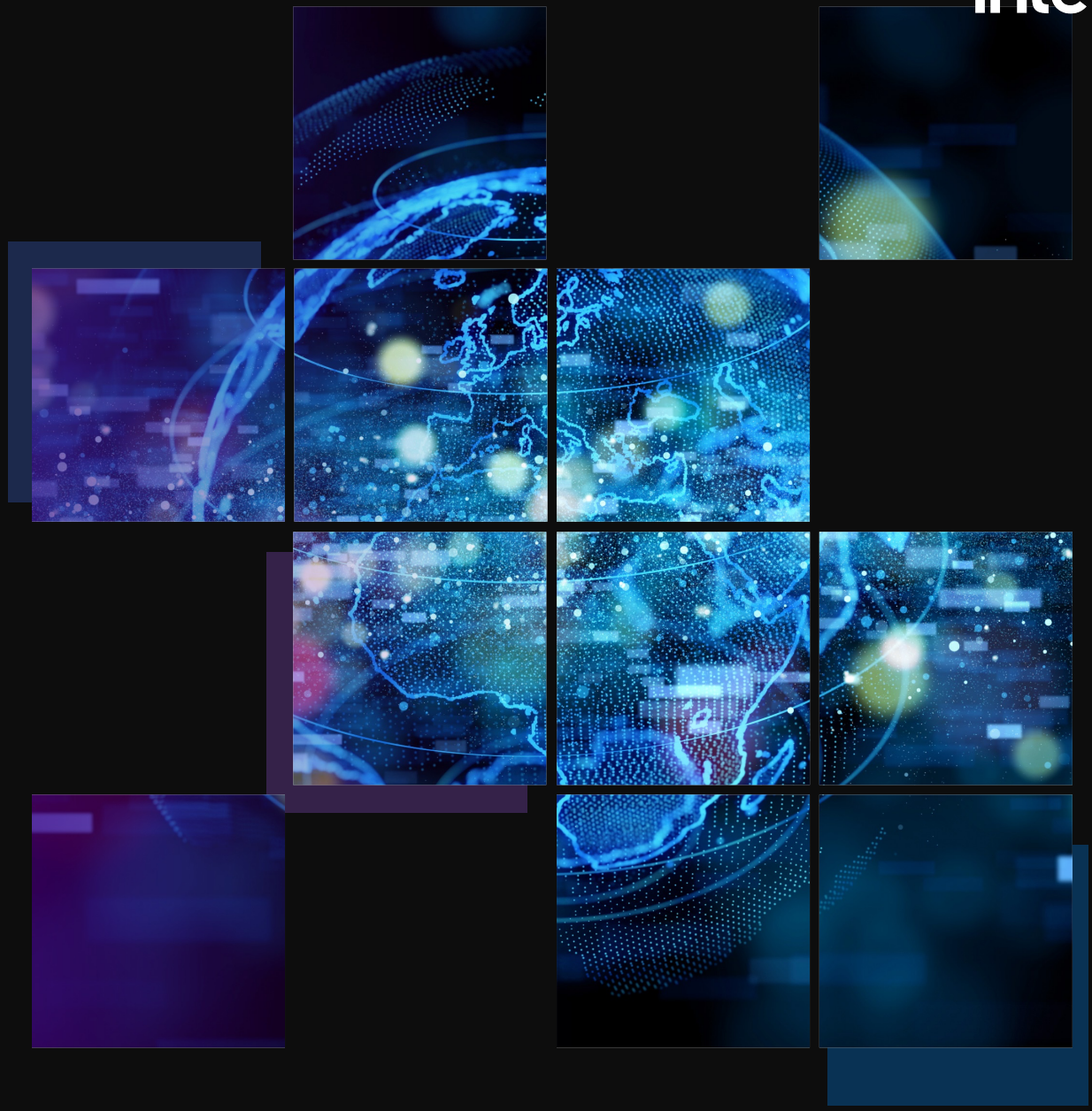
oneAPI como Plataforma de Desarrollo en FPGAs

Rafael Asenjo
oneAPI Innovator & SYCL Advisory Panel Member



Danysoft
Haciendo visible lo invisible

intel
software elite reseller

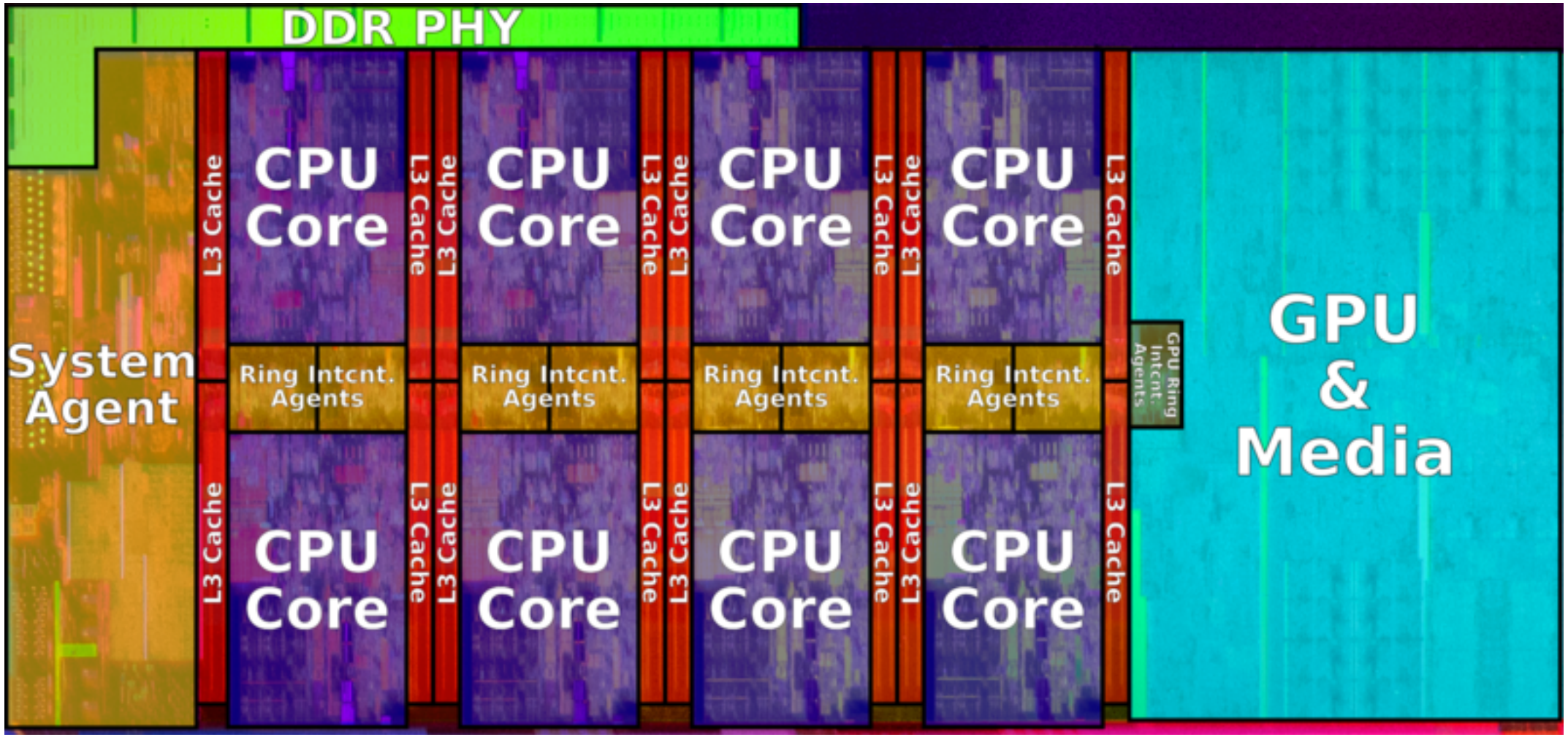
Agenda

- Introduction
- Lab: DevCloud
- Platform & Compilation
- FPGA optimizations
- Lab: DevCloud

Motivation

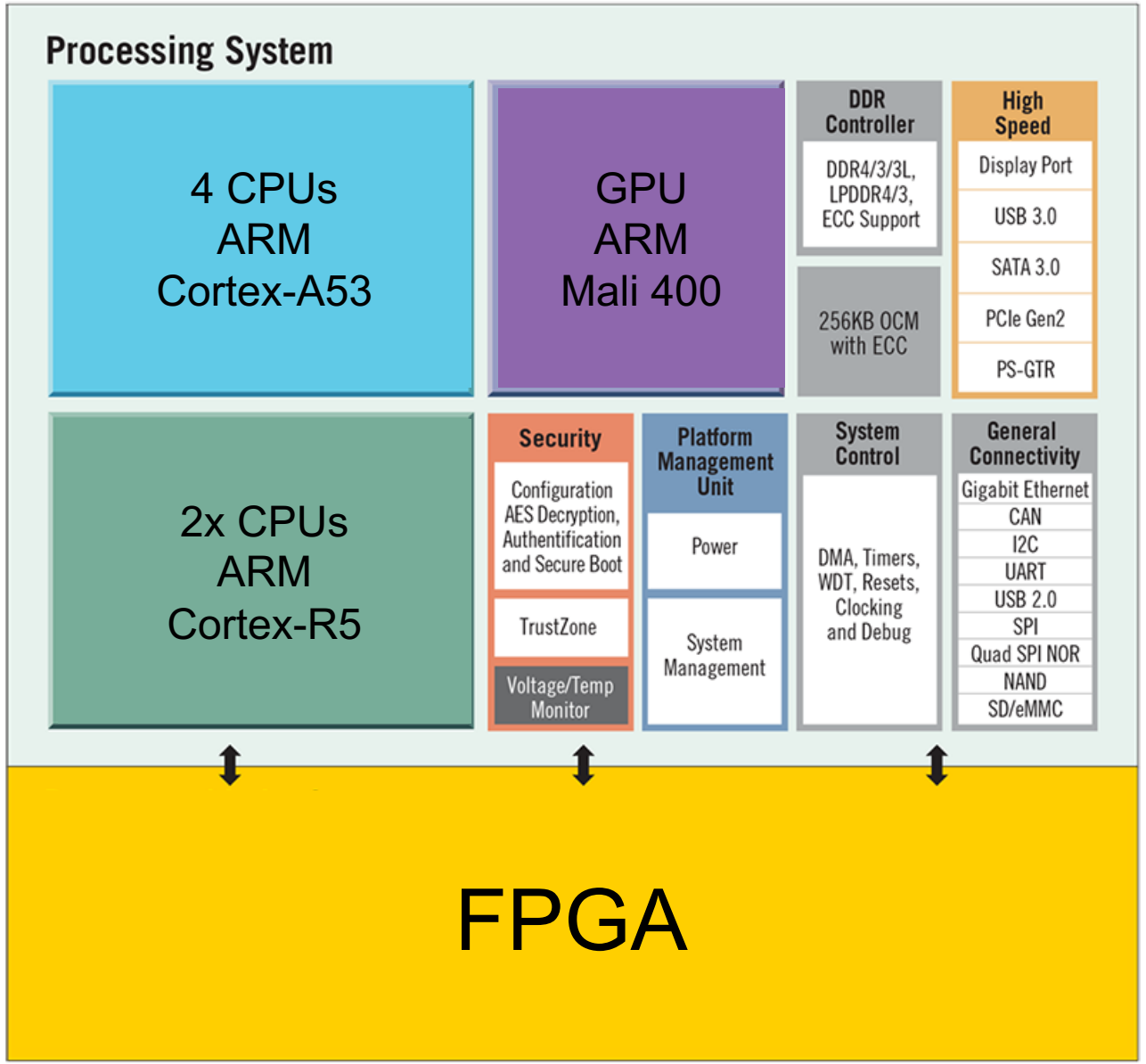
Overwhelming heterogeneity

Intel Coffee Lake



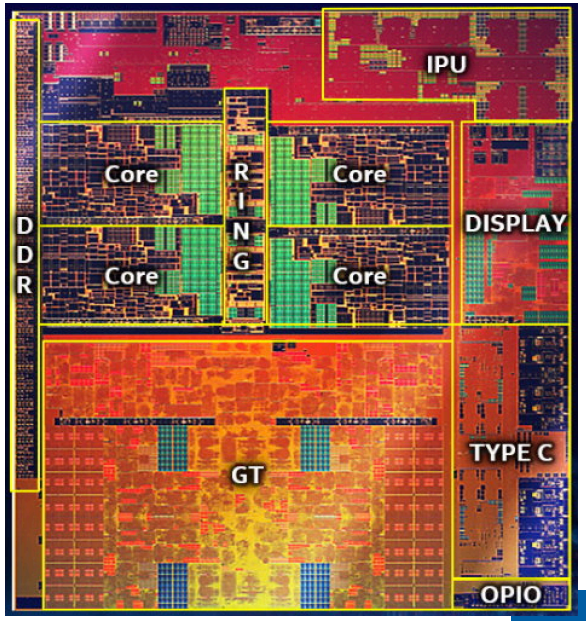
https://en.wikichip.org/wiki/intel/microarchitectures/coffee_lake

Xilinx Zynq UltraScale+ MPSoC



Note: Illustration not drawn to scale.

Heterogeneous architectures are pervasive



Motivation

- Want to **make the most** out of the CPU, the GPU and the FPGA
- Taking **productivity** into account:

  Productivity = Performance - Pain

  - High level abstractions in order to hide HW details
  - “Code once, run everywhere”
  - “Homogeneous programming of heterogeneous architectures”
- Taking **energy consumption** into account:

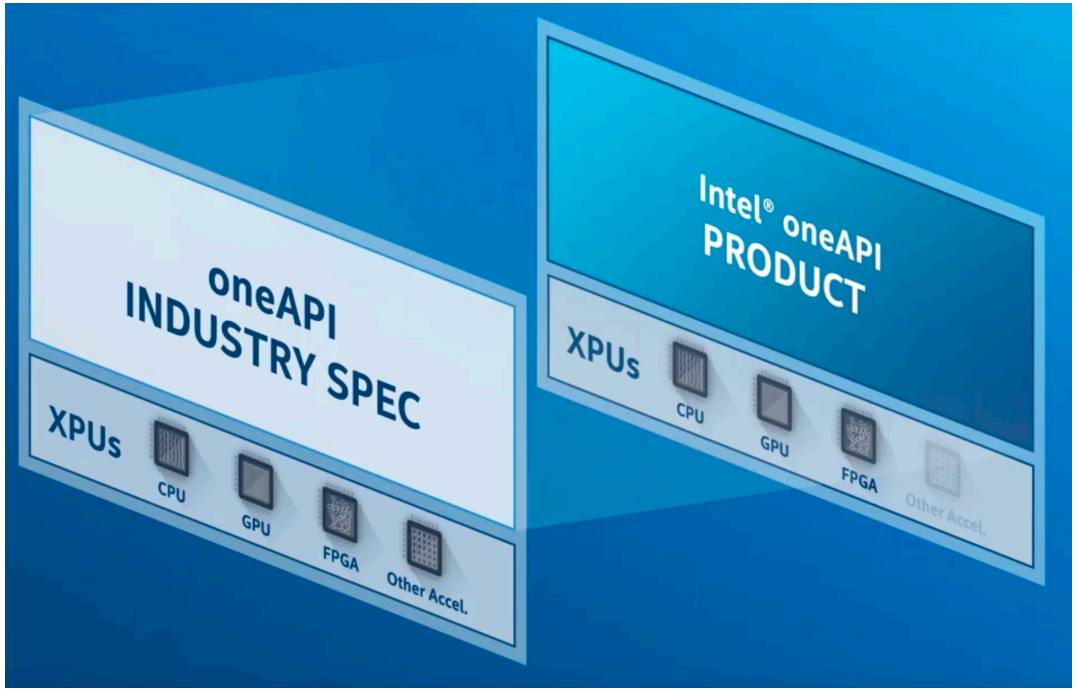
  ~~Performance ➔ Performance / Watt~~

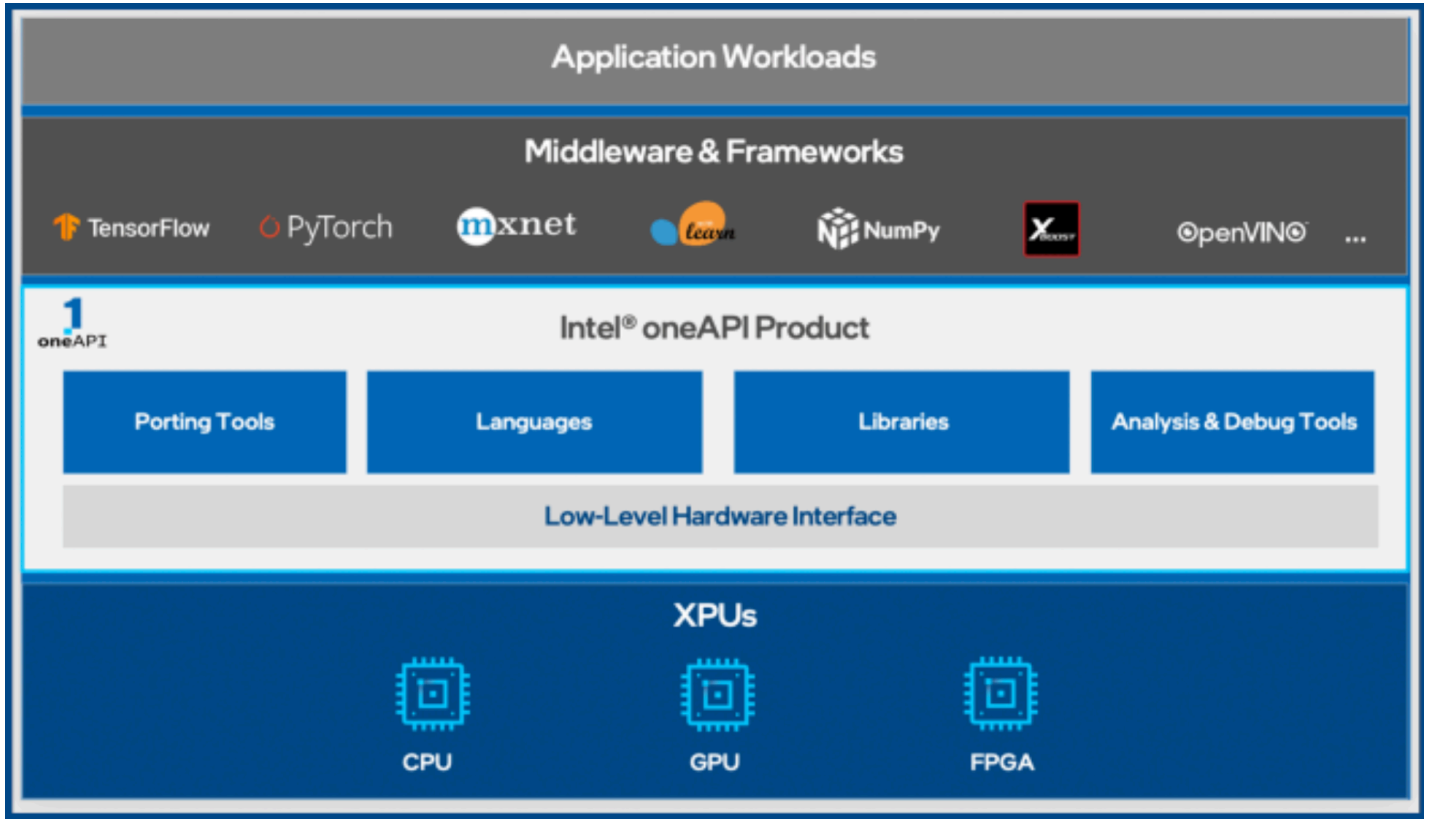
  Performance ➔ Performance / Joule

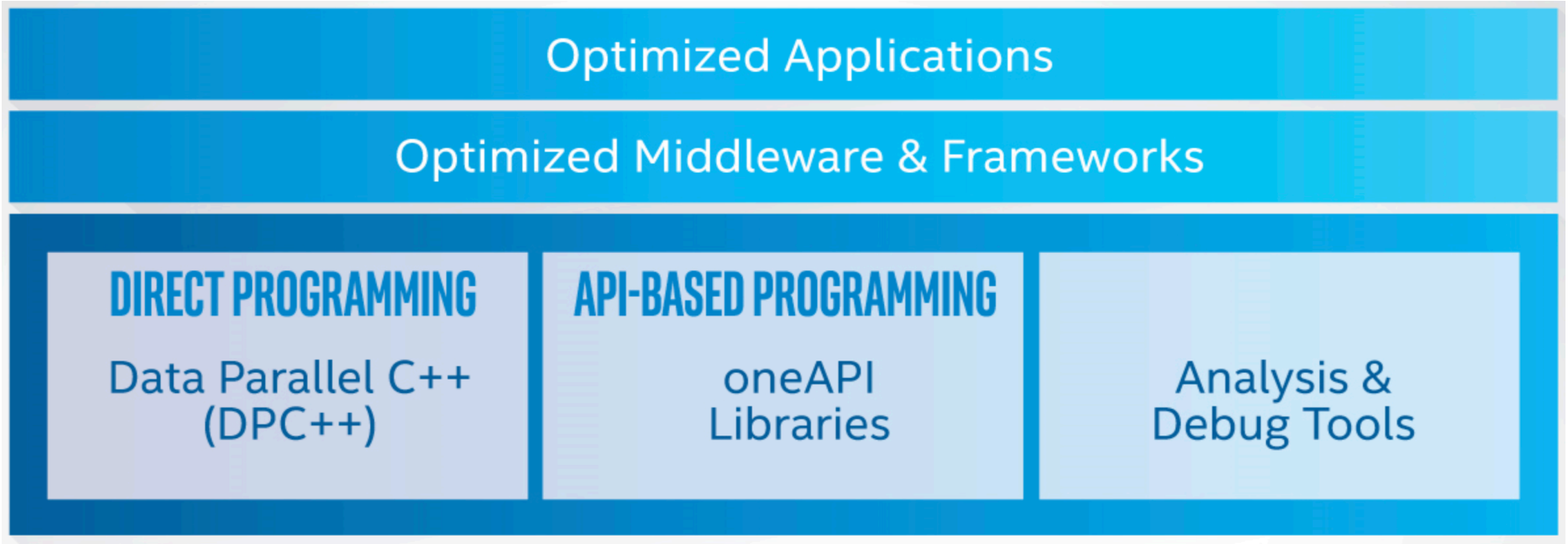
No transistor left behind

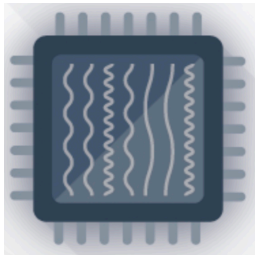
Productivity = Performance / Joule - Pain

