XILINX
ALL PROGRAMMABLE™

Post-modern C++17 abstractions for heterogeneous computing with Khronos OpenCL SYCL

Ronan Keryell @ Dublin C++ User Group Meetup
Xilinx Research Labs

10/04/2017

Power wall & speed of light: the final frontier...

- Current physical limits
  - Power consumption
    - Cannot power-on all the transistors without melting (↝ *dark silicon*)
    - Accessing memory consumes orders of magnitude more energy than a simple computation
    - Moving data inside a chip costs quite more than a computation
  - Speed of light
    - Accessing memory takes the time of 10^4+ CPU instructions
    - Even moving data across the chip (cache) is slow at 1+ GHz...

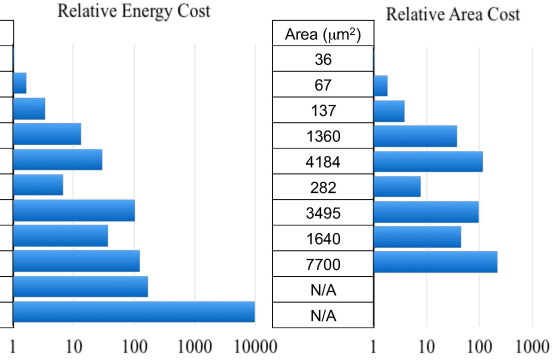
(Rather old) 45nm technology characteristics

Tutorial on "*High-Performance Hardware for Machine Learning*", William Dally at NIPS, December 7th, 2015

`https://media.nips.cc/Conferences/2015/tutorialslides/Dally-NIPS-Tutorial-2015.pdf`

Cost of Operations

Operation:	Energy (pJ)	Area (μm²)
8b Add	0.03	36
16b Add	0.05	67
32b Add	0.1	137
16b FP Add	0.4	1360
32b FP Add	0.9	4184
8b Mult	0.2	282
32b Mult	3.1	3495
16b FP Mult	1.1	1640
32b FP Mult	3.7	7700
32b SRAM Read (8KB)	5	N/A
32b DRAM Read	640	N/A

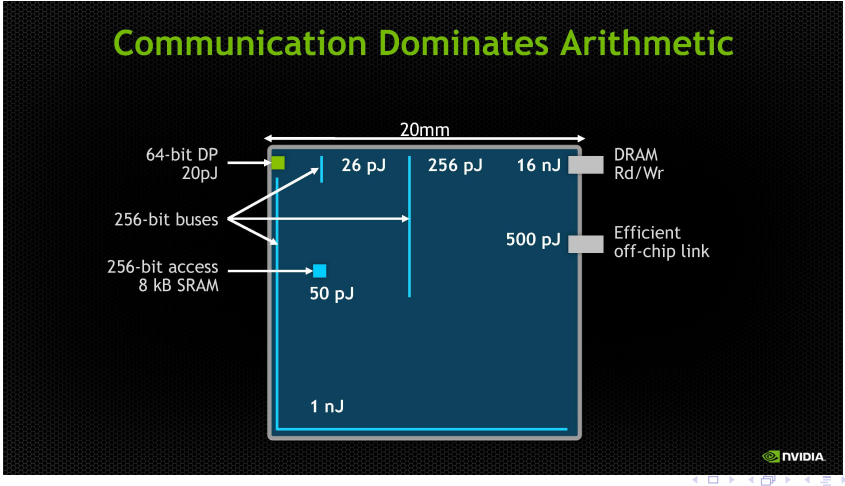


Energy numbers are from Mark Horowitz "Computing's Energy Problem (and what we can do about it)", ISSCC 2014
Area numbers are from synthesized result using Design Compiler under TSMC 45nm tech node. FP units used DesignWare Library.

Space-time traveling

"*Challenges for Future Computing Systems*", William J. Dally, January 19, 2015, HiPEAC 2015.

`http://www.cs.colostate.edu/~cs575dl/Sp2015/Lectures/Dally2015.pdf`



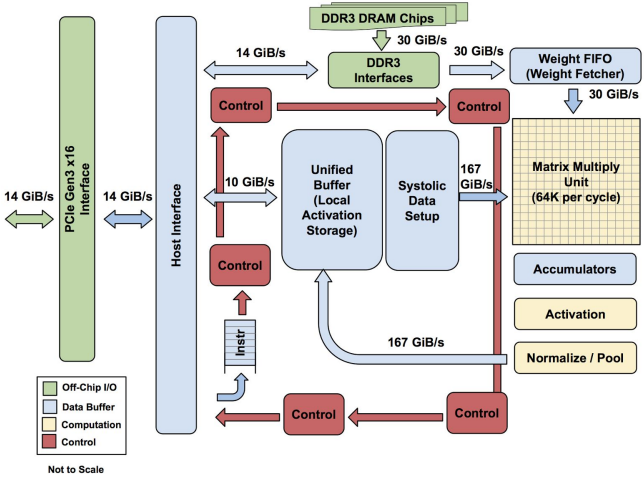
Power wall & speed of light: implications

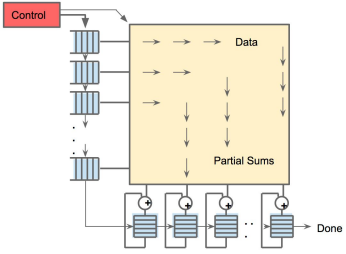
- Change hardware and software
  - Use locality & hierarchy
  - Massive parallelism
- NUMA & distributed memories
  - ⇝ New memory address spaces (local, constant, global, non-coherent...)
  - ⇝ PiM (Processor-in-Memory), Near-Memory Processing (in 3D memory stack controller)...
- Specialize architecture
- Power on-demand only what is required

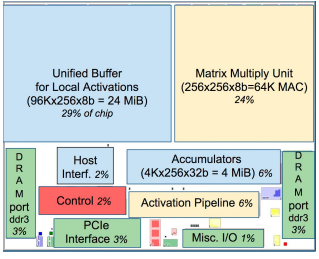
Nice take-away: the battery limitation may produce better programmers in the future ☺

From AMD Fiji XT GPU (2015)...

... Google Tensor Processing Unit (TPU)

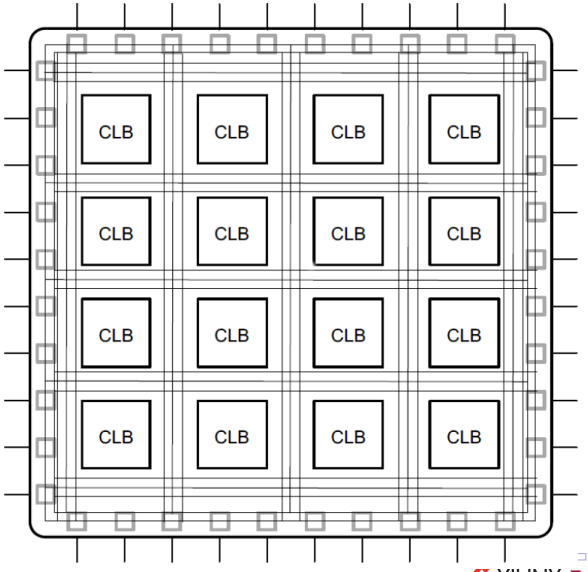




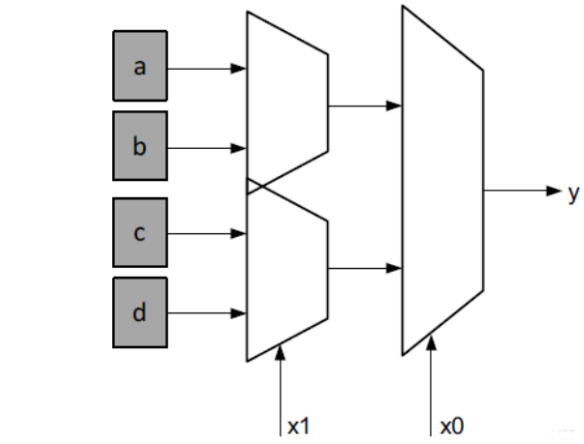


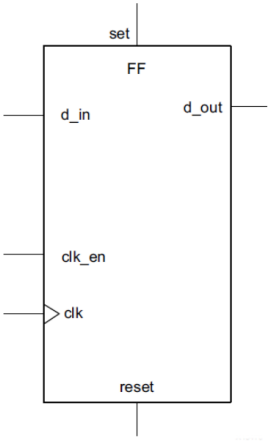
https://en.wikipedia.org/wiki/Tensor_processing_unit

... to Field-Programmable Gate Array (FPGA)



