MOST 科技部
Ministry of Science and Technology
A世代半導體計畫
Angstrom Semiconductor Initiative

LabA: Host Code Optimization

Speaker: Edan
Advisor: Prof. Jiin Lai
Date: 2022/10/20

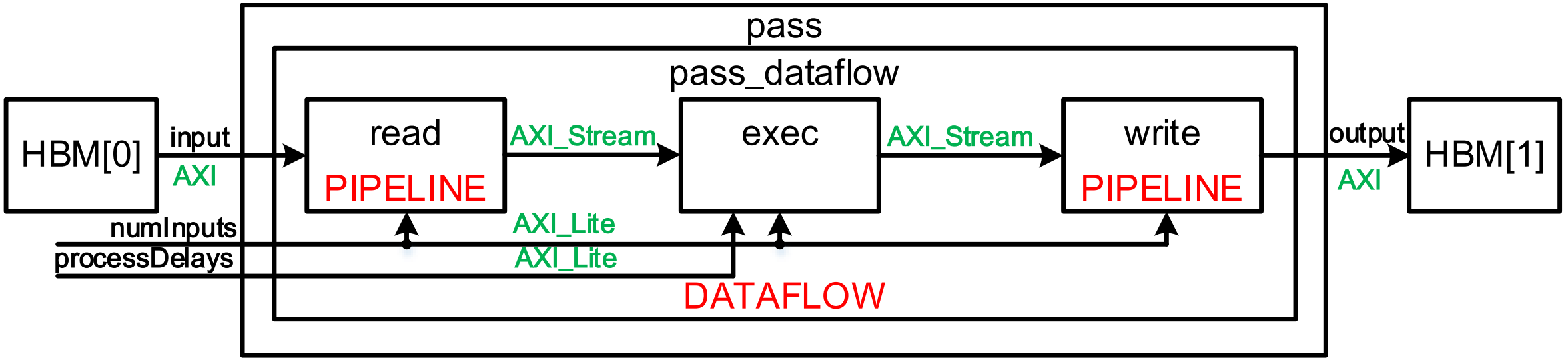
Performance Optimization of HLS

MOST 科技部 Ministry of Science and Technology
A世代半導體計畫 Angstrom Semiconductor Initiative

- Host program optimization
  - Schedule the tasks to increase the u-rate of kernels
- Kernel code optimization
  - Pipeline/parallelize the kernel
- Topological optimization
  - Connect a specific DDR to a specific kernel
- Implementation optimization
  - DSP or LUT for the kernel

