

Creating
World
Changing
Technologies

intel®

Creating
World
Changing
Technologies

Spatial DPC++ constructs for algorithm acceleration with FPGAs

Mike Kinsner

Principal Engineer @ Intel

intel_®

Notices & Disclaimers

Performance varies by use, configuration and other factors. Learn more at www.intel.com/PerformanceIndex.

Performance results are based on testing as of dates shown in configurations and may not reflect all publicly available updates. See backup for configuration details.
No product or component can be absolutely secure.

Your costs and results may vary.

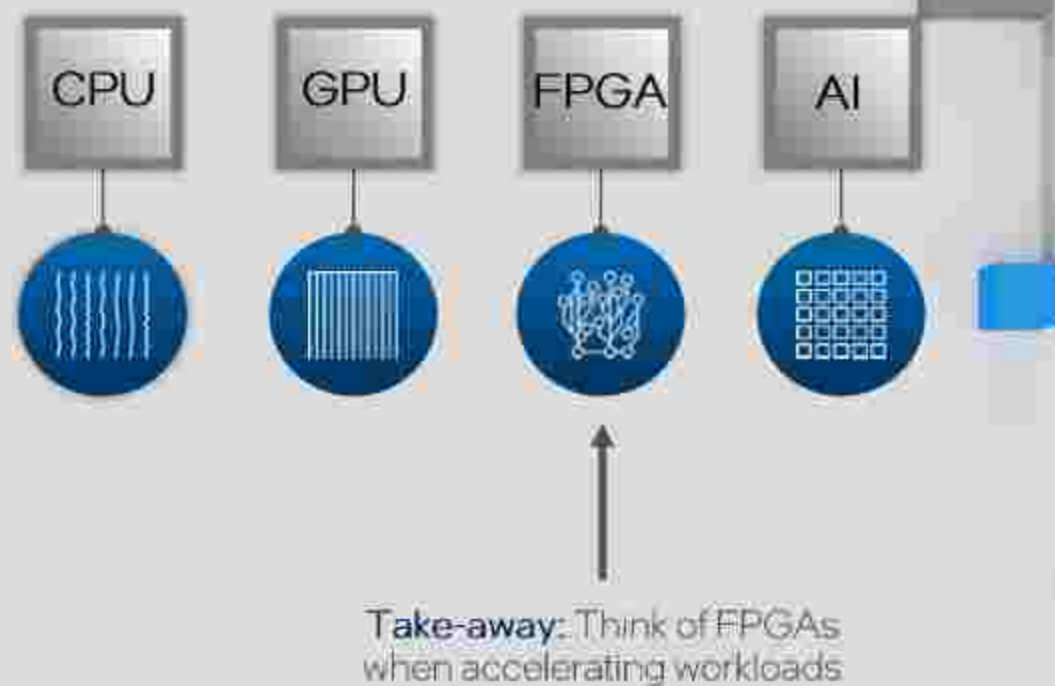
Intel technologies may require enabled hardware, software or service activation.

Intel does not control or audit third-party data. You should consult other sources to evaluate accuracy.

© Intel Corporation. Intel, the Intel logo, Xeon, Core, VTune, OpenVINO, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Khronos and the Khronos Group logo are trademarks of the Khronos Group Inc. in the U.S. and/or other countries. OpenCL and the OpenCL logo are trademarks of Apple Inc. in the U.S. and/or other countries. SYCL and the SYCL logo are trademarks of the Khronos Group Inc. in the U.S. and/or other countries. SPIR and the SPIR logo are trademarks of the Khronos Group Inc. in the U.S. and/or other countries. Other names and brands may be claimed as the property of others.

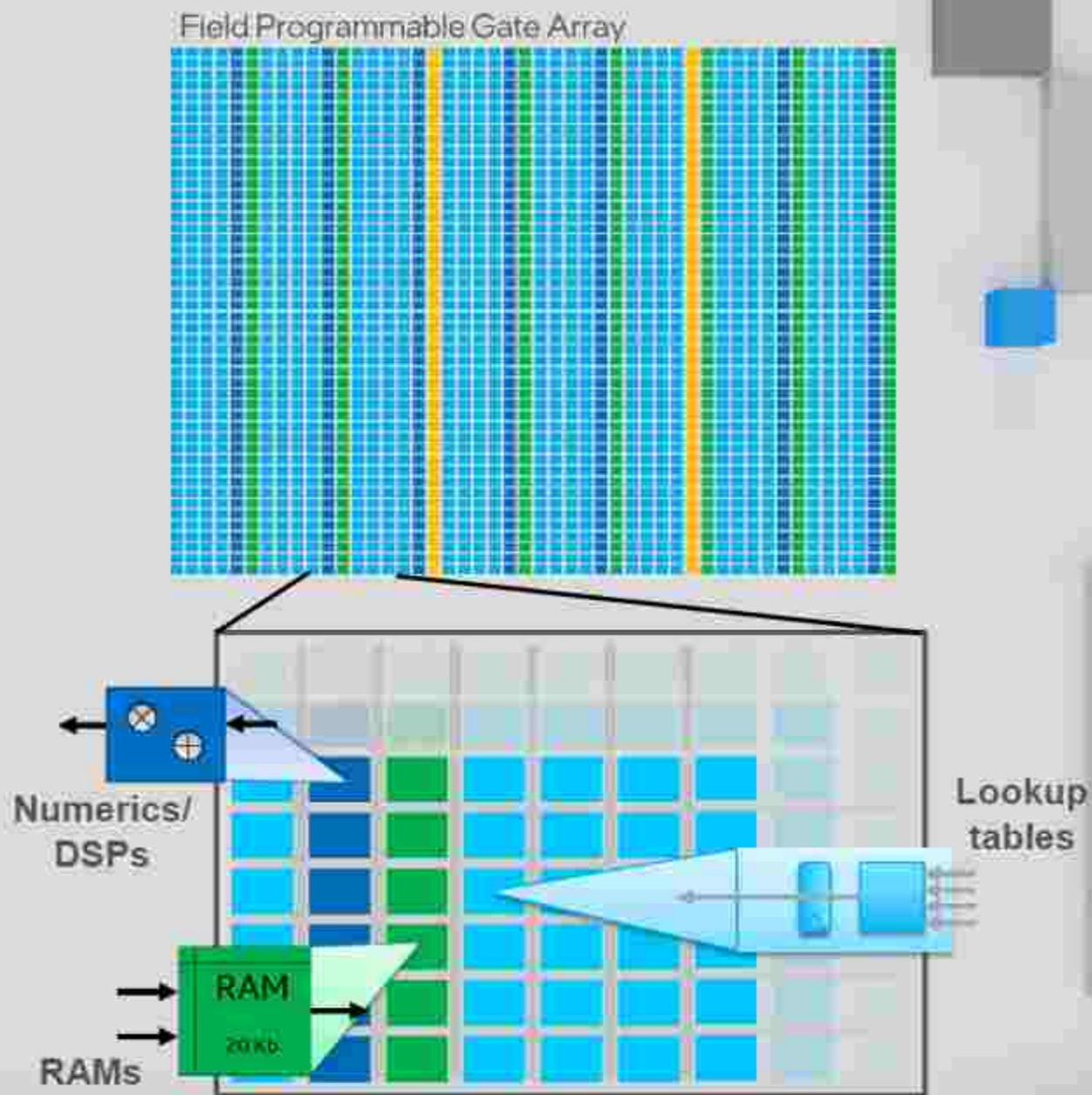
Topics

- What are FPGAs?
- How to think about FPGAs
- Data Parallel C++ for FPGAs
 - Parallelism and kernel executions
 - Pipes
 - Loop controls
 - Memory controls
 - Generating work



What is a Modern FPGA?

