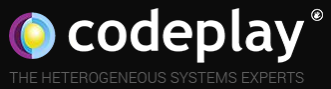


The future of C++ and the Four Horsemen of Heterogeneous C++

Michael Wong, Codeplay VP of R & D
Khronos SYCL team

LLVM 2018

Legal Disclaimer

This work represents the view of the author and does not necessarily represent the view of Codeplay.

Other company, product, and service names may be trademarks or service marks of others.

Acknowledgement Disclaimer

Numerous people internal and external to the original C++/Khronos group, in industry and academia, have made contributions, influenced ideas, written part of this presentations, and offered feedback to form part of this talk.

We even lifted this acknowledgement and disclaimer from some of them.

But we claim all credit for errors, and stupid mistakes. **These are ours, all ours!**

Codeplay - Connecting AI to Silicon

Products

ComputeCpp™

C++ platform via the SYCL™ open standard, enabling vision & machine learning e
TensorFlow™

ComputeAorta™

The heart of Codeplay's compute technology
enabling OpenCL™, SPIR™, HSA™ and Vulkan™

Addressable Markets

Automotive (ISO 26262)
IoT, Smartphones & Tablets
High Performance Compute (HPC)
Medical & Industrial

Technologies: Vision Processing
Machine Learning
Artificial Intelligence
Big Data Compute

Company

High-performance software solutions for custom heterogeneous systems

Enabling the toughest processor systems with tools and middleware based on open standards

Established 2002 in Scotland

~70 employees

Customers

BROADCOM
RENESAS
Imagination
QUALCOMM
Movidius
intel Partners AMD

3 Act Play

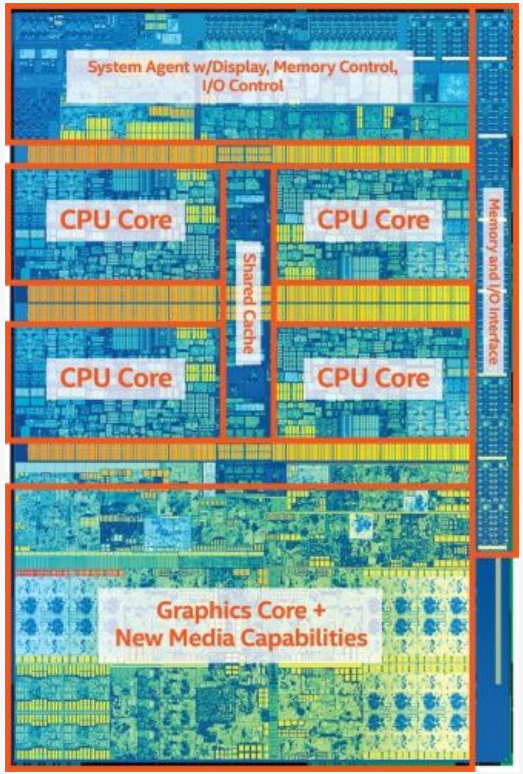
What excites me?

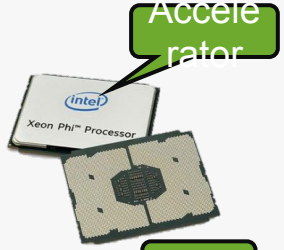
What are the 4 horsemen of Heterogeneous Computing?

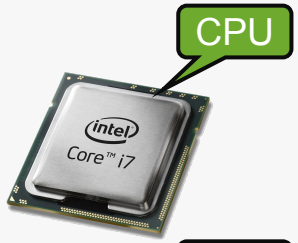
Can we do it in ISO C++ and implement it in Clang?

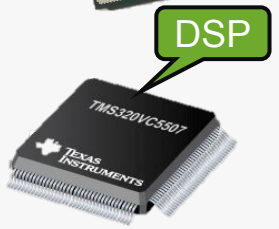
Act 1

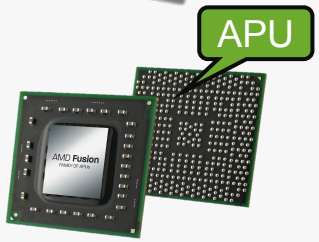
What gets me up every morning?











Firing on all cylinders

What this means to you

Iron Triangle of Parallel Programming Language Nirvana

Portability

Performance

Productivity

In 1998, a typical machine had the following flops

.45 GFLOPS, 1 core

Single threaded C++98/C99/Fortran dominated this picture

In 2011, a typical machine had the following flops

80 GFLOPS 4 cores

To program the CPU, you might use C/C++11, OpenMP, TBB, Cilk, OpenCL

In 2011, a typical machine had the following flops

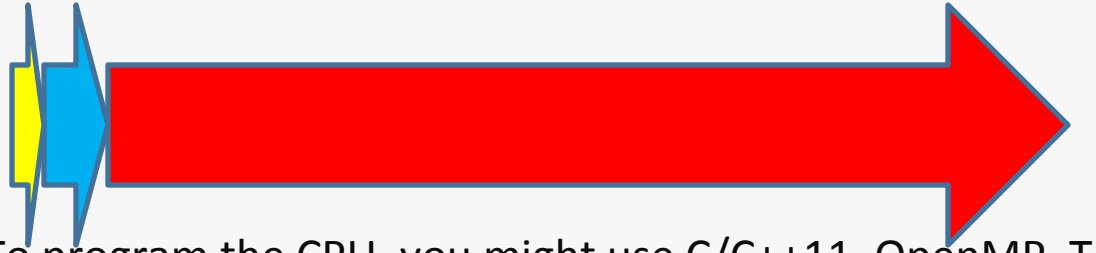
80 GFLOPS 4 cores+140 GFLOPS AVX

To program the CPU, you might use C/C++11, OpenMP, TBB, Cilk, CUDA, OpenCL

To program the vector unit, you have to use Intrinsics, OpenCL, CUDA, or auto-vectorization

In 2011, a typical machine had the following flops

80 GFLOPS 4 cores+140 GFLOPS AVX+2500 GFLOPS GPU



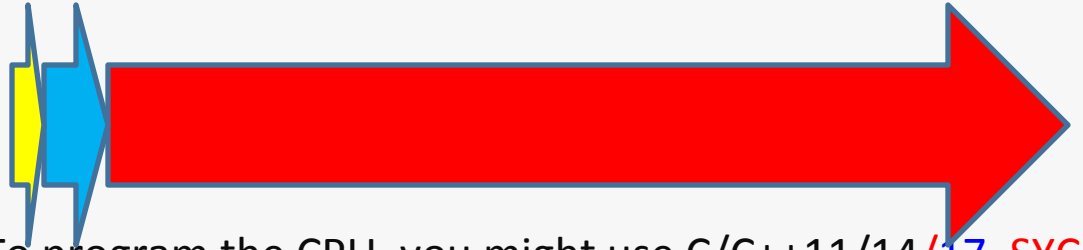
To program the CPU, you might use C/C++11, OpenMP, TBB, Cilk, CUDA, OpenCL

To program the vector unit, you have to use Intrinsics, OpenCL, CUDA or auto-vectorization

To program the GPU, you have to use CUDA, OpenCL, OpenGL, DirectX, Intrinsics, C++AMP

In 2017, a typical machine had the following flops

140 GFLOPS + 560 GFLOPS AVX + 4600 GFLOPS GPU



To program the CPU, you might use C/C++11/14/17, SYCL, OpenMP, TBB, Cilk, CUDA, OpenCL

To program the vector unit, you have to use SYCL, Intrinsics, OpenCL, CUDA or auto-vectorization, OpenMP

To program the GPU, you have to use SYCL, CUDA, OpenCL, OpenGL, DirectX, Intrinsics, OpenMP

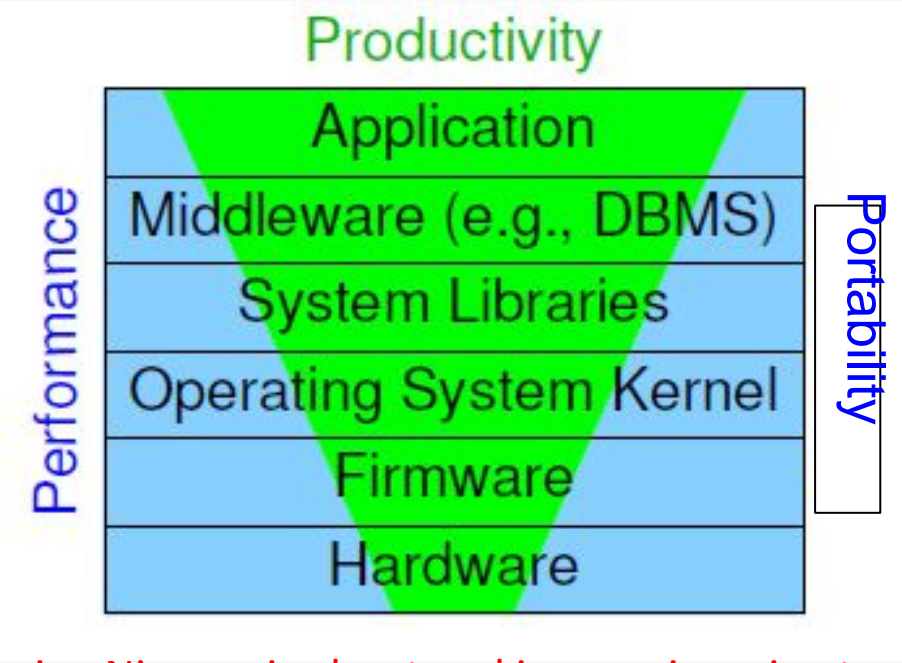
Performance Portability Productivity

OpenCL

OpenMP

CUDA

SYCL



Iron Triangle of Parallel Programming Nirvana is about making engineering tradeoffs