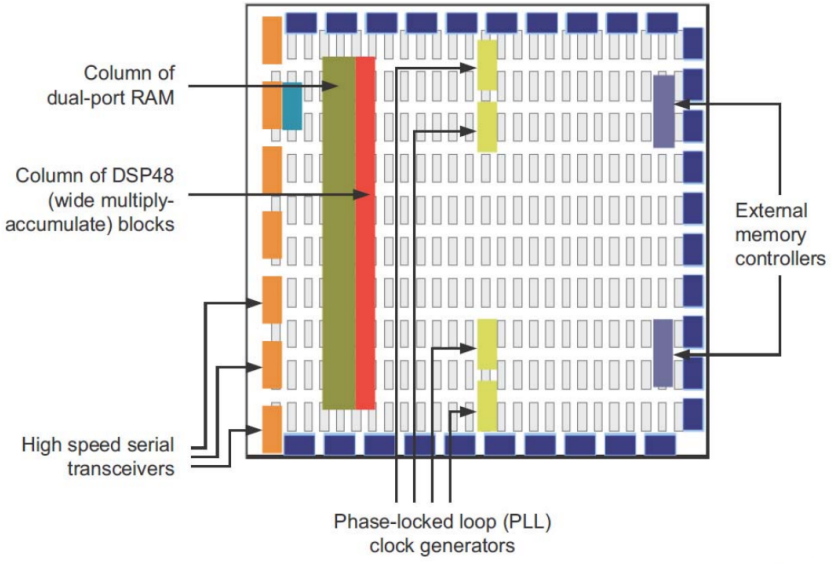
Basic architecture = Lookup Table + Flip-Flop storage + Interconnect



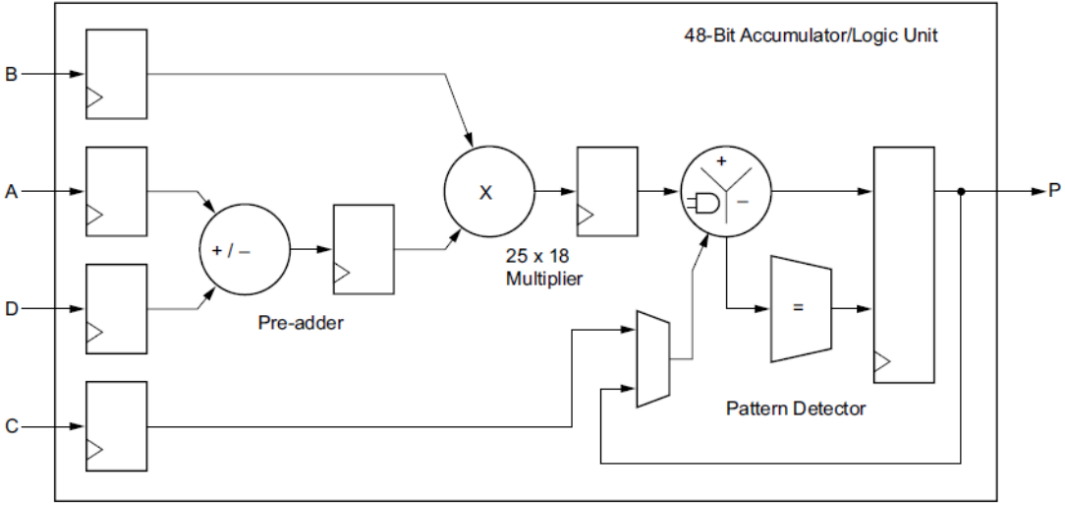


Typical Xilinx LUT have 6 inputs

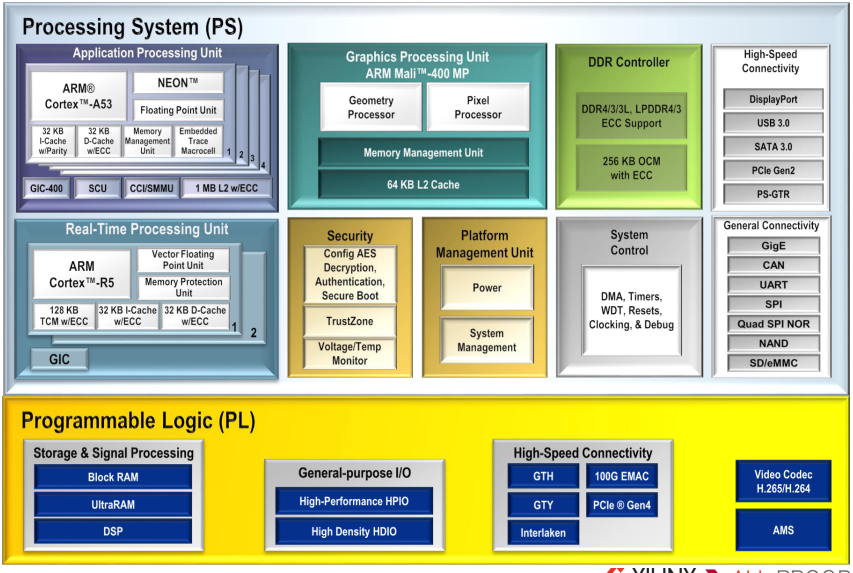
Global view of programmable logic part



DSP48 block overview



...FPGA in MPSoC: Xilinx Zynq UltraScale+ MPSoC



¿¿¿ How to avoid Frankenstein programming ???

Outline

Connecting Software to Silicon

COLLADA

3D for the Web
- Real-time apps and games in-browser
- Efficiently delivering runtime 3D assets

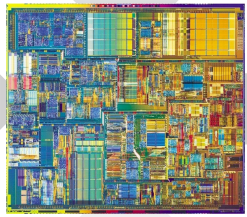
WebGL glTF

NNEF

OpenVX

Vision and Neural Networks
- Tracking and odometry
- Scene analysis/understanding
- Neural Network inferencing

Parallel Computation
- Machine Learning acceleration
- Embedded vision processing
- High Performance Computing (HPC)



OpenCL

SPIR SYCL

OpenGL|SC OpenVG

Vulkan OpenGL|ES

OpenGL

'VR Initiative'

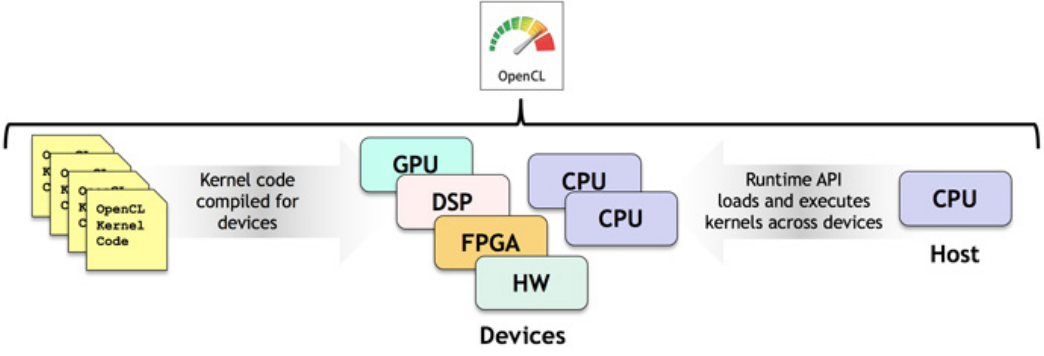
Real-time 2D/3D
- Virtual and Augmented Reality
- Cross-platform gaming and UI
- CG Visual Effects
- CAD and Product Design
- Safety-critical displays

KHRONOS GROUP

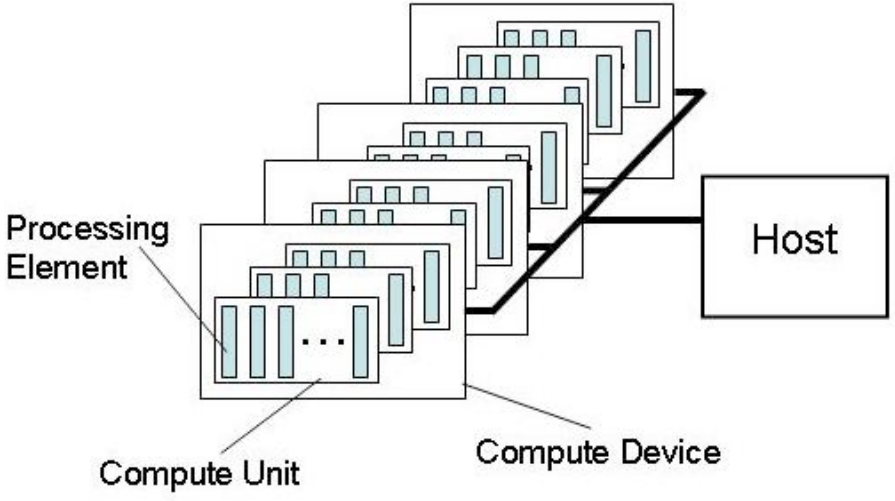
Outline

OpenCL

- OpenCL: 2 APIs and 2 kernel languages
  - C Platform Layer API to query, select and initialize compute devices
  - OpenCL C 2.0 and OpenCL C++ 2.2 kernel languages to write separate parallel code
  - C Runtime API to build and execute kernels across multiple devices
- One code tree can be executed on CPU, GPU, DSP, FPGA and hardware
  Dynamically balance work across available processors

