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

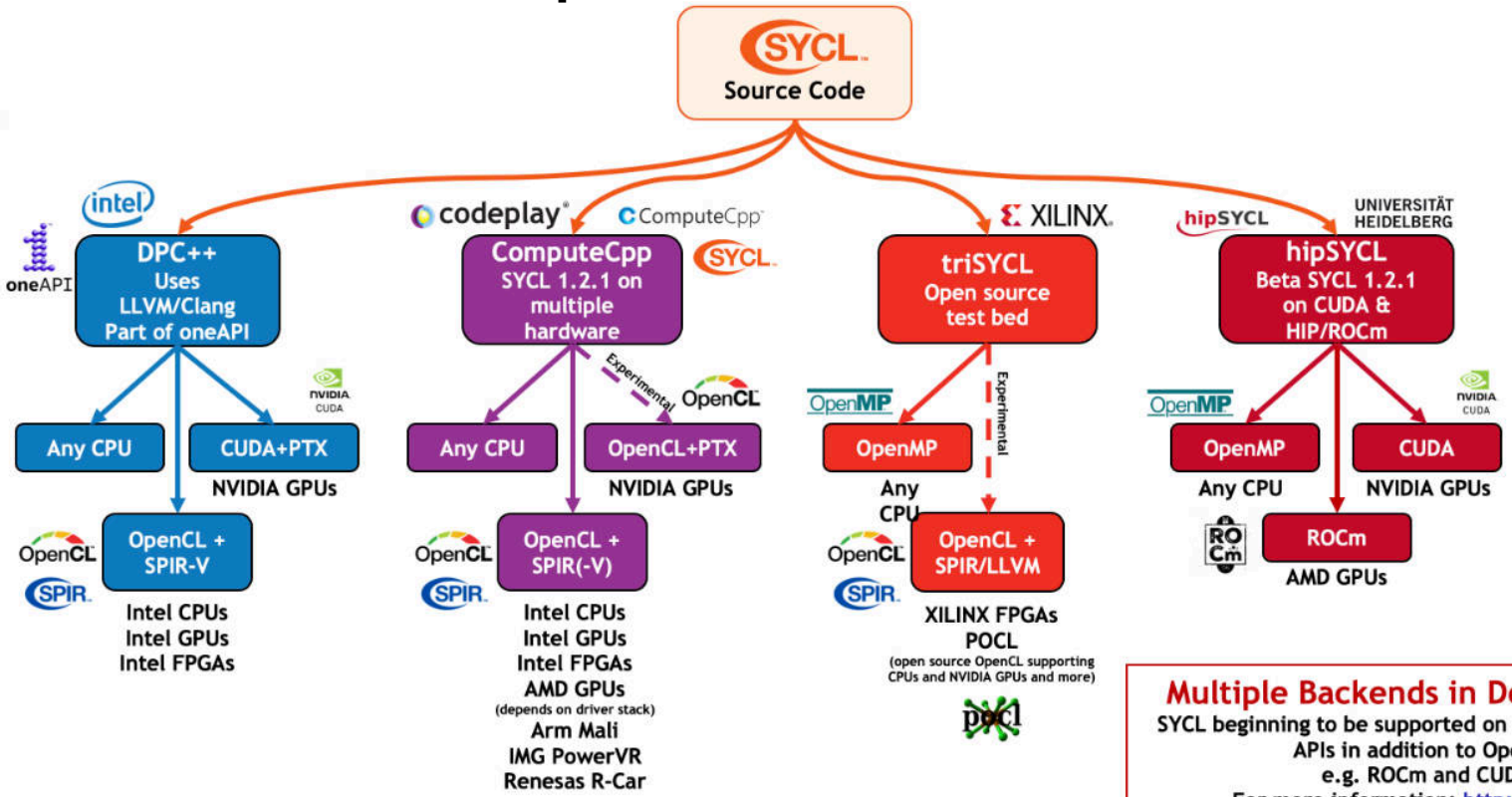
UMASS LOWELL
Learning with Purpose

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

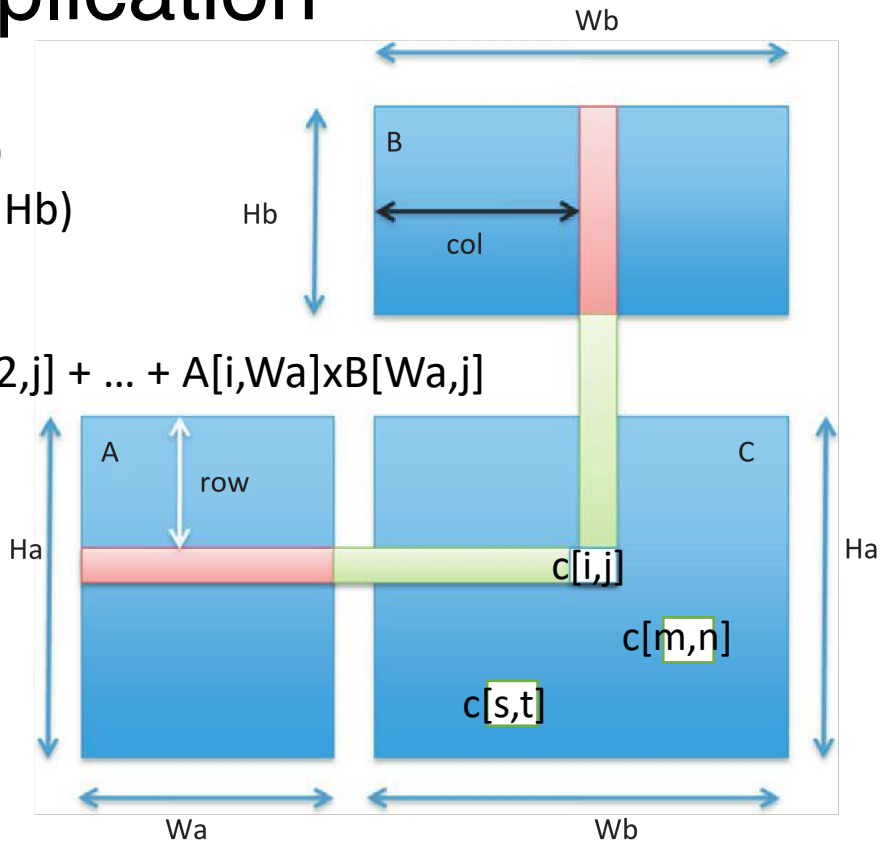
UMASS LOWELL
Learning with Purpose

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

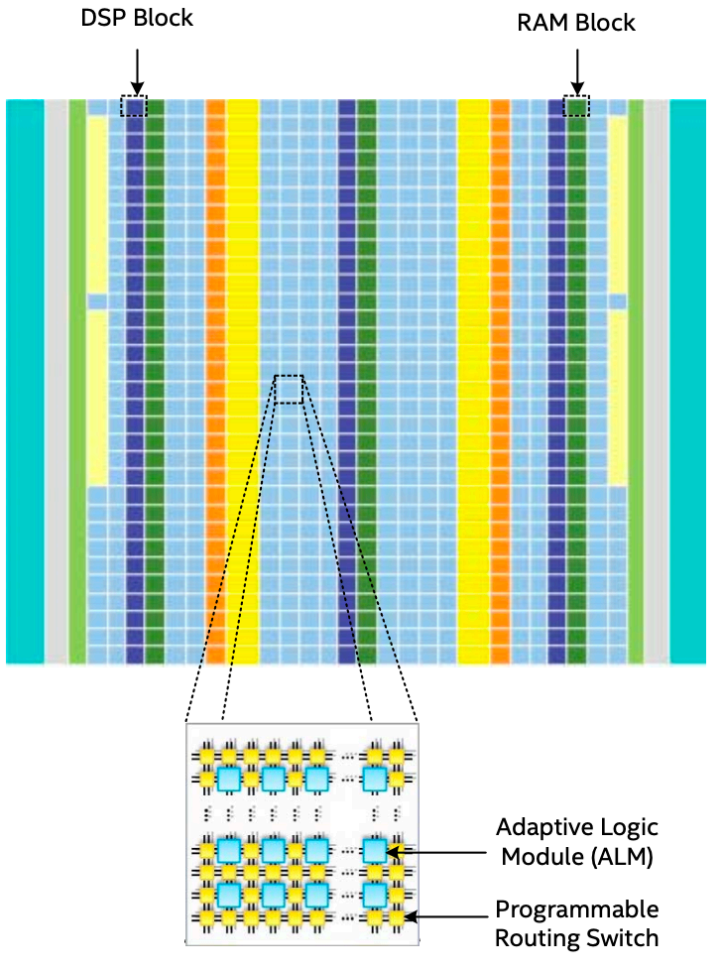
FPGA vs CPU

- 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



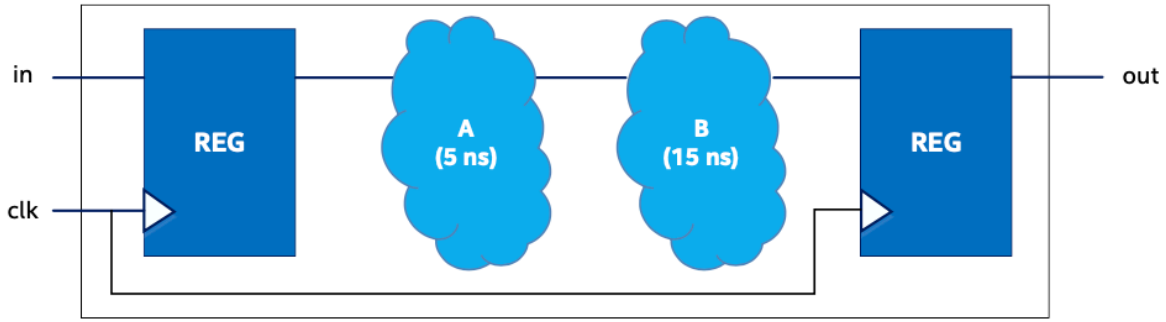
