Introduction to DPC++ Programming for FPGA

A part of the DPC++ Tutorial Series

Prof. Yan Luo

Acknowledgement
This work is supported by Intel Corporation 2020-2021

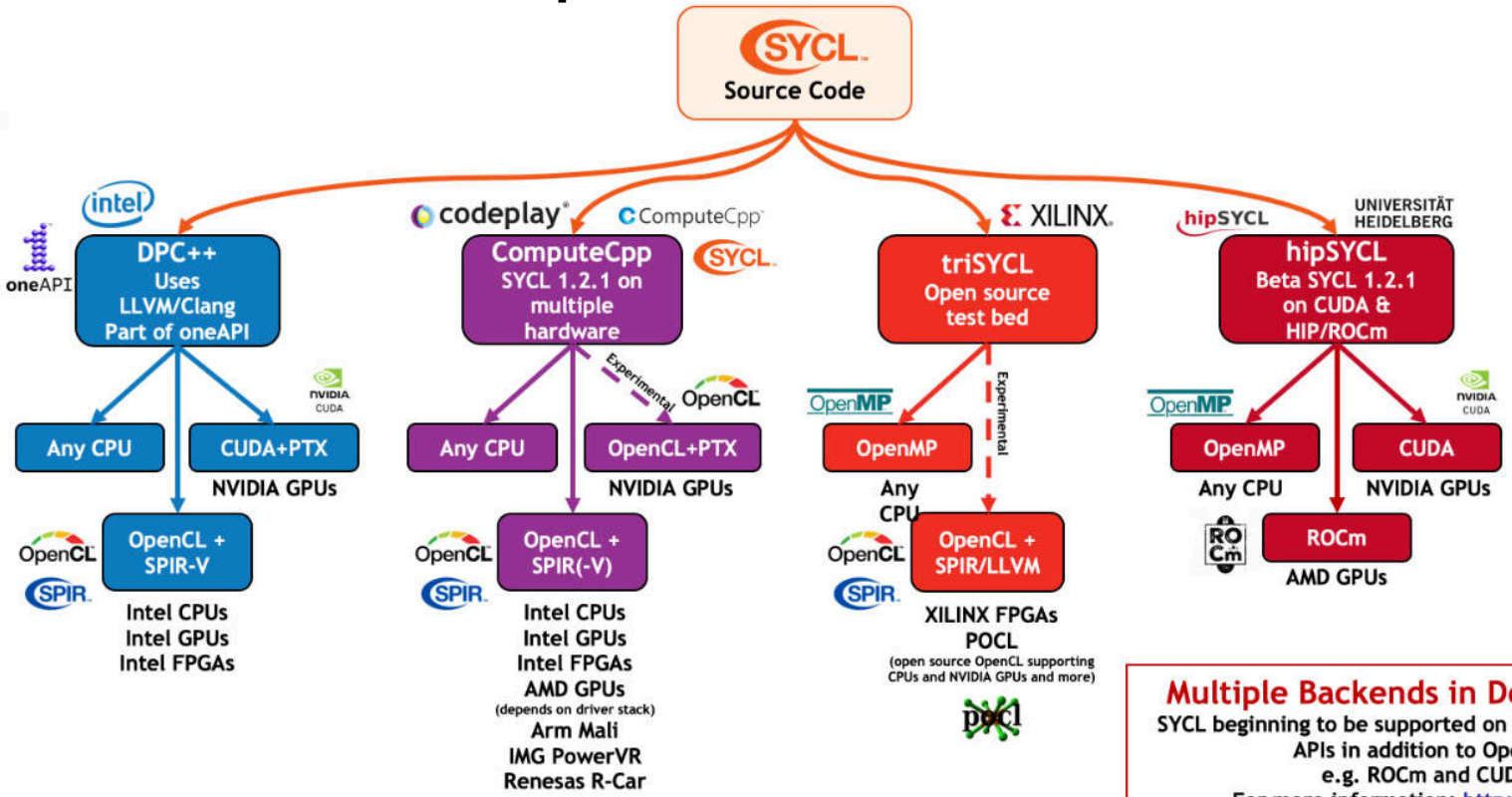
UMASS LOWELL
Learning with Purpose

Data Parallel C++

- A high-level language for data parallel programming
- Based on modern C++
- Single source for heterogeneous computing architectures
- Offloading computing to accelerators (e.g. FPGA and GPU)
- Speedup on data parallel workloads
  - Algorithm and parallelism analysis
  - Data/task decomposition
  - Architecture oriented performance optimization (e.g. for FPGA)