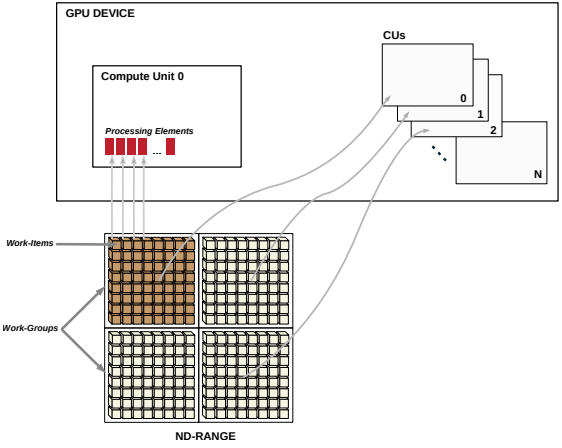


Architecture model

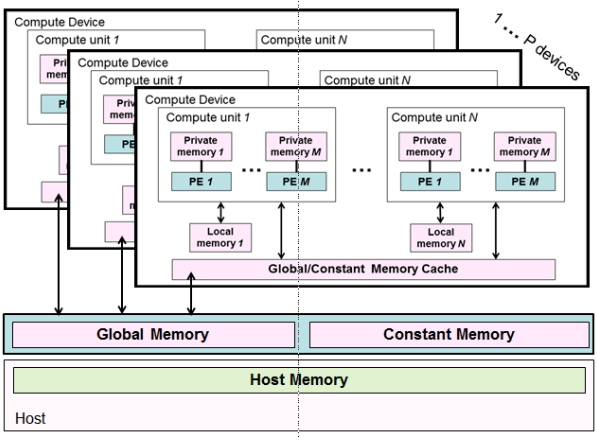


- Host threads launch computational *kernels* on *accelerators*

Execution model



Memory model



Vector add (≈ Hello World) in C++ using OpenCL C host API *(I)*

$$\vec{c} = \vec{a} + \vec{b}$$

```
1  #include <iostream>
2  #include <string>
3  #include <vector>
4
5  #if defined(__APPLE__)
6  #include <OpenCL/cl.h>
7  #else
8  #include <CL/cl.h>
9  #endif
10
11 /* Transform the value of a given symbol to a string. Since we expect a
12    macro symbol, use a double evaluation... */
13 #define _stringG(s) #s
14
15 #define _stringifY(s) _stringG(s)
16
17 /** Throw a nicer error message in the code by adding the file name and
18     the position */
19 #define THROW_ERROR(message)                                          \
20   throw std::domain_error(std::string("In file " __FILE__ " at line "   \
21                                       _stringifY(__LINE__) "\n") + message)
```

Vector add (≈ Hello World) in C++ using OpenCL C host API (II)

```
22
23  /** Test for an OpenCL error and display a message */
24  #define OCL_TEST_ERROR_MSG(status, msg) do {                              \
25      if ((status) !=  CL_SUCCESS)                                          \
26        THROW_ERROR(std::string(msg) + std::to_string(status));             \
27    } while(0)
28
29    /** Do an OpenCL function call and test for execution error */
30  #define OCL_ERROR(func) do {                                              \
31      cl_int _st = func;                                                    \
32      if (_st !=  CL_SUCCESS)                                               \
33        THROW_ERROR(_stringifY(func) " returns error " + std::to_string(_st)); \
34    } while(0)
35
36  constexpr size_t N = 3;
37
38  using Vector = float[N];
39
40  int main() {
41    Vector a = { 1, 2, 3 };
42    Vector b = { 5, 6, 8 };
43    Vector c;
44
45    cl_int status;
46
47    // Get the number of OpenCL platforms on the machine
```

Vector add (≈ Hello World) in C++ using OpenCL C host API *(III)*

```
48    cl_uint num_platforms;
49    OCL_ERROR(clGetPlatformIDs(0, NULL, &num_platforms));
50
51    std::vector<cl_platform_id> platforms(num_platforms);
52    OCL_ERROR(clGetPlatformIDs(num_platforms, platforms.data(), NULL));
53
54
55    cl_context context;
56    bool found_context = false;
57    for (auto platform : platforms) {
58      // Describe the context to query
59      cl_context_properties cps[] = {
60        CL_CONTEXT_PLATFORM, (cl_context_properties)platform,
61        0
62      };
63      // Create an OpenCL context from our platform
64      context = clCreateContextFromType(cps,
65                                        CL_DEVICE_TYPE_ALL,
66                                        NULL,
67                                        NULL,
68                                        &status);
69      if (status ==  CL_SUCCESS) {
70        found_context = true;
71        break;
72      }
73    }
```

Vector add (≈ Hello World) in C++ using OpenCL C host API *(IV)*

```
74    if (!found_context)
75      THROW_ERROR("Cannot found a context");
76
77    // Get the first device
78    cl_device_id device;
79    OCL_ERROR(clGetContextInfo(context, CL_CONTEXT_DEVICES,
80                               sizeof(device), &device, NULL));
81
82    // Create an OpenCL command queue
83    cl_command_queue command_queue =
84      clCreateCommandQueueWithProperties(context, device, NULL, &status);
85    OCL_TEST_ERROR_MSG(status, "Cannot create the command queue");
86
87    // The input buffers for OpenCL
88     cl_mem buffer_a =
89       clCreateBuffer(context, CL_MEM_READ_ONLY, sizeof(a), NULL, &status);
90     OCL_TEST_ERROR_MSG(status, "Cannot create buffer_a");
91     cl_mem buffer_b =
92       clCreateBuffer(context, CL_MEM_READ_ONLY, sizeof(b), NULL, &status);
93     OCL_TEST_ERROR_MSG(status, "Cannot create buffer_b");
94
95    // The output buffer for OpenCL
96     cl_mem buffer_c =
97       clCreateBuffer(context, CL_MEM_WRITE_ONLY, sizeof(c), NULL, &status);
98     OCL_TEST_ERROR_MSG(status, "Cannot create buffer_c");
99
```

Vector add (≈ Hello World) in C++ using OpenCL C host API *(V)*

```
100     // Construct an OpenCL program from the source file
101     const char kernel_source[] = R"(
102   __kernel void vector_add(const __global float *a,
103                            const __global float *b,
104                            __global float *c) {
105     c[get_global_id(0)] = a[get_global_id(0)] + b[get_global_id(0)];
106   }
107   )";
108     const char *kernel_sources = kernel_source;
109     const size_t kernel_size = sizeof(kernel_source);
110     cl_program program = clCreateProgramWithSource(context, 1, &kernel_sources,
111                                                    &kernel_size, &status);
112     OCL_TEST_ERROR_MSG(status, "Cannot create program");
113
114     OCL_ERROR(clBuildProgram(program, 1, &device, "", NULL, NULL));
115
116     cl_kernel kernel = clCreateKernel(program, "vector_add", &status);
117     OCL_TEST_ERROR_MSG(status, "Cannot find the kernel");
118
119     // Send the input data to the accelerator
120     OCL_ERROR(clEnqueueWriteBuffer(command_queue, buffer_a, true, 0 /* Offset */,
121                                    sizeof(a), &a[0], 0, NULL, NULL));
122     OCL_ERROR(clEnqueueWriteBuffer(command_queue, buffer_b, true, 0 /* Offset */,
123                                    sizeof(b), &b[0], 0, NULL, NULL));
124
125     OCL_ERROR(clSetKernelArg(kernel, 0, sizeof(buffer_a), &buffer_a));
```

Vector add (≈ Hello World) in C++ using OpenCL C host API *(VI)*

```
126     OCL_ERROR(clSetKernelArg(kernel, 1, sizeof(buffer_b), &buffer_b));
127     OCL_ERROR(clSetKernelArg(kernel, 2, sizeof(buffer_c), &buffer_c));
128
129     // Launch the kernel
130     const size_t global_work_size { N };
131     OCL_ERROR(clEnqueueNDRangeKernel(command_queue, kernel, 1, NULL,
132                                      &global_work_size, NULL,
133                                      0, NULL, NULL));
134
135     // Get the output data from the accelerator
136     OCL_ERROR(clEnqueueReadBuffer(command_queue, buffer_c, true, 0 /* Offset */,
137                                   sizeof(c), &c[0], 0, NULL, NULL));
138
139
140     std::cout << std::endl << "Result:" << std::endl;
141     for(auto e : c)
142       std::cout << e << " ";
143     std::cout << std::endl;
144   }
```

C++ wrapper for OpenCL API from Khronos: CL/cl2.hpp *(I)*

https://github.com/KhronosGroup/OpenCL-CLHPP

```
1  #include <iostream>
2  #include <iterator>
3  #define CL_HPP_ENABLE_EXCEPTIONS
4  #include <CL/cl2.hpp>
5
6  constexpr size_t N = 3;
7  using Vector = float[N];
8
9  int main() {
10   Vector a = { 1, 2, 3 };
11   Vector b = { 5, 6, 8 };
12   Vector c;
13
14   // The input read-only buffers for OpenCL on default context
15   cl::Buffer buffer_a { std::begin(a), std::end(a), true};
16   cl::Buffer buffer_b { std::begin(b), std::end(b), true};
17   // The output buffer for OpenCL on default context
18   cl::Buffer buffer_c { CL_MEM_WRITE_ONLY, sizeof(c) };
19
20   // Construct an OpenCL program from the source file
21   const char kernel_source[] = R"(
```

