

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



Learning with Purpose

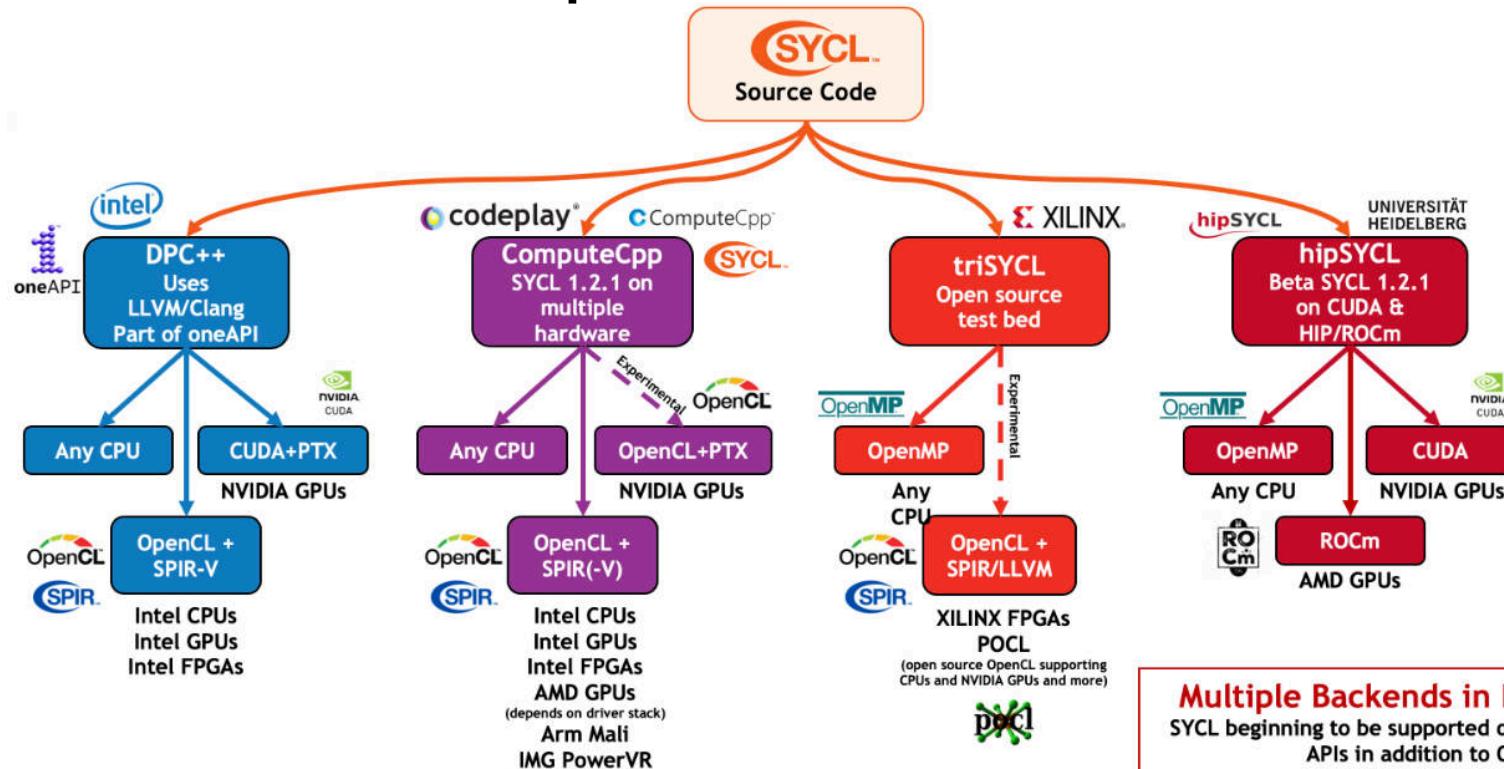
Data Parallel C++

- 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)



Learning with Purpose

DPC++: an Implementation of SYCL



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>

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 }
```

device queue
& command

kernel function
on device

Header Files

namespace
scope

SYCL buffer
declaration

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



Learning with Purpose

Matrix Multiplication

$$A \times B = C$$

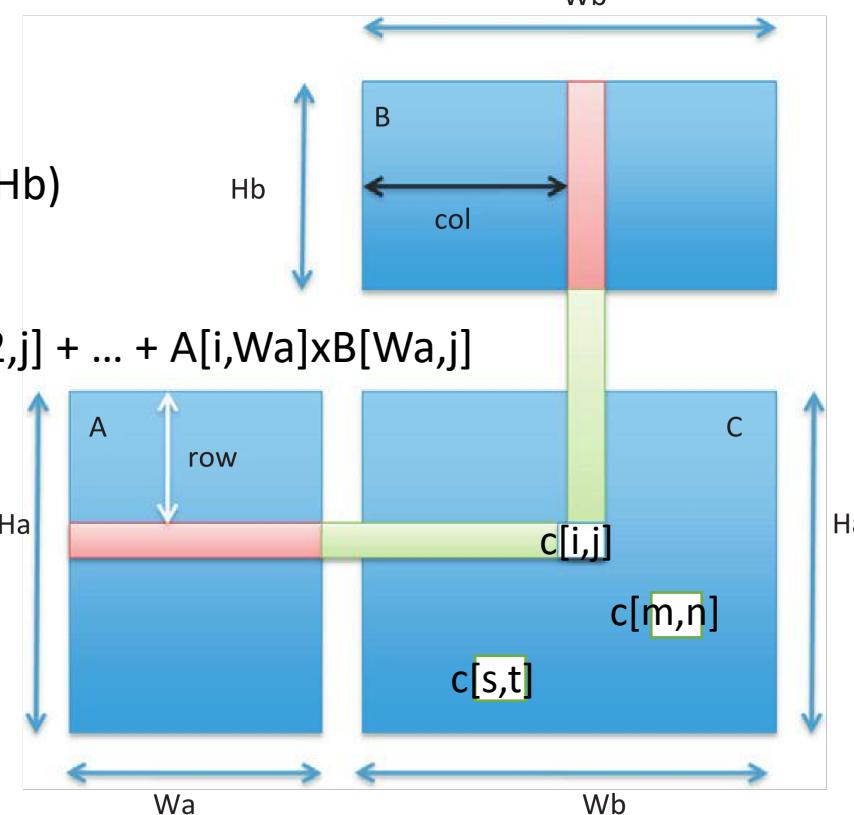
A is $W_a \times H_a$

B is $W_b \times H_b$

(Note: $W_a = H_b$)

C is $W_b \times H_a$

$$C[i,j] = A[i,1] \times B[1,j] + A[i,2] \times B[2,j] + \dots + A[i,W_a] \times B[W_a,j]$$



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];
    }
```



