XDF XILINX DEVELOPER FORUM

Develop Computational Storage Applications using SDAccel

Presented By

Sumit Roy
Senior R&D Director, SDAccel, SDSoC, HLS
October 2nd, 2018

XILINX

Agenda

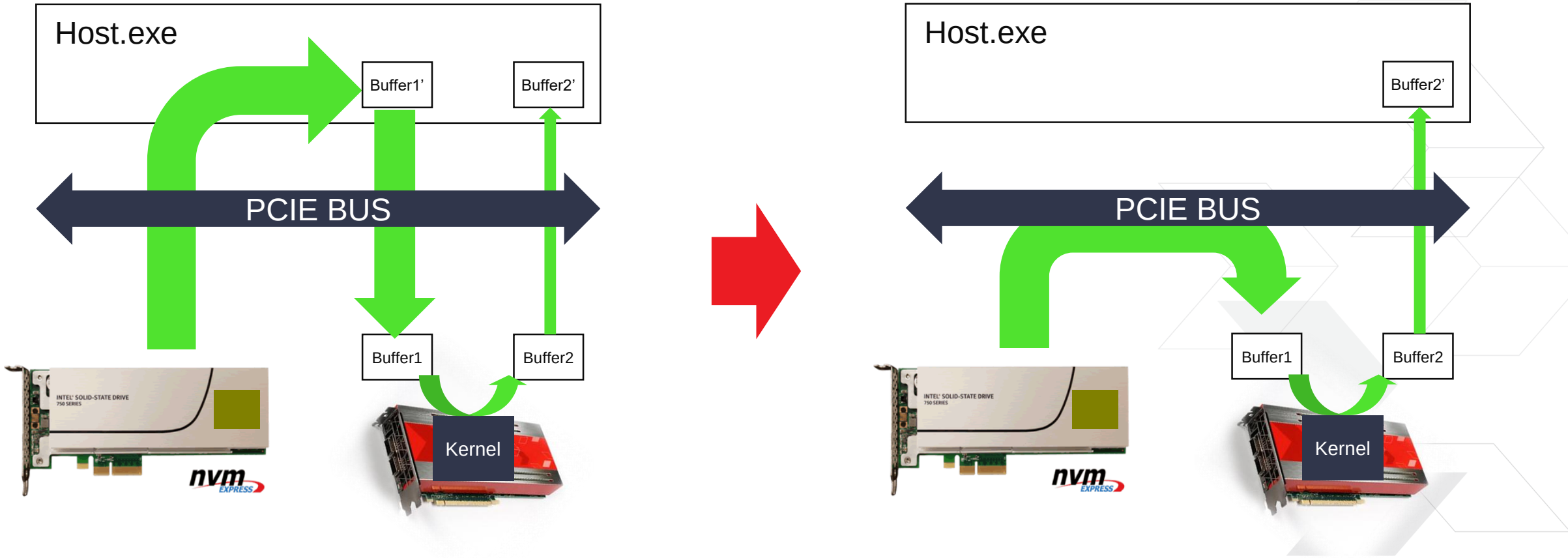
- **Introduction**
  - SDAccel Benefits
  - Computational Storage Programmer's View
- **What to accelerate**
  - 3 rules of the game
- **How to accelerate**
- **Summary**

Why Use SDAccel For Computational Storage Acceleration?

- Platform and runtime library optimized for performance
- Choice of HLS or RTL for acceleration kernel
- Dedicated visualization, profiling and debug Tools
- Optimized libraries
- Portability: Easy porting from existing SDAccel applications

Performance – Productivity – Portability

Computational Storage – Overview

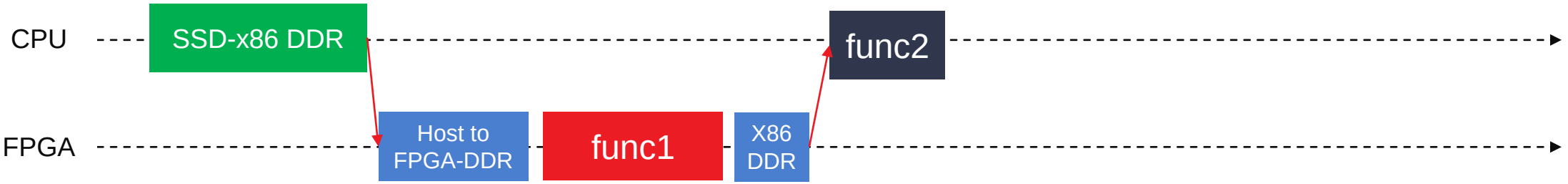


Key benefit : Avoiding extra copies into Host DDR

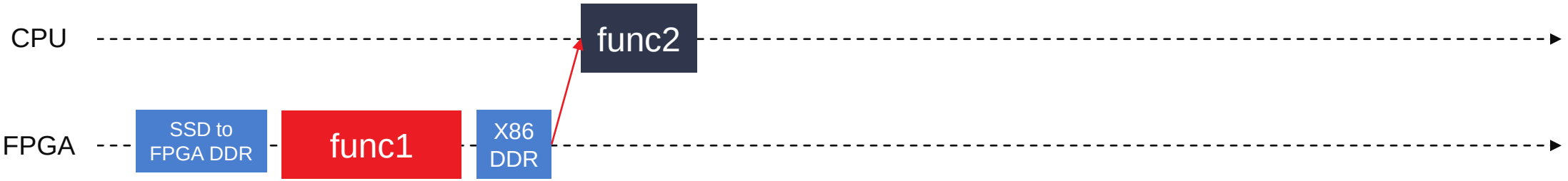
Computational Storage Benefit

Without acceleration

With FPGA compute acceleration (offload compute)