- A chip that you can buy today, containing:
 - Massive array of small processing units
 - 1-bit ALUs (~millions)
 - Dual-ported memories (~ten thousand)
 - Floating point MAC (~thousands)
 - Connected by mesh of programmable wires
 - Sitting inside a ring of high-speed I/O
- Common within infrastructure hardware
- Can buy on PCIe cards ready to plug in

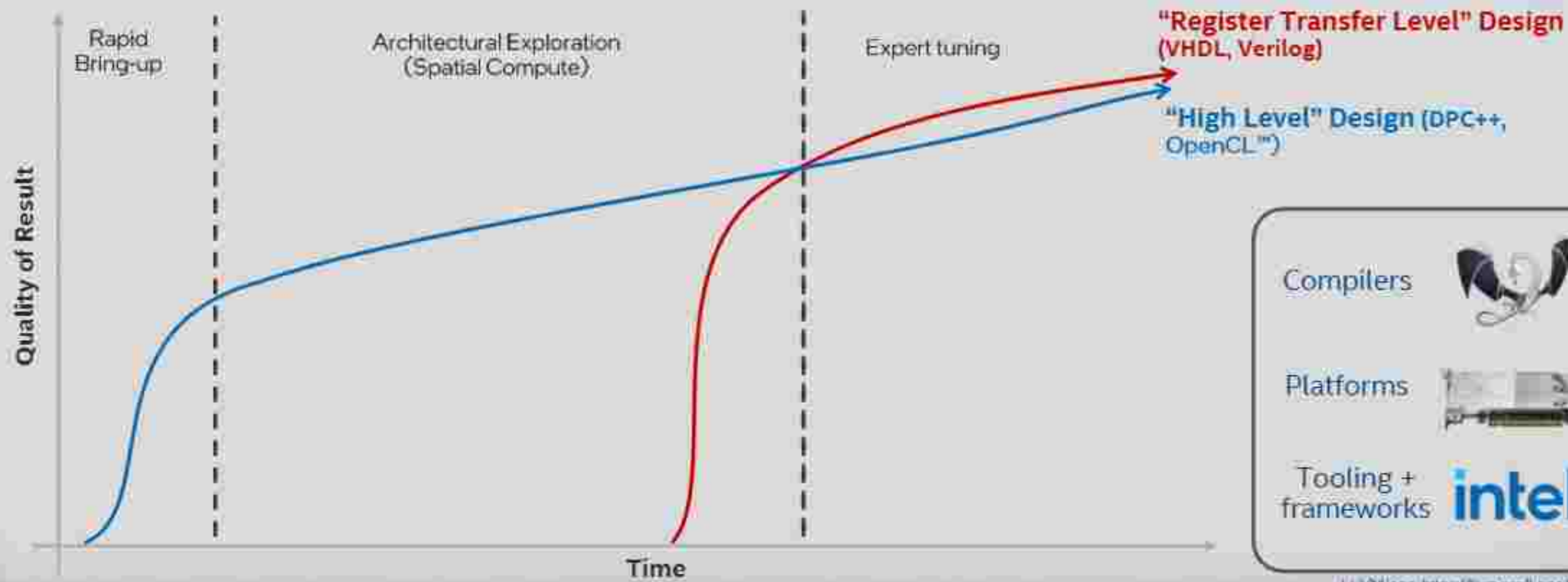


FPGA programming has evolved

More Accessible

Easier to Meet Schedules + Validate

Easier to Achieve Performance



OpenCL and the OpenCL logo are trademarks of Apple Inc. used by permission of The Khronos Group.

LLVM logo <https://llvm.org/Logo.html>

Mapping an Algorithm to Hardware

A framework for thinking about FPGAs

Data Flow Graph: The Core of Modern Compilers

```
for (int i=1; i < 10; ++i) {  
  A[i] += B[i-1] * 5;  
}
```



Data Flow Graph: The Core of Modern Compilers

```
for (int i=1; i < 10; ++i) {  
    A[i] += B[i-1] * 5;  
}
```

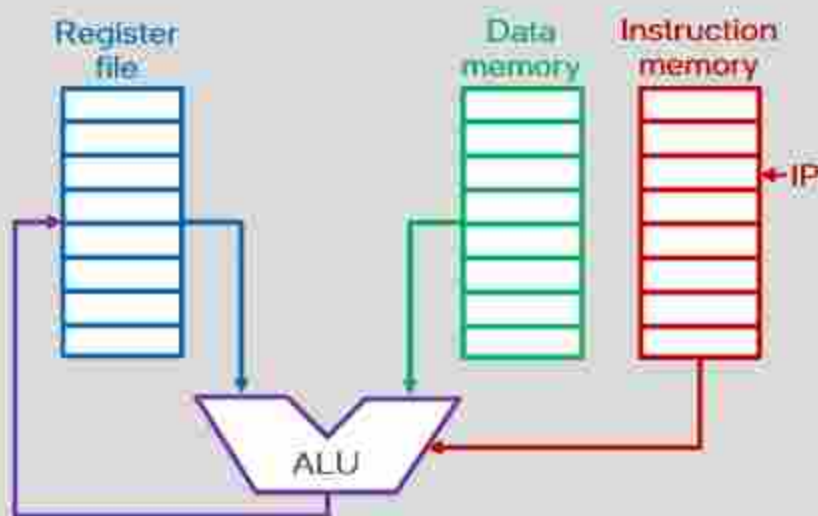


Hardware?

Executing in Silicon: Two key options

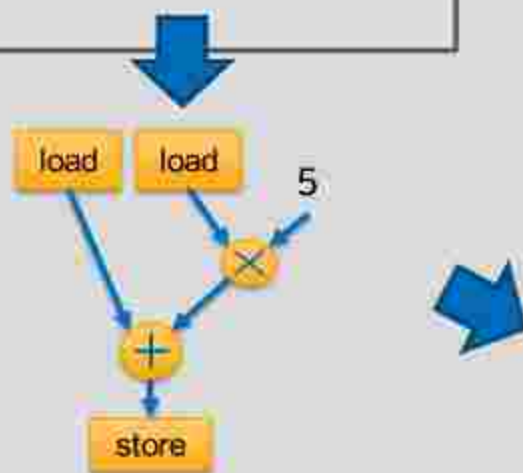
Option 1:

Instruction set architecture
(ISA) machine



Execute consecutive instructions on **same** hardware via **time multiplexing**

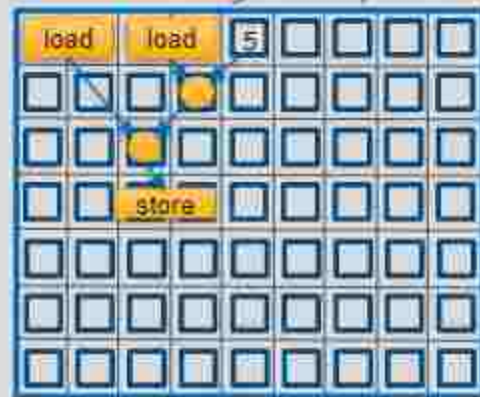
```
for (int i=1; i < 10; ++i) {  
    A[i] += B[i-1] * 5;  
}
```



Option 2:

Materialize data flow graph
directly on reconfigurable
spatial hardware

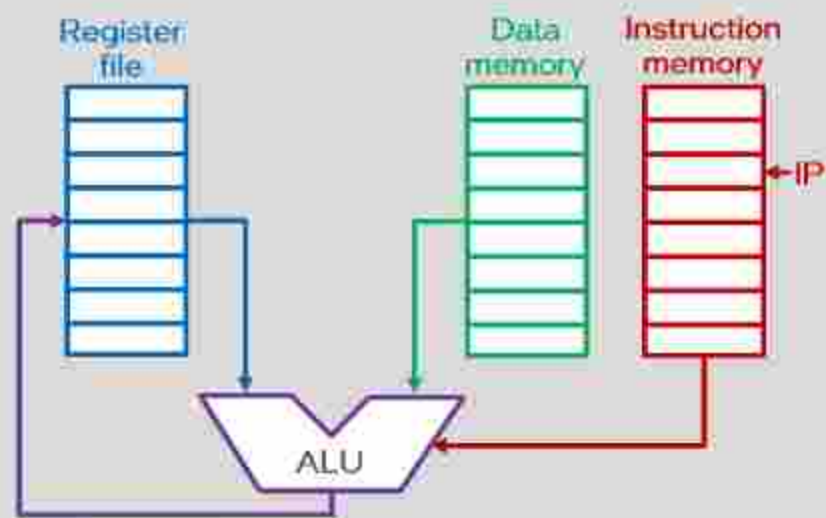
Hardware with configurable spatial "regions"



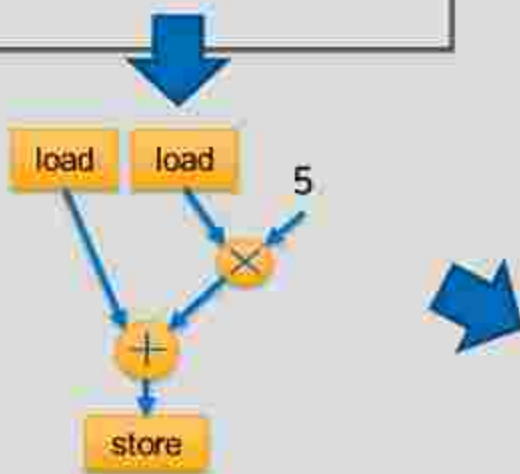
Execute instructions on **different** hardware by **specializing device regions in pipeline**

Many Architectural Tradeoffs

Instruction Set Architecture



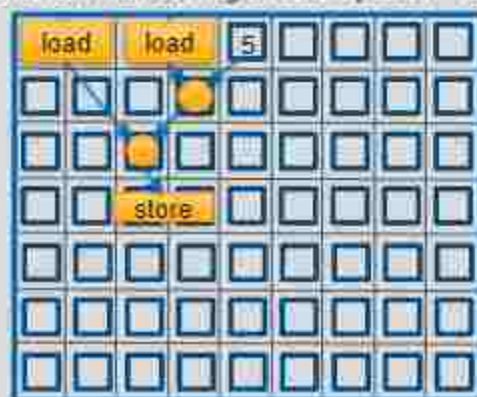
```
for (int i=1; i < 10; ++i) {  
  A[i] += B[i-1] * 5;  
}
```



Spatial Architecture

Materialize data flow graph directly on reconfigurable spatial hardware

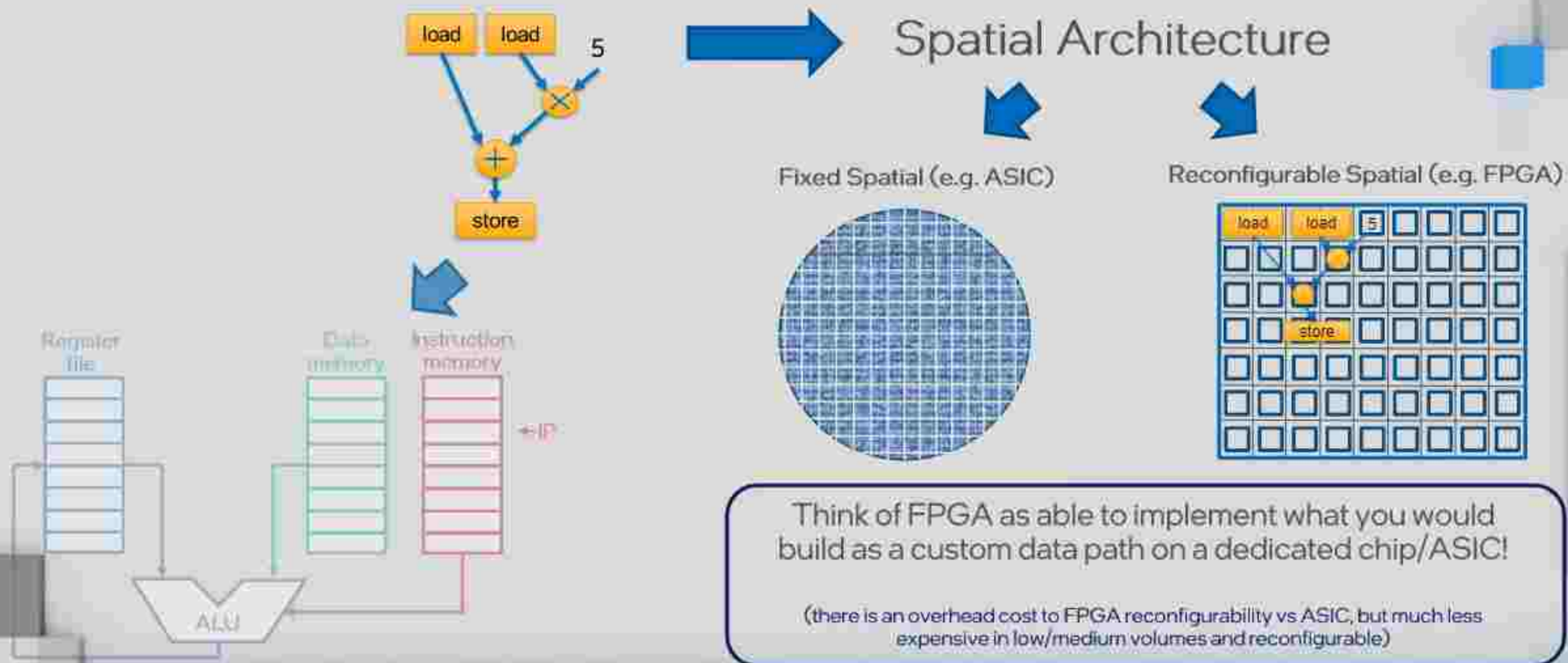
Hardware with configurable spatial "regions"



- + Easy to program
- + General purpose at runtime
- Not all hardware may be used every clock cycle

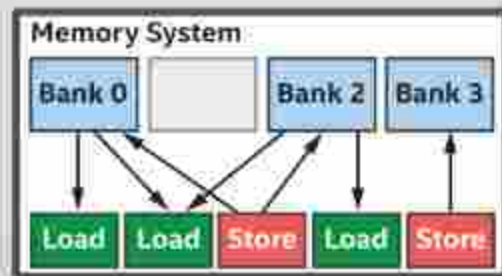
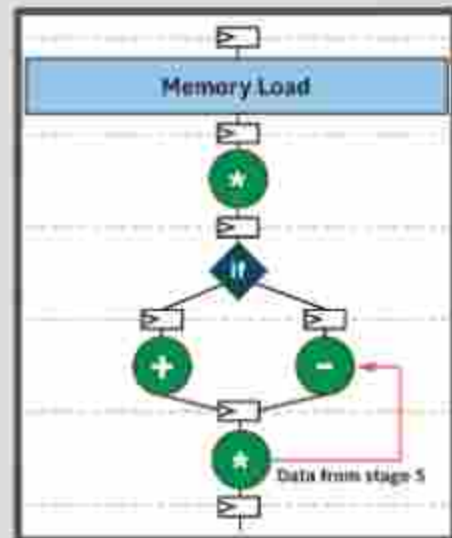
- + Excellent Perf, Perf/Watt from specialization
- Previously inaccessible by SW developers
- Data path not typically runtime context switched

