Karlsruhe Institute of Technology

UFO: A Real-time Data Acquisition and Processing Platform

Matthias Vogelgesang et al.

matthias.vogelgesang@kit.edu

4th Annual MT Meeting


THE DAWN OF MAN

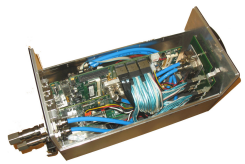

THE DAWN OF MAN
UFO

Synchrotron micro CT

Origins

- BMBF project to develop high-throughput µCT
- Covers beamline, detectors and software
- Ultimate goal: provide a feedback loop to assess quality and improve experimental conditions



Challenges

- Complex synchrotron environment
- A zoo of area detectors and other instruments
- High automation degree and compute demands

High-level solution

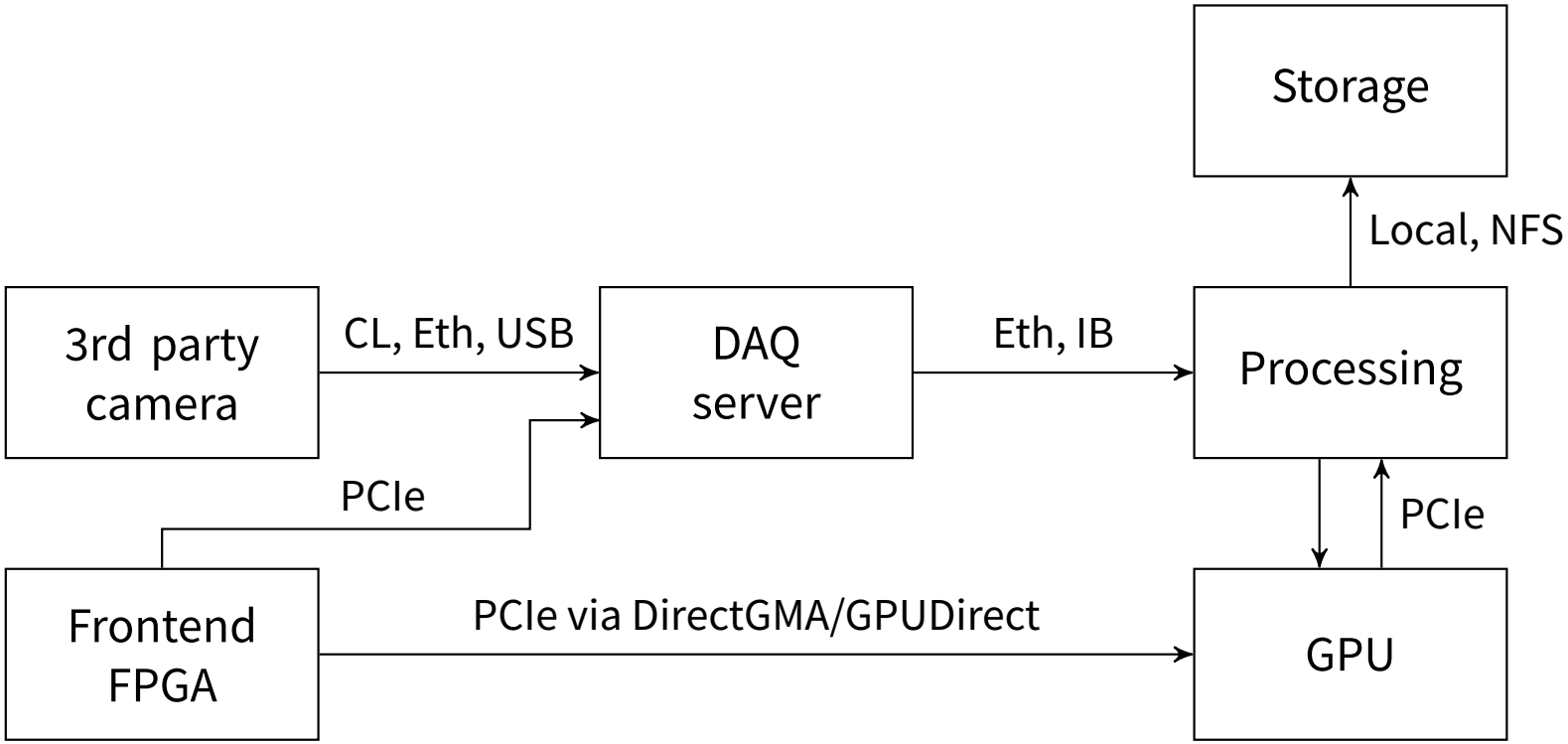
- Optimize acquisition and processing pipeline for throughput and latency
- Bridge the gap between hardware and software
- Bring data processing closer to the experiment
- Keep the solution flexible to adapt to different conditions

High-level solution

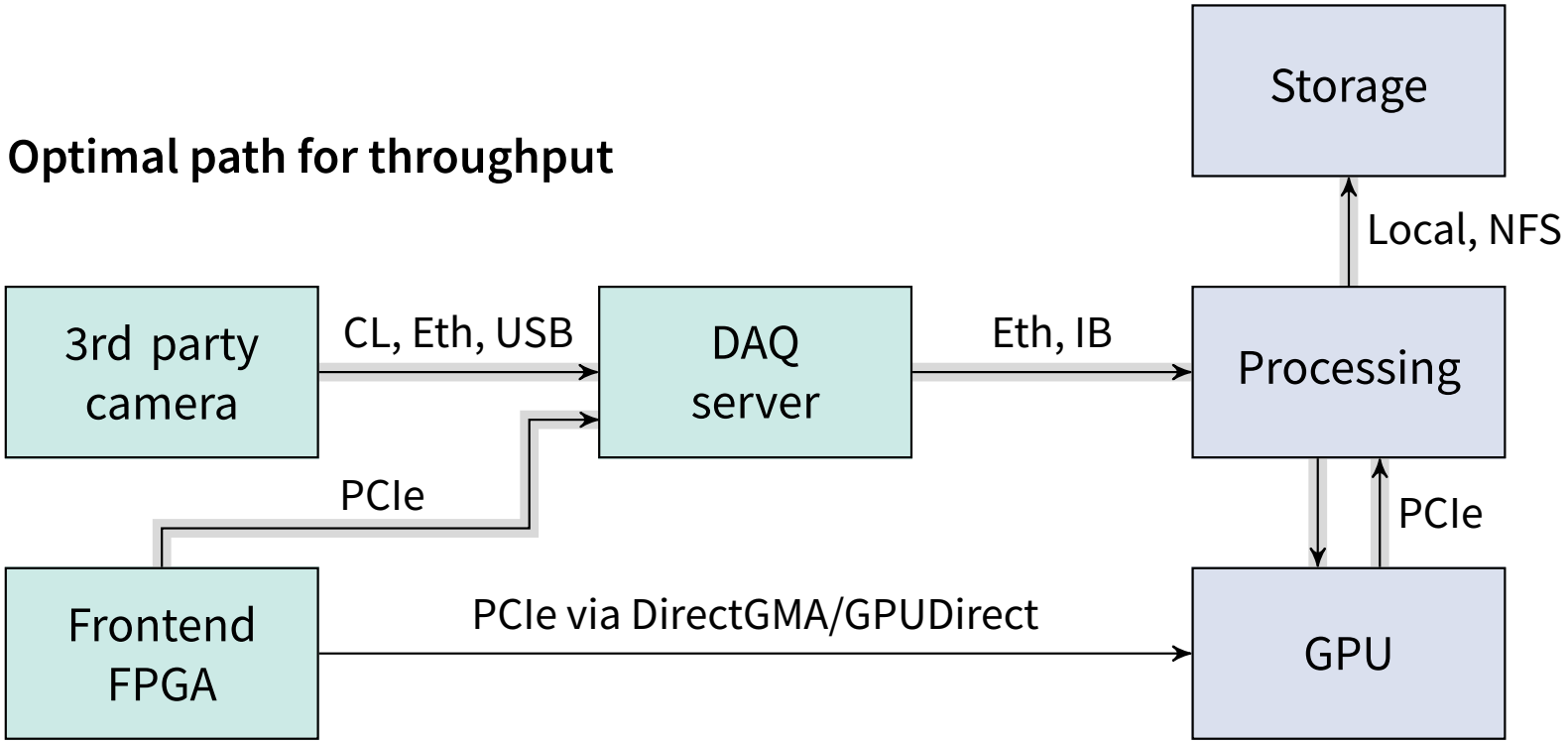
- Optimize acquisition and processing pipeline for throughput and latency
- Bridge the gap between hardware and software
- Bring data processing closer to the experiment
- Keep the solution flexible to adapt to different conditions

