The Challenges of Delivering Massive Parallelism to Real-World Software

Andrew Richards, CEO, Codeplay

UKMAC, May 2016
• CEO Codeplay
• Chair SYCL group in Khronos
• Chair HSA System Runtime Group
• Chair HSA Tools Group
Codeplay: Heterogeneous Systems Experts

• GPGPU and graphics languages, compilers, debuggers, testing
• Work on open-source development tools
• Infrastructure for testing and tracking performance

• Collaborative R&D with academics and companies
• Defining new standards
• Machine vision and AI libraries
• Graphics and games software

• 45 expert development staff, based in Edinburgh, Scotland
Connecting All Parties

Codeplay’s far-reaching involvement

- Algorithms
- Software/Middleware Tools & layers to Standards
- Processors IP (CPU, GPU, DSP, VPU, ViSP)
- Semiconductors
- Module Integration
- Benchmarking
- Conformance
- Testing
- ISO 26262
- Open Standards

OSA Connectivity Initiative - Connected Vehicles

OpenCL

LLVM

SYCL
Open Standard Acceleration Products

- **Solution – ComputeAorta™**: Delivers low-level open-standards, e.g. OpenCL, on custom processor cores.
- **Solution - ComputeCpp™**: Enables high-level performance-portable C++.
- **Solution – LLVM**: Existing open-source compiler can be targeted at any processor core.

**Adaptable, Open, Standard**
- Growing ecosystem of open-source and commercial software
- Easy to hook-in pre-existing solutions
- Easy to add more cores and more compilers

**Range of open-source projects** for vision, AI, easy parallelism

- **TensorFlow™**
- **VisionCpp™**
- **ComputeAorta™**
- **ComputeCpp™**
- 🔗 **OpenCL**
- 📗 **OpenCV**
- 🗿 **OpenVX**
- 🟢 **Sycl**
- 🟢 **C++ 17**
- 🟢 **Proprietary Processor**
- 🟢 **LLVM**
- 🟢 **CPU**
- 🟢 **DSP**
- 🟢 **GPU**
Massive parallelism and heterogeneous processing

GPGPU is now mainstream and commercial
Heterogeneous processing

- Smartphones
- Drones
- Self-driving cars
- Datacentres
- Vacuum cleaners

(it isn’t just GPUs)

- AI
  - Neural networks
  - Machine vision
  - Machine learning

The killer app for massive parallelism turns out to be AI in all its forms
What do you want to do with your code?

What people think about when they first design a parallel model:

• Compile
• Run

What developers really need to do with a parallel model:

• Profile
• Debug
• Validate safety
• Check-in to source repos
• Do ‘merge requests’
• Validate the code matches a known algorithm
• Static analysis checking
• Run-time validation
• Link with other projects
• Compose generics from different sources
• Call between different languages
• Run in a virtual machine
• Port between different kinds of platforms
• Try out different optimization strategies
• Embed documentation
• Reproduce bugs
• Track versions and revisions
• Benchmarking and performance tracking
Three Key challenges

1. **Safety**
   - How do we deliver massively-parallel artificial intelligence into self-driving cars that can operate without an expert monitor?

2. **Performance portability**
   - How can developers write code that performs well everywhere?

3. **Interoperability**
   - Enabling different libraries, tools and languages to work with a variety of different accelerators

*In reverse order of difficulty!*
Safety

Safety systems are a key market for massively-parallel vision and AI
How do we make massively parallel AI safe?

• *Left as an exercise for the reader...*
What we know we need to do for safety

• Validate that software:
  • Has been fully tested (e.g. code coverage tools)
  • Completes in predictable time
  • Does not exhibit undefined behaviour
What we don’t yet know we need for safety
Performance portability

How to write code that performs well everywhere
Performance portability is a hard problem!

1. Do we: provide a totally-performance portable programming model?
   • This is magic

2. or: Do we provide all the capabilities we need?
   • A high-performance, but portable, low-level API, e.g. Vulkan
   • A standardized, composable, programming model
   • Performance portable algorithmic skeletons (e.g. C++ 17 Parallel STL)
Processor types

• CPU
• GPU
• DSP
• + highly parallel vision DSPs
• FPGAs
• PIM – Processor-In-Memory

• SIMT
• MIMD
• SIMD

How do we abstract across all this difference, but still provide programmers enough control to be able to optimize?
Performance portability is a hard problem!

1. Do we: provide a totally-performance portable programming model?
   • This is magic

2. or: Do we provide all the capabilities we need?
   • A high-performance, but portable, low-level API, e.g. Vulkan
   • A standardized, composable, programming model
   • Performance portable algorithmic skeletons (e.g. C++ 17 Parallel STL)

This is the SYCL vision
The standards efforts

Experts in different areas can produce their own solutions that work together

→ Independent innovation
→ Interoperability
OpenCL

• The most widely-implemented open heterogeneous standard
  • Supports GPUs, CPUs, FPGAs and DSPs
  • Latest released spec: OpenCL v2.1, with SPIR-V

• Open-source:
  • Wide support, including implementations: pocl and Clang OpenCL C front-end

• Low-level and can be customized per-architecture
  • In conjunction with SPIR-V, OpenCL is now: a low-level API for optimizing, and an enabling platform for higher-level abstractions
SPIR-V

• Allows any language (almost) to run on any device (almost)
• Can be used within both graphics and compute APIs
• Original SPIR was based on LLVM, but SPIR-V is totally independent of any one compiler technology
  • Now, the intermediate representation is independent of both the compiler front-end and back-end
• Continuing to be developed and supported by both OpenCL and Vulkan
Vulkan