FPGA Hardware Design Concepts

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

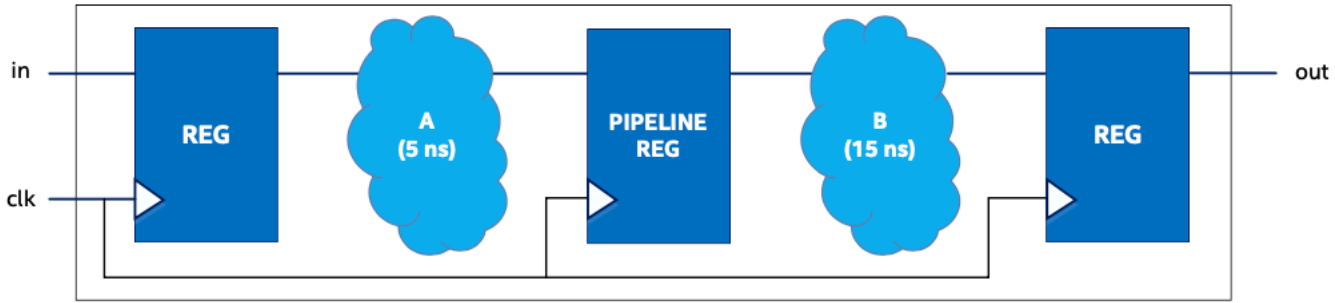
UMASS LOWELL
Learning with Purpose

Pipelining to improve f_{MAX}

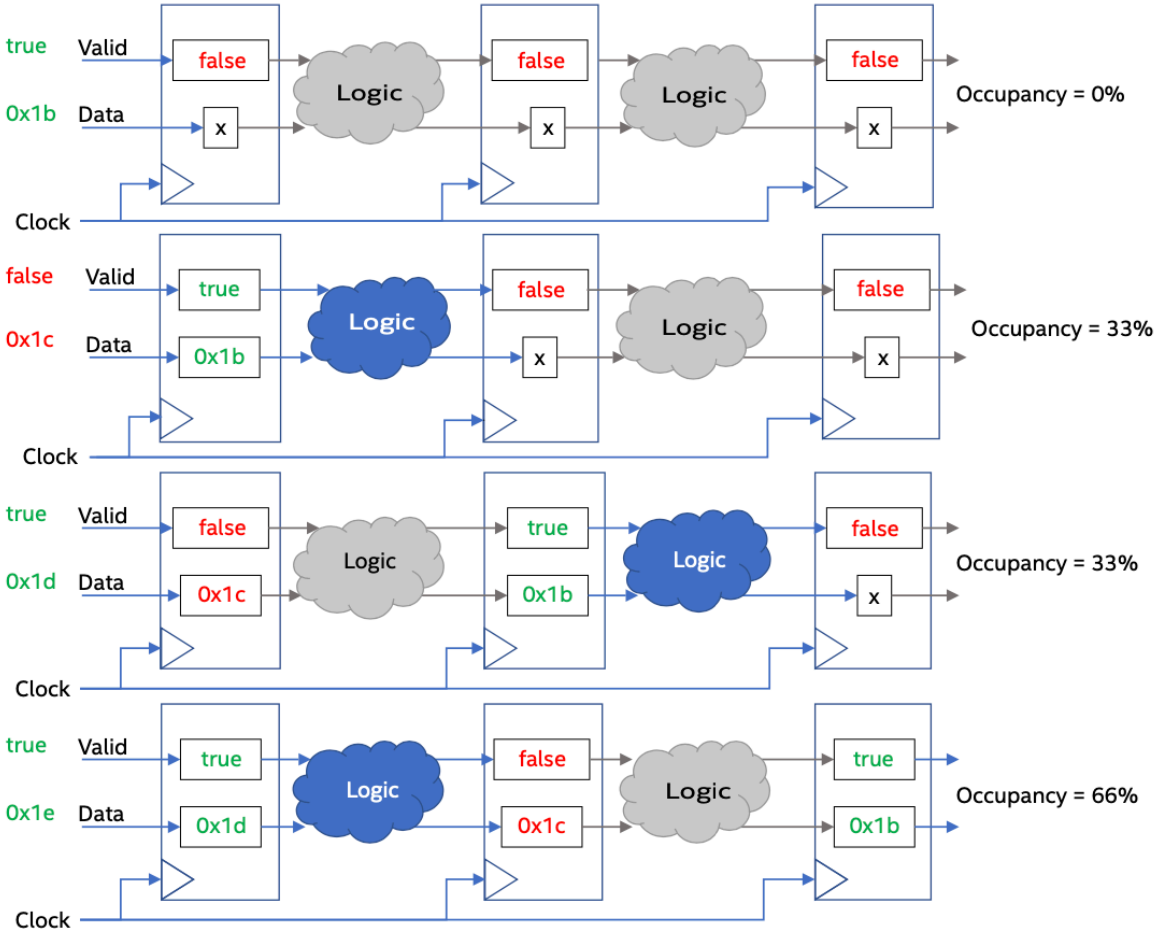
unpipelined
f_{MAX} = 50MHz



pipelined
f_{MAX} = 66.7MHz



Occupancy



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

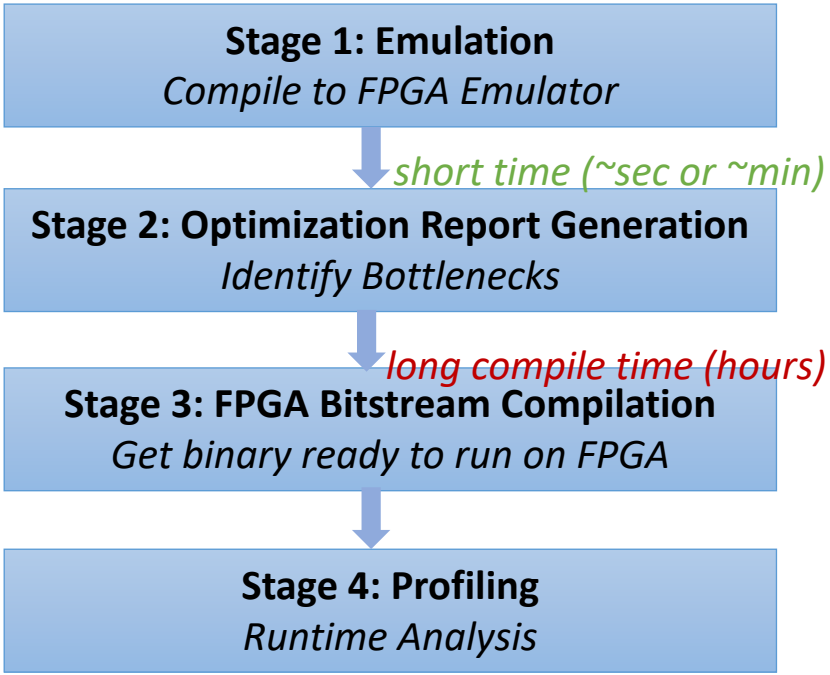
A part of the DPC++ Tutorial Series