With FPGA computational storage acceleration (offload compute & I/O)

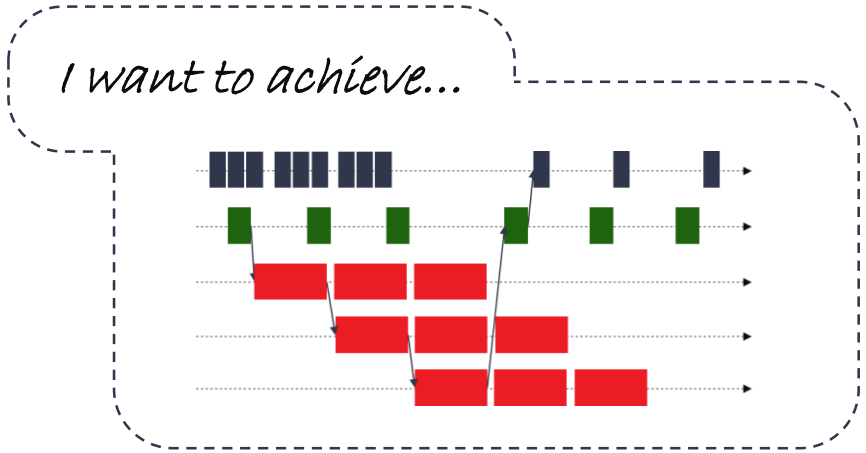


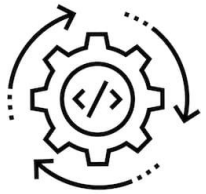
Computational Storage Solution avoids copying to x86 DDR

Agenda

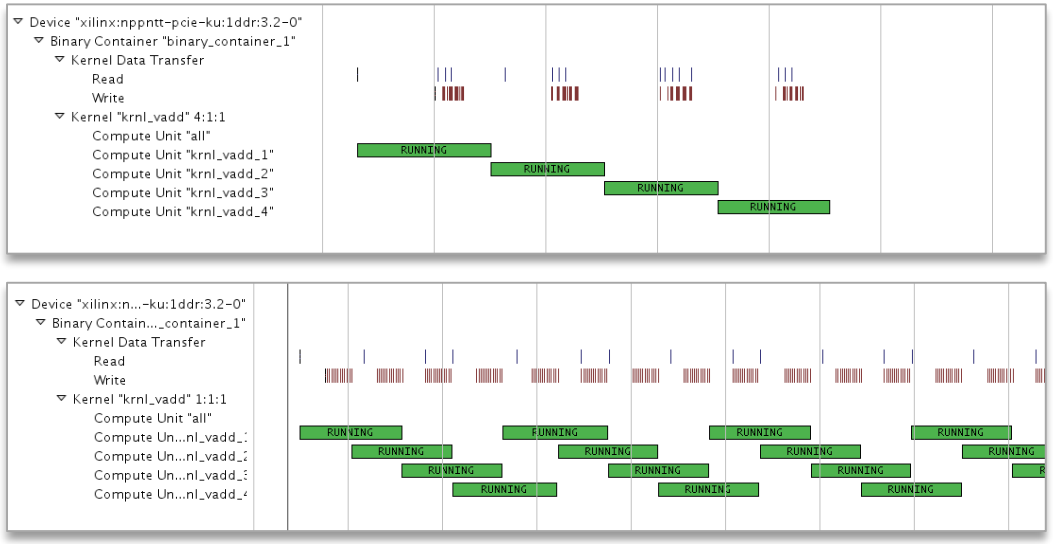
- **Introduction**
  - SDAccel Benefits
  - Computational Storage Programmer's View
- **What to accelerate**
  - 3 rules of the game
- **How to accelerate**
- **Summary**

