High Performance
Computing &
Big Data Services

hpc.uni.lu

hpc@uni.lu

@ULHPC

LUXEMBOURG
LET'S MAKE IT HAPPEN

Reporting and profiling OpenCL kernels

Emmanuel Kieffer

UNIVERSITÉ DU
LUXEMBOURG

Define a kernel application

UNIVERSITÉ DU LUXEMBOURG

SDK example

- Intel FPGA SDK provides examples
- Let's take a simple example to see how it works
  - Examples are located at `$INTELFPGAOCLSDKROOT/examples_aoc`

```
[u100057@mel3001 hld]$ ls $INTELFPGAOCLSDKROOT/examples_aoc/
asian_option       library_hls_sot
channelizer        library_matrix_mult
common             local_memory_cache
compression        loopback_hostpipe
compute_score      Makefile
double_buffering   mandelbrot
extlibs            matrix_mult
fd3d               multithread_vector_operation
fft1d              n_way_buffering
fft1d_offchip      optical_flow
fft2d              sobel_filter
hello_world        tdfir
jpeg_decoder       vector_add
library_example1   video_downscaling
library_example2   web
```

Vector_add

- vector_add example
  - `mkdir $HOME/FPGA-OpenCL`
  - `cp -R $INTELFPGAOCLSDKROOT/examples_aoc/vector_add $HOME/FPGA-OpenCL/.`
  - `cp -R $INTELFPGAOCLSDKROOT/examples_aoc/common $HOME/FPGA-OpenCL/.`

- Just add to vector and retrieve the results
  - NDRange on a single dimension

```
// ACL kernel for adding two input vectors
__kernel void vector_add(__global const float *x, __global const float *y, __global float *restrict z){
   // get index of the work item
   int index = get_global_id(0);
   // add the vector elements
   z[index] = x[index] + y[index];
}
```

Project structure

- The directory `common` includes helper functions
  - **options.h**: C/C++ command parser
  - **opencl.h** : Custom OpenCL function
  - **scoped_ptrs**: smart pointers implementation
- No mandatory to use it but it can be really helpful
- The directory `vector_add` contains:
  - A Makefile to compile the host
  - A directory `**device**` containing the OpenCL kernel
  - A directory `**host**` containing the host code

```
~/sshfs/mel/home/FPGA-OpenCL  tree
.
├── common
│   ├── inc
│   │   └── AOCLUtils
│   │       ├── aocl_utils.h
│   │       ├── opencl.h
│   │       ├── options.h
│   │       └── scoped_ptrs.h
│   ├── readme.css
│   └── src
│       └── AOCLUtils
│           ├── opencl.cpp
│           └── options.cpp
└── vector_add
    ├── device
    │   └── vector_add.cl
    ├── host
    │   └── src
    │       └── main.cpp
    ├── Makefile
    └── README.html

10 directories, 11 files
```

Makefile

- Makefile provided by Intel
- You can also generate a makefile template using:
  - `aocl makefile`
- Important variables:
  - `AOCL_COMPILE_CONFIG := $(shell aocl compile-config )`
  - `AOCL_LINK_LIBS := $(shell aocl ldlibs )`
  - `AOCL_LINK_FLAGS := $(shell aocl ldflags )`
- aocl provides options to retrieve all compile, link and flags options

```
ifeq ($(VERBOSE),1)
ECHO := 
else
ECHO := @
endif

# Where is the Intel(R) FPGA SDK for OpenCL(TM) software?
ifeq ($(wildcard $(INTELFPGAOCLSDKROOT)),)
$(error Set INTELFPGAOCLSDKROOT to the root directory of the Intel(R) FPGA SDK for OpenCL(TM) software installation)
endif
ifeq ($(wildcard $(INTELFPGAOCLSDKROOT)/host/include/CL/opencl.h),)
$(error Set INTELFPGAOCLSDKROOT to the root directory of the Intel(R) FPGA SDK for OpenCL(TM) software installation.)
endif

# OpenCL compile and link flags.
AOCL_COMPILE_CONFIG := $(shell aocl compile-config )
AOCL_LINK_LIBS := $(shell aocl ldlibs )
AOCL_LINK_FLAGS := $(shell aocl ldflags )
# Linking with defences enabled
AOCL_LINK_FLAGS += -z noexecstack
AOCL_LINK_FLAGS += -Wl,-z,relro,-z,now
AOCL_LINK_FLAGS += -Wl,-Bsymbolic
AOCL_LINK_FLAGS += -pie
AOCL_LINK_CONFIG := $(AOCL_LINK_FLAGS) $(AOCL_LINK_LIBS)

# Compilation flags
ifeq ($(DEBUG),1)
CXXFLAGS += -g
else
CXXFLAGS += -O2
endif

# Compiling with defences enabled
CXXFLAGS += -fstack-protector
CXXFLAGS += -D_FORTIFY_SOURCE=2
CXXFLAGS += -Wformat -Wformat-security
CXXFLAGS += -fPIE

# We must force GCC to never assume that it can shove in its own
# sse2/sse3 versions of strlen and strcmp because they will CRASH.
# Very hard to debug!
CXXFLAGS += -fPIC

# Compiler
CXX := g++

# Target
TARGET := host
TARGET_DIR := bin

# Directories
INC_DIRS := ../common/inc
LIB_DIRS := 

# Files
INCS := $(wildcard )
SRCS := $(wildcard host/src/*.cpp ../common/src/AOCLUtils/*.cpp)
LIBS := rt pthread

# Make it all!
all : $(TARGET_DIR)/$(TARGET)

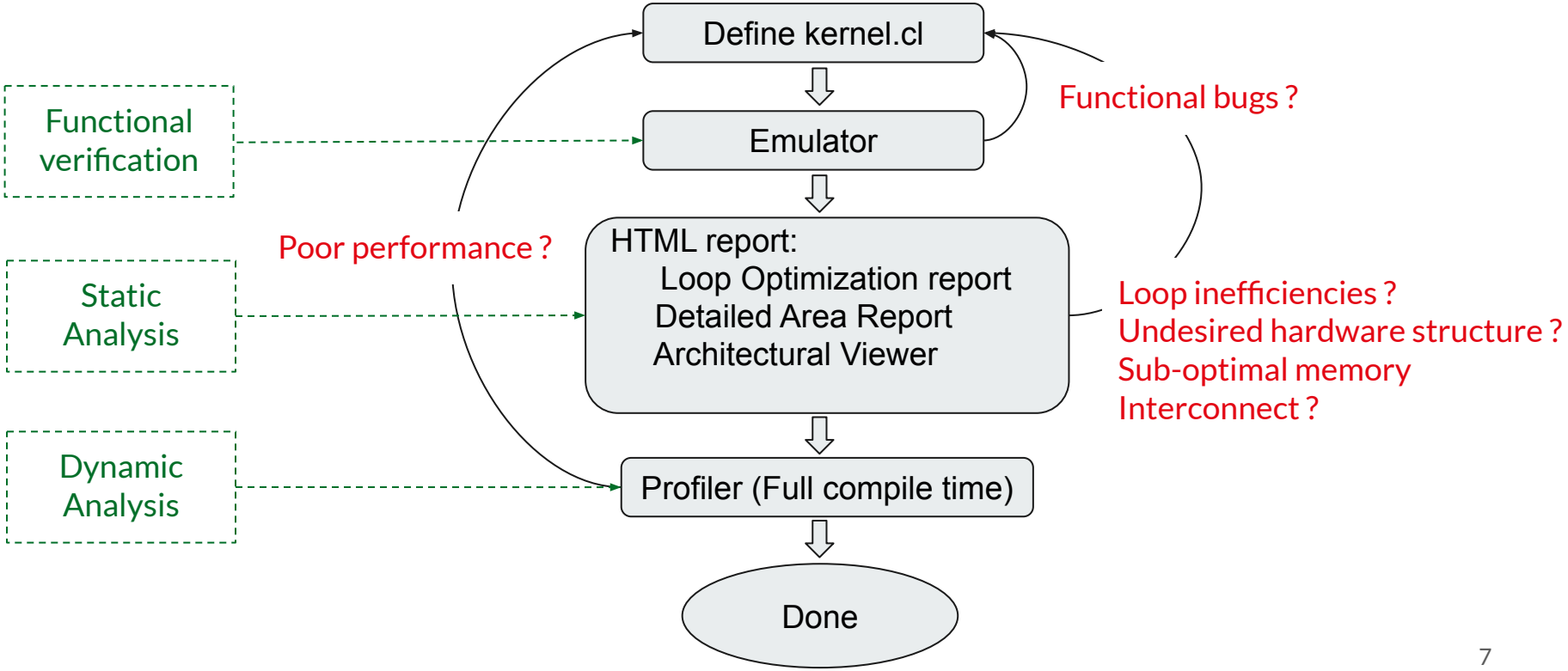
# Host executable target.
$(TARGET_DIR)/$(TARGET) : Makefile $(SRCS) $(INCS) $(TARGET_DIR)
    $(ECHO)$(CXX) $(CPPFLAGS) $(CXXFLAGS) $(EXTRACXXFLAGS) -fPIC $(foreach D,$(INC_DIRS),-I$D) \
            $(AOCL_COMPILE_CONFIG) $(SRCS) $(AOCL_LINK_CONFIG) \
            $(foreach D,$(LIB_DIRS),-L$D) \
            $(foreach L,$(LIBS),-l$L) \
            -o $(TARGET_DIR)/$(TARGET)

$(TARGET_DIR) :
    $(ECHO)mkdir $(TARGET_DIR)

# Standard make targets
clean :
    $(ECHO)rm -f $(TARGET_DIR)/$(TARGET)

.PHONY : all clean
```