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

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

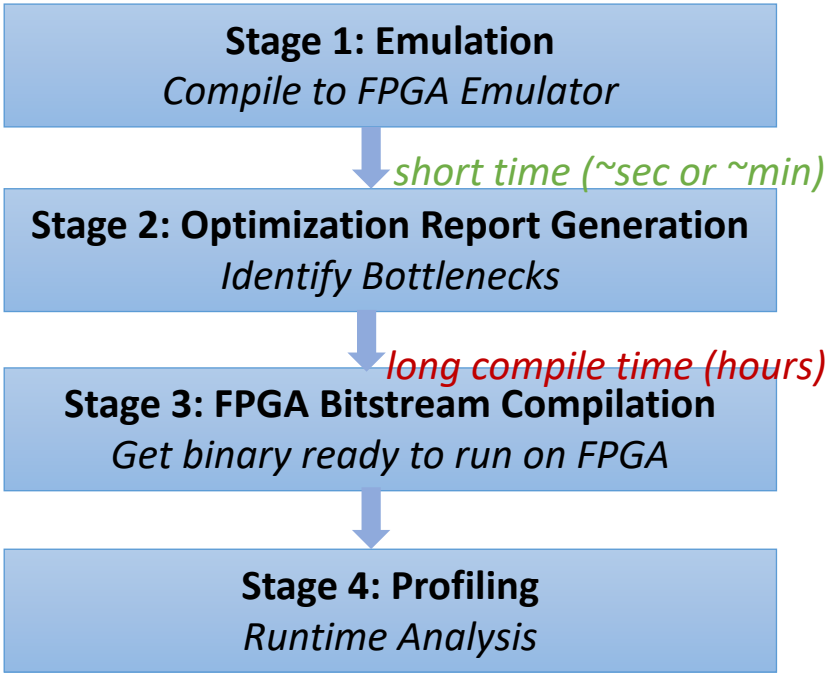
A part of the DPC++ Tutorial Series

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

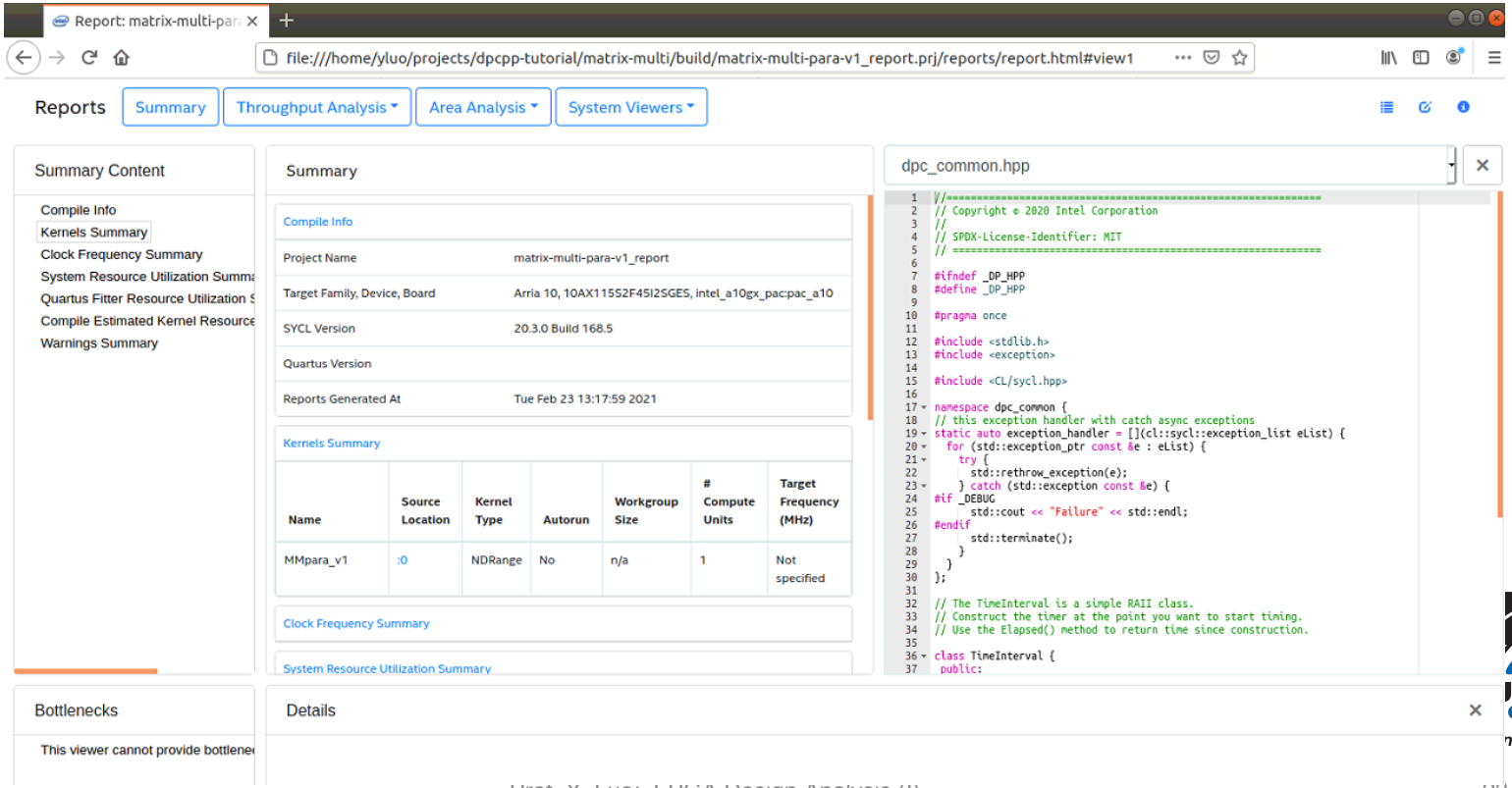
UMASS LOWELL
Learning with Purpose

Demo: Compilation Report Analysis

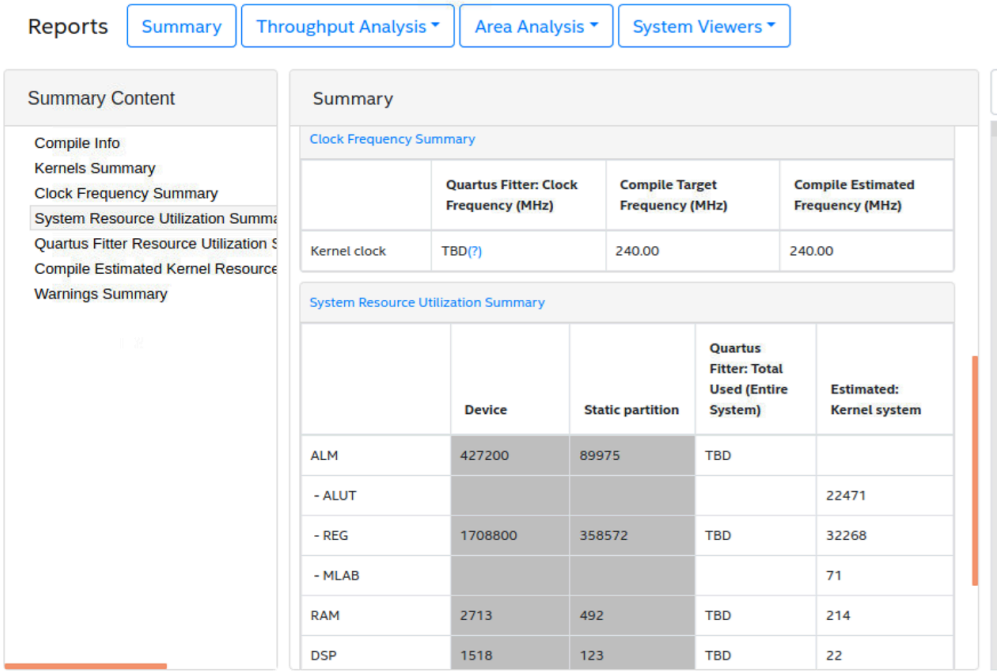
Look through compilation report of Matrix Multiplication example