oneAPI to the rescue

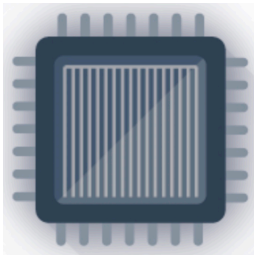




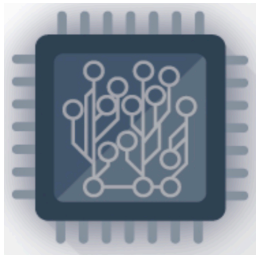




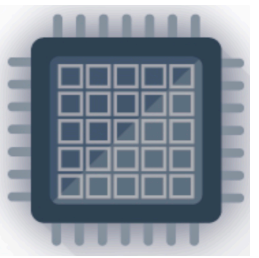
CPU



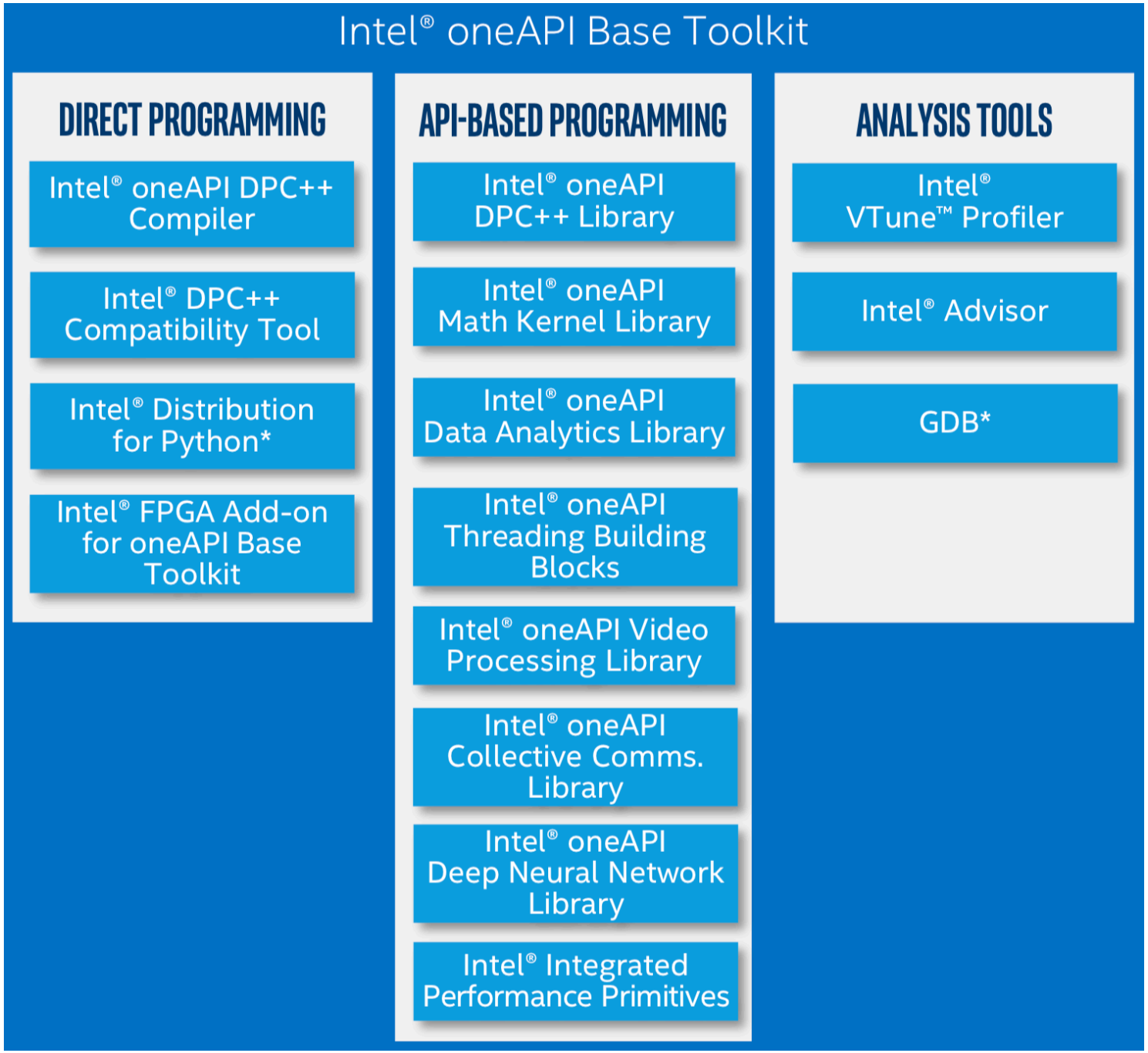
GPU



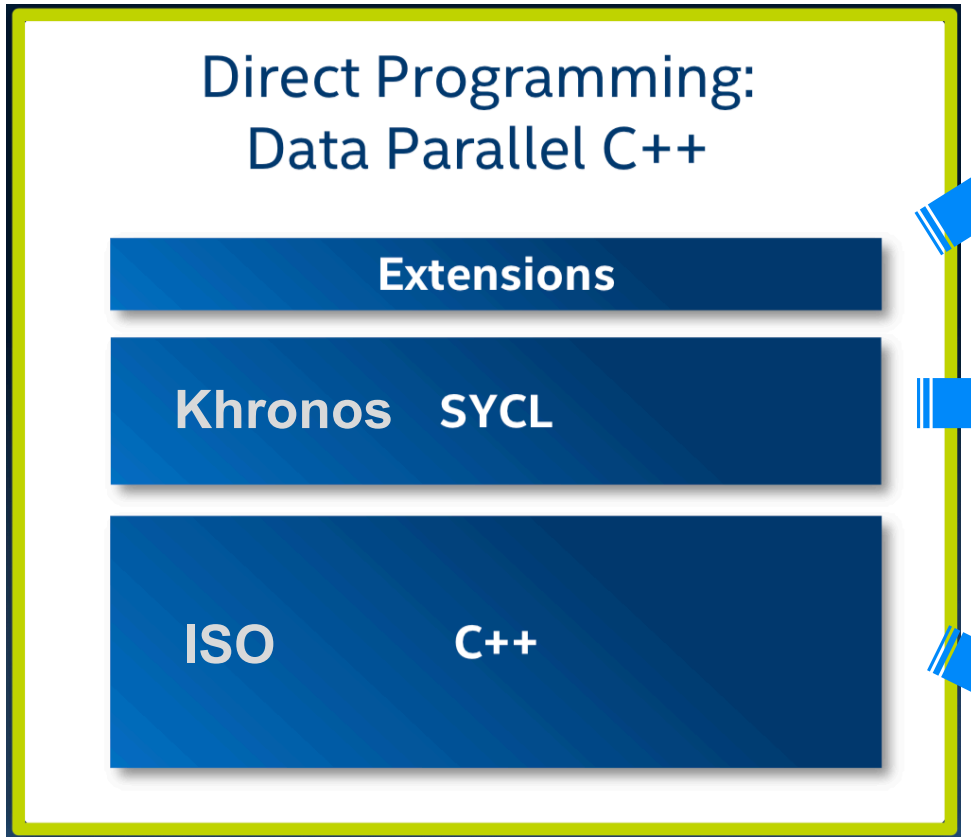
FPGA



Accel


Intel® oneAPI Base Toolkit
DIRECT PROGRAMMING
Intel® oneAPI DPC++ Compiler
Intel® DPC++ Compatibility Tool
Intel® Distribution for Python*
Intel® FPGA Add-on for oneAPI Base Toolkit
API-BASED PROGRAMMING
Intel® oneAPI DPC++ Library
Intel® oneAPI Math Kernel Library
Intel® oneAPI Data Analytics Library
Intel® oneAPI Threading Building Blocks
Intel® oneAPI Video Processing Library
Intel® oneAPI Collective Comms. Library
Intel® oneAPI Deep Neural Network Library
Intel® Integrated Performance Primitives
ANALYSIS TOOLS
Intel® VTune™ Profiler
Intel® Advisor
GDB*

Data Parallel C++: DPC++



- USM (unified shared memory)
- Sub-groups
- Ordered queues and pipes

- Cross-architecture language
- OpenCL killer
- Royalty-free industry standard

- Zero-cost abstractions
- Separation of concerns
- Composability & interoperability

Show me the code!

```
#include <CL/sycl.hpp>
#include <iostream>
constexpr int num=16;
using namespace cl::sycl;

int main() {
  auto R = range<1>{ num };
  buffer<int> A{ R };

  queue{}.submit([&](handler& h) {
    auto out =
      A.get_access<access::mode::write>(h);
    h.parallel_for(R, [=](id<1> idx) {
      out[idx] = idx[0]; }); });

  auto result =
    A.get_access<access::mode::read>();
  for (int i=0; i<num; ++i)
    std::cout << result[i] << "\n";

  return 0;
}
```

HEY!

A 1D range of 16 elements

A buffer of 16 ints

Submit work to a queue

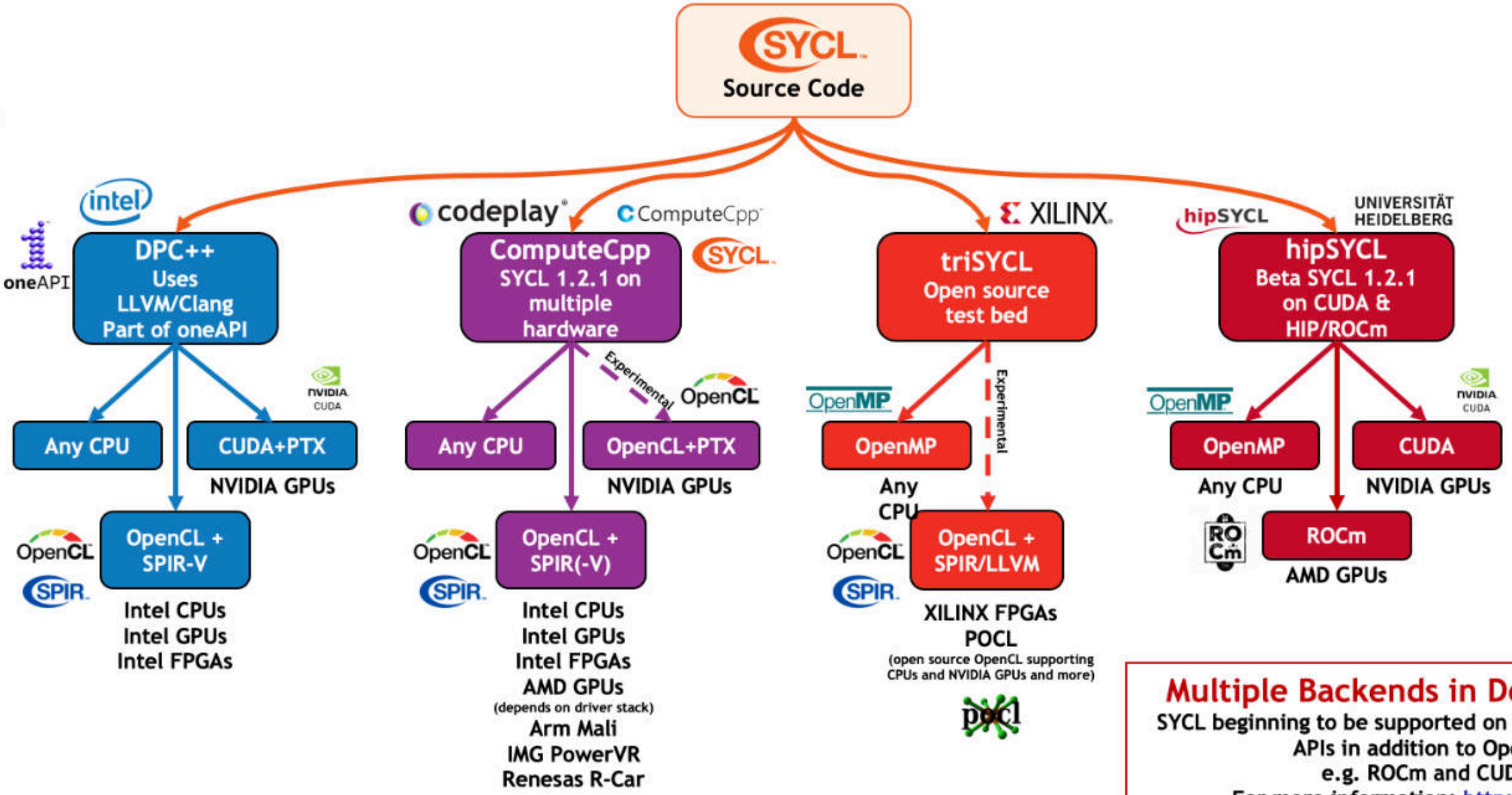
Get access to the data

Data parallel work

Get access from the host

Construct	Purpose
queue	Work targeting
buffer	Data management
parallel_for	Parallelism

```
queue{cpu_selector{}};
queue{gpu_selector{}};
queue{INTEL::fpga_selector{}};
queue{accelerator_selector{}};
queue{host_selector{}};
```



Multiple Backends in Development

SYCL beginning to be supported on multiple low-level APIs in addition to OpenCL
e.g. ROCm and CUDA
For more information: http://sycl.tech

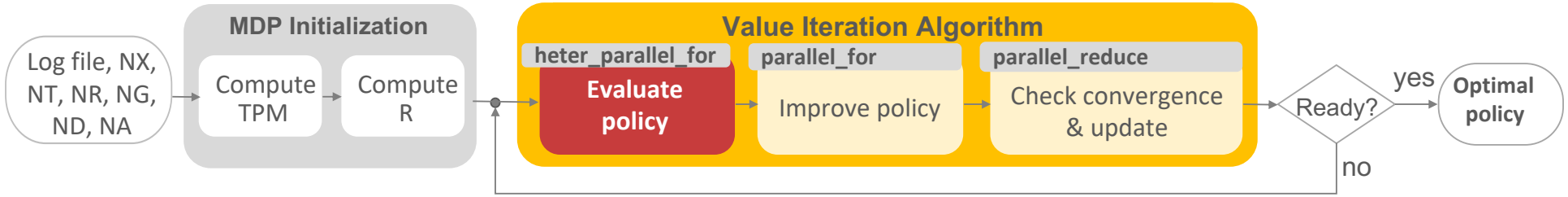
Productivity

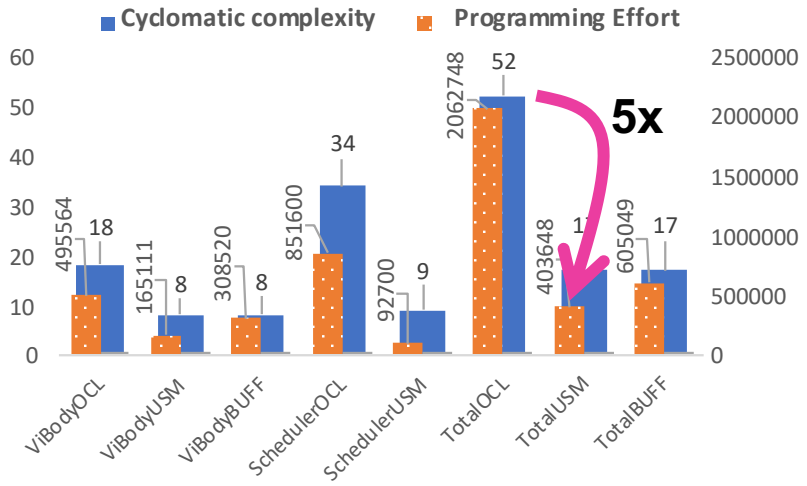
OpenCL Hello World

SYCL
Hello World

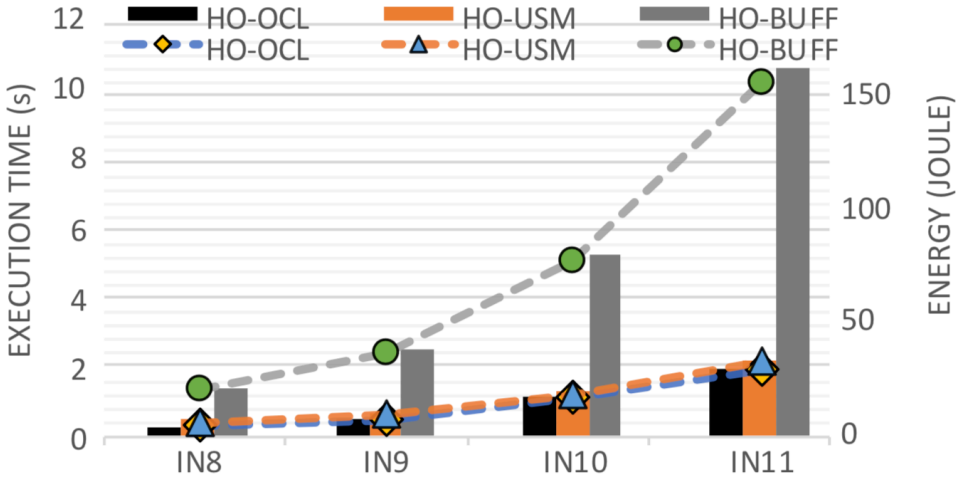
https://github.com/codeplaysoftware/syclacademy/
Lesson_Materials/Lesson-1-Introduction-to-SYCL/Lesson-1-Introduction-to-SYCL.pdf

Example: value iteration on CPU+GPU





Programmability



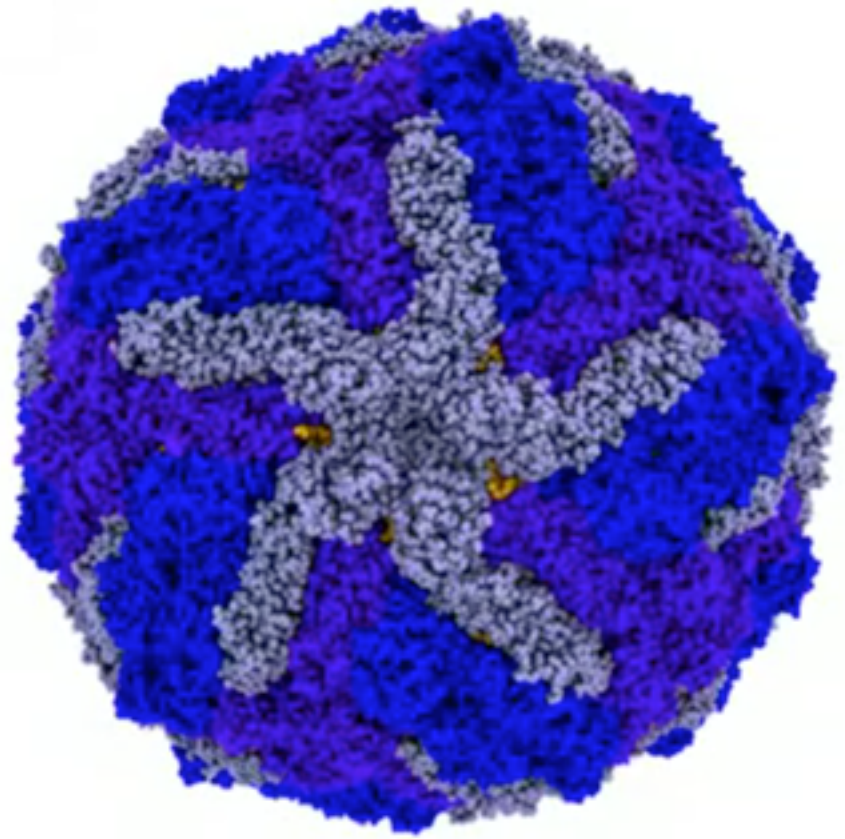
Performance

Platform	CPU	Integrated GPU	RAM	TDP (Watts)	SW Configuration
Kaby Lake	i5-8250U @1.60GHz Intel(R) quad core	UHD 620 @300MHz 24 EUs	8GB of DDR4	10 to 15W	Ubuntu 18.04 LTS OS gcc 6.2.0 C/C++ compiler (C++14) OpenCL 1.2, oneAPI 2021.1-beta03 Intel(R) NEO (Gen9) Graphics Driver

Efficiency and productivity for decision making on low-power heterogeneous CPU+GPU SoCs. J. Supercomp. Denisa-Andreea Constantinescu, A. Navarro, F. Corbera, J.A. Fernández-Madrigal, R. Asenjo, 2020