Learning with Purpose

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};
```

create
accessors

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

set up
problem size

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

lambda function
for device



Learning with Purpose

Demonstration

Compilation and execution of Matrix Multiplication example on Intel FPGA DevCloud



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



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



Learning with Purpose

FPGA vs CPU

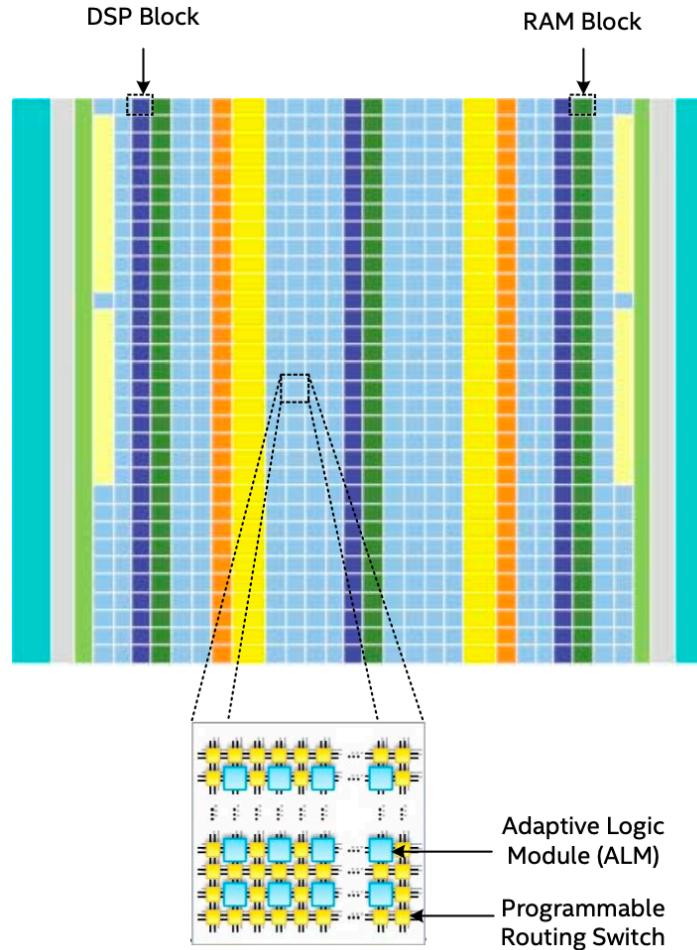
- FPGA does not have a fixed datapath
 - that is why it is “field programmable” !
- “Program” the hardware resources
 - You have a lot more control on how your design is mapped
- Function as accelerators to offload intensive computing
- Design methodologies
 - Hardware Description Language
 - High level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



Learning with Purpose

FPGA Architecture

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



FPGA Hardware Design Concepts

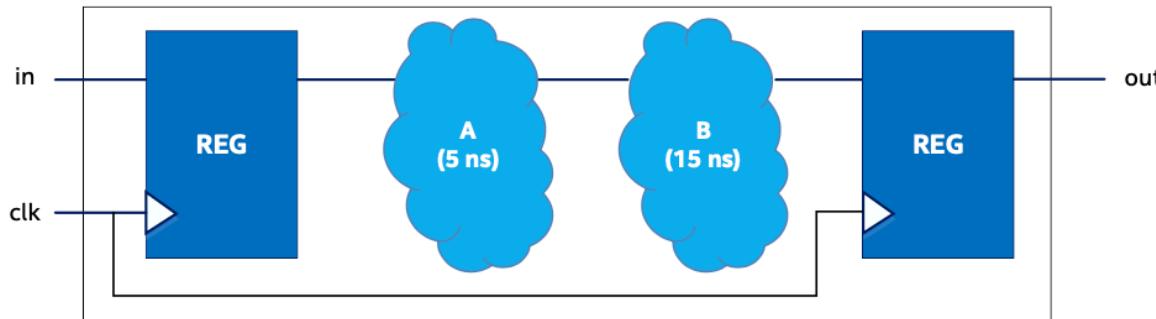
- Maximum frequency f_{MAX}
- Latency
- Pipelining
- Throughput
- Datapath
- Control path
- Occupancy



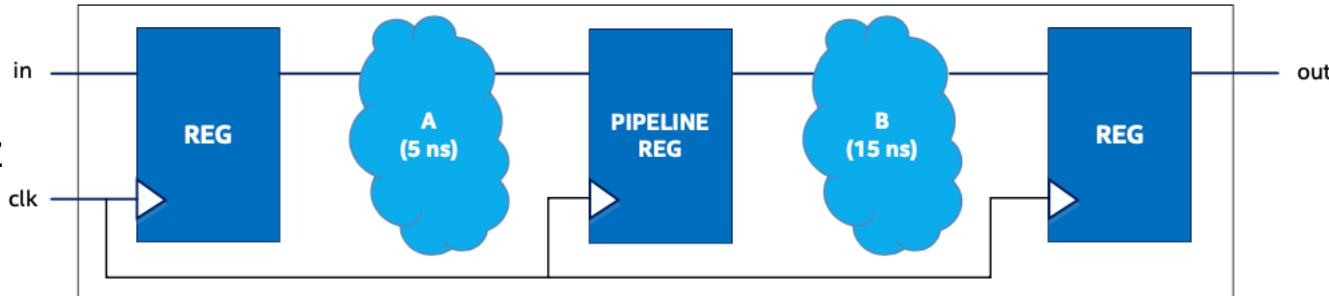
Learning with Purpose

Pipelining to improve f_{MAX}

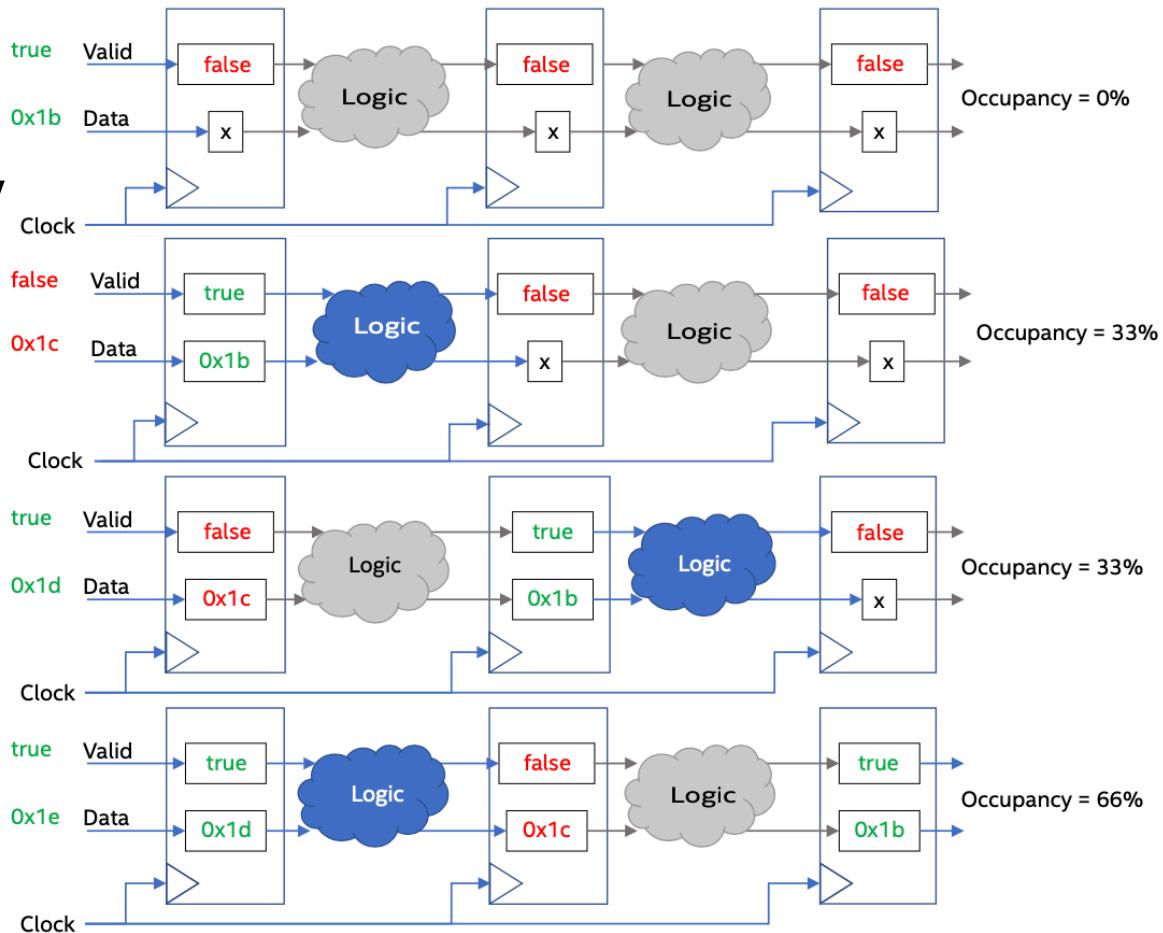
unpipelined
 $f_{MAX} = 50\text{MHz}$



pipelined
 $f_{MAX} = 66.7\text{MHz}$



Occupancy



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

A part of the DPC++ Tutorial Series

Prof. Yan Luo

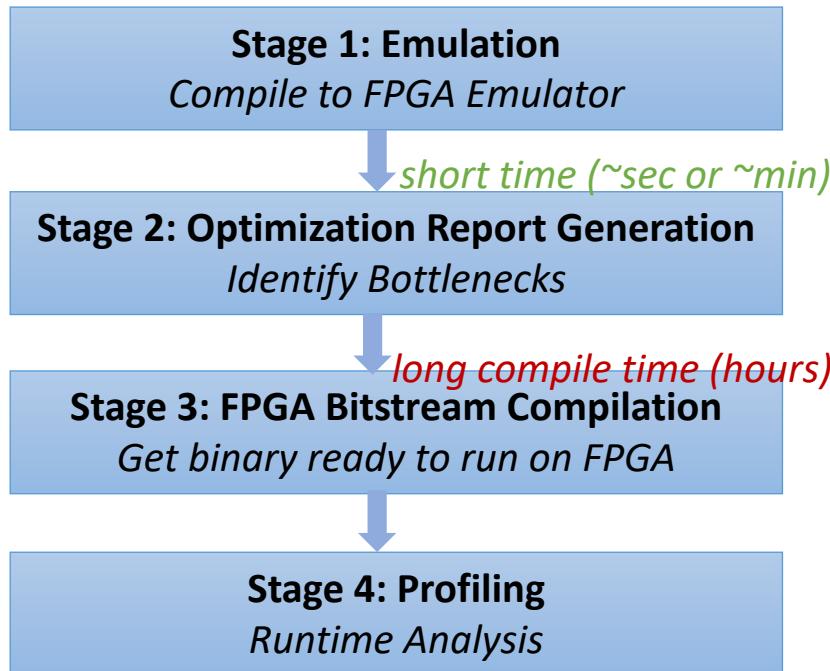
Acknowledgement

This work is supported by Intel Corporation 2020-2021



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



Learning with Purpose

Demonstration

Look through compilation report of Matrix Multiplication example