OpenCL

OpenMP

SYCL

CUDA

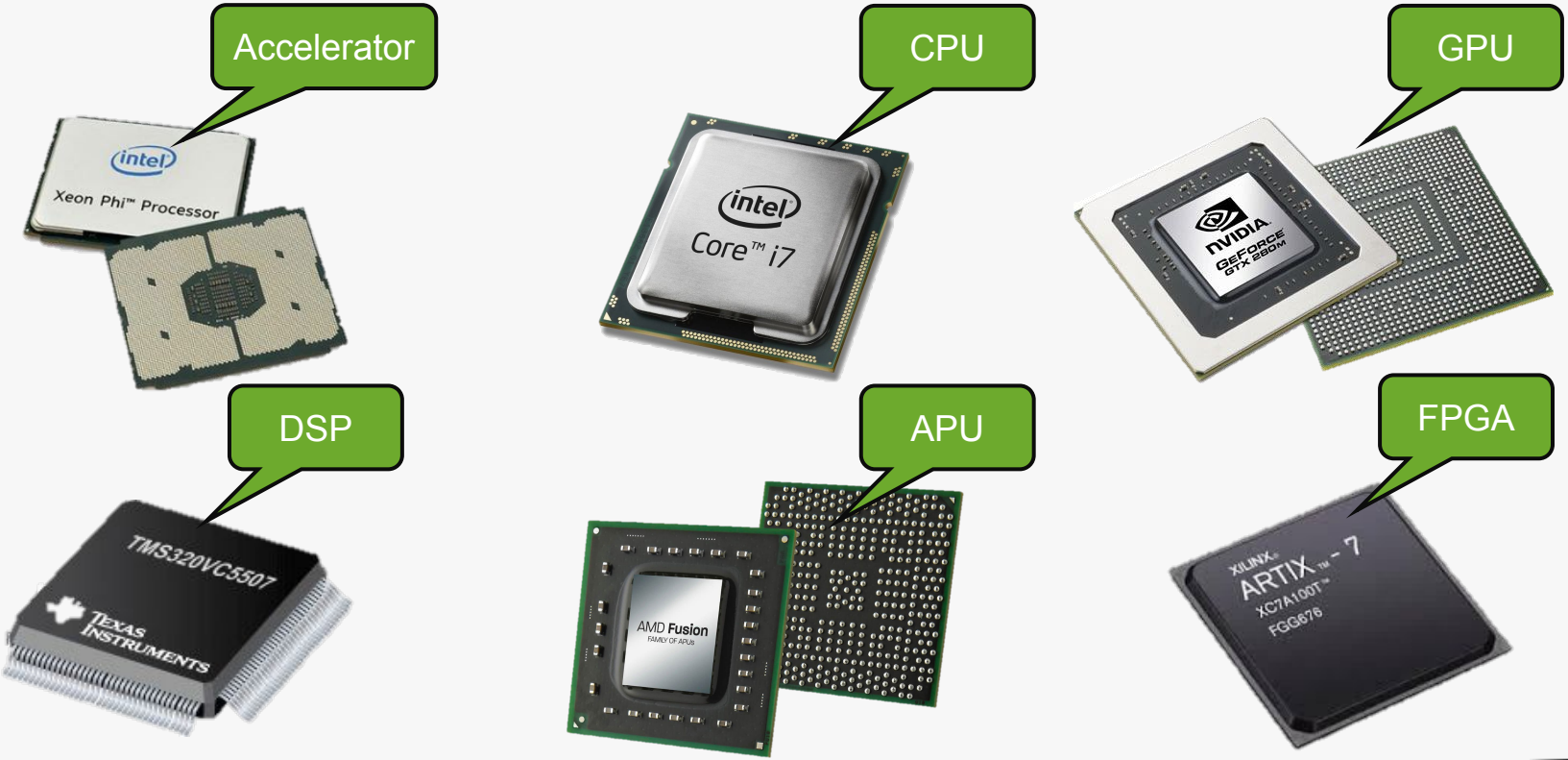
Kokkos

HPX

Raja

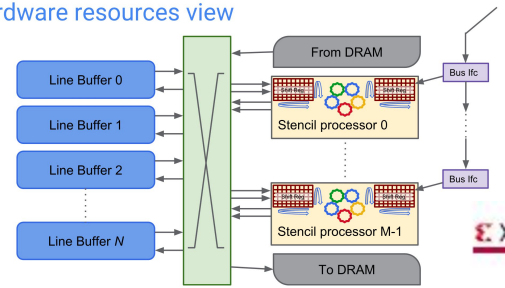
Boost.Compute

Heterogeneous Devices



Hot Chips

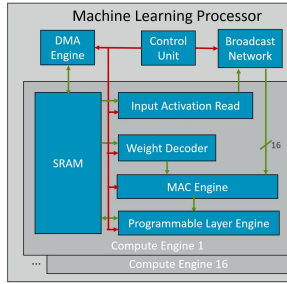
Hardware resources view



Google

Arm's ML processor: Summary

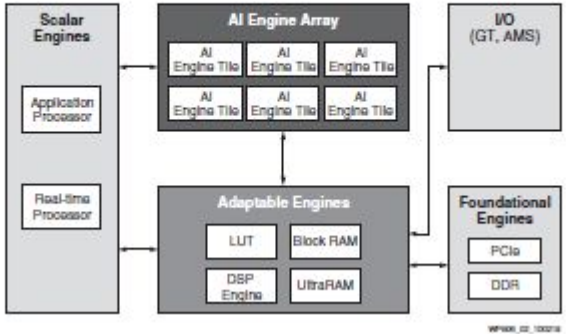
- 16 Compute Engines
- ~ 4 TOP/s of convolution throughput (at 1 GHz)
- Targeting > 3 TOP/W in 7nm and ~2.5mm²
- 8-bit quantized integer support
- 1MB of SRAM
- ...roid NNAPI and



XILINX.

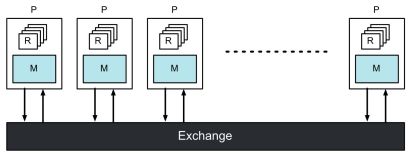
Xilinx AI Engines and Their Applications

018



Pure distributed machine with compiled communica

- Static partitioning of work and memory
- Threads hide only local latencies (arithmetic, memory, branch)
- Deterministic communication over a stateless "exchange"



ScaledML 2018

13

XAVIER INNOVATIONS

World's First Autonomous Machines Processor

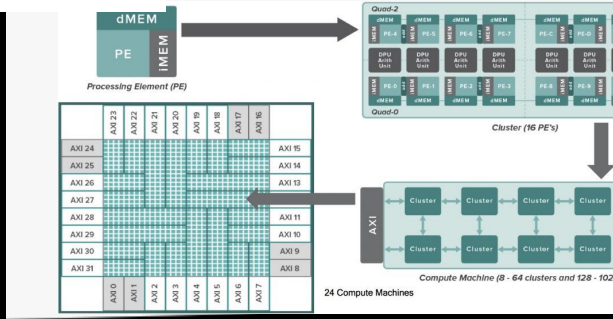
Carmel CPU	Volta GPU	DLA	PVA	ISP	MM-Accelerators
8 custom cores, ARM v8.2	512 CUDA Tensor Cores 22.6 Int8 DL TOPs	Int8/Int16/FP16 11.4 DL Int8 TOPs	7-slot VLIW 1.7 TOPs	Native HDR Processing 2.4 GPIX/sec	Stereo, Optical Flow, LDC