We will focus on the ***host program optimization***

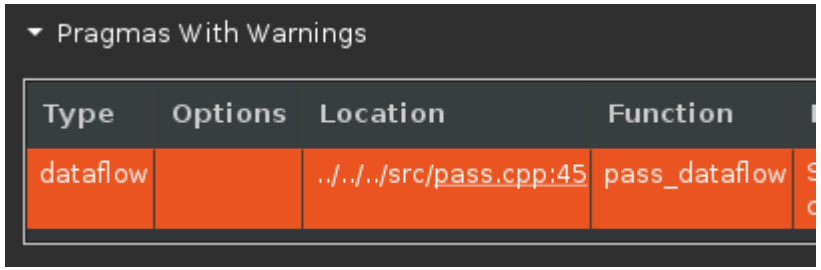
System Diagram

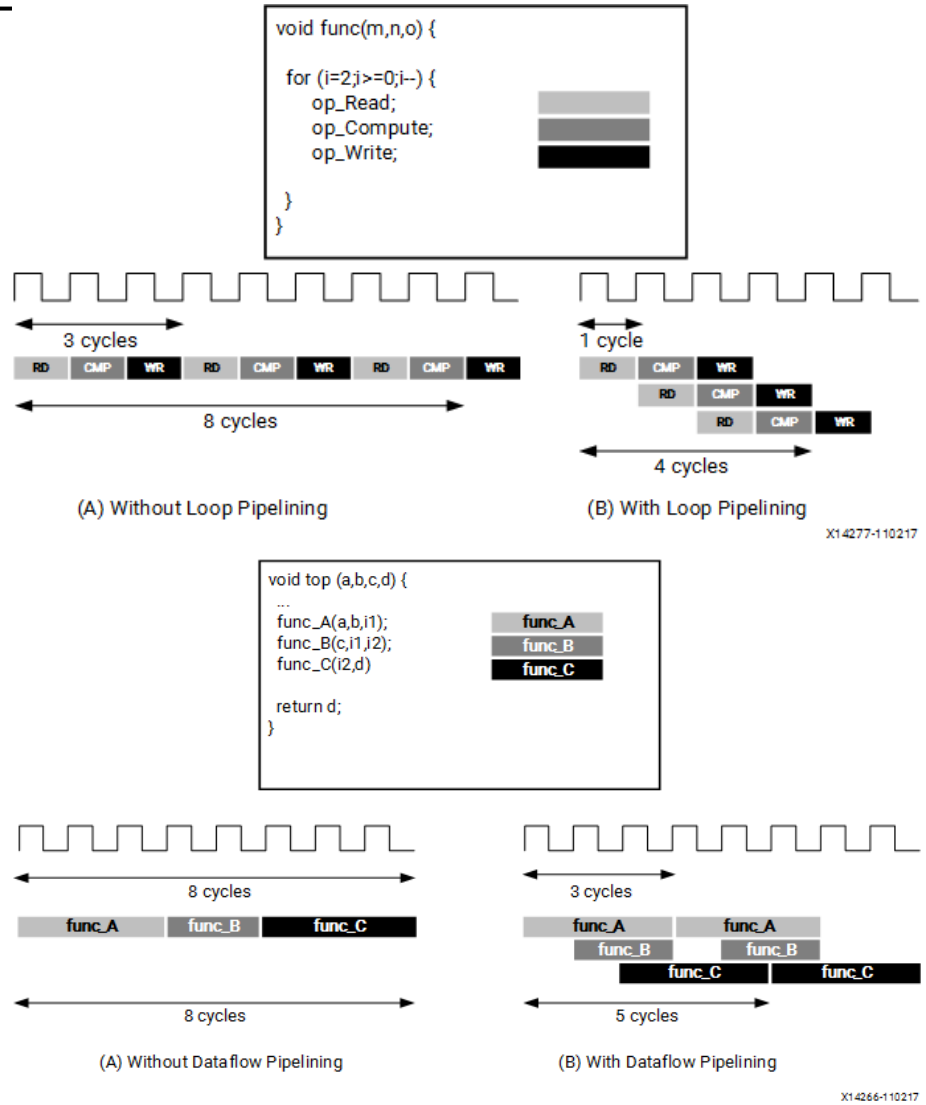


- read: convert AXI to AXI_Stream
- write: convert AXI_Stream to AXI
- exec: add processDelays to the input and write out the result after processDelays cycles

Used Pragma

- PIPELINE
  - Reduces the II into 1 as possible
  - Depends on the read and write latency
    - ➢ How many copies of HW required
- DATAFLOW
  - Enable task-level pipelining
  - Especially for adjacent for loops
  - Also reduce the II of the entire function
  - Not working in current example
    - ➢ Shown in the result





Code of Module-pass

- read

```
void read(const ap_int<512>         *input,
    hls::stream<ap_int<512> > &inStream,
    unsigned int              numInputs) {
  for(unsigned int i = 0; i < numInputs; i++) {
    #pragma HLS PIPELINE
    inStream.write(input[i]);
  }
}
```

- write

```
void write(hls::stream<ap_int<512> > &outStream,
     ap_int<512>               *output,
     unsigned int              numInputs) {
  for(unsigned int i = 0; i < numInputs; i++) {
    #pragma HLS PIPELINE
    output[i] = outStream.read();
  }
}
```

- exec

```
void exec(hls::stream<ap_int<512> > &inStream,
    hls::stream<ap_int<512> > &outStream,
    unsigned int              numInputs,
    unsigned int              processDelay) {
  for(int num = 0; num < numInputs; num++) {
    ap_int<512> in = inStream.read();
    for(int i = 0; i < processDelay; i++) {
      in += 1;
    }
    outStream.write(in);
  }
}
```

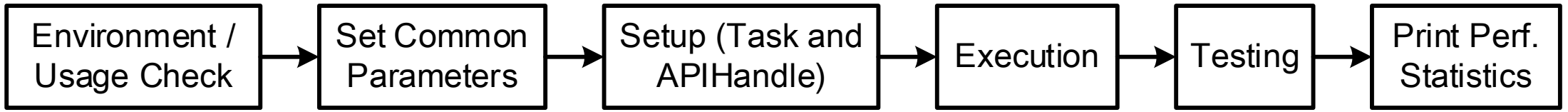
- pass_dataflow

```
void pass_dataflow(const ap_int<512> *input,
       ap_int<512>       *output,
       unsigned int      numInputs,
       unsigned int      processDelay) {
  #pragma HLS DATAFLOW
  assert(numInputs >= 1);
  assert(processDelay >=1);
  hls::stream<ap_int<512> > inStream;
  hls::stream<ap_int<512> > outStream;
  read(input,      inStream,  numInputs);
  exec(inStream,   outStream, numInputs, processDelay);
  write(outStream, output,    numInputs);

}
```

Host Program

MOST 科技部
Ministry of Science and Technology
A世代半導體計畫
Angstrom Semiconductor Initiative

- Flow chart



- ApiHandle
  - Setup device
  - Create kernel
  - Create queue
- Task
  - Create global buffer for input and output
  - Set kernel args
  - Move input data to global buffer
  - Start kernel
  - Read output data from the global buffer