Think of FPGAs as Reconfigurable Custom Chips



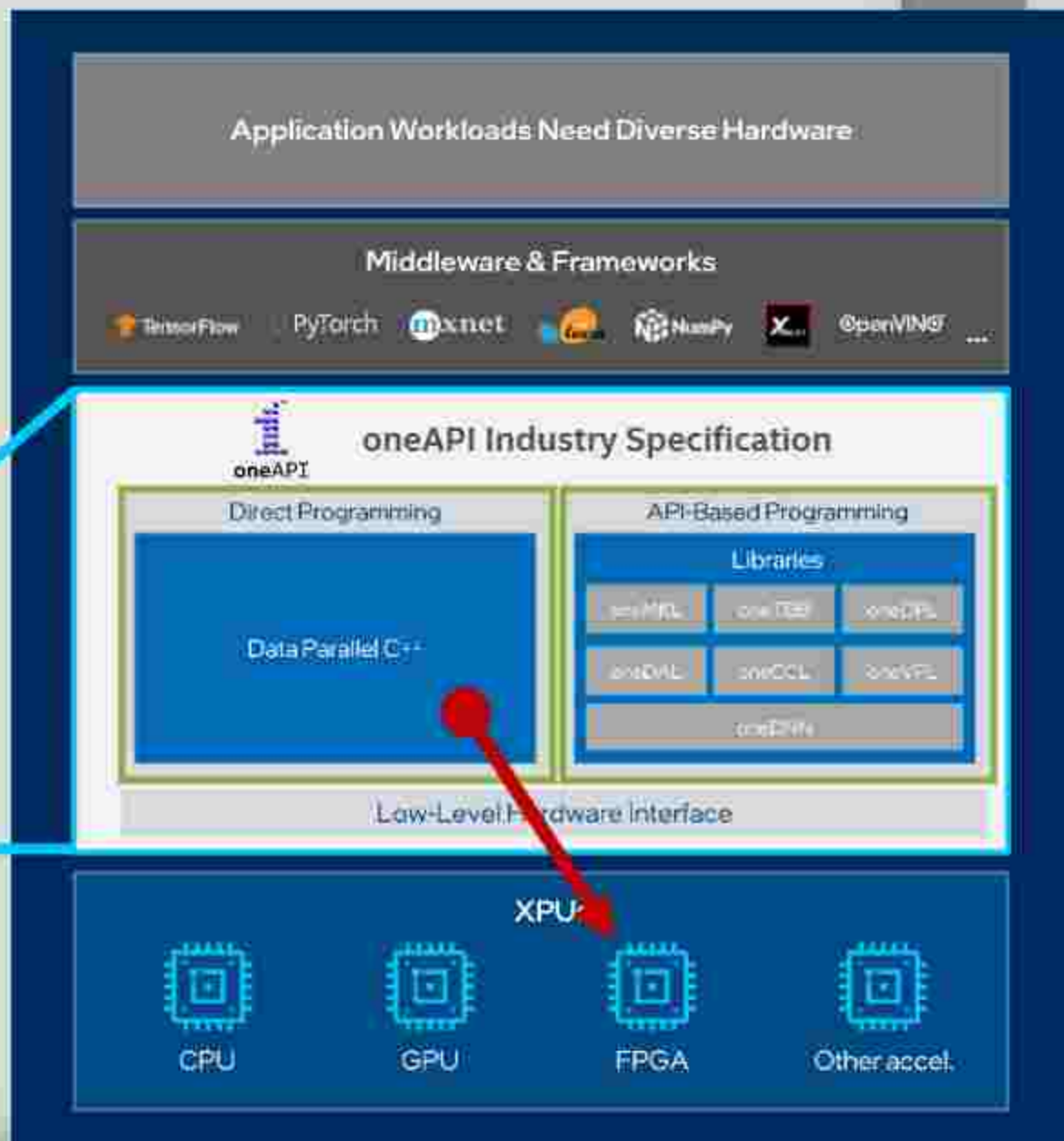
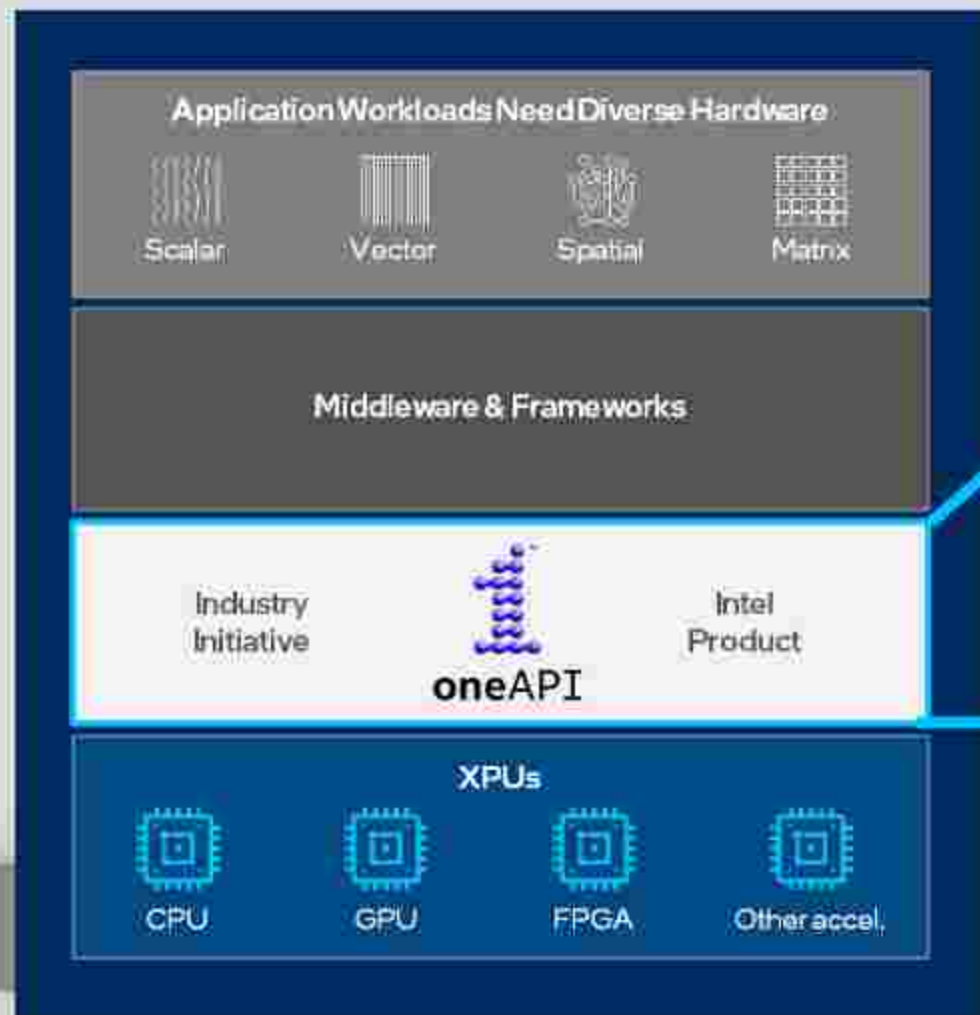
FPGA Value Characteristics

1. **Spatial** architecture – many data flow algorithms map well
 - Data dependences across parallel work
 - Streaming and graph processing patterns
 - Enables some algorithms, makes others tractable to express
2. Rich **I/O**
 - Network, memory, custom interfaces and protocols
 - Very low + deterministic latencies
3. Large distributed on-chip **memory** BW
 - Custom topologies tuned to algorithm