Learning with Purpose

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

A part of the DPC++ Tutorial Series

Prof. Yan Luo

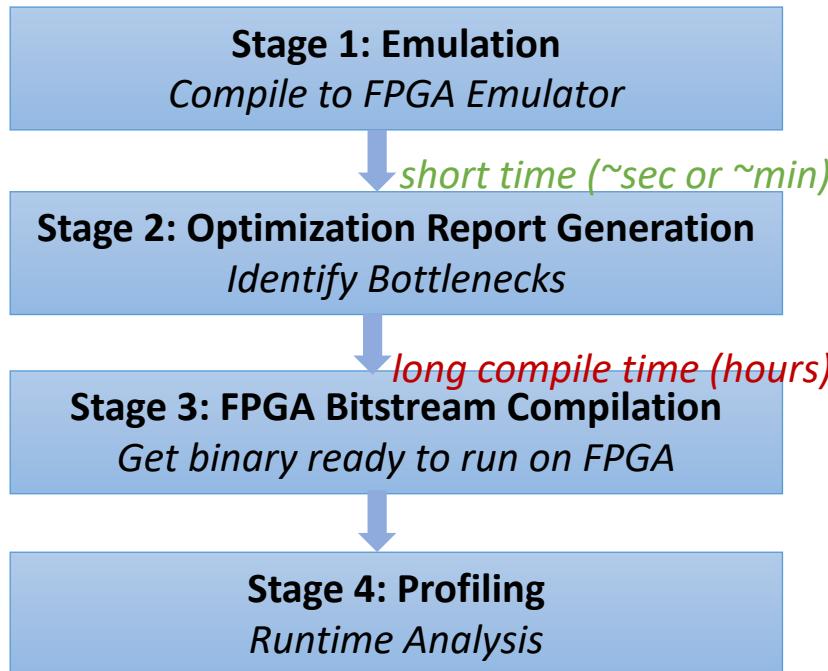
Acknowledgement

This work is supported by Intel Corporation 2020-2021



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



Demo: Compilation Report Analysis

Look through compilation report of Matrix Multiplication example



Learning with Purpose

Static Analysis Report

The screenshot shows a static analysis report for a matrix multiplication kernel. The report is titled "matrix-multi-para-v1_report". The main interface includes a sidebar with navigation links and tabs for different analysis types. The "Reports" tab is currently selected. The "Summary" tab is active, displaying sections for Compile Info, Kernels Summary, Clock Frequency Summary, System Resource Utilization Summary, Quartus Fitter Resource Utilization Summary, Compile Estimated Kernel Resources, and Warnings Summary. The "Kernels Summary" section lists a single kernel named "MMpara_v1" with details such as Source Location (:0), Kernel Type (NDRange), Autorun (No), Workgroup Size (n/a), # Compute Units (1), and Target Frequency (MHz) (Not specified). The "Code View" tab is open, showing the source code for "dpc_common.hpp". The code defines a namespace "dpc_common" containing exception handling logic. It includes #ifndef _DP_HPP and #define _DP_HPP guards, a pragma once directive, and includes for stdlib.h, exception, and CL/sycl.hpp. The namespace contains a static auto exception_handler variable and a try-catch block for std::exception. If DEBUG is defined, it outputs "Failure" to std::cout. The code ends with std::terminate(). The "Bottlenecks" and "Details" tabs are also visible at the bottom.