C++ wrapper for OpenCL API from Khronos: CL/cl2.hpp (II)

```
22  __kernel void vector_add(const __global float *a,
23                           const __global float *b,
24                           __global float *c) {
25    c[get_global_id(0)] = a[get_global_id(0)] + b[get_global_id(0)];
26  }
27  )";
28    // Compile and build the program
29    cl::Program p { kernel_source, true };
30    // Create the kernel functor taking 3 buffers as parameter
31    cl::KernelFunctor<cl::Buffer, cl::Buffer, cl::Buffer> k { p, "vector_add" };
32
33    // Call the kernel with N work-items on default command queue
34    k(cl::EnqueueArgs(cl::NDRange(N)), buffer_a, buffer_b, buffer_c);
35
36    // Get the output data from the accelerator
37    cl::copy(buffer_c, std::begin(c), std::end(c));
38
39    std::cout << std::endl << "Result:" << std::endl;
40    for(auto e : c)
41      std::cout << e << " ";
42    std::cout << std::endl;
43  }
```

Outline

Interoperability nightmare in heterogeneous computing & graphics

- ∃ Many programming languages for heterogeneous computing
  - Writing compiler front-end may not be *the* real value for a hardware vendor...
    - Writing a C++17 compiler from scratch is almost impossible...
- ∃ Many programming languages for writing shaders
- Convergence in computing (Compute Unit) & graphics (Shader) architectures
  - Same front-end & middle-end compiler optimizations
- Need for some non source-readable portable code for IP protection

⇝ Defining common low-level representation !

SPIR-V transforms the language ecosystem

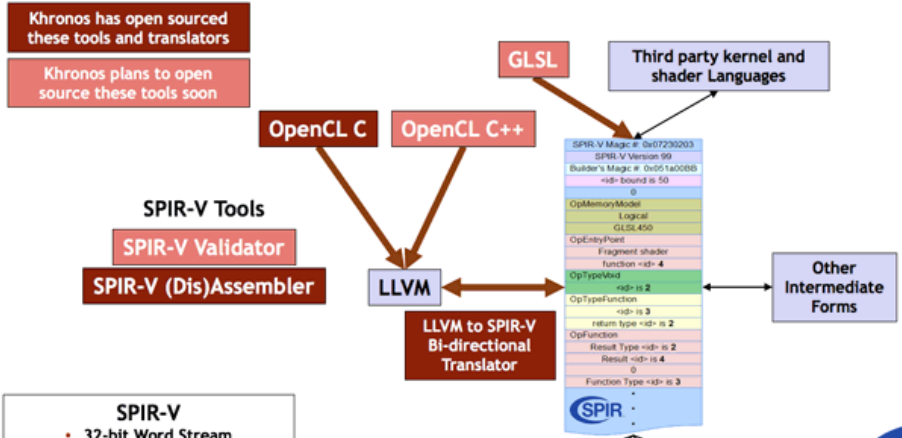
- First multi-API, intermediate language for parallel compute *and* graphics
  - Native representation for Vulkan shader and OpenCL kernel source languages
  - `https://www.khronos.org/registry/spir-v/papers/WhitePaper.pdf`
- Cross-vendor intermediate representation
  - Language front-ends can easily access multiple hardware run-times
  - Acceleration hardware can leverage multiple language front-ends
  - Encourages tools for program analysis and optimization in SPIR form

Evolution of SPIR family

SPIR	SPIR 1.2	SPIR 2.0	SPIR-V 1.0
LLVM Interaction	Uses LLVM 3.2	Uses LLVM 3.4	100% Khronos defined Round-trip lossless conversion
Compute Constructs	Metadata/Intrinsics	Metadata/Intrinsics	Native
Graphics Constructs	No	No	Native
Supported Language Feature Sets	OpenCL C 1.2	OpenCL C 1.2 OpenCL C 2.0	OpenCL C 1.2 / 2.0 OpenCL C++ and GLSL
OpenCL Ingestion	OpenCL 1.2 Extension	OpenCL 2.0 Extension	OpenCL 2.1 Core OpenCL 1.2 / 2.0 Extensions
Vulkan Ingestion	-	-	Vulkan 1.0 Core

Not based on LLVM to isolate from LLVM roadmap changes

Driving SPIR-V Open Source ecosystem



Outline

Position argument

Which language for unified heterogeneous computing?

-  Entry cost
- ∃ thousands of dead parallel languages...
  -  Exit cost
- Use standard solutions with open source implementations
- Start with modern C++
  - Very successful & ubiquitous language
  - Very active since C++11 (C++14, C++17...)
  - Interoperability: seamless interaction with embedded world, libraries, OS...
  - Combine both low-level aspects with high-level programming
    - Pay only for what you need (garbage collector or not...)
  - Classes can be used to define Domain Specific Embedded Language (DSEL)
  - Not directly targeting FPGA...
    - But extensible through classes ( DSEL)
    - Extensible with #pragma (already in Xilinx tools Vivado HLS, SDSoC & SDAccel) and attributes

C++

2-line description by Bjarne Stroustrup

- Direct mapping to hardware
- Zero-overhead abstraction

Even better with modern C++ (C++14, C++17) *(I)*

- Huge library improvements, parallelism...
- Simpler syntax, type inference in constructors

```
std::vector my_vector { 1, 2, 3, 4, 5 };
for (auto &e : my_vector)
  e += 1;
```

Even better with modern C++ (C++14, C++17) *(II)*

- Automatic type inference for terse generic programming
  - Python 3.x (interpreted):

```
def add(x, y):
  return x + y
print(add(2, 3))      # 5
print(add("2", "3")) # 23
print(add(2, "Boom")) # Fails at run-time :-(
```

  - Same in C++14 but compiled + static compile-time type-checking:

```
auto add = [] (auto x, auto y) { return x + y; };
std::cout << add(2, 3) << std::endl;        // 5
std::cout << add("2"s, "3"s) << std::endl; // 23
std::cout << add(2, "Boom"s) << std::endl; // Does not compile :-)
```

Without using templated code! ~~`template <typename >`~~ ☺

Even better with modern C++ (C++14, C++17) *(III)*

- Generic variadic lambdas & operator interpolation

```
#include <iostream>

// Define an adder on anything.
// Use new C++14 generic variadic lambda syntax
auto add = [] (auto... args) {
  // Use new C++17 operator folding syntax
  return (... + args);
};

int main() {
 std::cout << "The result is: " << add(1, 2, 3) << std::endl;
}
```

Without using templated code! ~~`template <typename >`~~ ☺
Try this example on `https://wandbox.org/permlink/wpW3PUOCxo5d`

1 C++ version/3 years ⇝ Parallelizing C++ committee itself

- SG1, Concurrency: Hans Boehm (Google). Concurrency and parallelism
- SG2, Modules. Work on possible refinement or replacement for the header-based build model
- SG3, File System: Beman Dawes (Boost). Boost.Filesystem v3
- SG4, Networking. Networking related libraries, including sockets and HTTP
- SG5, Transactional Memory: Michael Wong (IBM). Exploring transactional memory constructs for potential future addition to the C++ language
- SG6, Numerics: Lawrence Crowl. Numerics topics, including but not limited to fixed point, decimal floating point, and fractions
- SG7, Reflection: Chandler Carruth (Google). Initially focusing on compile-time reflection capabilities
- SG8, Concepts. Near-term focus is on a convergence between the static if proposals and the parameter-type-constraints subset of concepts
- SG9, Ranges: Marshall Clow (Qualcomm). How to update the standard library with a range concept rather than naked iterator pairs, including containers and range-based algorithms
- SG10, Feature Test: Clark Nelson (Intel). Investigation into whether and how to standardize a way for portable code to check whether a particular C++ product implements a feature yet, as we continue to extend the standard
- SG11, Databases. Database-related library interfaces
- SG12, Undefined and Unspecified Behavior: Gabriel Dos Reis (Microsoft). A systematic review to catalog cases of undefined and unspecified behavior in the standard and recommend a coherent set of changes to define and/or specify the behavior
- SG13, HMI (Human/Machine Interface). Selected low-level graphic/pointing I/O primitives
- SG14, Game Development & Low Latency: Michael Wong (IBM). Topics of interest to game developers and (other) low-latency programming requirements

SG14

C++ sub-group working on “Games, Low Latency, Real-time applications, Graphics, Financial Trading” `https://groups.google.com/a/isocpp.org/forum/#!forum/sg14`

- Very intense activity & lot of attraction right now
- Example of proposals
  - SIMD & DSP operations
  - GPU/Accelerator design
  - P0037R0 Fixed point real numbers John McFarlane LEWG SG14/SG6: Lawrence Crowl
  - Low level bit manipulation
  - Lower precision floating point
  - P0059R0 Add rings to the Standard Library Guy Davidson LEWG SG14: Michael
  - Array View and Bounds checking
  - P0038R0 Flat Containers Sean Middleditch LEWG SG14: Patrice Roy
  - P0039R0 Extending `raw_storage_iterator` Brent Friedman LEWG SG14: Billy Baker
  - P0040R0 Extending memory management tools Brent Friedman LEWG SG14: Billy Baker
- Fusion of Embedded C++ working group into SG14 on 2016/02/16

How to express details?