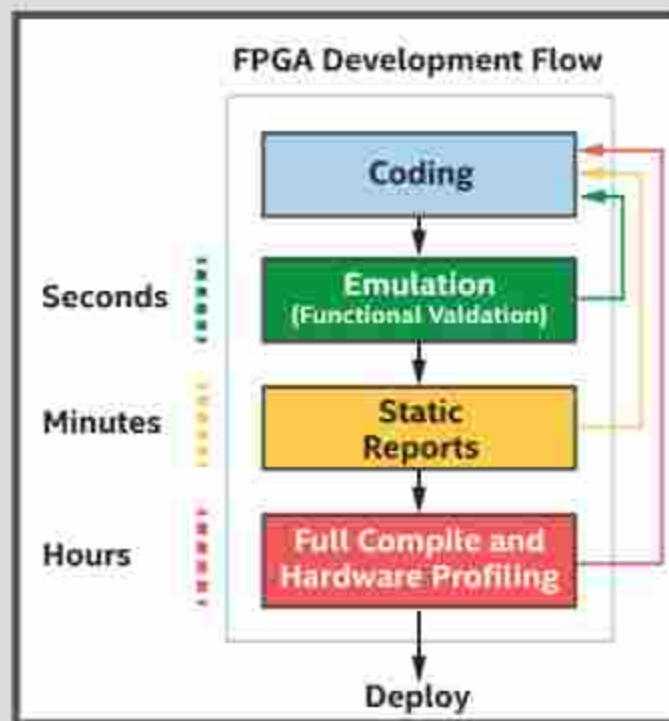
Data Parallel C++ (DPC++) for FPGAs

DPC++ can target FPGA



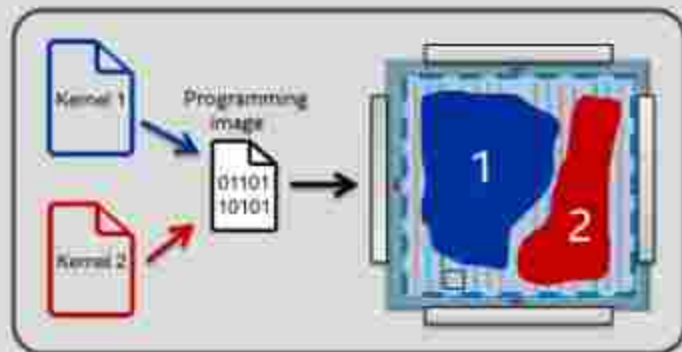
Development Flows

- Compiling to FPGA hardware takes longer than most SW developers are used to
 - Tools + reporting allow most development to be done without HW compilation



DPC++ on FPGAs: Parallelism!

1. **Kernel level:** Kernels can execute in parallel

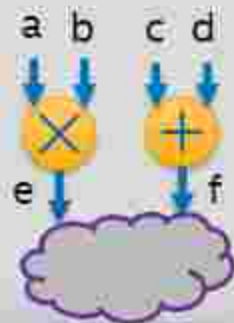


2. **Within a kernel:** Instructions are executed at the same time in various ways

- Implemented in different areas of chip, so not competing for resources like ALUs

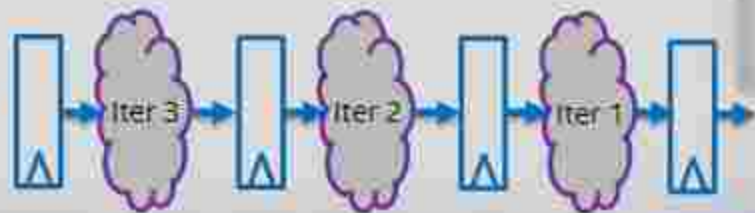
A. Scheduling:

```
e = a + b;  
f = c + d;
```



B. Pipelining:

```
for (int i=1; i < 10; ++i) {  
  A[i] += B[i-1] * 5;  
}
```



SYCL Runtime Kernel Scheduling

↓ = data dependence

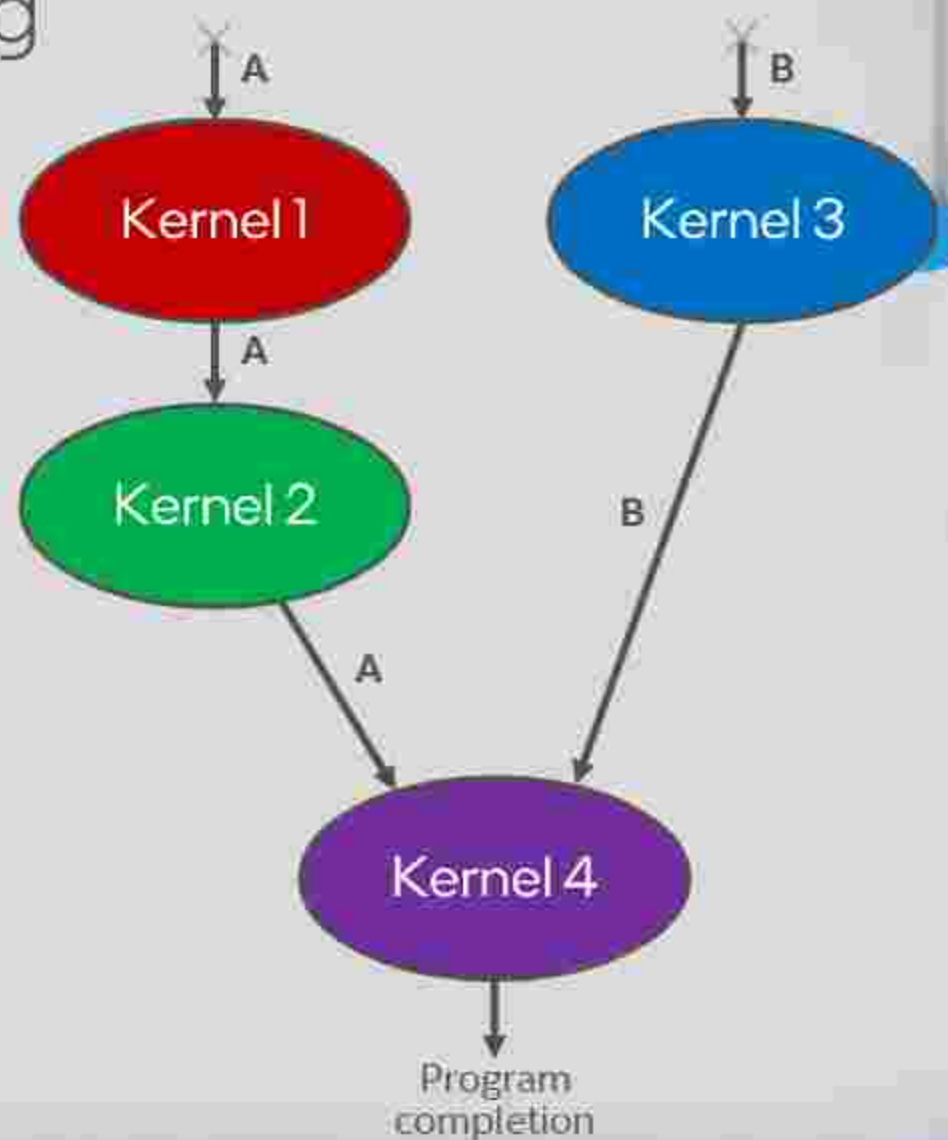
```
int main() {  
  buffer<int> A[ rng ], B[ rng ];  
  queue Q;  
  
  Q.submit([&](handler& h) {  
    accessor out(A, h, write_only);  
    h.parallel_for(rng, [=](auto idx) {  
      out[idx] = idx[0]; }); });  
  
  Q.submit([&](handler& h) {  
    accessor out(A, h, write_only);  
    h.parallel_for(rng, [=](auto idx) {  
      out[idx] = idx[0]; }); });  
  
  Q.submit([&](handler& h) {  
    accessor out(B, h, write_only);  
    h.parallel_for(rng, [=](auto idx) {  
      out[idx] = idx[0]; }); });  
  
  Q.submit([&](handler& h) {  
    accessor in(A, h, read_only);  
    accessor inout(B, h, read_write);  
    h.parallel_for(rng, [=](auto idx) {  
      inout[idx] *= in[idx]; }); });  
}
```