```
// Copyright © 2020 Intel Corporation
// SPDX-License-Identifier: MIT
// =====
#ifndef _DP_HPP
#define _DP_HPP
#pragma once
#include <stdlib.h>
#include <exception>
#include <CL/sycl.hpp>
namespace dpc_common {
// this exception handler with catch async exceptions
static auto exception_handler = []()>>cl::sycl::exception_list elist) {
    for (std::exception_ptr const &e : elist) {
        try {
            std::rethrow_exception(e);
        } catch (std::exception const &e) {
#ifdef _DEBUG
            std::cout << "Failure" << std::endl;
#endif
            std::terminate();
        }
    }
}
// The TimeInterval is a simple RAII class.
// Construct the timer at the point you want to start timing.
// Use the Elapsed() method to return time since construction.
class TimeInterval {
public:
```

Clock Frequency and System Resource Utilization Summary

Reports Summary Throughput Analysis ▾ Area Analysis ▾ System Viewers ▾

Summary				
Clock Frequency Summary				
	Quartus Fitter: Clock Frequency (MHz)	Compile Target Frequency (MHz)	Compile Estimated Frequency (MHz)	
Kernel clock	TBD(?)	240.00	240.00	

System Resource Utilization Summary				
	Device	Static partition	Quartus Fitter: Total Used (Entire System)	Estimated: Kernel system
ALM	427200	89975	TBD	
- ALUT				22471
- REG	1708800	358572	TBD	32268
- MLAB				71
RAM	2713	492	TBD	214
DSP	1518	123	TBD	22



Learning with Purpose

System Graph Viewer

Report: matrix-multi-par x +

file:///home/yluo/projects/dpcpp-tutorial/matrix-multi/build/matrix-multi-para-v1_report.prj/reports/report.html# 120% ...

Reports Summary Throughput Analysis Area Analysis System Viewers

Graph List (Beta)

- System
 - MMpara_v1
 - MMpara_v1.B0
 - MMpara_v1.B1
 - Cluster 0
 - Cluster 1
 - MMpara_v1.B2
 - Cluster 2
 - Cluster 3

Graph Viewer (Beta)

Graph Viewer (Beta)

Kernel Memory Viewer

Schedule Viewer (Beta)

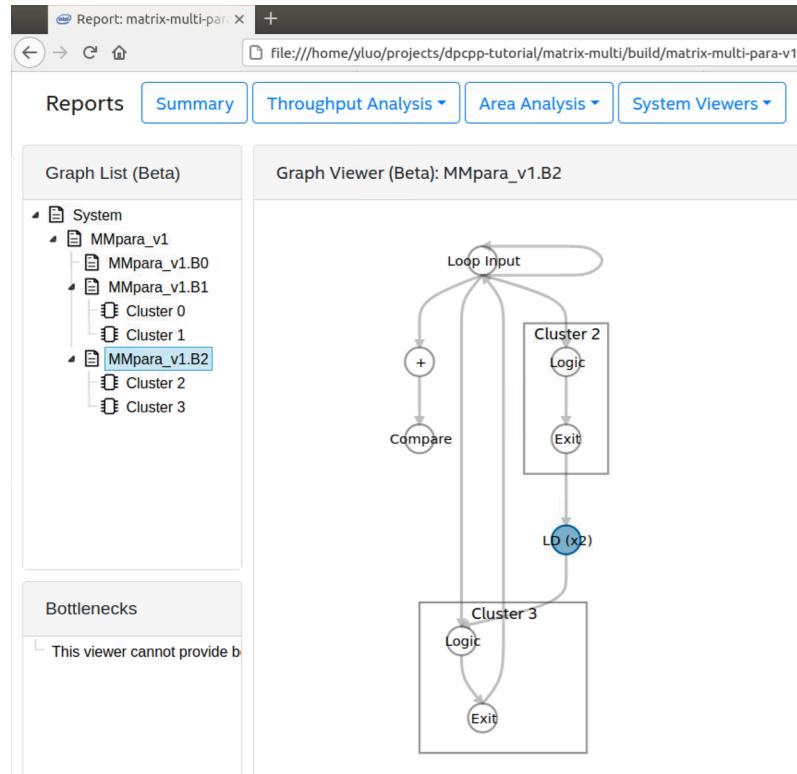
Graph Viewer (beta)
Provides views of different levels of your design: system, Kernel, block, and cluster.