Stream data and reduce dead times where possible

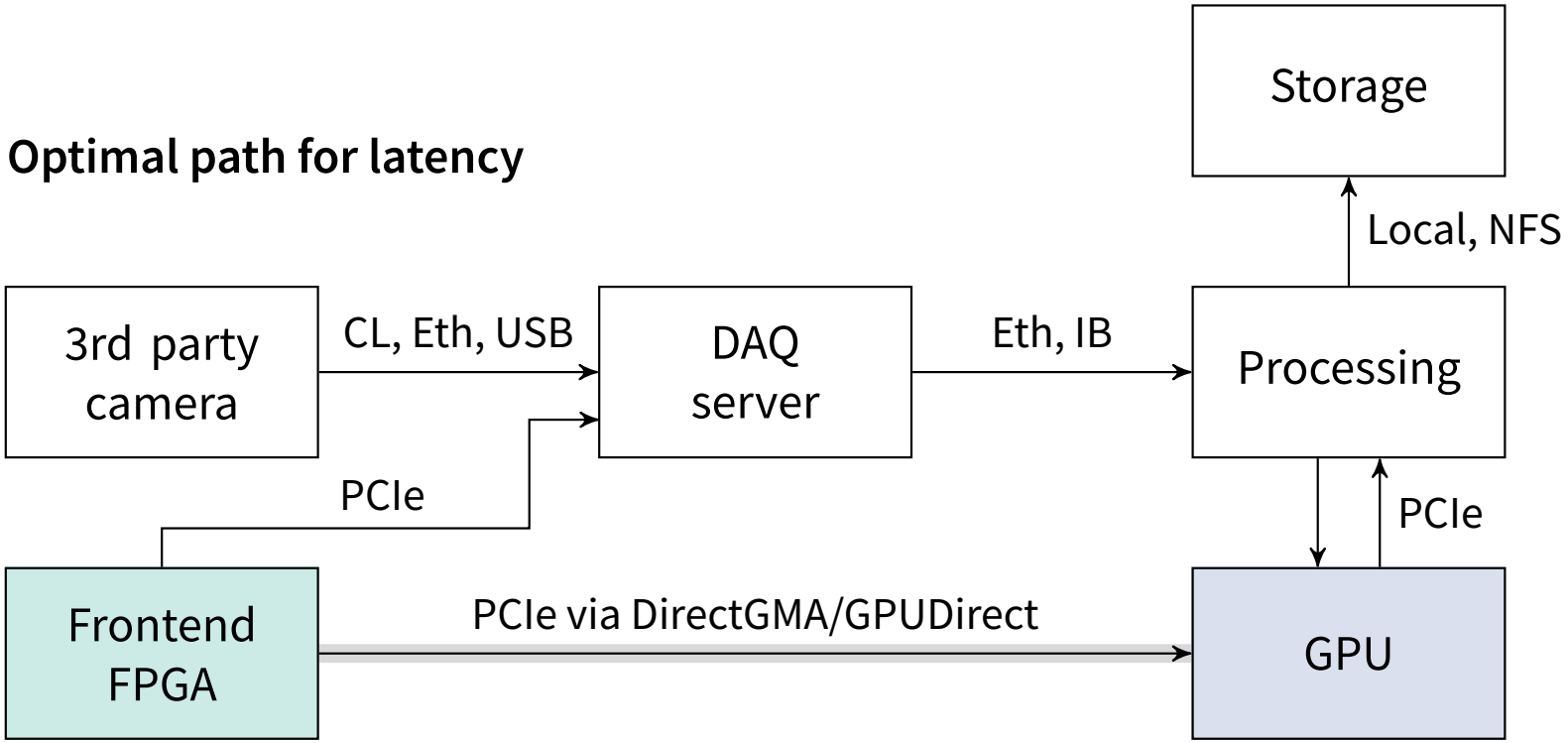
System design



System design



System design

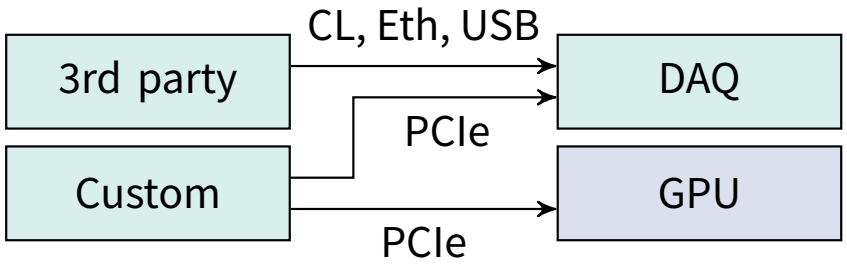


DATA ACQUISITION

Requirements

User

- Flexible and accessible
- Modular and extensible
- No policies, only mechanisms



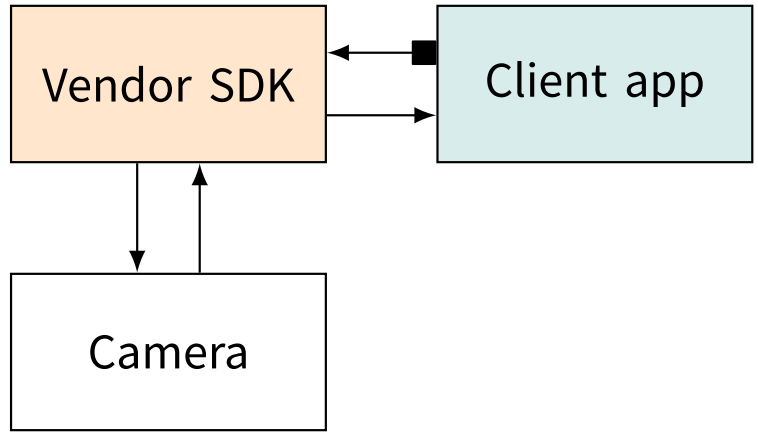
Detector

- Synchronous, asynchronous and buffered readout
- Different trigger and readout scenarios
- Camera-specific properties
- Zero-copy data transfers if possible
- Different means of data transport

General DAQ design

Typical approach

- Vendor provides a detector interface (“SDK”)
- Client application uses this interface



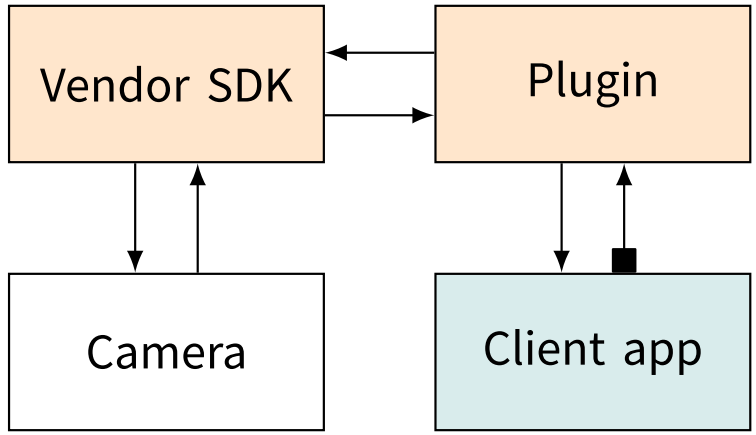
General DAQ design

Typical approach

- Vendor provides a detector interface (“SDK”)
- Client application uses this interface

Unified interface

- Specify a small, generic interface
- Provide indirection by vendor-specific implementation
- Encapsulation through plugin architecture

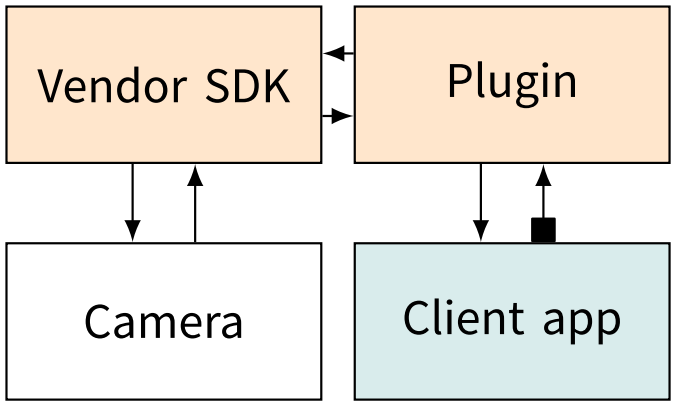


Remote data transfers

Motivation

- Separation of detector and processor
- Stability, heterogenic environment (Windows detector, Linux processing)

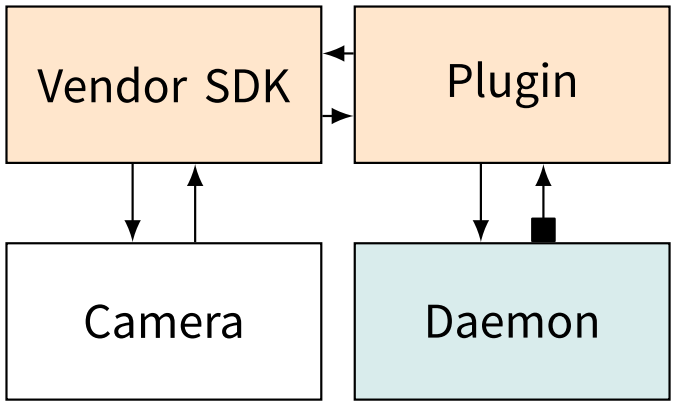
Approach



Remote data transfers

Motivation

- Separation of detector and processor
- Stability, heterogenic environment (Windows detector, Linux processing)



Approach

1. Server process controls camera

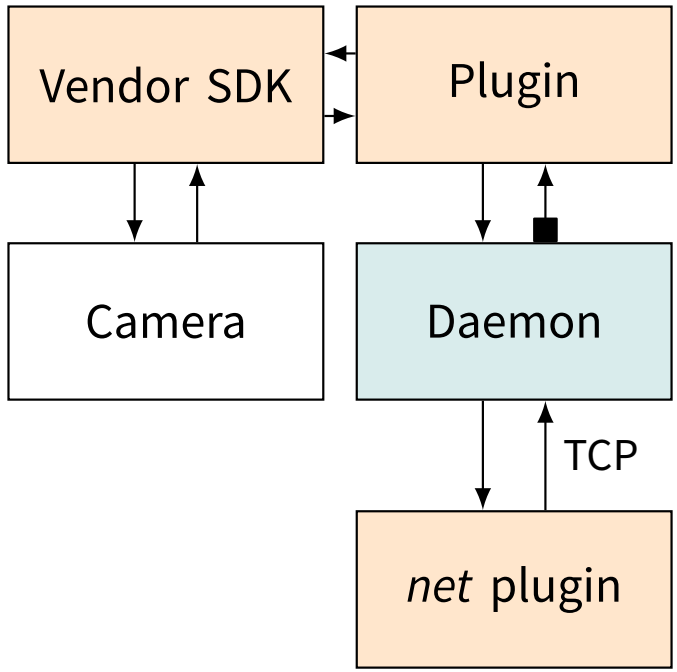
Remote data transfers

Motivation

- Separation of detector and processor
- Stability, heterogenic environment (Windows detector, Linux processing)