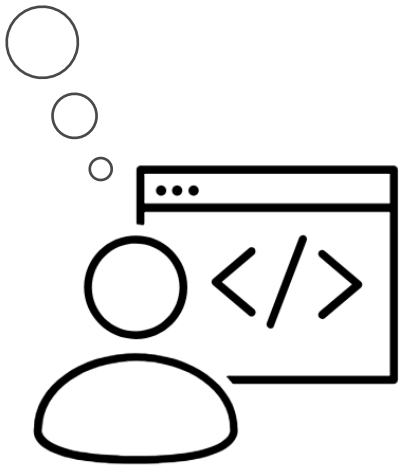
Developing Applications Using SDAccel





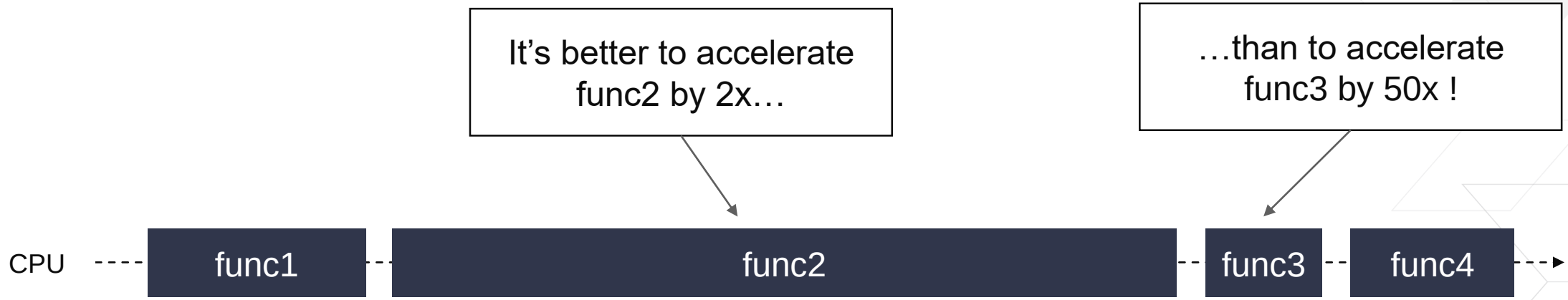
SDAccel Application Timeline View





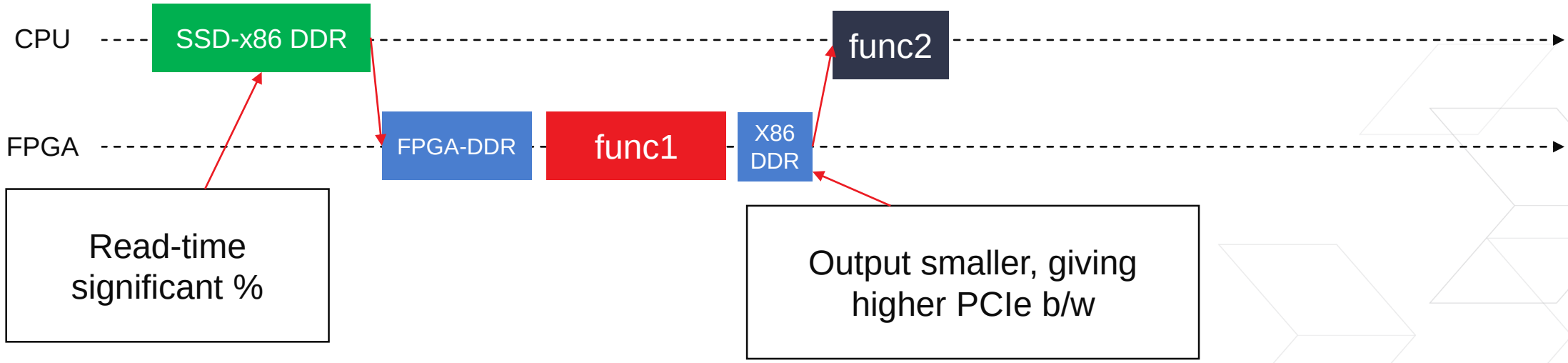
- **Start with the end in mind → conceptualize system architecture**
- **Use visualization and guidance tools → confirm and converge**

Rule #1 – Remember Amdahl's Law



- **Consider overall performance, not just individual functions**
- **When working "top-down", identify performance bottlenecks in the application**
  - Use profiling tools, analyze the "roof line" of a Flame Graph
- **Target accelerators that will impact end-to-end performance of the application**

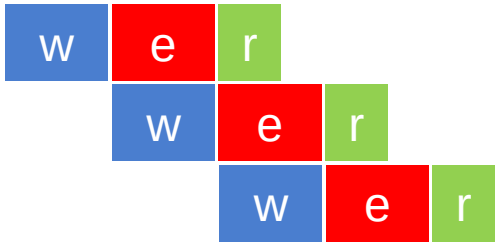
Rule #2 – Target Tasks with high % of I/O time



- **Look for functions with where {read/write SSD time} is significant % of total time**
  - Good: file-compression– reading file from SSD to host DDR significant compared total time
  - Bad: Video-compression – HEVC, VP9 encoding significantly larger than streaming video

- **Prefer functions that are filters, encoders**
  - Increases overall bandwidth of the PCIe throughput to x86

Rule #3 – Balance Compute and Communication

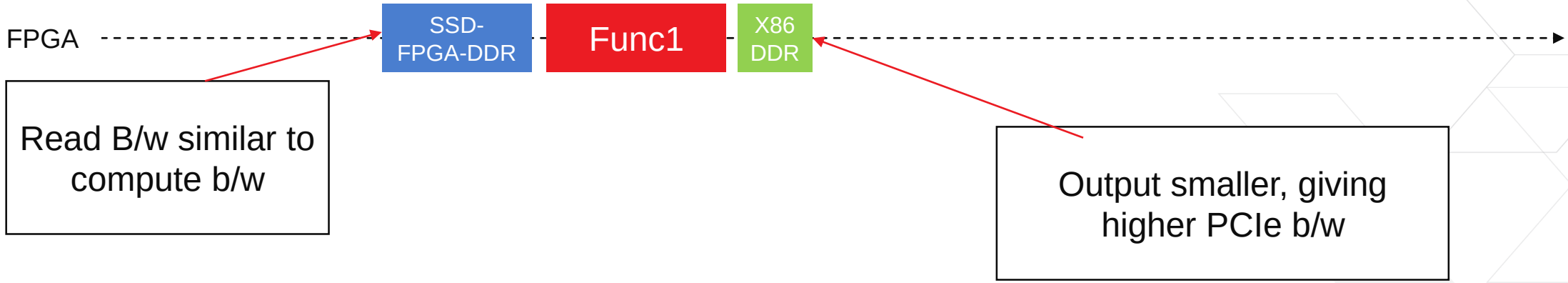


Software Pipelining

w: write inputs from SSD to FPGA

e: execute kernel device function

r: read outputs back to x86 host



- **Target applications with streaming data**
  - Task-level, Data-level streaming like Apache Spark
- **Balance throughput of data-movement and computation**
  - Data-movement throughput should be high and similar to compute

