Processing HDF5 data with FPGAs

Zdenek Matej [1], Kenneth Skovhede [1,2], Carl Johnsen [2], Artur Barczyk [1], Clemens Weninger [1], Andrii Salnikov [1] and Brian Vinter [2]

[1)] **MAX IV Laboratory, Lund, Sweden**

[2)] **Niels Bohr Institute, København, Denmark**

This project has received funding from the European Union´s Horizon 2020 research and innovation programme under grant agreement no. 101004728

How one can get to HDF5 data processing with FPGAs?

MAX IV synchrotron laboratory … few years ago

We are writing a grant application for dealing with synchrotron and XFEL data and we will use GPUs.

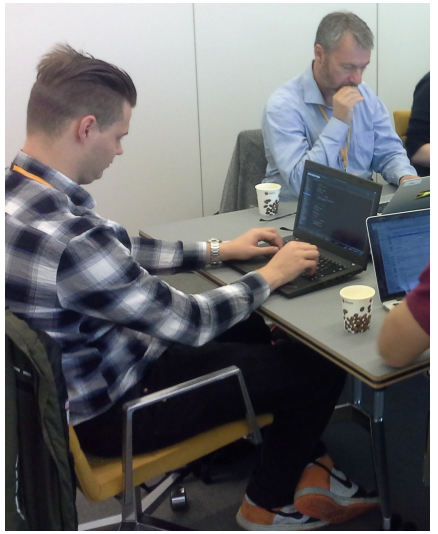
Hmm, GPUs. Are you using also FPGAs for these applications ?

No, bla, bla, bla … (many excuses).

(Thought): How one can do it?

Artur – physics, CERN, MAX IV networks and infrastructure

Zdenek - scientific sw, material science & diffraction methods



- Xilinx Kintex Ultrascale+ xcku5p at 590 MHz at ~3 % utilization, resulting in an inner processing speed of 32 x 590 = ~18880 Mbps. I.e. 18880 x 33 = ~623040 Mbps = ~623 Gbps

"bincounting"

github.com/bh107/SME-Binning

Kenneth – expert in FPGA languages, "inventor" of SME*

Carl … another FPGA genius from NBI

Brian … professor at NBI

Processing photon science data with FPGAs

HDF5 and heterogenous hw accelerated compute

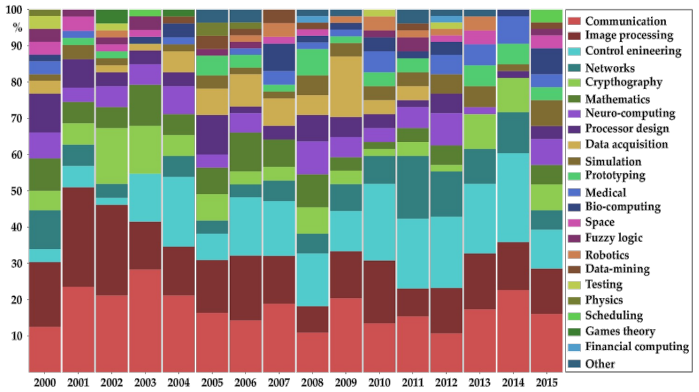
- frame corrections
  - conversions
  - mask
  - flat field
- simple processing
  - image binning
  - roi-counting
- spot finding
- GPU like applications
  - **azimuthal integration**
  - phase retrieval
  - full field tomography reconstructions
- (de)compression
- ML inference
  - image classification
- frame & event filtering

Credit: The HDF Group - maintainers of HDF file format

- all x-ray scattering and imaging data acquired by MAX IV DAQ are stored in the HDF5 format
- bslz4 is the default compression for 2D detector image data (frames)

FPGAs pros

- real time
- parallel, high performance and throughput
- integrated networking - couple with detectors
- power effective
- common hw with many other applications (like GPUs)



2010–2015: *800 FPGA application reported* by J. Romoth et al. (2017), doi: 10.13140/RG.2.2.16364.56960 + *increasing variety*

cons

- ~~difficult to program~~
- ~~difficult to install~~
- not so rich scientific sw ecosystem (?)

Existing FPGA applications

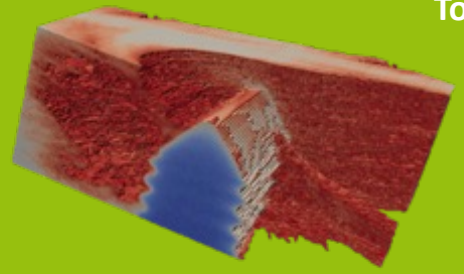
in Photon and Neutron (PaN) sciences

- *Maxime Martelli et al.* (CNRS, Paris)
  3D Tomography Back-Projection Parallelization on Intel FPGAs Using OpenCL
  - doi: 10.1007/s11265-018-1403-6

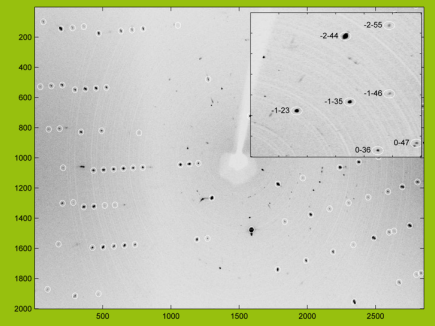
- *Filip Leonarski et al.* (SLS/PSI)
  Fast and accurate data collection for macromolecular crystallography using the JUNGFRAU detector
  - doi: 10.1038/s41592-018-0143-7

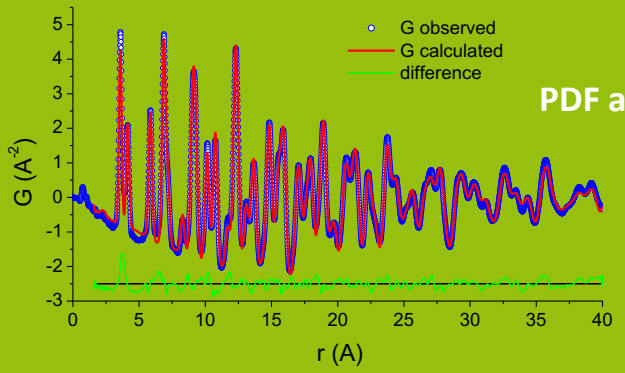
- *César González et al.* (Barcelona)
  High Performance Computing PP-Distance Algorithms to Generate X-ray Spectra from 3D Models
  - doi: 10.3390/ijms231911408