```
// Copyright © 2020 Intel Corporation  
// SPDX-License-Identifier: MIT  
// ======  
6 ifndef _DP_HPP  
7 define _DP_HPP  
8  
9 pragma once  
10  
11 include <stdlib.h>  
12 #include <exception>  
13  
14 include <CL/sycl.hpp>  
15  
16 namespace dpc_common {  
17 // this exception handler with cat  
18 static auto exception_handler = []  
19 for (std::exception_ptr const &e:  
20 try {  
21 std::rethrow_exception(e);  
22 } catch (std::exception const &e:  
23 #ifdef _DEBUG  
24 std::cout << "Failure" << e.what();  
25 #endif  
26 std::terminate();  
27 }  
28 }  
29 };  
30  
31 // The TimeInterval is a simple RAII class.  
32 // Construct the timer at the point of creation.  
33 // Use the Elapsed() method to get the time.  
34
```



Learning with Purpose

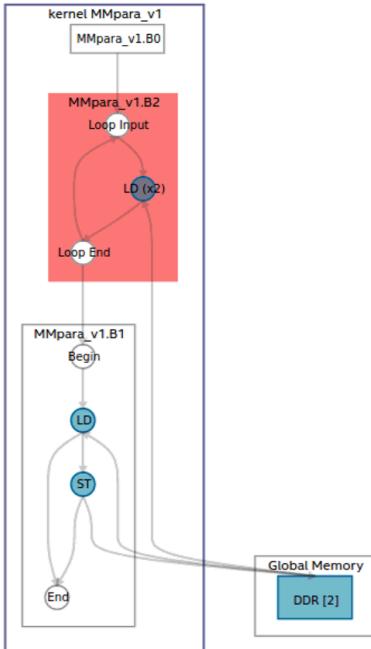
Zoom into a Module



Learning with Purpose

parallel for() v1

Graph Viewer (Beta)



matrix-multi-para-v1.cpp

