Real-Time Azimuthal Integration of X-Ray Scattering Data on FPGAs

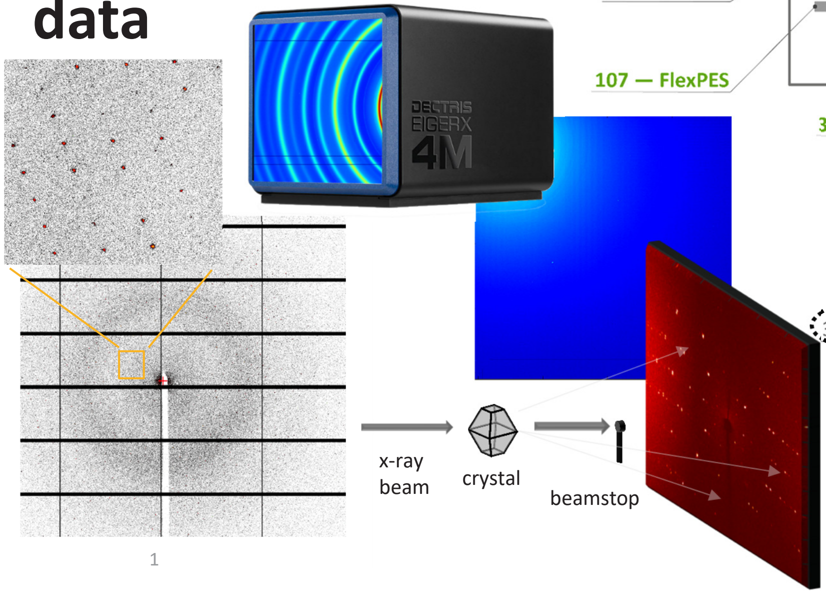
Zdenek Matej [1], *Kenneth Skovhede* [1,2], *Carl Johnsen* [2], *Artur Barczyk* [1], *Andrii Salnikov* [1], *Clemens Weninger* [1] *and Brian Vinter* [2]

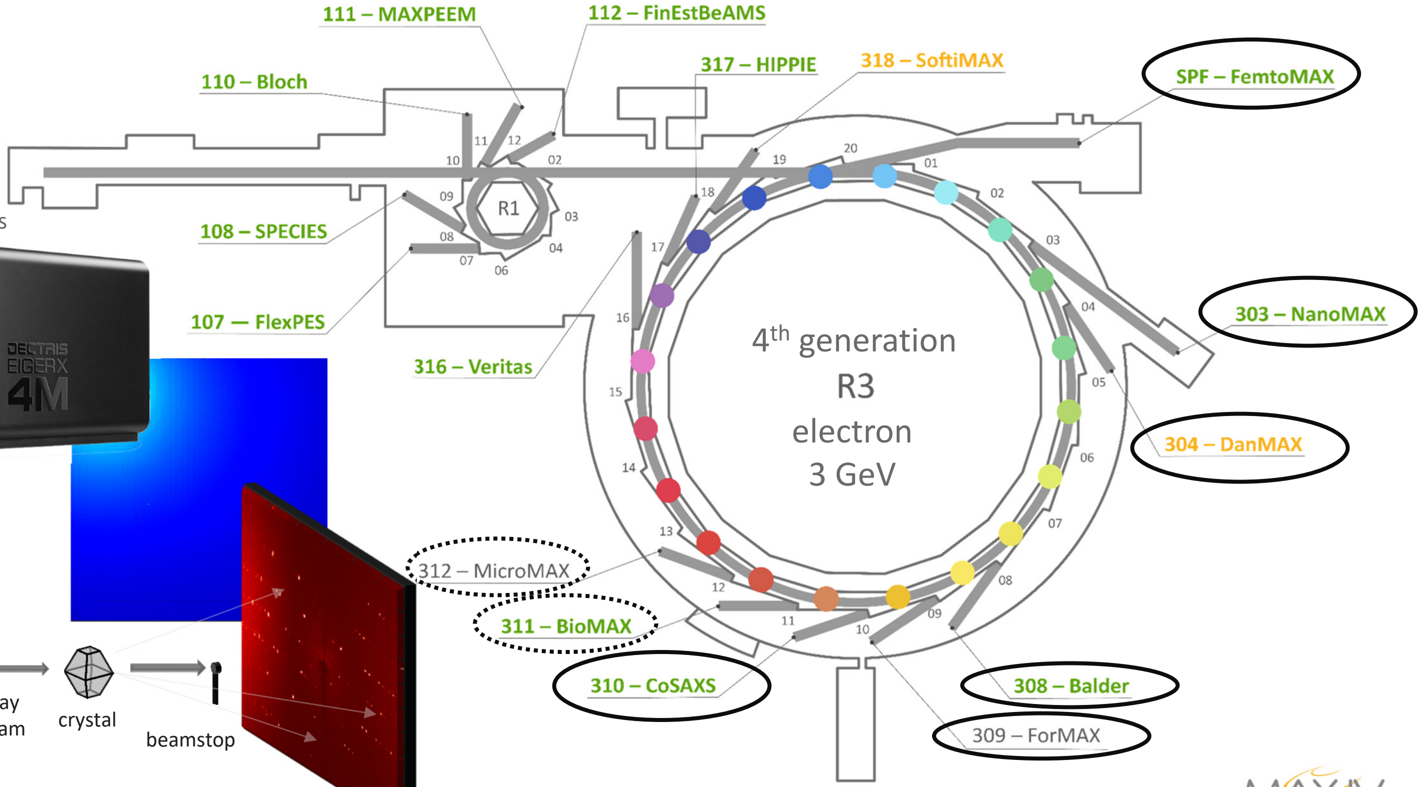
[1] **MAX IV Laboratory, Lund, Sweden**
[2] **Niels Bohr Institute, København, Denmark**

