

Layering Abstractions

Heterogeneous Programming and Performance Portability

Alastair Murray, Principal Software Engineer, Compilers

November 2nd 2017

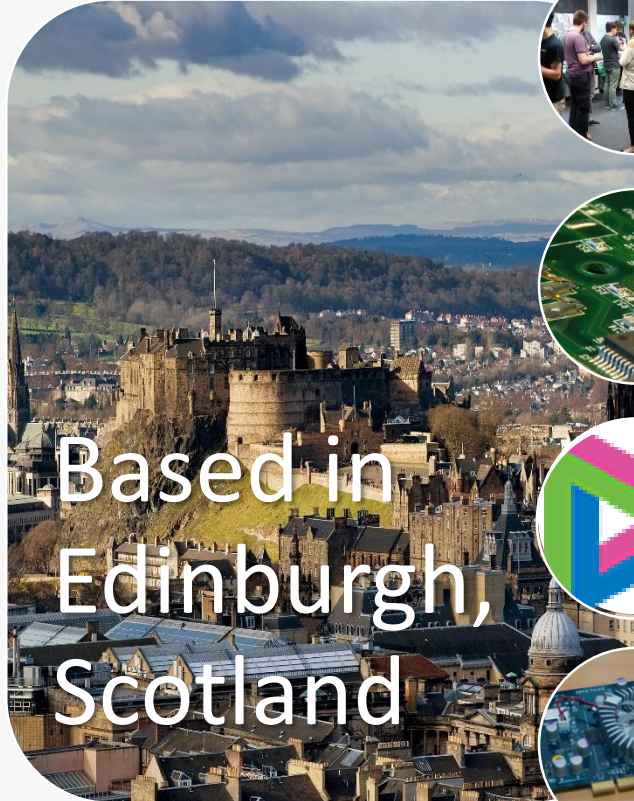
About me

- PhD at School of Informatics.
 - Developing compiler techniques for heavily customised embedded hardware.
- Post-doc at Virginia Tech.
 - Developing compiler and program mapping techniques for heterogeneous OS.
- Engineer/team-lead at Codeplay.
 - Developing compiler and language runtimes for heterogeneous hardware.
 - Team-lead for “ComputeAorta”: implementing OpenCL and Vulkan.
 - Member of Khronos OpenCL and OpenCL Safety Critical groups.
- Finding new ways to let programmers exploit new hardware.
 - Compilers, language-runtimes, language-design.

Overview

- Why worry about heterogeneous programming languages.
- How to push heterogeneous languages higher level.
- Why we also want to push them lower level.
- Where does performance portability fit into this.

Codeplay



66 staff, mostly engineering



License and customize technologies for semiconductor companies

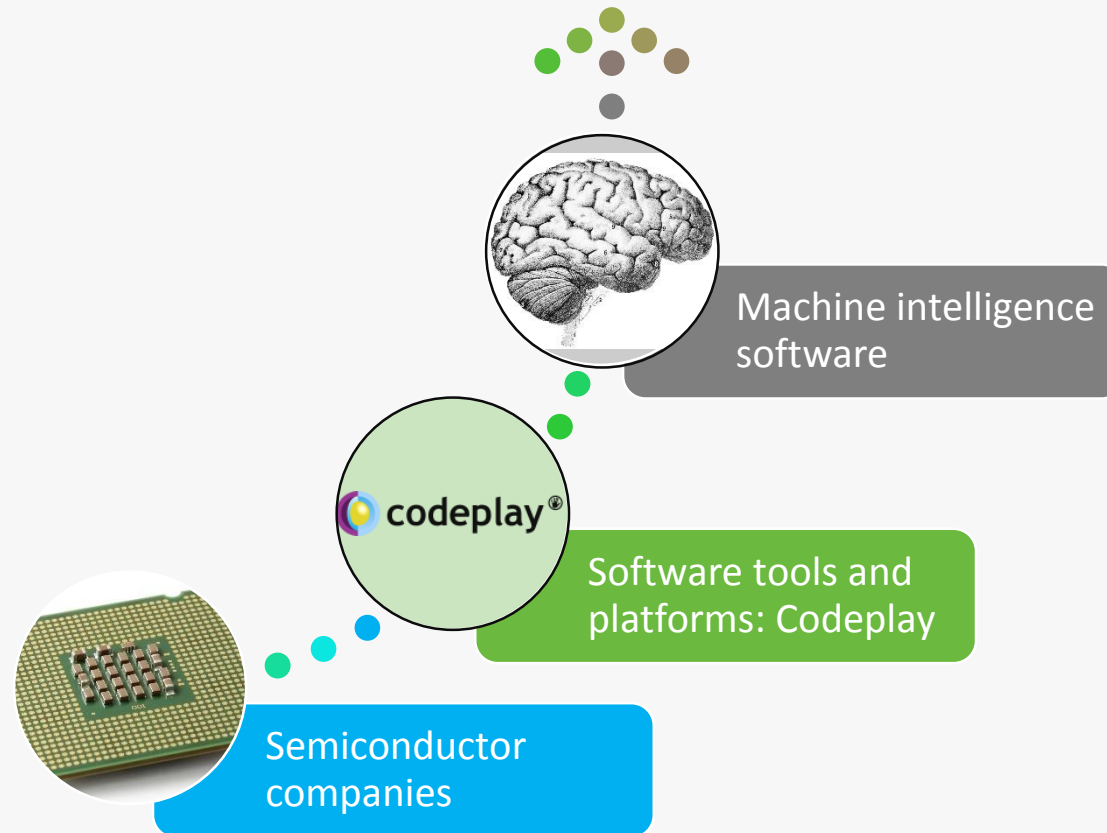


Products: ComputeAorta and ComputeCpp - implementations of OpenCL, Vulkan and SYCL



15+ years of experience in building heterogeneous systems tools

Where Codeplay fits in



Why heterogeneous languages are important

NHTSA's full final investigation into Tesla's Autopilot shows 40% crash rate reduction

Posted Jan 19, 2017 by [Darrell Etherington](#) (@etherington)



The U.S. National Highway Traffic Safety Administration has released its full findings following the investigation into last year's fatal crash involving a driver's use of Tesla's semi-autonomous Autopilot feature. The report clears Tesla's Autopilot system of any fault in the incident, and in fact at multiple points within the report praises its design in terms of

<https://techcrunch.com/2017/01/19/nhtsas-full-final-investigation-into-teslas-autopilot-shows-40-crash-rate-reduction/>
<http://www.forbes.com/sites/bernardmarr/2017/01/20/first-fda-approval-for-clinical-cloud-based-deep-learning-in-healthcare/#6e60fc8246e6>

ADVERTISEMENT

First FDA Approval For Clinical Cloud-Based Deep Learning In Healthcare



Bernard Marr, CONTRIBUTOR

I write about big data, analytics and enterprise performance [FULL BIO](#) ✓

Opinions expressed by Forbes Contributors are their own.