OpenCL design flow



Emulation phase

UNIVERSITÉ DU LUXEMBOURG

Emulation of FPGA OpenCL kernels

- Hardware synthesis can be very long
- Emulation is a practical way of testing your OpenCL kernels

Design properties	Estimated time	Estimated memory	Suggested arguments for *fpgasyn* partition
Low resource utilization (<10% in Kernel System) Simple memory interface (Global interconnect for < 10 global loads + stores) Loops with low to medium latency (<500 cycles)	2-4h	45 GB	--mem=45000MB --time=8:00:00
Medium resource utilization (<40% ALUTs and FFs, and <60% RAMs and DSPs in Kernel System) Simple to medium memory interface (Global interconnect for < 20 global loads + stores) Loops with low to medium latency (<500 cycles)	8-12h	60-90 GB	--mem=90000MB --time=24:00:00
High resource utilization (>50% ALUTs and FFs, or >70% RAMs and DSPs in Kernel System) Simple to medium memory interface (Global interconnect for < 20 global loads + stores) Loops with low to medium latency (<500 cycles)	12-20h	90-120 GB	--mem=120000MB --time=48:00:00
Any resource utilization Simple to medium memory interface (Global interconnect for > 100 global loads + stores) or Loops with high to very high latency (>2000 cycles)	30-60h	120+ GB	--exclusive --time=72:00:00

(source: wikis.uni-paderborn.de)

Emulation of FPGA OpenCL kernels

- Two emulation modes:
  - Fast emulation
  - Legacy emulation (still supported but deprecated)

Intel FPGA SDK for OpenCL Versions	Legacy Emulation	Fast Emulation	Recommended for Noctua
18.x.x	**default**	not available	Legacy Emulation
19.1.0	**default**	-fast-emulator	Legacy Emulation
19.2.0	**-legacy-emulator**	default	Legacy Emulation
19.3.0	-legacy-emulator	**default**	Fast Emulation
19.4.0	-legacy-emulator	**default**	Fast Emulation
20.1.0	-legacy-emulator	**default**	Fast Emulation
20.2.0	-legacy-emulator	**default**	Fast Emulation

(source: wikis.uni-paderborn.de)

Legacy emulation of FPGA OpenCL kernels

- OpenCL 1.0
- vector_add example
  - `cd  FPGA-OpenCL/vector_add/ device`
  - `aoc -v -Wall -march=emulator -legacy-emulator  -o ../bin/vector_add.aocx vector_add.cl`
  - `cd .. && make`
  - `CL_CONTEXT_EMULATOR_DEVICE_INTELFPGA=1 ./host`

```
[u100057@mel3017 bin]$ aoc -v -Wall -march=emulator -legacy-emulator  -o ../bin/vector_add.aocx ../device/vector_add.cl
aoc: Environment checks completed successfully.
aoc: Cached files in /var/tmp/aocl/ may be used to reduce compilation time
You are now compiling the full flow!!
aoc: Selected target board package /apps/USE/easybuild/staging/2022.1/software/520nmx/20.4
aoc: Selected target board p520_hpc_m210h_g3x16
aoc: Running OpenCL parser....
aoc: OpenCL parser completed
aoc: Linking Object files....
aoc: Compiling for Emulation ....
aoc: Emulator Compilation completed successfully.
Emulator flow is successful.
To execute emulated kernel, invoke host with
        env CL_CONTEXT_EMULATOR_DEVICE_INTELFPGA=1 <host_program>
 For multi device emulations replace the 1 with the number of devices you wish to emulate
[u100057@mel3017 bin]$ CL_CONTEXT_EMULATOR_DEVICE_INTELFPGA=1 ./host
Initializing OpenCL
Platform: Intel(R) FPGA SDK for OpenCL(TM)
Using 1 device(s)
  EmulatorDevice : Emulated Device
Using AOCX: vector_add.aocx
Launching for device 0 (1000000 elements)

Time: 1667.388 ms
Kernel time (device 0): 1663.478 ms

Verification: PASS
[u100057@mel3017 bin]$
```