- Generalized attributes [[foo::bar]]
  - Allow terse incremental programming
  - Might not change semantics
  - Might be ignored by the compiler
- #pragma
  - Allow terse incremental programming
  - Simple to do simple things
  - Might not compose nicely with generic programming
- Use wrapper objects
  - Normal classes ⇝ integrated in type system
  - Poor (wo)man introspection with type traits
    - Specialization possible according to wrapper kind
  - Allow RAII
  - Plain C++: do not require a compiler to prototype
  - Require deeper program change and generic programming
- Language extensions
- ...

Example of C++ extensions: OpenMP 4.5

- Single source #pragma extensions
- Support C, C++, Fortran with same #pragma
  - Other unofficial standards (Pythran for Python-to-C++ translation...)
- Task-graph oriented programming
- Multithread SMP support
- Loop parallelization
- Relaxed memory consistency model
- SIMD vectorization
- Accelerator offloading with hierarchical parallelism
- Transactional memory (optimistic speculative execution)
- Runtime API (environment queries, locks)

OpenMP 4.5 example: tasks

```
int fib(int n) {
  int i, j;
  if (n < 2)
    return n;
  else {
    // Execute in another task, sharing i variable
#pragma omp task shared(i)
    i = fib(n - 1);
    // Execute in another task, sharing j variable
#pragma omp task shared(j)
    j = fib(n - 2);
    // Wait for execution of children tasks
#pragma omp taskwait
    return i + j;
  }
}
```

OpenMP 4.5 example: SIMD

```
void sprod(float a[], float b[], int n) {
  float sum = 0;
  // Execute loop in a SIMD way, with reduction on 'sum' with '+' operation
#pragma omp for simd reduction(+:sum)
  for (int k=0; k<n; k++)
    sum += a[k] * b[k];
  return sum;
}

void multithread_sprod(float a[], float b[], int n) {
  float sum = 0;
  // The same with threads too
#pragma omp parallel for simd reduction(+:sum)
  for (int k=0; k<n; k++)
    sum += a[k] * b[k];
  return sum;
}
```

OpenMP 4.5 example: C++ & heterogeneous computing

```
std::vector<float> x(n), y(n);
// Map/copy somehow the data part of x to the accelerator
#pragma omp target data map(to: x.data()[0:n])
{ // Map/copy y.data to accelerator and back later
#pragma omp target map(tofrom: y.data()[0:n])
// Create num_blocks teams of block_size threads
  int num_blocks = (n + block_size - 1)/block_size;
#pragma omp teams num_teams(num_blocks) thread_limit(block_size)
  // Distribute on the master threads of the teams
#pragma omp distribute
  for (int i = 0; i < n; i += block_size) {
    // Distribute iterations on the threads of a team
#pragma omp parallel for
    for (int j = i; j < i + num_blocks; j++) {
      y.data()[j] = a*x.data()[j] + y.data()[j];
    } //< Implicit barrier here
  }
} //< Get back y.data to the host
```

CUDA Runtime API

- Proprietary language for nVidia GPU

```
#include <iostream>
#include <stdexcept>

constexpr size_t N = 3;
using Vector = float[N];

__global__ void vector_add(const float *a,
                           const float *b,
                           float *c) {
  int idx = blockIdx.x*blockDim.x+threadIdx.x;
  if(idx < N)
    c[idx] = a[idx] + b[idx];
}

void checkError(cudaError_t err) {
  if(err != cudaSuccess) {
    throw std::domain_error("CUDA ERROR: "
     + std::string { cudaGetErrorString(err) });
  }
}

int main() {
  Vector a = {1, 2, 3};
  Vector b = {5, 6, 8};
  Vector c;

  float *d_a, *d_b, *d_c;

  checkError(cudaMalloc((void**) &d_a, N*sizeof(*d_a)));
  checkError(cudaMalloc((void**) &d_b, N*sizeof(*d_b)));
  checkError(cudaMalloc((void**) &d_c, N*sizeof(*d_c)));

  checkError(cudaMemcpy(d_a, a, N*sizeof(*a), cudaMemcpyHostToDevice));
  checkError(cudaMemcpy(d_b, b, N*sizeof(*b), cudaMemcpyHostToDevice));

  vector_add<<<1, N>>>(d_a, d_b, d_c);

  checkError(cudaMemcpy(c, d_c, N*sizeof(*c), cudaMemcpyDeviceToHost));

  std::cout << std::endl << "Result: " << std::endl;
  for(auto e: c)
    std::cout << e << " ";
  std::cout << std::endl;

  checkError(cudaFree(d_a));
  checkError(cudaFree(d_b));
  checkError(cudaFree(d_c));

  return 0;
}
```

- C++14 single-source + extensions & restrictions
- Not very modern C++ style nor portable

Boost.Compute (I)

- Boost library accepted in 2015 `https://github.com/boostorg/compute`
- Provide 2 levels of abstraction
  - High-level parallel STL
  - Low-level C++ wrapping of OpenCL concepts

Boost.Compute (II)

```
// Get a default command queue on the default accelerator
auto queue = boost::compute::system::default_queue();
// Allocate a vector in a buffer on the device
boost::compute::vector<float> device_vector { N, queue.get_context() };
boost::compute::iota(device_vector.begin(), device_vector.end(), 0);

// Create an equivalent OpenCL kernel
BOOST_COMPUTE_FUNCTION(float, add_four, (float x), { return x + 4; });

boost::compute::transform(device_vector.begin(), device_vector.end(),
                          device_vector.begin(), add_four, queue);

boost::compute::sort(device_vector.begin(), device_vector.end(), queue);
// Lambda expression equivalent
boost::compute::transform(device_vector.begin(), device_vector.end(),
                          device_vector.begin(),
                          boost::compute::lambda::_1 * 3 - 4, queue);
```

Boost.Compute (III)

- Elegant C++ conversions between OpenCL and Boost.Compute types for finer control and optimizations

```
1  #include <boost/compute.hpp>
2  #include <iostream>
3  #include <iterator>
4
5  constexpr size_t N = 3;
6  using Vector = float[N];
7
8  int main() {
9    Vector a = { 1, 2, 3 };
10   Vector b = { 5, 6, 8 };
11   Vector c;
12
13   // Create the OpenCL context to attach resources on the device
14   auto context = boost::compute::system::default_context();
15   // Create the OpenCL command queue to control the device
16   auto command_queue = boost::compute::system::default_queue();
17
18   // The input buffers for OpenCL
19   boost::compute::buffer buffer_a { context, sizeof(a), CL_MEM_READ_ONLY };
20   boost::compute::buffer buffer_b { context, sizeof(b), CL_MEM_READ_ONLY };
21
22   // The output buffer for OpenCL
```

Boost.Compute (IV)

```
23    boost::compute::buffer buffer_c { context, sizeof(c), CL_MEM_WRITE_ONLY };
24
25    // Construct an OpenCL program from the source file
26    auto program =
27      boost::compute::program::create_with_source_file("vector_add.cl", context);
28    program.build();
29
30    auto kernel = boost::compute::kernel { program, "vector_add" };
31
32    // Send the input data to the accelerator
33    command_queue.enqueue_write_buffer(buffer_a, 0 /* Offset */,
34                                       sizeof(a), &a[0]);
35    command_queue.enqueue_write_buffer(buffer_b, 0 /* Offset */,
36                                       sizeof(b), &b[0]);
37
38    kernel.set_args(buffer_a, buffer_b, buffer_c);
39
40    boost::compute::extents<1> offset { 0 };
41    boost::compute::extents<1> global { N };
42    // Use only 1 CU
43    boost::compute::extents<1> local { N };
44    // Launch the kernel
45    command_queue.enqueue_nd_range_kernel(kernel, offset, global, local);
46
47    // Get the output data from the accelerator
48    command_queue.enqueue_read_buffer(buffer_c, 0 /* Offset */,
```

Boost.Compute (V)

```
49                                          sizeof(c), &c[0]);
50
51    std::cout << std::endl << "Result:" << std::endl;
52    for(auto e : c)
53      std::cout << e << " ";
54    std::cout << std::endl;
55  }
```

- Provide program caching
- Direct OpenCL interoperability for extreme performance
- No specific compiler required ⇝ some special syntax to define operation on device
- Probably the right tool to use to move CUDA & Thrust applications to OpenCL world

Other (non-)OpenCL C++ framework

- VexCL, ViennaCL, ArrayFire, Aura, CLOGS, hemi, HPL, Kokkos, MTL4, SkelCL, SkePU, EasyCL, Bolt C++, C++AMP, HCC...
- nVidia CUDA 8 now C++14-based
  - Single source ⇝ simpler for the programmer
  - nVidia Thrust ≈ parallel STL+map-reduce on top of CUDA, OpenMP or TBB
    `https://github.com/thrust/thrust`
    - Not very clean because device pointers returned by `cudaMalloc()` do not have a special type ⇝ use some ugly casts
- OpenACC ≈ OpenMP 4 restricted to accelerators + finer control on CU local storage

Outline

Missing link...