Example: GROMACS and NAMD

- GROMACS
  - One of the world's most widely used apps
  - Folding@Home (from laptops to supercomputers, including Playstations)
- NAMD
  - Popular parallel molecular dynamics application.
  - Atomic model of zika virus
  - Ported to Aurora Supercomputer
    - Including Intel GPUs



Portability vs Performance portability

DPC++ offers portability

- Run the same kernel on different devices
- Pursues the goal of “Write once, run everywhere”

But different accelerators exist for one reason

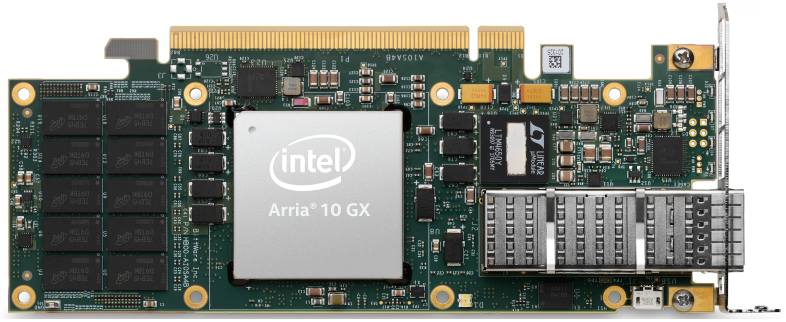
- Not to run all the code → but the piece of code most suitable for this architecture
- Not to be redundant with each other → each excel at a different kind of computation

The art is in:

- Partition an application into different kernels
- Optimize each kernel for the best suited device
- To optimize for a given architecture, you are better off **knowing that architecture**

Architectures on DevCloud

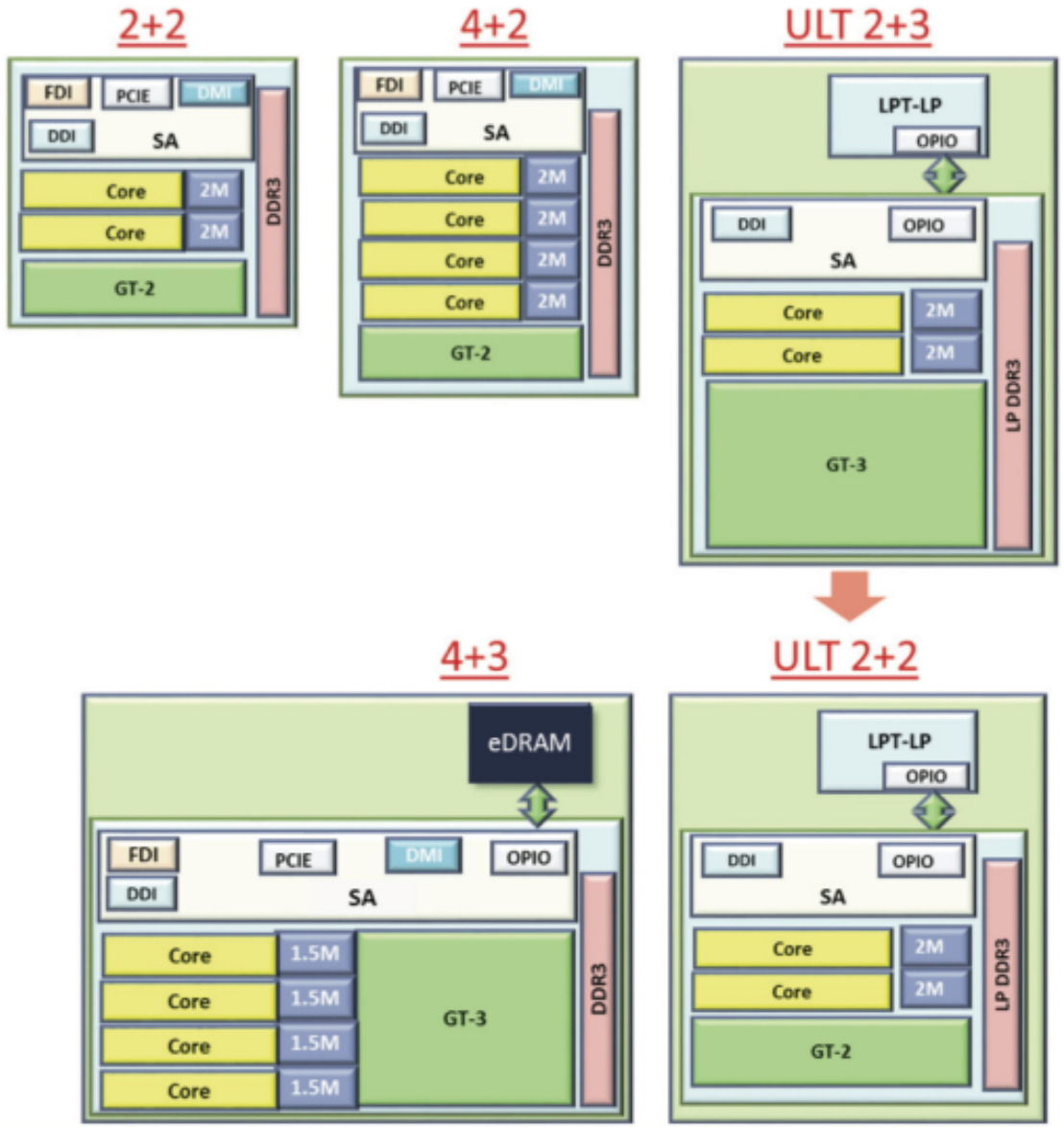
- CPUs:
  - Skylake (Xeon Gold 6128, Xeon Platinum 8153),
  - Coffee Lake (Xeon E2176G, Core i9-10920x)
- GPUs
  - UHD Graphics P630: gen9
  - Iris® Xᵉ MAX GPU: iris_xe_max
- FPGAs
  - Arria 10 PAC
  - Stratix 10 PAC



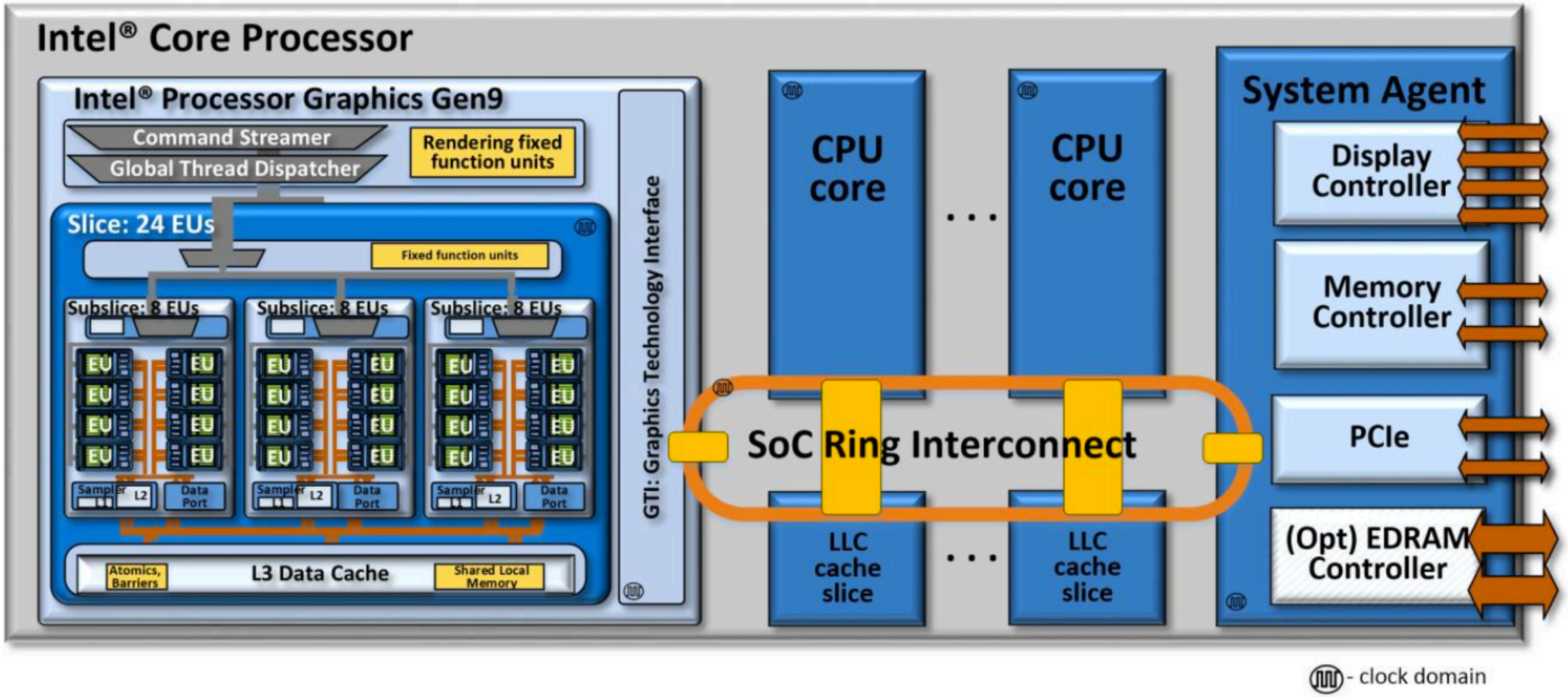
Intel Skylake

- Modular design
  - 2 or 4 cores
  - GPU
    - GT-1: 12 EU
    - GT-2: 24 EU
    - GT-3: 48 EU
    - GT-4: 72 EU

- SKU options
  - 2+2 (4.5W, 15W)
  - 4+2 (35W -- 91W)
  - 2+3 (15W, 28W,...)
  - 4+4 (65W)
  - ....

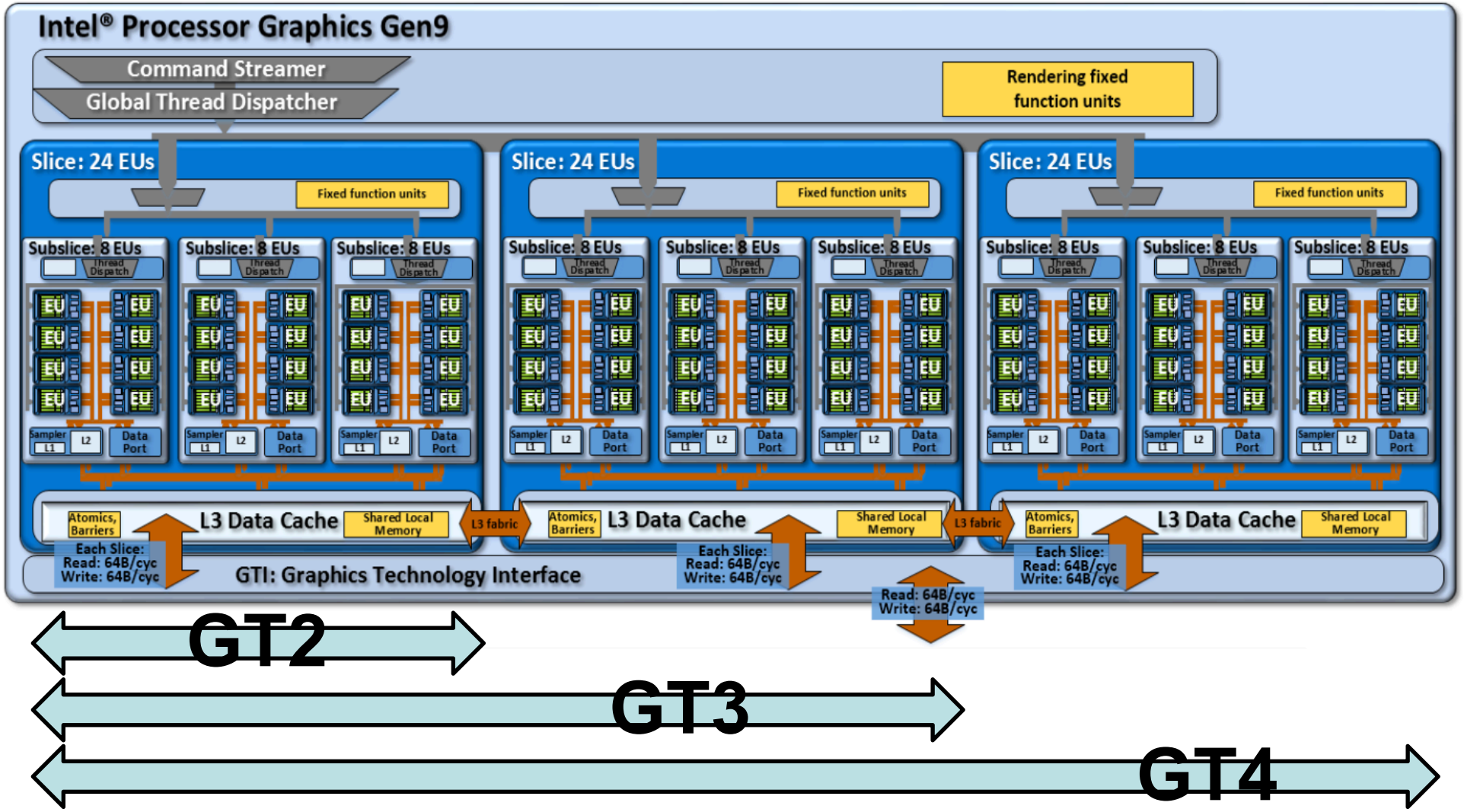


Intel Skylake



https://software.intel.com/sites/default/files/managed/c5/9a/The-Compute-Architecture-of-Intel-Processor-Graphics-Gen9-v1d0.pdf

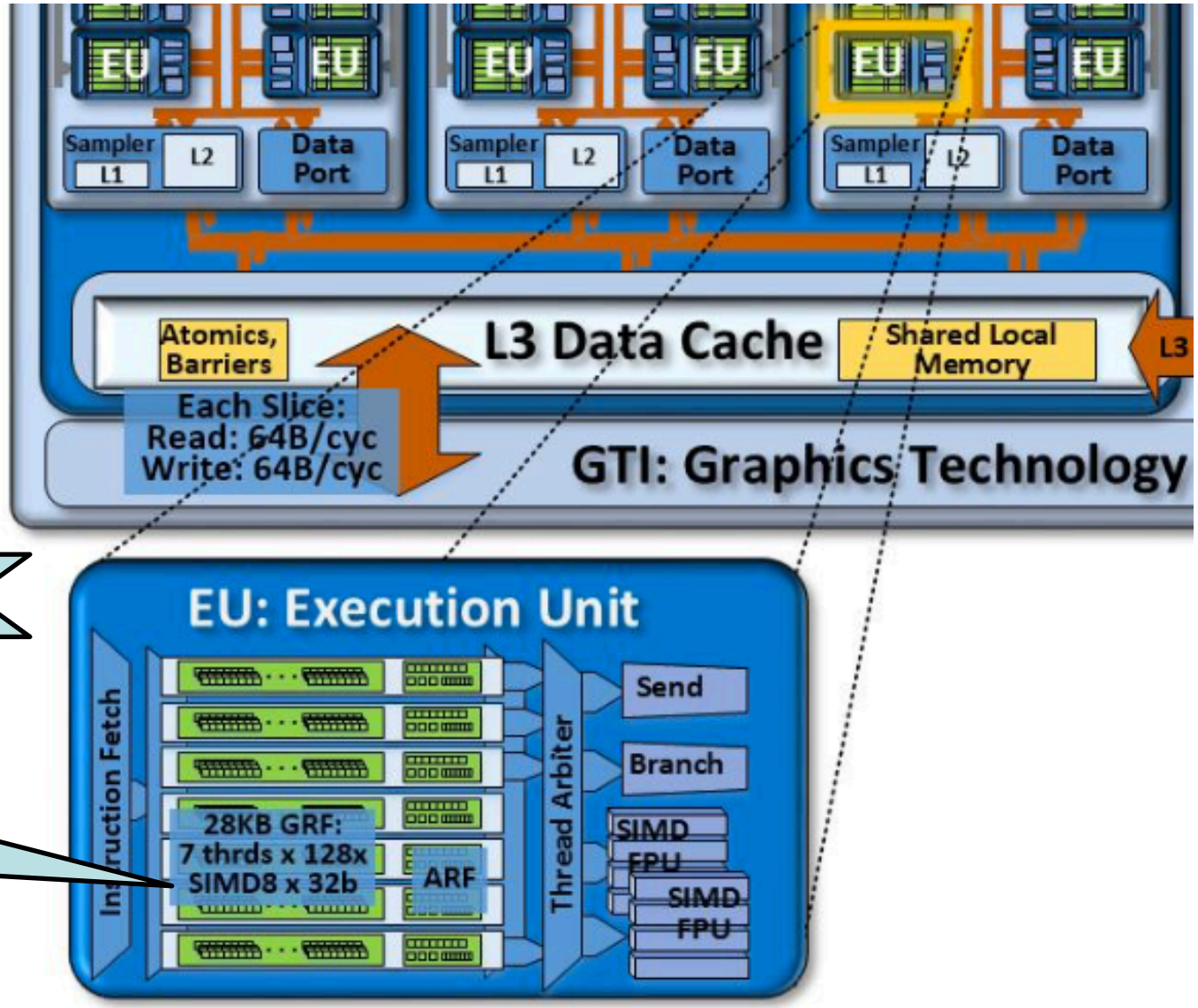
Intel Processor Graphics Gen9



Intel Processor Graphics Gen9

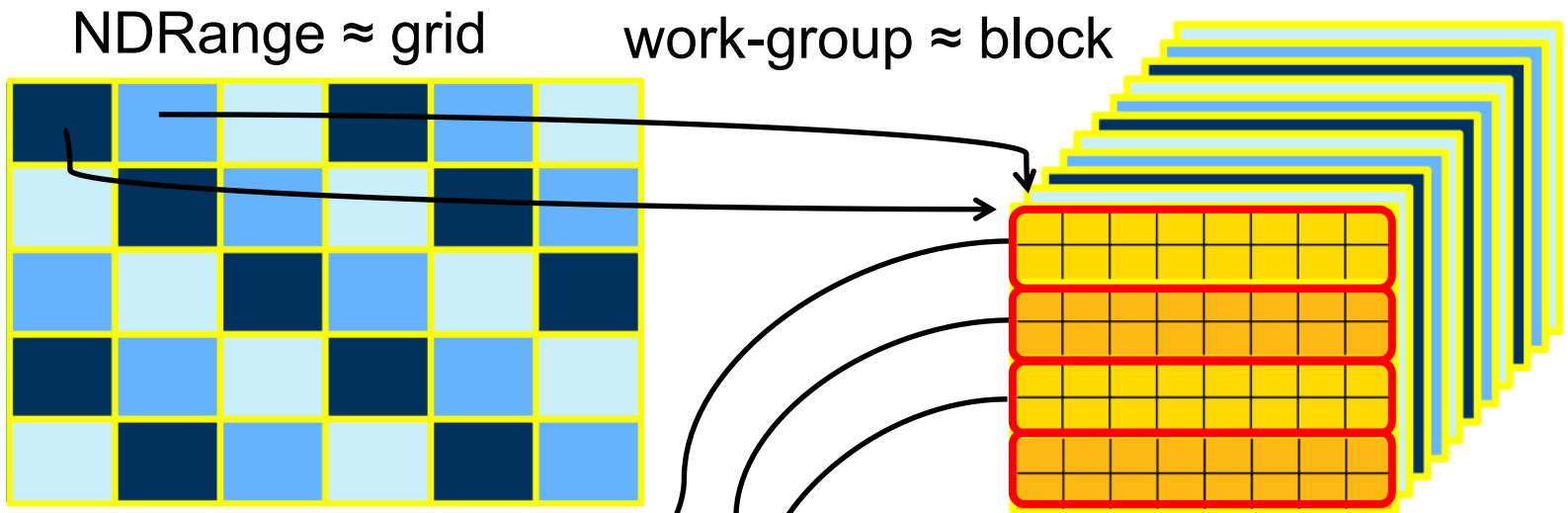
- Peak performance:
  - 72 EUs
  - 2 x 4-wide SIMD
  - 2 flop/ck (fmadd)
  - 1GHz