MAX IV synchrotron lab

Scattering detector data

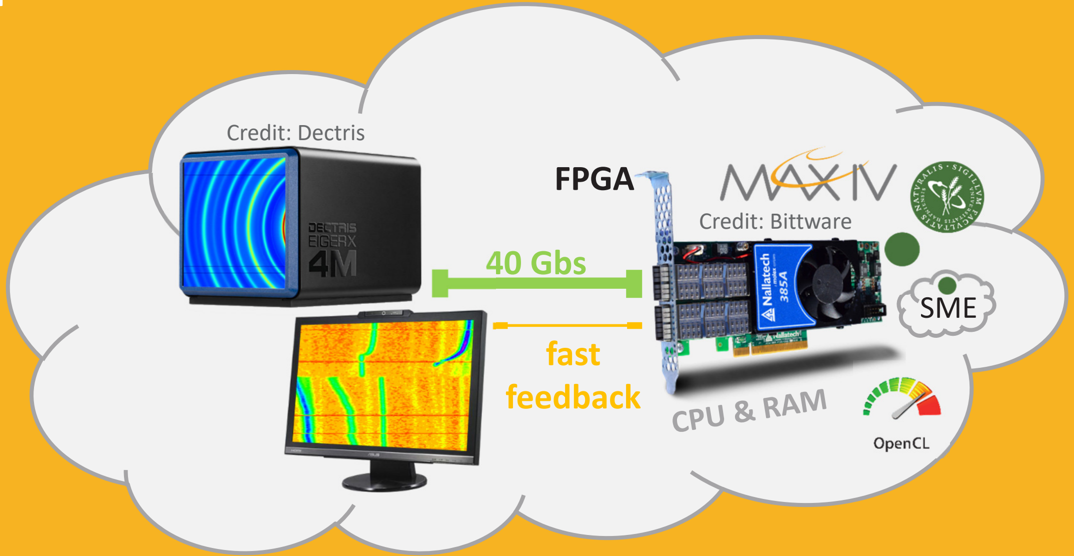
Credit: Dectris



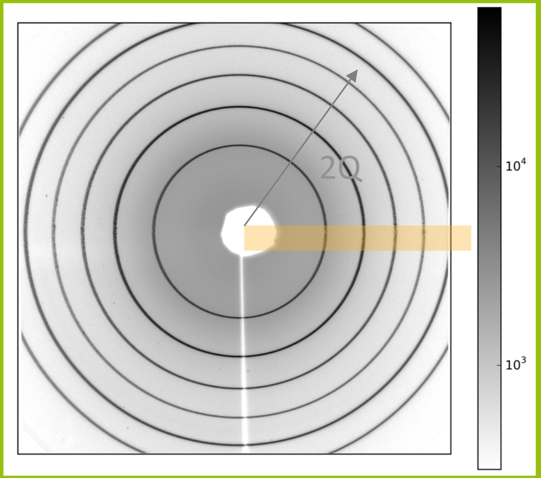


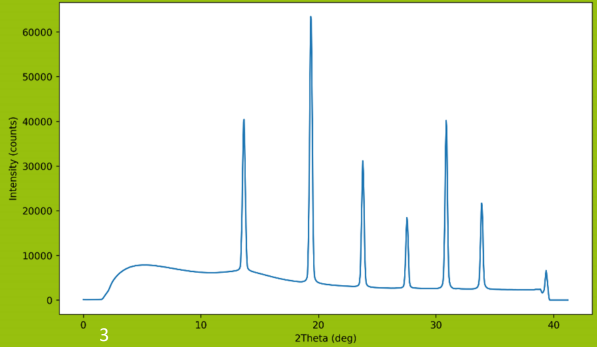
Outline

- intro to azimuthal integration
  - physics and computing
  - motivation
  - existing solutions
- basic algorithm on FPGAs
- tools we are using
- few technicalities
- performance figures
- next and future steps
- references