Approach

1. Server process controls camera
2. Proxy communicates with the server



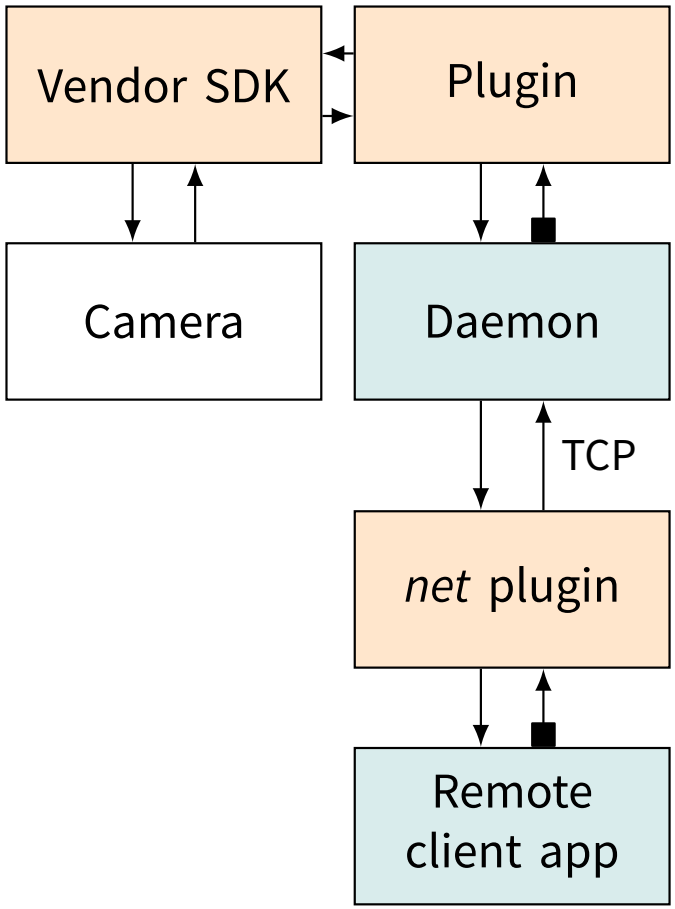
Remote data transfers

Motivation

- Separation of detector and processor
- Stability, heterogenic environment (Windows detector, Linux processing)

Approach

1. Server process controls camera
2. Proxy communicates with the server
3. Client application uses proxy to control actual camera



Remote data transfers

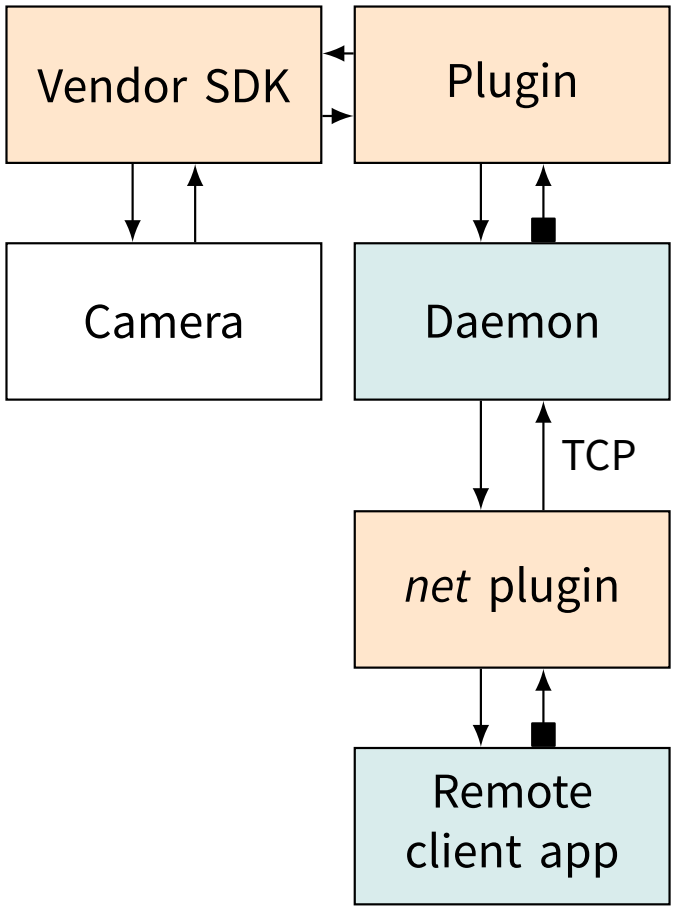
Motivation

- Separation of detector and processor
- Stability, heterogenic environment (Windows detector, Linux processing)

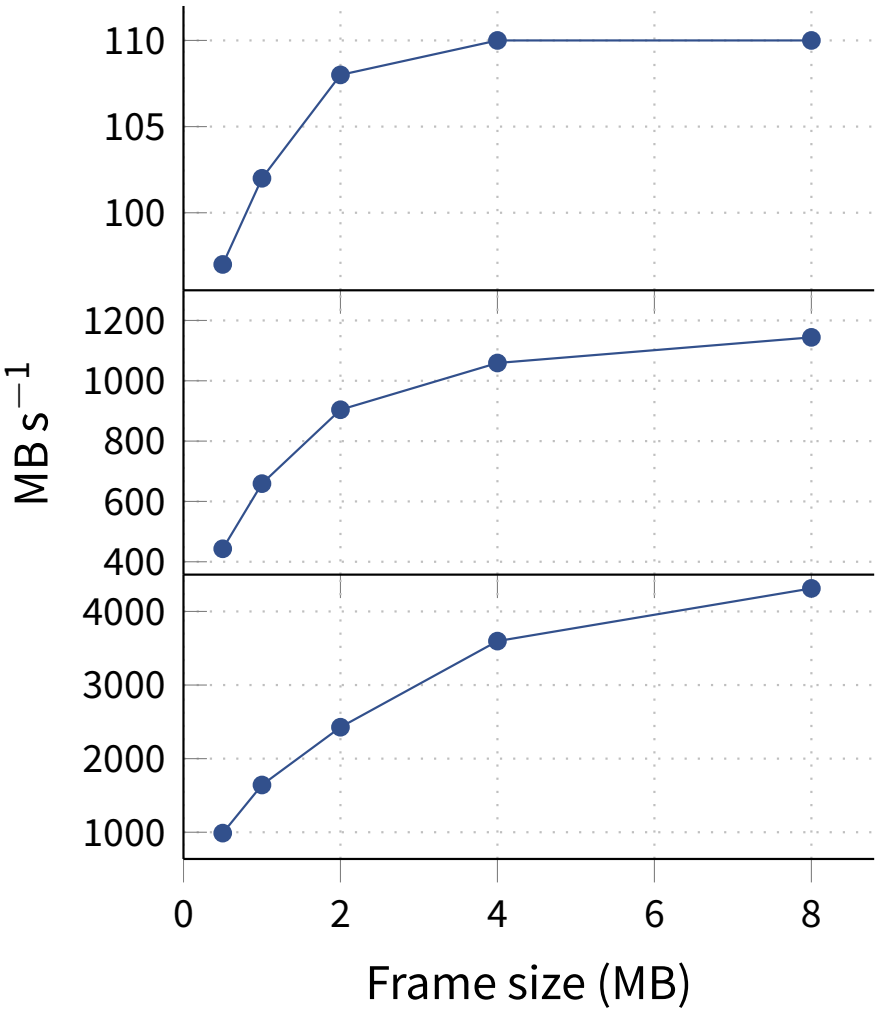
Approach

1. Server process controls camera
2. Proxy communicates with the server
3. Client application uses proxy to control actual camera

No user changes necessary!



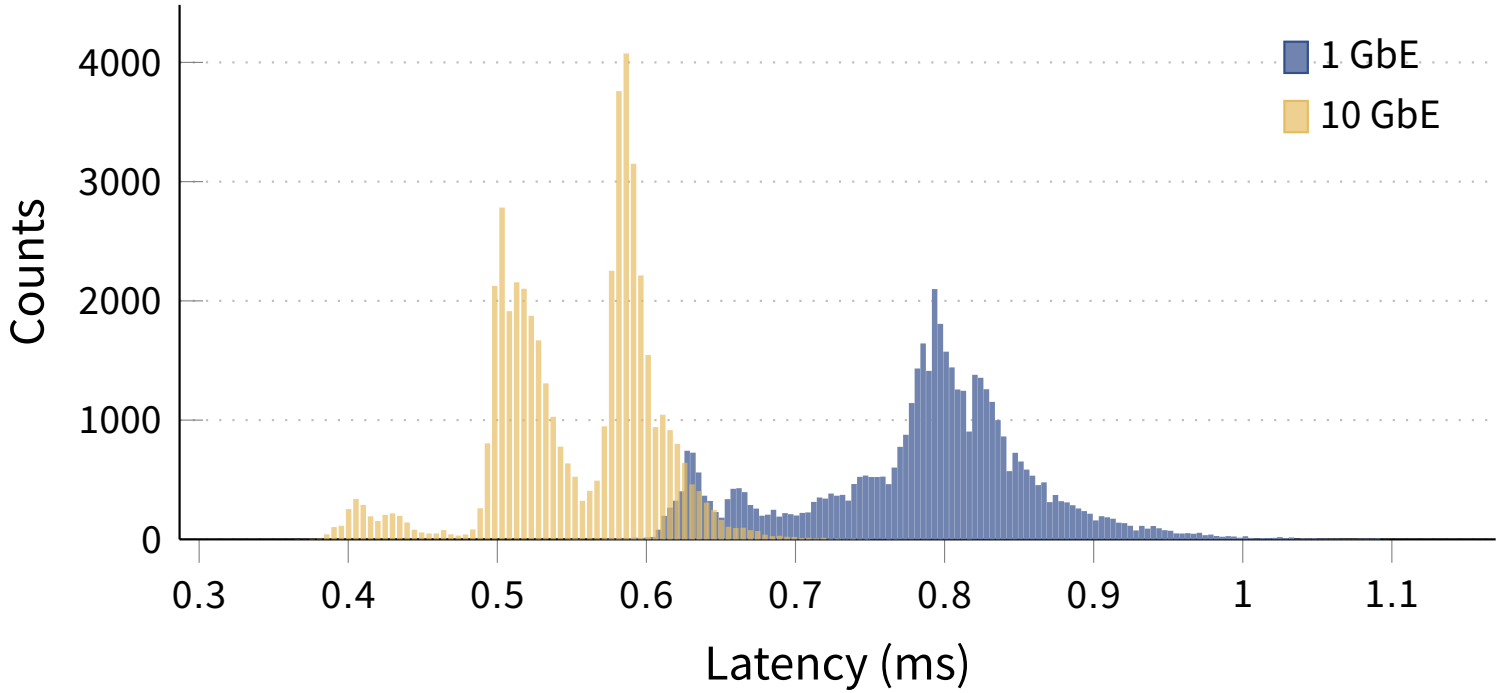
Throughput



Results

- Top 1 GbE, center 10 GbE, bottom loopback
- Close to maximum possible network and memory throughput
- For 10 GbE connections, most cameras become the bottleneck