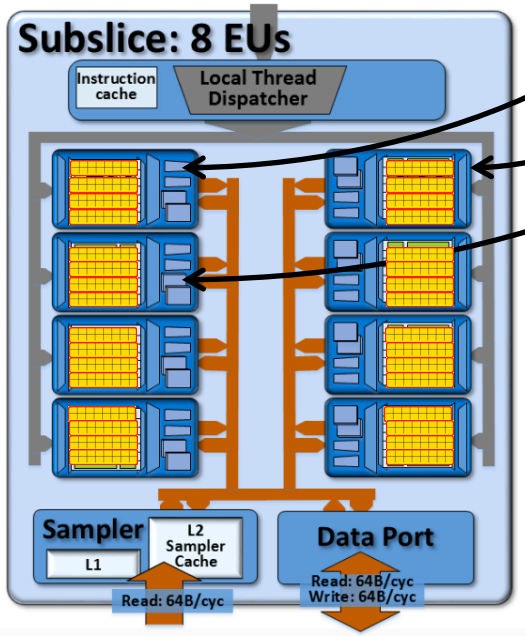
1.15 TFlops



7 in-flight wavefronts

Intel Processor Graphics Gen9





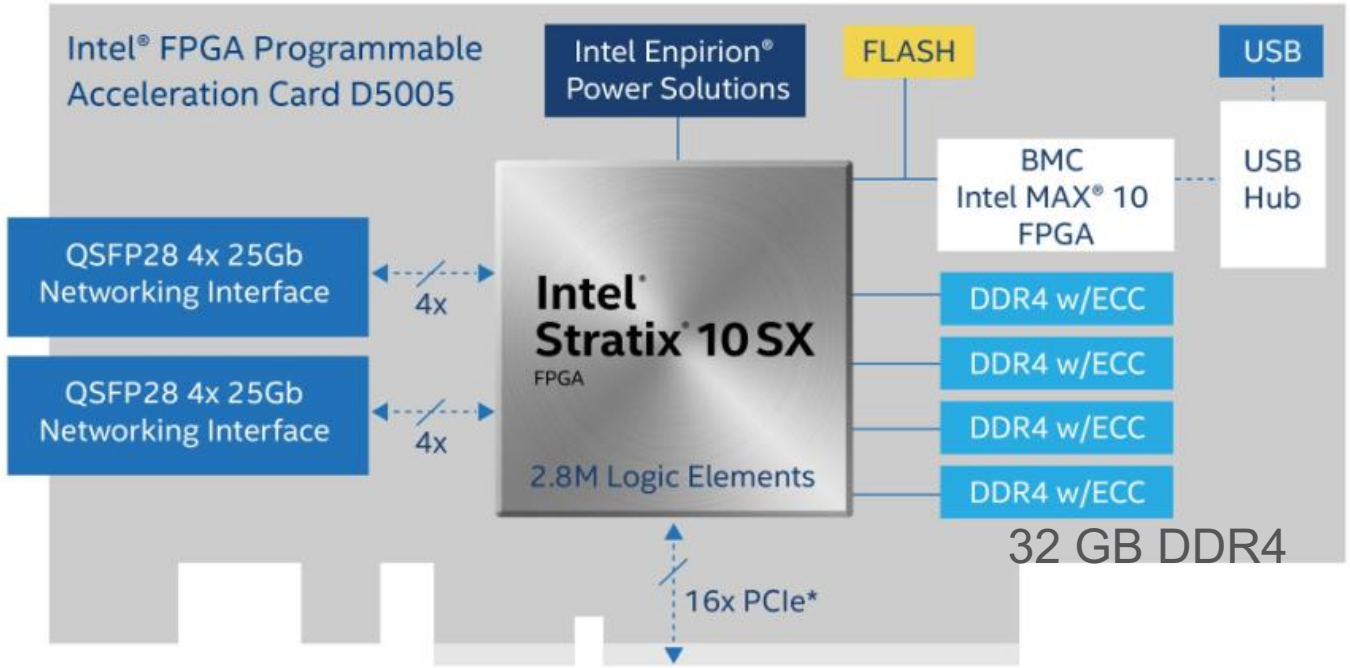
= EU-thread (SIMD16) ≈ sub-group
≈ wavefront ≈ warp:
16 work-items ≈ 16 thrds

- Each SIMD16 sub-group takes 2cks
- IMT: fine-grained multi-threading
  - Latency hiding mechanism

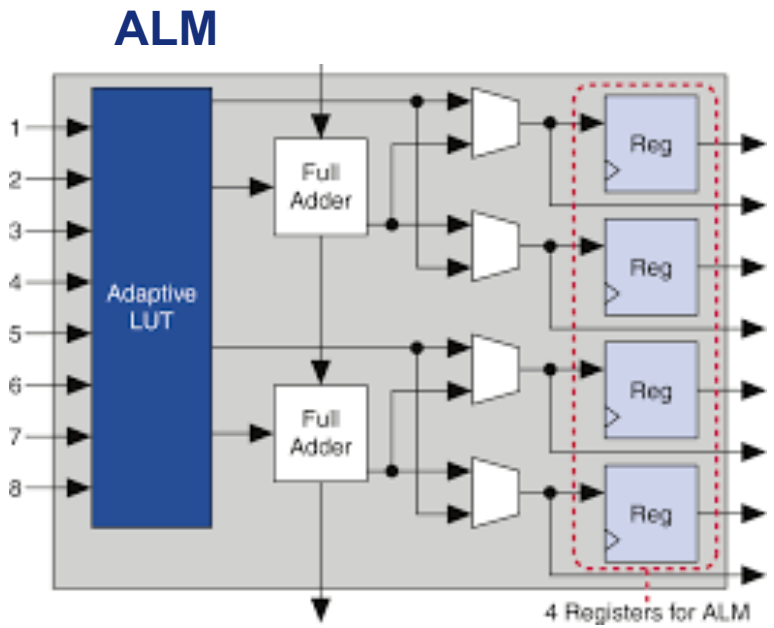
FPGA architecture

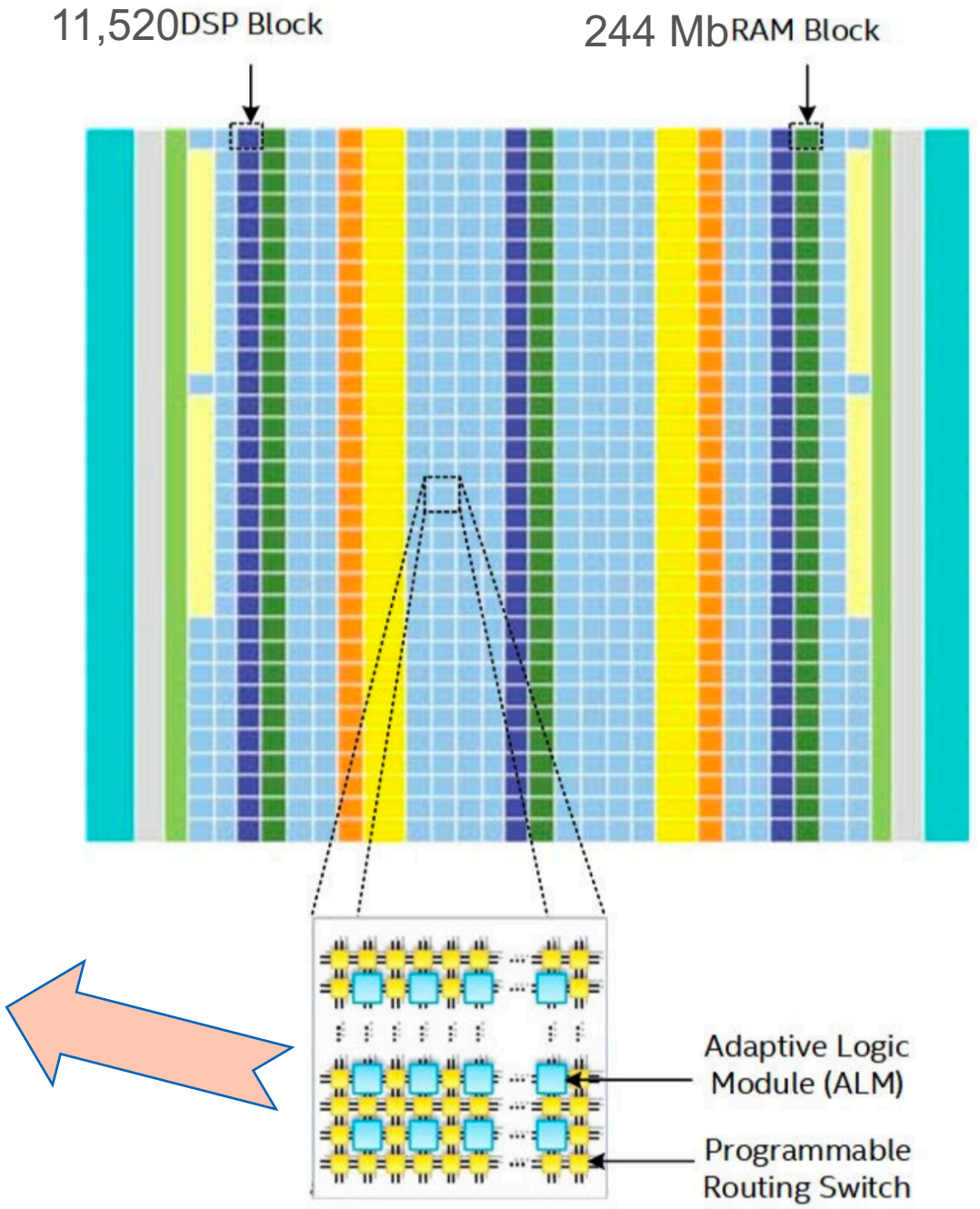
Field Programmable Gate Array

FPGA PAC: Programable Acceleration Card



FPGA architecture



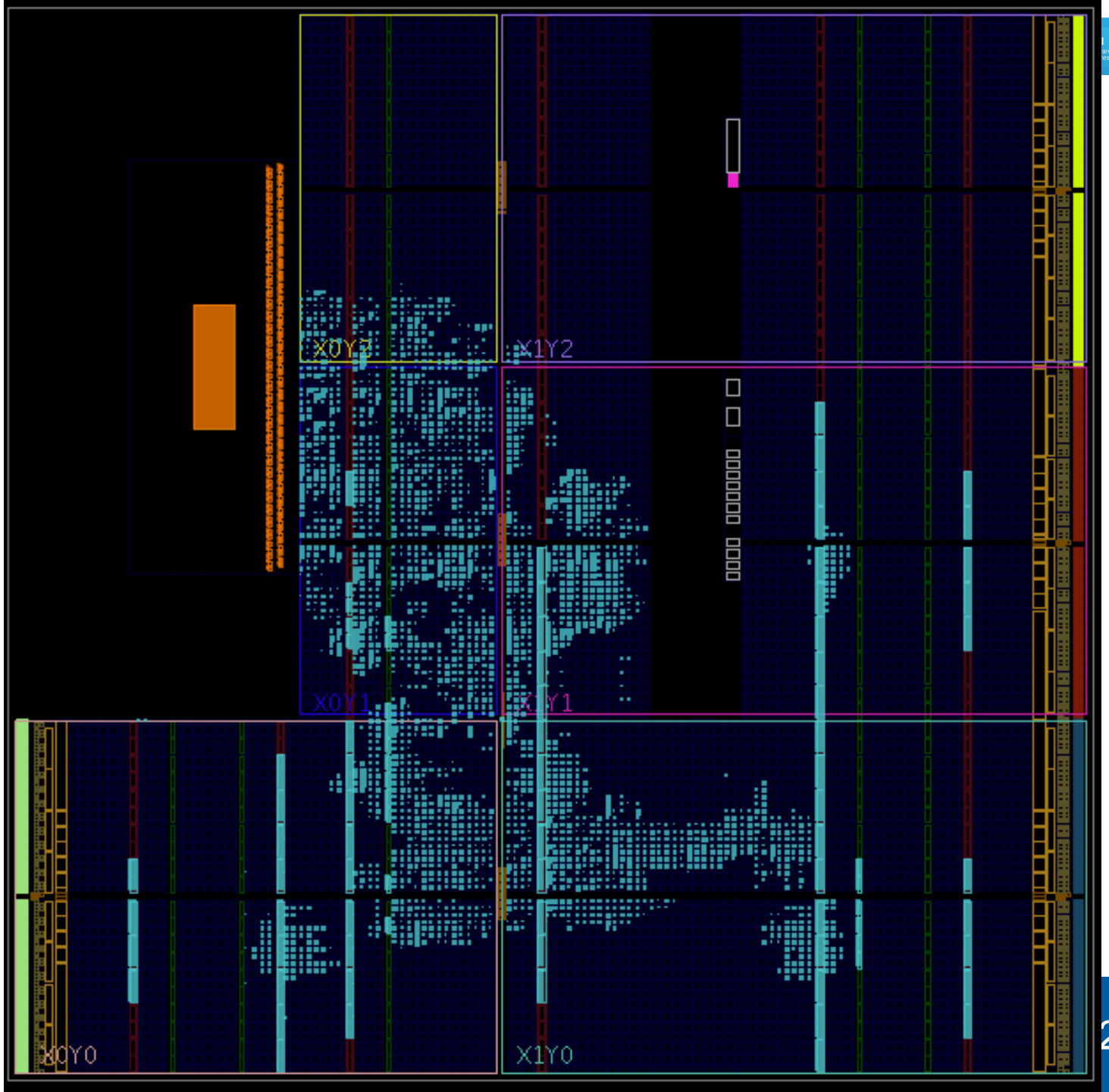


FPGA

- Programmable HW
- “Liquid silicon”
- Can “draw hardware”

Picture of a
Barnes-Hut algorithm

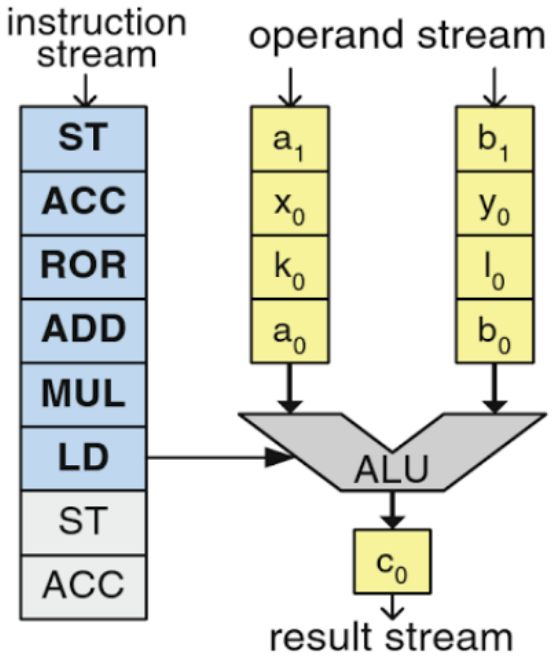
(used logic in blue)



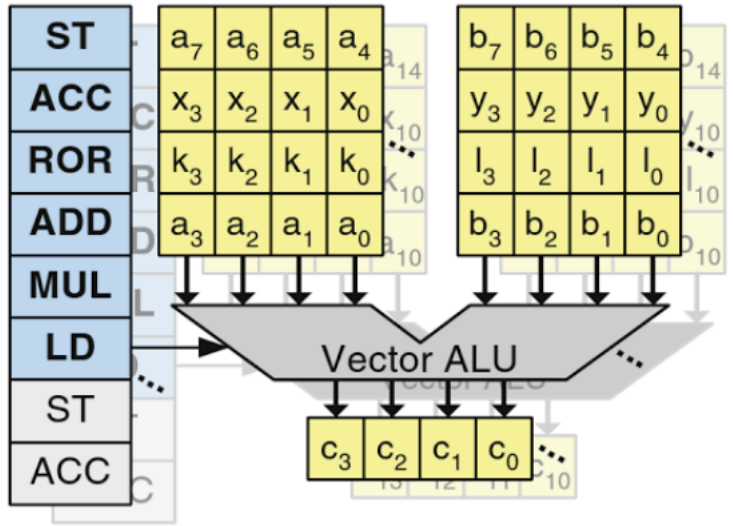
What do we “mold in this liquid silicon”?

- FPGA overlays: Map a CPU or GPU on the FPGA
  - 👍 100s of simple CPU can fit on large FPGAs
  - 👍 The CPU or GPU overlay can change/adapt
  - 👎 Resulting overlay is not as efficient as a “real” one
  - 👎 Same drawbacks as “general purpose” processors

CPU:



GPU:

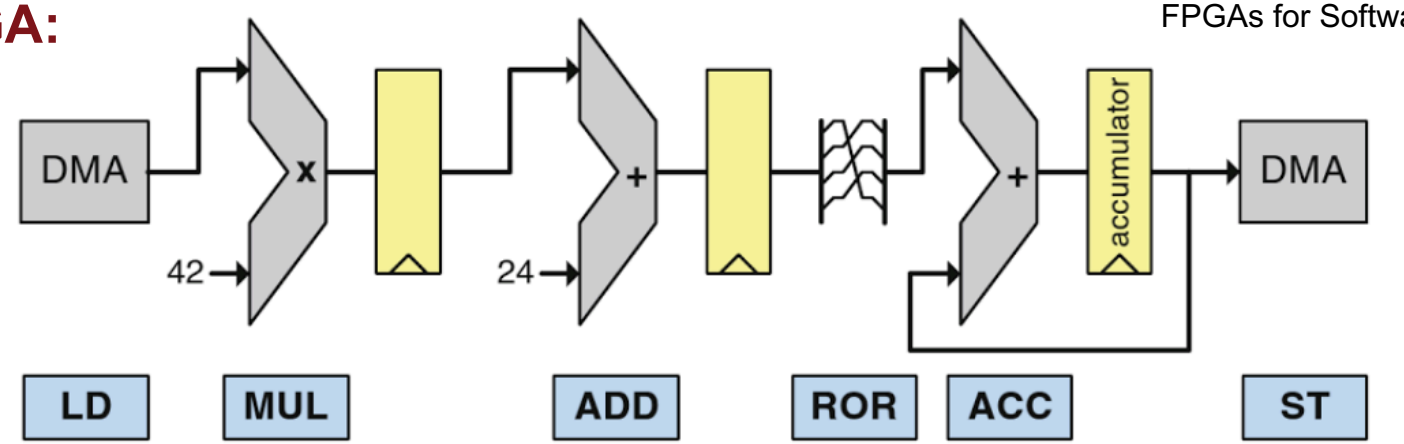


Courtesy: FPGAs for Software Programmers, Dirk Kock, Daniel Ziener

What do we “mold in this liquid silicon”?

FPGA:

FPGAs for Software Programmers, Dirk Kock, Daniel Ziener



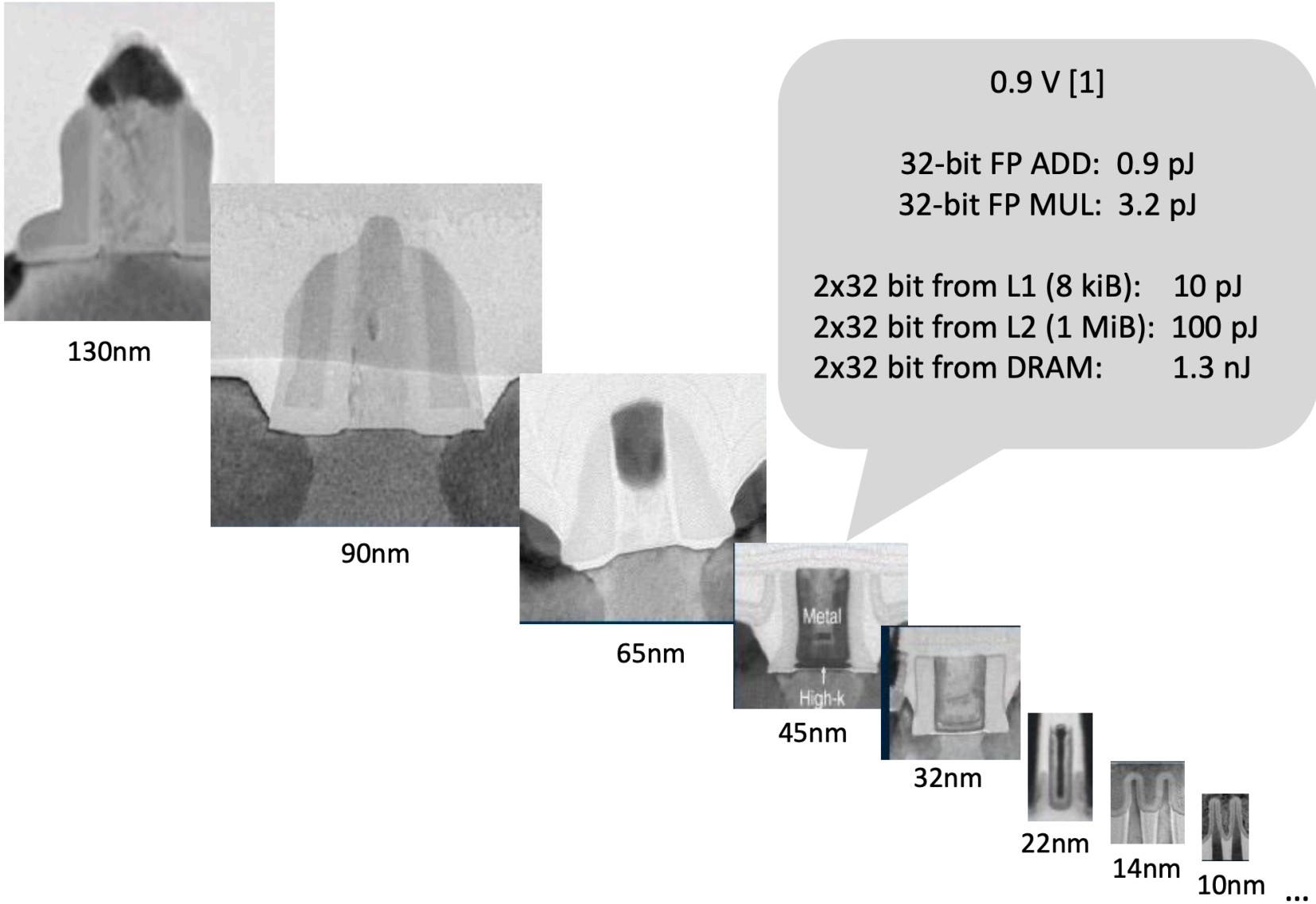
- ✓ No FETCH nor DECODE of instructions → already hardwired
- ✓ Data movement (Exec. Units ⇔ MEM) reduction → Power save
- ✓ Not constrained by a fixed ISA:
  - ✓ Bit-level parallelism; shift, bit mask, ...; variable precision arithmetic

Less frequency (hundreds of MHz)

A program may not fit (if it requires more than 100% of the available resources)