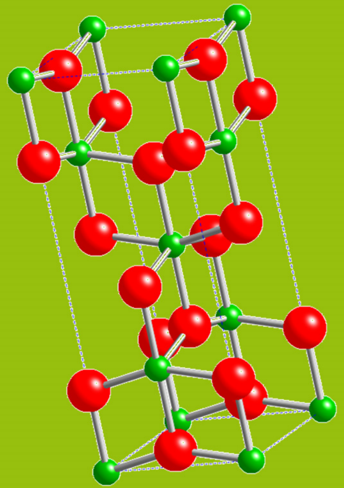


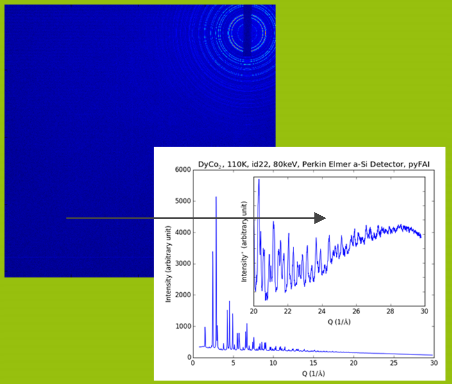
Azimuthal integration



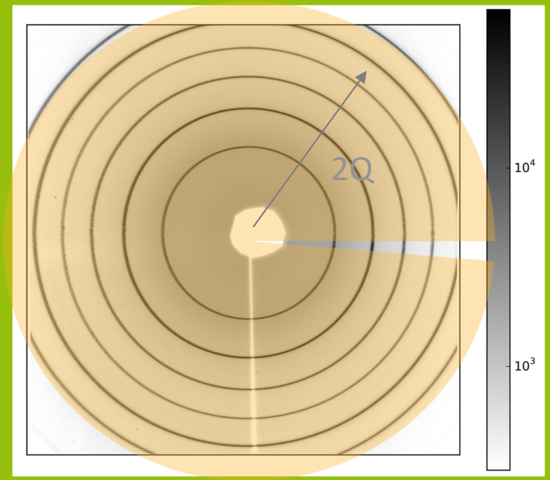


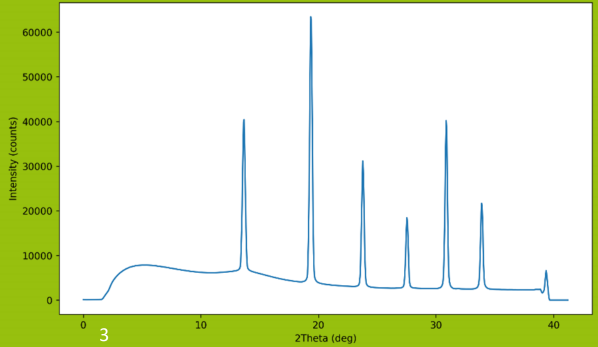
- geometry: scattering angle 2Q for each pixel
  - peak position – accuracy (lattice params., strain)
  - peak width & shape – resolution
- physics: intensity corrections
  - peak intensities – atom positions
  - preferred orientation of crystallites
  - overall intensity normalisation
- experimental:
  - oversampling – resolution (pixel-splitting)
  - error estimates for experimental data
- technical:
  - very wide dynamical range (>> 10^9)
  - data compression
  - high data rates
    - nowadays 1 Mpix cameras can run on 3 kHz -> 3 Gpix/s



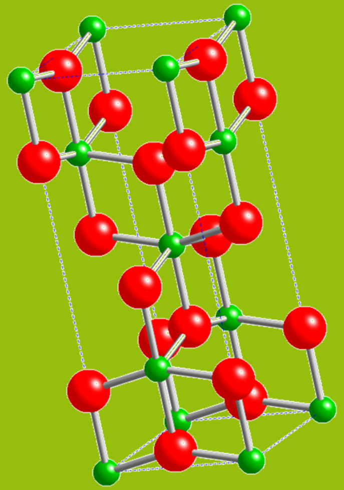


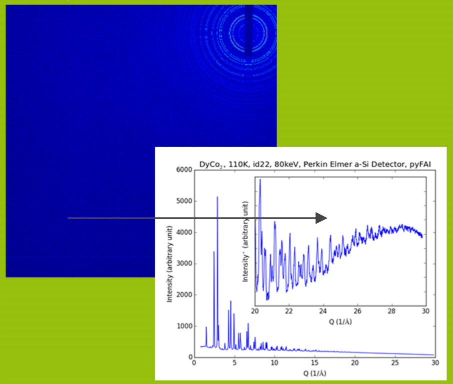
Azimuthal integration





- geometry: scattering angle 2Q for each pixel
  - peak position – accuracy (lattice params., strain)
  - peak width & shape – resolution
- physics: intensity corrections
  - peak intensities – atom positions
  - preferred orientation of crystallites
  - overall intensity normalisation
- experimental:
  - oversampling – resolution (pixel-splitting)
  - error estimates for experimental data
- technical:
  - very wide dynamical range (>> 10^9)
  - data compression
  - high data rates
    - nowadays 1 Mpix cameras can run on 3 kHz -> 3 Gpix/s