High Speed I/Os : >40GB/s of IO Bandwidth

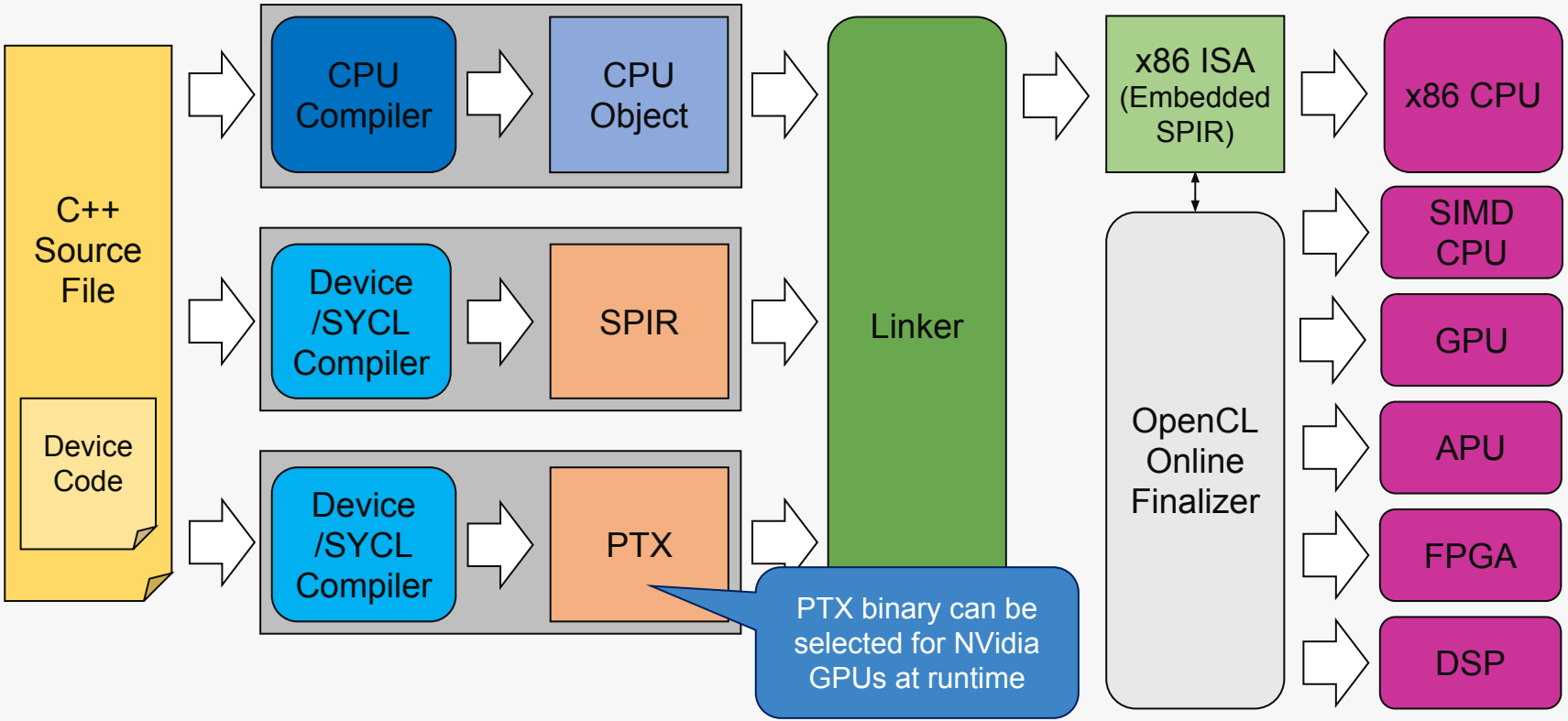
Designed for Safety & Resiliency : ISO26262; ASIL-C

Enhanced Security

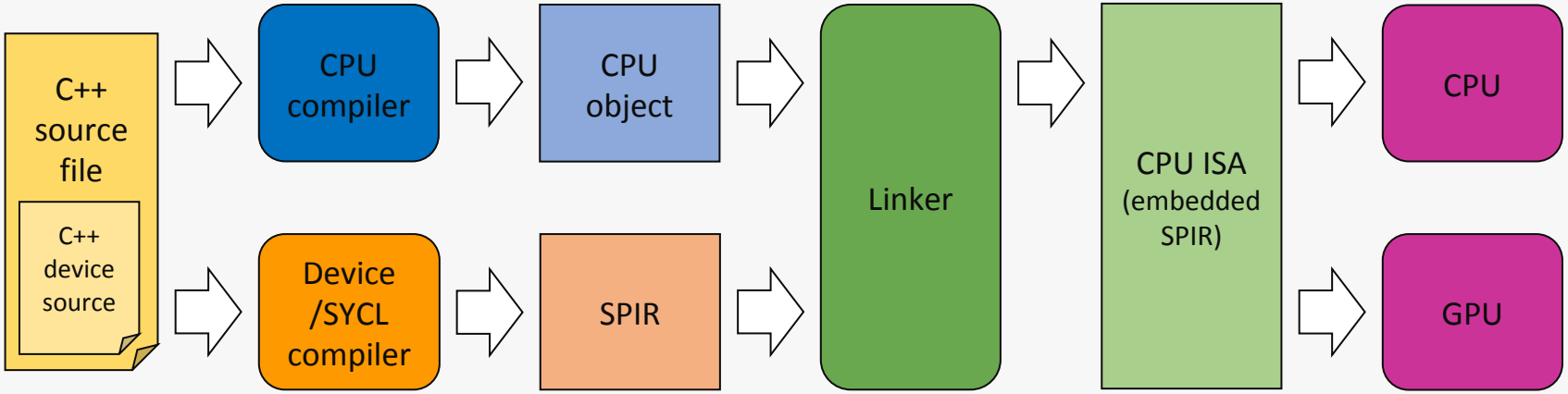
Optimized for Energy Efficiency; TSMC 12FFN



Multi Compilation Model like Shaders

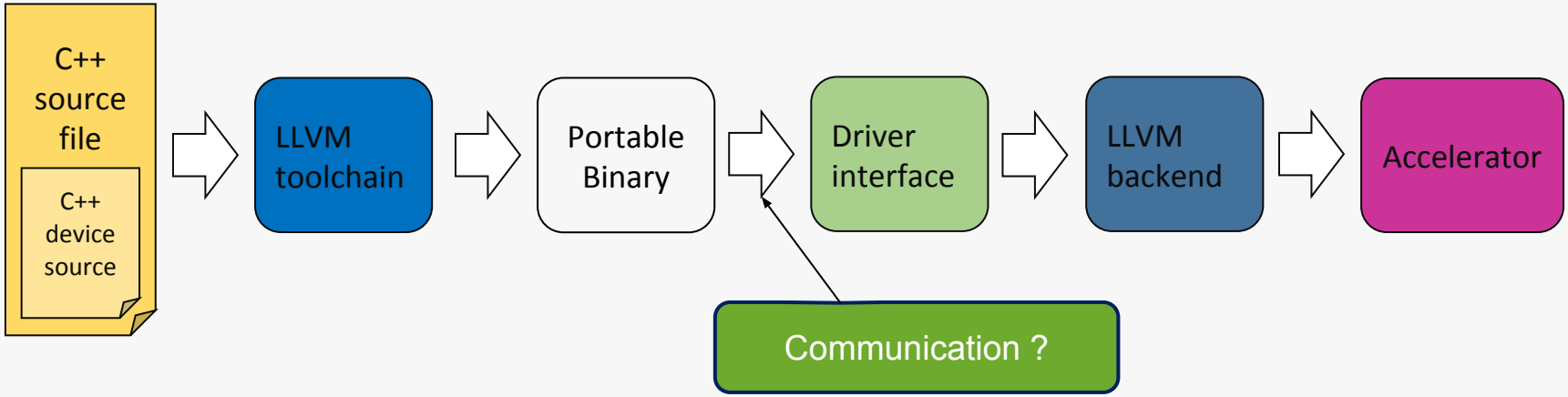


SYCL single source compilation model

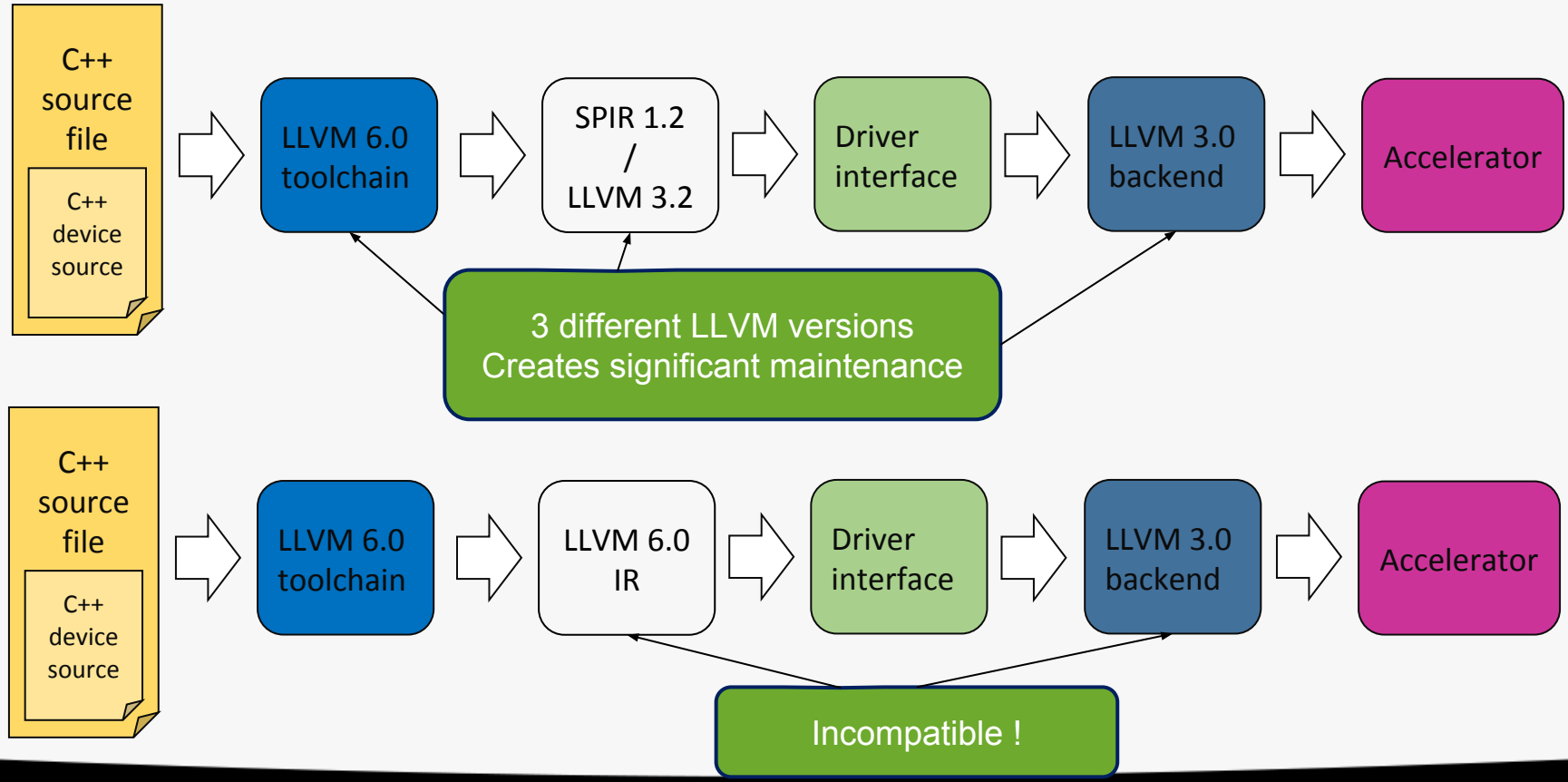



```
auto aAcc = aBuf.get_access<access::read>(cgh);
auto bAcc = bBuf.get_access<access::read>(cgh);
auto oAcc = oBuf.get_access<access::write>(cgh);
cgh.parallel_for<add>(range<1>(a.size()), [=](id<1> idx) {
  oAcc[i] = aAcc[i] + aAcc[i];
}));
```