Programming effort

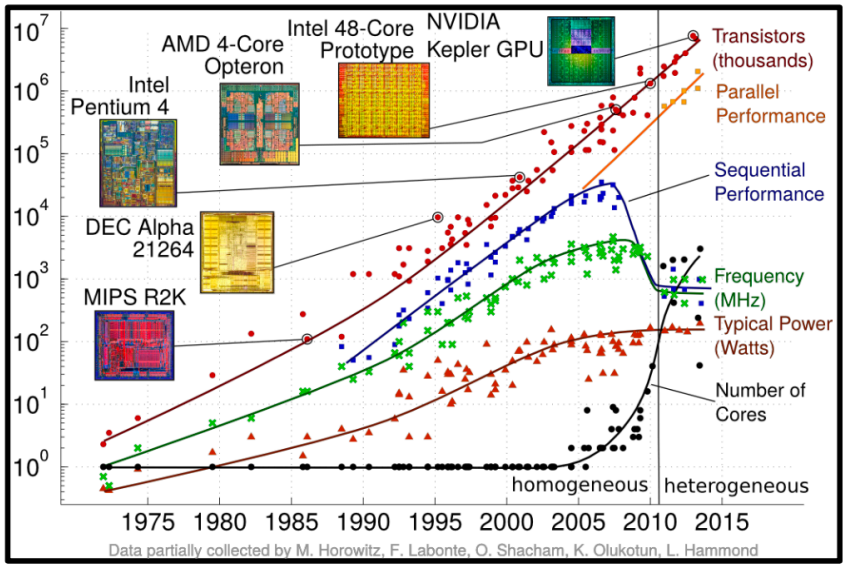
Changing hardware constraints and the physics of computing



0.9 V [1]

32-bit FP ADD: 0.9 pJ
32-bit FP MUL: 3.2 pJ

2x32 bit from L1 (8 kiB): 10 pJ
2x32 bit from L2 (1 MiB): 100 pJ
2x32 bit from DRAM: 1.3 nJ



Three L's of modern computing:
1. Spatial Locality
2. Temporal Locality
3. Control Locality

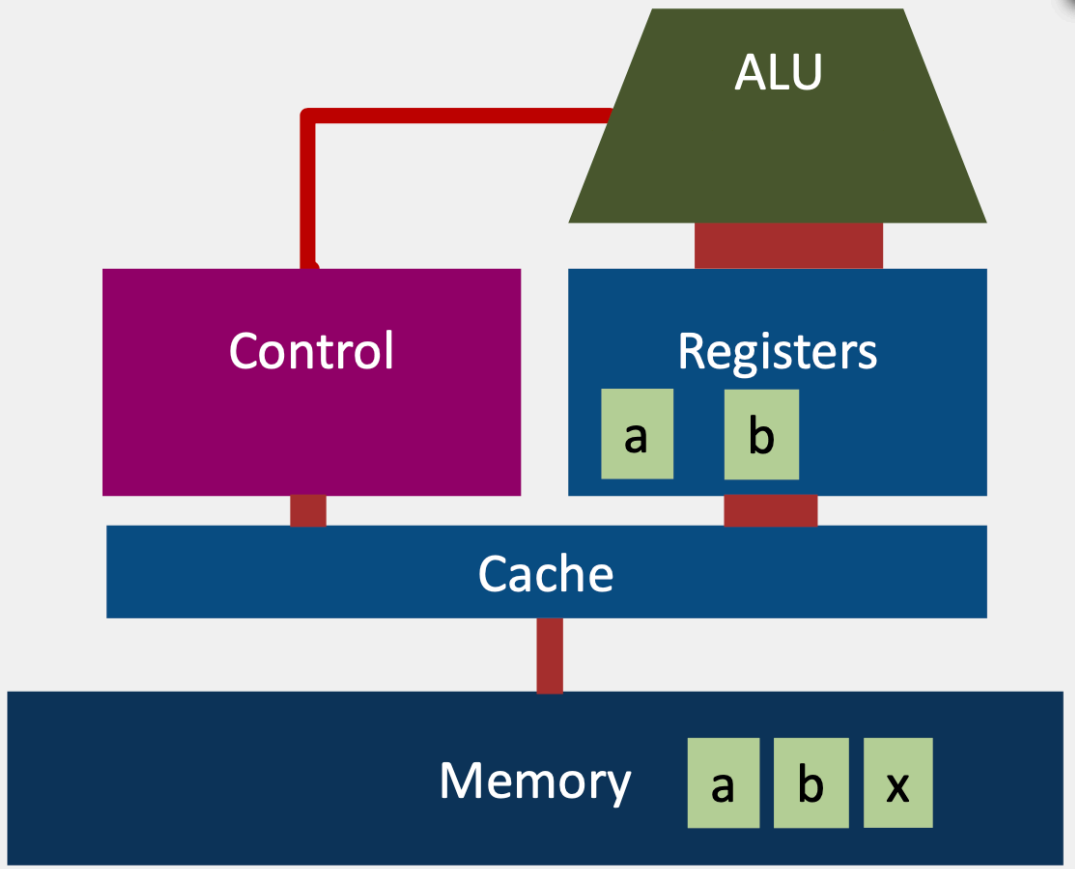
SC'20 Tutorial:
J. De Fine Licht and T. Hoefler
"Productive Parallel Programming on FPGA with High-Level Synthesis"

[1]: Marc Horowitz, Computing's Energy Problem (and what we can do about it), ISSC 2014, plenary
[2]: Moore: Landauer Limit Demonstrated, IEEE Spectrum 2012

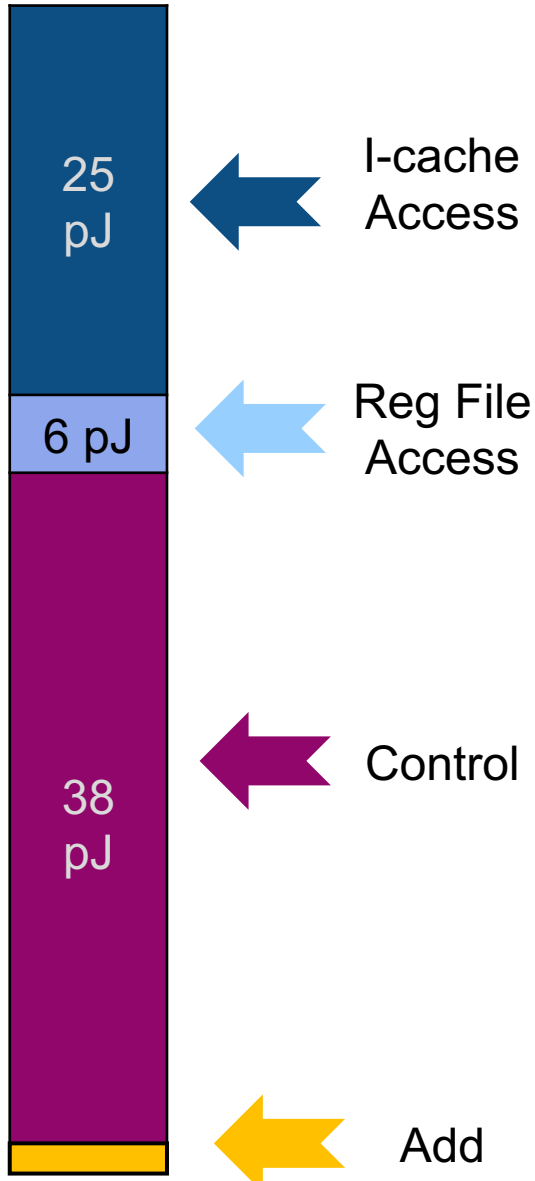
Load-store vs. Dataflow architectures

Load-store ("von Neumann")

x=a+b



70 pJ, **Energy per instruction**



Source: Mark Horowitz, ISSC'14

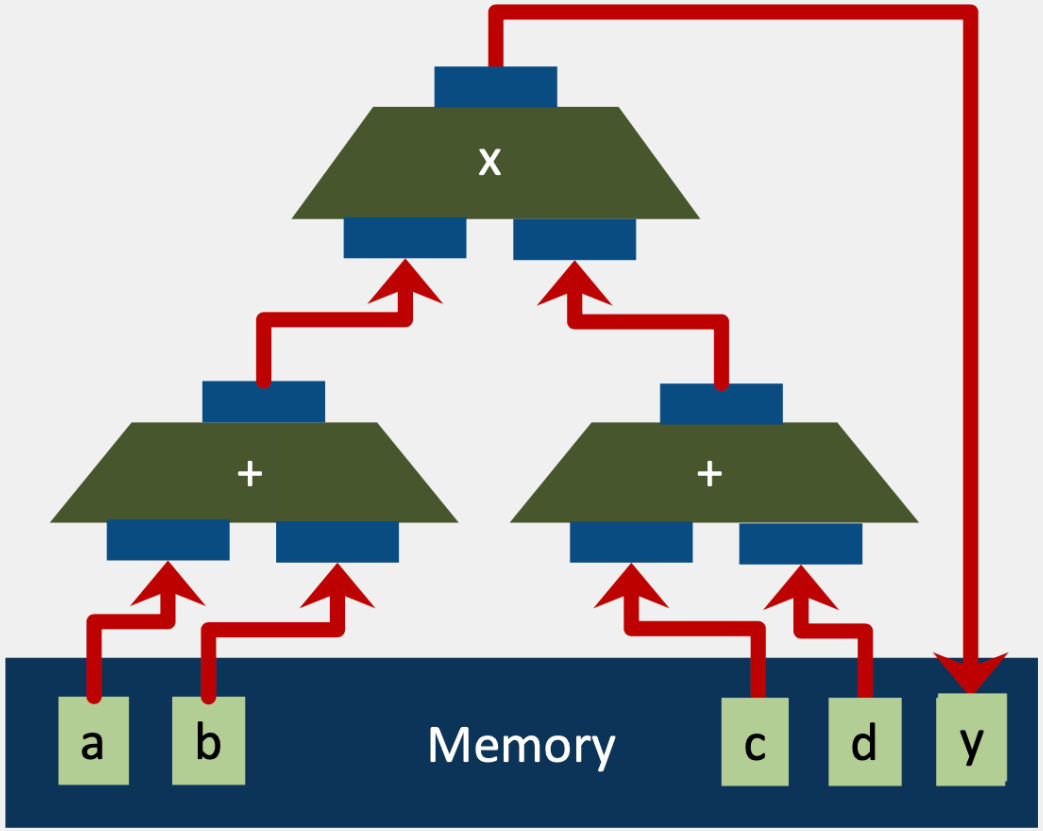
Dataflow architecture

Turing Award 1977 (Backus): "Surely there must be a less primitive way of making big changes in the store than pushing vast numbers of words back and forth through the von Neumann bottleneck."

Energy per operation:
1-3 pJ

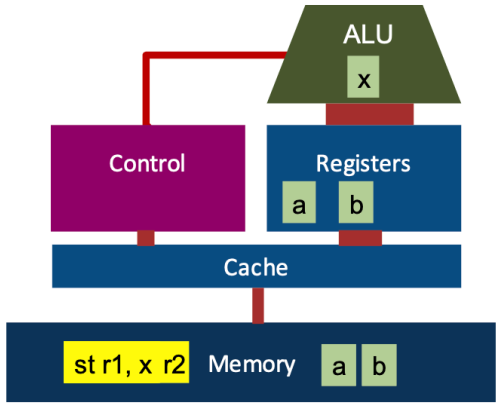
Static Dataflow ("non von Neumann")

y=(a+b)*(c+d)

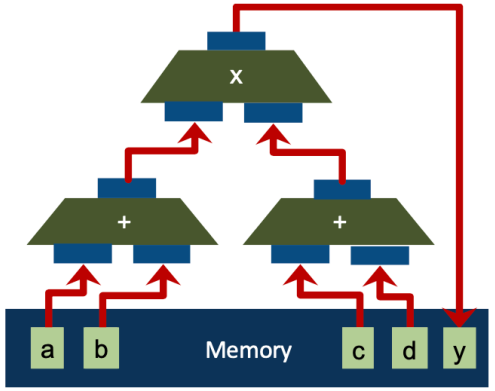


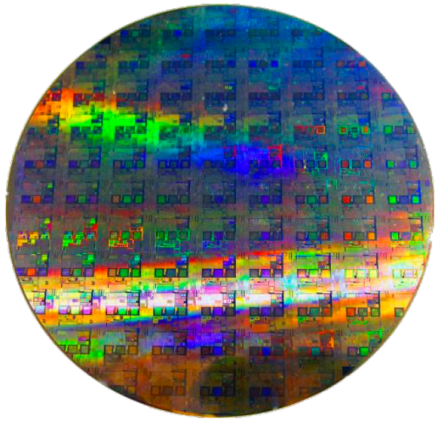
intel oneAPI BASE TOOLKIT

The paradox of FPGA efficiency

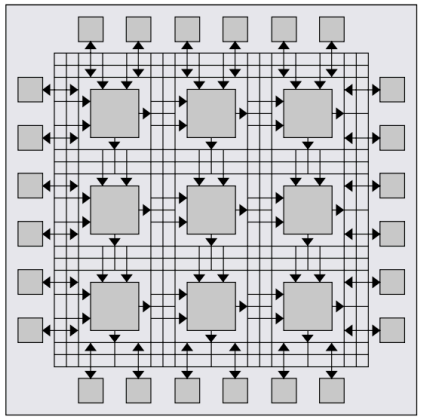


~100x





~0.05x

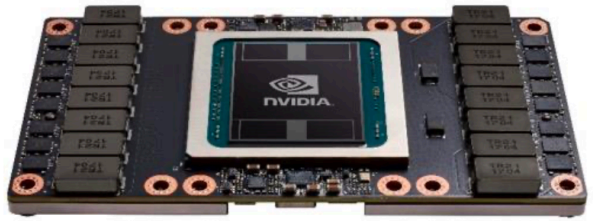


5x

High-Performance FPGA Hardware Overview

- **14 nm Intel Tri-Gate**
  - 1 GHz
- **10 TF single precision**
- **5.5M Logic Elements**
  - 4-input LUT, register, carry, etc.
- **Block RAM: 28.6 MiB**
- **Hardened DRAM controller DDR 4**
  - Various options for memory
- **Hyper Flex Interconnect with Regs.**
- **TDP: 125W (estimated)**

- **14 nm**
  - ~600 MHz
- **12,288 DSPs** (5.5 TF single prec.)
- **2.5M Logic Elements**
  - 1,728,000 5-input LUTs
  - 3,456,000 FFs
- **Block RAM: 54 MiB**
- **TDP: 225 W**



- **8 nm**
  - 1410 MHz
- **6,912 cores** (19.5 TF single prec.)
- **On-chip memory:**
  - Registers: 27.6 MB
  - L1/Shared Memory: 17.7 MB
  - L2 Cache: 40 MB
- **TDP: 400W**

FPGAs in the cloud since a while

IBM and Altera Collaborate on OpenCL

"IBM's collaboration with Altera on OpenCL and support of the IBM Power architecture with the Altera SDK for OpenCL can bring more innovation to address Big Data and cloud computing challenges," said Tom Rosamilia, senior vice president, IBM Systems

Intel Reveals FPGA and Xeon in One Socket

"That allows end users that have applications that can benefit from acceleration to load their IP and accelerate that algorithm on that FPGA as an offload," explained the vice president of Intel's data center group, Diane Bryant

Search Engine Gets Help From FPGA

"Altera was really interesting in helping with the development—the resources they were willing to throw our way were more significant than those from Xilinx" Microsoft Engr Manager

Baidu and Altera Demonstrate Faster Image Classification

"Altera Corp. and Baidu, China's largest online search engine, are collaborating on using FPGAs and convolutional neural network (CNN) algorithms for deep learning applications…."

Xilinx Announces SDAccel Development Environment for OpenCL

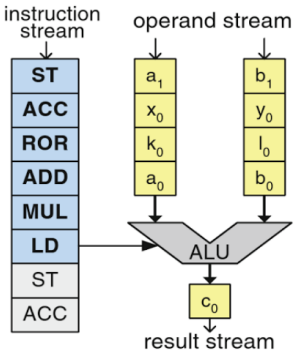
"Delivering Up to 25X Better Performance/Watt to the Data Center"

COULD FPGAS OUTWEIGH CPUS IN COMPUTE SHARE?

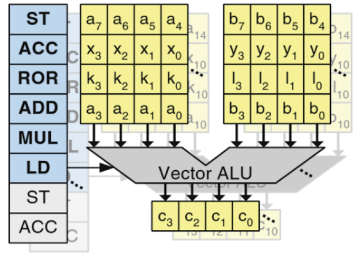
CPU vs GPU vs FPGA

- All of them are Turing complete, but which architecture suits me best?
- It depends on the problem:
  - CPU:
    - For control-dominant problems
    - Frequent context switching
  - GPU:
    - Data-parallel problems (vectorized/SIMD)
    - Little control flow and little synchronization
  - FPGA:
    - Stream processing problems
    - Large data sets processed in a pipelined fashion
    - Low power constraints

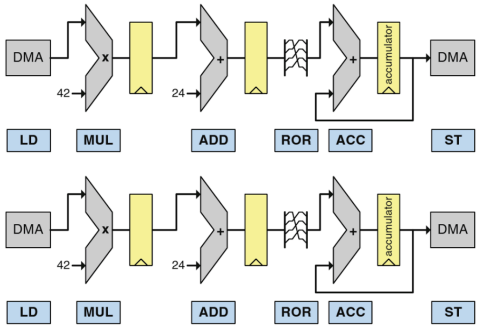
CPU:



GPU:



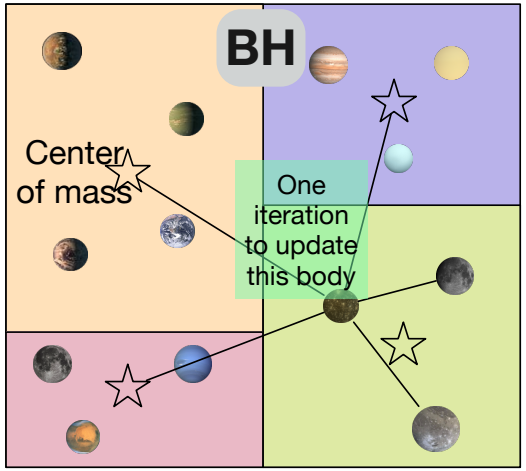
FPGA:

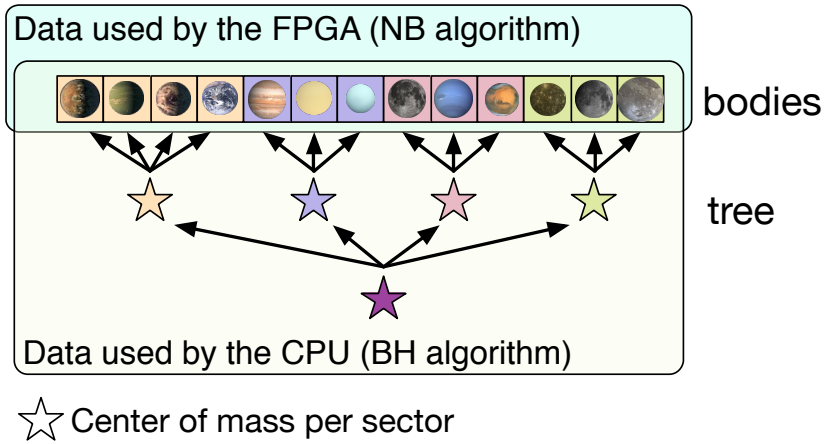


Heterogeneous computation

Regular: Nbody

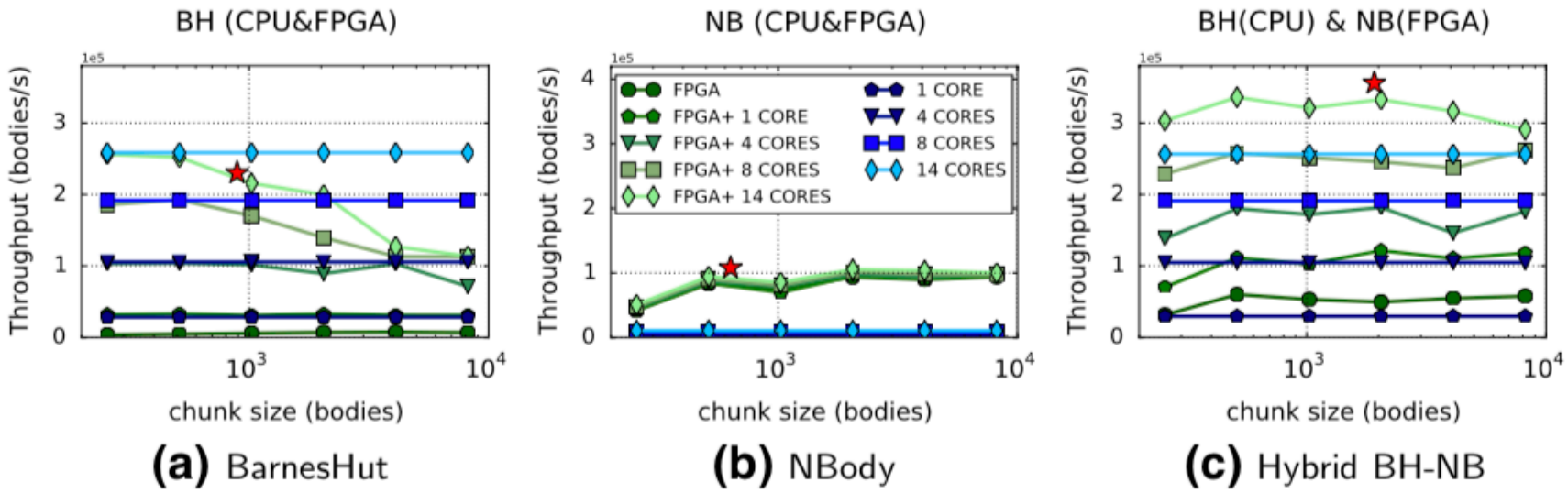
Irregular: Barnes-Hut





Results on

Xeon+FPGA Arria10

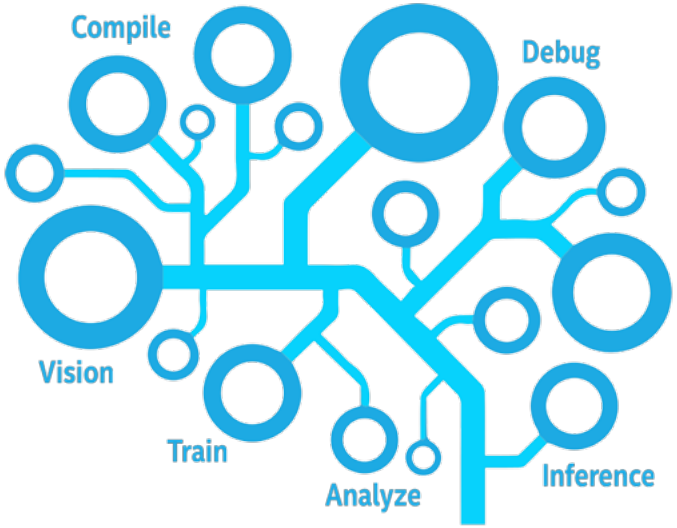


Parallel multiprocessing and scheduling on the heterogeneous Xeon+FPGA platform, Journal of Supercomputing, 2019

Agenda

- ✓ Introduction
- ➢ Lab: DevCloud
- Platform & Compilation
- FPGA optimizations
- Lab: DevCloud

Fine, but how do I use it?