FPGA-Based Computational Storage Acceleration

- **When to USE**
  - PCIe b/w is bottleneck
  - Time to read from SSD significant % of total

- **When May Not be beneficial**
  - Small problem size
  - Additional preprocessing done on host

- **When NOT beneficial**
  - Little to no parallelism
    - Algorithm is highly sequential over multiple data
  - Compute heavy with very small data-transfer overhead

Agenda

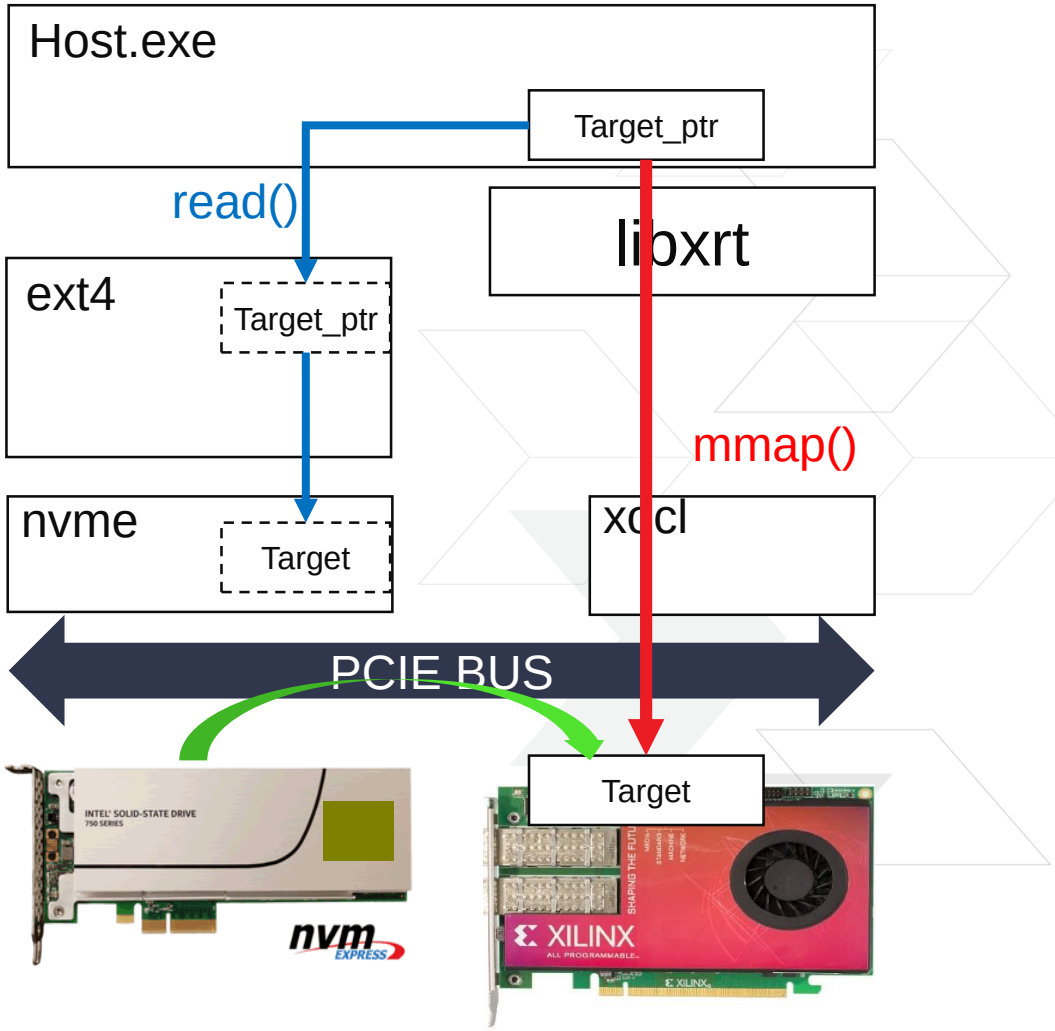
- **Introduction**
  - SDAccel Benefits
  - Computational Storage Programmer's View
- **What to accelerate**
  - 3 rules of the game
- **How to accelerate**
- **Summary**

Porting Existing Compute Acceleration Apps

- **Step 1: Move data directly from SSD to FPGA**
- **Step 2: Move pre/postprocess from Host to FPGA**
- **Step 3: Target multi-board for max. performance**
- **Step 4: Multi-process if FPGA throughput is higher than single application throughput**