} Kernel 1

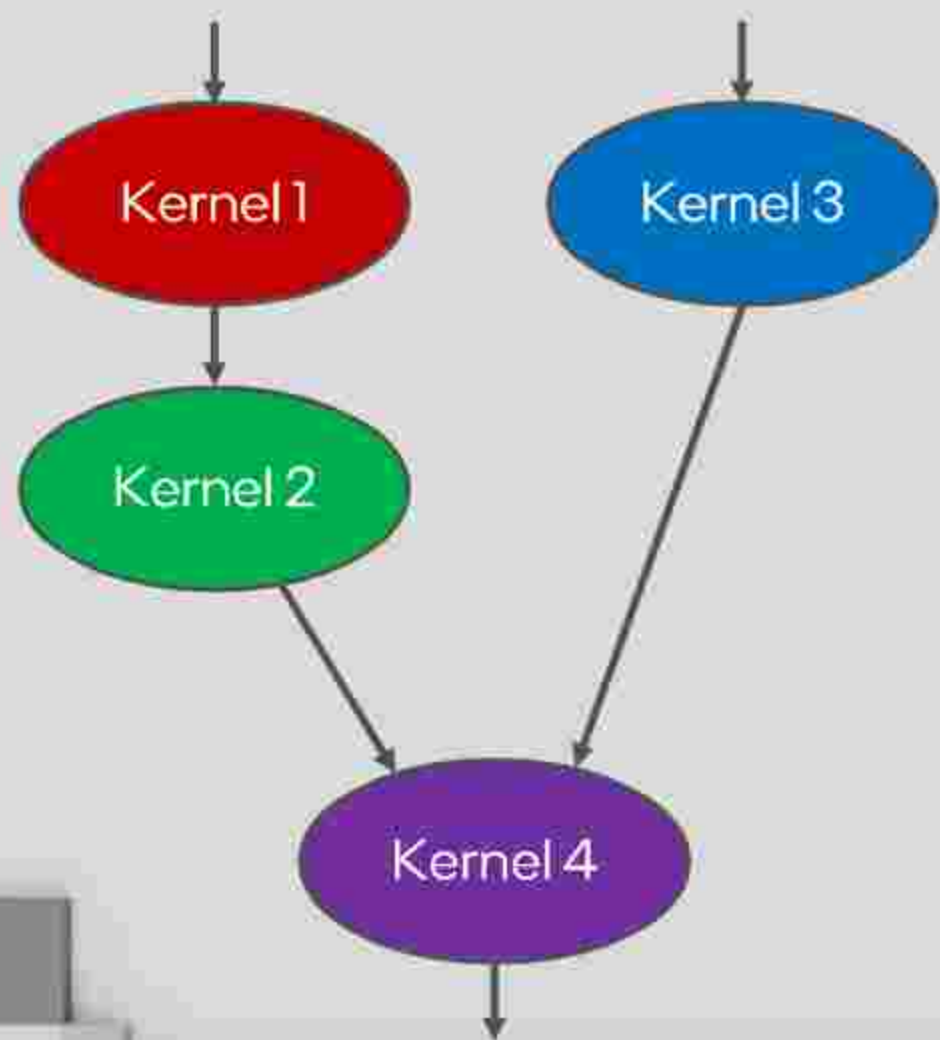
} Kernel 2

} Kernel 3

} Kernel 4

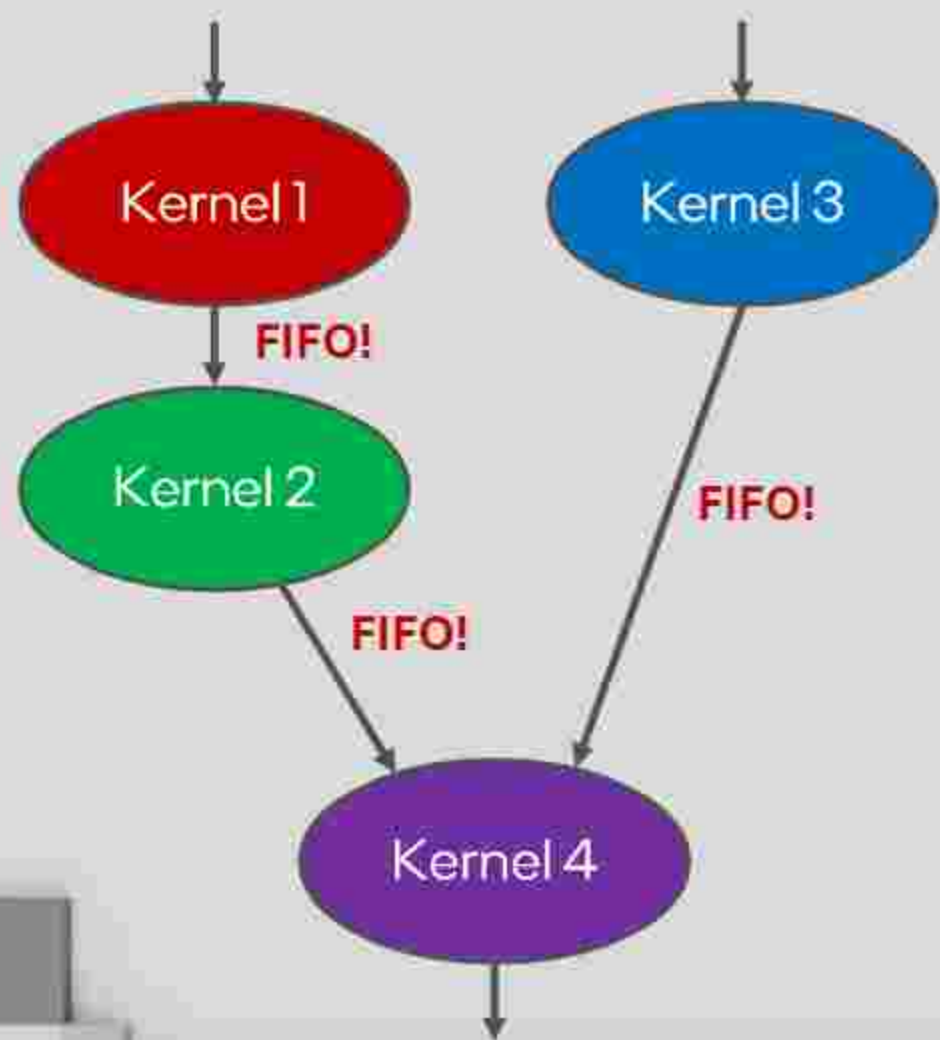


SYCL Runtime Kernel Scheduling



- The SYCL runtime graph model
 - A data flow graph
 - Based on data or control dependencies
 - Coarse grained dependencies/sharing

Data Flow Natural Fit on FPGAs



- The SYCL runtime graph model
 - A data flow graph
 - Based on data or control dependencies
 - Coarse grained dependencies/sharing
- Leverage same model with FIFOs as edges
 - Kernels execute concurrently to minimize storage on edges/in FIFOs

FIFO = First In First Out buffer

Common spatial pattern: "Pipes"

```
// Defining a type alias is the recommended practice
```

```
using my_pipe = pipe<class some_pipe, int>;
```

```
auto R = range<1>{1024};
```

```
myQueue.submit([&](handler& cgh) {  
    auto read_acc = ...
```