Current Settings

MOST 科技部
Ministry of Science and Technology
A世代半導體計畫
Angstrom Semiconductor Initiative

- numBuffers
  - Total number of tasks
  - Set to 10 in the first and 100 in the second task
- oooQueue
  - Set the queue as the out-of-order queue
  - Set to true except the first experiment of Lab1
- processDelay
  - The number required to be added to the output
  - Set to 1
- bufferSize
  - The number of input data per task
  - Set as 16384 in the first two labs and sweep in the third lab

Lab1: Use the In-order Queue

MOST 科技部 Ministry of Science and Technology
A世代半導體計畫 Angstrom Semiconductor Initiative

- Setup params

```
// -- Common Parameters ----------------------------

unsigned int numBuffers                 = 10;
bool         oooQueue                   = false;
unsigned int processDelay               = 1;
unsigned int bufferSize                 = 8 << 11;
```

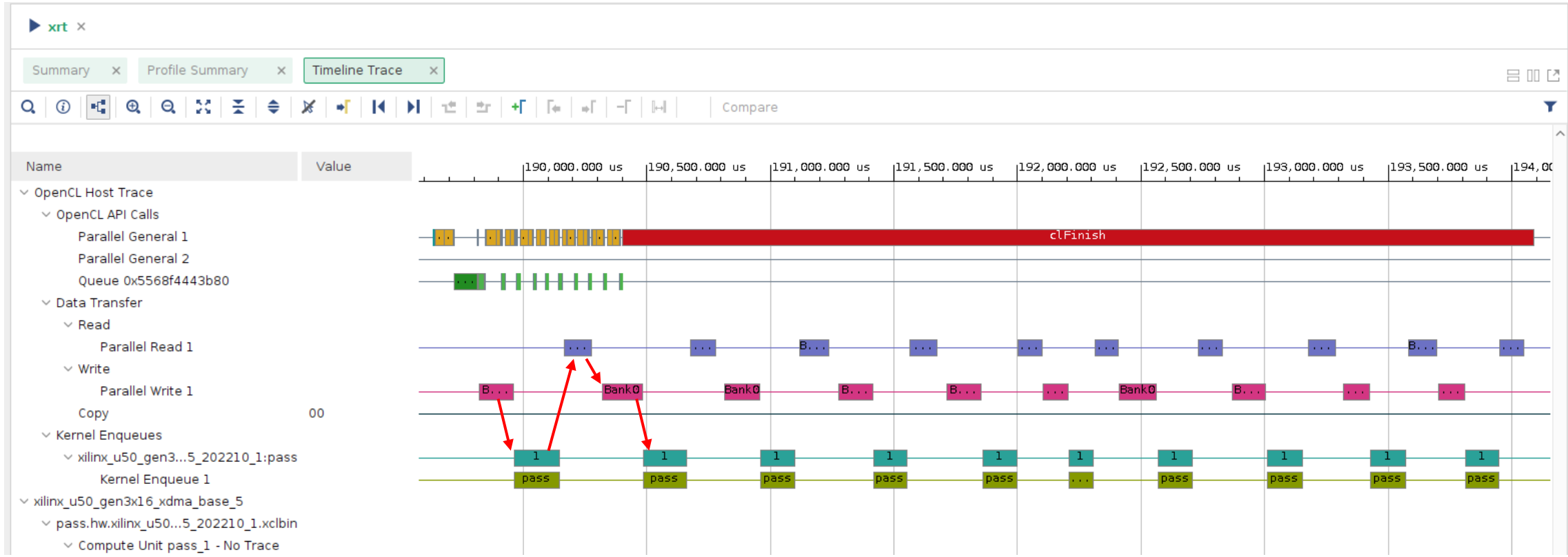
- Execution code

```
// -- Execution ------------------------------------

for(unsigned int i=0; i < numBuffers; i++) {
  tasks[i].run(api);
}
clFinish(api.getQueue());
```

Lab1: Use the In-order Queue

MOST 科技部
Ministry of Science and Technology
A世代半導體計畫
Angstrom Semiconductor Initiative

- Timeline



- Total execution time: 0.043s

Lab1: Use the Out-of-order Queue

MOST 科技部
Ministry of Science and Technology
A世代半導體計畫
Angstrom Semiconductor Initiative

- Setup params

```
// -- Common Parameters ----------------------------

unsigned int numBuffers                 = 10;
bool         oooQueue                   = true;
unsigned int processDelay               = 1;
unsigned int bufferSize                 = 8 << 11;
```