The first FDA approval for a machine learning application to be used in a clinical setting is a big step forward for AI and machine learning in healthcare and industry as a whole.

Why heterogeneous languages are important

Heterogeneous Computing **HERE TO STAY**

**HARDWARE
AND SOFTWARE
PERSPECTIVES**




MOHAMED ZAHARAN

Mentions of the buzzword *heterogeneous computing* have been on the rise in the past few years and will continue to be heard for years to come, because

December 23, 2016

Chips for Deep learning continue to leapfrog in capabilities and efficiency

artificial intelligence, computers, deep learning, europe, future, google

Deep learning has continued to drive the computing industry's agenda in 2016. But come 2017, experts say the Artificial Intelligence community will intensify its demand for higher performance and more power efficient "inference" engines for deep neural networks.

The current deep learning system leverages advances in large computation power to define network, big data sets for training, and access to the large computing system to accomplish its goal.

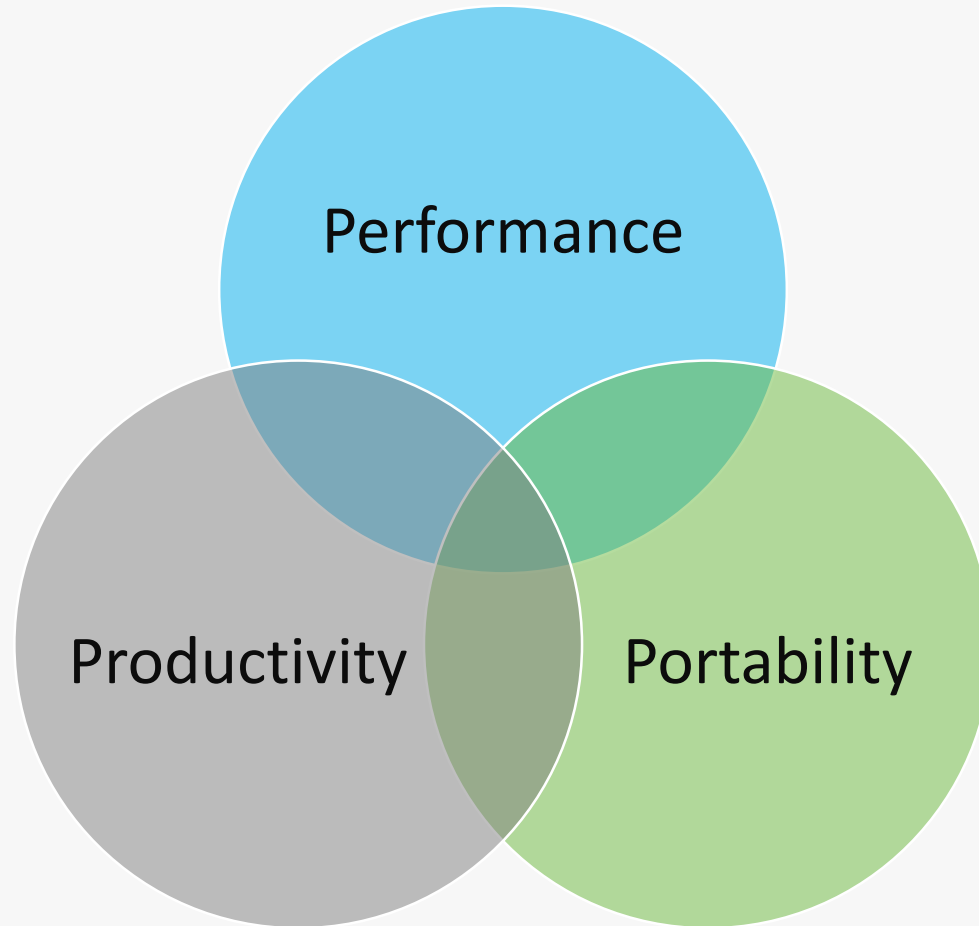
<https://doi.org/10.1145/3028687.3038873>

