuration file (shown in figure 2). Oclgrind also provides an implementation of the OpenCL 1.2 runtime API, which allows existing OpenCL applications to target the simulator without requiring any modifications. OpenCL programs can be presented to Oclgrind in either OpenCL C or SPIR form, allowing the simulator to be targeted from applications that use high-level programming models such as SYCL, C++ AMP or Halide.

```
example.cl  # File containing OpenCL program
example     # Name of kernel to run
1024 1 1    # NDRange
16   1 1    # Work-group size

# First argument - 4K buffer initialized with range
<size=4096 range=0:1:4095>

# Second argument - 4K buffer initialized with zeros
<size=4096 fill=0>

# Third argument - single value interpreted as a float
<size=4 float>
42.874
```

Figure 2: An example configuration file for running kernels in isolation

Development Tools

A number of development tools have already been implemented within Oclgrind. In particular, the following utilities are currently available:

- Detecting memory access errors
- Detecting work-group divergence
- Detecting data-races
- Interactive kernel debugging
- Collecting histograms of instructions executed

Since the simulation being performed understands the OpenCL execution model, the error messages that Oclgrind produces when it encounters a problem with a kernel are OpenCL-aware, and contain all the information needed to pin-point the problem. Figures 3 and 4 show examples of the information provided when Oclgrind detects an error while executing a kernel.

```
Invalid read of size 4 at global memory address 1000000000040
    Work-item:  Global(16,0,0) Local(0,0,0)
    Work-group: (16,0,0)
    Kernel:     vecadd
    %tmp8 = load float addrspace(1)* %tmp7, align 4, !dbg !47
    At line 16 of input.cl
```

Figure 3: Example error message for an invalid memory access

The interactive debugger provides a simple, GDB-style interface to step through kernel execution. This can make it easier to diagnose the cause of any problems that Oclgrind detects, or to debug functional issues with a kernel. As well as stepping through kernel source code, the debugger allows you to set breakpoints, switch between work-items, view stack traces and inspect variables or memory. If Oclgrind is interpreting SPIR generated by a high-level programming model, the debugger will step through the execution of SPIR instructions, instead of OpenCL C source code.

```
Read-write data race at global memory address 1000000000004
    Work-item:  Global(2,0,0) Local(0,0,0)
    Work-group: (2,0,0)
    Kernel:     global_read_write_race
    %tmp6 = load i32 addrspace(1)* %arrayidx, align 4
    At line 6 of input.cl

    Race occured with work-item (1,0,0)
    store i32 %tmp6, i32 addrspace(1)* %arrayidx3, align 4
    At line 7 of input.cl
```

Figure 4: Example error message for a data-race

Future Work

Future work on Oclgrind will involve adding new debugging aids such as detecting use of uninitialised variables and additional features in the interactive debugger. We will also explore the use of Oclgrind for profiling and performance analysis of OpenCL applications, such as divergent branch analysis and profiling memory accesses. When SPIR is updated to support OpenCL 2.0 language features, we will also update Oclgrind to support OpenCL 2.0 applications. As Oclgrind is an open source project, contributions from other members of the OpenCL community are always welcome, along with bug reports and feature suggestions.

github.com/jrprice/Oclgrind

University of Bristol

UOB HPC

This work was partly funded by Imagination Technologies. We would also like to thank Codeplay for providing useful feedback about Oclgrind.