Step 1: Move data directly from SSD to FPGA

- **Source file needs to be opened with O_DIRECT to bypass page cache**
  - Source_file_fd = open(path, O_RDWR | O_DIRECT);
  - Read(source_file_fd, target_ptr, size);
  - Success!
- **Restrictions for O_DIRECT read() from file**
  - Read() can only be done in block size unit with block size aligned buffer pointer
- **Restrictions for O_DIRECT write() to file**
  - Write() can only be done in block size unit with block size aligned buffer pointer
  - Call fallocate() to allocate blocks in file system before write, or page cache may kick in implicitly
- **Kernel may need to be modified to iterate on data in block-size chunks**



Step 1: Move data directly from SSD to FPGA

No Preprocess

> Read to x86 buffer from file

```
/* Allocate BO */
cl_mem_ext_ptr_t clmem_ext = { 0 };
clmem_ext.flags = get_bank_flag(bank);
target = clCreateBuffer(context,
                        CL_MEM_EXT_PTR_XILINX | CL_MEM_READ_WRITE,
                        size,
                       &clmem_ext, &err);

/* Read source data from file */
target_ptr = clEnqueueMapBuffer(cmdq, target,
                                CL_TRUE, CL_MAP_WRITE | CL_MAP_READ,
                                0, size, 0, NULL, NULL, &err);
source_file_fd = open(path, O_RDONLY);
read(source_file_fd, target_ptr, size);
clEnqueueMigrateMemObject(…, target,…);

/* Kick off kernel */
setKernelArgAndExecKernel();
```

> Read to DDR from file

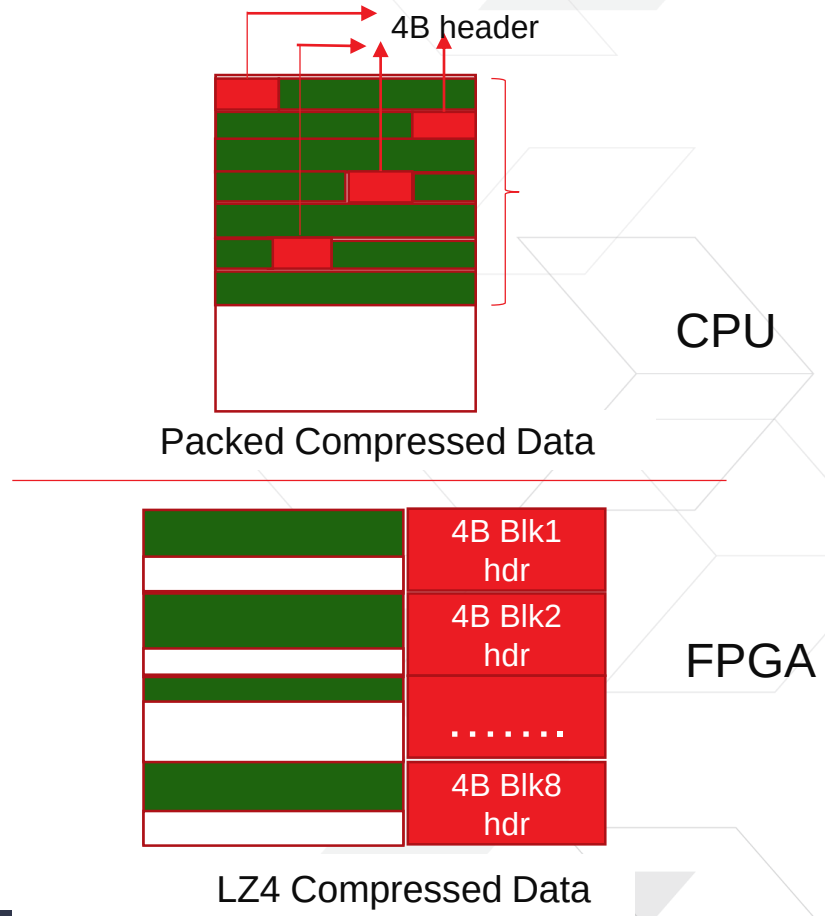
```
/* Allocate BO */
cl_mem_ext_ptr_t clmem_ext = { 0 };
clmem_ext.flags = get_bank_flag(bank) | XCL_MEM_EXT_P2P_BUFFER;
target = clCreateBuffer(context,
                        CL_MEM_EXT_PTR_XILINX | CL_MEM_READ_WRITE,
                        size  /* Multiple of blk size of FS */,
                       &clmem_ext, &err);

/* Read source data from file */
target_ptr = clEnqueueMapBuffer(cmdq, target,
                                CL_TRUE, CL_MAP_WRITE | CL_MAP_READ,
                                0, size, 0, NULL, NULL, &err);
source_file_fd = open(path, O_RDONLY | O_DIRECT);
read(source_file_fd, target_ptr, size);
clEnqueueMigrateMemObject(…, target,…);

/* Kick off kernel */
setKernelArgAndExecKernel();
```