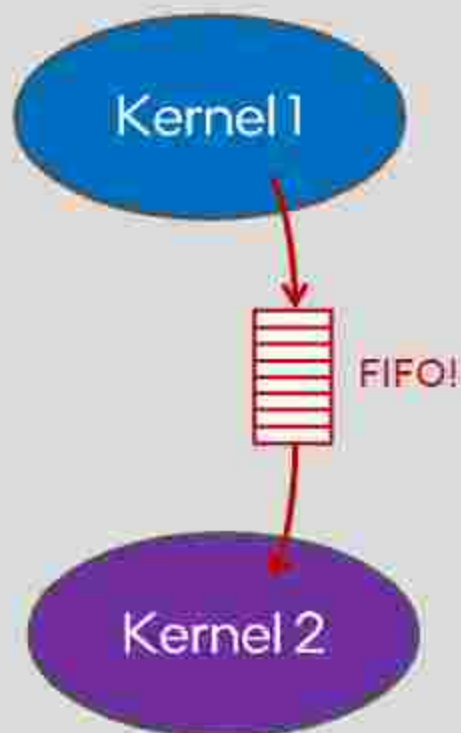
```
    cgh.parallel_for(R, [=](auto idx) {  
        my_pipe::write( read_acc[idx] );  
    });  
});
```

} Kernel 1

```
myQueue.submit([&](handler& cgh) {  
    auto write_acc = ...
```

```
    cgh.parallel_for(R, [=](auto idx) {  
        write_acc[idx] = my_pipe::read();  
    });  
});
```

} Kernel 2



FIFO (pipe) very efficient on FPGA:
Avoids memory accesses + address
computation, stays on-chip

Pipe connectivity options

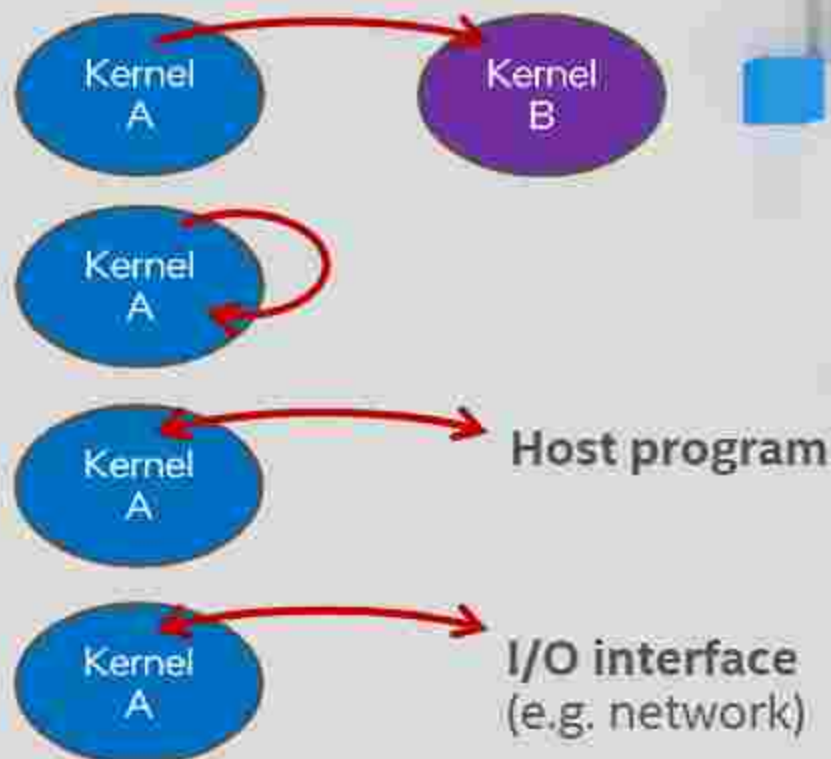
- Types of connectivity:

1. Cross kernel: Kernel A \Rightarrow Kernel B

2. Intra-kernel: Kernel A \Rightarrow Kernel A

3. Host pipe: Kernel A \Leftrightarrow host program

4. I/O pipe: Kernel A \Leftrightarrow I/O peripheral



Memory Controls

- FPGA memory systems are customized to your application
 - There are controls to let you customize beyond the compiler's extensive optimizations
 - Only required for advanced tuning. Use alongside compiler reports

```
struct State {  
    [[intel::fpga_memory]] int array[100];  
    [[intel::fpga_register]] int reg[4];  
};  
  
cgh.single_task<class test>([=] {  
    struct State S1;  
    [[intel::fpga_memory]] struct State S2;  
    // some uses  
});
```

bank_bits	[[intel::bank_bits(b ₀ ,b ₁ ,...,b _n)]]
bankwidth	[[intel::bankwidth(N)]]
doublepump	[[intel::doublepump]]
force_pow2_depth	[[intel::force_pow2_depth(N)]]
max_replicates	[[intel::max_replicates(N)]]
fpga_memory	[[intel::fpga_memory("impl_type")]]
merge	[[intel::merge("key", "direction")]]
numbanks	[[intel::numbanks(N)]]
private_copies	[[intel::private_copies(N)]]
fpga_register	[[intel::fpga_register]]
simple_dual_port	[[intel::simple_dual_port]]
singlepump	[[intel::singlepump]]

Loop Controls

- Loop optimizations are performance critical in most spatial applications
 - There are controls to let you customize beyond what the compiler decides
 - Only required for advanced tuning. Use alongside compiler reports

```
// No loop-carried dependencies for
// accesses to arrays A and B
[[intel::ivdep]]
for (int i = 0; i < N; i++) {
    A[i] = A[i - X[i]];
    B[i] = B[i - Y[i]];
}

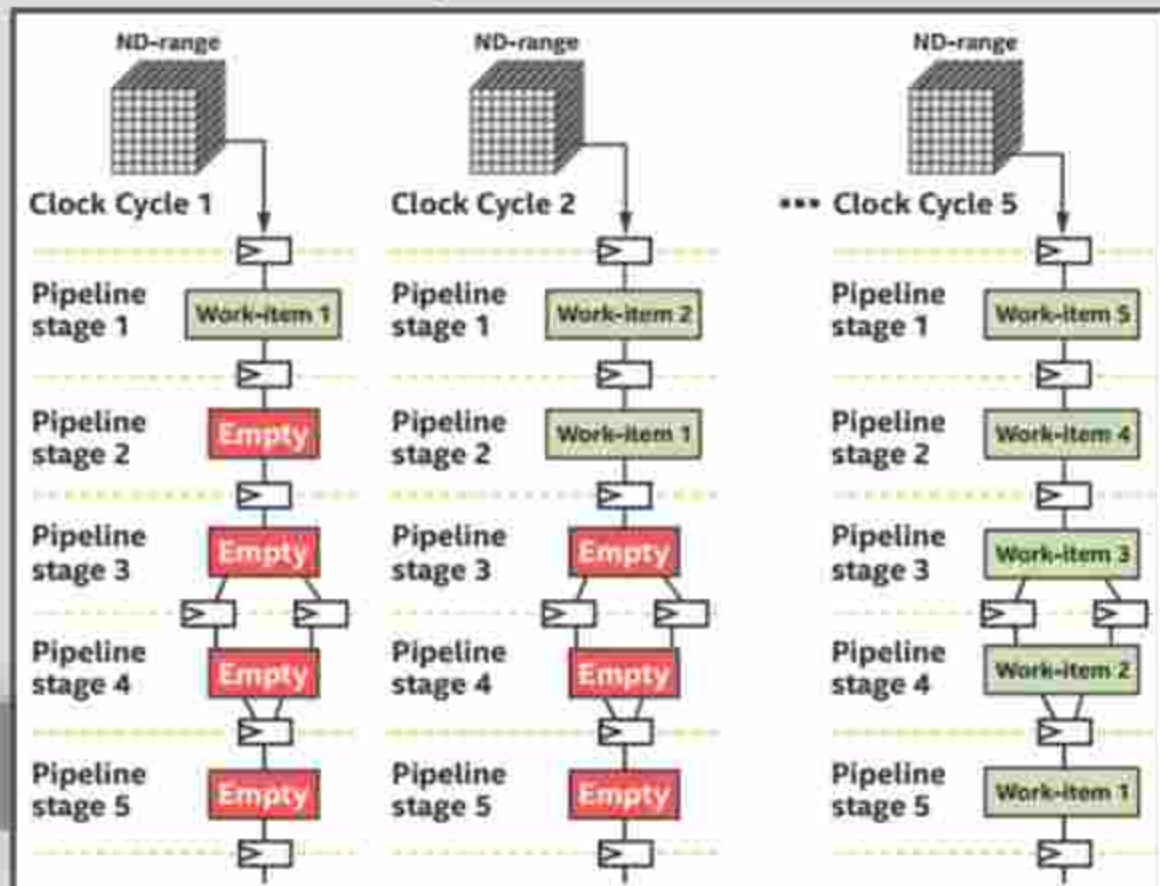
#pragma unroll
for(i = 0 ; i < 5; i++){
    a[i] += 1;
}
```