Latency

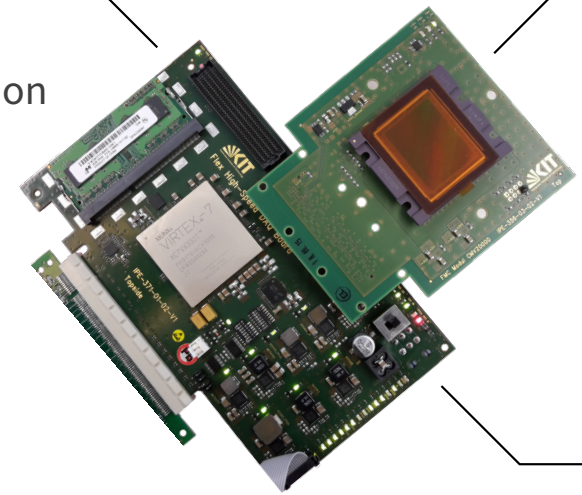

1 GbE
10 GbE
Counts
4000
3000
2000
1000
0
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
1.1
Latency (ms)

DAQ platform

PCIe x8 Gen3

6.6 GB s^{-1}

100 m optical connection



e.g. CMOS image sensor

5.5 µm^2 to 6.4 µm^2

up to 300 frames/s

2–20 megapixel

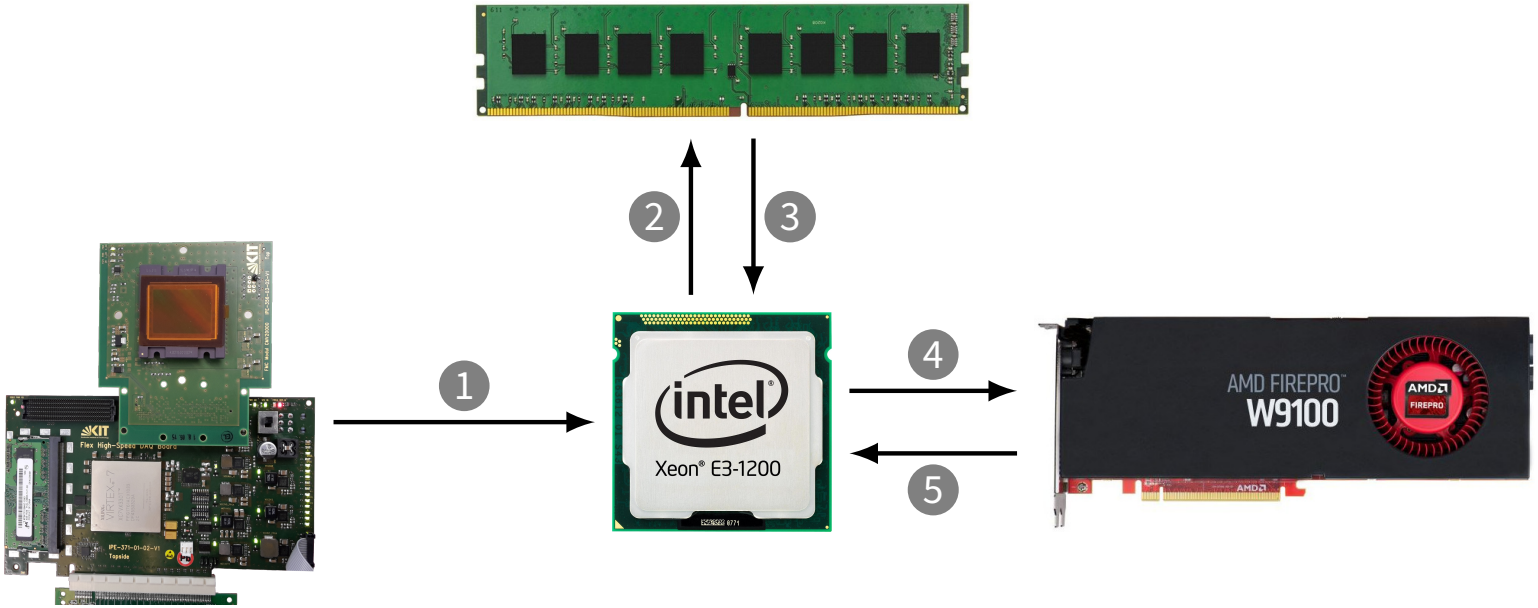
2 HPC FMC connectors

2 GB DDR3 memory

Xilinx Virtex 7 FPGA

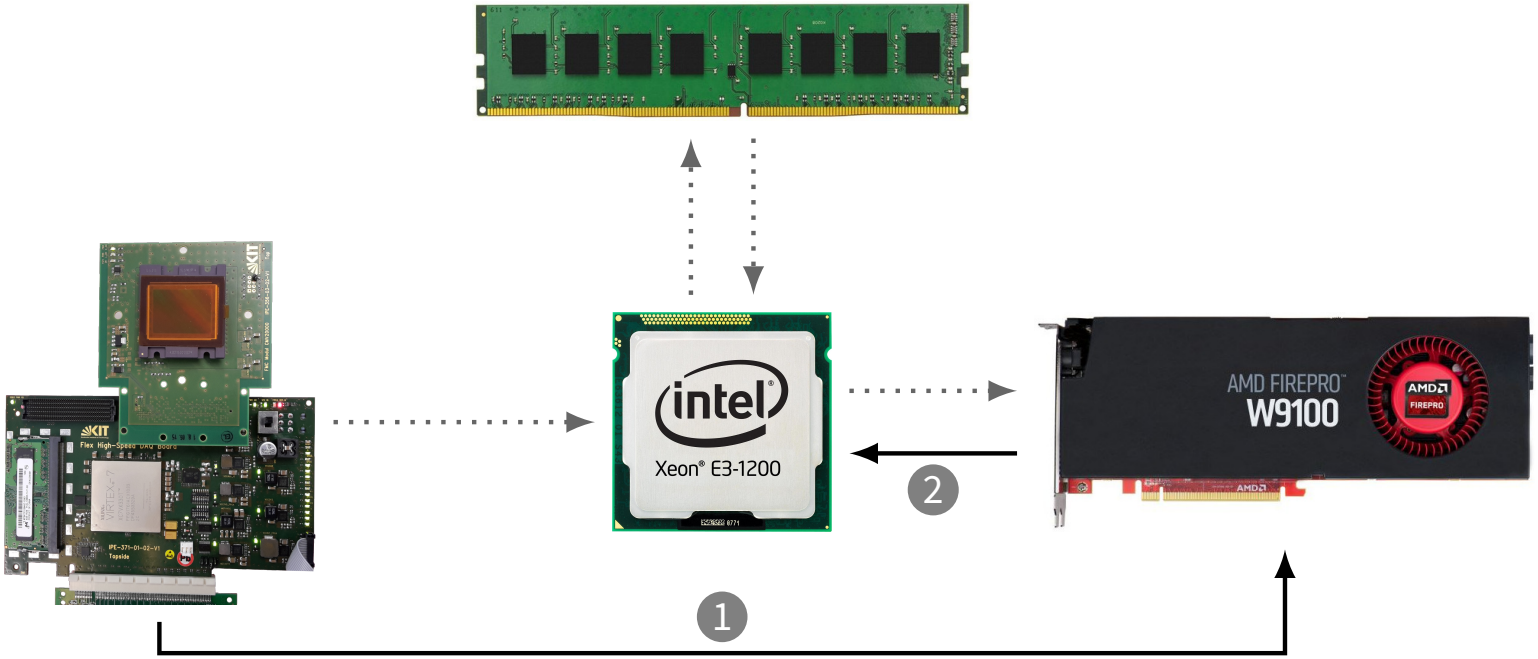
- Inhouse DMA firmware and Linux kernel driver
- Sensor throughput covered by PCIe data transfer rate
- Used for high speed and phase contrast CT experiments

Improving throughput with direct access



- Usually, three copies are necessary (FPGA → RAM → GPU → RAM)

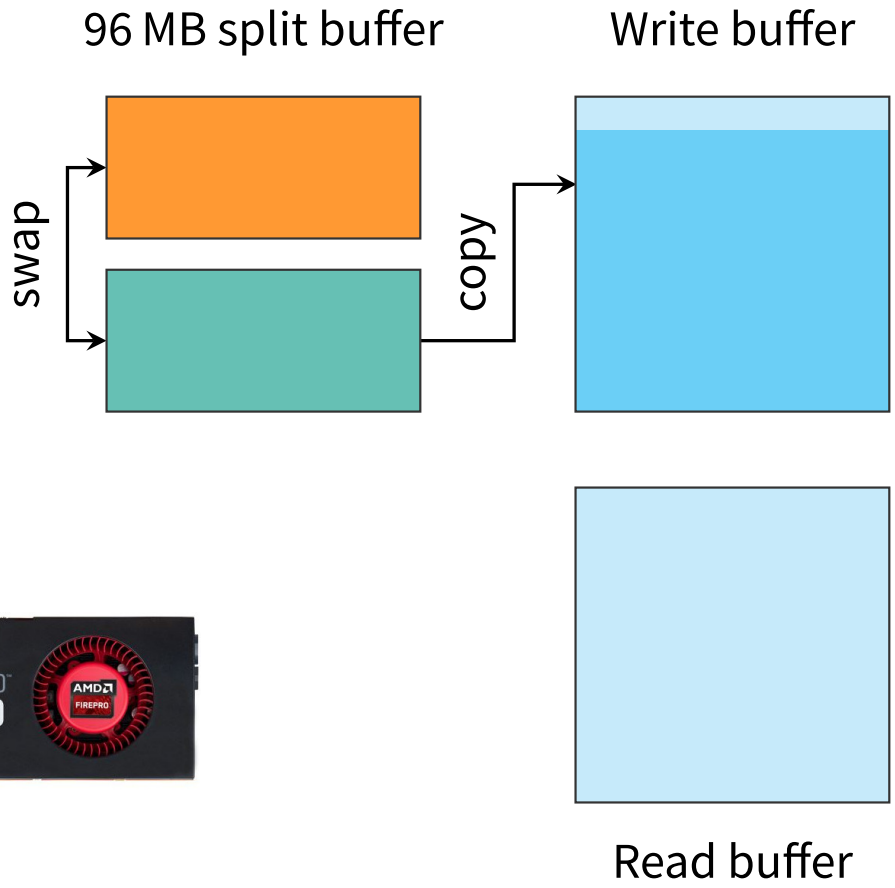
Improving throughput with direct access



- Usually, three copies are necessary (FPGA → RAM → GPU → RAM)
- DirectGMA/GPUdirect reduces this to two copies (FPGA → GPU → RAM)