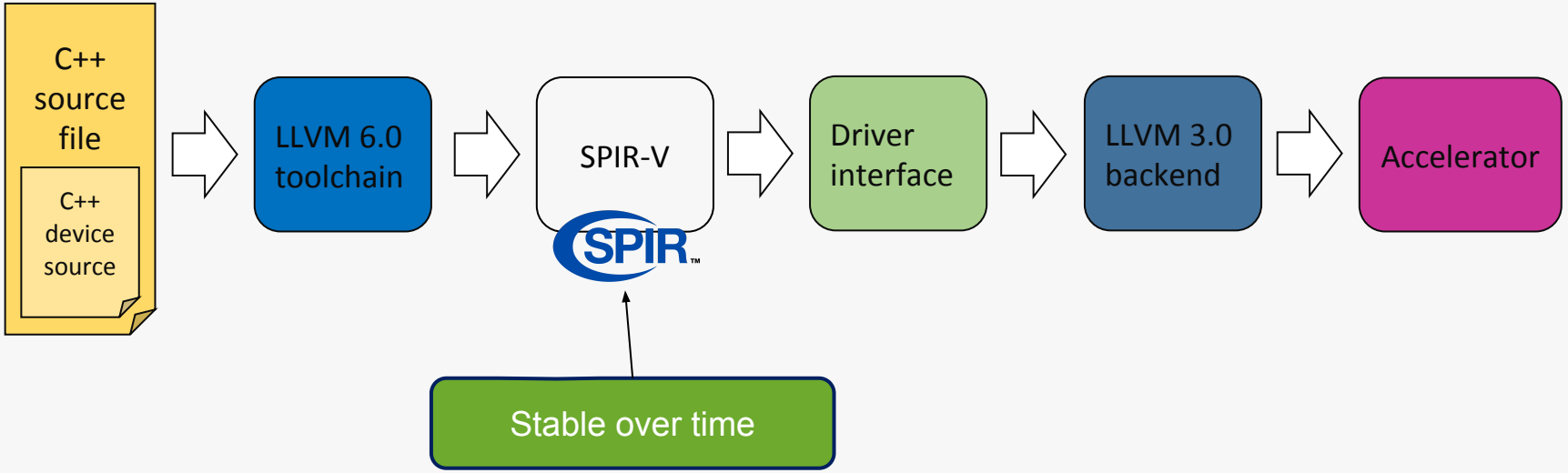
SPIR-V Transforms the Language Ecosystem



