Delivering Heterogeneous Programming in C++

Duncan McBain, Codeplay Software Ltd.

About me

- Graduated from Edinburgh University 3 years ago
- Postgrad course got me interested in GPU programming
- Worked at Codeplay since graduating
- Research projects, benchmarking, debuggers
- Most recently on C++ library for heterogeneous systems

© 2016 Codeplay Software Ltd.

Contents

- What are heterogeneous systems?
- How can we program them?
- The future of heterogeneous systems

© 2016 Codeplay Software Ltd.

Heterogeneous device

- By this, I mean devices like GPUs, DSPs, FPGAs...
- Generally a bit of hardware that is more specialised than, and fundamentally different to, the host CPU
- Specialisation can make it very fast for certain tasks
- Can also be harder to program because of specialisation

© 2016 Codeplay Software Ltd.

What are heterogeneous systems

- A heterogeneous system is therefore a system composed of:
  - A host CPU
  - One or more heterogeneous devices
  - A way to transfer data between them (be it shared memory, like a mobile phone, or a bus, e.g. PCIe)
- Mobile phones are actually a good example of heterogeneous systems

© 2016 Codeplay Software Ltd.

Some definitions

- *Host*
  - The CPU/code that runs on the CPU, controls main memory (RAM), might control many devices
- *Device*
  - A GPU, DSP, or something more exotic
- *Heterogeneous system*
  - A host, a device and an API tying them together

© 2016 Codeplay Software Ltd.

Some definitions

- *Kernel*
  - Code representing the computation to be performed on the device.
- *Work group*
  - A collection of many *work items* executing on a *device*. Has shared local memory and executes same instructions

© 2016 Codeplay Software Ltd.

Some definitions

- *Work item*
  - A single thread or task on a device that executes in parallel
- *Parallel for*
  - Some collection of *work items*, in many *work groups*, executing a *kernel* in parallel. In general, cannot return anything, and must be enqueued asynchronously

© 2016 Codeplay Software Ltd.