Buffered transfer

Buffered transfer



Buffered transfer

fill

96 MB split buffer

Write buffer

swap

process

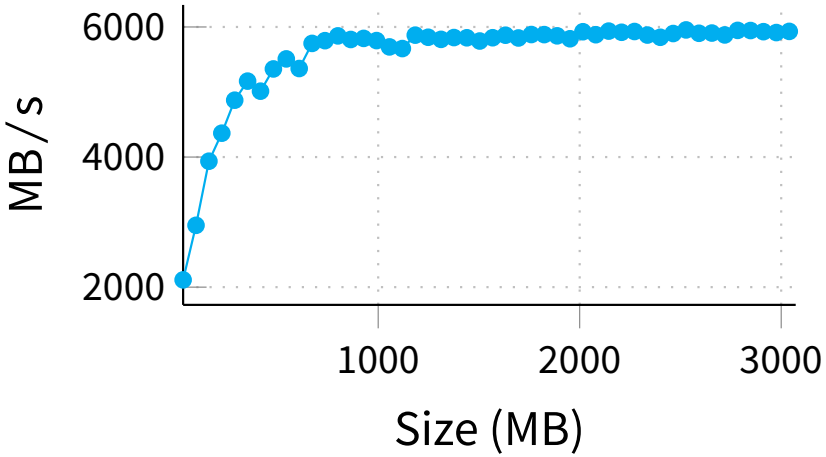
Read buffer

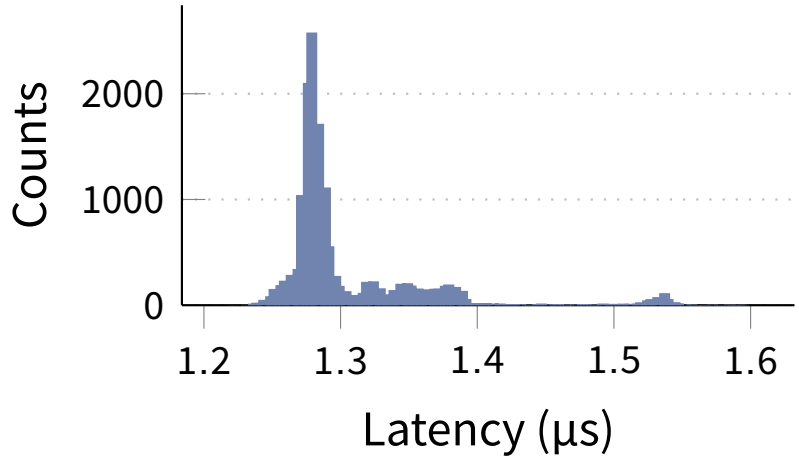
Buffered transfer

Benefits

- GPU can process data while FPGA writes new data
- Keep DMA running and avoid write stalls
- No penalties because intra-GPU writes are faster than PCIe transfers

Throughput and latency





DATA PROCESSING

Heterogeneous and streamed data processing

Karlsruhe Institute of Technology

Applications

- Data pre-processing: filtering, noise reduction, cropping/binning
- Tomo- and laminographic reconstruction
- Post-processing: downsampling, storage, …

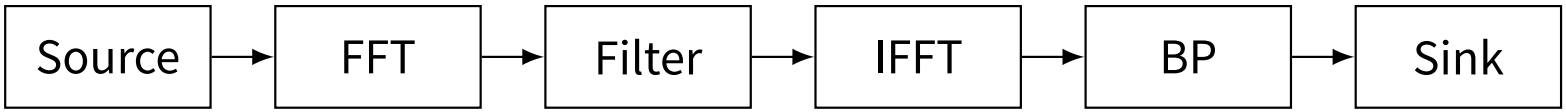
Requirements

- Enable streamed processing and allow integration of data acquisition system
- Utilize all computational resources including multi-core CPUs and multiple GPUs
- Offer wide range of accessibility, from novices to expert programmers

Concept

Core idea

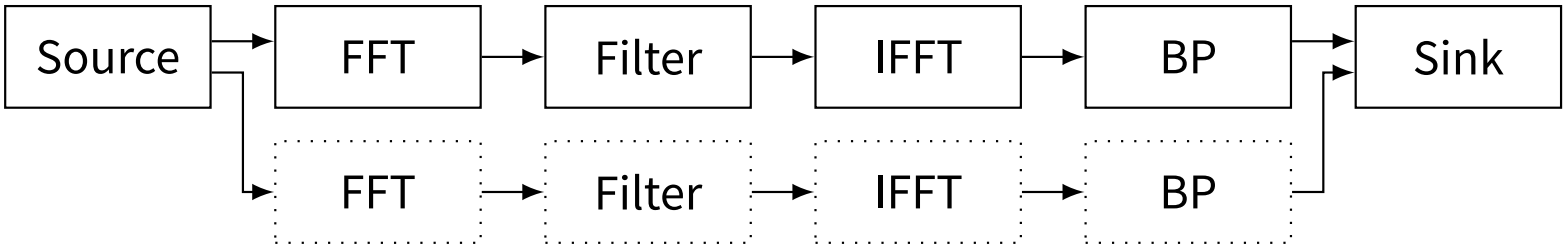
- Connect independent tasks to achieve the desired data flow and results
- Each task may process data in SIMD fashion on a GPU
- Run-time schedules distribution of data and execution of tasks



Concept

Core idea

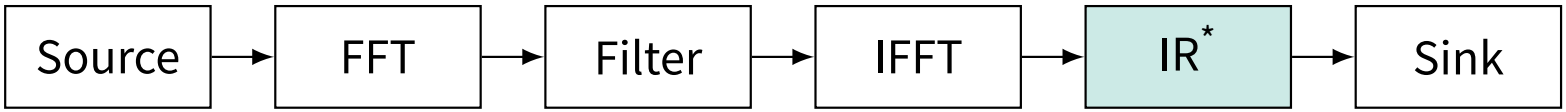
- Connect independent tasks to achieve the desired data flow and results
- Each task may process data in SIMD fashion on a GPU
- Run-time schedules distribution of data and execution of tasks
- Duplicate sub paths for multi GPU execution



Concept

Core idea

- Connect independent tasks to achieve the desired data flow and results
- Each task may process data in SIMD fashion on a GPU
- Run-time schedules distribution of data and execution of tasks
- Duplicate sub paths for multi GPU execution

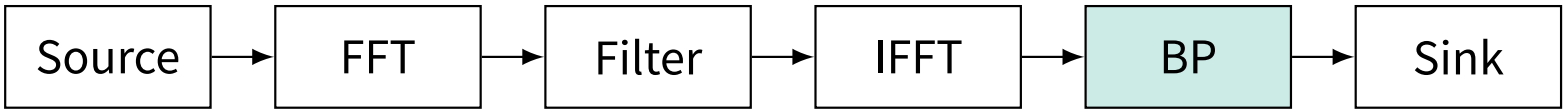


* i.e. SIRT, SART, ASD-POCS, SBTV

Concept

Core idea

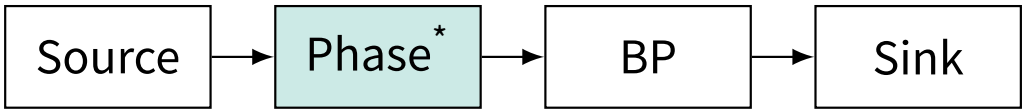
- Connect independent tasks to achieve the desired data flow and results
- Each task may process data in SIMD fashion on a GPU
- Run-time schedules distribution of data and execution of tasks
- Duplicate sub paths for multi GPU execution



Concept

Core idea

- Connect independent tasks to achieve the desired data flow and results
- Each task may process data in SIMD fashion on a GPU
- Run-time schedules distribution of data and execution of tasks
- Duplicate sub paths for multi GPU execution



* Moosmann et al., Opt. Express 19, 12066-12073 (2011)

Implementation

Plugins

- Each task is self-contained and encapsulated in a plugin
- Processes *n* input ports and may reduce *k* input to *m* output items
- Use OpenCL to process data on accelerator, SSE/AVX or regular C on CPUs

DirectGMA plugin

- Handles handshake communication with low-level driver
- Implements presented double buffering strategy
- Converts data to suitable format

Optimization

Multi-GPU computing

- Scheduler duplicates unique paths for execution on multiple GPUs
- Ensures data locality and minimizes data transfers

Pipelining

- Tasks only consume input buffers and only write into output buffers
- Double buffering allows adjacent tasks to work in parallel

Benefits

- Pipelining strategies avoid unnecessary data copies
- Overheads become negligible for realistic datasets
- Close to “bare-metal” performance similar to “monolithic” designs

Command line interface

Idea

- Describe pipeline as program arguments
- Connect tasks with exclamation marks
- Group multiple inputs with square brackets
- Parameterize with key-value assignment