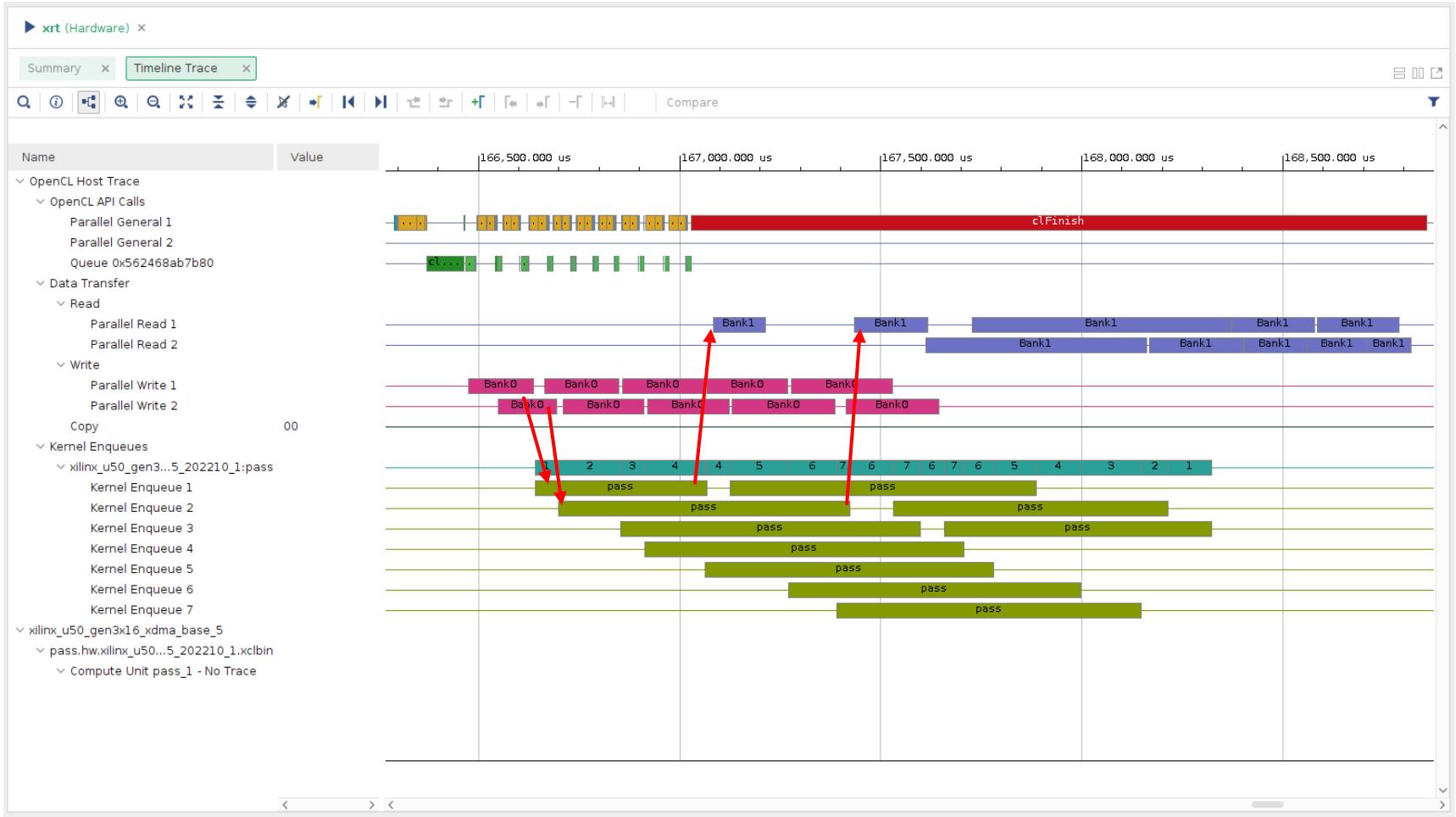
- Execution code

```
// -- Execution ------------------------------------

for(unsigned int i=0; i < numBuffers; i++) {
  tasks[i].run(api);
}
clFinish(api.getQueue());
```

Lab1: Use the In-order Queue

MOST 科技部
Ministry of Science and Technology
A世代半導體計畫
Angstrom Semiconductor Initiative

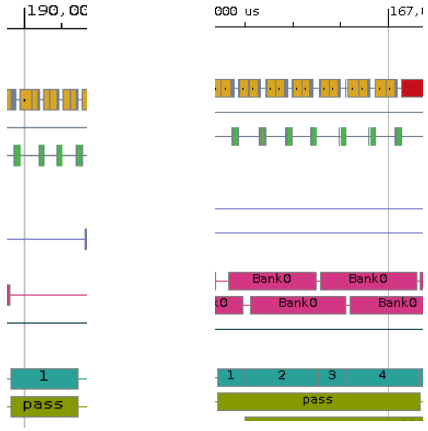
- Timeline



- Total execution time: 0.043s → **0.024s**

Observation

- The execution of one pass is longer in oooqueue
  - Longer latency per task



- Problem in the current design
  - No synchronization until the end
  - If the numBuffers is too large → FPGA will consume too much host memories, which can only be released at the end of the program.

