The Relevance of OpenCL to HPC


preney@sharcnet.ca

School of Computer Science
University of Windsor
Windsor, Ontario, Canada

Copyright © 2015 Paul Preney. All Rights Reserved.

March 4, 2015
Today’s high performance computing programs are evolving to be increasingly more parallel, increasingly more wait-free, increasingly deployed on many different kinds of hardware including general purpose CPUs, GPGPUs, FPGAs, other custom specific-purpose hardware, etc. The OpenCL standards are platforms providing interfaces that enable deployment of programs to virtually any heterogeneous computing device. The OpenCL standard defines a highly-vectorizable programming language, OpenCL C, which enables the deployment of programming logic to arbitrary hardware without requiring low-level, “machine-coding” knowledge of such. The OpenCL standard is a critical component of exascale initiatives given that it is hardware neutral, with significant support and participation from all the major processor vendors. Unfortunately the main source of information about OpenCL is in the form of its final specifications so there is a lot of misinformation about it. This talk will explain the relevance of the OpenCL standard to the HPC community, and offer a glimpse into what high-level abstractions for OpenCL, under development by software engineers, might look like.
Presentation Overview

1. HPC and OpenCL
2. The Relevance Of OpenCL To HPC
3. A Glimpse Into Possible High-Level Abstractions
4. The Future and Discussion
5. References
HPC and OpenCL

- What is HPC?
- What is OpenCL?
- What is OpenCL C?
- What is OpenCL C++? (NEW)
- What is SPIR?
- What is SPIR-V? (NEW)
What is HPC?

High Performance Computing (HPC) is the use of parallel computation to solve problems efficiently and reliably.

Some organizations and meta-organizations that provide access to HPC resources and services to researchers at academic institutions are:

- SHARCNET
- Compute Ontario (SciNET, HPCVL, and SHARCNET)
- Compute Canada (ACEnet, Calcul Québec, Compute Ontario, WestGrid)
What is OpenCL?

Open Computing Language (OpenCL):

- is open and royalty-free standard
- for use with CPUs, GPUs, and any other computational hardware
- provides portable efficient heterogeneous hardware access via a subset of ISO C99 (with parallel extensions)
- uses programming abstractions that transparently leverage SIMD and/or threading parallelism on the back-end
  - N.B. Through the Khronos Group, all hardware vendors decide on the abstractions used.
- inter-operates with graphics APIs (e.g., OpenGL)
- with WebCL can be used over the Internet (e.g., via a web browser)

[6, §1], [8, §1], [9, §1], [10, §1], [11, §1]
What is OpenCL? (con't)

OpenCL enables heterogeneous platform computations by enabling one to:

- **discover** the computational hardware components of a system,
- **query hardware characteristics** to select proper code and/or to exploit unique hardware features,
- **compile programs and extract functions** to run (i.e., “kernels”) and **asynchronously call those kernels** on the target hardware, and,
- **control the ordering** of kernel executions and **memory operations** on desired hardware components.

[12, §1.1]
What is OpenCL? (con't)

OpenCL as Parallel Language Backend

- WebCL: JavaScript binding for initiation of OpenCL C kernels
- Halide: Language for image processing and computational photography
- MulticoreWare: Open source project on Bitbucket
- Embedded array language for Haskell
- Java language extensions for parallelism
- River Trail: Language extensions to JavaScript
- Compiler directives for Fortran, C and C++
- PyOpenCL: Python wrapper around OpenCL
- Harlan: High level language for GPU programming

OpenCL provides vendor optimized, cross-platform, cross-vendor access to heterogeneous compute resources

[13, Slide 20]
What is OpenCL C?

OpenCL C:

- is used to create kernels that are executed on OpenCL devices
- is a programming language based on C99 (i.e., ISO/IEC 9899:1999)
  - only a subset of C99 is supported
  - but, for example, adds vector types, atomic operations, some new types
  - no recursion, and no function pointers
  - restrictions on pointers, no struct bit-fields, many C Standard Library functions are not available,
- memory consistency model is based on §7.17 in C11 (i.e., ISO/IEC 9899:2011)

[5, §6, §6.9, §6.13.11]
What is OpenCL C++? (NEW)

OpenCL C++:

- is currently a provisional specification
- is used to create **kernels** that are executed on OpenCL devices
- is a programming language **based on C++14**
  - N.B. The provisional spec cites ISO/IEC JTC1 SC22 WG21 N3690
  - only a **subset of C++14** is supported
- C++14 features not supported are:
  - **dynamic_cast**, type identification, recursive function calls, **new** and **delete**, **noexcept**, **goto**, **register**, **thread_local**, **virtual**, function pointers, exception handling, C++ Standard Library
- Supports **templates** and **metaprogramming**.