Tomography reconstructions



Spot-finding



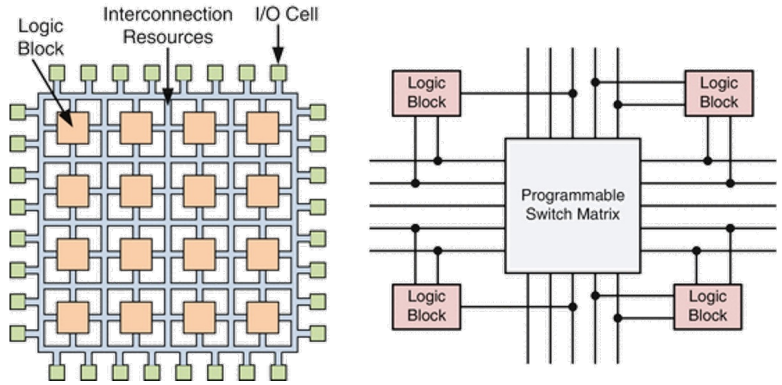


PDF and small angle scattering

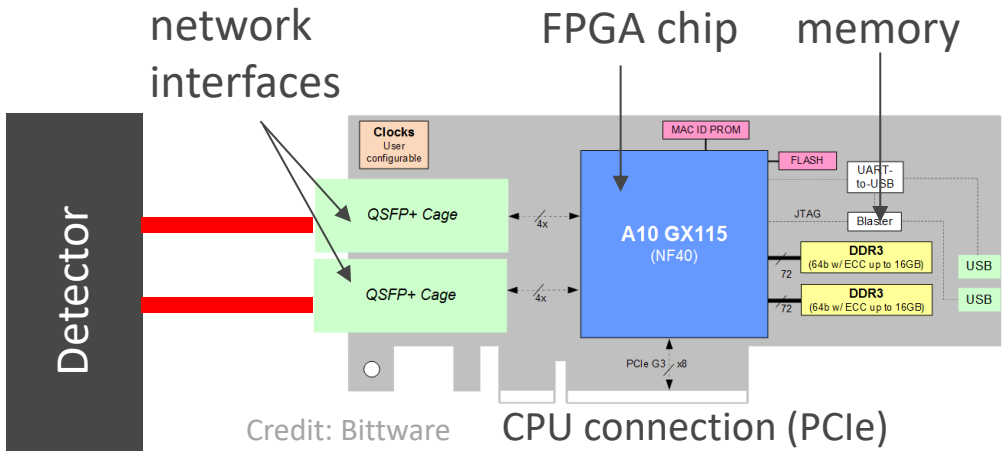
Field Programmable Gate Arrays

FPGAs

- arrays of logic gates
  - O(10) million gates today in largest FPGAs
- ‘programmed’ by creating connections between logic blocks
- arranged in functional blocks
  - DSP, logic, memory, I/O
- O(10) Tflops, comparable to GPUs
- (ordinary) programmed in specialized languages, adapted to electronics development (Verilog, VHDL)
  - intellectual property cores (~sw module) can be purchased
- ideal for bit-level data manipulation
- FPGA boards: integrated network interfaces