UMASS LOWELL
Learning with Purpose

Static Analysis Report

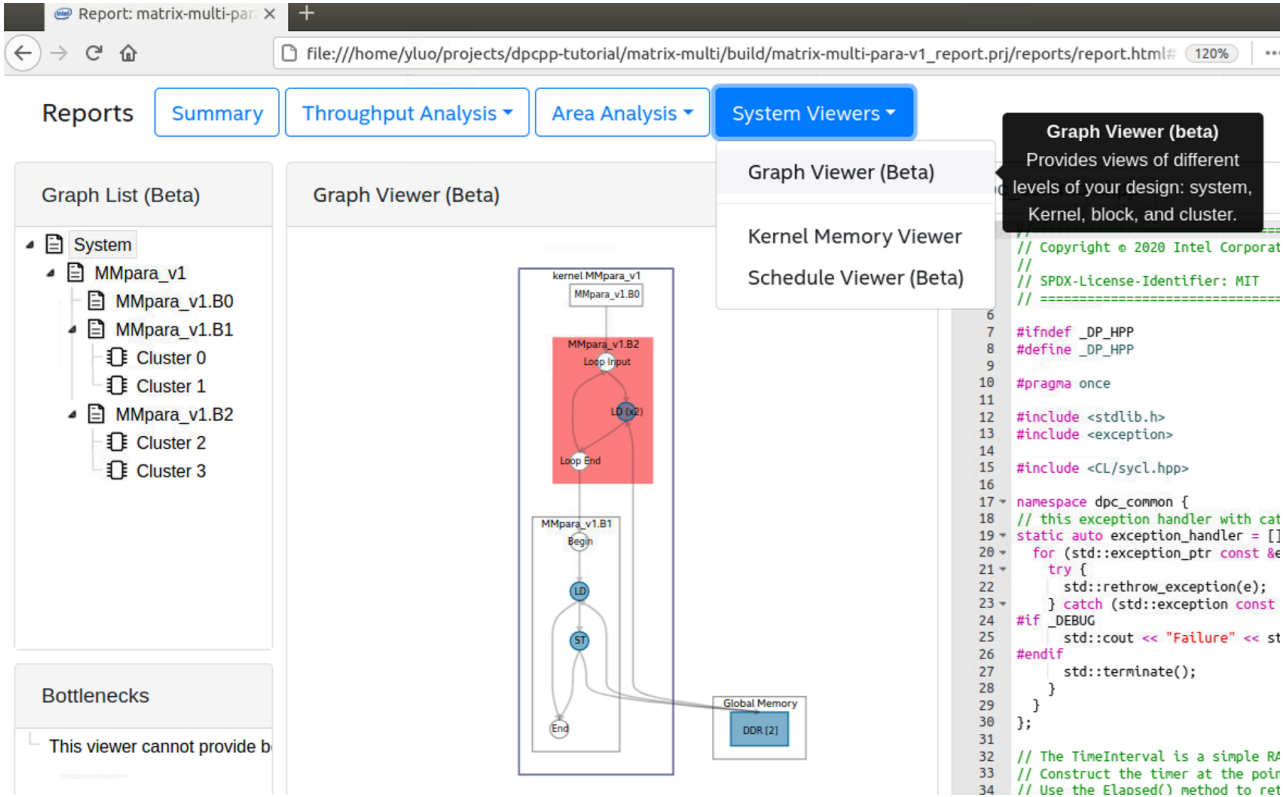


Clock Frequency and System Resource Utilization Summary

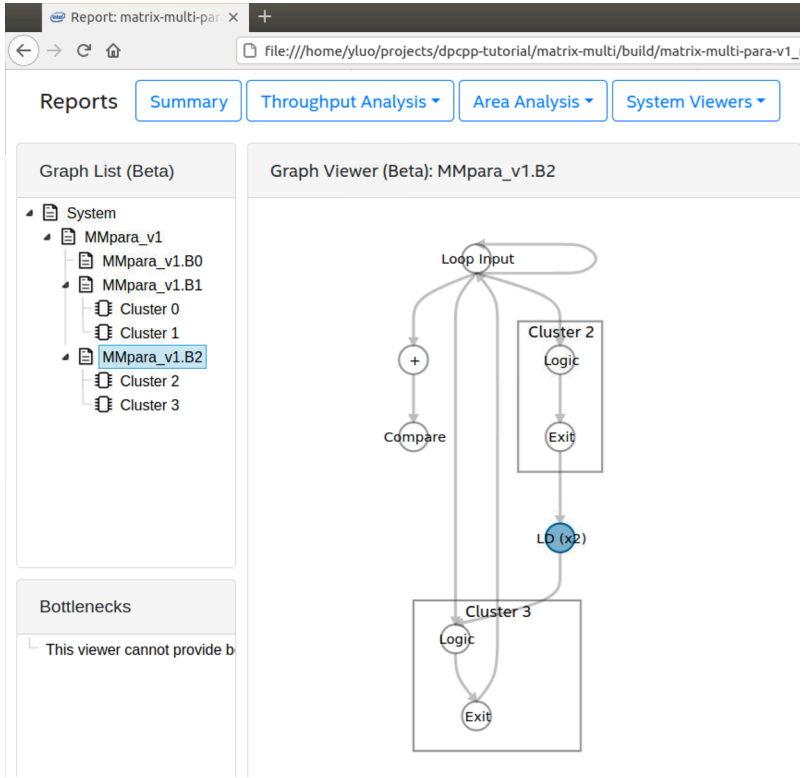


UMASS LOWELL
Learning with Purpose

System Graph Viewer



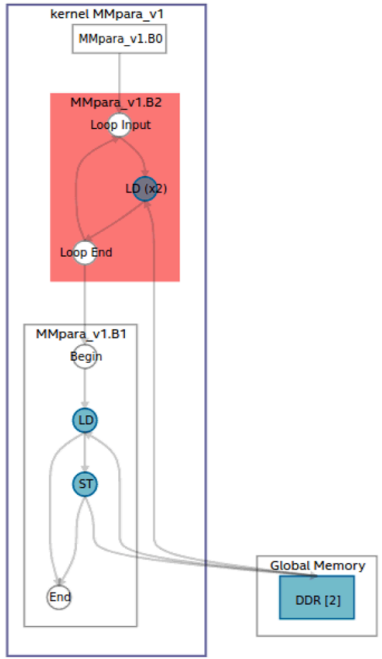
Zoom into a Module



UMASS LOWELL
Learning with Purpose

parallel for() v1

Graph Viewer (Beta)

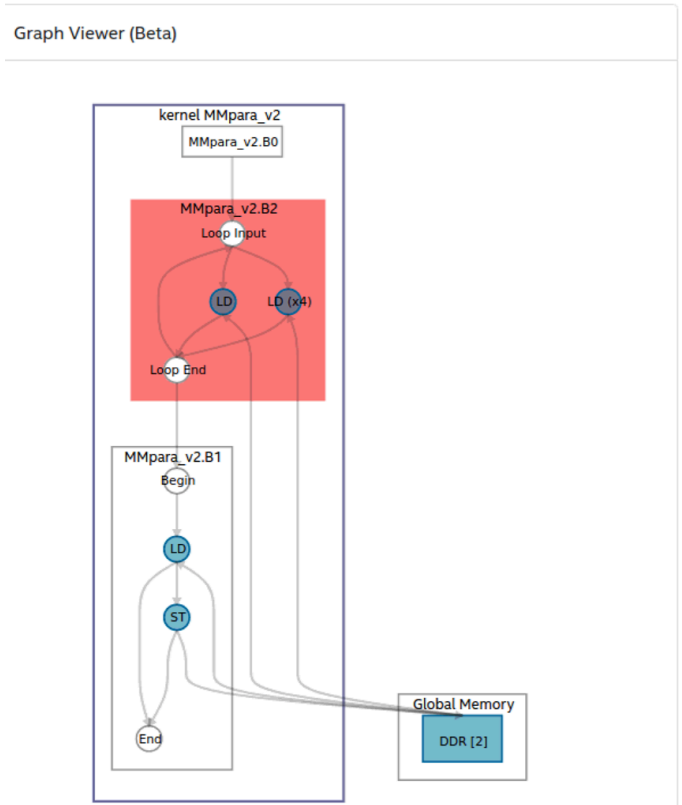


matrix-multi-para-v1.cpp

```
63      //    In our case, we have a two-dimensional nd_range:
64      //    num_items: the global size, or 'all' the work in 2D, i.e. the
          size
65      //                of the matrix Sum in 'row' and 'column'
          dimensions
66      //    range<2>(1,1) : the workgroup size. (1,1) means a workgroup has
          1 work item
67      //                  in each dimension
68      // 2nd parameter is the kernel, a lambda that specifies what to do
          per
69      //    work item. The parameter of the lambda is the work item id.
70      // DPC++ supports unnamed lambda kernel by default.
71      auto kernel_range = nd_range<2>(num_items, range<2>(1,1));
72      h.parallel_for<MMpara_v1>(num_items, [=](id<2> i)
73        { size_t row = i[0], col = i[1];
74
75          float s = 0;
76          //#pragma unroll 2
77          for (size_t k = 0; k < a_columns; k++)
78            s += a[row][k] * b[k][col];
79
80          sum[row][col]  = c[row][col] + s;
81        });
82    });
83
84  #if FPGA || FPGA_PROFILE
85    // Query event e for kernel profiling information
86    // (blocks until command groups associated with e complete)
87    double kernel_time_ns =
88      e.get_profiling_info<info::event_profiling::command_end>() -
89      e.get_profiling_info<info::event_profiling::command_start>();
90
91    // Report profiling info
92    std::cout << "Kernel compute time:  " << kernel_time_ns * 1e-6 << "
          ms\n";
93  #endif
94  }
95
96  //************************************
97  // Demonstrate matrix multiplication both in sequential on CPU and in
       parallel on device.
98  //************************************
99  int main() {
100   // Create device selector for the device of your interest
```