Fast emulation of FPGA OpenCL kernels

- OpenCL > 1.0
- vector_add example
  - `cd  FPGA-OpenCL/vector_add/device`
  - `aoc -v -Wall -march=emulator  -o ../bin/vector_add.aocx vector_add.cl`
  - `cd .. && make`
  - `./host -emulator`

```
[u100057@mel3017 bin]$ aoc -v -Wall -march=emulator  -o ../bin/vector_add.aocx ../device/vector_add.cl
aoc: Cached files in /var/tmp/aocl/ may be used to reduce compilation time
You are now compiling the full flow!!
aoc: Selected target board package
aoc: Selected target board
aoc: OpenCL kernel compilation completed successfully.
aoc: Linking Object files....
aoc: Compiling for Emulation ....
aoc: Emulator Compilation completed successfully.
Emulator flow is successful.
To execute emulated kernel, ensure host code selects the Intel(R) FPGA OpenCL
emulator platform.
[u100057@mel3017 bin]$ ./host -emulator
Initializing OpenCL
Platform: Intel(R) FPGA Emulation Platform for OpenCL(TM)
Using 1 device(s)
  Intel(R) FPGA Emulation Device
Using AOCX: vector_add.aocx
Launching for device 0 (1000000 elements)

Time: 15.809 ms
Kernel time (device 0): 0.666 ms

Verification: PASS
[u100057@mel3017 bin]$
```

Notes on Emulation of FPGA OpenCL kernels

- **NOT** required to have a FPGA card
- **Required** to have a BSP
- Emulation is executed sequentially like a typical C-code
- Parallelism is ignored during emulation
- Always emulate small parts because you cannot take advantage of emulation
- Emulation is useful to debug your application, not for performance analysis !!

Performance Tuning phase

UNIVERSITÉ DU LUXEMBOURG

Multi-stage compilation

- The OpenCL™ Offline Compiler generate intermediate object file for each source file (.cl) and links them without building the hardware => this is a multi-stage compilation
- To compile one or several kernel source files and obtain all intermediate files for analysis
  - ```
    aoc -v -rtl -board=p520_hpc_m210h_g3x16 ../device/vector_add.cl
    ```
- Invoking the aoc command with the -rtl flag, the offline compiler compiles the kernels and creates the following files and directories:
  - An intermediate .aoco file for each .cl kernel source file. You must specify the -save-temps aoc command option.
  - A .aocr file generated after linking then links them and generates a .aocr file.
  - A <your_kernel_filename> folder or subdirectory containing HTML reports
    ```

HTML static report

- First compilation phase without hardware generation (-rtl flags)
  - Kernel code is converted to HDL code
  - Compilation reports are generated
  - During this compilation step, static html reports are generated:
    - `<your_kernel_filename>/reports/report.html file`
    - It provides kernel analytical data such as area and memory usages, as well as loop structure and kernel pipeline information
  - Get Actual resource utilization and Clock frequency
    - `<your_kernel_filename>/ acl_quartus_report .txt`

HTML static report

- Provide information information resources utilization
- A source code pane is also provided on the right of the report

Reports | Summary | Views * | Throughput Analysis * | Area Analysis *

Summary Content

- Compile Info
- Kernels Summary
- Clock Frequency Summary
- System Resource Utilization Summary
- Quartus Fitter Resource Utilization Summary
- Compile Estimated Kernel Resource Utilization Summary
- Warnings Summary

Summary

Compile Info

Project Name	data_dep
Target Family, Device, Board	Stratix 10, 1SM21BHU2F53E2VCS1, bitt_s10mx_pcieg520_hpx_m210h_g3x16
SYCL Version	2022.2.0 Build 133.4
Quartus Version	
Reports Generated At	Wed Apr 26 11:19:16 2023

Kernels Summary

Name	Source Location	Kernel Type	Autorun	Workgroup Size	# Compute Units	Target Frequency (MHz)	Hyper-Optimized Handshaking
data_dep	data_dep.cl:3	Single work-item	No	1,1,1	1	Not specified	Off

Clock Frequency Summary

System Resource Utilization Summary

Quartus Fitter Resource Utilization Summary

Compile Estimated Kernel Resource Utilization Summary

Warnings Summary

data_dep.cl

```
1  #define N 1024*1024
2
3  __kernel void data_dep( __global const int * restrict X,
4                          __global const int * restrict Y,
5                          __global int * restrict Z )
6  {
7    unsigned int sum = 0;
8    for (unsigned int i = 0; i < N; i++){
9        for (unsigned int j = 0; j < N; j++){
10         sum += Y[j]*X[j];
11       }
12       Z[i]= sum;
13     sum=0;
14   }
15   }
16
17
18
19
```

Loop analysis

Reports | Summary | Views | Throughput Analysis | Area Analysis

Loop List

- Kernel System
  - Kernel: memory_dependency (data_dep.cl:1)
    - memory_dependency.B2 (data_dep.cl:4)

Loop Analysis — Show blocks

Name	Source Location	Pipelined	Block Scheduled II	Block Scheduled fMAX	Latency	Speculated Iterations	Max Interleaving Iterations	Brief Info
Kernel: memory_dependency	data_dep.cl:3							Single work-item execution
memory_dependency.B2	data_dep.cl:4	Yes	243	480.00	251.000000	0	1	Memory dependency

data_dep.cl