<http://www.nextbigfuture.com/2016/12/chips-for-deep-learning-continue-to.html>

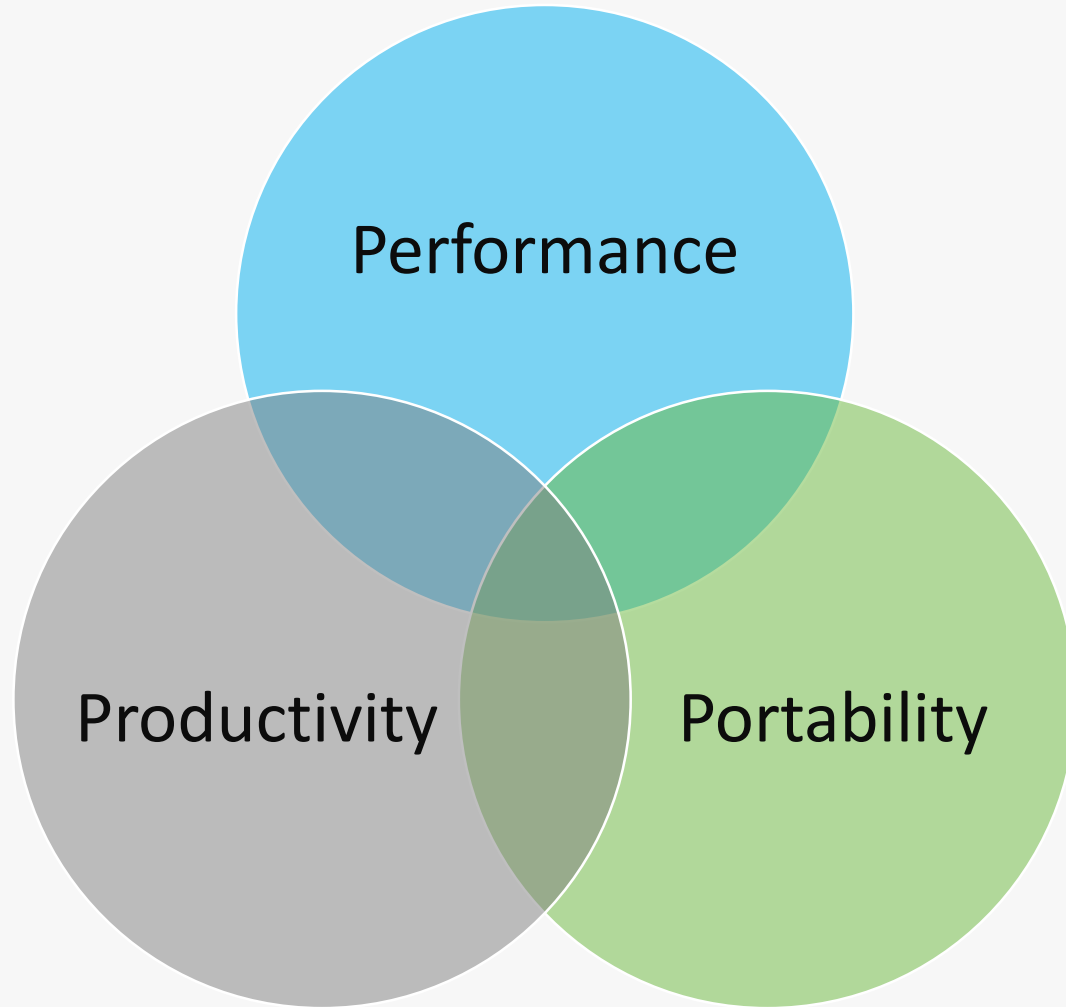
Gartner Hype Cycle

- See:
 - <https://www.gartner.com/smarterwithgartner/top-trends-in-the-gartner-hype-cycle-for-emerging-technologies-2017/>
- Many technologies require heterogeneous hardware.
 - Deep reinforcement learning.
 - Deep learning.
 - Machine learning.
 - Autonomous vehicles.
 - Cognitive computing.
 - Blockchain, etc.

Heterogeneous Programming Wishlist



Heterogeneous Programming Wishlist



- And ...
 - Correctness
 - Reliability
 - Predictability
 - Conciseness
 - Expressivity
 - Scalability
 - Tool support
 - Ecosystem
 - Etc.

Stacking Heterogeneous Languages

Low-level

- Low-level hardware-orientated programming models.
- Programmer has precise control of how everything is executed.

Stacking Heterogeneous Languages



- High-level programmer-orientated programming models.
- Programmer specifies what is to be executed.



- Low-level hardware-orientated programming models.
- Programmer has precise control of how everything is executed.

Stacking Heterogeneous Languages



DSLs

- Domain specific languages or libraries.
- Frequently use graph-based computational models.

High-level

- High-level programmer-orientated programming models.
- Programmer specifies what is to be executed.

Low-level

- Low-level hardware-orientated programming models.
- Programmer has precise control of how everything is executed.

Stacking Open Standards

DSLs

- Domain specific languages or libraries.
- Can be implemented using SYCL.

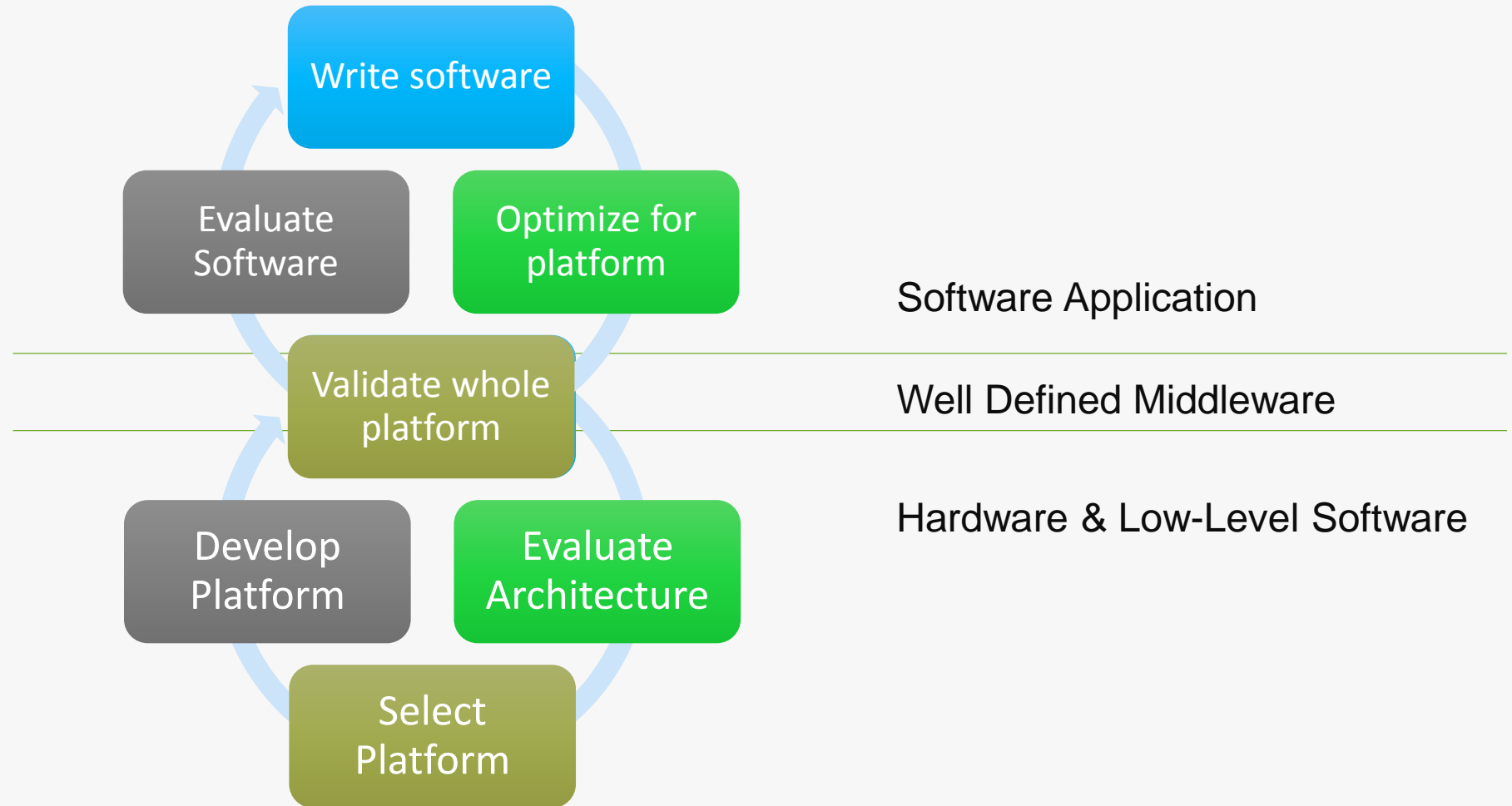
SYCL

- High-level C++ programming model.
- Builds on top of OpenCL & SPIR.