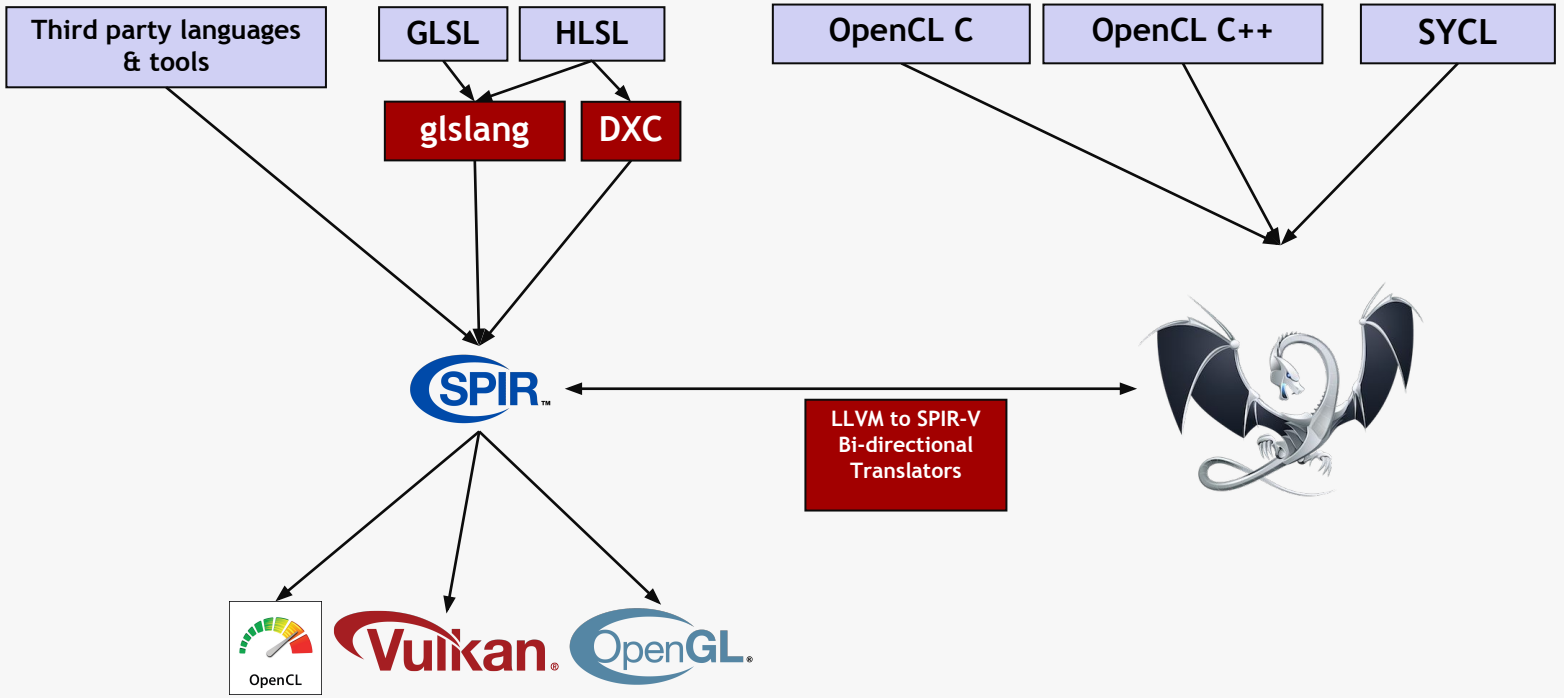
Need for a stable IR



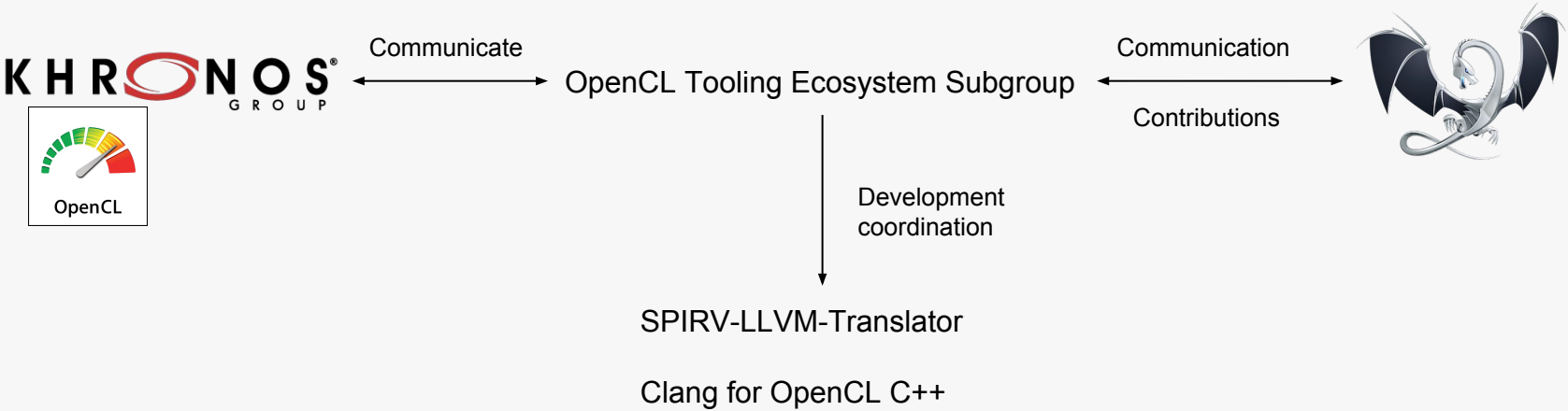
Support for Both SPIR-V and LLVM



SPIR-V Ecosystem



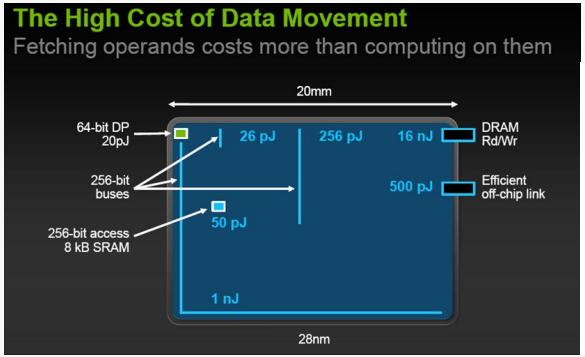
OpenCL Tooling Ecosystem Subgroup

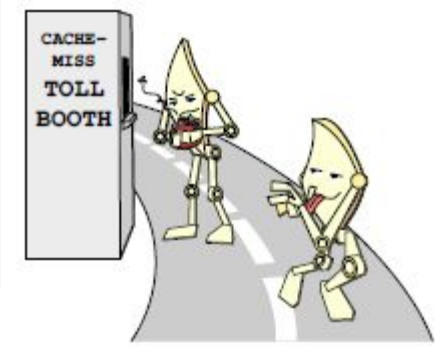


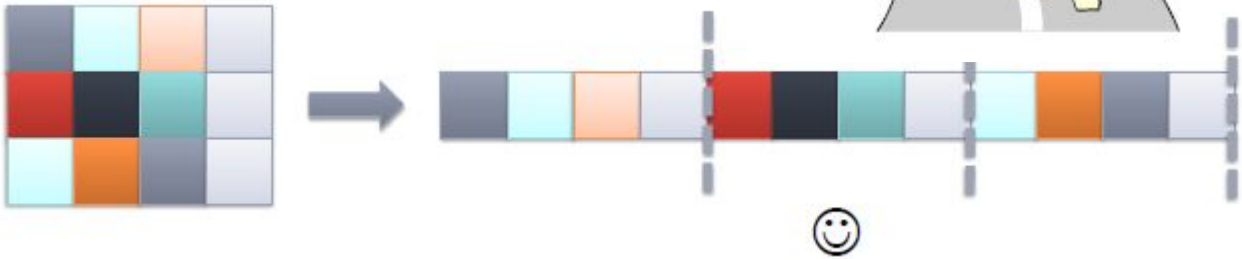
Act 2

What are the four Horsemen of Heterogeneous Computing?

The Four Horsemen

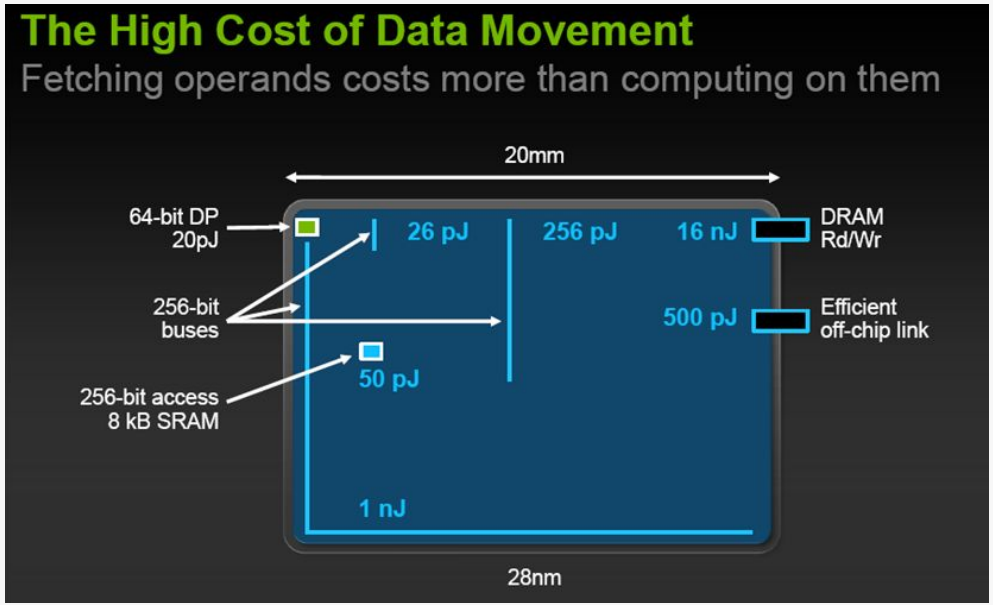






Socket 0								Socket 1							
Core 0				Core 1				Core 0				Core 1			
0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3
0	4			1	5			2	6			3	7		

Cost of Data Movement



Credit: Bill Dally, Nvidia, 2010

- 64bit DP Op:
  - 20pJ
- 4x64bit register read:
  - 50pJ
- 4x64bit move 1mm:
  - 26pJ
- 4x64bit move 40mm:
  - 1nJ
- 4x64bit move DRAM:
  - 16nJ

Implicit vs Explicit Data Movement

Examples:

- SYCL, C++ AMP

Implementation:

- Data is moved to the device implicitly via cross host CPU / device data structures

Here we're using C++ AMP as an example

```
array_view<float> ptr;
extent<2> e(64, 64);
parallel_for_each(e, [=](index<2> idx)
restrict(amp) {
  ptr[idx] *= 2.0f;
});
```

Examples:

- OpenCL, CUDA, OpenMP

Implementation:

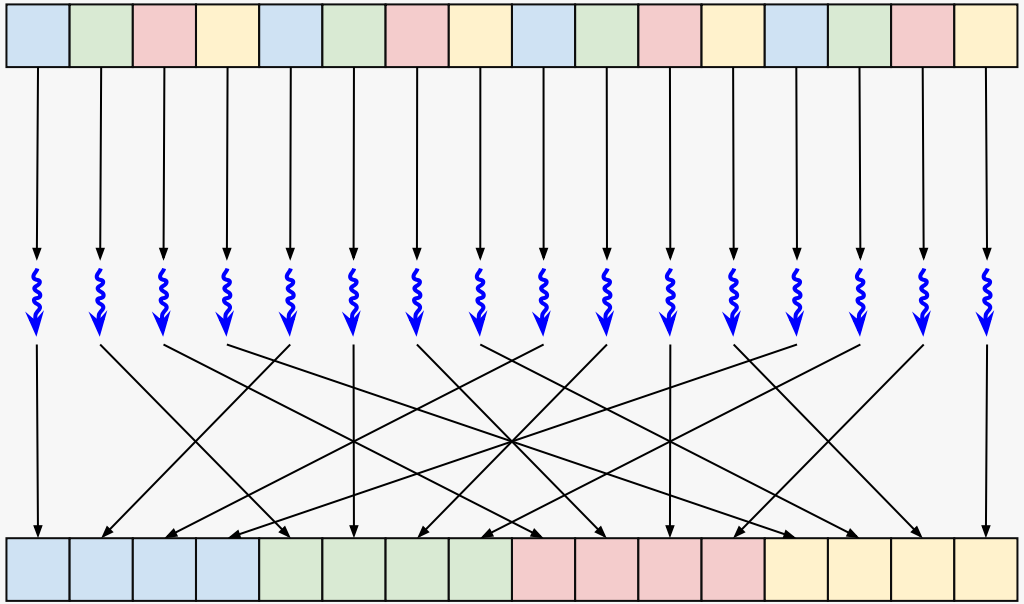
- Data is moved to the device via explicit copy APIs

Here we're using CUDA as an example

```
float *h_a = { … }, d_a;
cudaMalloc((void **)&d_a, size);
cudaMemcpy(d_a, h_a, size,
  cudaMemcpyHostToDevice);
vec_add<<<64, 64>>>(a, b, c);
cudaMemcpy(d_a, h_a, size,
  cudaMemcpyDeviceToHost);
```