Lab2: Kernel and Host Code Synchronization

MOST 科技部 Ministry of Science and Technology
A世代半導體計畫 Angstrom Semiconductor Initiative

- OpenCL framework has two methods for synchronization
  - clFinish and clWaitForEvents
- Modify the execution section

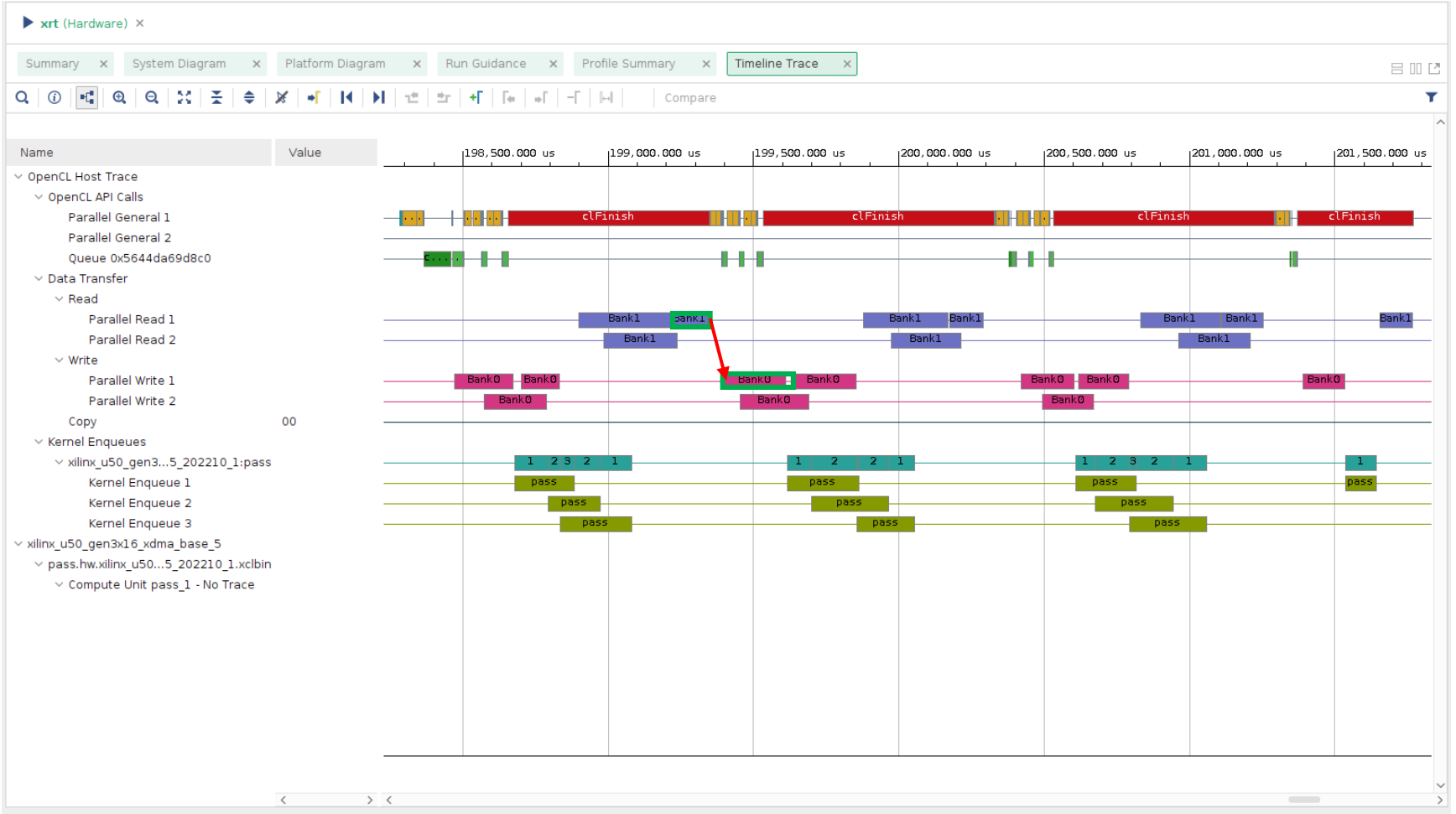
```
// -- Execution -----------------------------

for(unsigned int i=0; i < numBuffers; i++) {
  tasks[i].run(api);
}
clFinish(api.getQueue());
```

```
int count = 0;
for(unsigned int i=0; i < numBuffers; i++) {
  count++;
  tasks[i].run(api);
  if(count == 3) {
    count = 0;
    clFinish(api.getQueue());
  }
}
clFinish(api.getQueue());
```