[4, §1, §18]
What is SPIR?

Standard Portable Intermediate Representation (SPIR):

- is a **partially compiled, binary OpenCL interchange format**
  - i.e., it efficiently maps OpenCL C into LLVM IR [14]

- is **vendor-neutral** but is not OpenCL C source code

- is designed to be a **compiler target format** for programming languages

- is designed to **support vendor extensions**

- is designed to be **efficiently loaded** by an OpenCL implementation

- is an **extension** to the OpenCL standard

[7, §1]
SPIR-V:

- is currently a provisional specification
- is an intermediate language (IL) for graphical shaders and compute kernels
- is only conceptually similar to SPIR in that it is an IR language
- is distinct from SPIR as it does not rely on or require the LLVM IR in any way

[2]

SPIR-V is a “single, common language for multiple languages feeding multiple drivers.” [1, p.2]
What is SPIR-V? (NEW) (con't)

[3, Slide from announcement]
The Relevance Of OpenCL To HPC

- Why Does OpenCL Matter To Me?
- Why Does OpenCL Matter To HPC?
- Programming With OpenCL
OpenCL matters to you because your OpenCL programs:

- can be run on/across many different devices without re-designing, re-factoring, or re-writing it
- are compiled and deployed using standard, non-proprietary API calls, OpenCL C/C++, and/or SPIR/SPIR-V
- have the ability to query and exploit device-specific abilities while being able to remain non-proprietary
Why Does OpenCL Matter To HPC?

OpenCL matters to HPC because:

- **heterogeneous computing** is the future
  - traditional, exascale, big data, digital humanities, etc. have **different** kinds of computing hardware needs

- power is very costly —power savings is essential
  - e.g., OpenCL deploys to FPGAs —not just GPUs and CPUs

- maximizing parallel performance is crucial
  - OpenCL's C, C++, SPIR, and SPIR-V are highly vectorizable

- OpenCL enables programs to be able to be **deployed** on and still exploit future hardware designs

- OpenCL has major traction and support with hardware vendors
Each OpenCL standard should be seen as a distinct release.

Vendors will release OpenCL implementations compliant with specific OpenCL standards.

Newer standards are not necessarily better or worse — they are different.
In terms of deploying your OpenCL code on any devices:

- Think of OpenCL C as very high-level, heterogeneous, portable, and human-readable “parallel assembly languages”.
- Think of SPIR and SPIR-V as heterogeneous and portable machine codes.
Beyond the near-term, an OpenCL user does not want to write raw OpenCL code when developing software.

- Using abstractions, libraries, middleware, and tools will be preferred.
- The latter is well-suited for computer scientists and software engineers.
A Glimpse Into Possible High-Level Abstractions

- Example OpenCL C
- Possible C++ Host Code Example
- Possible OpenCL C++ Example
Example OpenCL C

Example courtesy of AJ Guillon.

```c
__kernel sum(
    __global float* out,
    __global float* in1,
    __global float* in2,
    uint64_t length
)
{
    size_t idx = get_global_id(0);
    if (get_global_id(0) >= length)
        return;
    out[idx] = in1[idx] + in2[idx];
}
```
// Get a collection of devices that support OpenCL
devices myDevices = get_devices();

// Compile a program for the devices
program my_program = compile("sum.cl");

// Extract the kernel we want
kernel sum = my_program.extract("sum");

// Allocate three arrays of values
auto x = allocate_vector<float>(100);
auto y = allocate_vector<float>(100);
auto z = allocate_vector<float>(100);
// Read the values for x, and y
x = file.read("x");
y = file.read("y");

// Do z = x + y on the first device
sum.call(my_devices[0])(z, x, y, x.length());

// Do y = x + x on the second device
sum.call(my_devices[1])(y, x, x, x.length());

// Print out the results
std::cout << "Z: " << z << std::endl;

// Print out the results
std::cout << "Y: " << y << std::endl;
Example courtesy of AJ Guillon.

```cpp
typedef compute_foo_strategy<
    conditional< numeric_limits<double>::supported() &&
        ! numeric_limits<double>::emulated() >,
    double,
    float>,
    hardware_traits::scalar_code_preferred() >
    compute_foo_type;

/* Now it is a simple matter to call foo */
long result = compute_foo_type::foo(x, y);
```
Table of Contents

4 The Future and Discussion
What is next for OpenCL and HPC?

Discussion and Q&A with guest: AJ Guillon

- AJ is a member of the Khronos OpenCL Standards Committee.
Thank you.
References