- Get the software:
  - The ninja way (install from sources):

    https://github.com/intel/llvm

  - The clever way (download the binaries):
    - oneAPI Base Toolkit (FPGA emulation and optimization report)
    - Intel FPGA Add-on for oneAPI Base Toolkit (Quartus included to compile for FPGA)

    https://software.intel.com/content/www/us/en/develop/tools/oneapi.html

  - The lazy way (get a free account on DevCloud)

    https://devcloud.intel.com/oneapi/

First demo

- Hands-on examples on github here.

- Enter on DevCloud, open a terminal and write:

```
git clone -b onetbb_tutorials https://github.com/oneapi-src/oneTBB.git
```

- Open Jupyter Notebook on DevCloud and navigate to `oneTBB/examples/SC20`
- Open `02-HandsOn-HelloWorld-DPC++.ipynb`
- Examples of the DPC++ book on github: here.
- Curious.cpp and Verycurious.cpp examples: here

Agenda

- ✓ Introduction
- ✓ Lab: DevCloud
- ➢ Platform & Compilation
- FPGA optimizations
- Lab: DevCloud

DPC++: Platform and Compilation model

- Platform model
  - Where should I run every kernel? → already discussed
  - Where can I run my kernels? → Device discovery
  - Where do I run my kernels? → Device selection

- Compilation model
  - How is the code compiled? → Just-in-time or Ahead-of-time compilation
  - How is it possible to target different devices? → Fat binaries

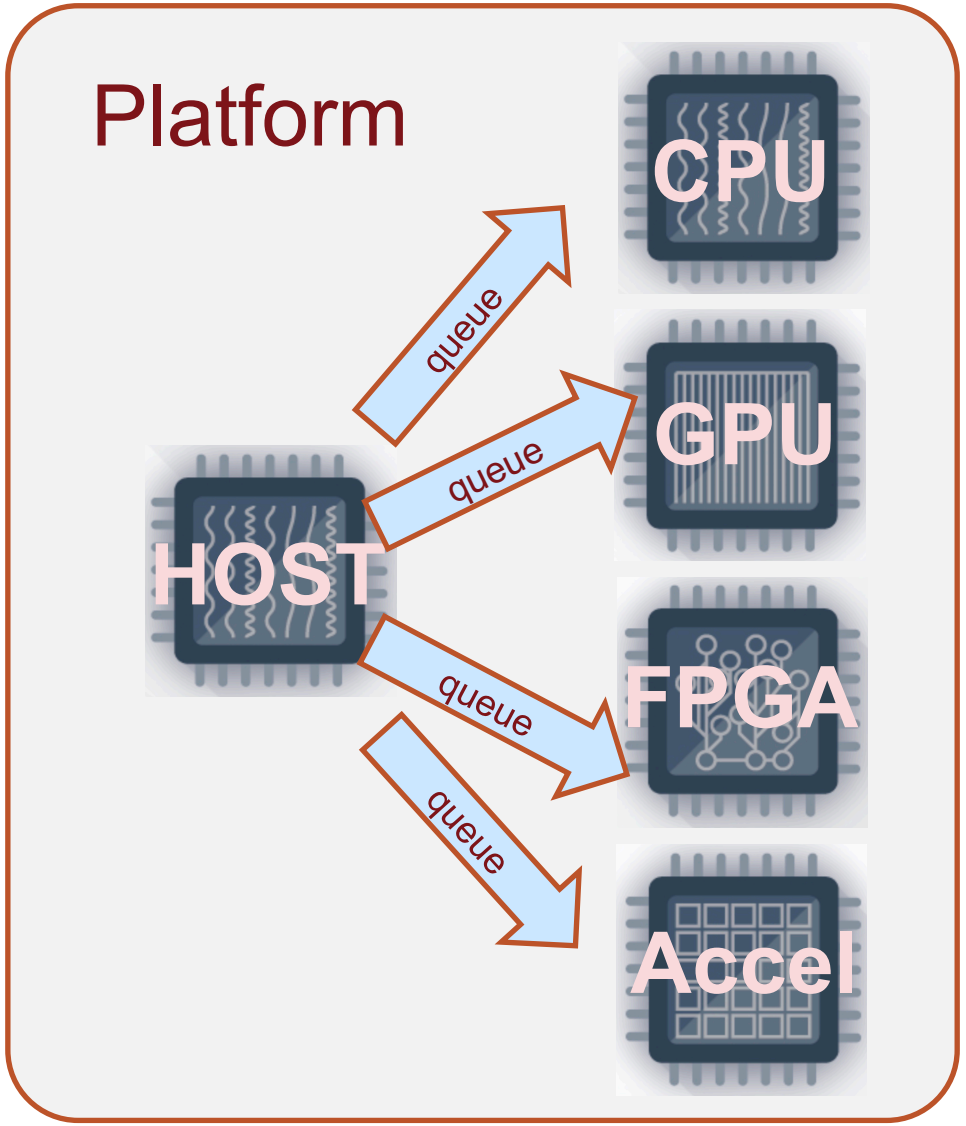
- FPGA Development flow

Platform

- Platform = host + devices
  - Host enqueue kernels and run host code
  - Devices run kernels

- Notes:
  - The host is usually the CPU
  - There is always a “host device”
    - The CPU can also run kernels

- Single source programming paradigm
  - A file can have host and device code



Platform model

- The number of available devices depends on:
  - Hardware: the devices we actually have installed in our platform
  - Drivers: the devices recognized by the OS or runtime
    - For oneAPI, it can be an OpenCL driver, but more recently is Level Zero (HAL)

On DevCloud we can inspect the available nodes using pbsnodes:

```
$ pbsnodes -l free              // list all available nodes

$ pbsnodes s001-n049            // reports on the information of that particular node

$ pbsnodes | tail -30           // shows the information of the last 3 nodes

$ pbsnodes | grep properties    // list the properties of all the nodes

$ pbsnodes | grep fpga          // list the nodes with fpga support
```

Select and query the available HW

- Once you know the node you want to use or the desired property
  - Use qsub to enqueue a job to the node (more info here)

```
$ qsub -l nodes=1:ppn=2:gpu l.sh        // Submit to a node with gpu

$ qsub -l nodes=1:ppn=2:fpga l.sh       // Submit to a node with fpga, there is also fpga_compile

$ qsub -l nodes=1:ppn=2:cfl l.sh        // Submit to a node with Coffee Lake architecture

$ qsub -d. -I                           // Occupies a node for interactive use (be polite!)
```

- Once you are in the node you can query more details

```
$ more /proc/cpuinfo                    // Info about the CPU
$ lscpu                                 // A better way to check the CPU features
$ clinfo                                // Info about the OpenCL devices
$ aoc list-boards                       // List of FPGA devices
```

Query platforms & devices from DPC++

```
1  #include <CL/sycl.hpp>
2
3  using namespace cl::sycl;
4
5  int main() {
6    unsigned number = 0;
7    auto MyPlatforms = platform::get_platforms();
8
9    /* Loop through the platforms SYCL can find
10      there is always ONE */
11   for (auto &OnePlatform : MyPlatforms) {
12     std::cout << ++number << " found..."
13           << std::endl
14           << "Platform: "
15           << OnePlatform.get_info<info::platform::name>()
16           << std::endl;
17
18     /* Loop through the devices SYCL can find
19        there is always ONE */
20     auto MyDevices = OnePlatform.get_devices();
21     for (auto &OneDevice : MyDevices ) {
22       std::cout << " Device: "
23         << OneDevice.get_info<info::device::name>()
24         << std::endl;
25     }
26     std::cout << std::endl;
27   }
28 }
```

Get all platforms

Traverse them

Print platform name

For each device in the platform

Print device name

Courtesy: James Reinders, DPC++ webinar for Colfax

Query platforms and devices from DPC++

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

using namespace cl::sycl;

int main() {
  unsigned number = 0;
  auto MyPlatforms = platform::get_platforms();

  /* Loop through the platforms SYCL can find
     there is always ONE */
  for (auto &OnePlatform : MyPlatforms) {
    std::cout << ++number << " found..."
          << std::endl
          << "Platform: "
          << OnePlatform.get_info<info::platform::name>()
          << std::endl;

    /* Loop through the devices SYCL can find
       there is always ONE */
    auto MyDevices = OnePlatform.get_devices();
    for (auto &OneDevice : MyDevices ) {
      std::cout << " Device: "
        << OneDevice.get_info<info::device::name>()
        << std::endl;
    }
    std::cout << std::endl;
  }
}
```

A possible output:

```
/**
 **  sample output while logged into a DevCloud node
 **  (output will vary by DevCloud node based on node configuration)
 **
 **  % make curious
 **  dpcpp curious.cpp -o curious
 **
 **  % ./curious
 **  1 found...
 **  Platform: Intel(R) FPGA Emulation Platform for OpenCL(TM)
 **   Device: Intel(R) FPGA Emulation Device
 **
 **  2 found...
 **  Platform: Intel(R) OpenCL HD Graphics
 **   Device: Intel(R) Gen9 HD Graphics NEO
 **
 **  3 found...
 **  Platform: Intel(R) OpenCL
 **   Device: Intel(R) Xeon(R) E-2176G CPU @ 3.70GHz
 **
 **  4 found...
 **  Platform: SYCL host platform
 **   Device: SYCL host device
 **
 **/
```

Courtesy: James Reinders, DPC++ webinar for Colfax

More details are available

- `platform` and `device` classes offer the `get_info` template member function:
  - Name, vendor, and version of the device
  - Number of compute units, max # of work item dimensions
  - Width for built in types, clock frequency, cache width and sizes, online or offline

```
OnePlatform.get_info<info::platform::name>()
```

vendor
version
profile
extensions

See `verycurious.cpp` example

```
OneDevice.get_info<info::device::name>()
```

vendor
driver_version
max_work_item_sizes
sub_group_sizes
global_mem_size
local_mem_size
usm_host_allocation
usm_shared_allocation
global_mem_cache_line_size
local_mem_type
preferred_vector_width_int
…

Also useful:

```
OneDevice.is_host()
OneDevice.is_cpu()
OneDevice.is_gpu()
OneDevice.is_accelerator()
```

Enqueue work (kernel) to your device

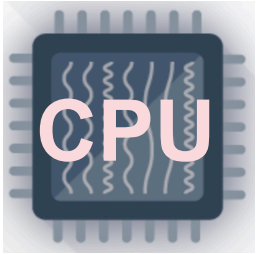
Create a queue for the devices you want to use

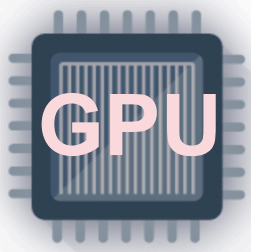
```
1   #include <CL/sycl.hpp>
2   #include <CL/sycl/INTEL/fpga_extensions.hpp>
3   using namespace cl::sycl;
4   int main() {
5       queue q;
6       queue q_def{default_selector{}};
7       queue q_host{host_selector{}};
8       queue q_cpu{cpu_selector{}};
9       queue q_gpu{gpu_selector{}};
10      queue q_acc{accelerator_selector{}};
11      queue q_fpga_emu{INTEL::fpga_emulator_selector{}};
12  //queue q_fpga{INTEL::fpga_selector{}};
13      std::cout << "Device: " << q.get_device().get_info<info::device::name>()
14                << "\nDefault device: " << q_def.get_device().get_info<info::device::name>()
15                << "\nHost device: " << q_host.get_device().get_info<info::device::name>()
16                << "\nCPU device: " << q_cpu.get_device().get_info<info::device::name>()
17                << "\nGPU device: " << q_gpu.get_device().get_info<info::device::name>()
18                << "\nAccelerator device: " << q_acc.get_device().get_info<info::device::name>()
19                << "\nFPGA emulator device: " << q_fpga_emu.get_device().get_info<info::device::name>() << "\n";
20  //std::cout << "\nFPGA device: " << q_fpga.get_device().get_info<info::device::name>() << "\n";
21      return 0;
22  }
```

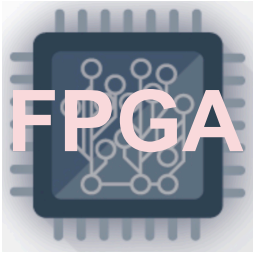
Note: required to use Intel FPGAs

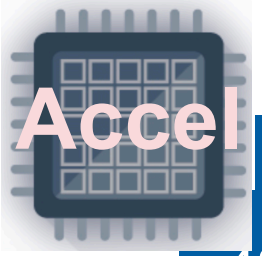
Constructing several queues

From each queue:
`q.get_device().get_info<…name>()`







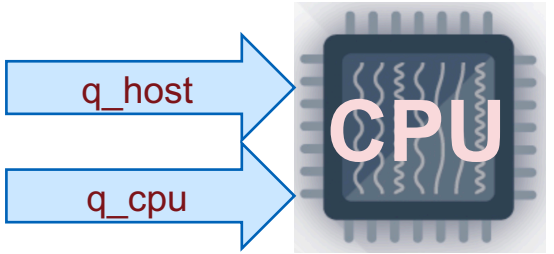


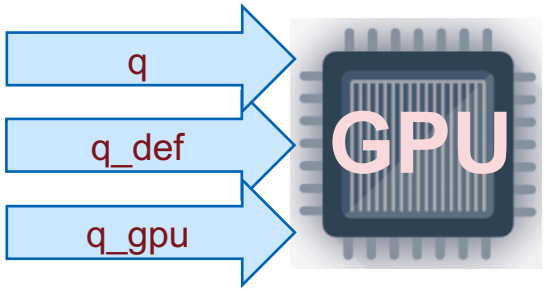
Enqueue work (kernel) to your device

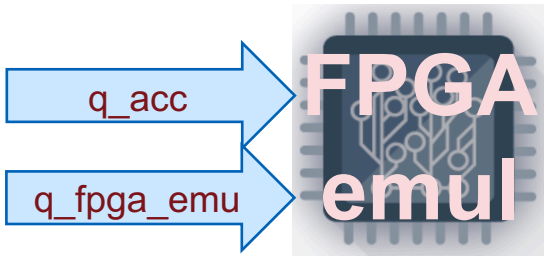
A possible output (on a node w/o FPGA):

```
queue q;
queue q_def{default_selector{}};
queue q_host{host_selector{}};
queue q_cpu{cpu_selector{}};
queue q_gpu{gpu_selector{}};
queue q_acc{accelerator_selector{}};
queue q_fpga_emu{INTEL::fpga_emulator_selector{}};
```

```
Device: Intel(R) Gen9 HD Graphics NEO
Default device: Intel(R) Gen9 HD Graphics NEO
Host device: Intel(R) Xeon(R) E-2176G CPU @ 3.70GHz
CPU device: Intel(R) Xeon(R) E-2176G CPU @ 3.70GHz
GPU device: Intel(R) Gen9 HD Graphics NEO
Accelerator device: Intel(R) FPGA Emulation Device
FPGA emulator device: Intel(R) FPGA Emulation Device
```







Custom device selector

What if there are several GPUs/FPGAs and you want to use one of them?

```
1  #include <CL/sycl.hpp>
2  #include <iostream>
3  using namespace sycl;
4  class my_device_selector : public device_selector {
5  public:
6      my_device_selector(std::string vendorName) : vendorName_(vendorName){};
7      int operator()(const device& dev) const override {
8          int rating = 0;
9          if (dev.is_gpu() & (dev.get_info<info::device::name>().find(vendorName_) != std::string::npos))
10             rating = 3;
11         else if (dev.is_gpu()) rating = 2;
12         else if (dev.is_cpu()) rating = 1;
13         return rating;
14     };
15
16 private:
17     std::string vendorName_;
18 };
19 int main() {
20     queue q{my_device_selector{"Intel"}};
21     std::cout << "Device: " << q.get_device().get_info<info::device::name>() << std::endl;
22     return 0;
23 }
```

My own class derived from `device_selector`

The constructor receives a `string vendorName`

The operator() runs in all devices

3 if a GPU with “`vendorName`”

2 if a GPU

1 if a CPU

The device returning the highest value is selected

Beware of incorrect device selection

If your desired device is not available:

customselector1.cpp

```
class my_selector : public device_selector {
public:
     int operator()(const device &dev) const {
        if (
              dev.get_info<info::device::name>().find("PAC")
              != std::string::npos &&
              dev.get_info<info::device::vendor>().find("Intel")
              != std::string::npos) {
            return 1;
        } else {
            return -1;
        }
    }
};
int main() {
    queue q_fpga{my_selector{}};
    std::cout << "Selected device: " <<
    q_fpga.get_device().get_info<info::device::name>() << "\n";
    return 0;
}
```

```
asenjo — u32284@s001-n081: ~/sc20tut — ssh ◂ ssh devcloud
[u32284@login-2:~/sc20tut$ dpcpp -o c customselector1.cpp
[u32284@login-2:~/sc20tut$ ./c
terminate called after throwing an instance of 'cl::sycl::runtime_error'
  what(): No device of requested type available. Please check https://software.
intel.com/en-us/articles/intel-oneapi-dpcpp-compiler-system-requirements-beta -1
 (CL_DEVICE_NOT_FOUND)
Aborted
[u32284@login-2:~/sc20tut$ qsub -d . -I -l nodes=1:ppn=2:fpga
qsub: waiting for job 695047.v-qsvr-1.aidevcloud to start
qsub: job 695047.v-qsvr-1.aidevcloud ready


################################################################
#      Date:           Sun Sep 27 09:49:18 PDT 2020
#    Job ID:           695047.v-qsvr-1.aidevcloud
#      User:           u32284
# Resources:           neednodes=1:ppn=2:fpga,nodes=1:ppn=2:fpga,walltime=06:00:
00
################################################################

[u32284@s001-n081:~/sc20tut$ ./c
Selected device: pac_a10 : Intel PAC Platform (pac_ee00000)
[u32284@s001-n081:~/sc20tut$
```

oopss!

change to an FPGA node

NOW!

Use try-catch blocks

Initialize a default device (there is always one)

```
1   int main() {
2       device my_device;
3       try{
4           my_device = device{my_selector{}};
5       } catch (...){
6           std::cout << "Warning! Can not get your preferred device.\n"
7                     << "Continuing on default device." << std::endl;
8       }
9       queue q{my_device};
10      std::cout << "Selected device: " <<
11                q.get_device().get_info<info::device::name>() << "\n";
12      return 0;
13  }
```

default device

try your preferred one

bad luck!