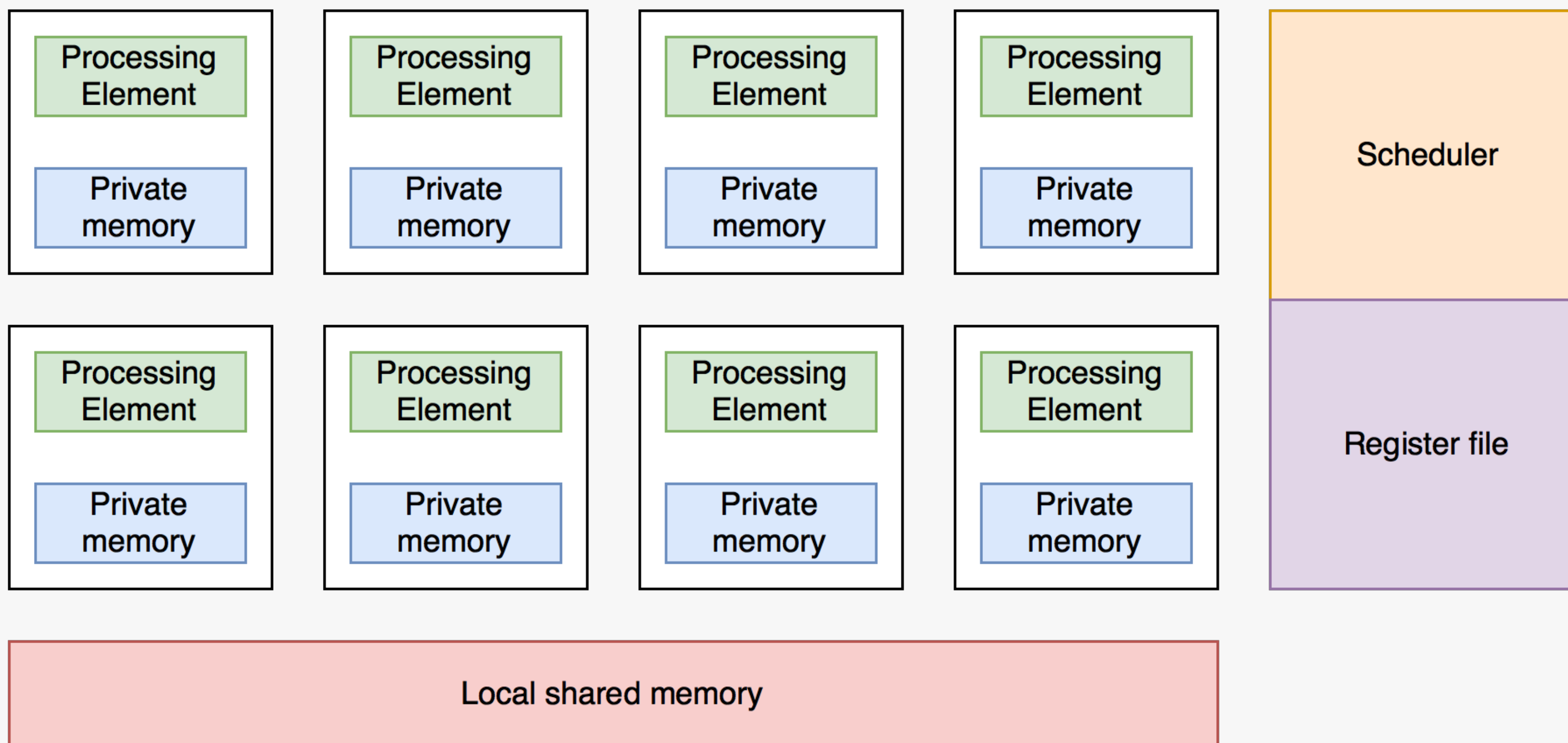
Example heterogeneous device

- CPUs today can execute instructions out-of-order, speculative execution, branch prediction
- Complexity hidden from programmer
- Contrast with e.g. GPU
- Most GPUs have many execution units (~100s), but far fewer scheduling units
- Here, all cores must execute same instructions on different data

© 2016 Codeplay Software Ltd.

Example continued

- Modern GPUs look something like this:



© 2016 Codeplay Software Ltd.

Example continued

- A modern GPU will typically have many (~32) processing elements in a block
- There will then be many blocks per GPU
- There is memory shared across all blocks, but this is very slow
- This memory is not generally shared with host CPU
- Fastest to swap out threads waiting for memory reads/writes for other threads that can do work

© 2016 Codeplay Software Ltd.

Other accelerators

- This was a GPU-like architecture, but the ideas apply across hardware
- For example, Digital Signal Processors (DSPs) have similarities like reduced instruction sets, maybe no virtual memory…
- So how do I program something like that?

© 2016 Codeplay Software Ltd.

How you program something like that

- Variety of APIs available
  - OpenMP/ACC
  - CUDA
  - OpenCL
  - SYCL
- Independent standards that work with languages, but are not core parts of a language

© 2016 Codeplay Software Ltd.

OpenMP/ACC

- “Directive-based” API – decorate standard C with pragmas telling the compiler how to parallelise
- Very easy to get started, falls back to linear execution when compiler doesn’t support it – might require no changes
- Limited in other areas – need OpenMP 4+ for “accelerator” support
- OpenACC is similar, originally targeted NVIDIA devices only

© 2016 Codeplay Software Ltd.

OpenMP 4 sample

```
#pragma omp target device
#pragma omp parallel for
for (j = 0; j < N; ++j) {
  int k = sin(PI/12 * j);
}
```

- Target device means use accelerator
- A parallel for is one of the most simple parallel constructs
- Code is calculating some trigonometry, though results are discarded

© 2016 Codeplay Software Ltd.

OpenMP example cont.

- Very simple to get started, as seen
- Allows for more complicated directives – can split work between blocks for better scheduling on GPU for example
- However, still somewhat limited – can only affect code in pragma blocks
- Lacking in fine-grained control which can lessen your options for performant code

© 2016 Codeplay Software Ltd.

CUDA

- CUDA is a proprietary standard for running parallel code on NVIDIA GPUs only
- Since NVIDIA solely develop, performance can be great
- However, you are tied in to one platform
- CUDA allows you to control where data lives, when it is transferred and how code is executed
- Code written in one source file with explicit device annotations

© 2016 Codeplay Software Ltd.

CUDA sample

```
void square_complex_gpu(cufftDoubleComplex * x, int len)
{
  int blocksPerGrid = len / 256;
  int threadsPerBlock = 256;
  square_elem<<<blocksPerGrid, threadsPerBlock>>>(x, len);
  retval = cudaGetLastError();
  if(retval != cudaSuccess)
    fprintf(stderr,"kernel launch failure: %d\n",retval);
  cudaDeviceSynchronize();
}

__global__ void square_elem(cufftDoubleComplex *x, int length)
{
  int i = blockDim.x * blockIdx.x + threadIdx.x;
  double re = x[i].x * x[i].x - x[i].y * x[i].y;
  x[i].y = 2 * x[i].x * x[i].y;
  x[i].x = re;
}
```


© 2016 Codeplay Software Ltd.

CUDA sample cont

- Freely available from NVIDIA
- Somewhat easy to integrate – behaves much like ordinary C/C++, but requires an additional compile step using nvcc
- Can progressively parallelise by moving more and more code to CUDA over time

© 2016 Codeplay Software Ltd.

OpenCL

- OpenCL is a very similar standard to CUDA
- Cross-platform, open, freely implementable, developed by cross-industry group (Khronos)
- Allows for execution on GPU, DSP, even humble CPU
- Host-side C API, quite verbose, but exposes flexibility
- Device-side code written in subset of C99 with extensive maths library

© 2016 Codeplay Software Ltd.

OpenCL continued

- Restrictions on device include no function pointers and no recursion
- Hardware in most cases is simply not capable
- In past, so-called kernels were stored as C strings and compiled at runtime
- However, recent versions allow intermediate binary data – something like Java bytecode or LLVM IR – as mid-way point between plain source and device-specific binaries

© 2016 Codeplay Software Ltd.