```
1  __kernel void memory_dependency( int max_i,
2                                   __global int* restrict out)
3  {
4    for (int i = 1; i < max_i; i++) {
5      out[max_i*2-i] = out[i];
6    }
7  }
```

Bottlenecks

- Throughput bottlenecks
  - memory_dependency
    - Global variable (in memory_dependency.B2)

Details

memory_dependency.B2:

- Hyper-Optimized loop structure: disabled.
- Compiler failed to schedule this loop with smaller II due to memory dependency:
  - From: Load Operation (data_dep.cl: 5)
  - To: Store Operation (data_dep.cl: 5)
- Most critical loop feedback path during scheduling:

Area Analysis of System

Reports | Summary | Views | Throughput Analysis | Area Analysis

Loop List

- Kernel System
  - Kernel: memory_dependency (data_dep.cl:1)
    - memory_dependency.B2 (data_dep.cl:4)

Loop Analysis — Show blocks

Name	Source Location	Pipelined	Block Scheduled II	Block Scheduled fMAX	Latency	Speculated Iterations	Max Interleaving Iterations	Brief Info
Kernel: memory_dependency	data_dep.cl:3							Single work-item execution
memory_dependency.B2	data_dep.cl:4	Yes	243	480.00	251.000000	0	1	Memory dependency

data_dep.cl

```
1  __kernel void memory_dependency( int max_i,
2                                   __global int* restrict out)
3  {
4    for (int i = 1; i < max_i; i++) {
5      out[max_i*2-i] = out[i];
6    }
7  }
```

Bottlenecks

- Throughput bottlenecks
  - memory_dependency
    - Global variable (in memory_dependency.B2)

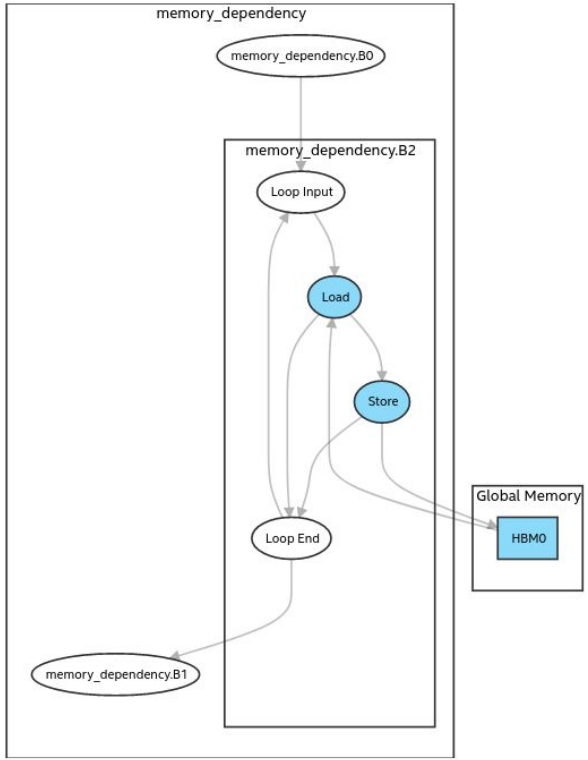
Details

memory_dependency.B2:

- Hyper-Optimized loop structure: disabled.
- Compiler failed to schedule this loop with smaller II due to memory dependency:
  - From: Load Operation (data_dep.cl: 5)
  - To: Store Operation (data_dep.cl: 5)
- Most critical loop feedback path during scheduling:

System viewer

- Structure of the FPGA acceleration
  - Kernels represented as combination of blocks
  - Information by clicking on block to see:
    - Memory access
    - Latency
    - Coalescing



Profiling

UNIVERSITÉ DU LUXEMBOURG

System viewer

- Dynamic analysis of the kernel helps to evaluate resource overuses and possible optimization
- Profiling the kernel requires the final hardware image (.aocx)
  - Adding the “--profile” option to generate performance counters
  - You need to execute the kernel to fill the profiling results into “profile.mon”
- Profiling may impact the clock frequency
- Processing time should not be evaluated with the “--profile” option
  - “--profile=all”: profile all kernels
  - “--profile=autorun”: profile autorun kernels
  - “--profile=enqueued”: profile only non-autorun kernels

Profiling Data During Runtime

- Two possibilities to obtain profiling data during runtime
  - Running Profiler Runtime Wrapper from the command line to obtain the data, i.e. profile.json file
  - Running your host application in Intel VTune Profiler using the CPU/FPGA Interaction
- Intel® VTune™ Profiler User Guide

Profiler Runtime Wrapper (PRW) from CLI

- Device and host compilation must be completed. **Don't forget the "--profile=all" option !!**

```
u100057@mel3017  ~/.../vector_add/bin  main  aocl profile -x vector_add.aocx host
Initializing OpenCL
Platform: Intel(R) FPGA SDK for OpenCL(TM)
Using 2 device(s)
Setting profiler start cycle from environment variable: 1
Setting profiler start cycle from environment variable: 1
  p520_hpc_m210h_g3x16 : BittWare Stratix 10 MX OpenCL platform (aclbitt_s10mx_pcie0)
Setting profiler start cycle from environment variable: 1
Setting profiler start cycle from environment variable: 1
  p520_hpc_m210h_g3x16 : BittWare Stratix 10 MX OpenCL platform (aclbitt_s10mx_pcie1)
Setting profiler start cycle from environment variable: 1
Setting profiler start cycle from environment variable: 1
Using AOCX: vector_add.aocx
Launching for device 0 (500000 elements)
Launching for device 1 (500000 elements)