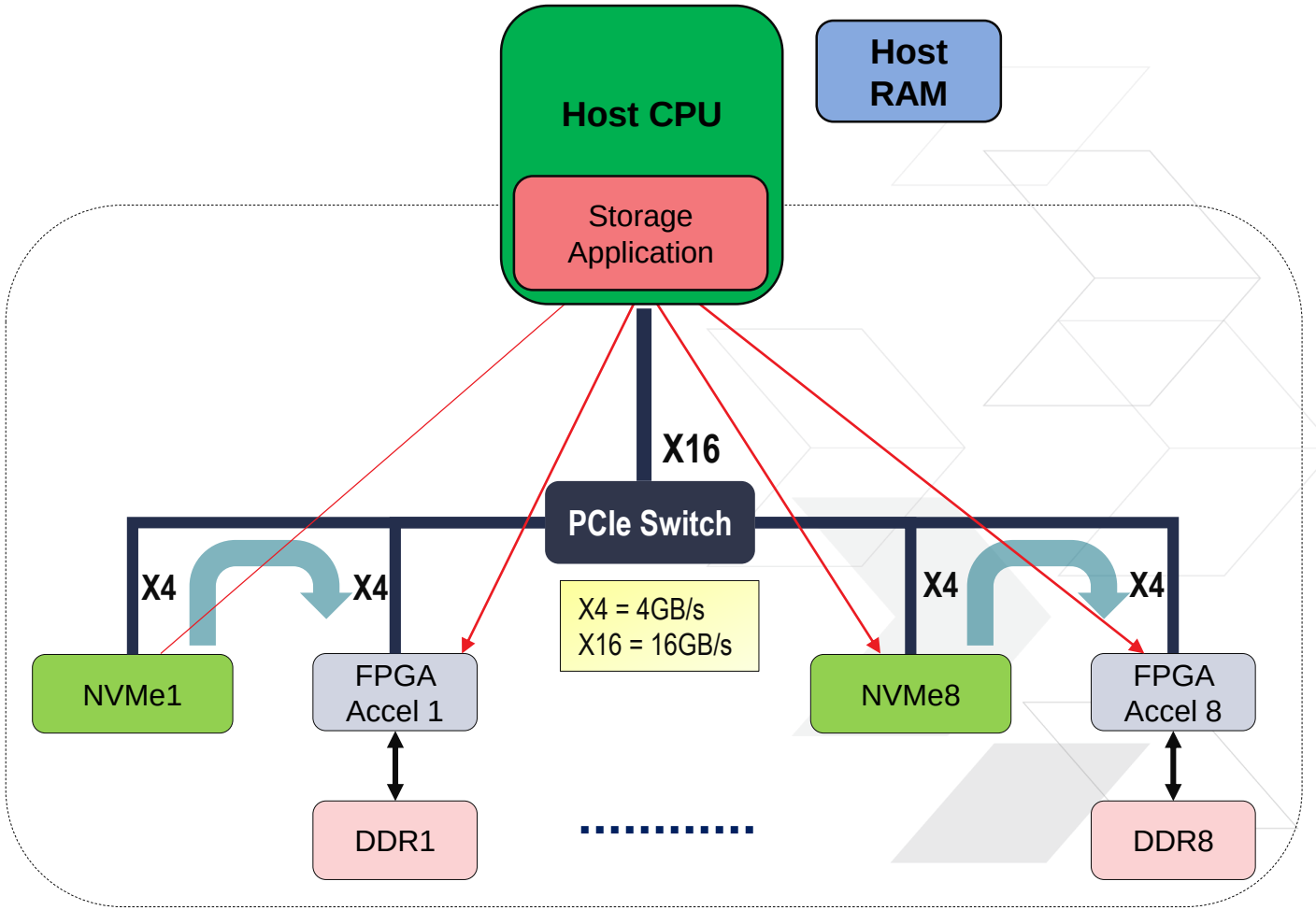
Step 2: Move Pre/Postprocess to FPGA

- **Applications may have pre/postprocessing functions of data read from SSD**
  - Accelerate function to FPGA to avoid copy into host
- **PostgreSQL: preprocess**
  - Table is read in 2MB block size in a loop
  - Preprocess relevant rows and sends data to FPGA
- **LZ4 : postprocess**
  - LZ4 compression kernel creates hole in the data layout
  - Postprocess compacts the data