OpenCL sample

```
__kernel void call_pow(__global float *input,
        __global float *output, int element_num) {
  int globalID = get_global_id(0);
  if(globalID < element_num)
    output[globalID] = pow(input[globalID],
        (input[globalID]/(globalID+1)));
}
```


© 2016 Codeplay Software Ltd.

Other APIs

- There are still other APIs that are all slightly different and solve different problems
- HSA, C++ AMP, Renderscript, Vulkan, DirectCompute…
- They are all relevant (and definitely worth looking into!)

© 2016 Codeplay Software Ltd.

Common ideas

- All these APIs are somewhat different, but often overlap:
  - Kernel code is separate and marked as such (CUDA, OpenCL)
  - Generally have separate memory between host and device (though not necessarily)
  - Work best when given broad arrays with same operation on each element
  - Fundamentally asynchronous – enqueue work & wait

© 2016 Codeplay Software Ltd.

SYCL

- SYCL is a newer spec from Khronos, similar to CUDA, but has lots of interesting and different features
- Based on idea that any code can be compiled to intermediate language, so why not C++?
- Can use C++ function objects to identify kernels in ordinary C++ code – even C++11 lambdas
- That way, SYCL code is also valid host code – don’t even need a device (though it will be slow)

© 2016 Codeplay Software Ltd.

SYCL continued

- Still maintains restrictions from OpenCL C
  - No recursion
  - No function pointers
  - Kernel return type must be `void`
- Uses the OpenCL API underneath to talk to devices and do work

© 2016 Codeplay Software Ltd.

SYCL continued

- C++ classes for each of the OpenCL types, wrapping them neatly
- Data controlled by `buffers` and `accessors`
- In contrast to OpenCL, where the programmer moves data around, SYCL lets you describe *where* you use data
- SYCL then ensures that the data is there for you
- Not just convenient – allows runtime to schedule efficiently

© 2016 Codeplay Software Ltd.

SYCL example

```
template <typename T, size_t N>
void simple_vadd(const std::array<T, N> &VA, const std::array<T, N> &VB,
                 std::array<T, N> &VC) {
  cl::sycl::queue q;
  cl::sycl::range<1> r{N};
  cl::sycl::buffer<T, 1> bA(VA.data(), r);
  cl::sycl::buffer<T, 1> bB(VB.data(), r);
  cl::sycl::buffer<T, 1> bC(VC.data(), r);

  q.submit([&](cl::sycl::handler &cgh) {
    auto pA = bA.template get_access<sycl_read>(cgh);
    auto pB = bB.template get_access<sycl_read>(cgh);
    auto pC = bC.template get_access<sycl_write>(cgh);

    cgh.parallel_for<class SimpleVadd<T>>(r,
    [=](cl::sycl::id<1> idx) {
      pC[idx] = pA[idx] + pB[idx];
    });
  });
}
```


© 2016 Codeplay Software Ltd.

SYCL continued

- This looks a lot like C++!
  - No separate kernel string like OpenCL C
  - No `__device` like CUDA
  - No pragmas like OpenMP
- So why am I showing you this?
- Currently, SYCL is an independent open standard (written by the Khronos group) – but what about ISO C++?

© 2016 Codeplay Software Ltd.

Parallel STL

- Part of C++17, this technical report extends many STL algorithms to the parallel domain
- As simple as adding an *execution policy* to the function call
- The policy is what allows the library writers to control how the code should be parallelised – e.g. POSIX threads, distributed computing… or dispatched to an OpenCL device

© 2016 Codeplay Software Ltd.

Parallelism in ISO C++

- Parallel STL makes it clear there is an appetite for improving ISO C++'s support for parallelism
- Already work happening in this area – for example, `std::future`
  - Can be used to hide the asynchronous part of heterogeneous code
  - Provides a clean demarcation of parallel code

© 2016 Codeplay Software Ltd.

Parallelism in Future C++

- There are lots of ideas at the moment, but no decisions
- Committee, as ever, wants to make sure the best proposals make it in
- I don't think any one API shown here tonight is the perfect solution, but I think there are lessons from each
- Ultimately requires input from community – anyone can write a proposal!

© 2016 Codeplay Software Ltd.

What about today?

- OpenMP implemented in many compilers
- OpenCL available from many vendors like Intel, AMD etc.
- CUDA available from NVIDIA’s website
- ComputeCpp, Codeplay’s implementation of SYCL, is available on our website
- Heterogeneous computing will only become more prevalent, so never too late to start?

© 2016 Codeplay Software Ltd.

What about tomorrow?

- C++17 ended up being a smaller release than anticipated
- Committee doesn’t want to introduce ideas that aren’t “battle-proven”
- Better to have some non-standard PoC out there early
- Other bodies not stopping, either – Khronos developing and improving standards all the time

© 2016 Codeplay Software Ltd.

Useful links

- http://www.nvidia.com/object/cuda_home_new.html
- http://openmp.org/wp/
- https://www.khronos.org/opencl/
- https://www.khronos.org/sycl
- https://github.com/KhronosGroup/SyclParallelSTL
- https://computecpp.codeplay.com/

© 2016 Codeplay Software Ltd.

@codeplaysoft

info@codeplay.
com

codeplay.com