Lab2: Kernel and Host Code Synchronization

MOST 科技部 Ministry of Science and Technology
A世代半導體計畫 Angstrom Semiconductor Initiative

- Timeline



- Total execution time: 0.024s → **0.033s**

Observation

MOST 科技部
Ministry of Science and Technology
A世代半導體計畫
Angstrom Semiconductor Initiative

- It is like a trade-off between in-order-queue and out-of-order queue
  - Out-of-order inside a batch, in-order between batches

- The clFinish is the synchronization point of the host program
  - Called four times in Lab2

Lab2: Kernel and Host Code Synchronization

MOST 科技部 Ministry of Science and Technology
A世代半導體計畫 Angstrom Semiconductor Initiative

- We can let the current task depends on the finish of the previous task
- Execution code

```
for(unsigned int i=0; i < numBuffers; i++) {
  if(i < 3) {
    tasks[i].run(api);
  } else {
    tasks[i].run(api, tasks[i-3].getDoneEv());
  }
}
clFinish(api.getQueue());
```

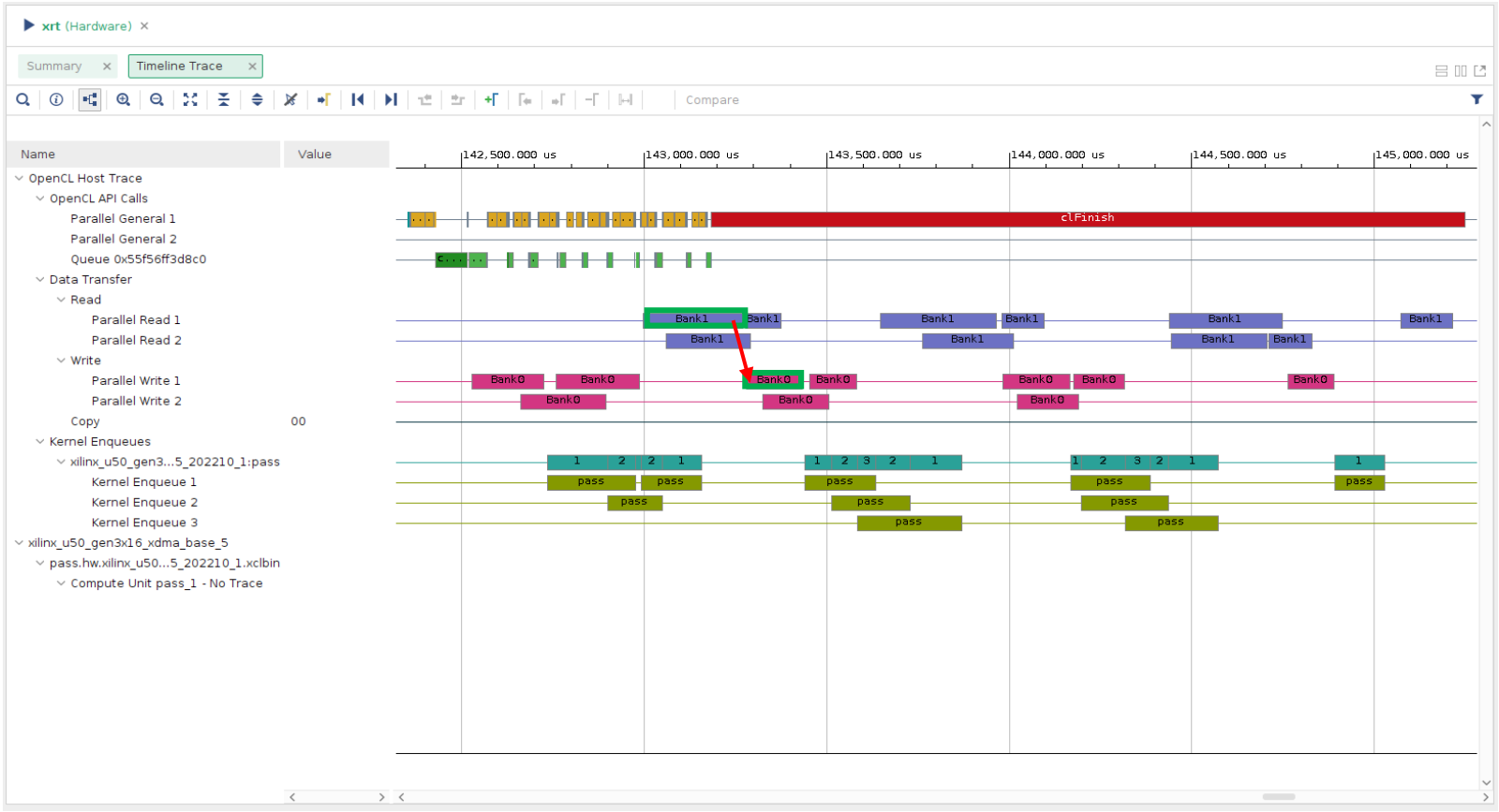
```
if(prevEvent != nullptr) {
  clEnqueueMigrateMemObjects(api.getQueue(), 1, &m_inBuffer[0],
    0, 1, prevEvent, &m_inEv);
} else {
  clEnqueueMigrateMemObjects(api.getQueue(), 1, &m_inBuffer[0],
    0, 0, nullptr, &m_inEv);
}
```

```
cl_event* getDoneEv()  { return &m_doneEv;  }
```