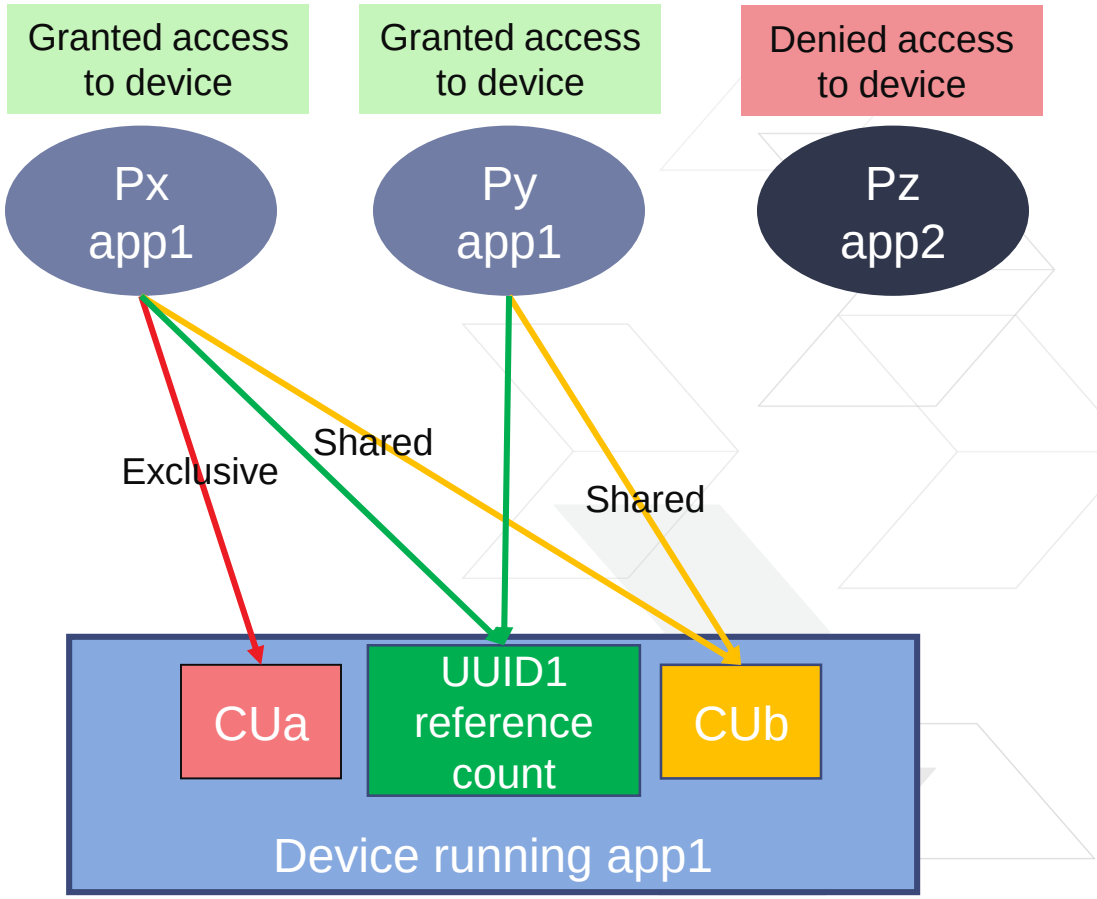
Step 3: Improve performance using multi-board

- **Single FPGA acceleration does not saturate PCIe b/w**
  - Application output uses 2GB/s for 1 card
  - PCIe x16 has 16GB b/w
- **Increase performance by multiple folds using multi-board**
  - Attach 8 cards to get 8X performance improvement
- **Steps for replicating across boards**
  - Open all FPGA cards and NVMe devices
  - Pair NVMe device with corresponding FPGA
  - Ensure that data used by FPGA is resident on paired NVMe device



Step 4: Multiple Applications

- **Supports multiple processes using 1 or more device concurrently**
- **LZ4 over PCIe gen3x16 supports 16 GB/s compression rate**
  - Can feed 8 instances of application requiring 2GB/s compression data
- **Application**
  - Create 1 or more CU per FPGA device
  - Runtime grants CU to a process in a first come first serve
  - Processes using same application is scheduled in a round-robin
  - SSD can be shared between all the processes
  - Application needs to ensure different processes do not modify the same file at same time



Agenda

- **Introduction**
  - SDAccel Benefits
  - Computational Storage Programmer's View
- **What to accelerate**
  - 3 rules of the game
- **How to accelerate**
- **Summary**

Summary

- Computational Storage Platform brings compute close to data, thereby improving performance, reducing power
- SDAccel provides productive way to develop new applications or port existing compute accelerated applications
- Considering system-level architecture is key to developing successfully accelerated application

Getting Started is Easy

- SDAccel Tutorials for U200 and VCU1525
- www.github.com/Xilinx/SDAccel-Tutorials

- On-Demand Developer Labs for AWS F1
- https://github.com/Xilinx/SDAccel-AWS-F1-Developer-Labs

- Free trial of the Nimbix FPGA Developer Program
- https://www.nimbix.net/fpga-developer-program/

Learn and practice how to accelerate applications with FPGAs

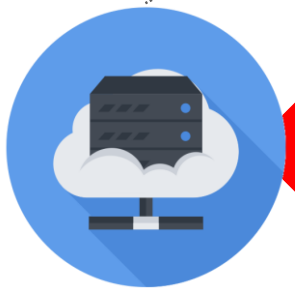
SDx value proposition: Improve EoU

SDx™ Environments

Develop applications

Deploy Applications

Leverage Applications



Virtex Ultrascale+ VU9P

Zynq Ultrascale+ MPSoC

Cloud (Compute)

On Premises (Compute)