UMASS LOWELL
Learning with Purpose

parallel for() v2

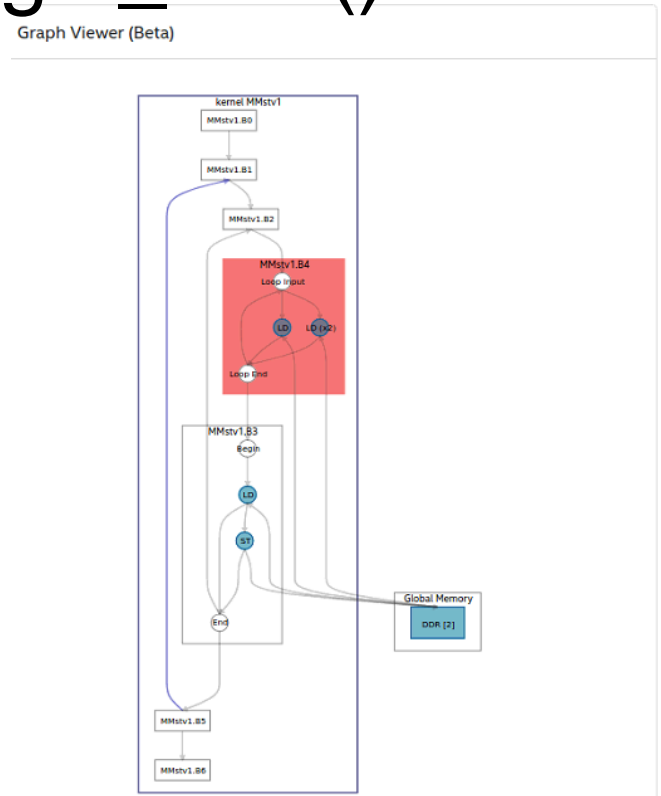


matrix-multi-para-v2.cpp

```
51      // Create an accessor for each buffer with access permission: read, write or
52      // read/write. The accessor is a mean to access the memory in the buffer.
53      auto a = a_buf.get_access<access::mode::read, access::target::global_buffer>(h);
54      auto b = b_buf.get_access<access::mode::read, access::target::global_buffer>(h);
55      auto c = c_buf.get_access<access::mode::read, access::target::global_buffer>(h);
56
57      // The sum_accessor is used to store (with write permission) the sum data.
58      auto sum = sum_buf.get_access<access::mode::write>(h);
59
60      // Use parallel_for to run vector addition in parallel on device. This
61      // executes the kernel.
62      // 1st parameter is the number of work items in total and in a workgroup
63      //    In our case, we have a two-dimensional nd_range:
64      //    num_items: the global size, or 'all' the work in 2D, i.e. the size
65      //                of the matrix Sum in 'row' and 'column' dimensions
66      //    range<2>(1,1) : the workgroup size. (1,1) means a workgroup has 1 work item
67      //                    in each dimension
68      // 2nd parameter is the kernel, a lambda that specifies what to do per
69      //    work item. The parameter of the lambda is the work item id.
70      // DPC++ supports unnamed lambda kernel by default.
71      auto kernel_range = nd_range<2>(num_items, range<2>(1,1));
72      h.parallel_for<MMpara_v2>(num_items, [=](id<2> i)
73        { size_t row = i[0], col = i[1];
74
75          float s = 0;
76          #pragma unroll 4
77          for (size_t k = 0; k < a_columns; k++)
78            s += a[row][k] * b[k][col];
79
80          sum[row][col]  = c[row][col] + s;
81        });
82    });
83
84  #if FPGA || FPGA_PROFILE
85    // Query event e for kernel profiling information
86    // (blocks until command groups associated with e complete)
87    double kernel_time_ns =
88      e.get_profiling_info<info::event_profiling::command_end>() -
89      e.get_profiling_info<info::event_profiling::command_start>();
90
91    // Report profiling info
92    std::cout << "Kernel compute time:  " << kernel_time_ns * 1e-6 << " ms\n";
93  #endif
94  }
95
96  //************************************
97  // Demonstrate matrix multiplication both in sequential on CPU and in parallel on device.
98  //************************************
99  int main() {
100   // Create device selector for the device of your interest.
101 #if FPGA_EMULATOR
102   // DPC++ extension: FPGA emulator selector on systems without FPGA card.
```