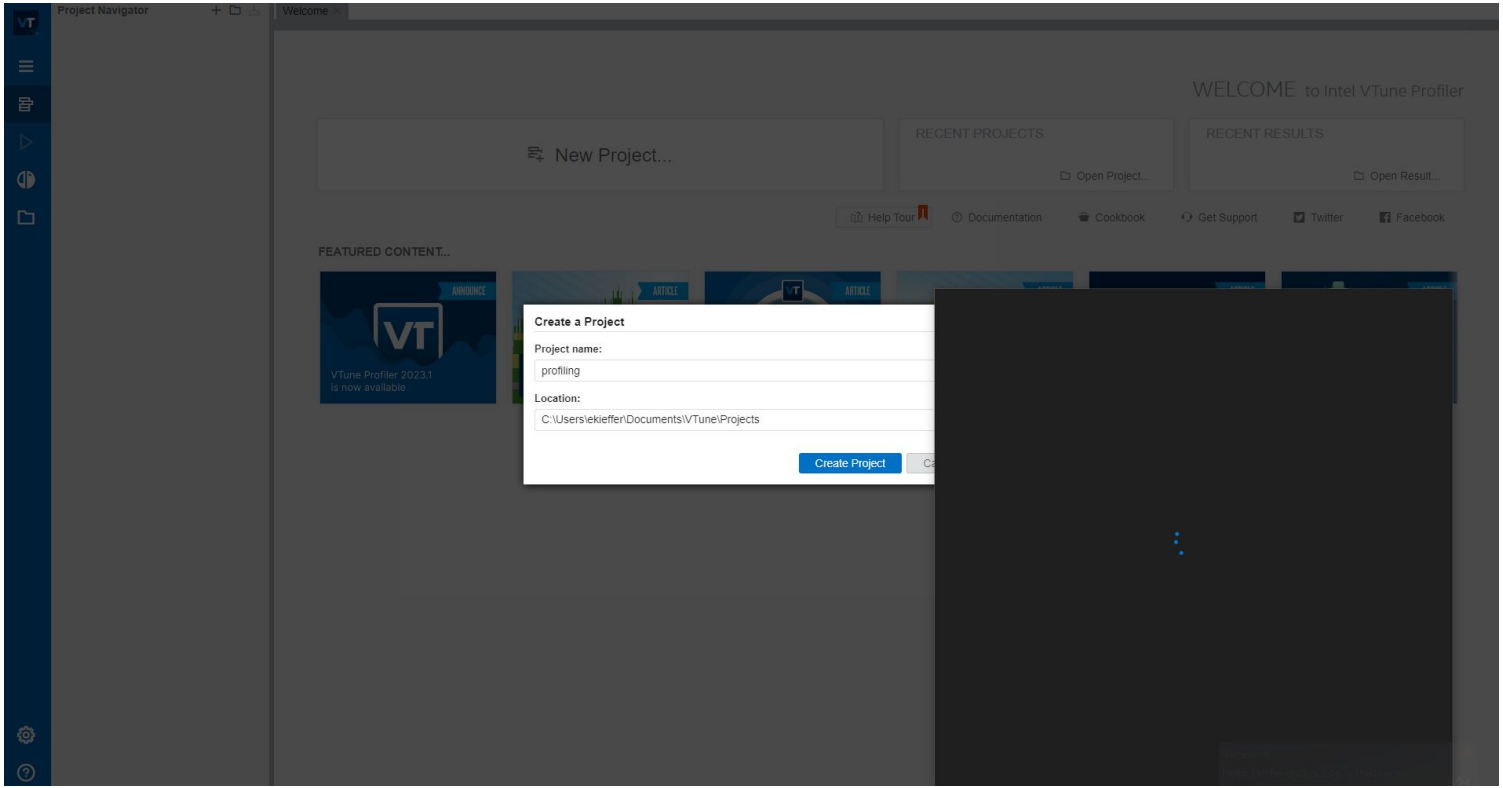
Time: 5.241 ms
Kernel time (device 0): 1.236 ms
Kernel time (device 1): 1.245 ms