OpenCL

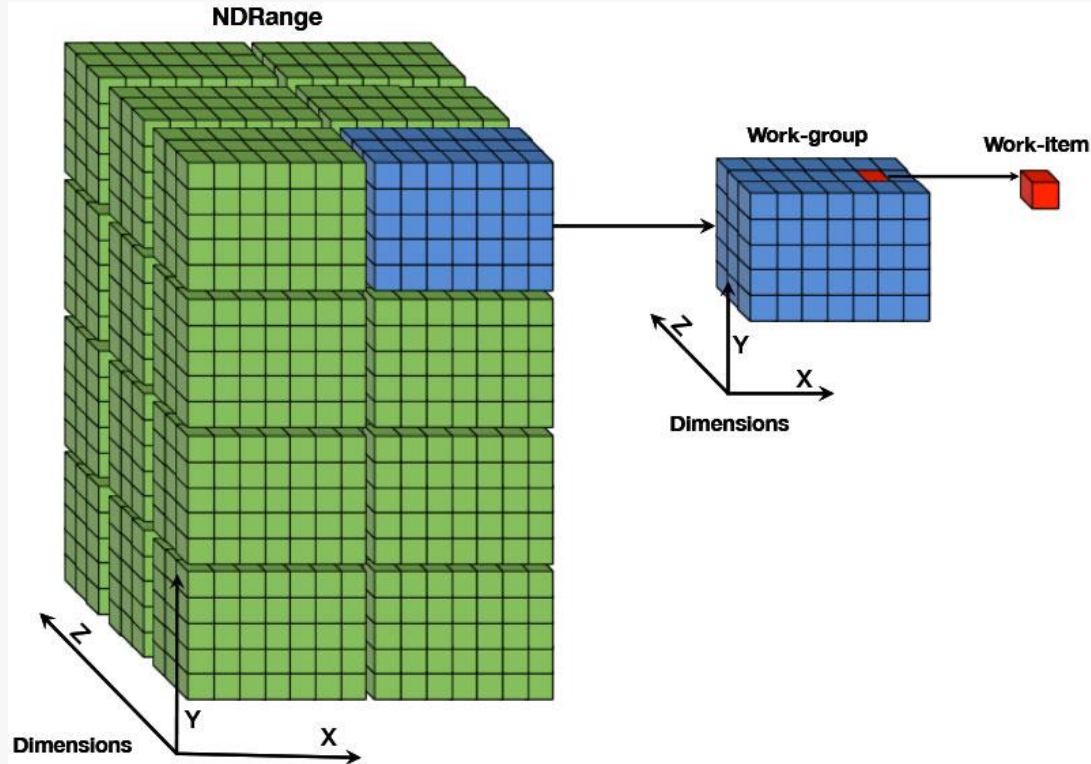
- Low-level heterogeneous C API.
- Widely supported.

Why open standards?



Abstractions: Going up the stack

Low-level Languages (OpenCL)



<http://rtcmagazine.com/articles/view/103610>

- Explicit work execution.
- Explicit memory management.
- Hierarchical [single node] parallelism model:
 - $\text{Work-item} \leq \text{sub-group} \leq \text{work-group} \leq \text{nd-range}$.
- Kernel memory model.

Problem: Performance portability

- Different hardware requires different approaches.
 - Functionally portable, but not performance portable.
 - An algorithm optimised for one architecture may perform terribly on another.
- E.g. Tiled matrix multiply.
 - GPU: Tile based on local memory size, explicit global to local copy, barriers.
 - DSP: Tile based on local memory size, `async_work_group_copy`.
 - CPU: Tile based on cache size, no local memory or barriers, let caches handle it.
- A fundamental property of being a low-level API?
 - So lets consider higher-level APIs.

OpenCL vs SYCL Kernel

OpenCL (Heavily Abbreviated)

```
const char *src =
    "__kernel void vecadd(global int *A,\n"
    "                      global int *B,\n"
    "                      global int *C) {\n"
    "    size_t gid = get_global_id(0);\n"
    "    C[gid] = A[gid] + B[gid];\n"
    "}\n"

clSetKernelArg(k, 0, sizeof(cl_mem), &ABuf);
clSetKernelArg(k, 1, sizeof(cl_mem), &BBuf);
clSetKernelArg(k, 2, sizeof(cl_mem), &CBuf);

clEnqueueNDRangeKernel(q, k, 1, NULL, {SIZE},
                      {32, 1, 1}, 0, NULL, NULL);
```

SYCL (Abbreviated)

```
auto A = ABuf.get_access<read>(cgh);
auto B = BBuf.get_access<read>(cgh);
auto C = CBuf.get_access<write>(cgh);

cgh.parallel_for<vecadd>(
    cl::sycl::range<1>(CBuf.size()),

    [=](cl::sycl::id<1> idx) {
        C[idx] = A[idx] + B[idx];
    });
```

OpenCL vs SYCL Kernel

OpenCL (Heavily Abbreviated)

```
const char *src =  
    "__kernel void vecadd(global int *A,\n"    "                        global int *B,\n"    "                        global int *C) {\n"    "    size_t gid = get_global_id(0);\n"    "    C[gid] = A[gid] + B[gid];\n"    "}"
```

```
clSetKernelArg(k, 0, sizeof(cl_mem), &ABuf);  
clSetKernelArg(k, 1, sizeof(cl_mem), &BBuf);  
clSetKernelArg(k, 2, sizeof(cl_mem), &CBuf);
```

```
clEnqueueNDRangeKernel(q, k, 1, NULL, {SIZE},  
                        {32, 1, 1}, 0, NULL, NULL);
```