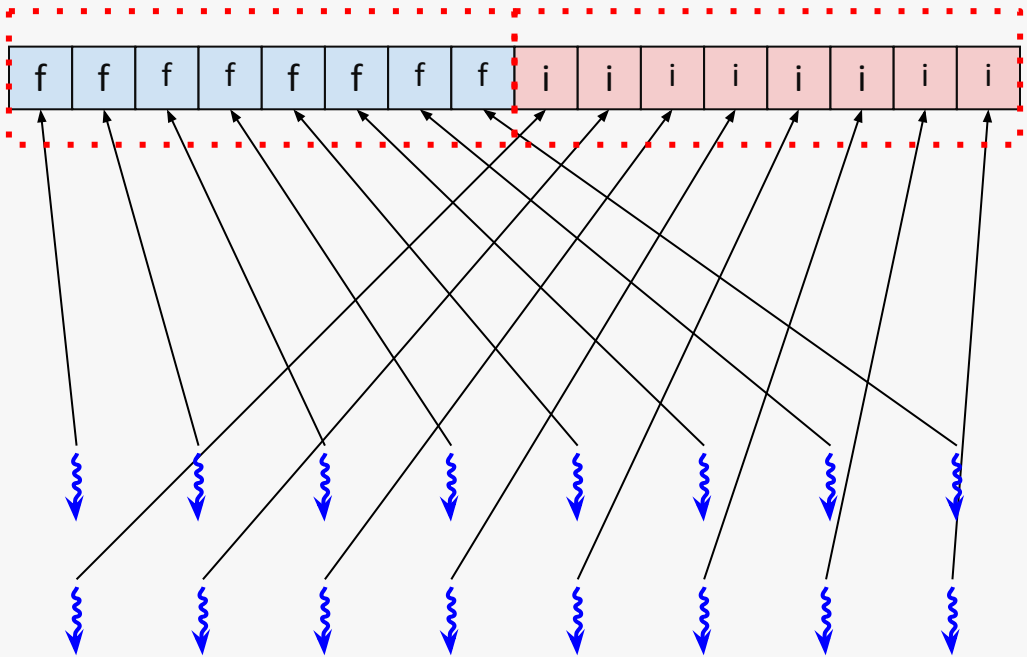
Row-major vs column-major

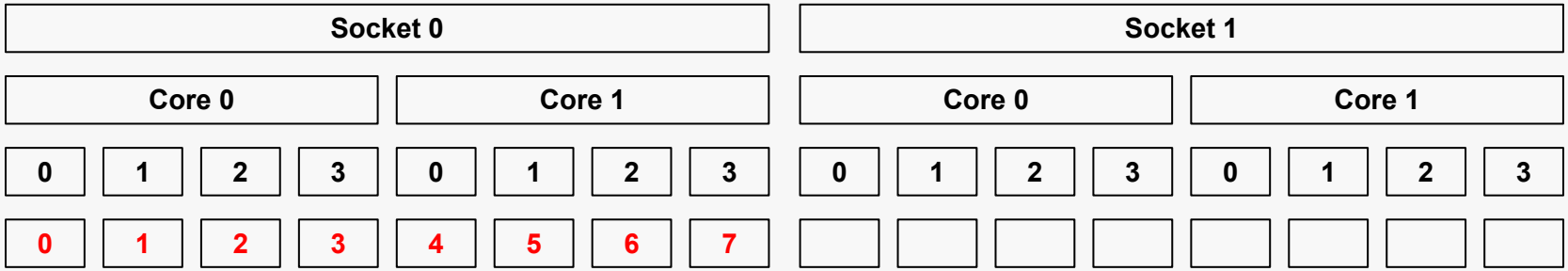
```
int x = globalId[0];
int y = globalId[1];
int stride = 4;

out[(x * stride) + y] =
  in[(y * stride) + x];
```



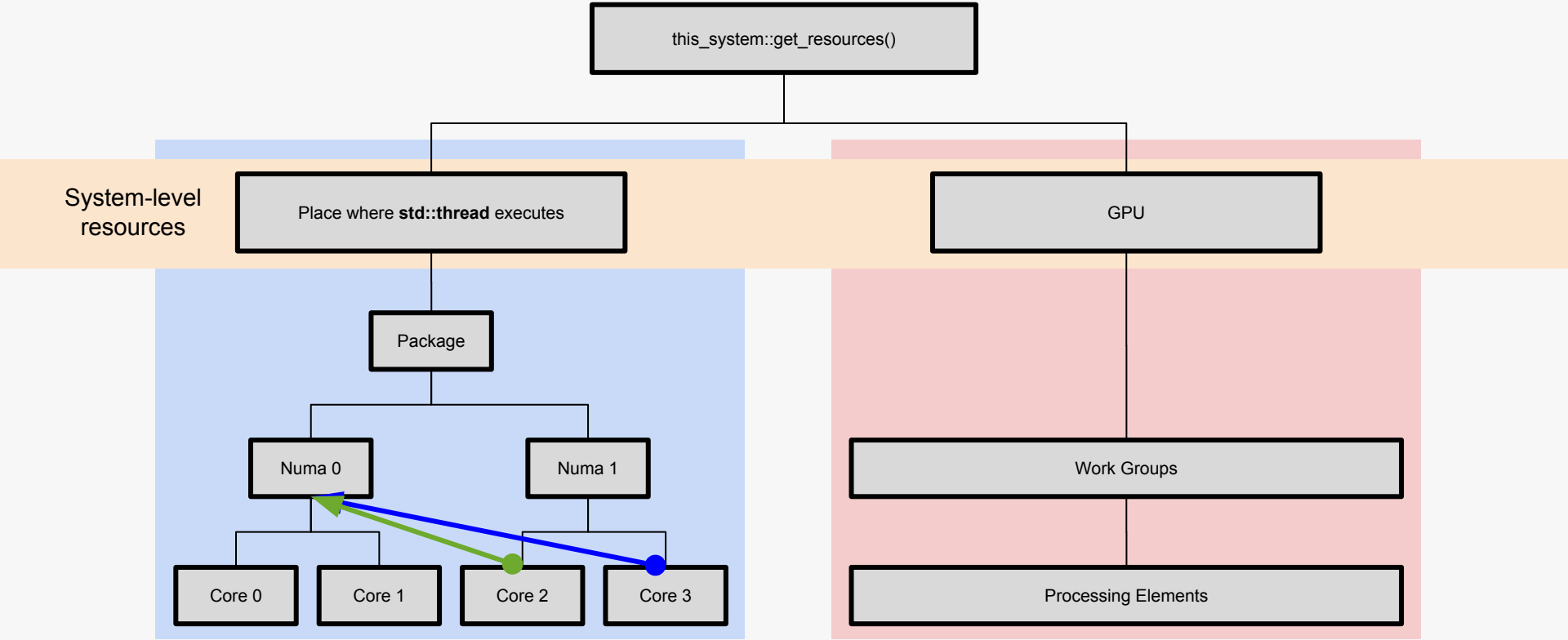
AoS vs SoA

```
struct str {
  float f[N];
  int i[N];
};

str s;

... = s.f[globalId];

... = s.i[globalId];
```






```
{
  auto exec = execution::execution_context{execRes}.executor();

  auto affExec = execution::require(exec, execution::bulk,
    execution::bulk_execution_affinity.compact);

  affExec.bulk_execute([](std::size_t i, shared s) {
    func(i);
  }, 8, sharedFactory);
}
```

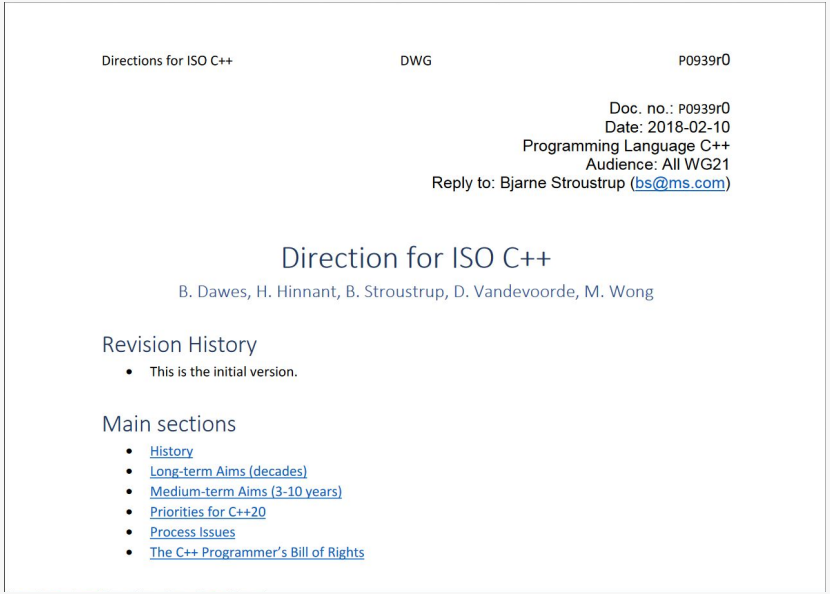


relativeLatency = affinity_query<read, latency>(core2, numa0) > affinity_query<read, latency>(core3, numa0)

Act 3

Can we do it in ISO C++ and implement it in clang

C++ Directions Group: P0939



Directions for ISO C++ DWG P0939r0

Doc. no.: P0939r0
Date: 2018-02-10
Programming Language C++
Audience: All WG21
Reply to: Bjarne Stroustrup (bs@ms.com)

Direction for ISO C++

B. Dawes, H. Hinnant, B. Stroustrup, D. Vandevoorde, M. Wong

Revision History

- This is the initial version.

Main sections

- History
- Long-term Aims (decades)
- Medium-term Aims (3-10 years)
- Priorities for C++20
- Process Issues
- The C++ Programmer's Bill of Rights

I have a big idea for a big change

Change gradually building on previous work

OR

Provide better alternative to existing feature

I have a secret to tell you

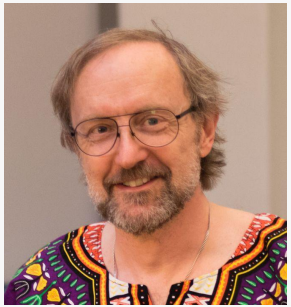
Direction Group created as response to Call to Action of

Operating Principles for C++ by Heads of Delegations

C++ in danger of losing coherency due to proposals with differ and contradictory design philosophies

The Direction Group
direction@lists.isocpp.org

We try to represent USERS: the Interest of the larger C++ community



WG 21 Direction Group

Long term Aim can not change every year

Wants lots of things for C++

But can’t have everything, and should not try

Type Safety for performance

C++ relies critically on static type safety for expressiveness, performance, and safety.