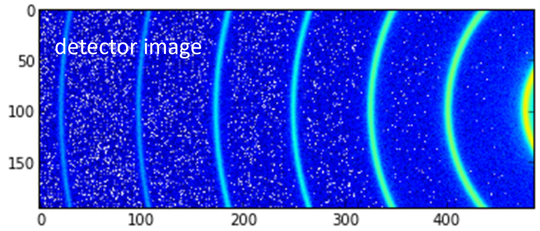


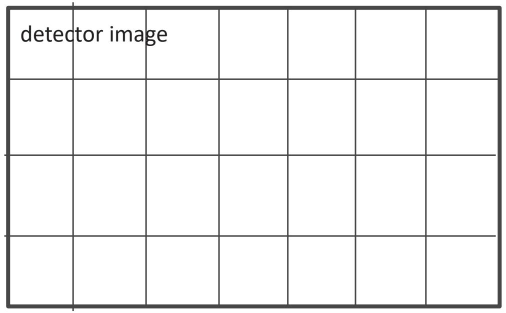


Azimuthal integration

procedure

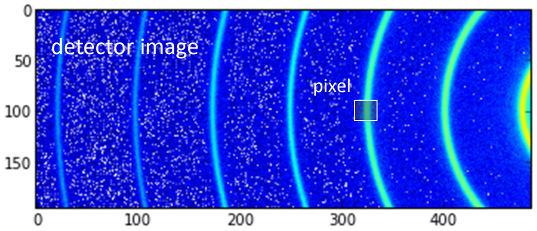
- 2D (~1M pix) -> 1D (~10k bins)
  data reduction factor (100 - 1000x)
- for each image pixel (n,m):





Azimuthal integration

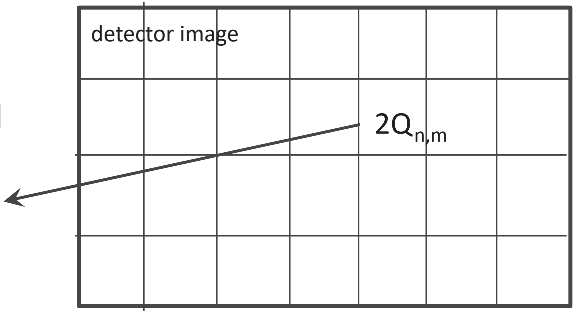
procedure



- 2D (~1M pix) -> 1D (~10k bins)
  data reduction factor (100 - 1000x)

(m,n) -> icol

- for each image pixel (n,m):
  1. trigonometric calc. -> scattering angle ($2Q_{n,m}$)

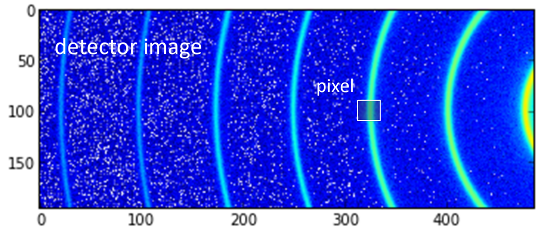


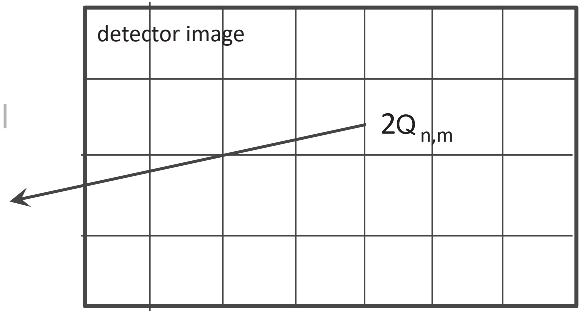
Azimuthal integration

procedure

- 2D (~1M pix) -> 1D (~10k bins)
  data reduction factor (100 - 1000x)
- for each image pixel (n,m):
  1. trigonometric calc. -> scattering angle ($2Q_{n,m}$)
  2. binning ($2Q_{n,m}$) -> single index (irow)