```
asenjo — u32284@s001-n081: ~/sc20tut — ssh ◂ ssh devcloud
u32284@login-2:~/sc20tut$ dpcpp -o c customselector2.cpp
u32284@login-2:~/sc20tut$ ./c
Warning! Can not get your preferred device.
Continuing on default device.
Selected device: SYCL host device
u32284@login-2:~/sc20tut$ qsub -d . -I -l nodes=1:ppn=2:fpga
qsub: waiting for job 695052.v-qsvr-1.aidevcloud to start
qsub: job 695052.v-qsvr-1.aidevcloud ready


################################################################################
#      Date:           Sun Sep 27 10:27:26 PDT 2020
#    Job ID:           695052.v-qsvr-1.aidevcloud
#      User:           u32284
# Resources:           neednodes=1:ppn=2:fpga,nodes=1:ppn=2:fpga,walltime=06:00:
00
################################################################################

u32284@s001-n081:~/sc20tut$ ./c
Selected device: pac_a10 : Intel PAC Platform (pac_ee00000)
```

Fallback command queue

Use in rare occasions:

```
queue q_gpu{gpu_selector{}};
queue q_host{host_selector{}};
buffer<int, 1> myBuffer(myData, range<1>{sz});

q_gpu.submit([&](handler &h) {
    auto myAccessor =
       myBuffer.get_access<access::mode::read_write>(h);
    h.parallel_for(range<1>{sz}, [=](id<1> myID) {
        myAccessor[myID]++;
    });
}, q_host);
```

two queues

submit to GPU

Host as fallback (2nd arg of submit)

- If submission to the queue fails
  - enqueue size too large
  - device fails
- Better catch the exception

Compilation model

Platform = host + devices. All require instructions to run our app

- Hide complexity: the dpcpp compiler looks like a regular “homogeneous” compiler
- But generate an executable with instruction for the host and devices
- Single file: a single source file can include host and device

Fat objects and binaries

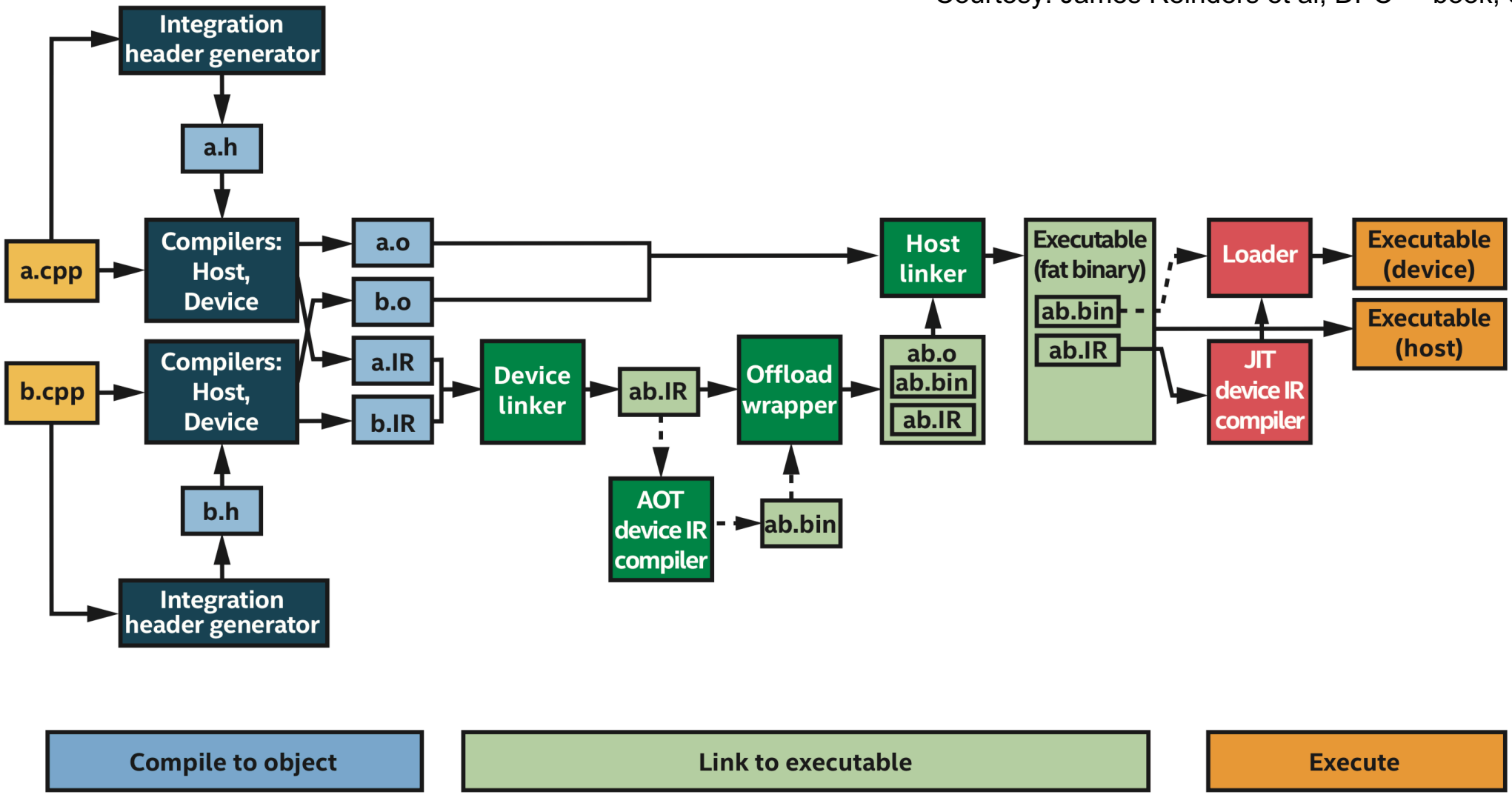
- Several compilers to generate object and binaries for the different devices
- All generated objects and binaries can be packed in a single object and binary file

Just-in-time and Ahead-of-time compilation

- Just-in-time or online compilation
  - When the compiler doesn’t know de final device and a JIT phase does not take too long
- Ahead-of-time or offline compilation
  - At compile time the target architecture is known, and specific optimizations can be applied

Compilation model

Courtesy: James Reinders et al, DPC++ book, chapter 13



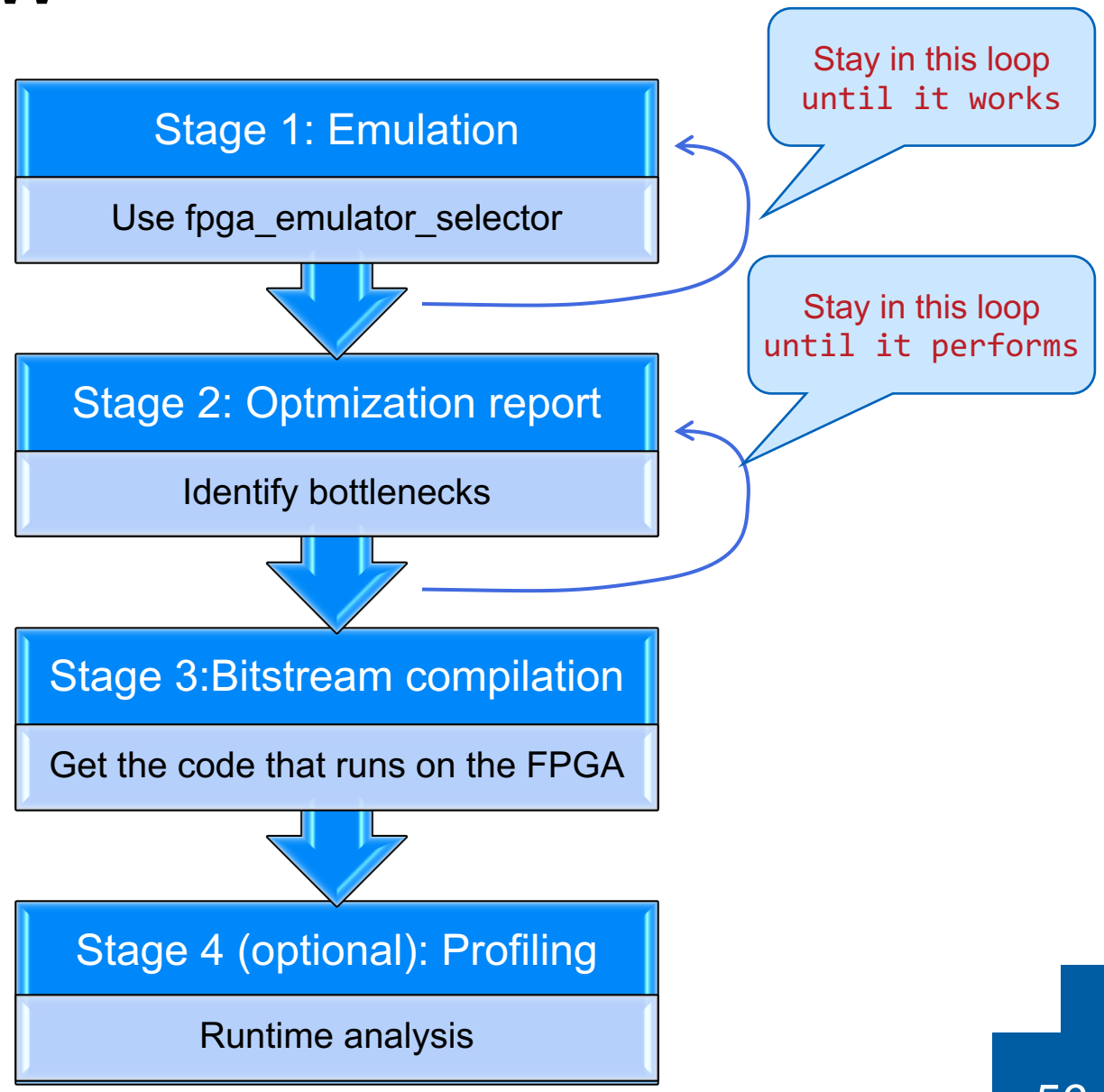
FPGA development flow

Emulation
- Compiles in second
- Runs on the host (debug your app)

Optimization report
- Compiles in seconds to minutes
- Get an idea of performance

Bitstream compilation
- Compiles in hours
- Real performance

Profiling compilation
- Compiles in hours
- Collect runtime samples for VTune



Emulation

General anatomy of the compiler command:

```
dpcpp –fintelfpga *.cpp/*.o [device link options] [-Xs FPGA_arguments]
```

Compile for emulation:

```
dpcpp –fintelfpga <source_file>.cpp -o <file_name>.emu –DFPGA_EMULATOR
```

```
#include <CL/sycl/INTEL/fpga_extensions.hpp>
using namespace cl::sycl;
...
#ifdef FPGA_EMULATOR
  INTEL::fpga_emulator_selector device_selector;
#else
  INTEL::fpga_selector device_selector;
#endif
queue deviceQueue{device_selector};
...
```

Remember: include required for FPGA

Compile-time selection of device

Construction of device queue

Optimization report

Compile to get the optimization report

```
dpcpp -fintelfpga <source_file>.cpp -c -o <file_name>.o
dpcpp -fintelfpga <file_name>.o -fsycl-link -Xshardware
```

Flag `-fsycl-link` can take two values

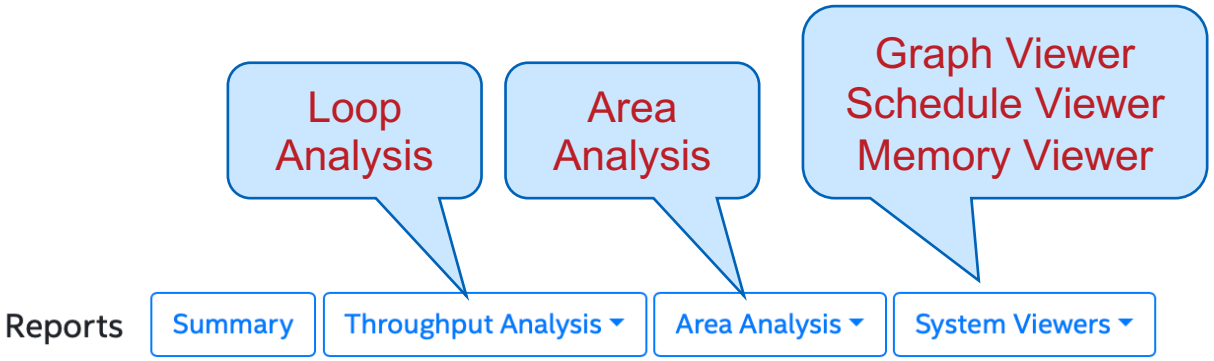
- `early` (by default) just produces the report and early image report
- `image` (`-fsycl-link=image`) takes its time to produce the bitstream

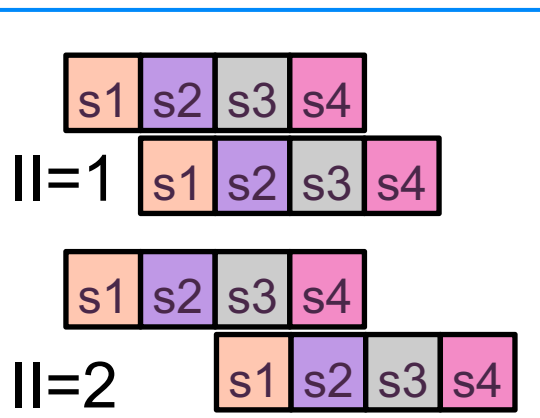
The compiler generates the file `<file_name>.prj/reports/report.html`

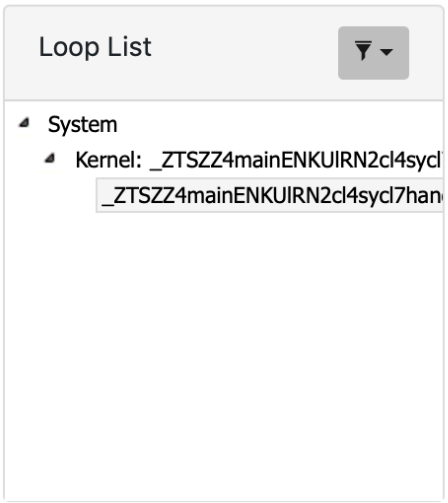
The report includes:

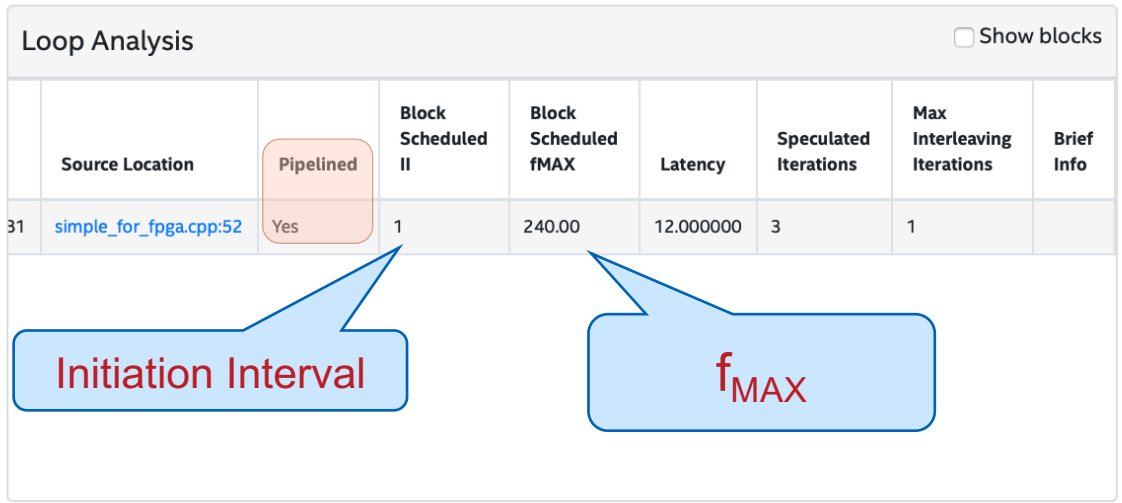
- Throughput Analysis: f_{MAX}, Initiation Interval (II)
- Area Analysis: estimated area utilization and resources (ALUTs, FFs, DSPs, Block RAM,…)
- Graph Viewer: high level view of the implemented HW
- Schedule Viewer: pipeline schedule in clock cycles for the blocks of your kernel
- Memory Viewer: load/store implementation

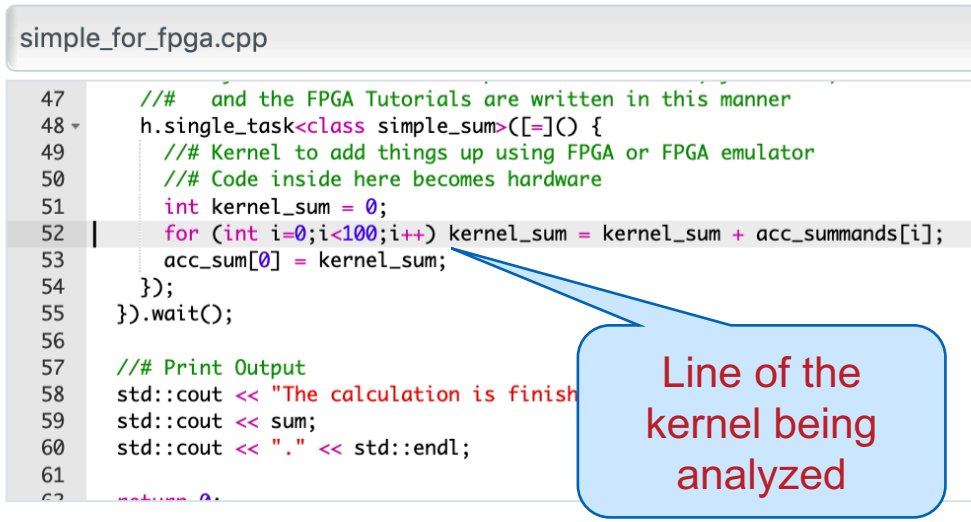
Optimization report











- Initiation Interval: number of ck between iterations of the loop (Ideal II=1)
- f_{MAX}: maximum clock frequency at which the circuit can operate

Bitstream compilation and profiling

Bitstream compilation using –fsycl-link=image

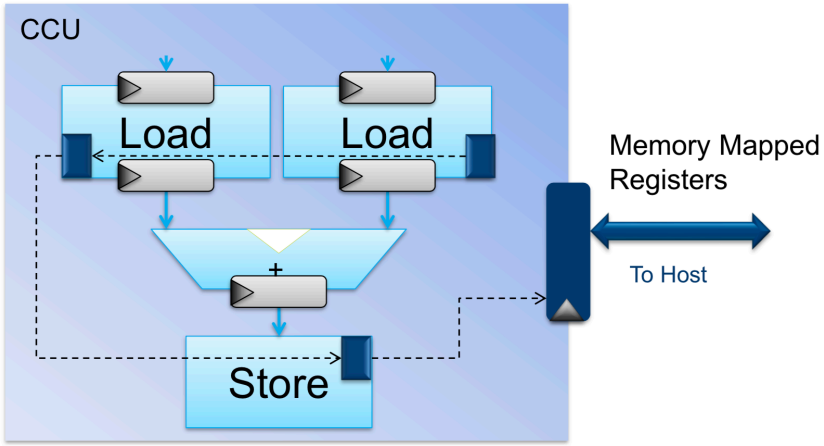
Since the kernel compilation takes longer → use a different file for the kernel

has_kernel.cpp

1. `dpcpp –fintelfpga has_kernel.cpp –o has_kernel.o –fsycl-link=image -Xshardware`
2. `dpcpp –fintelfpga host_only.cpp –c –o host_only.o`
3. `dpcpp –fintelfpga has_kernel.o host_only.o –o a.out -Xshardware`

Profiling data generated if using –Xsprofile

- Inserts performance counters into the HW



Collecting profiling data

- `aocl profile –s has_kernel.cpp a.out`
- Import the resulting `profile.json` file into VTune

Agenda

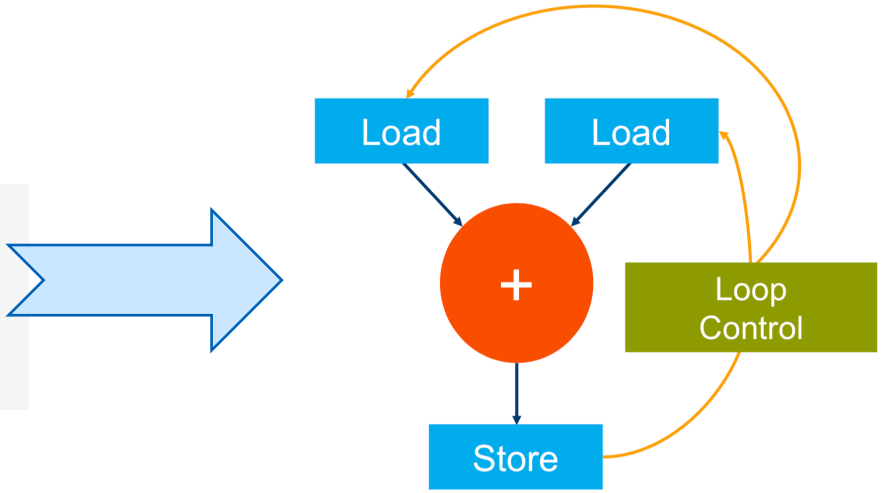
- ✓ Introduction
- ✓ Lab: DevCloud
- ✓ Platform & Compilation
- ➢ FPGA optimizations
- Lab: DevCloud

Optimizations

Courtesy: Intel (Susannah Martin, oneAPI FPGA workshop)

DPC++ compiler can map code into hardware

```
for (int i=0; i<size; i++) {
    c[i] = a[i] + b[i];
}
```



But compilers are not almighty!:

- Neither for CPU → can not convert Bubble sort into Quicksort, vectorize all loops, etc.
- Nor for GPU → can not automatically avoid control/data divergence, exploit locality, etc.

We learned how to optimize code for CPU, GPU, shared memory arch….

We also need to learn how to optimize code for the FPGA architecture

Summary of optimizations

Throughput
- Single work-item kernels (a.k.a. single_task kernels)
- More pipelines (#pragma unroll)
- Optimized tree-like reductions

Locality:
- Shift-Register Pattern
- Local memory
- Memory banking
- Data coalesce

Kernel Replication
- Try to use more FPGA area if available
- Downsides: memory pressure ⬆️, complexity ⬆️, frequency ⬇️

Single work-item vs NDRange kernels

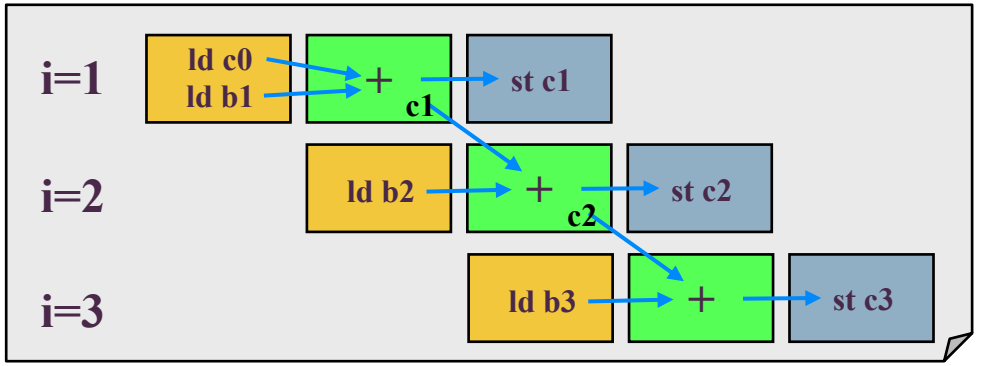
NDRange Kernel