Storage/Network

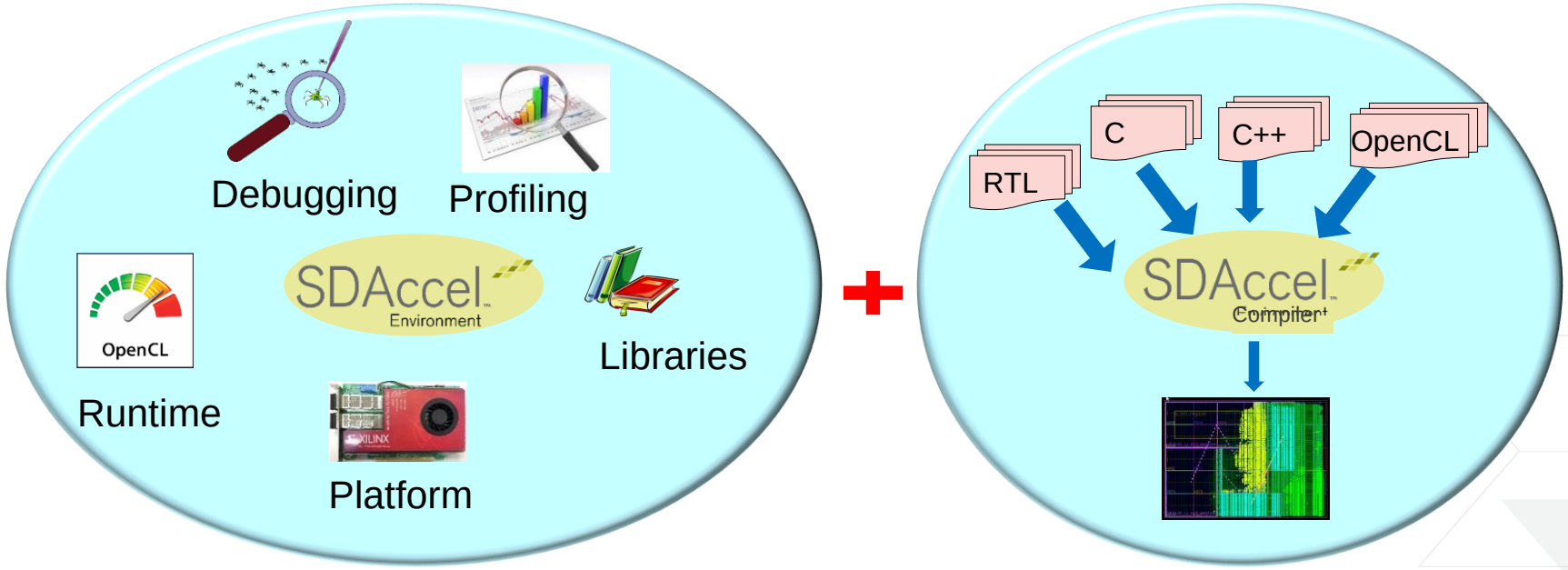
NIMBIX

Alibaba Cloud
aliyun.com

HUAWEI

XILINX

SDAccel: Faster path to FPGA Acceleration



Optimized Framework for Acceleration

Accelerator Kernel Compiler

SDAccel enables **performance**, **productivity** and **portability**

P2P Across FPGA and NVMe SSD – Through Filesystem

> Write Non-P2P buffer to file

```
/* Allocate BO */
cl_mem_ext_ptr_t clmem_ext = { 0 };
clmem_ext.flags = get_bank_flag(bank);
source = clCreateBuffer(context, CL_MEM_EXT_PTR_XILINX |
CL_MEM_READ_WRITE, size, &clmem_ext, &err);

/* Kick off kernel */
setKernelArgAndExecKernel(…);

/* Write result back to file */
clEnqueueMigrateMemObject(…, source,…);
source_ptr = clEnqueueMapBuffer(cmdq, target, CL_TRUE, CL_MAP_READ
| CL_MAP_WRITE, 0, size, 0, NULL, NULL, &err);
target_file_fd = open(path, O_CREAT | O_WRONLY);
write(target_file_fd, source_ptr, size);
```

> Write P2P buffer to file

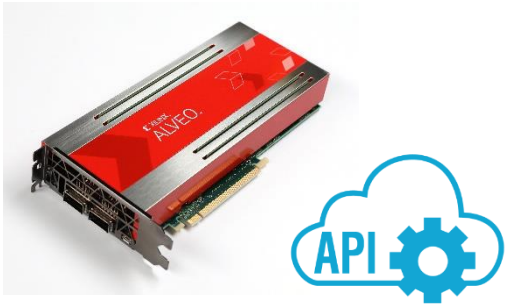
```
/* Allocate BO */
cl_mem_ext_ptr_t clmem_ext = { 0 };
clmem_ext.flags = get_bank_flag(bank) | XCL_MEM_EXT_P2P_BUFFER;
source = clCreateBuffer(context, CL_MEM_EXT_PTR_XILINX |
CL_MEM_READ_WRITE, size /* Multiple of blk size of FS */, &clmem_ext, &err);

/* Kick off kernel */
setKernelArgAndExecKernel(…);

/* Write result back to file */
clEnqueueMigrateMemObject(…, source,…);
source_ptr = clEnqueueMapBuffer(cmdq, target, CL_TRUE, CL_MAP_WRITE |
CL_MAP_READ, 0, size, 0, NULL, NULL, &err);
target_file_fd = open(path, O_CREAT | O_WRONLY | O_DIRECT);
fallocate(target_file_fd, ..., size);
write(target_file_fd, source_ptr, size);
ftruncate(target_file_fd, actual_size);
```

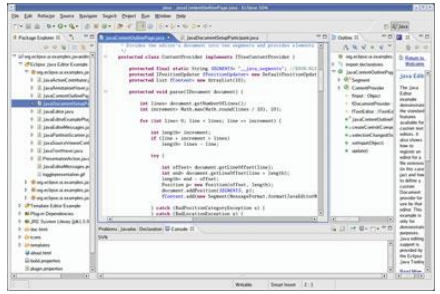
SDAccel: Faster Path to FPGA Acceleration

SDAccel™ Environment



High Performance Platform and Runtime Library

Advanced FPGA Compiler



Productive IDE & Optimized Libraries

User Onboarding