- No tool providing
  - Modern C++ environment
  - Heterogeneous computing (FPGA...) *à la* SG14
  - Single source for programming productivity
    - Templated code across kernels
    - Type safety across heterogeneous execution
  - OpenCL interoperability to recycle other libraries and portability
  - Backed by open standard

What about C++ for heterogenous computing???

- C++ std::thread is great...
- ...but assume shared unified memory (SMP) ☹
  - What if accelerator with own separate memory? Not same address space?
  - What if using distributed memory multi-processor system (MPI...)?
- ⇝ Extend the concepts...
  - Replace raw unified-memory with buffer objects
  - Define with accessor objects which/how buffers are used
  - Since accessors are already here to define dependencies, no longer need for std::future/std::promise! ☺
  - Add concept of queue to express where to run the task
  - Also add all goodies for massively parallel accelerators (OpenCL/Vulkan/SPIR-V) in clean C++

SYCL 2.2 ≡ pure C++17 DSEL

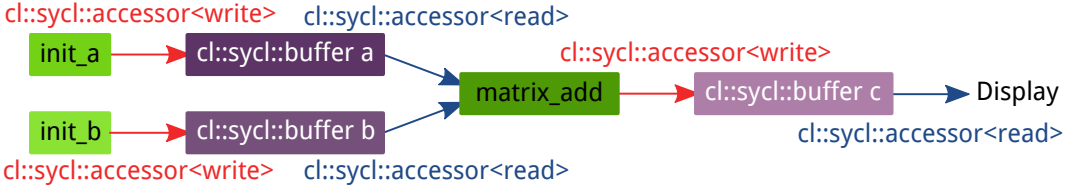
- Implement concepts useful for heterogeneous computing
- Asynchronous task graph
- Buffers to define location-independent storage (no explicit move)
- Accessors to express usage for buffers and pipes: read/write/...
- Hierarchical parallelism & kernel-side enqueue
- Hierarchical storage
  - Rely on C++ allocator to specify storage (SVM...)
- Single source programming model
  - Take advantage of CUDA on steroids & OpenMP simplicity and power
  - Compiled for host *and* device(s)
  - Enabling the creation of C++ higher level programming models & C++ templated libraries
- Most modern C++ features available for OpenCL
  - Programming interface based on abstraction of OpenCL components (data management, error handling...)
  - Provide OpenCL interoperability
- Host fallback (debug and symmetry for SIMD/multithread on host)
- Directly executable DSEL
- Host emulation for free & no compiler needed for experimenting

Complete example of matrix addition in OpenCL SYCL

```
#include <CL/sycl.hpp>
#include <iostream>

using namespace cl::sycl;

constexpr size_t N = 2;
constexpr size_t M = 3;
using Matrix = float[N][M];

// Compute sum of matrices a and b into c
int main() {
 Matrix a = { { 1, 2, 3 }, { 4, 5, 6 } };
 Matrix b = { { 2, 3, 4 }, { 5, 6, 7 } };

 Matrix c;

 {// Create a queue to work on default device
  queue q;
  // Wrap some buffers around our data
  buffer A { &a[0][0], range { N, M } };
  buffer B { &b[0][0], range { N, M } };
  buffer C { &c[0][0], range { N, M } };
  // Enqueue some computation kernel task
  q.submit([&](handler& cgh) {
   // Define the data used/produced
   auto ka = A.get_access<access::mode::read>(cgh);
   auto kb = B.get_access<access::mode::read>(cgh);
   auto kc = C.get_access<access::mode::write>(cgh);
   // Create & call kernel named "mat_add"
   cgh.parallel_for<class mat_add>(range { N, M },
      [=](id<2> i) { kc[i] = ka[i] + kb[i]; }
   );
  }); // End of our commands for this queue
 } // End scope, so wait for the buffers to be releas
 // Copy back the buffer data with RAII behaviour.
 std::cout << "c[0][2] = " << c[0][2] << std::endl;
 return 0;
}
```

Asynchronous task graph model

- Change example with initialization kernels instead of host?...
- Theoretical graph of an application described *implicitly* with kernel tasks using buffers through accessors



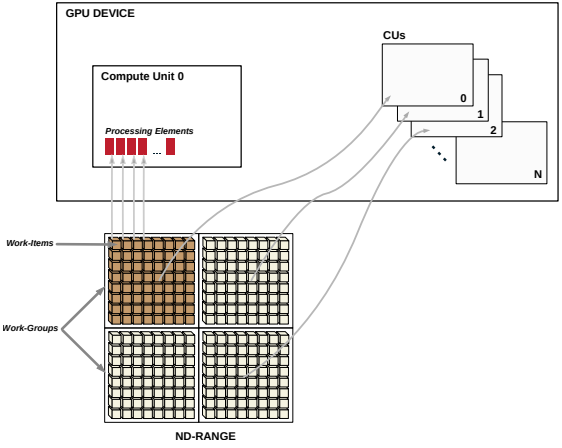
- Possible schedule by SYCL runtime:

⇝ Automatic overlap of kernels & communications
  - Even better when looping around in an application
  - Assume it will be translated into pure OpenCL event graph
  - Runtime uses as many threads & OpenCL queues as necessary (GPU synchronous queues, AMD compute rings, AMD DMA rings...)

Task graph programming — the code

```
#include <CL/sycl.hpp>
#include <iostream>
using namespace cl::sycl;
// Size of the matrices
constexpr size_t N = 2000;
constexpr size_t M = 3000;
int main() {
  { // By sticking all the SYCL work in a {} block, we ensure
    // all SYCL tasks must complete before exiting the block

    // Create a queue to work on default device
    queue q;
    // Create some 2D buffers of float for our matrices
    buffer<double, 2> a({ N, M });
    buffer<double, 2> b({ N, M });
    buffer<double, 2> c({ N, M });
    // Launch a first asynchronous kernel to initialize a
    q.submit([&](auto &cgh) {
        // The kernel write a, so get a write accessor on it
        auto A = a.get_access<access::mode::write>(cgh);

        // Enqueue parallel kernel on a N*M 2D iteration space
        cgh.parallel_for<class init_a>({ N, M },
                                       [=] (auto index) {
                                         A[index] = index[0]*2 + index[1];
                                       });
      });
    // Launch an asynchronous kernel to initialize b
    q.submit([&](auto &cgh) {
        // The kernel write b, so get a write accessor on it
        auto B = b.get_access<access::mode::write>(cgh);
        /* From the access pattern above, the SYCL runtime detect
           this command_group is independant from the first one
           and can be scheduled independently */

        // Enqueue a parallel kernel on a N*M 2D iteration space
        cgh.parallel_for<class init_b>({ N, M },
                                       [=] (auto index) {
                                         B[index] = index[0]*2014 + index[1]*42;
                                       });
      });
    // Launch an asynchronous kernel to compute matrix addition c = a + b
    q.submit([&](auto &cgh) {
        // In the kernel a and b are read, but c is written
        auto A = a.get_access<access::mode::read>(cgh);
        auto B = b.get_access<access::mode::read>(cgh);
        auto C = c.get_access<access::mode::write>(cgh);
        // From these accessors, the SYCL runtime will ensure that when
        // this kernel is run, the kernels computing a and b completed

        // Enqueue a parallel kernel on a N*M 2D iteration space
        cgh.parallel_for<class matrix_add>({ N, M },
                                           [=] (auto index) {
                                             C[index] = A[index] + B[index];
                                           });
      });
    /* Request an access to read c from the host-side. The SYCL runtime
       ensures that c is ready when the accessor is returned */
    auto C = c.get_access<access::mode::read, access::target::host_buffer>
    std::cout << std::endl << "Result:" << std::endl;
    for(size_t i = 0; i < N; i++)
      for(size_t j = 0; j < M; j++)
        // Compare the result to the analytic value
        if (C[i][j] != i*(2 + 2014) + j*(1 + 42)) {
          std::cout << "Wrong value " << C[i][j] << " on element "
                    << i << ' ' << j << std::endl;
          exit(-1);
        }
  }
  std::cout << "Good computation!" << std::endl;
  return 0;
}
```

Remember the OpenCL execution model?