```
cgh.parallel_for (range, [=](id<1> i){
   c[i] = a[i] + b[i];
});
```

Single work-item Kernel

```
cgh.single_task ([=](){
  for (int i = 1; i < range; i++){
    c[i] = c[i-1] + b[i];
  }
});
```

Loop analysis and most loop optimizations **only** for single work-item kernels



Loop unrolling

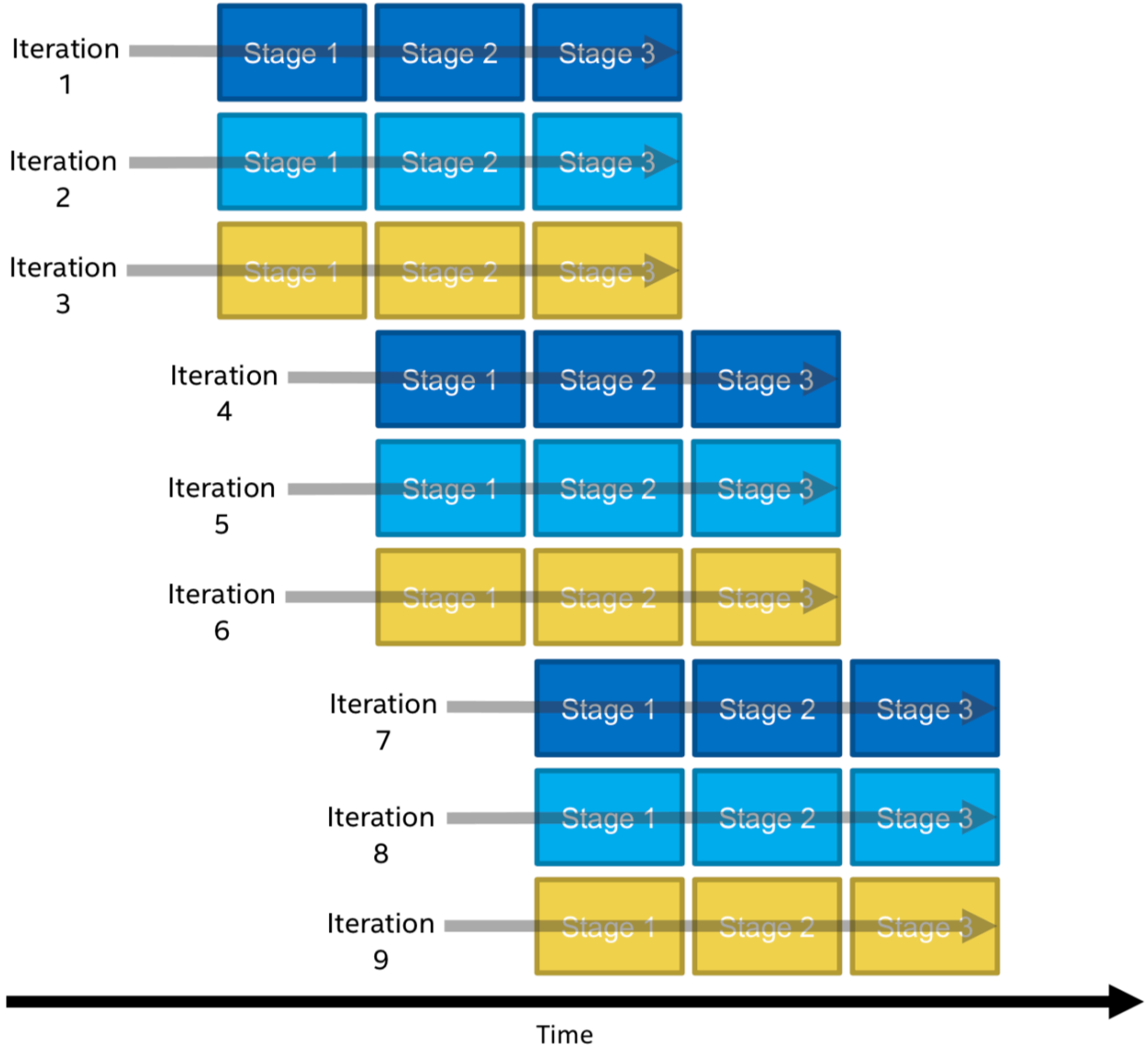
Pipeline + Spatial parallelism

Note: If full unrolled

- there is no pipelining
- all iterations kick off at once

```
handle.single_task<>([=]() {
... //accessor setup
  #pragma unroll 3
  for (int i=1; i<9; i++) {
    c[i] += a[i] + b[i];
  }
});
```

Courtesy: Intel (Susannah Martin, oneAPI FPGA workshop)



Loop unrolling

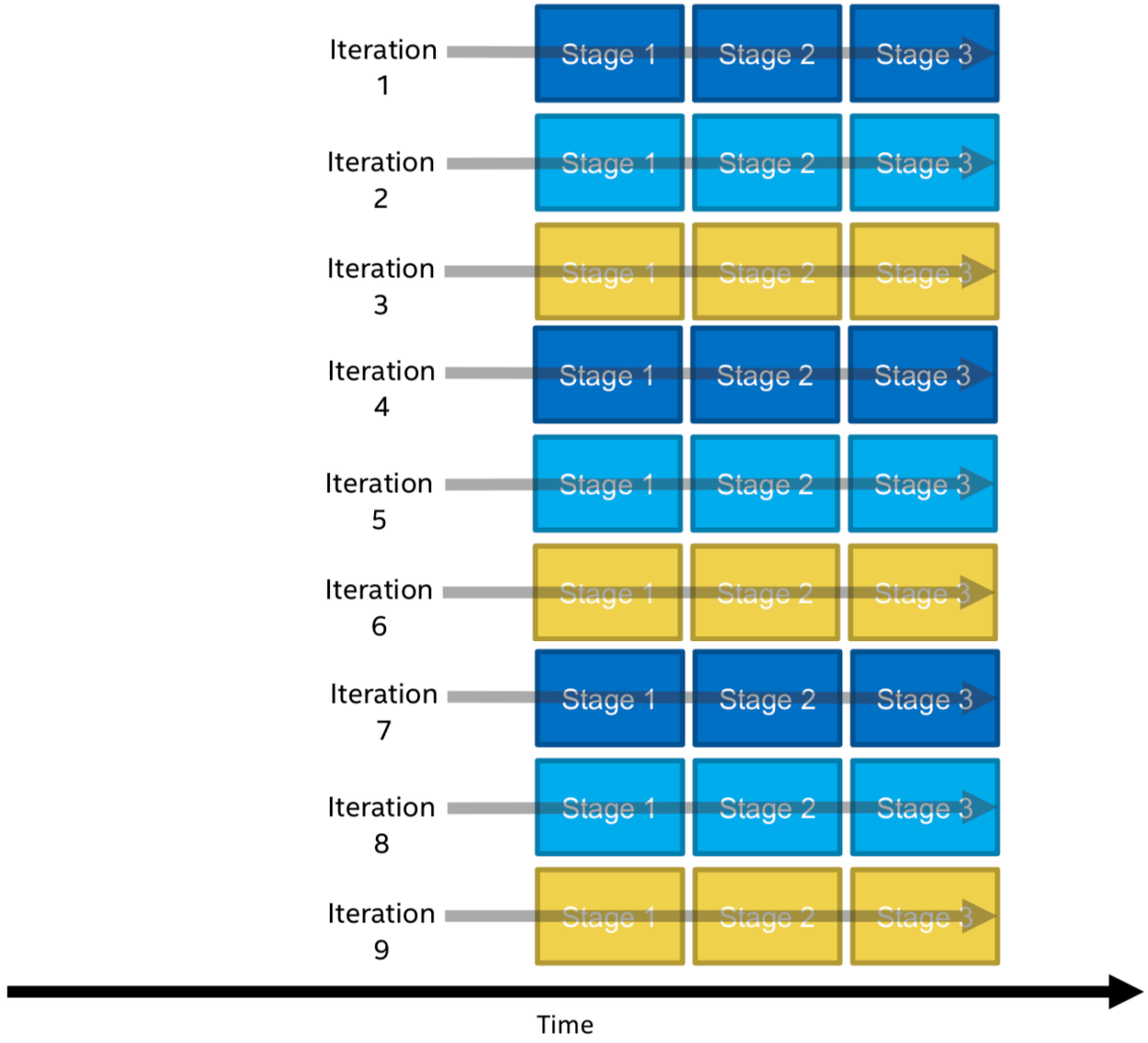
Courtesy: Intel (Susannah Martin, oneAPI FPGA workshop)

Pipeline + Spatial parallelism

Note: If full unrolled

- there is no pipelining
- all iterations kick off at once

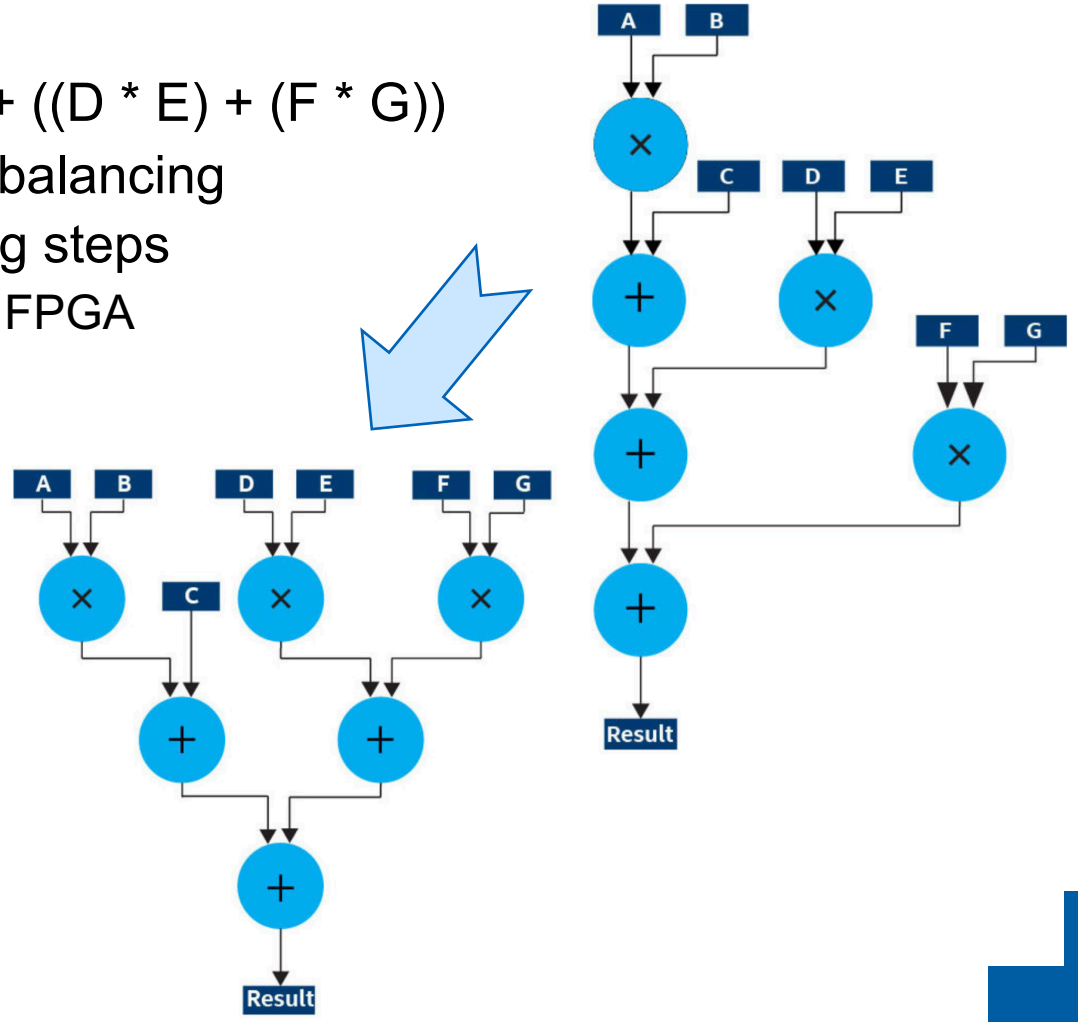
```
handle.single_task<>([=]() {
... //accessor setup
  #pragma unroll
  for (int i=1; i<9; i++) {
    c[i] += a[i] + b[i];
  }
});
```



Tree balancing and fused FP operations

For floating point operations:

- (((A * B) + C) + (D * E)) + (F * G) != ((A * B) + C) + ((D * E) + (F * G))
- But the the second is faster on the FPGA → tree balancing
- A series of FP operations require several rounding steps
  - But performing just one at the end is faster on the FPGA
  - Similar to the FMAC on CPUs and GPU



If some rounding error is acceptable:

- Enable tree balancing with –Xsfp-relaxed=true
- Enable fused FP operations with -Xsfpc

Loop carried dependencies

Inform the compiler that there are no dependencies between accessors

`[[intel::kernel_args_restrict]]`

- Equivalent to C `restrict` keyword

```
void updatePtrs(size_t *restrict ptrA, size_t *restrict ptrB, size_t *restrict val);
```

- Now:

```
q.submit([&](handler &h) {
  auto accessor_in = in.get_access<access::mode::read>(h);
  auto accessor_out = out.get_access<access::mode::discard_write>(h);

  h.single_task([=]() [[intel::kernel_args_restrict]] {
    for (size_t i = 0; i < N; i++) {
       ... = accessor_in[i];
       ...
       accessor_out[i] =
    }
  });
});
```

Ignore dependencies between accessors

We ensure that these accessors point to non-overlapping regions of memory

Loop carried dependencies

Inform the compiler that there are no memory loop-carried dependencies

- `[[intelfpga::ivdep]]` attribute (equivalent to `#pragma ivdep`)

```
[[intelfpga::ivdep]]
for (size_t j = 0; j < N * size; j++) {
  #pragma unroll
  for (size_t i = 0; i < size; i++) {
    A[i] = A[i - X[i]];
  }
}
```

Ignore memory loop-carried dependencies

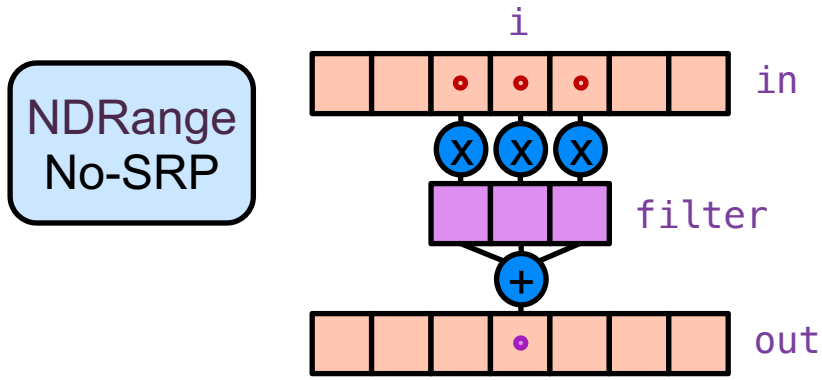
We ensure that these loads and stores are independent

`[[intelfpga::ivdep(safelen)]]` attribute

- E.g.: `ivdep(32)` guarantees that there does not exist a loop-carried dependence with a dependence distance less than 32

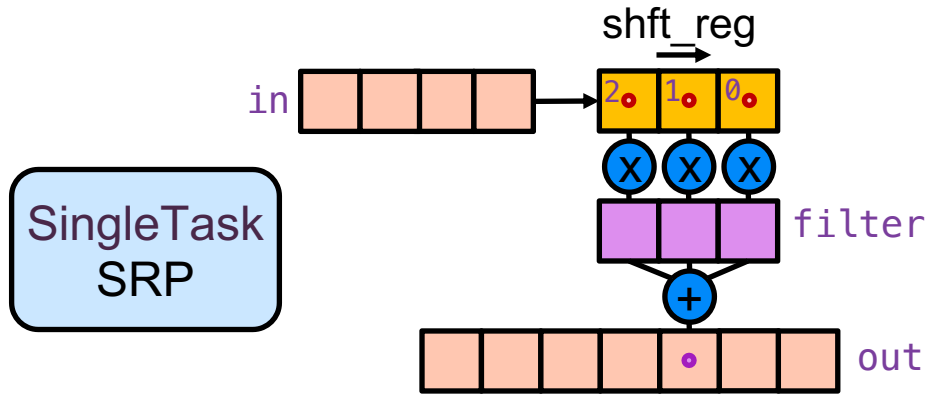
Locality

Shift register pattern (SRP). Example: 1D stencil computation




```
cgh.parallel_for (range, [=](id<1> i){
  float acc;
  acc  = in[i-1] * filter[0];
  acc += in[ i ] * filter[1];
  acc += in[i+1] * filter[2];
  out[i] = acc;
});
```

SRP is 64x faster than No-SRP (17 elem. filter)
"Tuning Stencil Codes in OpenCL for FPGAs", ICCD'16



Shift !!

```
cgh.single_task ([=](){
  float shft_reg[3], acc;
  shft_reg[1] = in[0];
  shft_reg[2] = in[1];
  for (int i = 1; i < N-1; i++){
    shft_reg[0] = shft_reg[1];
    shft_reg[1] = shft_reg[2];
    shft_reg[2] = in[i+1];

    acc  = shft_reg[0] * filter[0];
    acc += shft_reg[1] * filter[1];
    acc += shft_reg[2] * filter[2];
    out[i] = acc;
  }
});
```

Local memory

Use local memory when possible

Courtesy: Intel (Susannah Martin, oneAPI FPGA workshop)

Memory Type	Physical Implementation	Latency (clock cycles)	Capacity (MB)
Global	DDR	240	8000
Local	On-chip RAM	2	66
Private	On-chip RAM / Registers	2-1	0.2

```
h.single_task([=](){
  float local_buffer[N];

// Initialize local buffer
  for (size_t i = 0; i < N; i++) {
    local_buffer[i] = accessor_in[i];
  }

  compute_locally(local_buffer);

// Write result to output
  for (size_t i = 0; i < N; i++) {
    accessor_out[i] = local_buffer[i];
  }
});
```

Local Memory

Initialize from Global Memory

Compute locally

Copy the result to Global Memory

Wrap up

oneAPI eases programming for heterogeneous platforms

FPGA architecture is completely different from CPU or GPU ones

Part of your application can be optimized on this architecture

Specific optimizations will be necessary to get the most of your FPGA

To know more

oneAPI training resources
https://software.intel.com/content/www/us/en/develop/tools/oneapi/training.html

oneAPI documentation
https://docs.oneapi.com/versions/latest/dpcpp/

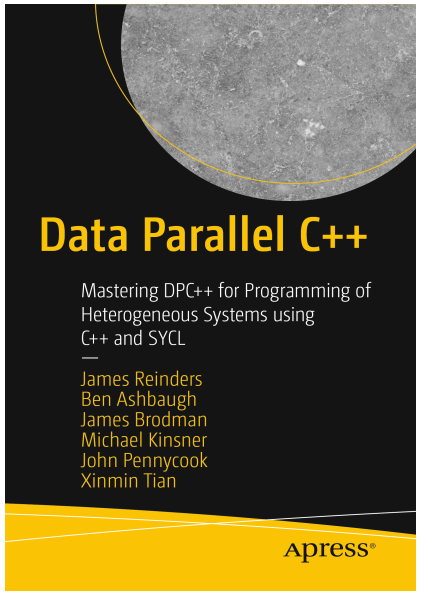
Intel® oneAPI DPC++ FPGA Optimization Guide
https://tinyurl.com/FPGAoptimizationGuide

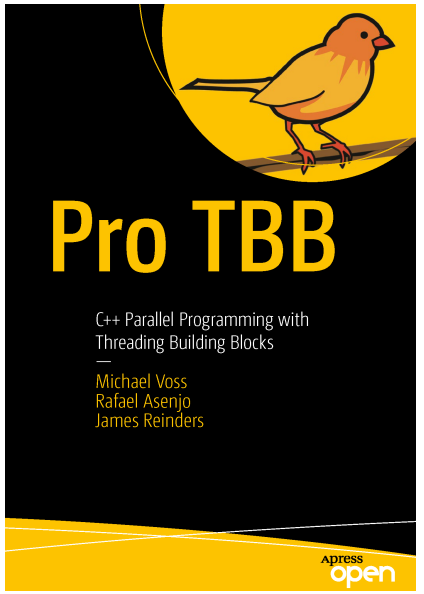
oneAPI Developer Summit:
https://www.oneapi.com/events/devcon2020/

SC'20 oneAPI tutorial:
https://sites.google.com/view/sc20-tutorial-oneapi/

SYCL.tech:
https://sycl.tech

SYC Academy:
https://github.com/codeplaysoftware/syclacademy





Agenda

- ✓ Introduction
- ✓ Lab: DevCloud
- ✓ Platform & Compilation
- ✓ FPGA optimizations
- ➢ Lab: DevCloud

Contacto

Dirección

Avda. de la industria 4, edif. 1
28108 Alcobendas | Madrid | España

Teléfono

[+34] 91 663 8683

Correo:

info@danysoft.com

Sitio Web

www.danysoft.com/intel

intel

Danysoft
Haciendo visible lo invisible

Gracias