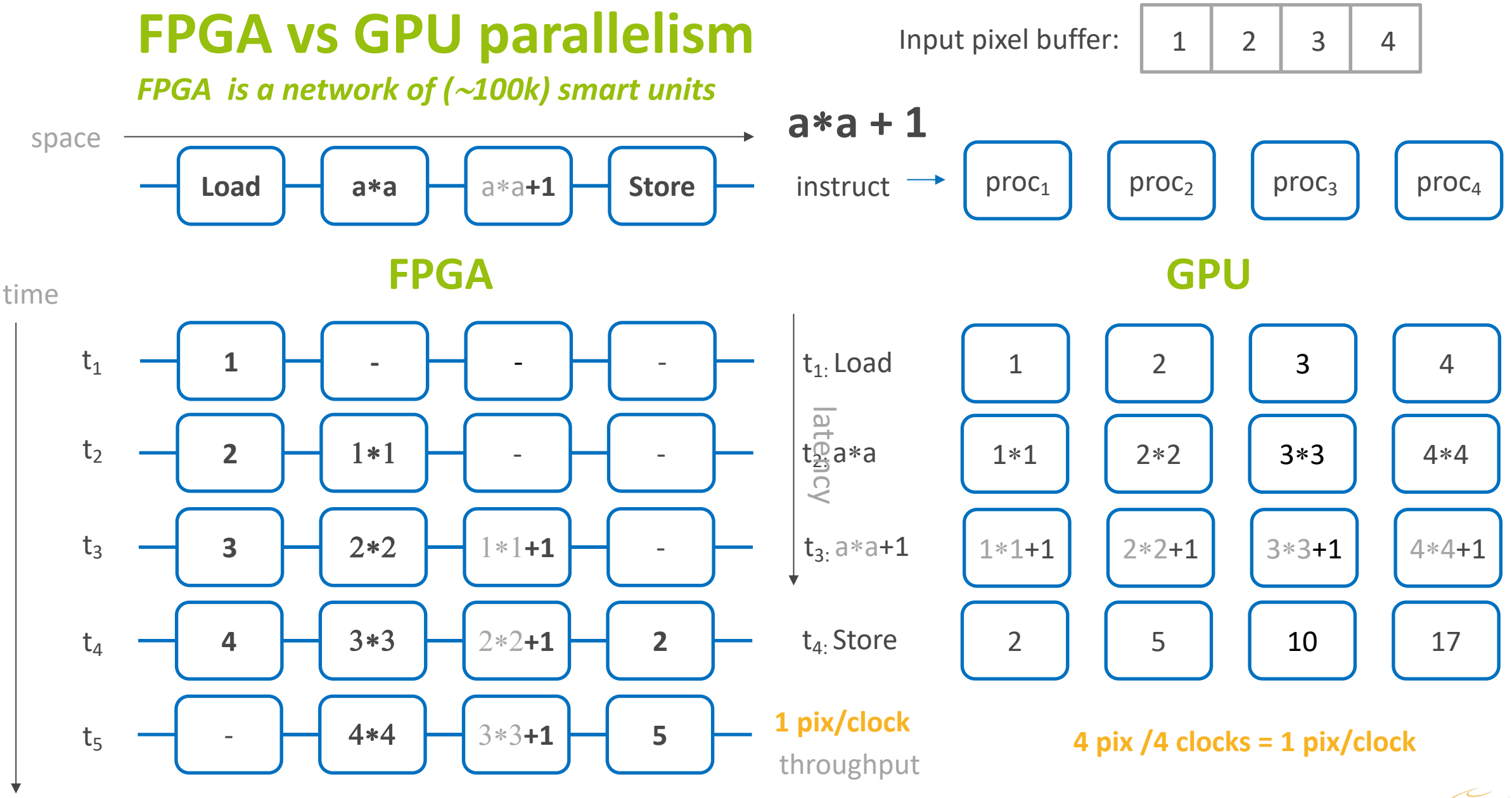


Credit: Kovačec, doi: 10.1007/978-3-319-14346-0_40



FPGA vs GPU parallelism

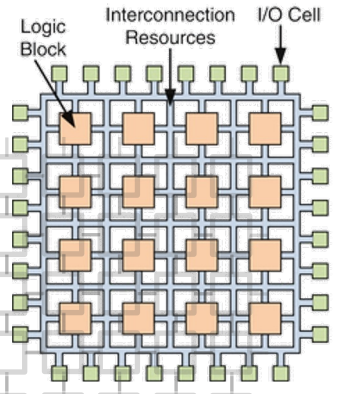
FPGA is a network of (~100k) smart units

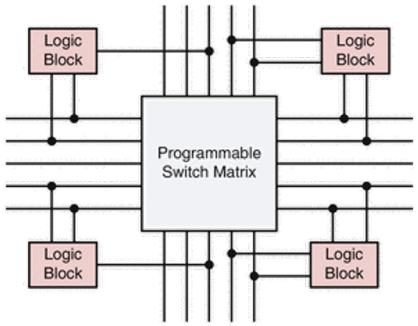


FPGAs as a flexible factory

very parallel hardware *~100k of logical units*

Credit: Kovačec, doi: 10.1007/978-3-319-14346-0_40





Framework 20%

Global MEM interconnect (PCIe/HBM) 10%

bszl4 decompression 5%

AZ INT 20%

other parallel applications ~35%

Local MEM interconnect (cache) 10%

Water mill powered saw in Wallachian museum (CZ)
Credit: Jan Polák

Note: frequencies are lower than on CPUs.FPGAs have ~100 MHz – 1.5 GHz

Access to FPGAs

One size does not fit all

- cost and power effective FPGAs … for professional applications and hobbyists … or low power (outer space applications)
  - 100k gates, O(0.5) GB mem, 1 Gbs Ethernet, O(100) MHz, O(2) W
  - O(200) USD (development kit or module/minicomputer)
  - shop: distributors of electronic components
  - popular development kits: PYNQ for Zynq SoC or DE10-Nano Kit
  - sign for academic programmes
- medium size FPGAs
  - 1M gates, O(16) GB mem, 40-160 Gbs, O(300) MHz, O(50) W
  - O(4-6) 1000 USD
- large and extra large FPGAs
  - O(10)M gates, O(32) GB mem, 400 Gbs – 1.2 Tbs, O(500) MHz, O(150) W
  - > O(6-14) 1000 USD
- DevCloud: github.com/intel/FPGA-Devcloud ⭐

Data processing with FPGAs

Definitions

Fast

- low latency or high throughput ?
- processing independent data elements: frames, pixels, bytes, bits:
  - 4 Mpix detector, 2 **kHz** frame rate
  - => 4Mpix * 2kHz = 8 **Gpix/s**
  - (16 bits int) 2 **B**ytes/pix => 16 **GB/s**

- **fast … high throughput**
  - … many frames per sec (Hz>>1)
  - … many pixels/s (Gpix/s>>1)
  - … many bytes/s (GB>>1)

Real time

- **predicable, fixed** and low **latency**
- FPGA clocks O(200) MHz -> 5 **ns**
- FPGA clock can be synchronized with external timing system (accelerator/machine) with sub-pico-second *precision*
- processing a diffraction **image** may be many clocks … 32 pix/clock, 4 Mpix image => 4 Mpix / 32pix * 5 ns = 0.625 **ms**

Curiosity:

- CPU servers … O(5) Tflops
- GPUs … O(10) Tflops
- Detectors: O(2) Gpix/s … factor of >2000 ?

Programming FPGAs for scientific data processing

tools

- highl level tools for FPGAs
  - **Synchronous Message Exchange** (SME): github.com/kenkendk/sme
    - *Kenneth Skovhede* & *Carl Johansen* (Niels Bohr Institute)
    - C#/C++/Python -> SMEIL (SME intermediate language) -> VHDL -> bitstream
    - vendor agnostic
  - **OpenCL or SYCL/OneAPI**
    - "C/C++ with pragmas", OpenCL is an older standard used also for GPUs
    - FPGAs in DevCloud or in production at MAX IV
  - **High Level Synthesis** (HLS)
    - "C/C++ with pragmas"
- orchestration on host (CPU)
  - **Python** or **C++**
- HDF5
  - **libhdf5** or **h5py**

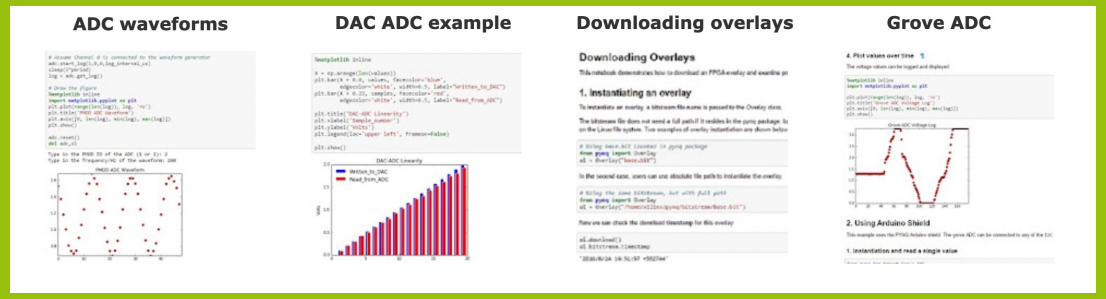
language: OpenCL/C
code is hw agnostic
alg. core: 36 lines
host: PyOpenCL

```
/* --- run bitonic merge sort network */
#pragma unroll
for(short row=BN_LEN-1; row>=0; --row) { // program
    #pragma unroll
    for(short col=0; col<BN_N; ++col) {
        // read the program
        const bool ishigh = (bool)_bp[row][col][0];
        if(ishigh) { // only one (high) needs to do an action
            const short remote = _bp[row][col][1];
            ulong2 slow, shigh;
            ushort xlow, xhigh;
            bool flow, fhigh;
            shigh = bisort_mem[ibis][row][col];
            slow  = bisort_mem[ibis][row][remote];
            xlow = GET_POS(slow.x);
```