- Mostly aimed at graphics, but does support compute
- Key benefit: low-CPU-overhead and low-latency
  - Explicit API that gives developers a lot of control...
  - ... but they have to do a lot of work
  - Suitable for engine and tools developers
  - Developers using Vulkan have good control over memory usage and expensive computation, such as shader compilation
- Uses SPIR-V for shaders/kernels
SYCL

• Higher-level single-source C++ on top of OpenCL
• Provides a standard platform for even higher-level C++, such as C++ 17 Parallel STL
• Enables C++ source, with lots of templates and classes to be accelerated on a wide range of OpenCL devices
• Current released spec is for OpenCL v1.2
  • but OpenCL 2.x capable version on its way
HSA

• One of the big problems of offloading to accelerators is the high cost of offload:
  • Moving data to the accelerator
  • May need to translate addresses and pointer sizes between CPU and accelerator
  • Going into OS kernel-level driver to start work or synchronize
  • High cost of compilation, especially for JIT languages
  • In a multi-accelerator system, this may get even more costly

• HSA solves this with *user-mode-queueing*, *user-mode synchronization*, and shared virtual memory
  • It also provides open-source software to help implement HSA

• *And, we’re even going further and working towards advanced tools standardization, such as profiling and debugging*
C++ standard

• The C++ standards body wants to standardize and support heterogeneous acceleration

• Some of the different efforts:
  • C++ 17 Parallel STL – already voted through and implementations available
  • C++ 20: next version of C++ standard targeted for 2020
  • SG14: open C++ standardization study group looking at some of these issues

• How do you standardize a *native* programming model for platforms with multiple processor architectures, some of which may not be known at compile time?
Safety standards

• The next challenge is enabling highly parallel software to work in safety-critical situations:
  • Self-driving cars and driving aids
  • Medical device applications

• e.g.
  • OpenGL SC – safety-critical graphics API
  • ISO 26262 – the automotive software standard
  • Misra C and C++

• Safety standards usually focus on process, not underlying technology
And there are others...

- OpenMP
- OpenAcc
- OpenVX - vision
Why so many standards?

• We have loads of standards for CPUs, for solving different kinds of problems

  • If we want heterogeneous systems to accelerate CPU software, we need to support as many languages and tools as CPUs do

• Without this work, we would have to decide to make no further improvements in performance for most applications domains
What do we need to standardize?

Data management; Memory models; Processor capabilities; How to execute code; Synchronization; Languages; Profiling; Debugging; Algorithmic skeletons
Processor capabilities

• Recursion?
• Memory allocation?
• Datatypes? 32-bit float, 64-bit float, 64-bit int? 16-bit int?
• Precision of floating-point operations
• Exception-management
Data management

• To scale processor performance, memory systems need to scale, too
• Multiple memory systems
• Referencing data
• Allocating data
• Datatype or address range to identify data location?
Memory models

• Caches and compilers increase performance
  • But they do it by re-ordering or removing memory accesses
  • How does this work between different threads?

• If different processors use different caches that change memory ordering per-core, how can you ensure all processors see consistent data?
  • A big issue for atomic operations, but also impact of atomics on non-atomic operations
  • Affects the compiler, too
How to execute code

• To execute code on a processor core, need to:
  • Determine the processor core to run the code on
  • Compile the code for that core, either from source a portable binary format
  • Load the compiled binary into instruction memory for the core
  • Determine and pass the arguments for the code:
    • References to data that will be shared
    • Value data to be copied
    • Iteration space for cores to iterate through
  • Add the code to a queue of work for the core, with all arguments

• These are all expensive operations
  • Ideally, want to prebuild some of this in advance, to reduce latency
Synchronization

• Independent forward progress
  • Normal threading models assume *independent forward progress*
  • SPMD machines like GPUs execute in lock-step so cannot guarantee independent forward progress

• Pre-emption:
  • without the ability to switch a core from one thread to another, a thread can block a whole core

• User-mode synchronization:
  • Entering the OS kernel to synchronize is a big performance hit. Providing this in user mode is much lower latency
Languages

• Most software is not written in C, or even C++
  • How do we accelerate all the other languages out there?
  • In reality, we need to change languages to enable heterogeneous acceleration in a useful way
Profiling

• Developers can’t optimize their code without understanding how it runs
  • For parallel software, the best information seems to be time-line profilers that show progress of the code over time. This shows parallel threads well

• Types of data that we want from a device:
  • Performance counters
  • Timed events showing when tasks start/stop
  • Occupancy of cores and core frequencies
  • Timed samples of what code is executing over time (ideally with a call stack)
Debugging

• We can now enable compiler developers to produce their own compilers for their own languages that will target a wide range of processor cores, with SPIR-V or HSAIL
  • *But:* How do we let them write debuggers, too?
  • This is a critical requirement to really enable any language to be accelerated on any device

• Requirements:
  • Debugging information in the intermediate representations and code binary
  • APIs to enable single-stepping, breakpoints and access to data
  • Debuggers that can cope with different threads executing different cores in potentially different execution environments
Algorithmic skeletons

- When we examine how programs run on heterogeneous devices, we find that:
  - Algorithms need to adapt to the different characteristics of cores
  - Commonly, algorithms vary by the different sizes of local memories
  - Also, capabilities of the processor cores
  - We would like to be able to write code and it adapts the algorithm to the system:
    ```
    parallel_sort<MyData>(myData);
    ```
- e.g.:
  - C++17 Parallel STL
Standards: How to get involved

• Most standards bodies are membership organizations
  • Can become a member
  • There are ways to feed back comments to standards bodies. Best to put in feedback on provisional specs if possible
  • Some bodies allow academic members to provide useful input

• Open-source projects help validate standards and drive standards forwards
  • Standards rise and fall by their ecosystems
Questions?