(m,n) -> icol





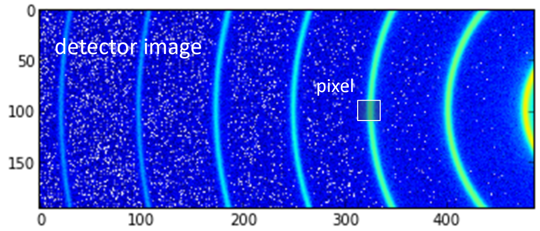
Azimuthal integration

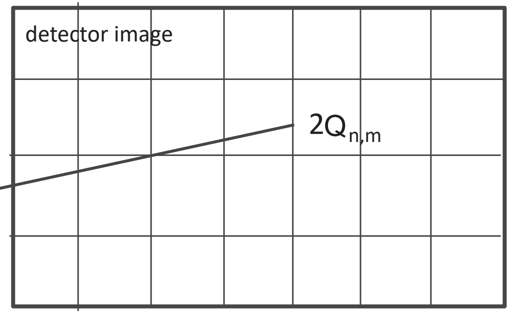
procedure

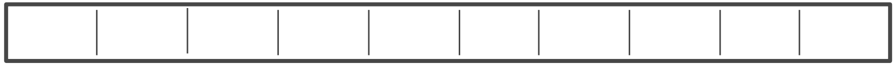
- 2D (~1M pix) -> 1D (~10k bins)
  data reduction factor (100 - 1000x)

- for each image pixel (n,m):
  1. trigonometric calc. -> scattering angle ($2Q_{n,m}$)
  2. binning ($2Q_{n,m}$) -> single index (irow)
  3. intensity corrections

(m,n) -> icol



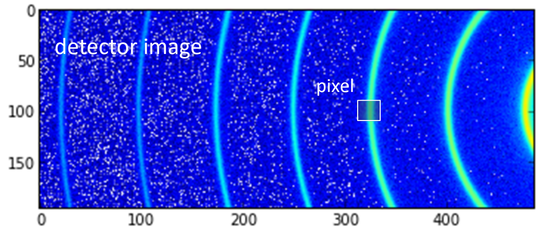


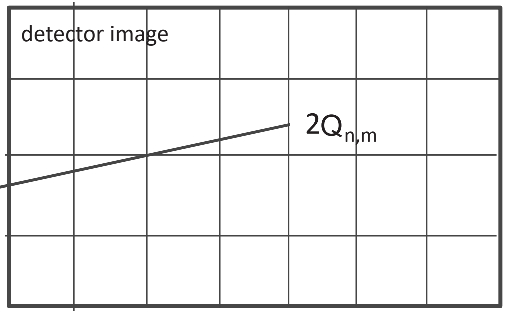


$2Q_{min}$ $2Q_{max}$

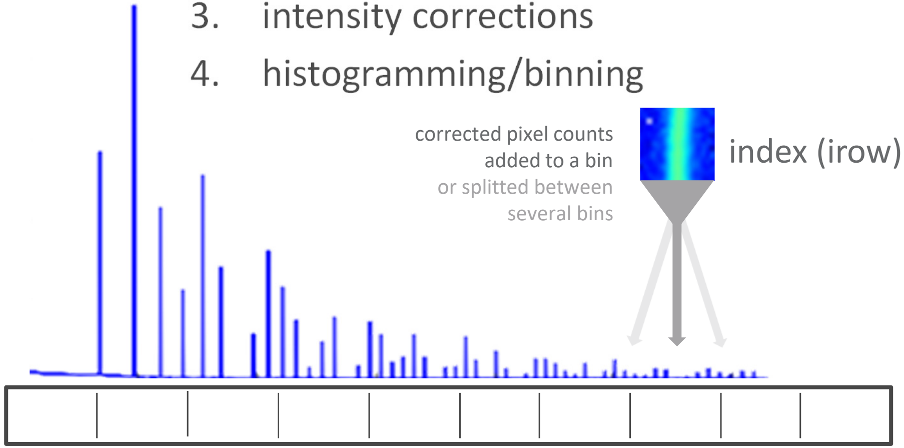
Azimuthal integration

procedure





- 2D (~1M pix) -> 1D (~10k bins)
  data reduction factor (100 - 1000x)
- for each image pixel (n,m):
  1. trigonometric calc. -> scattering angle ($2Q_{n,m}$)
  2. binning ($2Q_{n,m}$) -> single index (irow)
  3. intensity corrections
  4. histogramming/binning