Frameworks

Hardware and language is not all that matters

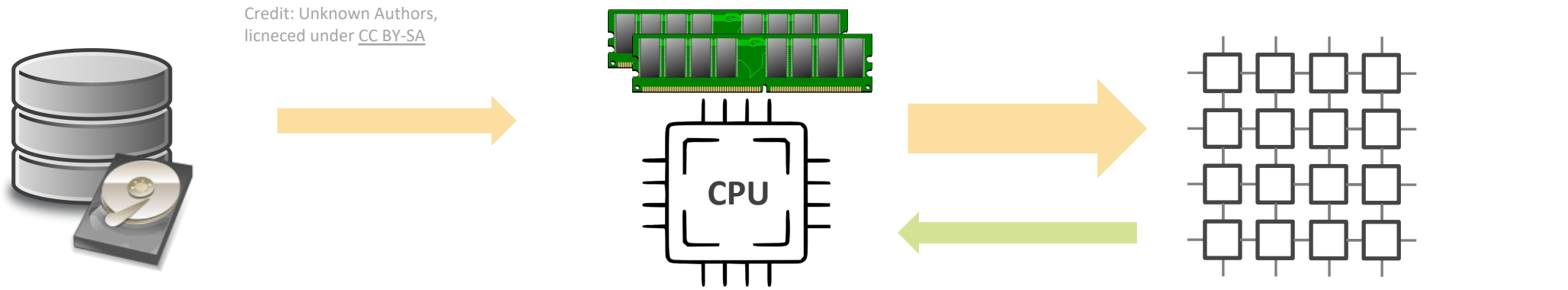
- **Using a “productivity” framework may accelerate the FPGA project, . . . especially if one is starting with programming hardware.**
- Popular frameworks:
  - PYNQ: HLS + Python and Jupyter (notebooks)
  - **OpenCL**
  - **OneAPI (C++) + SYCL**
  - OpenCAPI + HLS
- Advantages:
  - No need of “writing Linux/Windows drivers”
  - No need of extensive knowledge of FPGA-tools ecosystem for electrical engineers
  - Simplified FPGA (re)programming
  - Diagnostic utilities
  - API for C++ or Python



www.pynq.io

HDF5 data with FPGAs

key components

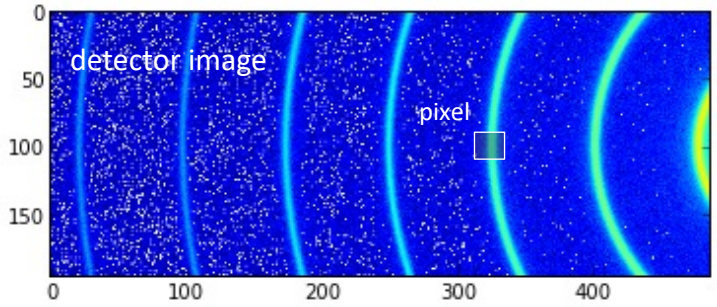


HDF5 direct chunk read
is the “key” ingredient that allows this work.

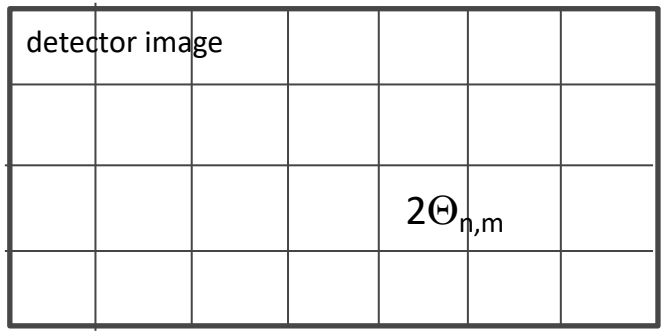
decompression on CPU is optional … should be removed in future … 2nd part of the talk

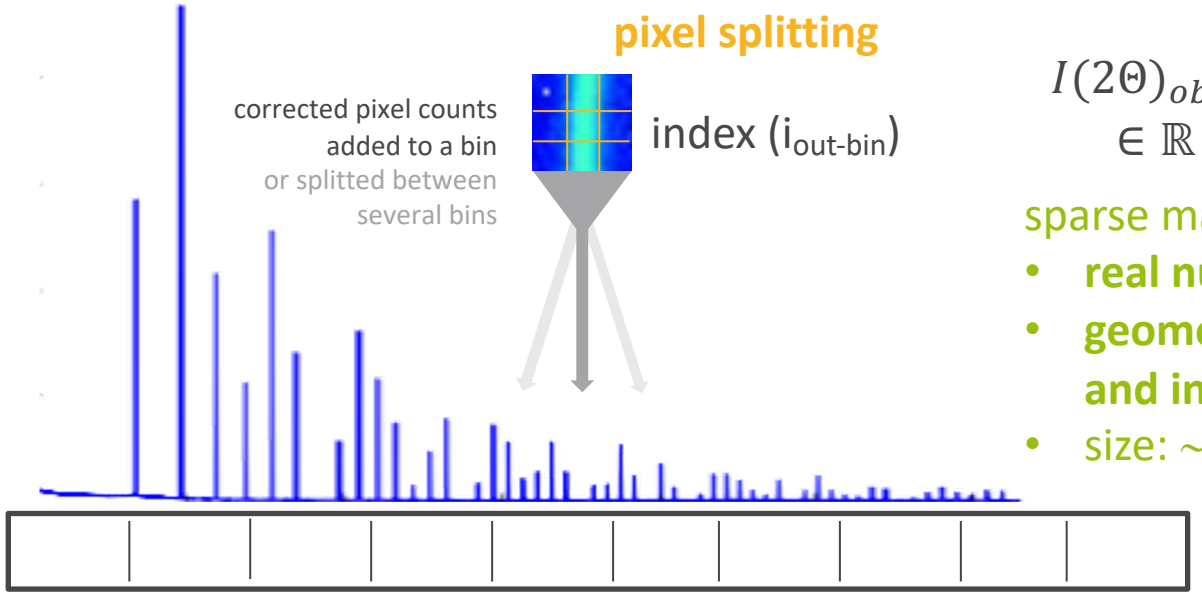
Azimuthal integration

Data processing example: 2D image -> 1D pattern



- for each image pixel (n,m->i_{pix} … linearized index):
  1. trigonometric calc. -> scattering angle ($2\Theta_{n,m}$)
  2. binning ($2\Theta_{n,m}$) -> single index ($i_{out\text{-}bin}$)
  3. intensity corrections
  4. histogram, bincount or sparse matrix multiplication





$$I(2\Theta)_{obin} = \sum_{ipix=0}^{N_{pix}} A_{obin,ipix} \; Img_{ipix}$$
$$\in \mathbb{R} \qquad \in \mathbb{R} \qquad \in \mathbb{Z}$$

sparse matrix ***A*** . . .