```

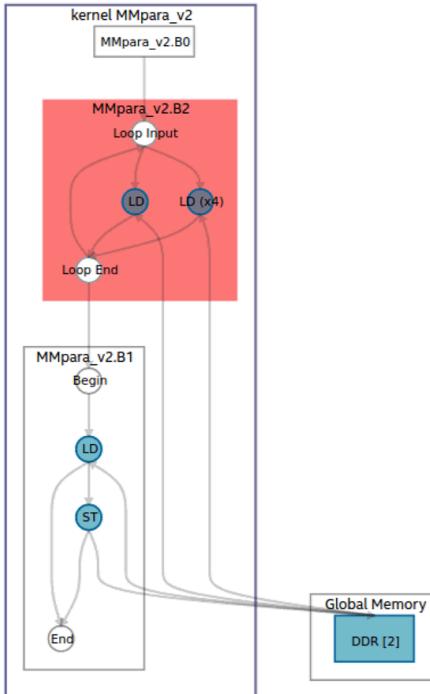
63 // In our case, we have a two-dimensional nd_range:
64 // num_items: the global size, or 'all' the work in 2D, i.e. the
65 // size
66 // dimensions of the matrix Sum in 'row' and 'column'
67 // range<2>(1,1) : the workgroup size. (1,1) means a workgroup has
68 // 1 work item
69 // in each dimension
70 // 2nd parameter is the kernel, a lambda that specifies what to do
71 // per
72 // work item. The parameter of the lambda is the work item id.
73 // DPC++ supports unnamed lambda kernel by default.
74 auto kernel_range = nd_range<2>(num_items, range<2>(1,1));
75 h.parallel_for<MMpara_v1>(num_items, [=](id<2> i)
76 {
77     size_t row = i[0], col = i[1];
78
79     float s = 0;
80     // #pragma unroll 2
81     for (size_t k = 0; k < a_columns; k++)
82         s += a[row][k] * b[k][col];
83
84     sum[row][col] = c[row][col] + s;
85 });
86 });
87
88 #if FPGA || FPGA_PROFILE
89 // Query event e for kernel profiling information
90 // (blocks until command groups associated with e complete)
91 double kernel_time_ns =
92     e.get_profiling_info<info::event_profiling::command_end>() -
93     e.get_profiling_info<info::event_profiling::command_start>();
94
95 // Report profiling info
96 std::cout << "Kernel compute time: " << kernel_time_ns * 1e-6 << "
97 ns\n";
98 #endif
99 }
100
101 //*****
102 // Demonstrate matrix multiplication both in sequential on CPU and in
103 // parallel on device.
104 //*****
105 int main()
106 {
107     // Create device selector for the device of your interest
  
```



Learning with Purpose

parallel for() v2

Graph Viewer (Beta)



matrix-multi-para-v2.cpp

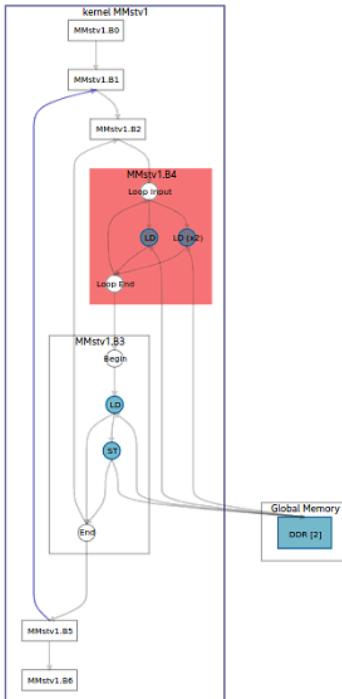
```

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 {
74     size_t row = i[0], col = i[1];
75
76     float s = 0;
77     #pragma unroll 4
78     for (size_t k = 0; k < a_columns; k++)
79         s += a[row][k] * b[k][col];
80
81     sum[row][col] = c[row][col] + s;
82 });
83
84 #if FPGA || FPGA_PROFILE
85 // Query event for kernel profiling information
86 // (blocks until command groups associated with a 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 {
101     // Create device selector for the device of your interest.
102     #if FPGA_EMULATOR
103     // DPC++ extension: FPGA emulator selector on systems without FPGA card.

```

single_task() v1

Graph Viewer (Beta)



matrix-multi-st-v1.cpp