UMASS LOWELL
Learning with Purpose

DPC++: an Implementation of SYCL



Source: https://www.khronos.org/sycl/

A DPC++ Example

```
1  #include <CL/sycl.hpp>
2  #include <iostream>
3
4  constexpr int num=16;
5  using namespace sycl;
6
7  int main() {
8    auto r = range{num};
9    buffer<int> a{r};
10
11   queue{}.submit([&](handler& h) {
12     accessor out{a, h};
13     h.parallel_for(r, [=](item<1> idx) {
14       out[idx] = idx;
15     });
16   });
17
18   host_accessor result{a};
19   for (int i=0; i<num; ++i)
20     std::cout << result[i] << "\n";
21 }
```

Header Files

namespace scope

device queue & command

SYCL buffer declaration

kernel function on device

Lambda function

host access results in buffer

Matrix Multiplication : How to Think in Parallel?

A part of the DPC++ Tutorial Series

Prof. Yan Luo

Acknowledgement
This work is supported by Intel Corporation 2020-2021

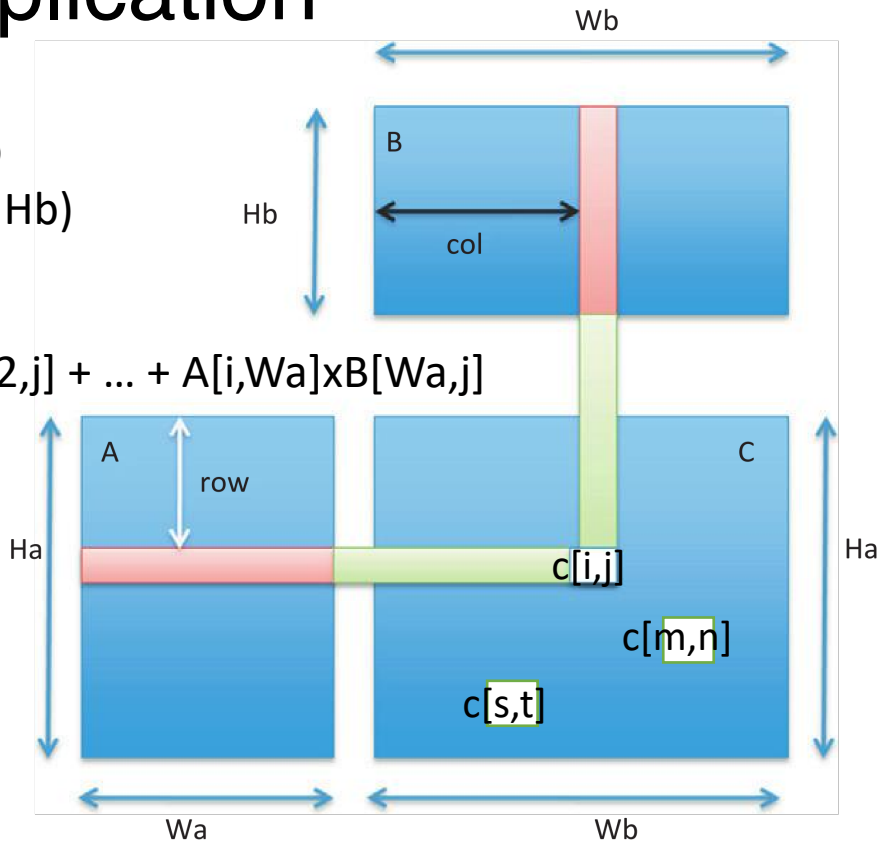
UMASS LOWELL
Learning with Purpose

Matrix Multiplication

A x B = C

A is Wa x Ha
B is Wb x Hb
(Note: Wa = Hb)
C is Wb x Ha

C[i,j] = A[i,1]xB[1,j]+A[i,2]xB[2,j] + … + A[i,Wa]xB[Wa,j]



A C++ Implementation

```
// iterate over the rows of Matrix A
for (int i = 0; i < Ha; i++)
    // iterate over the columns of Matrix B
    for (int j = 0; i < Wb; j++) {
        C[i][j] = 0;
        // element-wise multiplication and accumulation
        for (int k = 0; k < Wa; k++)
            C[i][j] += A[i][k] * B[k][j];
    }
```