Verification: PASS
Starting post-processing of the profiler data...
Post-processing complete.
```

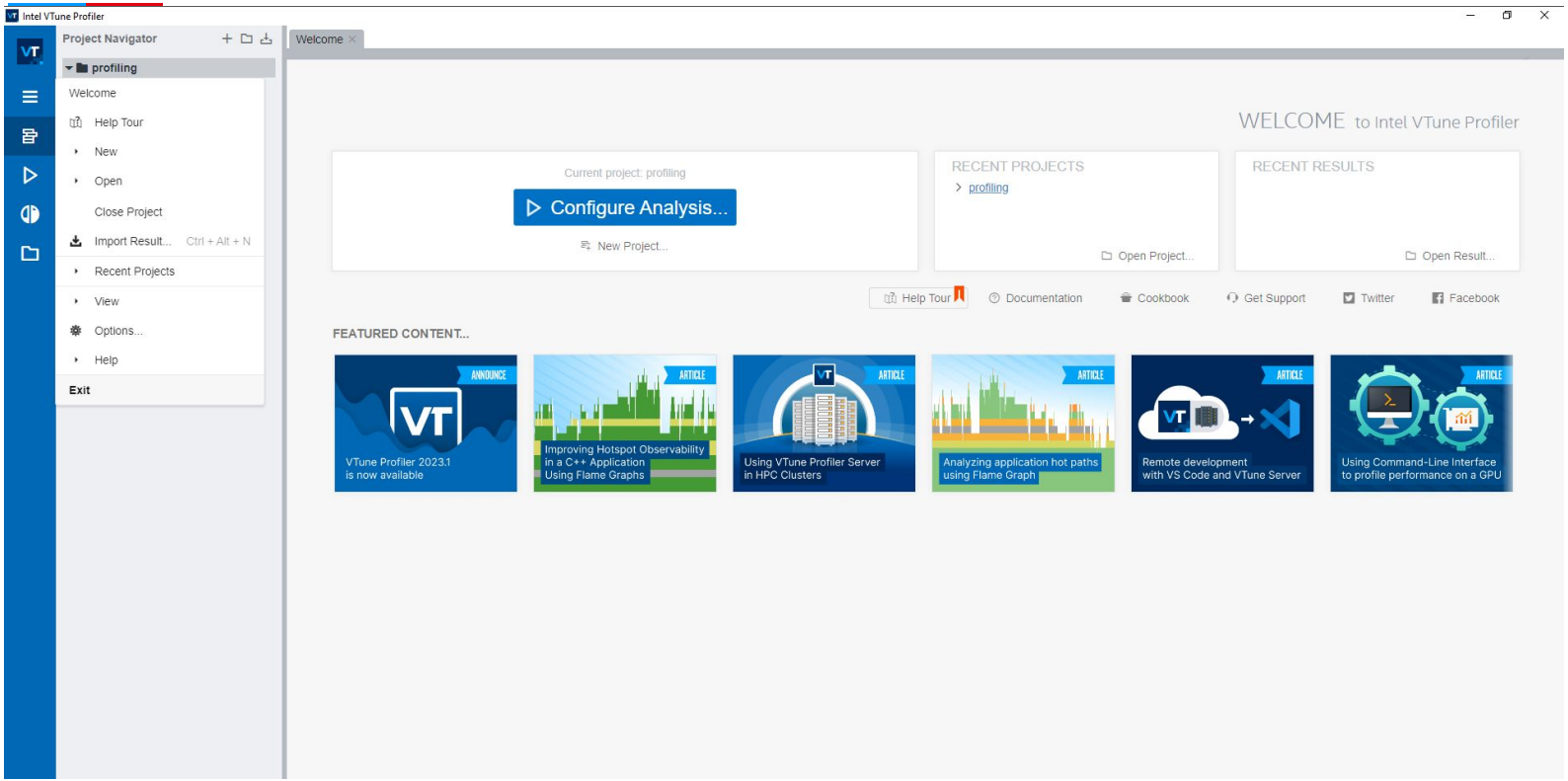
Profiler Runtime Wrapper (PRW) from CLI

- You should see two new files after calling the PRW:
  - `profile.mon` : data is collected from the performance counters and stored in this file
  - `profile.json` : PRW then process the `profile.mon` file and converts into a readable `profile.json` file
- Install the Intel VTune profiler (free)
- Create an empty directory, e.g. , `profiling`
- Copy-paste the profile.json file to the folder

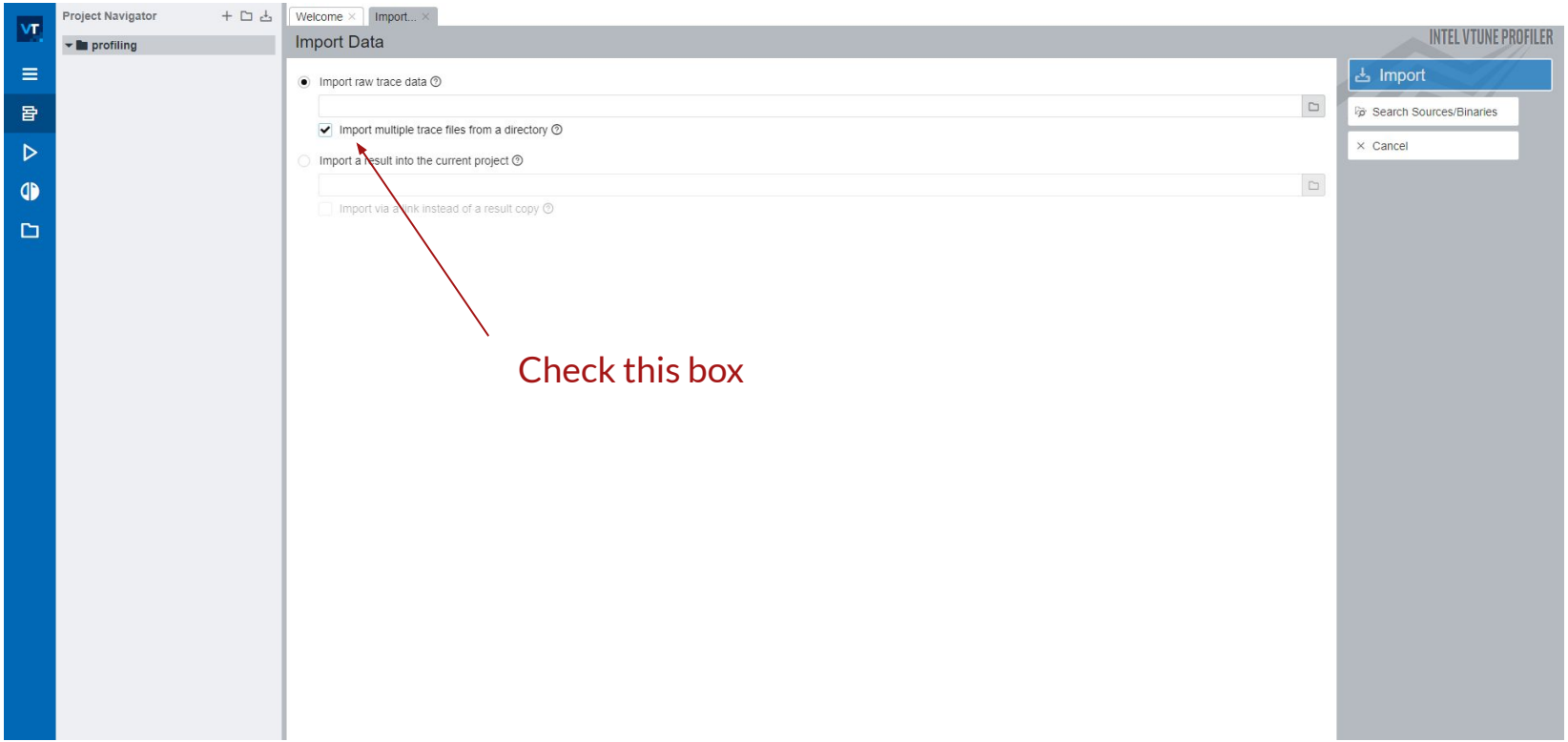
Create a new VTune profiler project



Import results



Import the profiling directory



CPU/FPGA Interaction analysis results

Project Navigator

profiling

r000

Welcome | r000

INTEL VTUNE PROFILER

Analysis Configuration | Collection Log | Summary | Bottom-up | Platform

Elapsed Time: 0.002s

Computing Task Time: 0.007s
Total Thread Count: 0
Paused Time: 0s

FPGA Top Compute Tasks

This section lists the most active FPGA compute tasks in your application.

Computing Task (FPGA)	Computing Task Time	Computing Task Count
vector_add	0.007s	8

*N/A is applied to non-summable metrics.

Collection and Platform Info

This section provides information about this collection, including result set size and collection platform data.

Result Size: 3.2 MB

Finalization mode: Fast. If the number of collected samples exceeds the threshold, this mode limits the number of processed samples to speed up post-processing.

CPU

Logical CPU Count: 0

Cache Allocation Technology

Level 2 capability: not detected
Level 3 capability: not detected

View Data

- The viewpoint contains these windows:
  - The **Summary window** displays statistics on the overall application execution, identifying CPU time and processor utilization, and execution time or OpenCL kernels.
  - **The Bottom-up window** displays functions in the Bottom-up tree, CPU time and CPU utilization per function.
  - **The Platform window** displays over-time metric and performance data for OpenCL kernels, memory transfers, CPU context switches, FPU utilization, and CPU threads with OpenCL kernels.

View source file

VirtWin (Snap28042020) [Running] - Oracle VM VirtualBox

File View Input Devices Help

Intel VTune Profiler

Project Navigator

profiling

r001

Welcome | r001

Analysis Configuration | Collection Log | Summary | Bottom-up | Platform | vector_add.cl

INTEL VTUNE PROFILER

Source | Assembly

Source Line	Source	Device Metrics: Stalls (%)	Occupancy (%)	Idle (%)	Activity (%)	Data Transferred: Data Transfer Size	Average Bandwid
17	// OTHER DEALINGS IN THE SOFTWARE.						
2	// Permission is hereby granted, free of charge, to any person obtaining a copy of this						
3	// software and associated documentation files (the "Software"), to deal in the Software						
4	// without restriction, including without limitation the rights to use, copy, modify, merge,						
5	// publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to						
6	// whom the Software is furnished to do so, subject to the following conditions:						
7	// The above copyright notice and this permission notice shall be included in all copies or						
8	// substantial portions of the Software.						
9	//						
10	// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,						
11	// EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES						
12	// OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND						
13	// NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT						
14	// HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,						
15	// WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING						
16	// FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR						
33							
18	//						
19	// This agreement shall be governed in all respects by the laws of the State of California and						
20	// by the laws of the United States of America.						
21							
22	// ACL kernel for adding two input vectors						
23	__kernel void vector_add(__global const float *x,						
24	__global const float *y,						
25	__global float *restrict z)						
26	{						
27	// get index of the work item						
28	int index = get_global_id(0);						
29							
30	// add the vector elements						
1	// Copyright (C) 2013-2020 Altera Corporation, San Jose, California, USA. All rights reserved.						
32	}						
31	z[index] = x[index] + y[index];	4.2%	98.3%	1.7%	98.3%	11.6 MB	

Performance metrics

UNIVERSITÉ DU LUXEMBOURG

Metrics interpretation

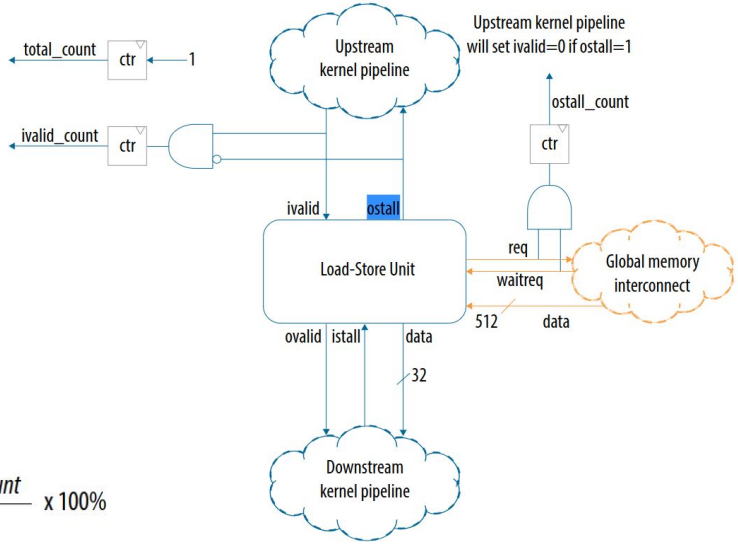
- **Stall%** : percentage of the time the memory or channels[1] is causing pipeline to stall
- **Occupancy%** : occupancy refers to the percentage of the time required for memory or channel access in total processing time.
- **Bandwidth** : global-memory access throughput during the kernel execution.
- **Idle%** : percentage of the overall profiled time frame when there are no valid work item executing or stalling the memory or channel instruction.
- **Channel Depth** : occupancy of the channel FIFO (in bytes) when the channel is not idling

1. We will see channels more in details later on. In a nutshell, a channel is a dedicated data-path between kernel to avoid the need to access global memory

Ideal Values

"An ideal kernel pipeline condition has a stall percentage of 0%, an occupancy percentage of 100%, and a bandwidth that equals the board's available bandwidth"

- All metrics are obtained with the performance counters
- You have to use "Profiling" in order to activate them



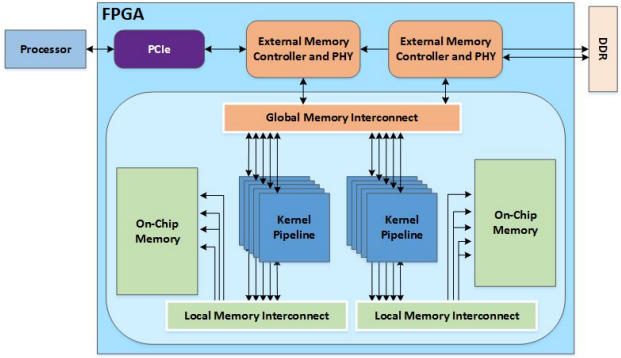
$$\text{Stall} = \frac{ostall\_count}{total\_count} \times 100\%$$

$$\text{Occupancy} = \frac{ivalid\_count}{total\_count} \times 100\%$$

$$\text{Bandwidth} = \frac{data\_width \times ivalid\_count}{kernel\_time} \times 100\%$$

(source: Intel)

High Stall Percentage



- Global memory access or channel data unavailable
- <u>Reasons</u>:
  - Linked to high occupancy
    - Intensive data access in few cycles, i.e. typically 1 clock-cycle
    - **Prefer local memory**
  - Imbalanced producer-consumer pattern for channel
    - Different speed for read-write

Low Occupancy

- Low occupancy in **non-critical paths**
- <u>Critical paths</u>: **"a critical path refers to the sequence of operations within the kernel that determines the longest execution time."**