Benefits

- Good for quick one-off jobs
- Elaborate glueing with shell scripts
- Setup and execution done behind the scenes

```
ufo-launch \
  direct-gma ! \
  fft ! filter ! ifft ! \
  backproject \
    axis=1002.03 ! \
  write \
    filename=out.h5:/slice
```

Example: reconstruct from FPGA

Programmatic interface

Idea

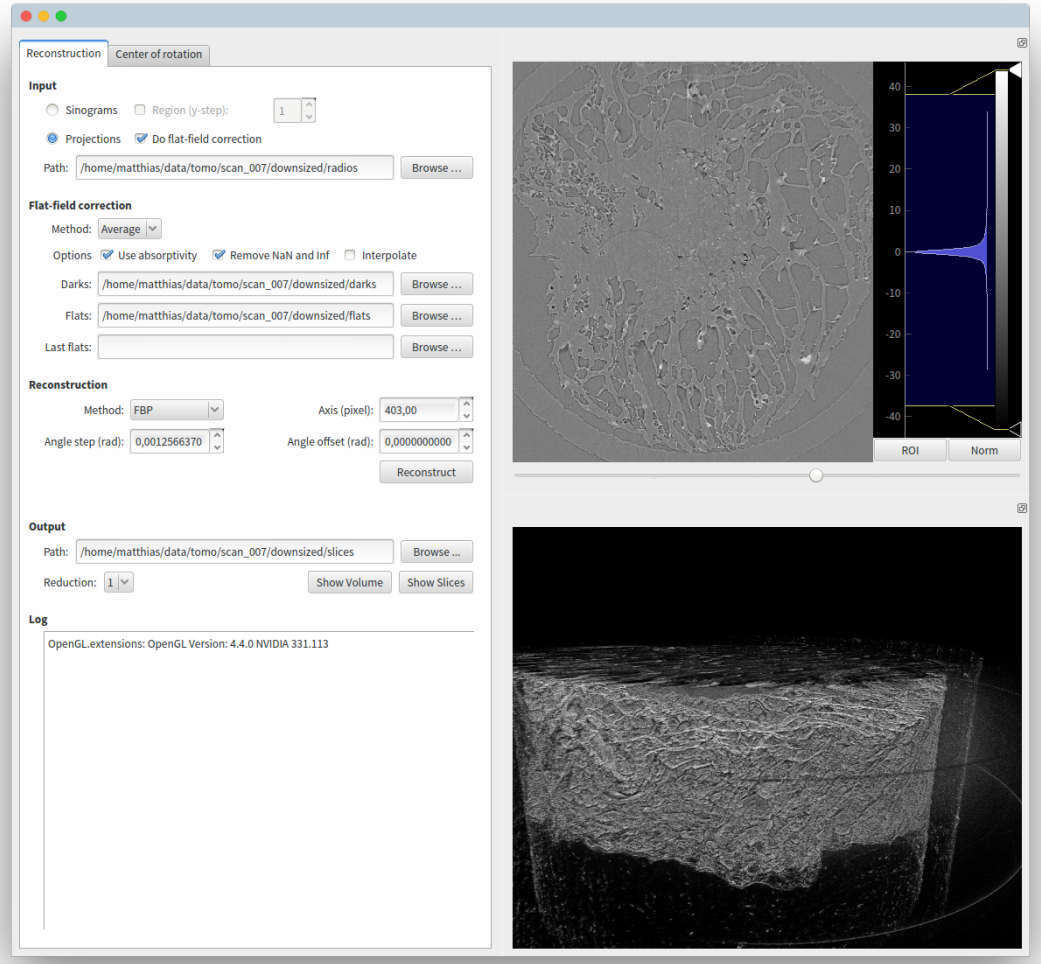
- Connect filters descriptively
- Parameterize execution
- Allow integration as backend

Types

- Native: C and C++
- Bindings: Python, Ruby, …
- High-level wrapper in Python

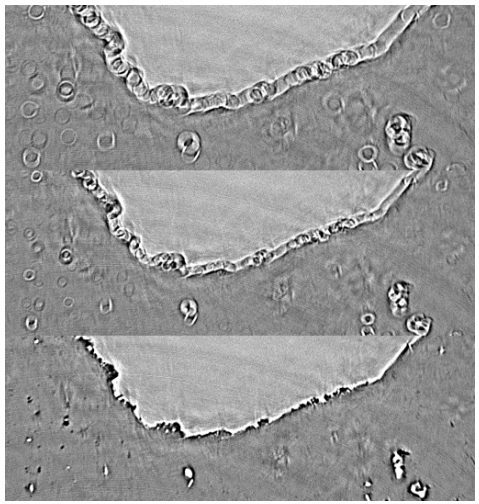
```
from Ufo import (DirectGma,
  Fft, Ifft, Filter, Backproject,
  Write)

dgma = DirectGma()
fft = Fft()
ifft = Ifft()
fltr = Filter()
bp = Backproject(axis=1002.03)
wr = Write(filename='slice-%05d.tif')

wr(bp(ifft(fltr(fft(dgma())))))).run().join()
```

CT reconstruction



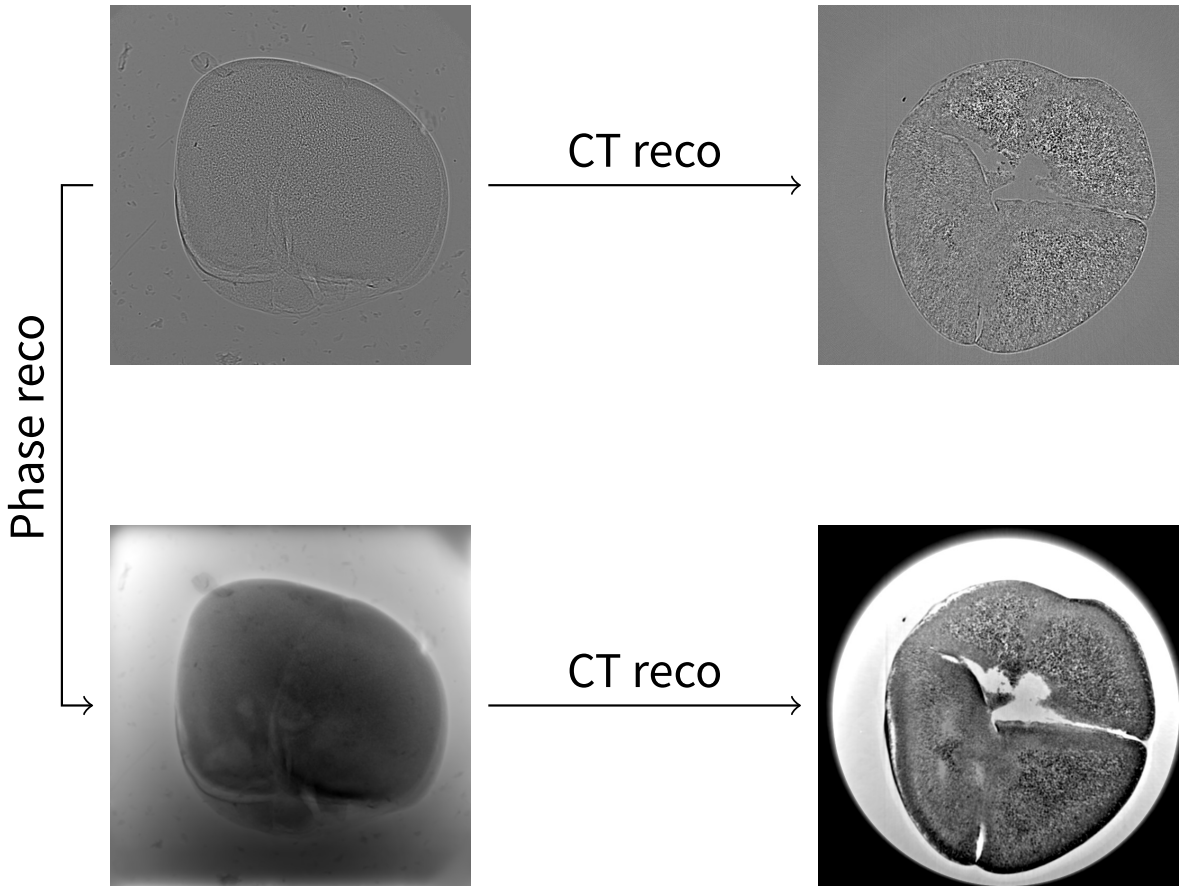
Laminography optimization

- Proper sample alignment “by eye” is difficult and time-consuming
- Reconstruct slices with set parameters
- Optimize rotation axis $\vec{r}$ and tilt angle θ based on an image metric