- The ideal is
  - Complete type-safety and resource-safety (no memory corruption and no leaks)

Strengthen two pillars

Better support for modern hardware (e.g., concurrency, GPUs, FPGAs, NUMA architectures, distributed systems, new memory systems)

More expressive, simpler, and safer abstraction mechanisms (without added overhead)

C++ domains examples

Safety and security

Simplification

Interoperability

Support for demanding applications in important application areas, such as medical, finance, automotive, and games

Embedded systems: make C++ easier to use and more effective for large and small embedded systems,

Alternatives for error-prone and unsafe facilities

We need Better tools

But cannot mandate

Don’t increase compile time

Don’t add complexity barriers

Concrete Library to support long-term goals

Pattern matching

Exception and error returns

Static reflection

Modern networking

Modern hardware

Simple graphics and interaction

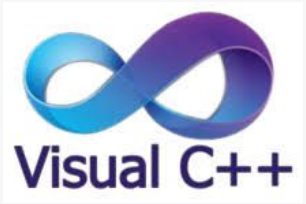
Anything from the Priorities for C++20 that didn't make C++20

Modern hardware

We need better support for modern hardware, such as executors/execution context, affinity support in C++ leading to heterogeneous/distributed computing support, ...


Ignite your applications
with XL Compilers
for C, C++, & Fortran
IBM Power
AIX
Linux


GCC


Visual C++

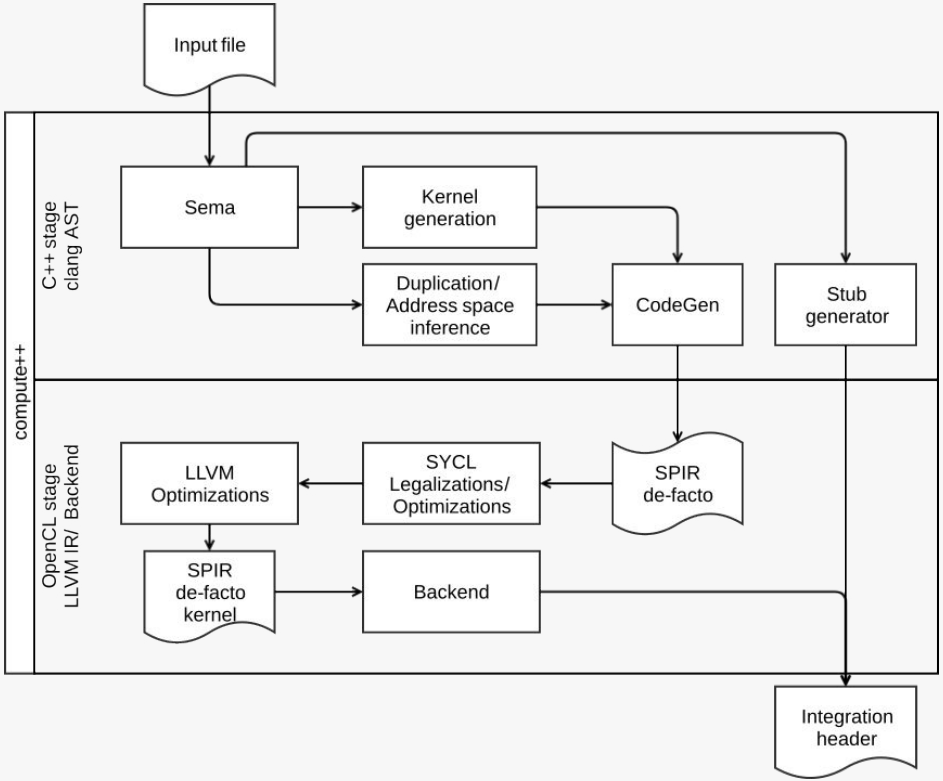

Edison
Design
Group


ComputeCpp™

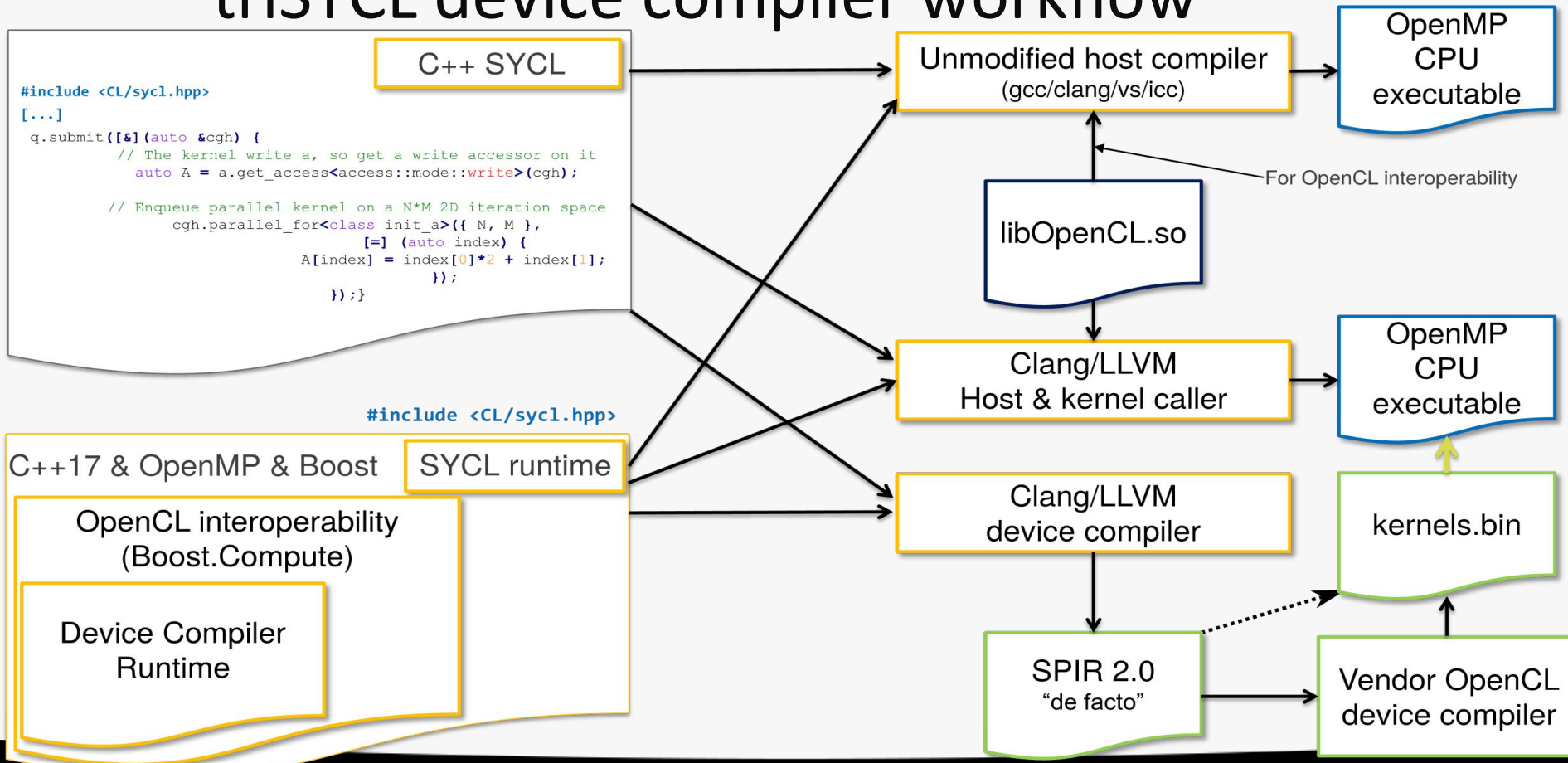

SYCL™


Gregory Stoner @angstroms · Sep 8
I saw this project today it very cool to see others building compilers on ROCm and using our solution also to target CUDA. Aksel Alpay has the hipSYCL project github.com/illuhad/hipSYCL. It is nice to see you can still access hardware specific optimizations
illuhad/hipSYCL
Implementation of SYCL 1.2.1 over AMD HIP/NVIDIA CUDA - illuhad/hipSYCL
github.com
3
5
19

Example: Codeplay compute++ flow

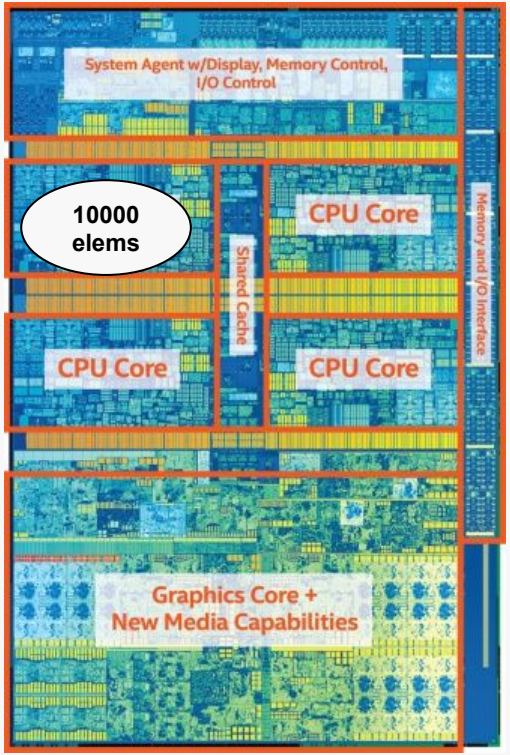


triSYCL device compiler workflow



Oh, and one more thing

What can I do with a Parallel For Each?