```
__kernel void example (__global int * restrict x, __global int * restrict y) {
float data0,data1;
for (int i = 0; i < N; i++) {
      for (int j = 0; j < 1000; j++) {
            data0 = x[i*100+j] + 40;
            write_channel_intel (c0, data0);
      }
      for (int k = 0; k < 3; k++) {
            data1 = y[i*100+j] - 50;
            write_channel_intel (c1, data1);
      }}}
```

critical path

non-critical path

low occupancy is ok

Low Bandwith Efficiency

- When excessive amount of bandwidth usage with poor memory accesses (e.g. random accesses)
- Solution:
  - Local memory
  - Memory coalescing

No Stall, Low Occupancy, and Low Bandwidth

- Serial execution due to data dependencies

Reports | Summary | Views | Throughput Analysis | Area Analysis

Loop List

- Kernel System
  - Kernel: NoStallNoOccNoBand (NoStallNoOccNoBand.cl:3)
    - NoStallNoOccNoBand.B1 (NoStallNoOccNoBand.cl:8)
      - NoStallNoOccNoBand.B4 (NoStallNoOccNoBand.cl:9)
      - NoStallNoOccNoBand.B6 (NoStallNoOccNoBand.cl:11)

Loop Analysis

Show blocks

Name	Source Location	Pipelined	Block Scheduled II	Block Scheduled fMAX	Latency	Speculated Iterations	Max Interleaving Iterations	Brief Info
NoStallNoOccNoBand.B1	NoStallNoOccNoBand.cl:8	Yes	1	480.00	52.000000	4	1	Serial exe: Memory dependency
NoStallNoOccNoBand.B4	NoStallNoOccNoBand.cl:9	Yes	1	480.00	836.000000	4	1	

NoStallNoOccNoBand.cl