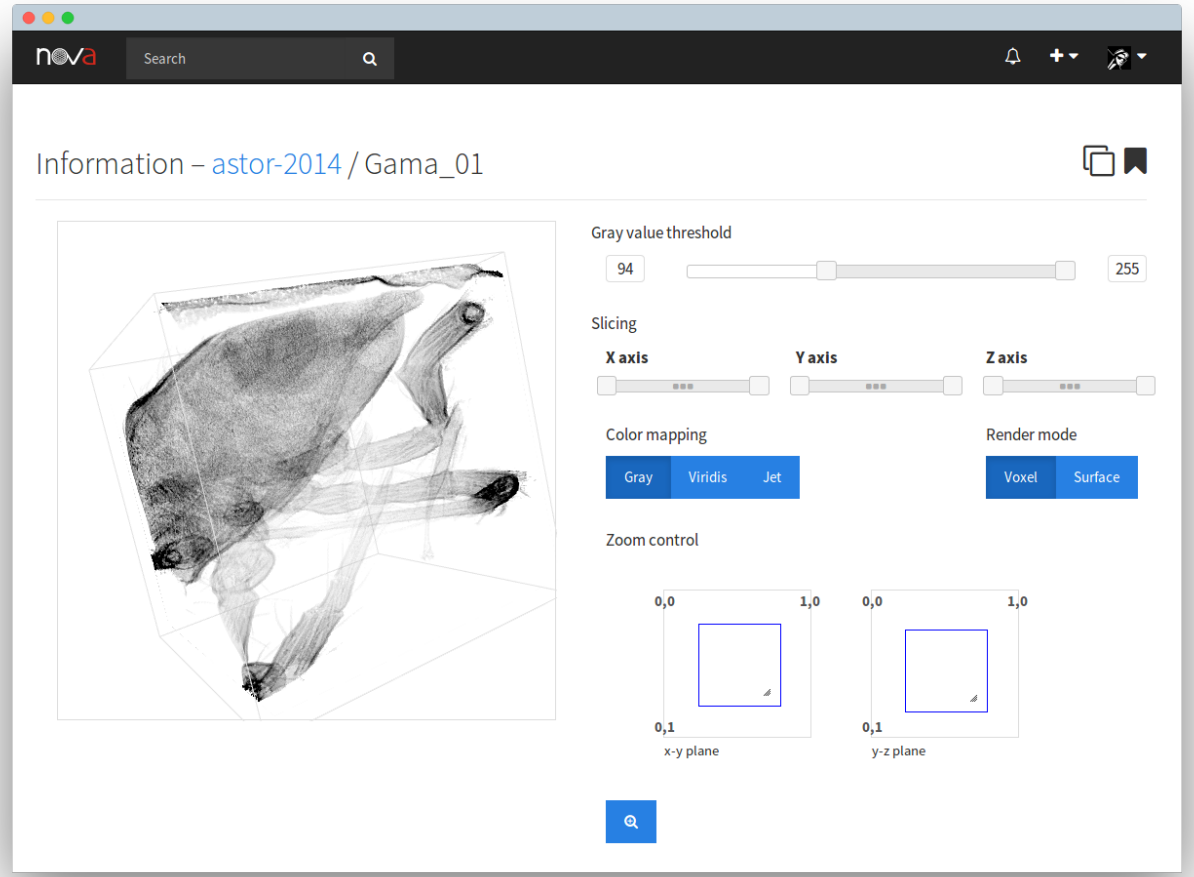


optimize θ and $\vec{r}$

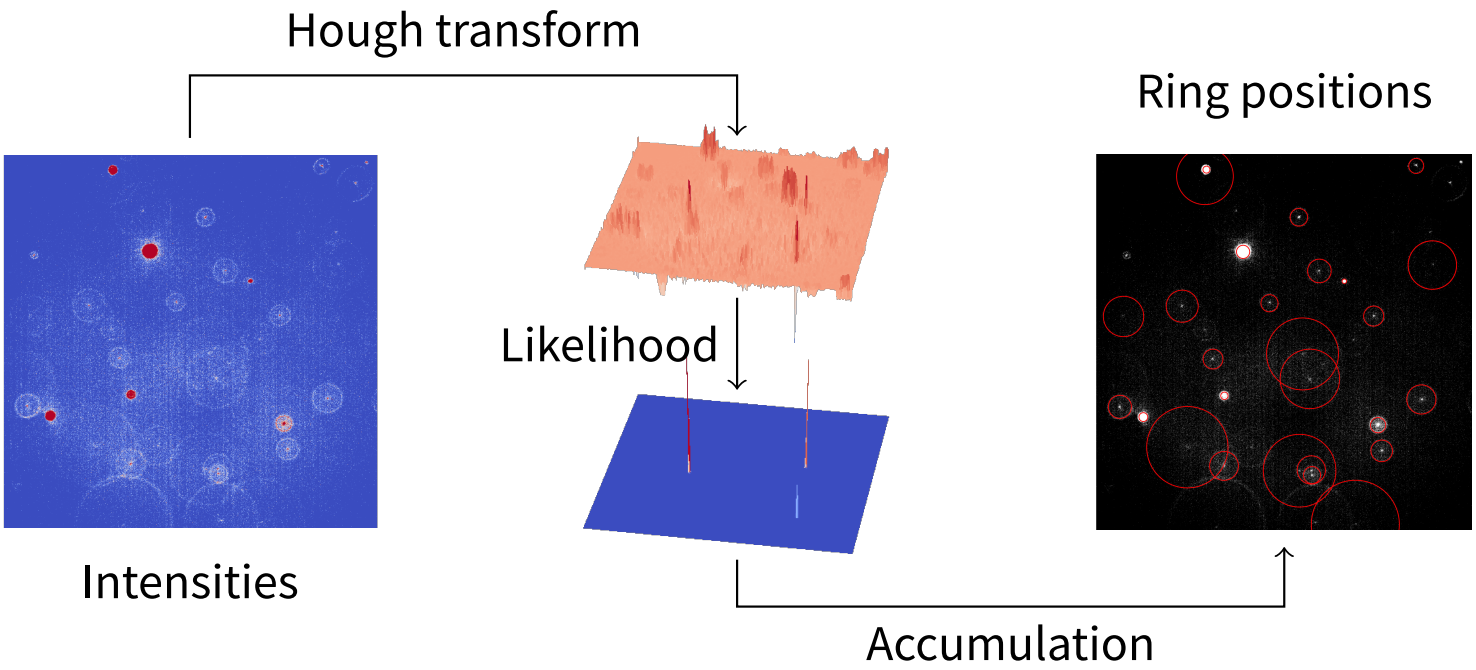
Phase reconstruction



Backend post-processing



Micro Particle Image Velocimetry



KALYPSO data processing

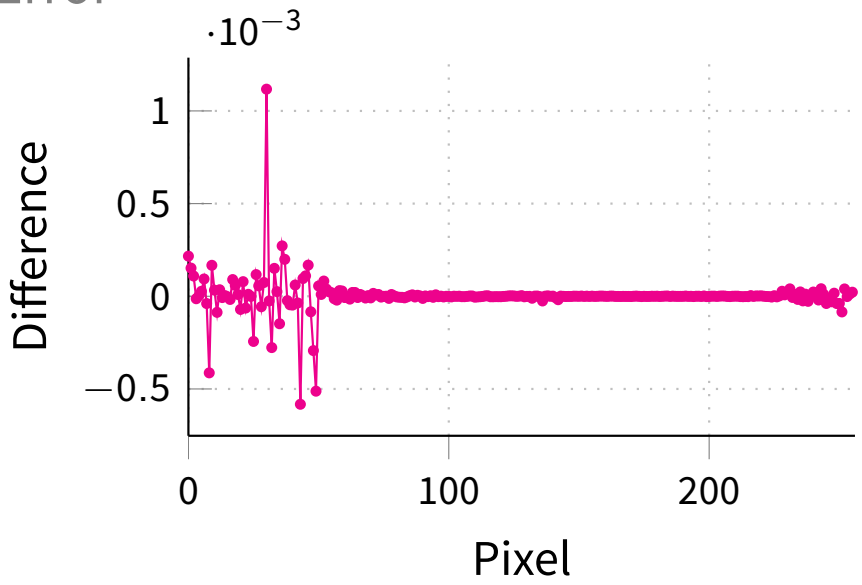
Electro-optical spectral decoding

- KALYPSO linear detector measures longitudinal bunch profiles
- GPU port matches original analysis code
- Data is comparatively small but frequent

Performance

	Standard	UFO
Read	93.41 ms	8.31 ms
Average	27.26 ms	14.97 ms
Correct	301.97 ms	4.56 ms

Error




C O N C L U S I O N

Summary

Data acquisition

- Fast and flexible data acquisition system
- Covers common and not-so-common acquisition scenarios
- Ultra-fast acquisition through DirectGMA integration

Data processing

- Fast and flexible data processing system
- Covers typical pre- and post-processing steps
- Used for experiments in streaming the data makes sense

Summary

Data acquisition

- Fast and flexible data acquisition system
- Covers common and not-so-common acquisition scenarios
- Ultra-fast acquisition through DirectGMA integration

Data processing

- Fast and flexible data processing system
- Covers typical pre- and post-processing steps
- Used for experiments in streaming the data makes sense

Building blocks to construct fast DAQ and processing systems thus bridging ST1 and ST2 of DTS!

Karlsruhe Institute of Technology

Any questions?

Plugin interface

Scope

- Seven core functions are defined and must be implemented
- Start/stop readout, acquire frames, …
- Other functionalities are mapped to type-safe properties

Benefits

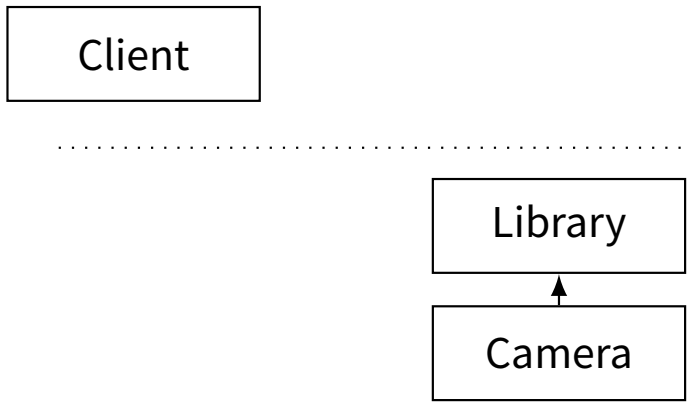
- Mechanisms ease integration and extensions
- Proprietary, binary blob solutions possible …
- Choosing a camera is a run-time decision

Supported cameras

Name	Connection	Pixel size	Resolution	Frames/s
pco.edge	CL/USB	6.5 µm	2560×2160	100
pco.dimax	CL/USB	11.0 µm	2016×2016	1300
pco.4000	CL/USB	9.0 µm	4008×2672	5
Andor Neo	CL	6.5 µm	2560×2160	100
UFO 4MP	PCIe	5.5 µm	2048×2048	180
UFO 20MP	PCIe	6.4 µm	5120×3840	30