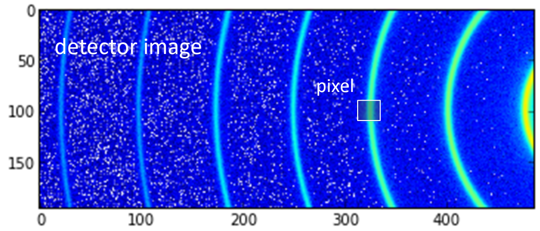


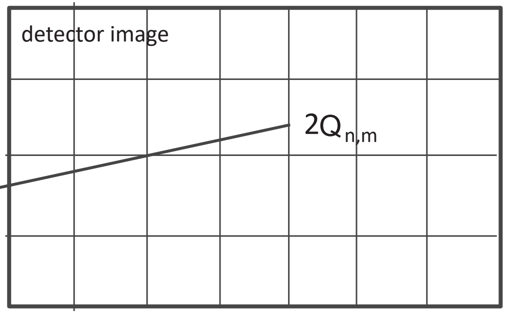
Azimuthal integration

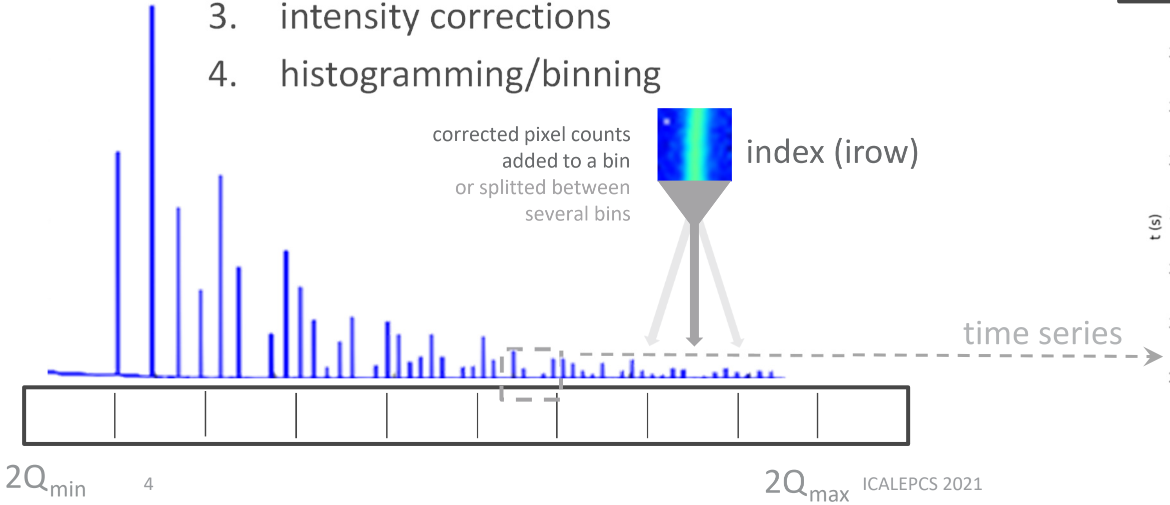
procedure

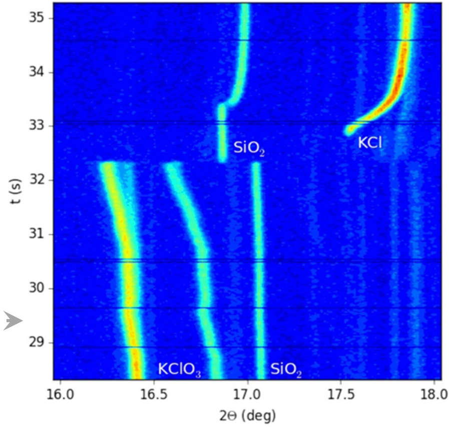
- 2D (~1M pix) -> 1D (~10k bins)
  data reduction factor (100 - 1000x)
- for each image pixel (n,m):
  1. trigonometric calc. -> scattering angle ($2Q_{n,m}$)
  2. binning ($2Q_{n,m}$) -> single index (irow)
  3. intensity corrections
  4. histogramming/binning

(m,n) -> icol



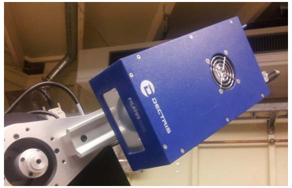






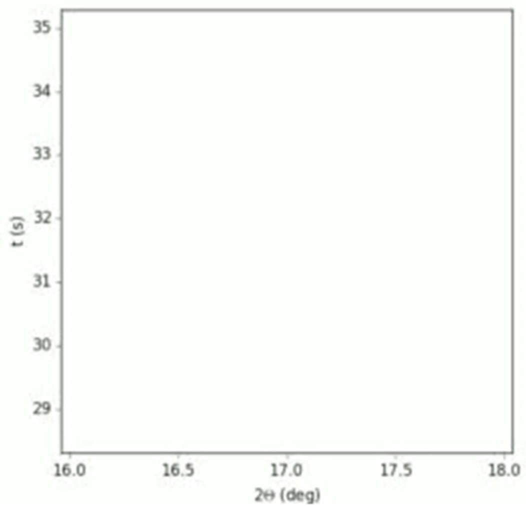
Motivation - fast experiments

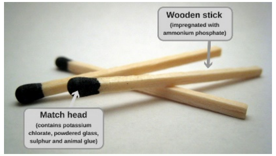
thermal decomposition of potassium chlorate



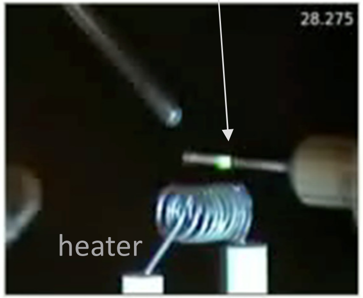
Pilatus detector

$2\ KClO_3$ -> $2\ KCl + 3\ 0_2$





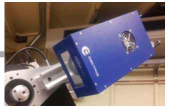
material of safety match heads filled in the capillary