- **real numbers**
- **geometry, calibration, physics of pixel and intensity corrections**
- size: ~ 10k x 1M

σ_I . . . **experimental error propagation**

keeping flexible on CPU

- split physics and computation
- use
  - bincount
  - sparse matrix multiplication

NumPy

fast with FPGA

Basic algorithm on FPGAs

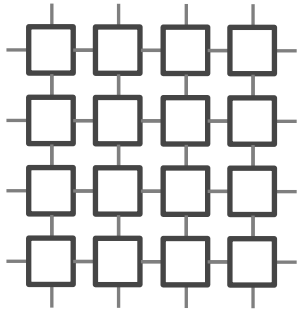
bincount - first implementation

- **bincount: simple algorithm similar to computing histogram**
  - result kept in “local” memory (FPGA Block or Ultra RAM)
  - If position_old == position_new:
    - True: accumulate (sum values)
    - False: store old acc, load new acc, accumulate
  - **1 pixel per clock (per processing unit)**
- by *Carl Johansen* (NBI) using Synchronous Message Exchange (SME) – in 1 day
- initially only for integers
- performance numbers:
  - small FPGA at 100 MHz: 1 Gpix/s (10% util. per processing unit)
  - large FPGA at 590 MHz: **20 Gpix/s** (3% util. per unit)
- ref: github.com/bh107/SME-Binning
- OpenCL/OneAPI/HLS:
  - one of many FPGA programming “tricks” often called “Shannonization” (Claude Shannon)

Basic algorithm on FPGAs

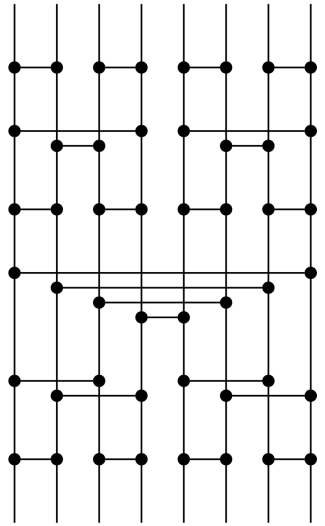
bincount + bisort & resort for floating point data

- initially only for integers -> real-numbers corrections -> floats(32)
- need for pixel “**reordering**” (bisort + resort)
- again 1 pix/clock/pipeline



can be expressed as
systolic array
-> very suitable for FPGA

Bitonic sorter



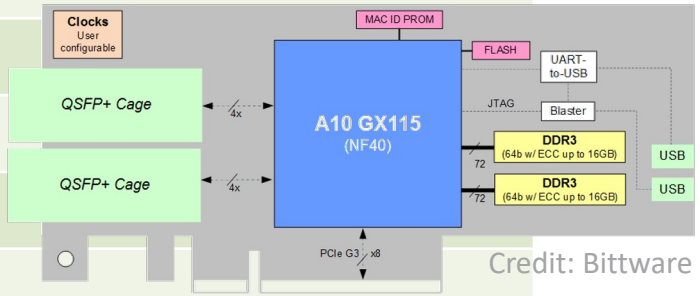
Credit: by Octotron - own work, CC BY-SA 3.0

- Performance prediction:
  - 1 pipeline … 1.25% of FPGA resources
  - 1 decompression pipeline … 1%
  - 20% for the framework + 32 pipelines ∗ (1.25+1)% = 92% FPGA utilization
  - 1 pix/s ∗ 32 pipelines ∗ 360 MHz = **11.5 Gpix/s**

Performance figures

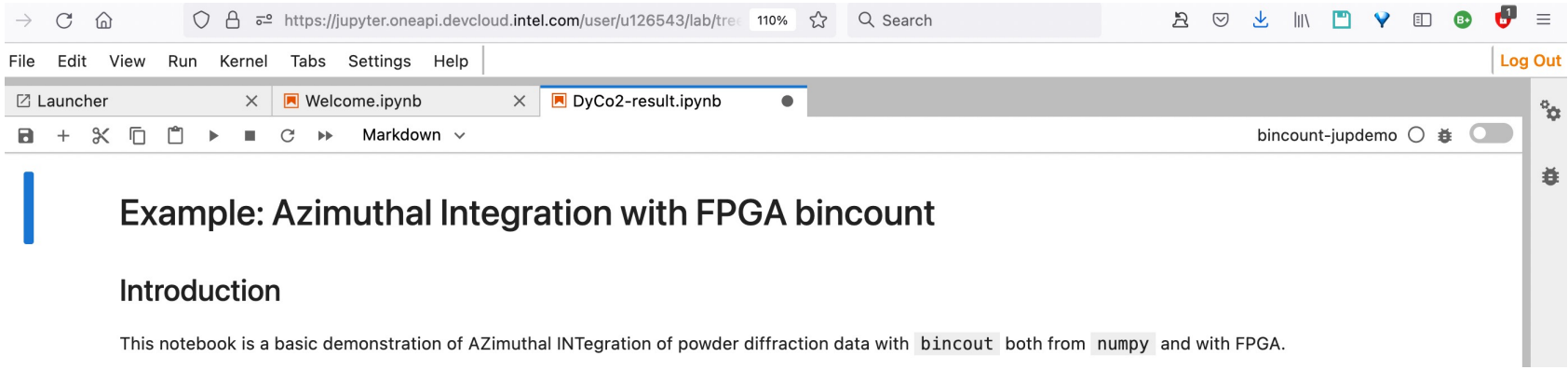
AZINT bincount – FPGAs with OpenCL, no decompression