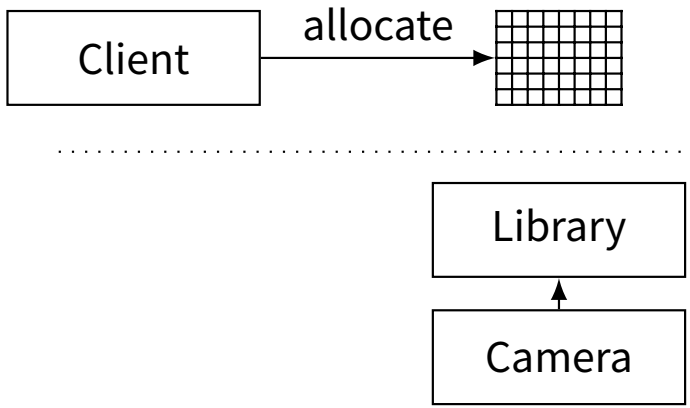
Local data transfers

- Synchronous acquisition and zero-copy



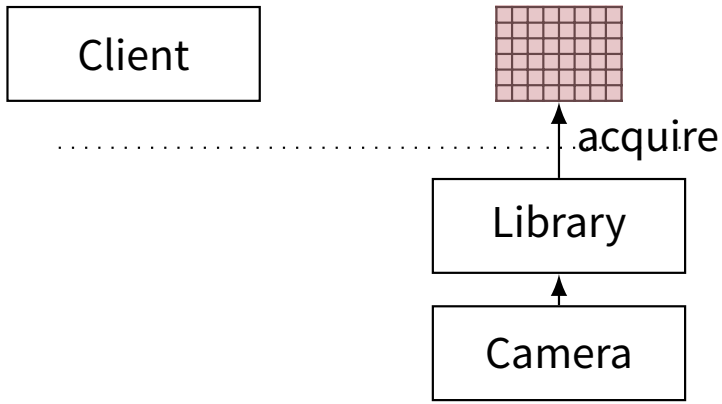
Local data transfers

- Synchronous acquisition and zero-copy means client **must** supply target buffer



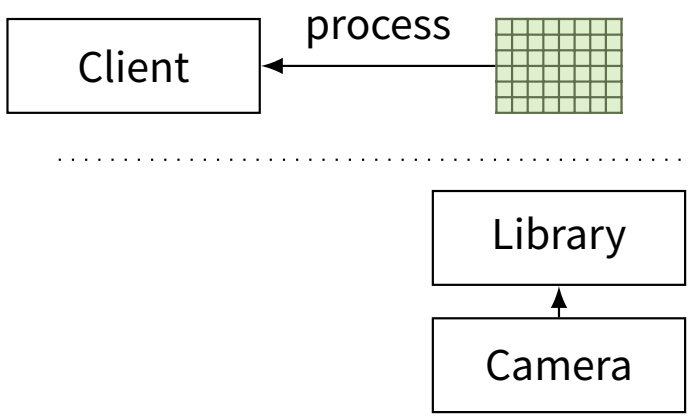
Local data transfers

- Synchronous acquisition and zero-copy means client **must** supply target buffer



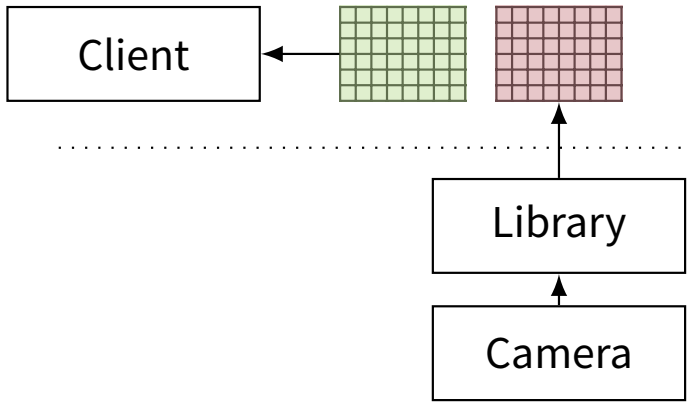
Local data transfers

- Synchronous acquisition and zero-copy means client **must** supply target buffer



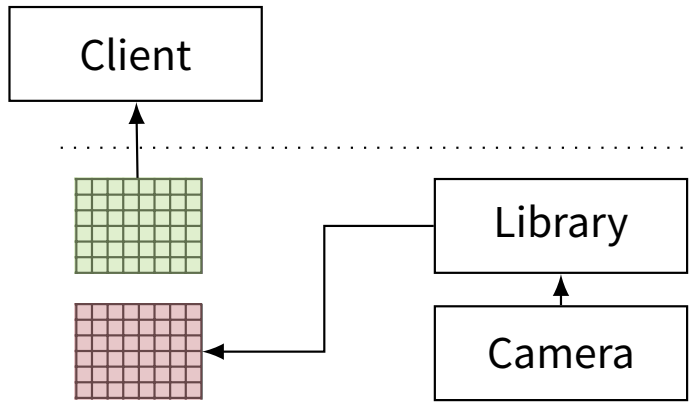
Local data transfers

- Synchronous acquisition and zero-copy means client **must** supply target buffer
- Ring-buffer policy can be implemented client-side



Local data transfers

- Synchronous acquisition and zero-copy means client **must** supply target buffer
- Ring-buffer policy can be implemented client-side
- or use built-in asynchronous, multi-buffered acquisition



DirectGMA usage

Writing to GPU

1. Create OpenCL buffer with `CL_MEM_BUS_ADDRESSABLE_AMD`
2. Determine address by passing buffer to `clEnqueueMakeBuffersResidentAMD()`
3. Set address and paging information in FPGA control registers
4. Busy-wait until FPGA flag signals completion

Writing to FPGA

1. Create OpenCL buffer with `CL_MEM_EXTERNAL_EXTERNAL_PHYSICAL_AMD`
2. Determine physical address of FPGA's "memory space"
3. Pass buffer and address to `clEnqueueMakeBuffersResidentAMD()`
4. Writes from the GPU are proxied transparently to the FPGA

DirectGMA jitter

Karlsruhe Institute of Technology

Problem

- Jitter is still too high for sensitive hard real-time HEP applications
- Longer latencies are caused by host DMA transfers and kernel launch overheads

Solution

- Initiate DMA transfers **within** the kernels
- Single kernel thread programs FPGA address register and polls for data arrival
- To avoid launch overheads, keep the kernel running in a continuous loop

DirectGMA jitter

Problem

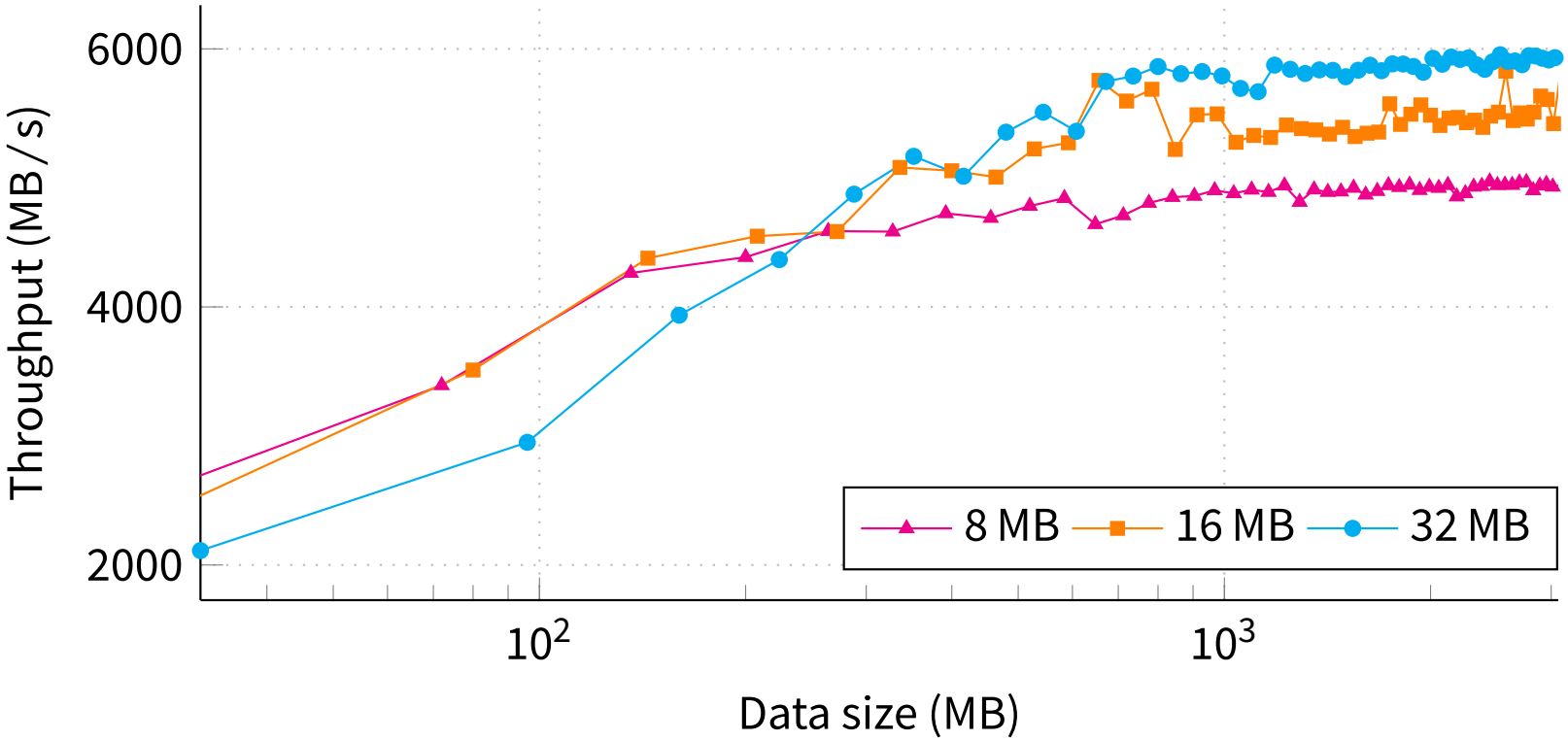
- Jitter is still too high for sensitive hard real-time HEP applications
- Longer latencies are caused by host DMA transfers and kernel launch overheads

Solution

- Initiate DMA transfers **within** the kernels
- Single kernel thread programs FPGA address register and polls for data arrival
- To avoid launch overheads, keep the kernel running in a continuous loop

Stable latencies but application code is tied to hardware …☹

DirectGMA throughput



DirectGMA latency