UMASS LOWELL
Learning with Purpose

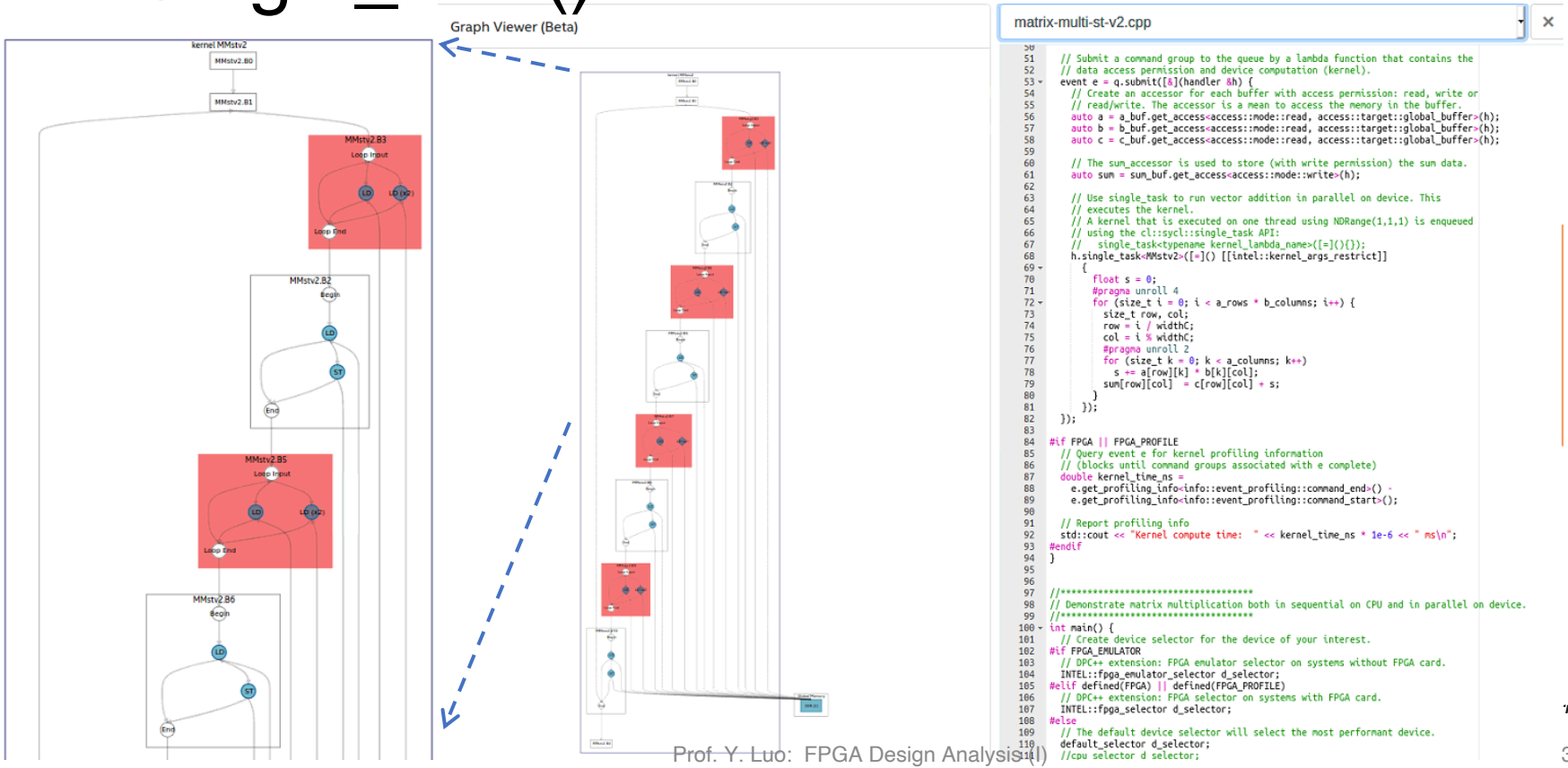
single_task() v1



matrix-multi-st-v1.cpp

```
  event e = q.submit([&](handler &h) {
    // Create an accessor for each buffer with access permission: read, write or
    // read/write. The accessor is a mean to access the memory in the buffer.
    auto a = a_buf.get_access<access::mode::read, access::target::global_buffer>(h);
    auto b = b_buf.get_access<access::mode::read, access::target::global_buffer>(h);
    auto c = c_buf.get_access<access::mode::read, access::target::global_buffer>(h);

    // The sum_accessor is used to store (with write permission) the sum data.
    auto sum = sum_buf.get_access<access::mode::write>(h);

    // Use single_task to run vector addition in parallel on device. This
    // executes the kernel.
    // A kernel that is executed on one thread using NDRange(1,1,1) is enqueued
    // using the cl::sycl::single_task API:
    //   single_task<typename kernel_lambda_name>([=](){});
    h.single_task<MMstv1>([=]() [[intel::kernel_args_restrict]]
      {
        size_t row, col;
        float s = 0;
        for (row = 0; row < a_rows; row++)
          for (col = 0; col < b_columns; col++) {
            #pragma unroll 2
            for (size_t k = 0; k < a_columns; k++)
              s += a[row][k] * b[k][col];
            sum[row][col]  = c[row][col] + s;
          }
      });
  });

#if FPGA || FPGA_PROFILE
  // Query event e for kernel profiling information
  // (blocks until command groups associated with e complete)
  double kernel_time_ns =
    e.get_profiling_info<info::event_profiling::command_end>() -
    e.get_profiling_info<info::event_profiling::command_start>();

  // Report profiling info
  std::cout << "Kernel compute time:  " << kernel_time_ns * 1e-6 << " ms\n";
#endif

}


//************************************
// Demonstrate matrix multiplication both in sequential on CPU and in parallel on device.
//************************************
int main() {
  // Create device selector for the device of your interest.
#if FPGA_EMULATOR
  // DPC++ extension: FPGA emulator selector on systems without FPGA card.
  INTEL::fpga_emulator_selector d_selector;
#elif defined(FPGA) || defined(FPGA_PROFILE)
  // DPC++ extension: FPGA selector on systems with FPGA card.
```

Single_task() v2



matrix-multi-st-v2.cpp

```
  // Submit a command group to the queue by a lambda function that contains the
  // data access permission and device computation (kernel).
  event e = q.submit([&](handler &h) {
    // Create an accessor for each buffer with access permission: read, write or
    // read/write. The accessor is a mean to access the memory in the buffer.