```
1  #define N 4096
2
3  __kernel void NoStallNoOccNoBand(__global int *restrict x,
4                                   __global int *restrict y,
5                                   __global int *restrict z,
6                int M){
7
8    for(unsigned int i=0; i<N; i++){
9      for(unsigned int j=0; j<2048; j++)
10       y[i*2048+j] = x[i*2048+j] + 40;
11     for(unsigned k=M; k<2048; k++)
12       z[i*2048+k] = y[i*2048+k] * 5;
```

Details

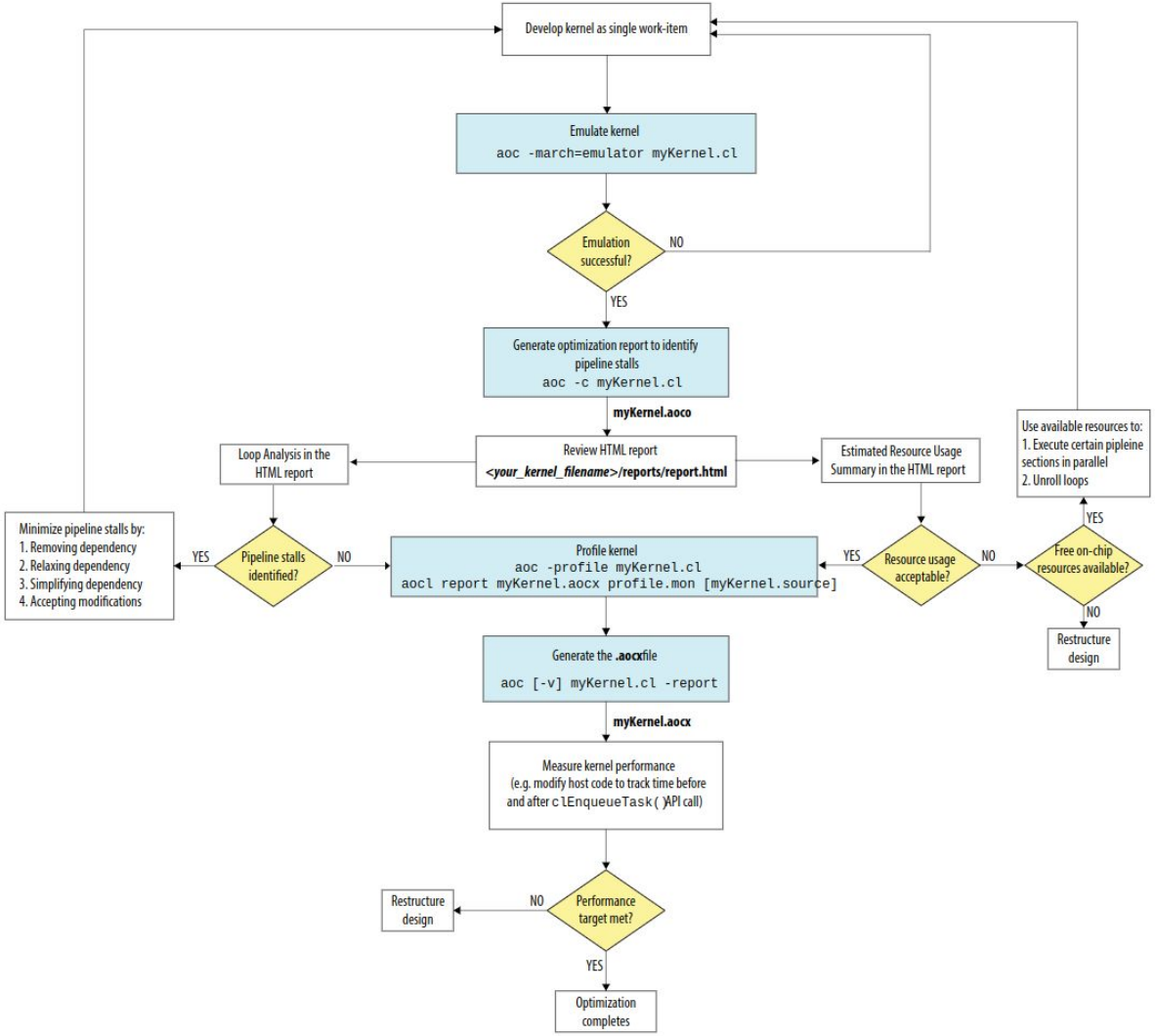
NoStallNoOccNoBand.B1:

- Hyper-Optimized loop structure: enabled.
- Iteration executed serially across NoStallNoOccNoBand.B4, NoStallNoOccNoBand.B6. Only a single loop iteration will execute inside this region due to memory dependency:
  - From: Store Operation (NoStallNoOccNoBand.cl: 10)
  - To: Load Operation (NoStallNoOccNoBand.cl: 12)
- Stallable instruction: n/a

No Stall, Low Occupancy, and Low Bandwidth

- Serial execution due to data dependencies
- Ran on Stratix 10 FPGA Board with 16GB HBM2 at up to 512GB/s

Computing Task / Module Instance / Compute Unit	Computing Task			Device Metrics								Number of Compute Units
	Total Time	Average Time	Instance Count	Stalls (%)	Occupancy (%)	Idle (%)	Activity (%)	Data Transferred, Global				
								Size	Average Bandwidth, GB/s	Total Size	Total Average Bandwidth, GB/s	
NoStallNoOccNoBand	39.240ms	39.240ms	1	0.5%	48.7%	36.8%	20.2%	108.5 MB	2.765	108.5 MB	2.765	1


Develop kernel as single work-item
Emulate kernel
aoc -march=emulator myKernel.cl
Emulation successful?
NO
YES
Generate optimization report to identify pipeline stalls
aoc -c myKernel.cl
myKernel.aoco
Review HTML report
<your_kernel_filename>/reports/report.html
Loop Analysis in the HTML report
Estimated Resource Usage Summary in the HTML report
Use available resources to:
1. Execute certain pipleine sections in parallel
2. Unroll loops
Minimize pipeline stalls by:
1. Removing dependency
2. Relaxing dependency
3. Simplifying dependency
4. Accepting modifications
YES
Pipeline stalls identified?
NO
Profile kernel
aoc -profile myKernel.cl
aocl report myKernel.aocx profile.mon [myKernel.source]
YES
Resource usage acceptable?
NO
YES
Free on-chip resources available?
NO
Restructure design
Generate the .aocx file
aoc [-v] myKernel.cl -report
myKernel.aocx
Measure kernel performance
(e.g. modify host code to track time before and after clEnqueueTask( )API call)
Restructure design
NO
Performance target met?
YES
Optimization completes

Notes

- Use local VTune to make the profiling
  - Careful the source file path code will be wrong
  - You can adapt it after the results import