Lab2: Kernel and Host Code Synchronization

MOST 科技部 Ministry of Science and Technology
A世代半導體計畫 Angstrom Semiconductor Initiative

- Timeline

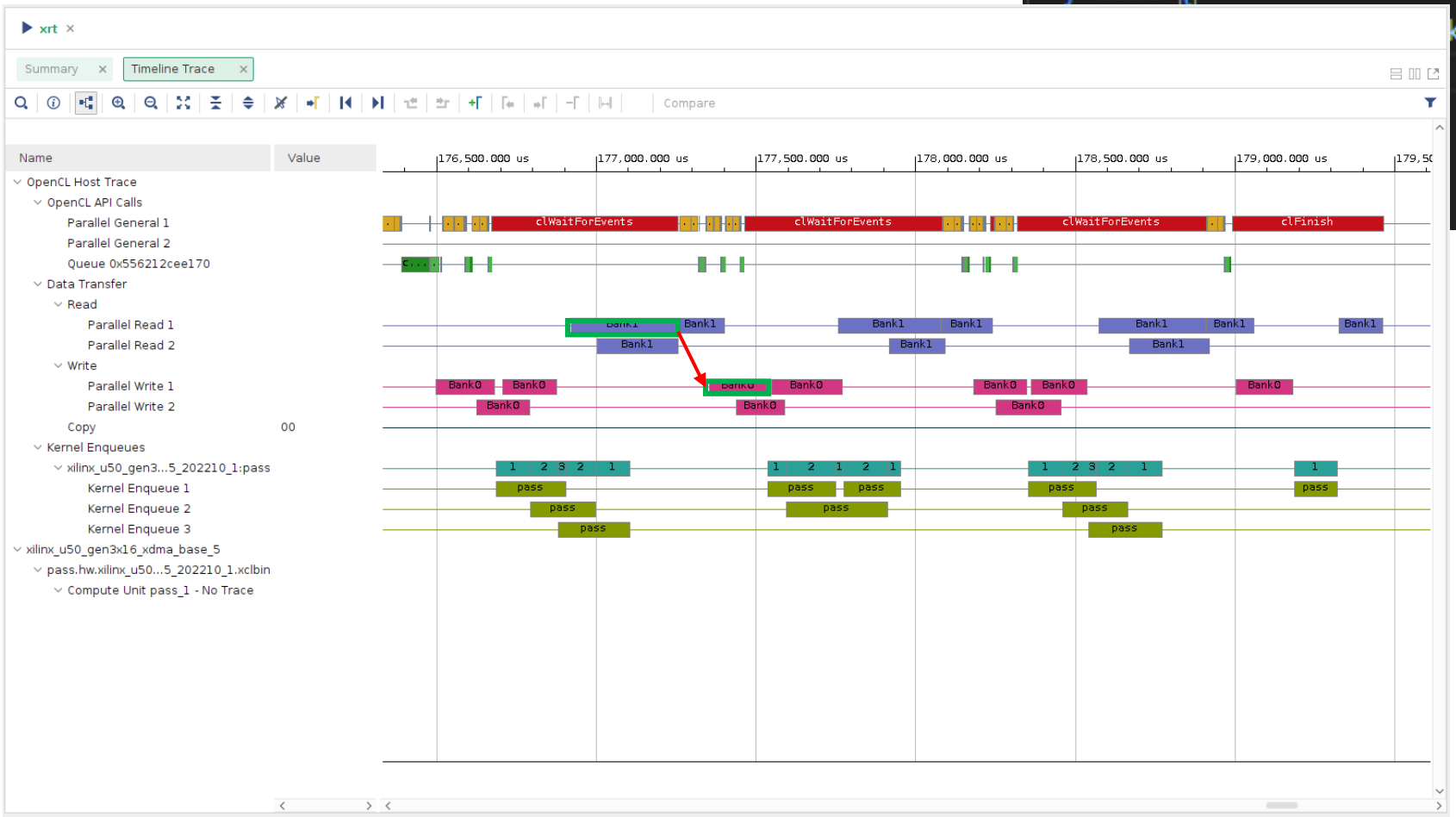


- Total execution time: 0.033s → **0.027s**
  - More overlap reduce execution time