    auto a = a_buf.get_access<access::mode::read, access::target::global_buffer>(h);
    auto b = b_buf.get_access<access::mode::read, access::target::global_buffer>(h);
    auto c = c_buf.get_access<access::mode::read, access::target::global_buffer>(h);

    // The sum_accessor is used to store (with write permission) the sum data.
    auto sum = sum_buf.get_access<access::mode::write>(h);

    // Use single_task to run vector addition in parallel on device. This
    // executes the kernel.
    // A kernel that is executed on one thread using NDRange(1,1,1) is enqueued
    // using the cl::sycl::single_task API:
    //   single_task<typename kernel_lambda_name>([=](){});
    h.single_task<MMstv2>([=]() [[intel::kernel_args_restrict]]
      {
        float s = 0;
        #pragma unroll 4
        for (size_t i = 0; i < a_rows * b_columns; i++) {
          size_t row, col;
          row = i / widthC;
          col = i % widthC;
          #pragma unroll 2
          for (size_t k = 0; k < a_columns; k++)
            s += a[row][k] * b[k][col];
          sum[row][col]  = c[row][col] + s;
        }
      });
  });

#if FPGA || FPGA_PROFILE
  // Query event e for kernel profiling information
  // (blocks until command groups associated with e complete)
  double kernel_time_ns =
    e.get_profiling_info<info::event_profiling::command_end>() -
    e.get_profiling_info<info::event_profiling::command_start>();

  // Report profiling info
  std::cout << "Kernel compute time:  " << kernel_time_ns * 1e-6 << " ms\n";
#endif
}


//************************************
// Demonstrate matrix multiplication both in sequential on CPU and in parallel on device.
//************************************
int main() {
  // Create device selector for the device of your interest.
#if FPGA_EMULATOR
  // DPC++ extension: FPGA emulator selector on systems without FPGA card.
  INTEL::fpga_emulator_selector d_selector;
#elif defined(FPGA) || defined(FPGA_PROFILE)
  // DPC++ extension: FPGA selector on systems with FPGA card.
  INTEL::fpga_selector d_selector;
#else
  // The default device selector will select the most performant device.
  default_selector d_selector;
  //cpu selector d selector;
```

DPC++ Design Analysis (II): Analyze Runtime Profiling Data

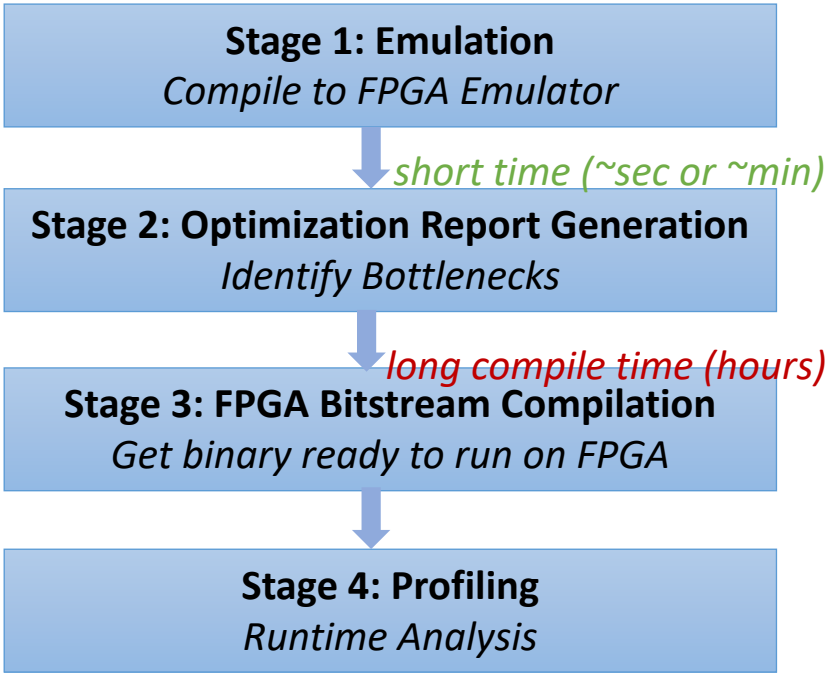
A part of the DPC++ Tutorial Series

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

Demo: Profiling Analysis

Run-time profiling analysis on Matrix Multiplication example

UMASS LOWELL
Learning with Purpose

Compile for profiling

Profiling data collection

- Execute FPGA binary with profiling tool. Do not generate json file

```
aocl profile -no-json ./matrix-multi-para-v1.fpga_profile
```