SYCL (Abbreviated)

```
auto A = ABuf.get_access<read>(cgh);  
auto B = BBuf.get_access<read>(cgh);  
auto C = CBuf.get_access<write>(cgh);
```

```
cgh.parallel_for<vecadd>(  
    cl::sycl::range<1>(CBuf.size()),
```

```
[=](cl::sycl::id<1> idx) {  
    C[idx] = A[idx] + B[idx];  
});
```

Example: Parallel STL on SYCL

```
std::vector<int> vec = ...;
```

```
// Execute the for_each algorithm.  
std::parallel::foreach(par,  
    buf.begin(),  
    buf.end(),  
    [=](int& x) {  
        x += 2;  
    });
```

- STL: Algorithms via template meta-programming.
- Parallel STL: Parallel algorithms via template-metaprogramming.
 - Part of C++17.

Example: Parallel STL on SYCL

```
sycl::sycl_execution_policy<> sycl_policy;  
std::vector<int> vec = ...;
```

```
// Execute the for_each algorithm.  
std::parallel::foreach(sycl_policy,  
    buf.begin(),  
    buf.end(),  
    [=](int& x) {  
        x += 2;  
    });
```

- Can be implemented in SYCL.
 - <https://github.com/KhronosGroup/SyclParallelSTL/>

Parallel STL and Performance Portability

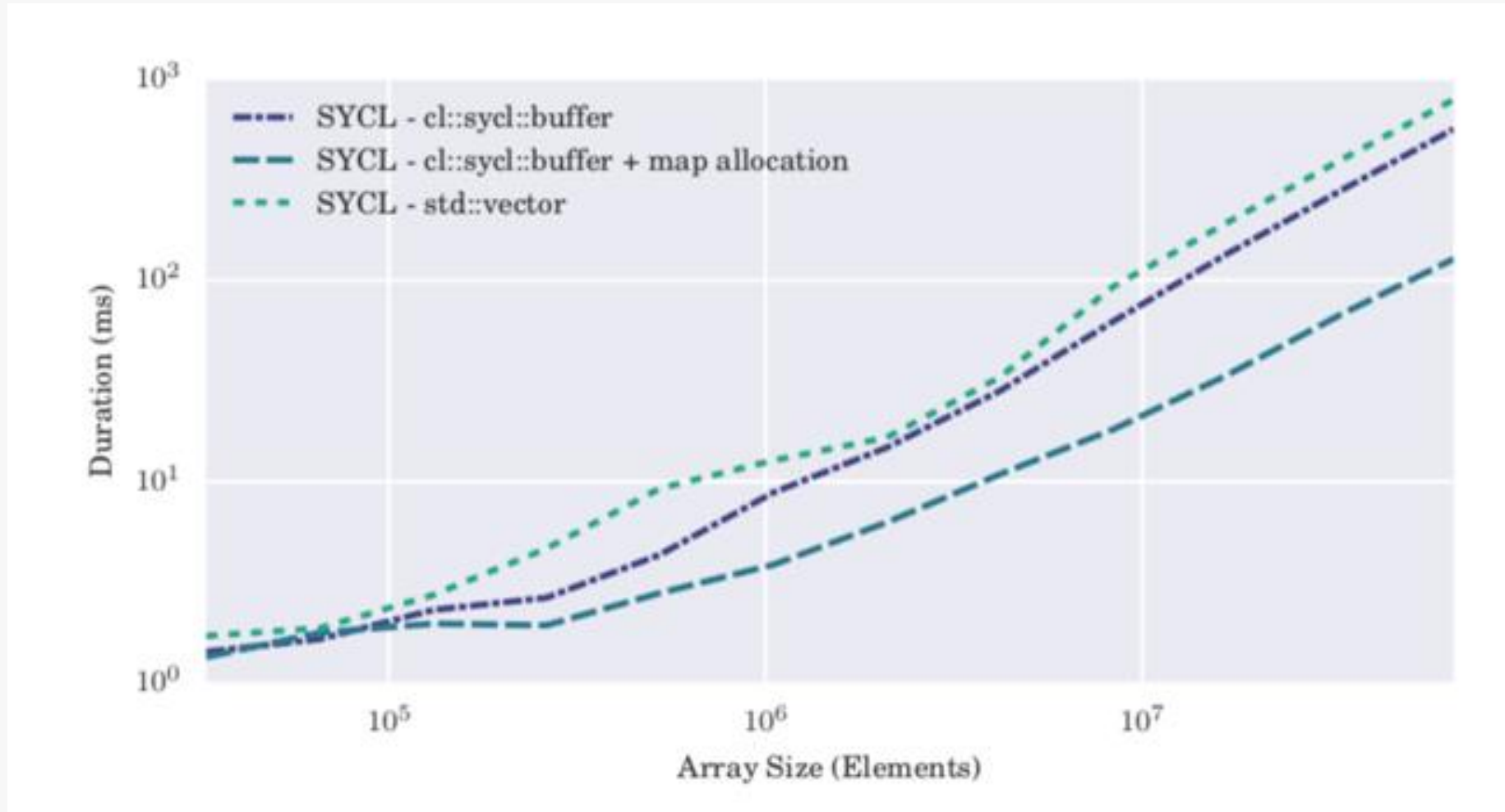
```
sycl::sycl_execution_policy<> sycl_policy;  
std::vector<int> vec = ...;
```

```
cl::sycl::range<1> range(vec.size());  
cl::sycl::buffer<int, 1, map_allocator<int>>  
    buf(vec.data(), range);
```

```
// Execute the for_each algorithm.  
std::parallel::foreach(sycl_policy,  
    sycl::helpers::begin(buf),  
    sycl::helpers::end(buf),  
    [=](int& x) {  
        x += 2;  
    });
```

- Manually specify a “map allocator”.
 - Tells the SYCL implementation that it can directly use the memory.
 - C++17 has contiguous iterator trait.
- Supports the case for DSLs:
 - General programming models never know exactly what the programmer will do.