	Medium	Large	comment
size	medium	large	
FPGA	Aria 10 GX	Stratix 10 MX	
process	20 nm	14 nm	
memory	2 x DDR3	2 x HBM2	
QSPF	2x10/40 Gbs	4 x 100 Gbs	
framework	OpenCL	OpenCL	
processing pipelines	32	32	pix/s
ALUTs utilization	**45%**	**40%**	
RAMs utilization	60%	25%	fp32, 8k bins
frequency / ideal (MHz)	**205** / 240	**360** / 480	
host-to-device bandwidth	4.7 GB/s	5.6 GB/s	x8 PCIe Gen3, can handle 4.5M x 500 Hz ✅
processing (virtual) pixel rate	**5.7 Gpix/s***	**8.9 Gpix/s**	allows pixel-splitting = 3 ✅



Credit: Bittware

*comparable to NVIDIA V100 (~ 6 Gpix/s, 12 nm process)



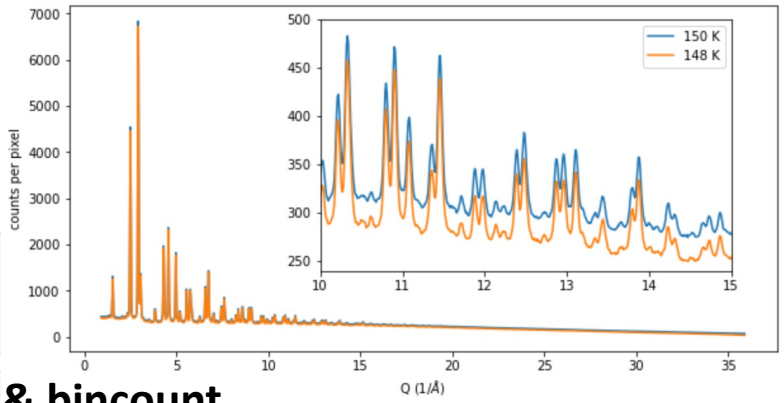
OpenCL

FPGA DEVCLOUD

CONDA

papermill

jupyter



Open-source license

AZINT & bincount

gitlab.com/MAXIV-SCISW/compute-fpgas/bincount

```
[azint-cl] OpenCL will use AOCX binary image: ./build/pac_a10/bincount2_intel.aocx
fpga integration (n=734), exec. time: 7.605 sec
fpga bincout-intel (16.777 Mpix): 96.5 fps, 1619.2 Mpix/s
[168. 178. 189. 192. 200.]
```

FPGA result in 7.6 sec

Conclusions

In this notebook it was demonstrated that

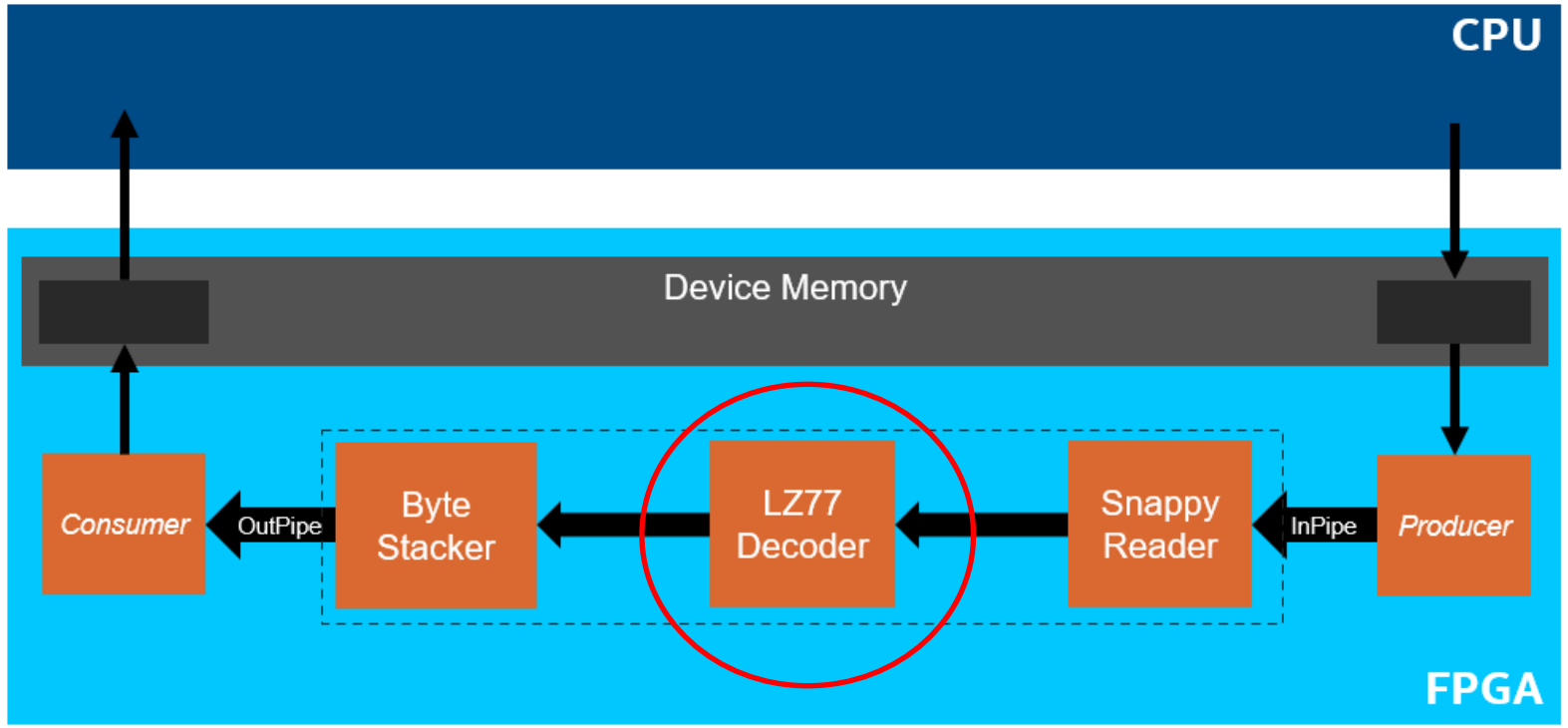
- azimuthal integration for powder diffraction data can be done with *bincount*
- *bincount* procedure can be evaluated effectively at FPGA
  - mainly metadata describing the geometry of the experiment are needed
  - the procedure can effectively orchestrated from Python
- FPGA based data crunching can be done with Jupyter notebooks in Cloud-like environement
- notebooks can be executed in particular on DeviceCloud with help of *papermill*

```
papermill examples/DyCo2.ipynb DyCo2-result.ipynb -k bincount-jupdemo -p PYOPENCL_CTX 0 -p board_name pac_a10
```