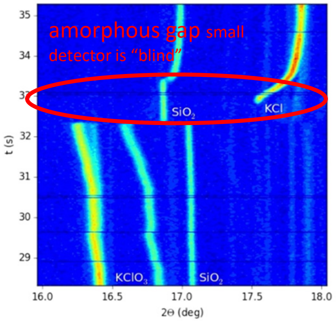


sample

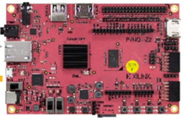
Real-time x-ray probe as trigger for slower detectors

fast but small detector with limited Q-rage





Credit: TUL



large but slow

Simple methods for AZINT calculation

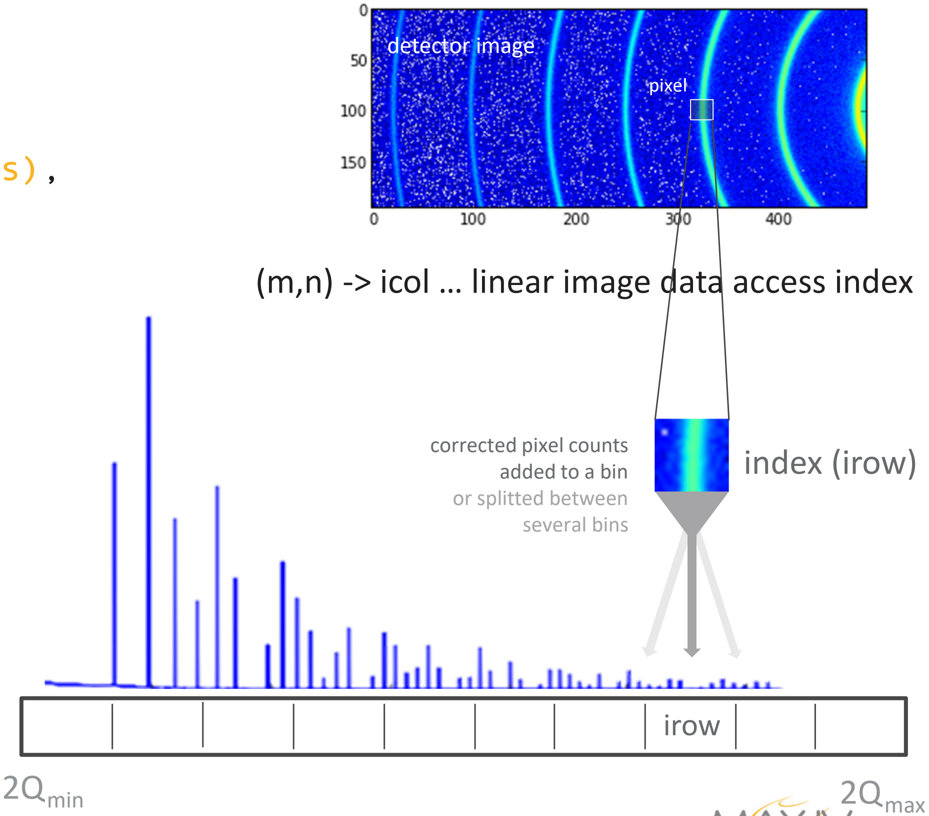
split mathematics and computation

A. bincout

- ```
  bin_dta = numpy.bincount(
        x=irows, weights = imgx * (cors * wgts),
        minlength=nbins)
  ```
- ```
  ncs_dta = numpy.bincount(
        x=irows, weights = wgts,
        minlength=nbins) # normalisation
  ```
- only approximative error estimation

B. sparse matrix multiplication

nnz. ~ num_pix * split_pix ~ 1M

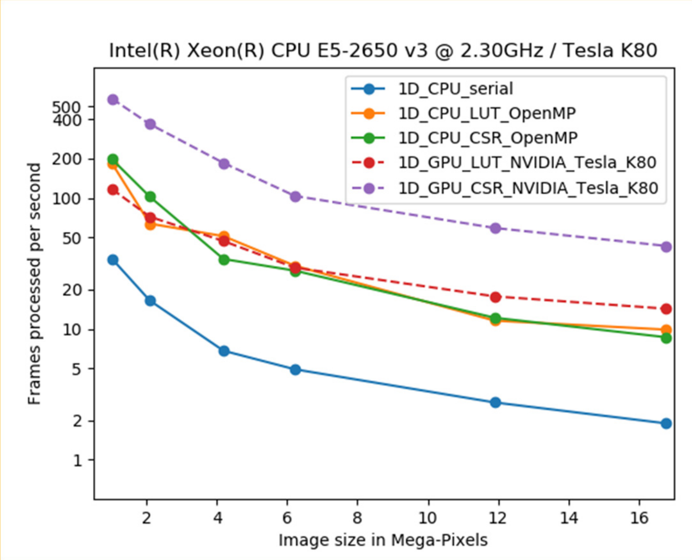


Existing implementations

CPUs and GPUs

- CPUs
  - **Fit2D** *(ESRF)*
  - **diffpy.srxplanar** *(APS)*
  - **Nika** *(APS)*
  - **PyFAI** *(ESRF)*
  - **matFRAIA** *(Aarhus University & MAX IV)*
  - . . .
- GPUs
  - (OpenCL) **PyFAI** *(ESRF)*