disable_loop_pipelining	[[intel::disable_loop_pipelining]]
initiation_interval	[[intel::initiation_interval(n)]]
ivdep	[[intel::ivdep]] [[intel::ivdep(safelen)]] [[intel::ivdep(array)]] [[intel::ivdep(array, safelen)]] [[intel::ivdep(safelen, array)]]
loop_coalesce	[[intel::loop_coalesce(N)]]
max_concurrency	[[intel::max_concurrency(n)]]
max_interleaving	[[intel::max_interleaving(n)]]
speculated_iterations	[[intel::speculated_iterations(N)]]
unroll	#pragma unroll #pragma unroll N

Deep Spatial Pipelines: Generating the Work

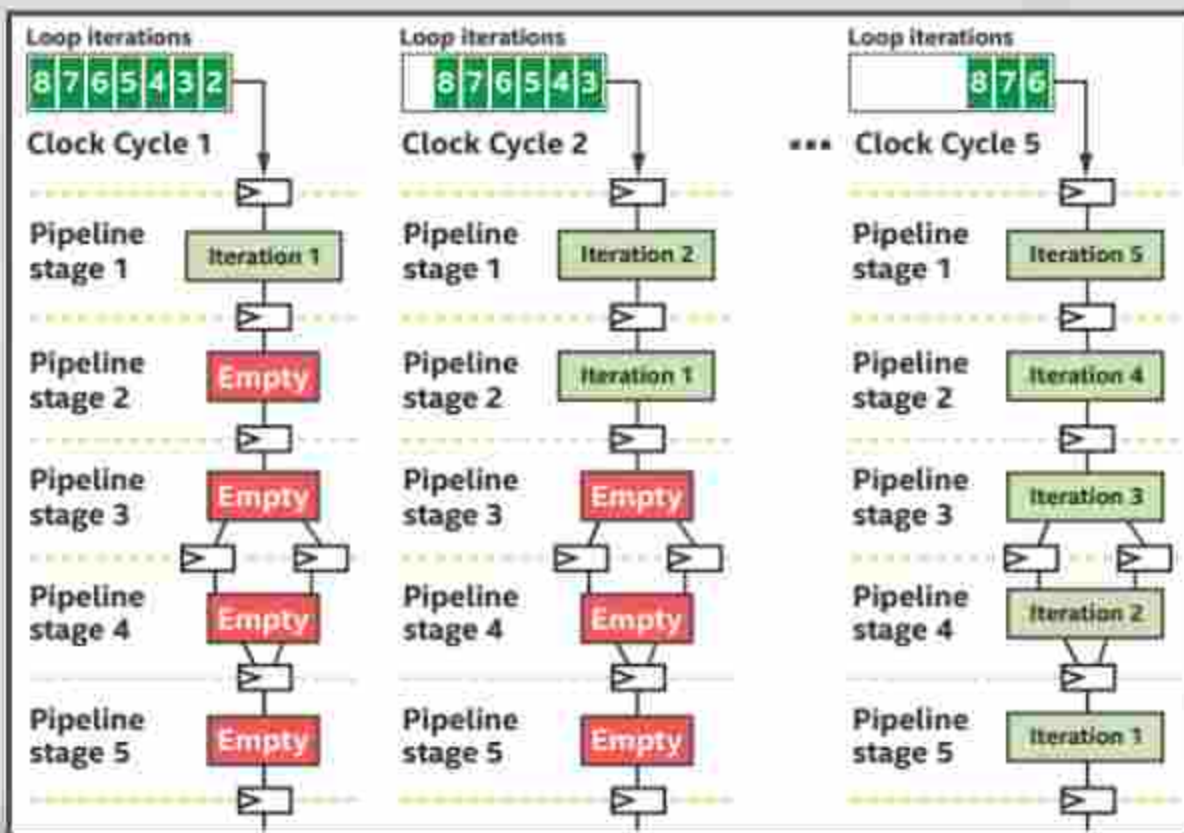
ND-range:

```
h.parallel_for({8,8,8}, [=](auto I) {  
    output[I] = in[I] * 5;  
});
```



Loop:

```
h.single_task([=] {  
    for (int i=2; i < size; i++) {  
        A[i] = A[i-1] * foo(A[i-2]);  
    }  
});
```



Conclusion

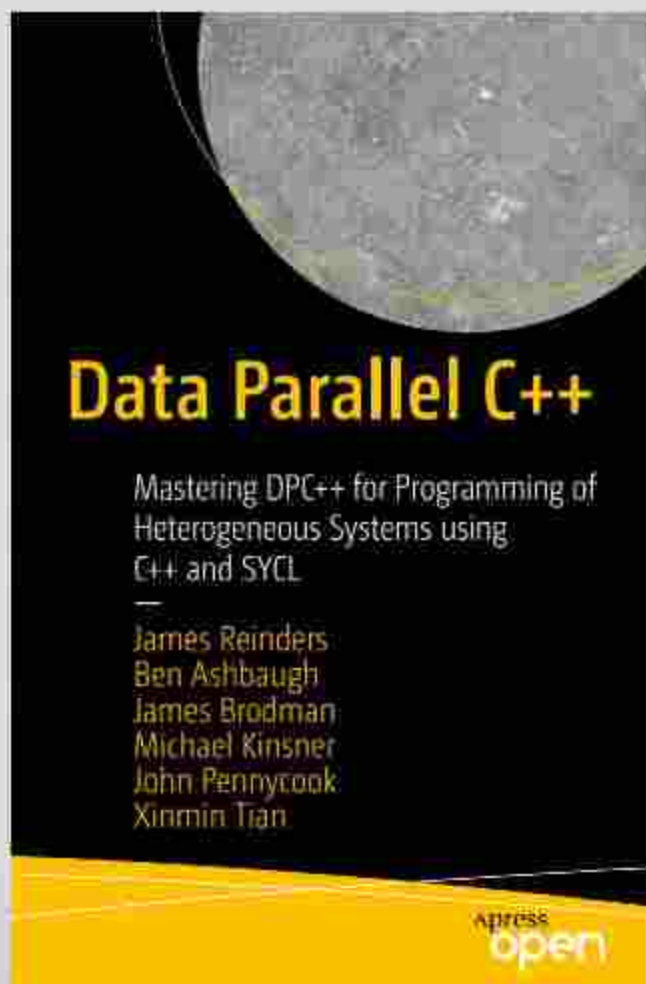
A Heterogeneous World

- Architectural diversity continues to increase
- oneAPI is tackling heterogeneous software challenges through open standards and open development

oneAPI includes the ability to target and optimize designs for FPGAs!

- FPGAs offer advantages in many applications and algorithms
 - Realizing the potential requires structuring an application in a spatial-friendly way
 - Architecture-specific formulation is needed across accelerators today
 - Spatial/FPGA language constructs and controls are exposed in DPC++

A Next Step



- Free in PDF form! Can purchase print form
 - www.apress.com/book/9781484255735
- First book on DPC++ or SYCL
 - Has a chapter on FPGA programming in DPC++

intel®