Parallel STL and Performance Portability



OpenCL vs SYCL Kernel

OpenCL (Heavily Abbreviated)

```
const char *src =  
    "__kernel void vecadd(global int *A,\n"    "                        global int *B,\n"    "                        global int *C) {\n"    "    size_t gid = get_global_id(0);\n"    "    C[gid] = A[gid] + B[gid];\n"    "}"
```

```
clSetKernelArg(k, 0, sizeof(cl_mem), &ABuf);  
clSetKernelArg(k, 1, sizeof(cl_mem), &BBuf);  
clSetKernelArg(k, 2, sizeof(cl_mem), &CBuf);
```

```
clEnqueueNDRangeKernel(q, k, 1, NULL, {SIZE},  
                        {32, 1, 1}, 0, NULL, NULL);
```

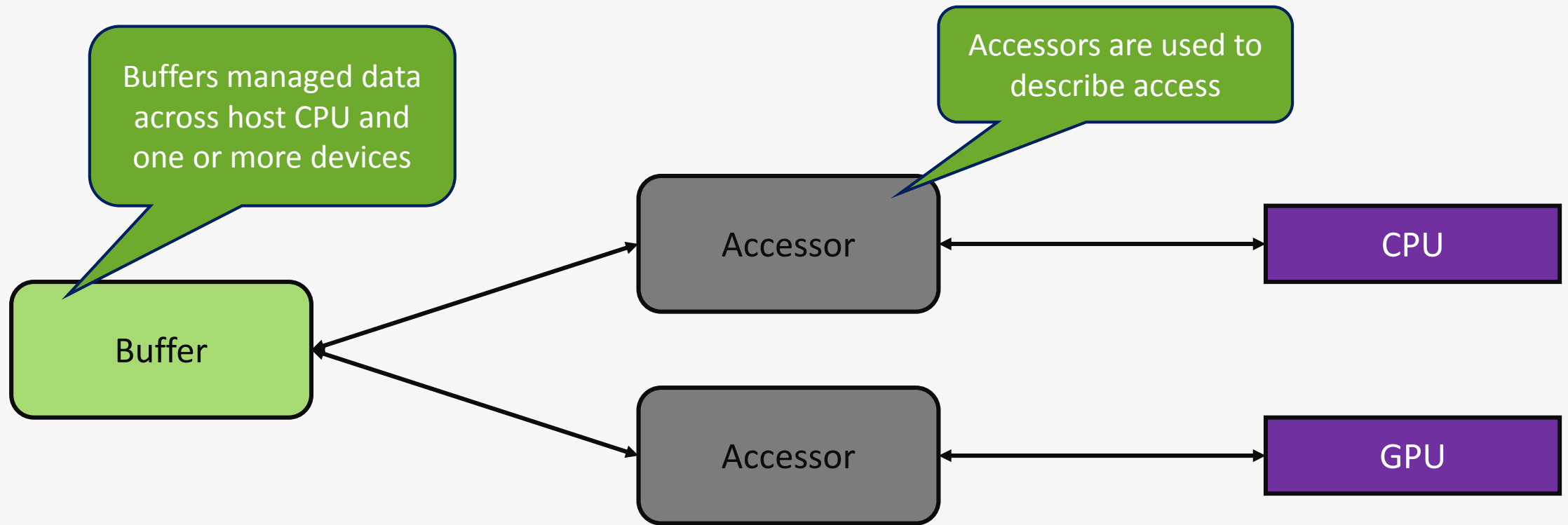
SYCL (Abbreviated)

```
auto A = ABuf.get_access<read>(cgh);  
auto B = BBuf.get_access<read>(cgh);  
auto C = CBuf.get_access<write>(cgh);
```