A DPC++ Implementation

```
q.submit([&](handler &h) {
    auto A = a_buf.get_access<access::mode::read>(h);
    auto B = b_buf.get_access<access::mode::read>(h);
    auto C = sum_buf.get_access<access::mode::write>(h);

    range<2> num_items{a_rows, b_columns};

    h.parallel_for<class MMpara>(num_items, [=](id<2> i) {
       size_t row = i[0], col = i[1];

       C[row][col] = 0;
       for (size_t k = 0; k < Wa; k++)
           C[row][col] += A[row * Wa + k] * B[k * Wb + col];
    });
});
```

create accessors

set up problem size

lambda function for device

Demonstration

Compilation and execution of Matrix Multiplication example on Intel FPGA DevCloud

UMASS LOWELL
Learning with Purpose

A Very Brief Introduction to FPGA Design Concepts

A part of the DPC++ Tutorial Series

Prof. Yan Luo

Acknowledgement
This work is supported by Intel Corporation 2020-2021

UMASS LOWELL
Learning with Purpose

Agenda

- Introduction to FPGA Architecture
- Concepts of FPGA Hardware Design
- Mapping Source Code to Hardware Datapath
- Scheduling
- Parallelism Models to FPGA Hardware
- Memory Types
- Trivia

Some materials used in this presentation are based on Intel® OneAPI DPC++ FPGA Optimization Guide

UMASS LOWELL
Learning with Purpose

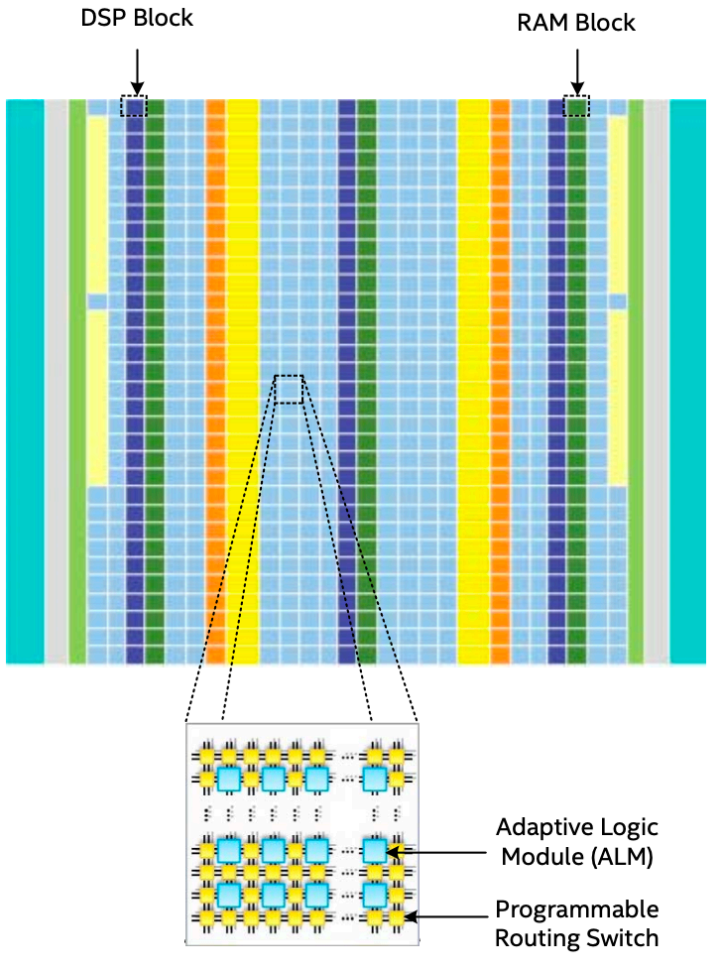
FPGA vs CPU

- FPGA does not have a fixed datapath
  - that is why it is “field programmable” !
- “Program” the hardware resources
  - Your have a lot more control on how your design is mapped
- Function as accelerators to offload intensive computing
- Design methodologies
  - Hardware Description Language
  - High level language (like DPC++)