Decompression on FPGAs with OpenCL

bitshuffle & lz4

oneAPI-samples / DirectProgramming / C++SYCL_FPGA / ReferenceDesigns / **decompress** /

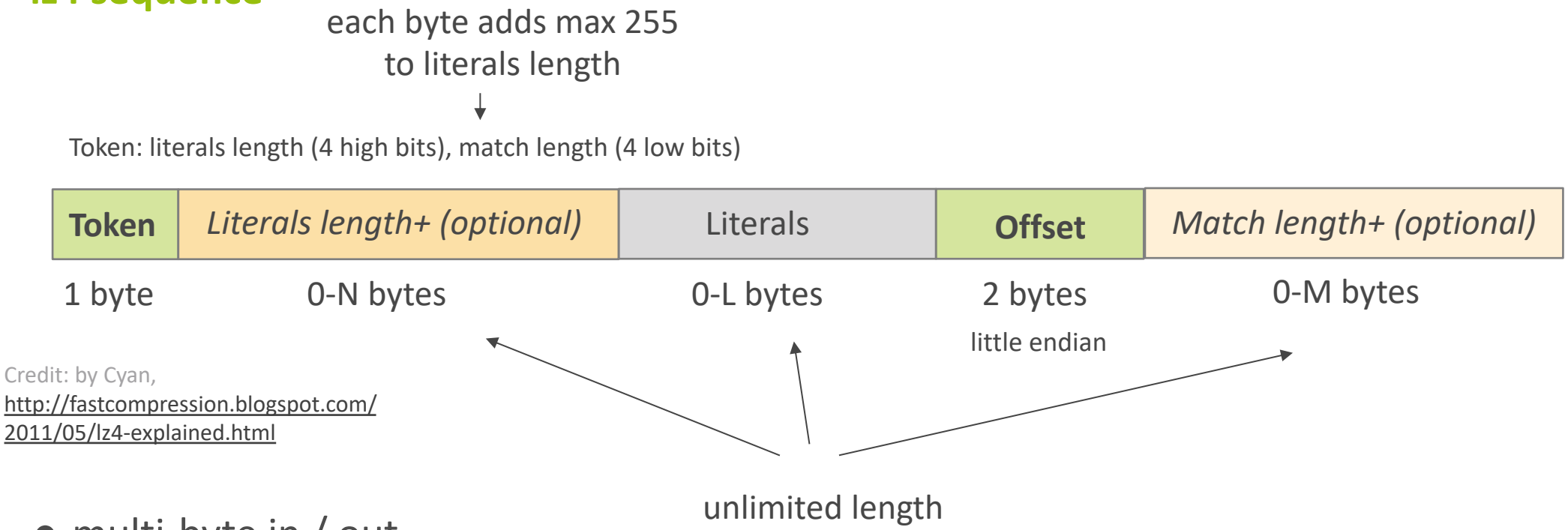


Credit: oneAPI-sample at GitHub.com

- key component: **LZ77 decoder**
  - history buffer
  - multi-byte in / out
- oneAPI -> OpenCL retrofitting done in few hours :-)
- Snappy reader being replaced by lz4 reader
- bitshuffle stage needs to be added

lz4 reader

lz4 sequence



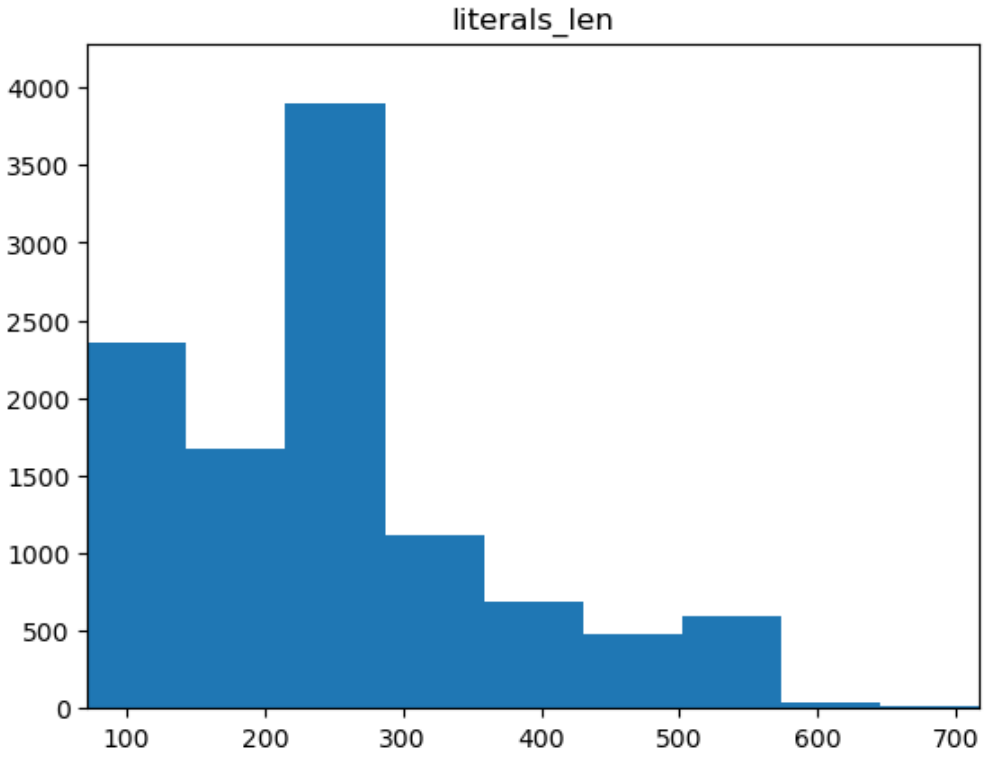
Credit: by Cyan,
http://fastcompression.blogspot.com/2011/05/lz4-explained.html