Lab2: Kernel and Host Code Synchronization

- Using clWaitForEvent
- Timeline

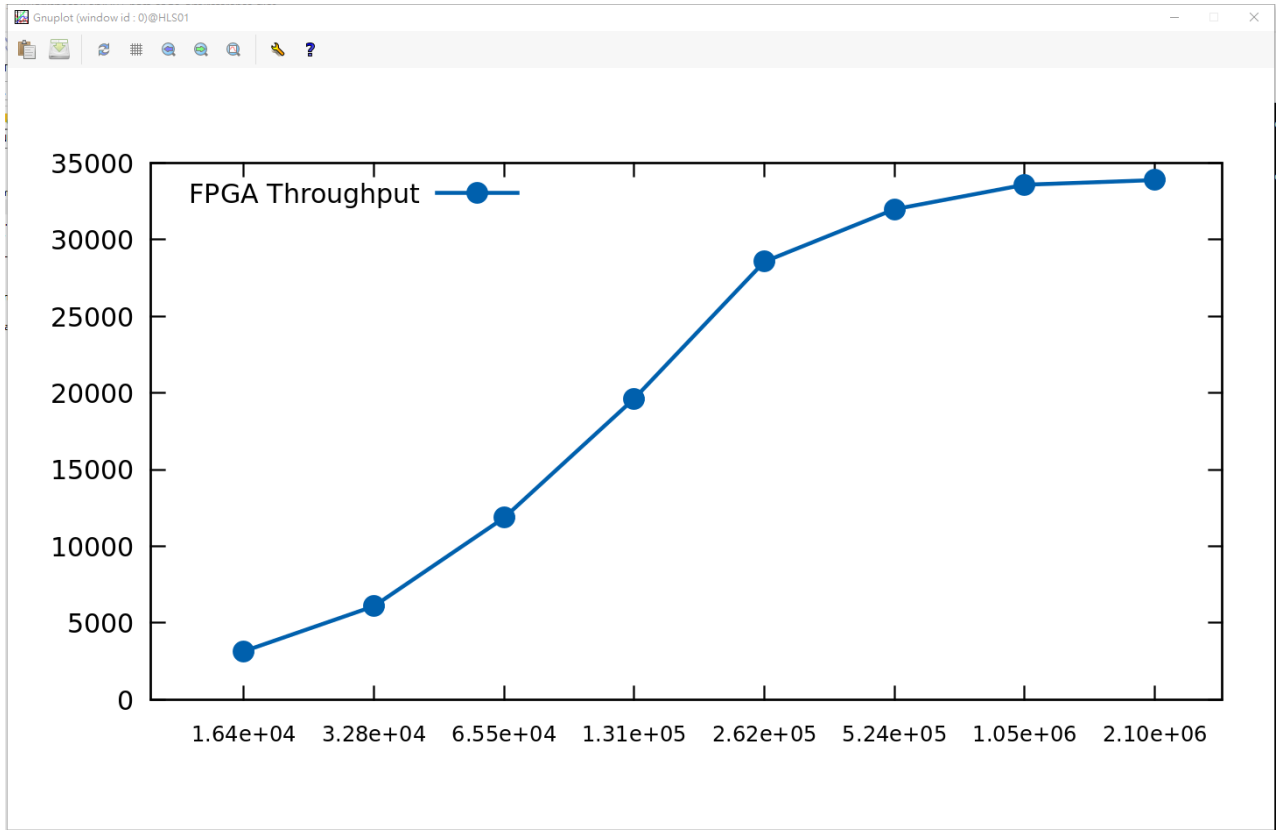
```
for(unsigned int i=0; i < numBuffers; i++) {
  if(i < 3) {
    tasks[i].run(api);
  } else {
                                    ks[i-3].getDoneEv());
```



Lab3: OpenCL API Buffer Size

MOST 科技部 Ministry of Science and Technology
A世代半導體計畫 Angstrom Semiconductor Initiative

- Set numBuffer to 100 to get a more accurate average throughput estimate per transfer
- We then sweep the buffer size from 2^{14} to 2^{21} to observe the change of the throughput
- Saturate when buffer size is close to $\mathbf{2^{20}}$
- The saturated throughput is approximate **34000Mb/s**



Summary

MOST 科技部
Ministry of Science and Technology
A世代半導體計畫
Angstrom Semiconductor Initiative

- Host code indeed influences the execution time and resource usage
  - Good host code make your kernel highly utilized without occupy too many host resources

- The final sweep of the buffer size indicates the proper buffer size to use in our system to make the interface fully utilized

Q&A

MOST 科技部
Ministry of Science and Technology
A世代半導體計畫
Angstrom Semiconductor Initiative

- How to release the buffer occupied by the host code?
  - Insert clFinish
  - Insert clWaitForEvent
  - Let event depends on the finish of the previous event