UMASS LOWELL
Learning with Purpose

FPGA Architecture

- Adaptive Logic Module
- RAM block
- DSP block
- Programmable routing switch



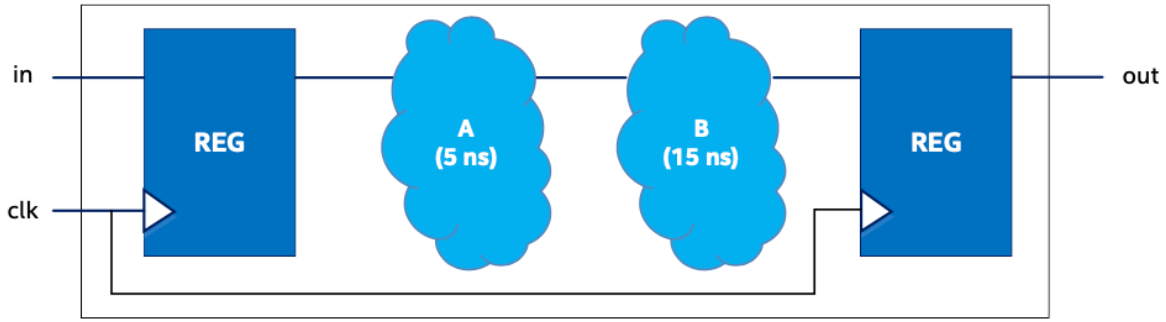
FPGA Hardware Design Concepts

- Maximum frequency f_{MAX}
- Latency
- Pipelining
- Throughput
- Datapath
- Control path
- Occupancy