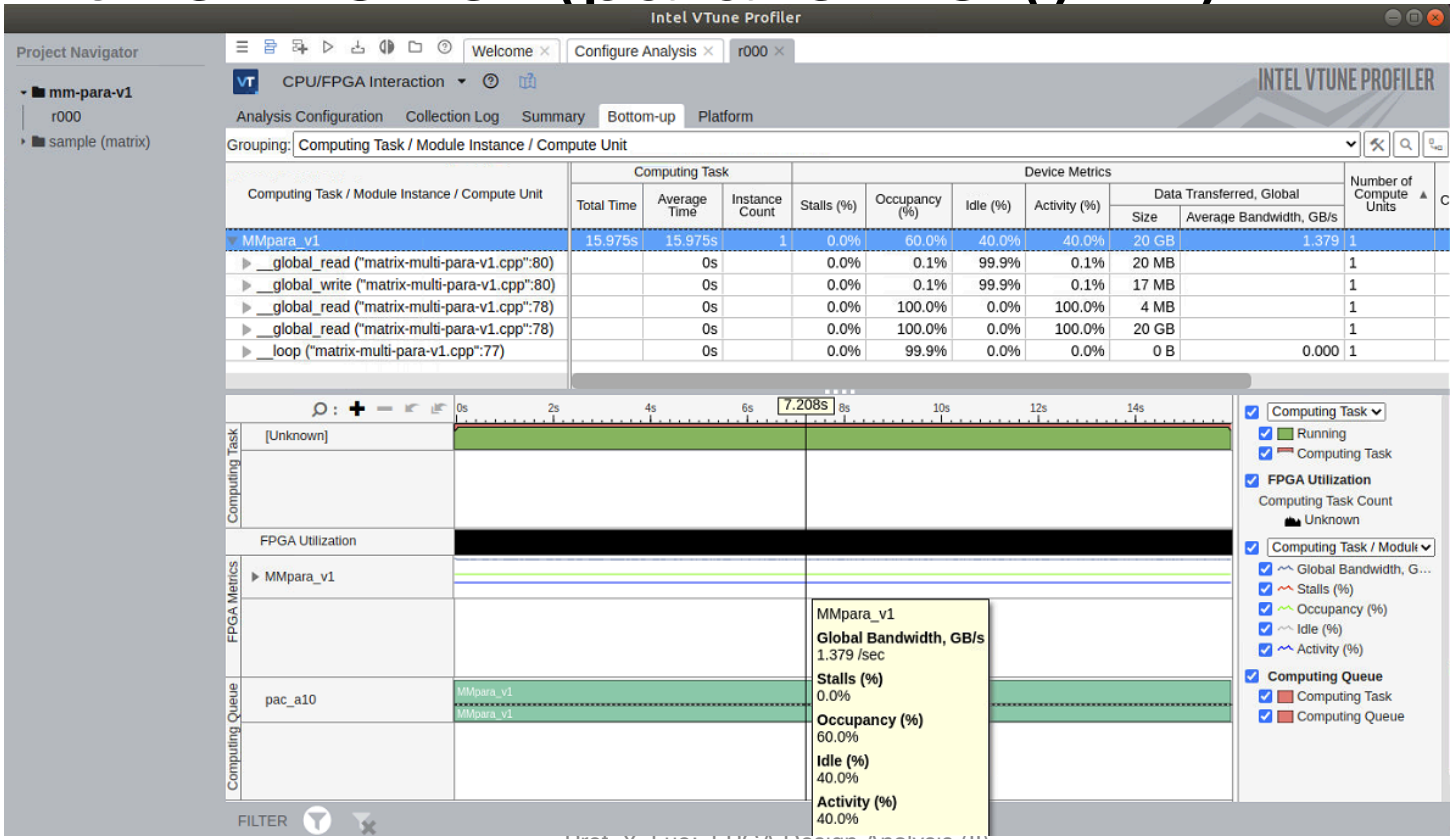
- Generate json results file using profile.mon

```
aocl profile -no-run profile.mon ./matrix-multi-para-v1.fpga_profile
```

- Load profile results folder to Intel VTune toolkit
  - Need to put the files (profile.mon and profile.json) in a folder
  - Check “import multiple trace files from a directory” option when importing profiling data in VTune.

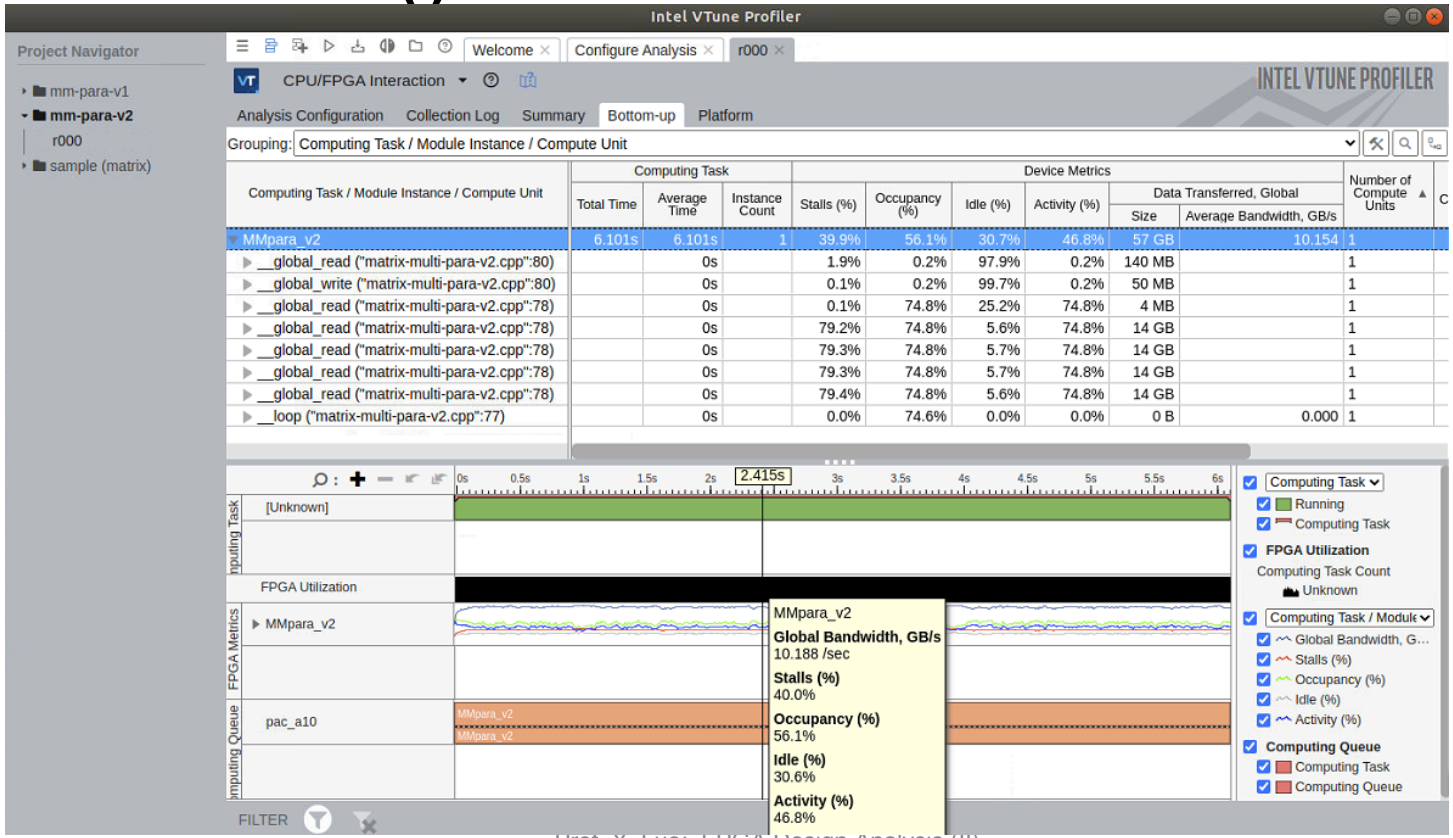
UMASS LOWELL
Learning with Purpose

VTune Profiler (parallel_for() v1)



UMASS LOWELL
Learning with Purpose

parallel for() v2



UMASS LOWELL
Learning with Purpose