```

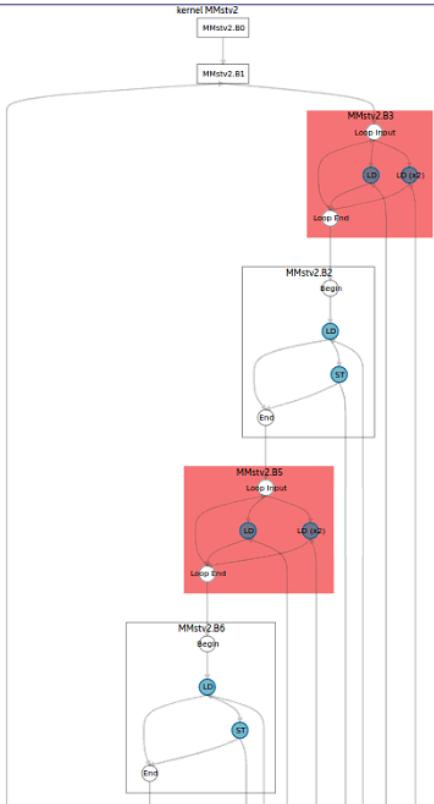
50 v event e = q.submit([&](handler &h) {
51   // Create an accessor for each buffer with access permission: read, write or
52   // read/write. The accessor is a mean to access the memory in the buffer.
53   auto a = a_buf.get_access<access::mode::read, access::target::global_buffer>(h);
54   auto b = 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 single_task to run vector addition in parallel on device. This
61   // executes the kernel.
62   // A kernel that is executed on one thread using NDRange(1,1,1) is enqueued
63   // using the cl::sycl:single_task API:
64   // _single_task<typename kernel_lambda_name>(<[]()>);
65   h.single_task<MMstv1>(<[]()> [[intel::kernel_args_restrict]])
66   {
67     size_t row, col;
68     float s = 0;
69     for (row = 0; row < a_rows; row++) {
70       for (col = 0; col < b_columns; col++) {
71         #pragma unroll 2
72         for (size_t k = 0; k < a_columns; k++) {
73           s += a[row][k] * b[k][col];
74           sum[row][col] = c[row][col] + s;
75         }
76       });
77     });
78
79 #if FPGA || FPGA_PROFILE
80   // Query event e for kernel profiling information
81   // (blocks until command groups associated with e complete)
82   double kernel_time_ns =
83     e.get_profiling_info<info::event_profiling::command_end>() -
84     e.get_profiling_info<info::event_profiling::command_start>();
85
86   // Report profiling info
87   std::cout << "Kernel compute time: " << kernel_time_ns * 1e-6 << " ms\n";
88 #endif
89
90 }
91
92
93 //*****
94 // Demonstrate matrix multiplication both in sequential on CPU and in parallel on device.
95 //*****
96 v int main() {
97   // Create device selector for the device of your interest.
98 #if FPGA_EMULATOR
99   // DPC++ extension: FPGA emulator selector on systems without FPGA card.
100   INTEL::fpga_emulator_selector d_selector;
101 #elif defined(FPGA) || defined(FPGA_PROFILE)
102   // DPC++ extension: FPGA selector on systems with FPGA card.

```

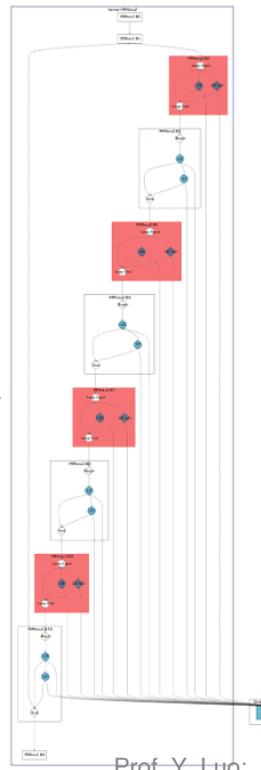


Learning with Purpose

Single_task() v2



Graph Viewer (Beta)



matrix-multi-st-v2.cpp