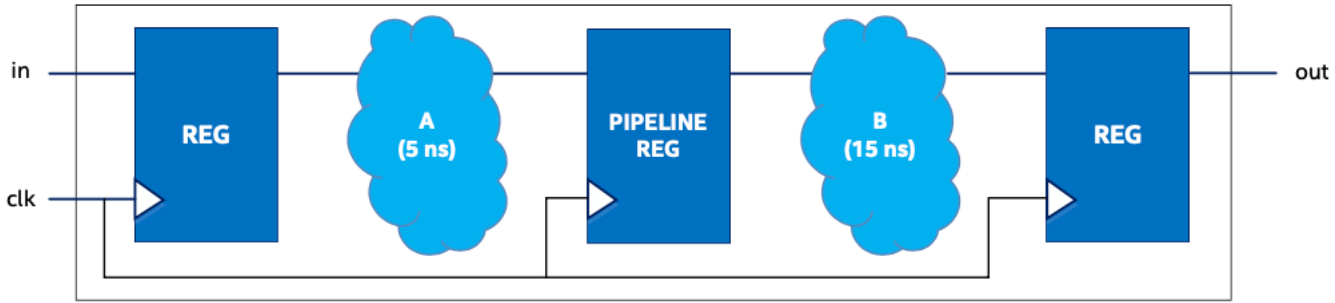
UMASS LOWELL
Learning with Purpose

Pipelining to improve f_{MAX}

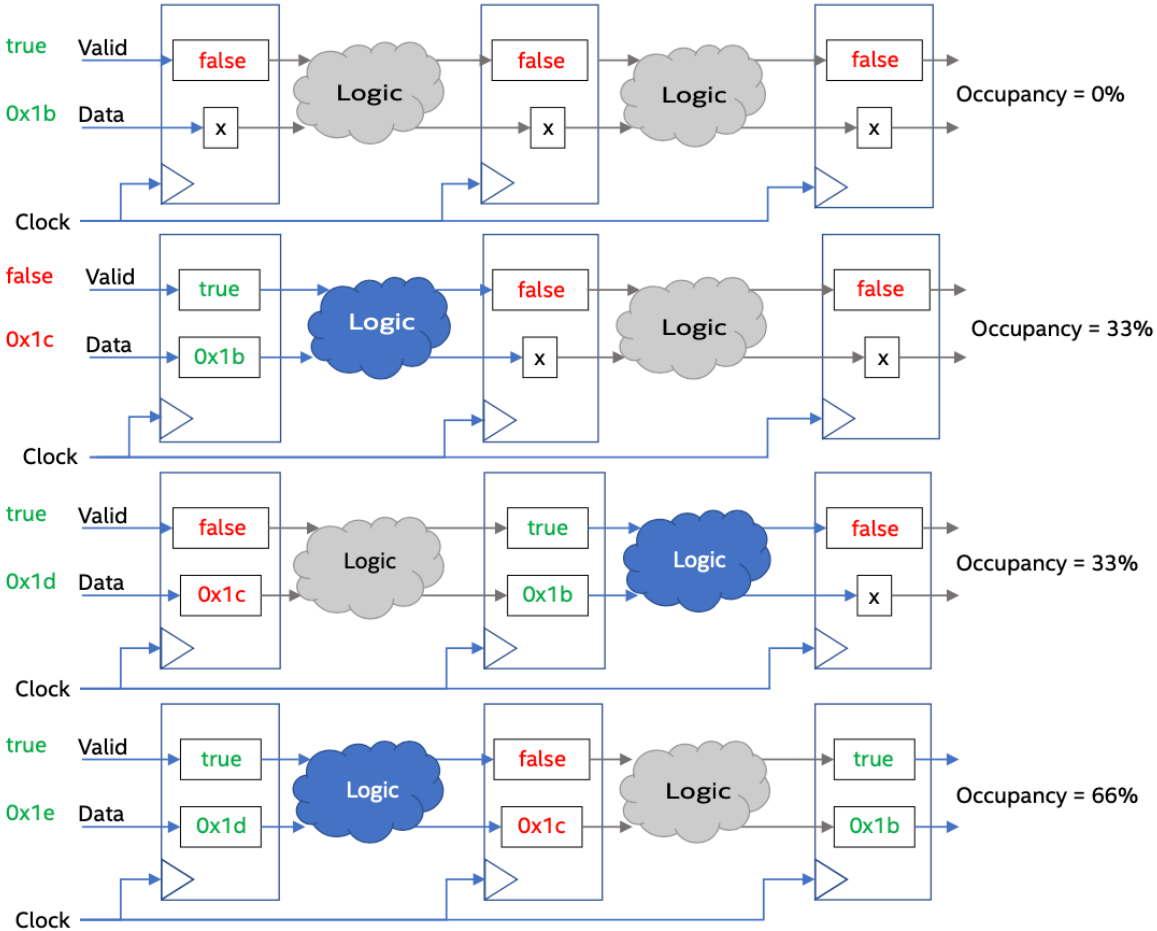
unpipelined
f_{MAX} = 50MHz



pipelined
f_{MAX} = 66.7MHz



Occupancy



DPC++ Design Analysis (I): Analyze FPGA Early Image

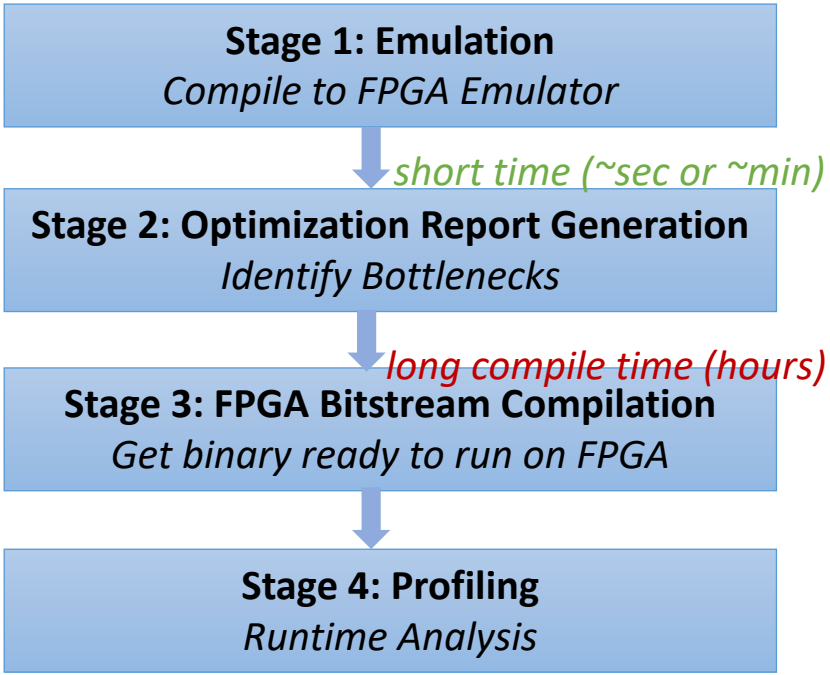
A part of the DPC++ Tutorial Series

Prof. Yan Luo

Acknowledgement
This work is supported by Intel Corporation 2020-2021

UMASS LOWELL
Learning with Purpose

Analyze your Design before Optimization



1. make sure the design is functionally correct
2. look for bottlenecks through compilation reports
3. revise the design to eliminate bottlenecks
4. repeat 1,2,3
5. profiling to analyze runtime performance

UMASS LOWELL
Learning with Purpose

Demonstration

Look through compilation report of Matrix Multiplication example

UMASS LOWELL
Learning with Purpose