From work-groups & work-items to hierarchical parallelism (I)

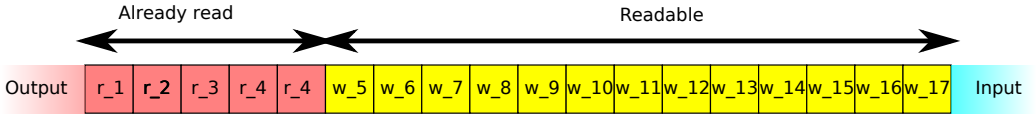
```
// Launch a 1D convolution filter
my_queue.submit([&](handler &cgh) {
  auto in_access = inputB.get_access<access::mode::read>(cgh);
  auto filter_access = filterB.get_access<access::mode::read>(cgh);
  auto out_access = outputB.get_access<access::mode::write>(cgh);
  // Iterate on all the work-group
  cgh.parallel_for_work_group<class convolution>({ size,
                                                   groupsize },
    [=](group<> group) {
      std::cout << "Group id = " << group.get(0) << std::endl;
      // These are OpenCL local variables used as a cache
      float filterLocal[2*radius + 1];
      float localData[blocksize + 2*radius];
      float convolutionResult[blocksize];
      range<1> filterRange { 2*radius + 1 };
      // Iterate on filterRange work-items
      group.parallel_for_work_item(filterRange, [&](item<1> tile) {
        filterLocal[tile] = filter_access[tile];
      });
      // There is an implicit barrier here
      range<1> inputRange{ blocksize + 2*radius };
      // Iterate on inputRange work-items
      group.parallel_for_work_item(inputRange, [&](item<1> tile) {
        float val = 0.f;
        int readAddress = group*blocksize + tile - radius;
        if (readAddress >= 0 && readAddress < size)
          val = in_access[readAddress];

        localData[tile] = val;
      });
      // There is an implicit barrier here
      // Iterate on all the work-items
      group.parallel_for_work_item([&](item<1> tile) {
        float sum = 0.f;
        for (unsigned offset = 0; offset < radius; ++offset)
          sum += filterLocal[offset]*localData[tile + offset + radius];
        float result = sum/(2*radius + 1);
        convolutionResult[tile] = result;
      });
      // There is an implicit barrier here
      // Iterate on all the work-items
      group.parallel_for_work_item(group, [&](item<1> tile)
        out_access[group*blocksize + tile] = convolutionResult[tile];
      });
      // There is an implicit barrier here
    });
});
```

From work-groups & work-items to hierarchical parallelism *(II)*

Very close to OpenMP 4 style! ☺

- Easy to understand the concept of work-groups
- Easy to write work-group only code
- Replace code + barriers with several parallel_for_work_item()
  - Performance-portable between CPU and device
  - No need to think about barriers (automatically deduced)
  - Easier to compose components & algorithms
  - Ready for future device with non uniform work-group size

Pipes in OpenCL 2.*x*



- Simple FIFO objects
- Useful to create dataflow architectures between kernels without host
- Created on the host with some message size + object number
- `read()`/`write()` functions
- Useful on FPGA: avoid global memory transfers!

Producer/consumer with blocking pipe in SYCL 2.*x*

```
#include <CL/sycl.hpp>
#include <iostream>
#include <iterator>

using namespace cl::sycl;
constexpr size_t N = 3;
using Vector = float[N];

int main() {
  Vector va = { 1, 2, 3 };
  Vector vb = { 5, 6, 8 };
  Vector vc;

  {
    // Create buffers from a & b vectors
    buffer<float> ba { std::begin(va), std::end(va) };
    buffer<float> bb { std::begin(vb), std::end(vb) };

    // A buffer of N float using the storage of vc
    buffer<float> bc { vc, N };

    // A pipe of 2 float elements
    pipe<float> p { 2 };

    // Create a queue to launch the kernels
    queue q;

    // Launch the producer to stream A to the pipe
    q.submit([&](handler &cgh) {
      // Get write access to the pipe
      auto kp = p.get_access<access::mode::write,
                             access::target::blocking_pipe>(cgh);

      // Get read access to the data
      auto ka = ba.get_access<access::mode::read>(cgh);

      cgh.single_task<class producer>([=] {
          for (int i = 0; i != N; i++)
            kp << ka[i];
        });
      });

    // Launch the consumer that adds the pipe stream with B to C
    q.submit([&](handler &cgh) {
      // Get read access to the pipe
      auto kp = p.get_access<access::mode::read,
                             access::target::blocking_pipe>(cgh);

      // Get access to the input/output buffers
      auto kb = bb.get_access<access::mode::read>(cgh);
      auto kc = bc.get_access<access::mode::write>(cgh);

      cgh.single_task<class consumer>([=] {
          for (int i = 0; i != N; i++)
            kc[i] = kp.read() + kb[i];
        });
      });
  }  ///< End scope for the buffers: wait for bc copied back to v

  std::cout << std::endl << "Result:" << std::endl;
  for(auto e : vc)
    std::cout << e << " ";
  std::cout << std::endl;
}
```

Motion detection on video in SYCL with blocking static pipes

```
auto window_name = "opencv test";
cv::namedWindow(window_name, cv::WINDOW_AUTOSIZE);

cv::Mat rgb_data_in { NUMROWS, NUMCOLS, CV_8UC4 };
cv::Mat rgb_data_prev { NUMROWS, NUMCOLS, CV_8UC4 };
cv::Mat rgb_data_out { NUMROWS, NUMCOLS, CV_8UC4 };

cv::VideoCapture capture;
capture.open("./optical_flow_input.avi");

cv::Mat frame;
capture.read(frame);

cv::Mat small_frame;
cv::resize(frame, small_frame, rgb_data_in.size());

const int from_to[] = { 0, 0, 1, 1, 2, 2 };
cv::mixChannels(&small_frame, 1, &rgb_data_in, 1, from_to, 3);

cv::imshow(window_name, rgb_data_in);
cv::waitKey(30);

// Create a queue to launch the kernels
cl::sycl::queue q;

int framecnt = 0;
// Processing loop
while (capture.read(frame)) {
  cv::swap(rgb_data_in, rgb_data_prev);
  cv::resize(frame, small_frame, rgb_data_in.size());
  cv::mixChannels(&small_frame, 1, &rgb_data_in, 1, from_to, 3);
```

```
  {
    cl::sycl::buffer<int>
      buf_in { (int *) rgb_data_in.data, NUMROWS*NUMCOLS },
      buf_prev { (int *) rgb_data_prev.data, NUMROWS*NUMCOLS },
      buf_out { (int *) rgb_data_out.data, NUMROWS*NUMCOLS };

    // Send the images to the pipes
    read_data(q, buf_in, buf_prev);

     // Color conversion and sobel on the current image
    rgb_pad2ycbcr_in(q);
    sobel_filter_pass(q);

    // Color conversion and sobel on the previous image
    // \todo Unify rgb_pad2ycbcr_in and rgb_pad2ycbcr_prev
    rgb_pad2ycbcr_prev(q);
    // \todo Unify sobel_filter and sobel_filter_pass
    sobel_filter(q);

    // Compare 2 sobel outputs
    diff_image(q);
    combo_image(q, 0);

    // Color conversion and receive image from pipe
    ycbcr2rgb_pad(q);
    write_data(q, buf_out);
  }
  std::cout << "frame " << framecnt++ << " done\n";
  cv::imshow(window_name, rgb_data_out);
  cv::waitKey(30);
}
```

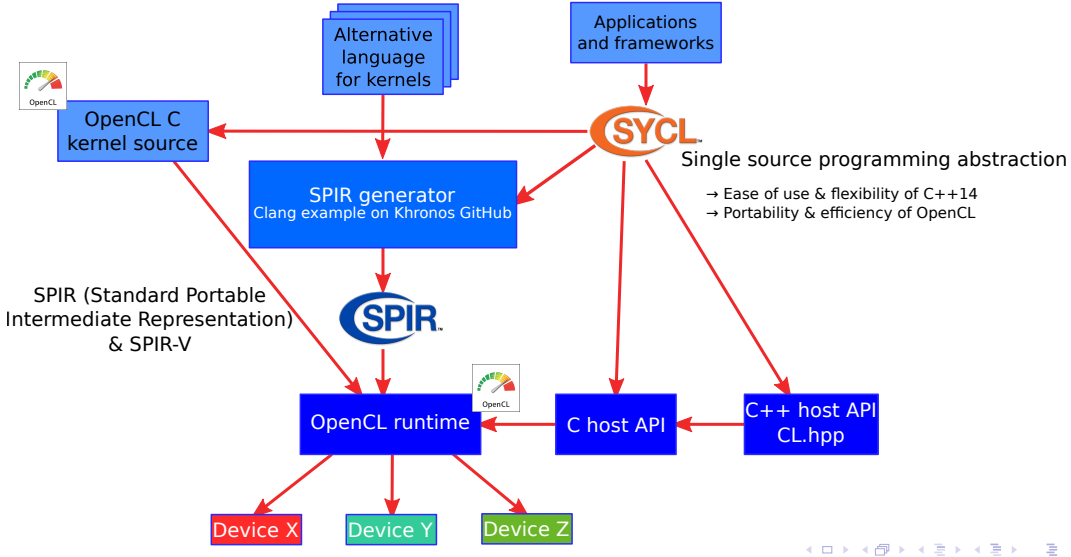
OpenCL interoperability mode

- SYCL $\equiv$ very generic parallel model
- Specific to SYCL: OpenCL interoperability *if needed*
- Can interact with OpenCL/Vulkan/OpenGL/... program or libraries *with no* overhead
- Has also some value for OpenCL programmers tool
  - Simplify boilerplate and housekeeping
    - Programming without explicit buffer transfer