```

50 // Submit a command group to the queue by a lambda function that contains the
51 // data access permission and device computation (kernel).
52 event_t q_submitit[8]_lambda[8];
53
54 // Create an accessor for each buffer with access permission: read, write or
55 auto a = a_buf.get_access<access::mode::read, access::target::global_buffer>(h);
56 auto b = b_buf.get_access<access::mode::read, access::target::global_buffer>(h);
57 auto c = c_buf.get_access<access::mode::read, access::target::global_buffer>(h);
58
59 // The sum_accessor is used to store (with write permission) the sum data.
60 auto sum = sum_buf.get_access<access::mode::write>(h);
61
62 // Use single_task to run vector addition in parallel on device. This
63 // executes the kernel.
64 // A kernel that is executed on one thread using NRange(1,1,1) is enqueued
65 // using the cl::sycl::single_task API:
66 // single_task<typename kernel_lambda_name>(<=1)(());
67 h.single_task<MMstv2>(<=1)([[intel::kernel_args_restrict]]);
68
69
70 float s = 0;
71 #pragma unroll 4
72 for (size_t i = 0; i < a_rows * b_columns; i++) {
73     size_t row, col;
74     row = i / widthC;
75     col = i % widthC;
76     #pragma unroll 2
77     for (size_t k = 0; k < a_columns; k++)
78         s += a[row][k] * b[k][col];
79     sum[row][col] = c[row][col] + s;
80 }
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<event_profiling::command_end>() -
89     e.get_profiling_info<event_profiling::command_start>();
90
91 // Report profiling info
92 std::cout << "Kernel compute time: " << kernel_time_ns * 1e-6 << " ms\n";
93 #endiff
94 }
95
96
97 //*****
98 // Demonstrate matrix multiplication both in sequential on CPU and in parallel on device.
99 //*****
100
101 the main() {
102     // Create device selector for the device of your interest.
103     #if FPGA_EMULATOR
104         // DPC++ extension: FPGA emulator selector on systems without FPGA card.
105         INTEL::fpga_emulator_selector d_selector;
106     #elif defined(FPGA) || defined(FPGA_PROFILE)
107         // DPC++ extension: FPGA selector on systems with FPGA card.
108         INTEL::fpga_selector d_selector;
109     #else
110         // The default device selector will select the most performant device.
111         default_selector d_selector;
112         //cpu_selector d_selector;
113     #endif

```



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

A part of the DPC++ Tutorial Series

Prof. Yan Luo

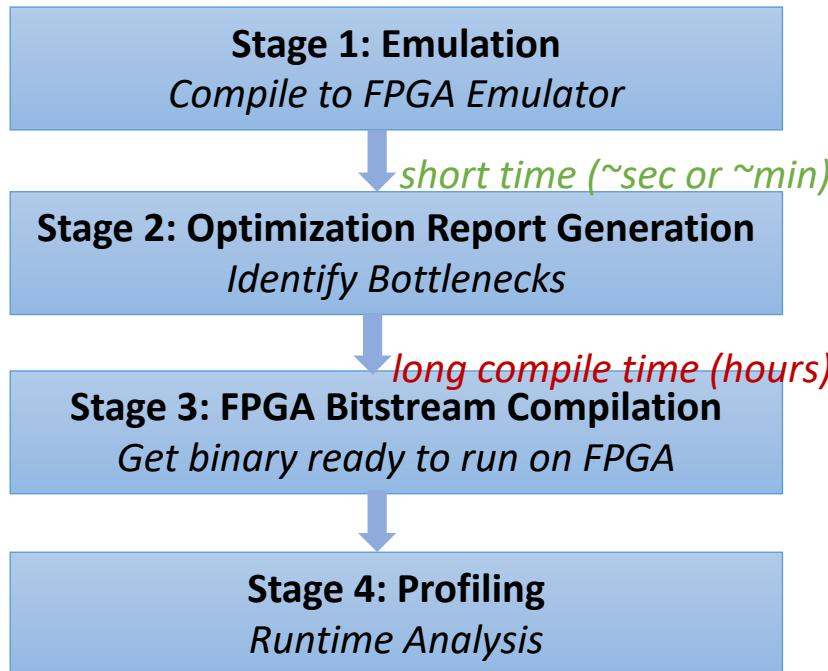
Acknowledgement

This work is supported by Intel Corporation 2020-2021



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



Demo: Profiling Analysis

Run-time profiling analysis on Matrix Multiplication example



Learning with Purpose

Compile for profiling



Learning with Purpose

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

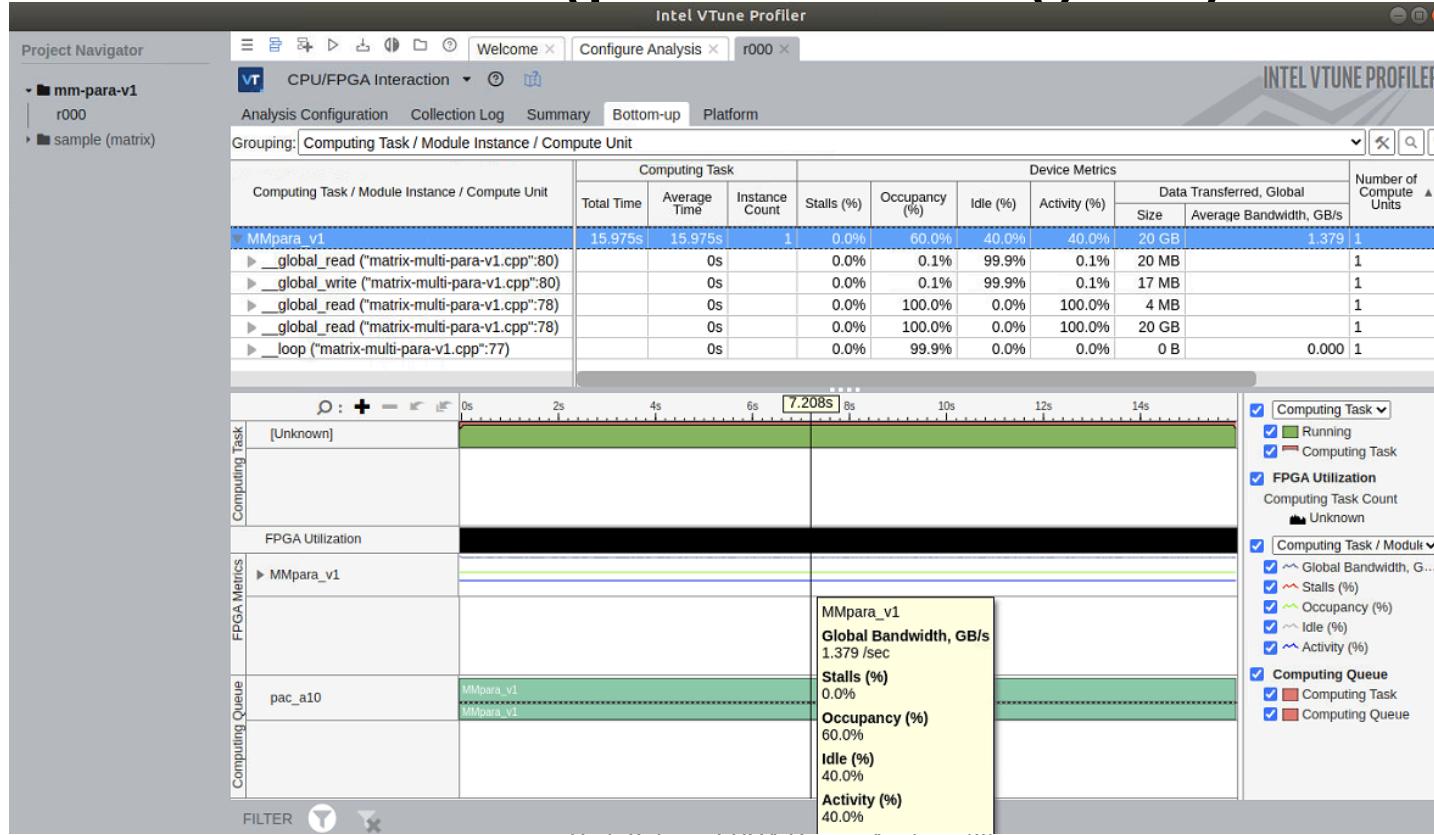
```
aocl profile -no-run profile.mon ./matrix-multi-para-v1.fpga_profile
```

- 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.



Learning with Purpose

VTune Profiler (parallel for() v1)



parallel for() v2