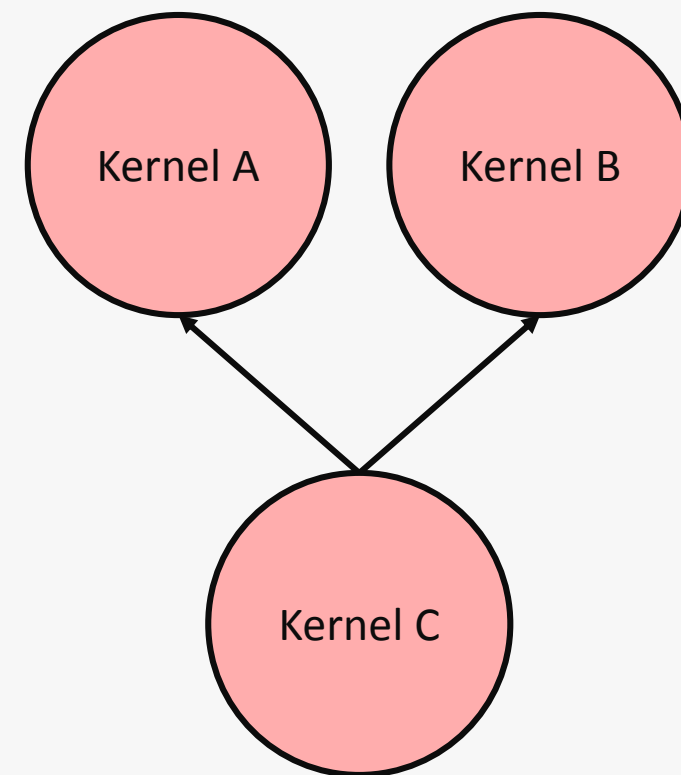
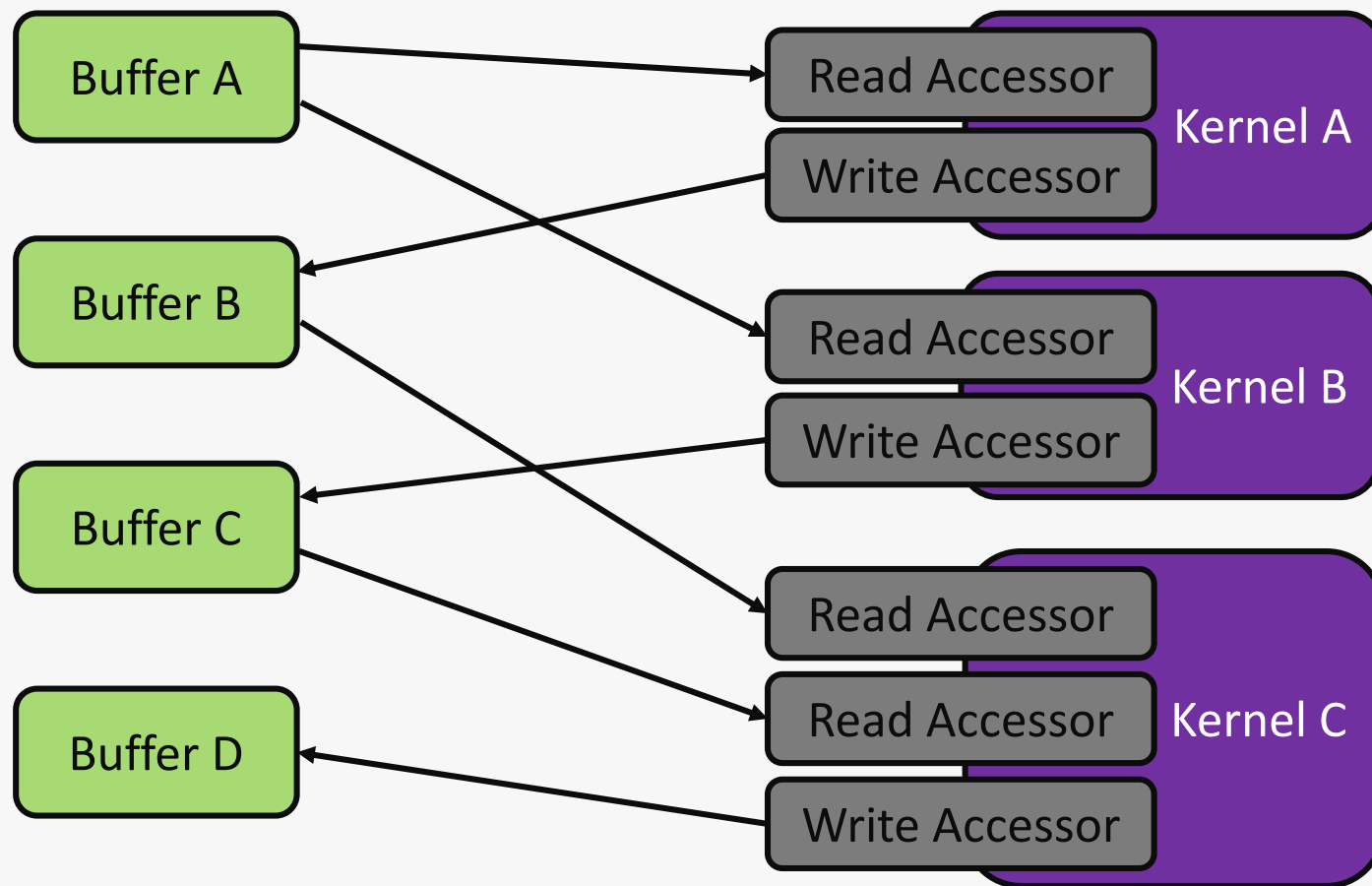
```
cgh.parallel_for<vecadd>(  
    cl::sycl::range<1>(CBuf.size()),
```

```
    [=](cl::sycl::id<1> idx) {  
        C[idx] = A[idx] + B[idx];  
    });
```

Separating Storage & Access



Data Dependency Task Graphs



SYCL and Performance Portability

- Builds on strengths of OpenCL, such as the optimised per-architecture implementation.
- Automatic memory management and dependency graph can often better utilise the hardware than a programmer can.
- Still requires the programmer to choose how best to map the problem to the parallelism model.
 - But at least it is comparatively easy to program it.