Example of SYCL program running explicit OpenCL code

```
#include <iostream>
#include <iterator>
#include <boost/compute.hpp>
#include <boost/test/minimal.hpp>

#include <CL/sycl.hpp>

using namespace cl::sycl;

constexpr size_t N = 3;
using Vector = float[N];

int test_main(int argc, char *argv[]) {
  Vector a = { 1, 2, 3 };
  Vector b = { 5, 6, 8 };
  Vector c;

  // Construct the queue from the defaul OpenCL one
  queue q { boost::compute::system::default_queue() };

  // Create buffers from a & b vectors
  buffer<float> A { std::begin(a), std::end(a) };
  buffer<float> B { std::begin(b), std::end(b) };

  {
    // A buffer of N float using the storage of c
    buffer<float> C { c, N };

    // Construct an OpenCL program from the source string
    auto program = boost::compute::program::create_with_source(R"(
      __kernel void vector_add(const __global float *a,
                               const __global float *b,
                               __global float *c) {
        c[get_global_id(0)] = a[get_global_id(0)] + b[get_global_id(0)];
      }
      )", boost::compute::system::default_context());

    // Build a kernel from the OpenCL kernel
    program.build();

    // Get the OpenCL kernel
    kernel k { boost::compute::kernel { program, "vector_add" } };

    // Launch the vector parallel addition
    q.submit([&](handler &cgh) {
        /* The host-device copies are managed transparently by these
           accessors: */
        cgh.set_args(A.get_access<access::mode::read>(cgh),
                     B.get_access<access::mode::read>(cgh),
                     C.get_access<access::mode::write>(cgh));
        cgh.parallel_for(N, k);
      }); //< End of our commands for this queue
  } //< Buffer C goes out of scope and copies back values to c

  std::cout << std::endl << "Result:" << std::endl;
  for (auto e : c)
    std::cout << e << " ";
  std::cout << std::endl;

  return 0;
}
```

SYCL in OpenCL ecosystem



Parallel STL towards C++17 proposal

- Parallel STL now in C++17 (from proposal N4507, 2015/05/05)
  `http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2015/n4507.pdf`

```
// Current C++11: standard sequential sort
std::sort(vec.begin(), vec.end());
// C++17: permitting parallel execution and vectorization as well
sort(std::experimental::parallel::par_vec, vec.begin(), vec.end());
```

- Easy to implement in SYCL
  - Could even be extended to give a kernel name (profile, debug...):
  - Load balancing between CPU and accelerator

```
sycl_policy<class kernelName1> pol;
sort(pol, begin(vec), end(vec));

sycl_policy<class kernelName2> pol2;
// But SYCL allows OpenCL intrinsics in the operation too
for_each(pol2, vec.begin(), vec.end(),
         [](float & ans) { ans += cl::sycl::sin(ans); });
```

Open Source implementation ☺ `https://github.com/KhronosGroup/SyclParallelSTL`

Using SYCL-like models in other areas

- SYCL ≡ generic heterogeneous computing model beyond OpenCL
  - `device` abstracts the accelerators
  - `queue` allows to launch tasks with computations overlapping communications and pipelining
  - `parallel_for<>` for // computations
  - `accessor` defines the way we access data
  - `buffer` to chose where to store data
  - `allocator` for defining how data are allocated/backed and how pointers work
- Example in PiM (Processor-in-Memory)/Near-Memory Computing world
  - Use `queue` to run on some PiM chips
  - Use `allocator` to distribute data structures or to allocate `buffer` in special memory (memory page, chip...)
  - Use `accessor` to use alternative data access (split address from computation, streaming only, PGAS...)
  - Use `pointer_trait` to use specific way to interact with memory such as bank/transposition or relocation

Exascale-ready

- Use your own C++ compiler
  - Only kernel outlining needs SYCL compiler
- SYCL as plain single-source C++ can address most of the hierarchy levels
  - MPI
  - OpenMP
  - C++-based PGAS (Partitioned Global Address Space) DSEL (Domain-Specific embedded Language, such as Coarray C++...)
  - Remote accelerators in clusters
  - Use SYCL buffer allocator for
    - RDMA
    - Out-of-core, mapping to a file
    - PiM (Processor in Memory)
    - ...

Outline

Known implementations of SYCL

- ComputeCPP by Codeplay `https://www.codeplay.com/products/computecpp`
  - Most advanced SYCL 1.2 implementation
  - Outlining compiler generating SPIR
  - Run on any OpenCL device and CPU
  - Google & CodePlay have SYCL version of Eigen & TensorFlow using ComputeCPP
    `https://bitbucket.org/benoitsteiner/opencl`
    `https://github.com/benoitsteiner/tensorflow-opencl`
    `https://docs.google.com/spreadsheets/d/1YbHn7dAFPPG_PgTtgCJlWhMGorUPYsF681TsZ4Y4LP0`
- sycl-gtx `https://github.com/ProGTX/sycl-gtx`
  - Open source
  - No (outlining) compiler ⇝ use some macros with different syntax
- triSYCL `https://github.com/triSYCL/triSYCL`

triSYCL

- Open Source implementation using templated C++17 classes
  - On-going implementation started at AMD and now led by Xilinx
  - `https://github.com/triSYCL/triSYCL`
  - $\approx$ 10 contributors
- Used by Khronos committee to define the SYCL & OpenCL C++ standard
  - Languages are now too complex to be defined without implementing...
  - $\exists$ private Git repositories for future Khronos & experimental Xilinx versions
- Pure C++ implementation & CPU-only implementation for now
  - Use OpenMP for computation on CPU + `std::thread` for task graph
  - Rely on STL & Boost for zen style
  - CPU emulation for free
    - Quite useful for debugging
  - More focused on correctness than performance for now (array bound check...)
- Provide OpenCL-interoperability mode: can reuse existing OpenCL code
- Some extensions (Xilinx blocking pipes)
- On-going OpenCL implementation of outlining compiler based on open source Clang/LLVM compiler

Outline

Do it the standard way!

- SYCL is Khronos Group standard
- Push heterogeneous computing & FPGA-friendly features in core C++
- SYCL is candidate for SG14, presented at C++ F2F committee
  - Jacksonville 2016/02
    - http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2016/p0236r0.pdf
    - https://groups.google.com/a/isocpp.org/forum/#!topic/sg14/8GWWDulGE7o
  - Oulu 2016/06
    - http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2016/p0362r0.pdf
    - http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2016/p0363r0.pdf
    - http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2016/p0367r0.pdf
  - Kona 2017/03
    - http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2017/p0567r0.html
- Khronos committed to provide Open Source implementations of OpenCL and SYCL

Future

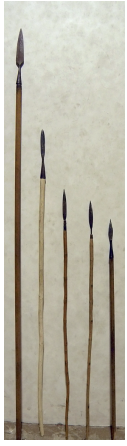
- SYCL DSEL task graph model is pretty generic and not only OpenCL-centric
  - Close to run-time such as StarPU, Nanos++, OpenAMP... and can deal with remote nodes, even with lower level API such as MPI, MCAPI...
    - SYCL can target these runtimes!
  - Actually even not restricted to C++ either (SYPyCL, SYJaCL, SYJSCL, SYCaml...). SYFortranCL on top of Fortran 2008?

Puns and pronunciation explained

OpenCL SYCL

sickle ['si-kəl]

OpenCL SPIR



spear ['spir]

Ecosystem: OpenCL, CUDA, SYCL, Vulkan, OpenMP, OpenACC... ?

- OpenCL 2.2 C++ $\approx$ NVIDIA CUDA Driver API (non single-source)
  - Low level for full control
  - Standard platform to build higher framework
- SYCL 2.2 C++ $\gtrapprox$ NVIDIA CUDA Runtime API, OpenMP, OpenACC (single-source)
  - Single-source higher-level C++ model for OpenCL programming
  - Domain-specific embedded language (DSEL) based on pure C++14 (1.2)/C++17 (2.2)
  - Do not require specific compiler for host code
  - Provide asynchronous task graph
  - Implicit buffer transfer between host and device with *accessors*

Conclusion

- Heterogeneous computing is everywhere
  - Modern FPGA are complex MP-SoC
  - Full systems & HPC machines add other complexity levels...
- In modern C++17 we trust
- SYCL C++ standard from Khronos Group
  - Pure modern C++17 DSEL for heterogeneous computing
  - Candidate for ISO C++ WG21 SG14 standard
  - Provide OpenCL interoperability if needed
  - SPIR-V extends OpenCL execution model to any language
- triSYCL
  - Open Source on-going implementation. Join us!
  - Use only pure C++17, OpenMP and Boost for CPU and OpenCL-compatible mode
  - On-going implementation of device compiler with Clang/LLVM
- Other implementations and libraries (Eigen, TensorFlow...) on `http://sycl.tech`
- A lot of hiring in the area...