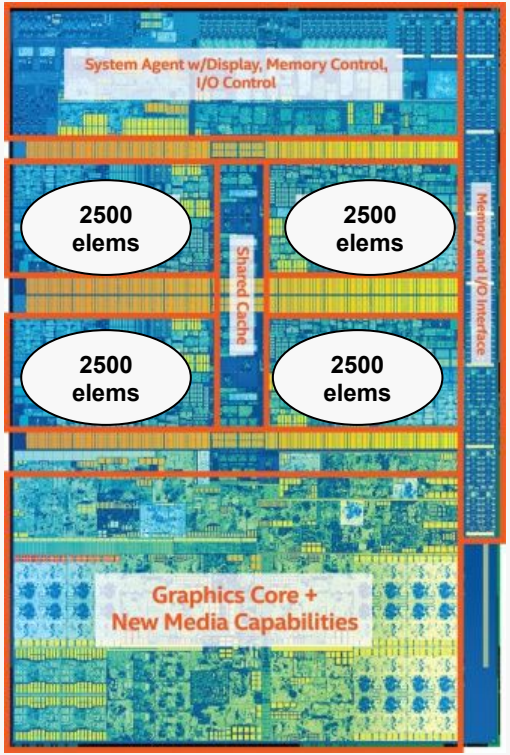


Intel Core i7 7th generation

```
size_t nElems = 1000u;
std::vector<float> nums(nElems);

std::fill_n(std::begin(v1), nElems, 1);

std::for_each(std::begin(v), std::end(v),
                    [=](float f) { f * f + f });
```

Traditional for each uses only one core, rest of the die is unutilized!

What can I do with a Parallel For Each?



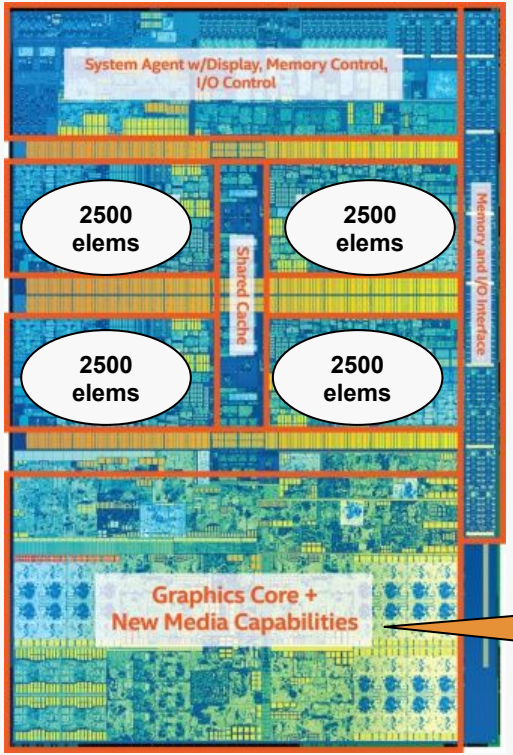
Intel Core i7 7th generation

```
size_t nElems = 1000u;
std::vector<float> nums(nElems);

std::fill_n(std::execution_policy::par,
            std::begin(v1), nElems, 1);

std::for_each(std::execution_policy::par,
              std::begin(v), std::end(v),
              [=](float f) { f * f + f });
```

Workload is distributed across cores!

(mileage may vary, implementation-specific behaviour)

What can I do with a Parallel For Each?



Intel Core i7 7th generation

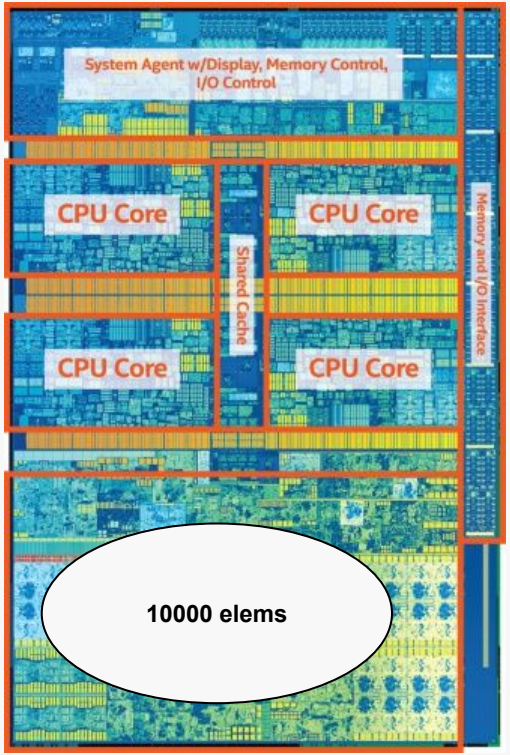
```
size_t nElems = 1000u;
std::vector<float> nums(nElems);

std::fill_n(std::execution_policy::par,
            std::begin(v1), nElems, 1);

std::for_each(std::execution_policy::par,
              std::begin(v), std::end(v),
              [=](float f) { f * f + f });
```

What about this part?

Workload is distributed across cores!

(mileage may vary, implementation-specific behaviour)

What can I do with a Parallel For Each?



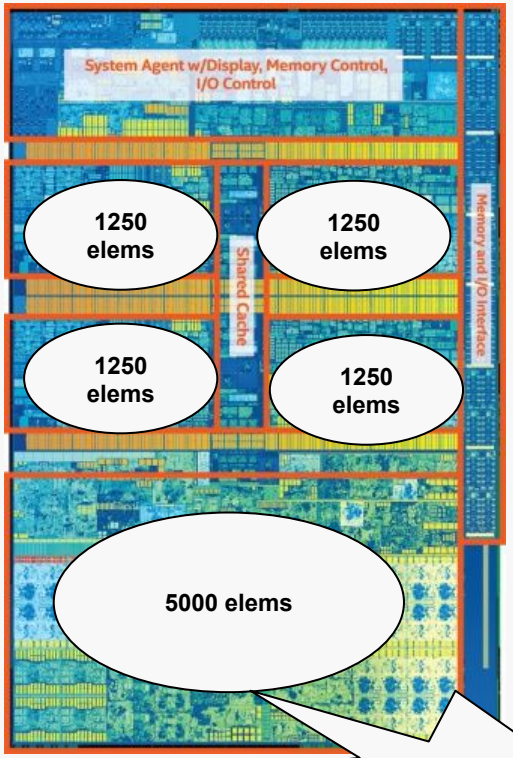
Intel Core i7 7th generation

```
size_t nElems = 1000u;
std::vector<float> nums(nElems);

std::fill_n(sycl_policy,
            std::begin(v1), nElems, 1);

std::for_each(sycl_named_policy
                <class KernelName>,
              std::begin(v), std::end(v),
              [=](float f) { f * f + f });
```

Workload is distributed on the GPU cores

(mileage may vary, implementation-specific behaviour)

What can I do with a Parallel For Each?



Intel Core i7 7th...

```
size_t nElems = 1000u;
std::vector<float> nums(nElems);

std::fill_n(sycl_heter_policy(cpu, gpu, 0.5),
            std::begin(v1), nElems, 1);

std::for_each(sycl_heter_policy<class kName>
              (cpu, gpu, 0.5),
              std::begin(v), std::end(v),
              [=](float f) { f * f + f });
```

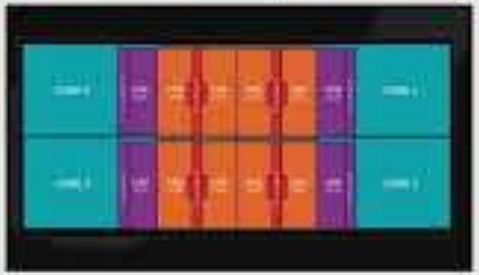
Workload is distributed on all cores!

Experimental!

(mileage may vary, implementation-specific behaviour)

cppcon | 2017

Current "Desktop" technology



AMD Ryzen (4 cores/socket)

Intel Core i7 7th generation (4 cores + GPU / socket)

codeplay

GORDON BROWN
RUYMAN REYES
MICHAEL WONG

Parallel STL for CPU and GPU The Future of Heterogeneous/ Distributed C++

CppCon.org

Demo Results - Running std::sort
(Running on Intel i7 6600 CPU & Intel HD Graphics 520)

size	2^16	2^17	2^18	*2^19*
std::seq	0.27031s	0.620068s	0.669628s	***1.48918s***
std::par	0.259486s	0.478032s	0.444422s	***1.83599s***
std::unseq	0.24258s	0.413909s	0.456224s	***1.01958s***
sycl_execution_policy	0.273724s	0.269804s	0.277747s	***0.399634s***

A unified C++ Heterogeneous programming model

Performance Portability

Across many many devices

Community Edition

Available now for free!

Visit:

computecpp.codeplay.com

- Open source SYCL projects:
  - ComputeCpp SDK - Collection of sample code and integration tools
  - SYCL ParallelSTL – SYCL based implementation of the parallel algorithms
  - VisionCpp – Compile-time embedded DSL for image processing
  - Eigen C++ Template Library – Compile-time library for machine learning

All of this and more at: http://sycl.tech

We're Hiring!
codeplay.com/careers/

codeplay
THE HETEROGENEOUS SYSTEMS EXPERTS

Thanks


@codeplaysoft


info@codeplay.com

codeplay.com