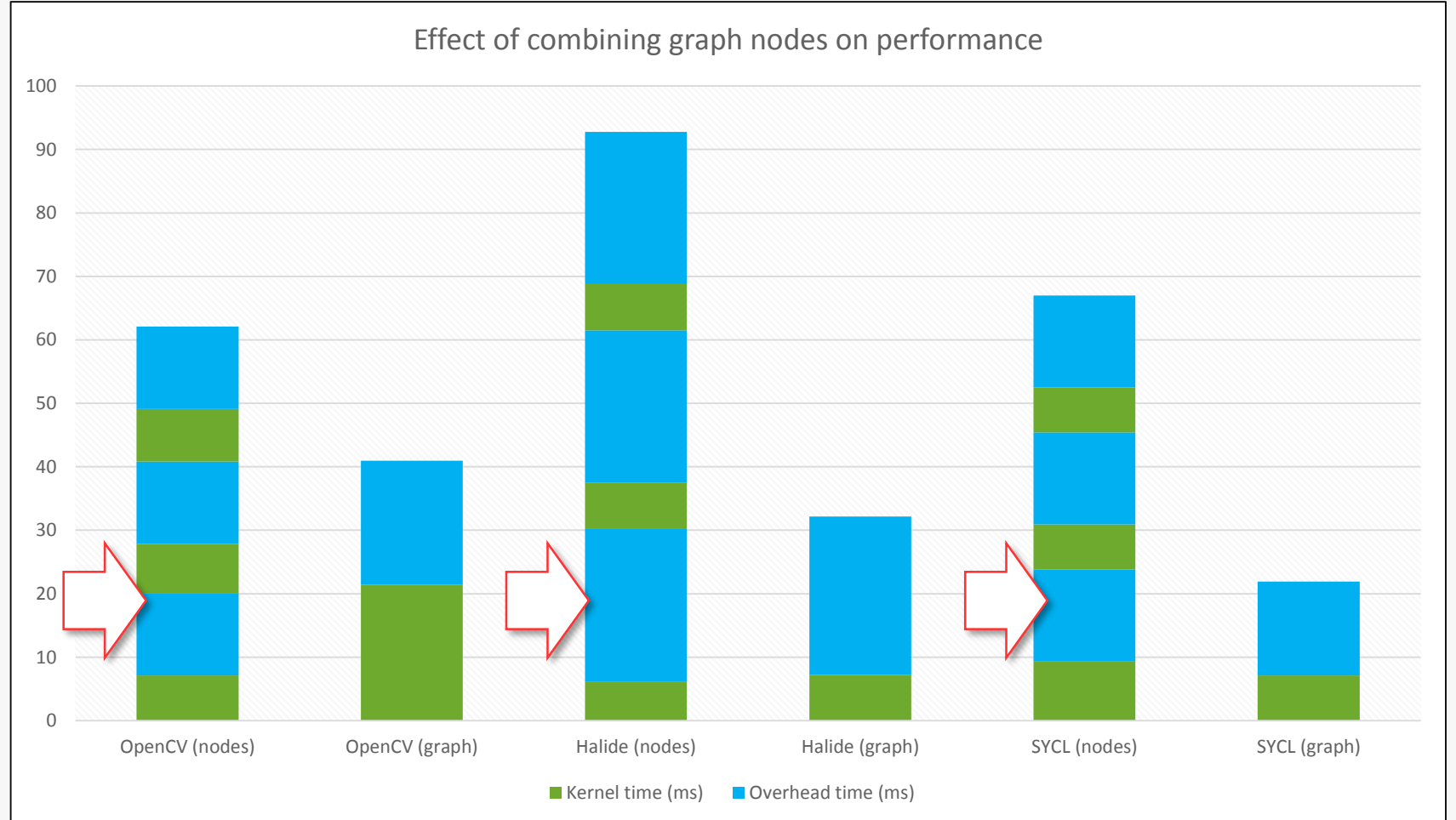
Domain Specific Languages and Libraries

- Key feature to enable DSLs: Metaprogramming.
 - In SYCL this primarily means C++ templates.
- Codeplay have implemented several DSLs on top of SYCL:
 - Tensorflow/Eigen
 - C++17 Parallel STL
 - VisionCpp
 - SYCL BLAS
- SYCL particularly suited to DSLs with graph execution models.
 - It's dependency tracking can create the graph automatically.
 - Possibility of implementing optimisations to fuse graph nodes.

Graph programming: Some Numbers

In this example, we perform 3 image processing operations on an accelerator and compare 3 systems when executing individual nodes, or a whole graph

The system is an AMD APU and the operations are: RGB->HSV, channel masking, HSV->RGB



DSLs and Performance Portability

- Programmers are solving their problems directly in the problem domain.
- Systems experts can put their knowledge into the DSL implementation.
- I.e., solutions must be implemented at the appropriate level.
 - High-level problems get implemented in their own domain.
 - DSLs get implemented in a high-level language.
 - Hardware-specific optimisations get done in a low-level language.

Abstractions: Going down the stack

OpenCL vs SYCL Kernel

OpenCL (Heavily Abbreviated)

```
const char *src =  
    "__kernel void vecadd(global int *A,\n"    "                        global int *B,\n"    "                        global int *C) {\n"    "    size_t gid = get_global_id(0);\n"    "    C[gid] = A[gid] + B[gid];\n"    "}"
```

```
clSetKernelArg(k, 0, sizeof(cl_mem), &ABuf);  
clSetKernelArg(k, 1, sizeof(cl_mem), &BBuf);  
clSetKernelArg(k, 2, sizeof(cl_mem), &CBuf);
```

```
clEnqueueNDRangeKernel(q, k, 1, NULL, {SIZE},  
                        {32, 1, 1}, 0, NULL, NULL);
```

SYCL (Abbreviated)

```
auto A = ABuf.get_access<read>(cgh);  
auto B = BBuf.get_access<read>(cgh);  
auto C = CBuf.get_access<write>(cgh);
```

```
cgh.parallel_for<vecadd>(  
    cl::sycl::range<1>(CBuf.size()),
```

```
    [=](cl::sycl::id<1> idx) {  
        C[idx] = A[idx] + B[idx];  
    });
```

OpenCL as a Base for Higher-level Languages

- Accelerating DSLs and libraries is a key use-case for OpenCL!
 - SYCL, Caffe, Halide, OpenVX, OpenCV, ViennaCL, ArrayFire, etc
- However, awkward to compile high-level kernels to OpenCL-C.
 - C was intended for programmers to write, not tools to generate.
- Much better to compile a high-level language to IR: SPIR-V
 - Primary use case is to abstract away the kernel language.
 - SPIR-V is slightly higher level than LLVM IR, has structured control flow.
 - “Next 700 heterogeneous languages.”



OpenCL vs SYCL Kernel

OpenCL (Heavily Abbreviated)

```
const char *src =
    "__kernel void vecadd(global int *A,\n"
    "                      global int *B,\n"
    "                      global int *C) {\n"
    "    size_t gid = get_global_id(0);\n"
    "    C[gid] = A[gid] + B[gid];\n"
    "}\n"

clSetKernelArg(k, 0, sizeof(cl_mem), &ABuf);
clSetKernelArg(k, 1, sizeof(cl_mem), &BBuf);
clSetKernelArg(k, 2, sizeof(cl_mem), &CBuf);

clEnqueueNDRangeKernel(q, k, 1, NULL, {SIZE},
                      {32, 1, 1}, 0, NULL, NULL);
```

SYCL (Abbreviated)

```
auto A = ABuf.get_access<read>(cgh);
auto B = BBuf.get_access<read>(cgh);
auto C = CBuf.get_access<write>(cgh);

cgh.parallel_for<vecadd>(
    cl::sycl::range<1>(CBuf.size()),

    [=](cl::sycl::id<1> idx) {
        C[idx] = A[idx] + B[idx];
    });
```

Problem: Mixing work creation with work dispatch

- Affects the efficiency of multi-threaded programs.
- Local work-group sizes set within `clEnqueueNDRangeKernel`.
 - If optimisations are to exploit local work group size, compilation must be deferred.

Vulkan

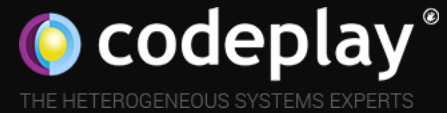
- Graphics API but with compute capabilities.
- Much lower-level than OpenCL, extremely explicit.
 - I.e. as a graphics API it is for engine developers rather than game creators.
 - Much like low-level compute APIs could be for language implementers.
- Sacrifices that Vulkan makes for the sake of performance:
 - Separate work construction from work execution.
 - Elide error handling (replace with validation and debug layers).
 - Keep the “fast path” fast (i.e. doing work).

Conclusion

- Heterogeneous programming languages are becoming:
 - Higher level ...
 - ... and lower level.
 - Pick the correct level to solve your problem.
- Heterogeneous programming languages can stack.
 - Helps to manage programming the wide variety of hardware out there.
 - Existence of high-level models free the low-levels to go even lower.
- Performance portability is still a lot of work.

We're
Hiring!

codeplay.com/careers/



Thank you!



[@codeplaysoft](https://twitter.com/codeplaysoft)



info@codeplay.com



codeplay.com