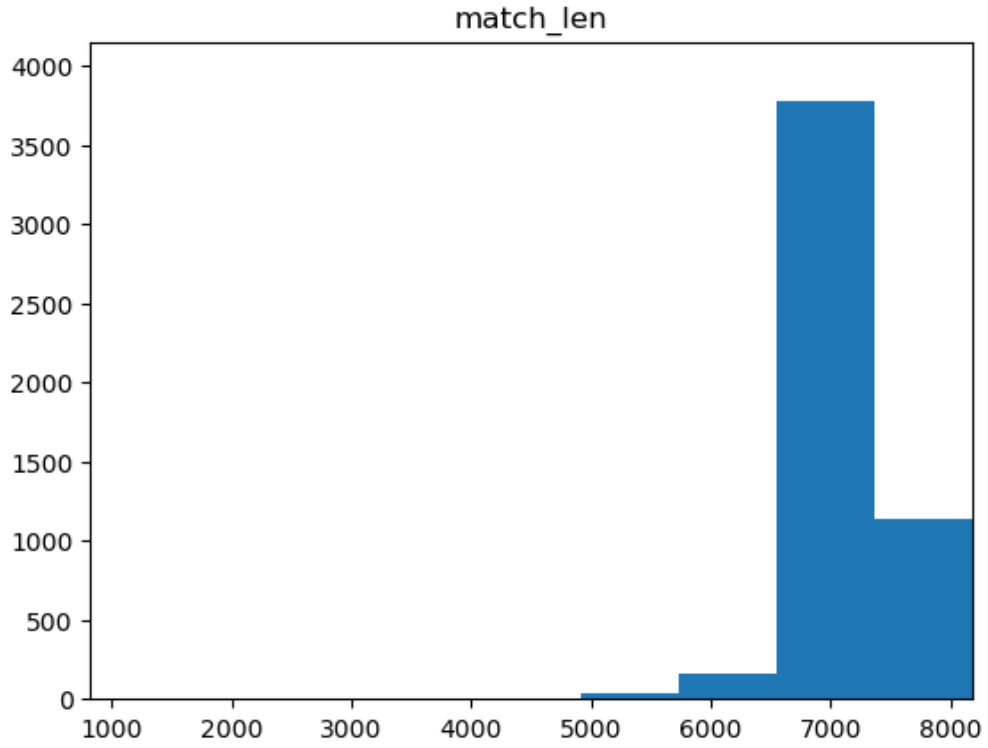
- multi-byte in / out
- out: LZ77 instructions
  - multiple literals
  - single match-copy operation

lz4 – statistics

bitshuffle & lz4 in pure Python and numpy

©Dectris Eiger, 32 bit, 16 Mpix, MX experiment

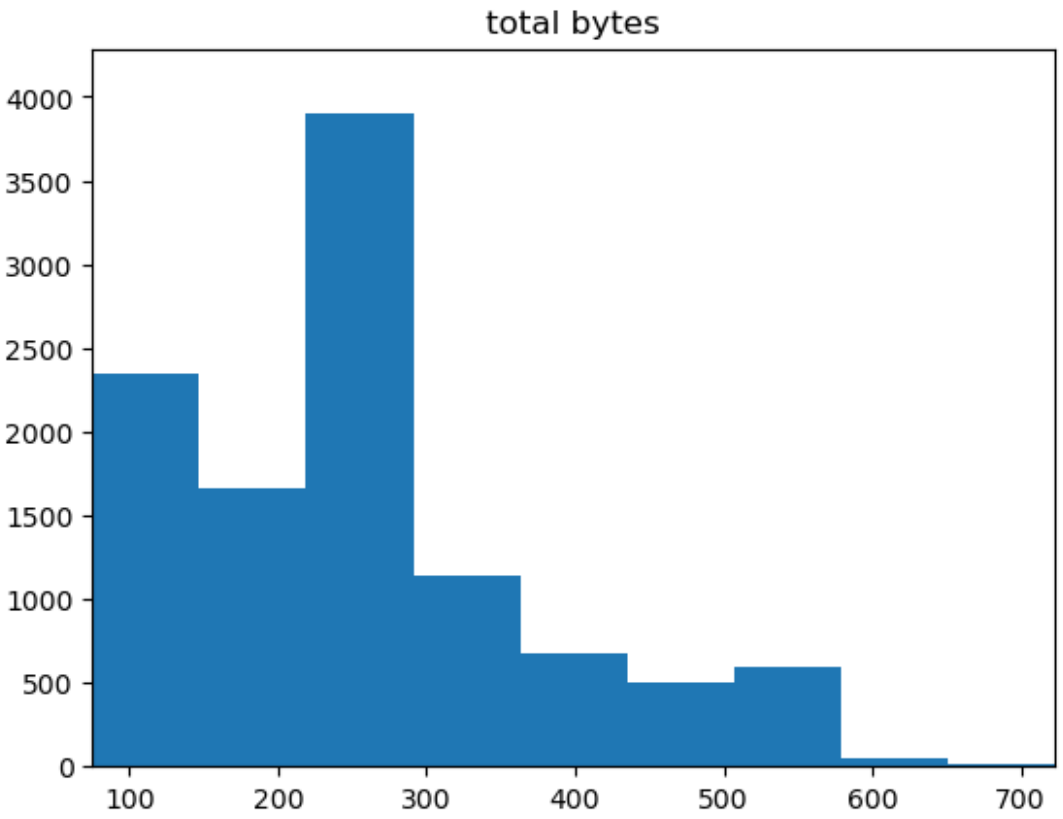




Note: each byte adds **only max 255** to length

lz4 – statistics

bitshuffle & lz4 in pure Python and numpy



85.0 percentile is 5.0

85.4% entries have max len 5 bytes

82.2% entries have max len **4 bytes**

33.8% entries have max len **3 bytes**

- e.g. 3 bytes are pure match-copy ops
- optimizations
  - speculate 3-5 bytes sequences

Summary

- **we can do data processing like AZINT on FPGAs with the standard scientific precision**
- **we use direct chunk read to get uncompressed data**
- **having decompression on FPGAs is close**
- AZINT & bincount & bslz4: gitlab.com/MAXIV-SCISW/compute-fpgas

This project has received funding from the European Union´s Horizon 2020 research and innovation programme under grant agreement no. 101004728

MAX IV Laboratory, Lund
Artur Barczyk
Andrii Salnikov
Clemens Weninger

Niels Bohr Institute, Copenhagen
Kenneth Skovhede
Carl Johnsen
Brian Vinter
Mads Jörgensen

Thank you for your attention