pyFAI - benchmark

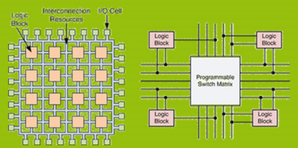


architecture	AZINT rate (Gpix/s)
1 cpu core	0.06
K80 gpu	0.8 – 2
P100, V100 gpu	2.8 – 6

MAX IV

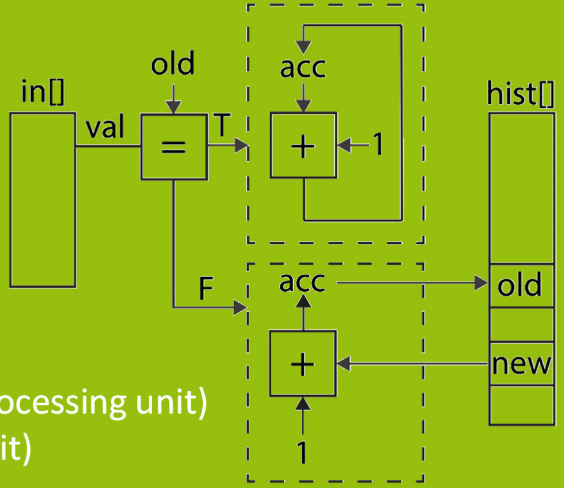
Basic algorithm on FPGAs

bincount - first implementation



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

- by *Carl Johansen* (NBI) using Synchronous Message Exchange (SME) – in 1 day
- initially only for integers
- **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)
- performance numbers:
  - small FPGA (Xilinx Zynq Z7020) at 100 MHz: 1 Gpix/s (10% util. per processing unit)
  - Large FPGA (Xilinx Ultrascale+) at 590 MHz: 20 Gpix/s (3% util. per unit)
- ref: github.com/bh107/SME-Binning



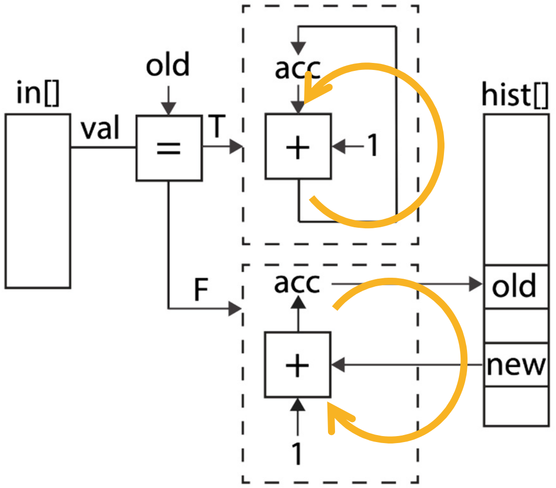
Credit: *The HLS book

- similar to textbook example: PREFIX SUM AND HISTOGRAM in *Kastner, Matai, Neuendorffer*: *Parallel Programming for FPGAs - the HLS book, kastner.ucsd.edu/hlsbook

Basic algorithm on FPGAs

bincount – with floating point data

- large dynamic range of the result: 1000x original image
- floating point corrections
- **floating point operations last multiple cycles**
  - intensity corrections ✅
  - adder operation on local memory (BRAM) ❌



Credit: *The HLS book

Example: 4 cycles for the add operation

A)	0: 10	1: 11	2: 12	3: 13	0: 10	1: 11	2: 12	3: 13	->	0: 20	1: 22	2: 24	3: 26	✅	
B)	0: 10	1: 11	0: 10	1: 11	2: 12	3: 13	2: 12	3: 13	->	0: 10	1: 11	2: 12	3: 13	❌	sort ?
C)	0: 10	0: 11	0: 12	0: 13	0: 14	0: 15	0: 16	0: 17	->	0: 30	1: 0	2: 0	3: 0	❌	merge ?

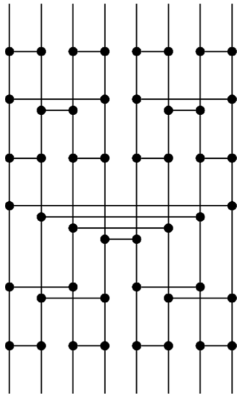
(B: distance between repeated bin 0 entries: 2 < 4)

MAX IV

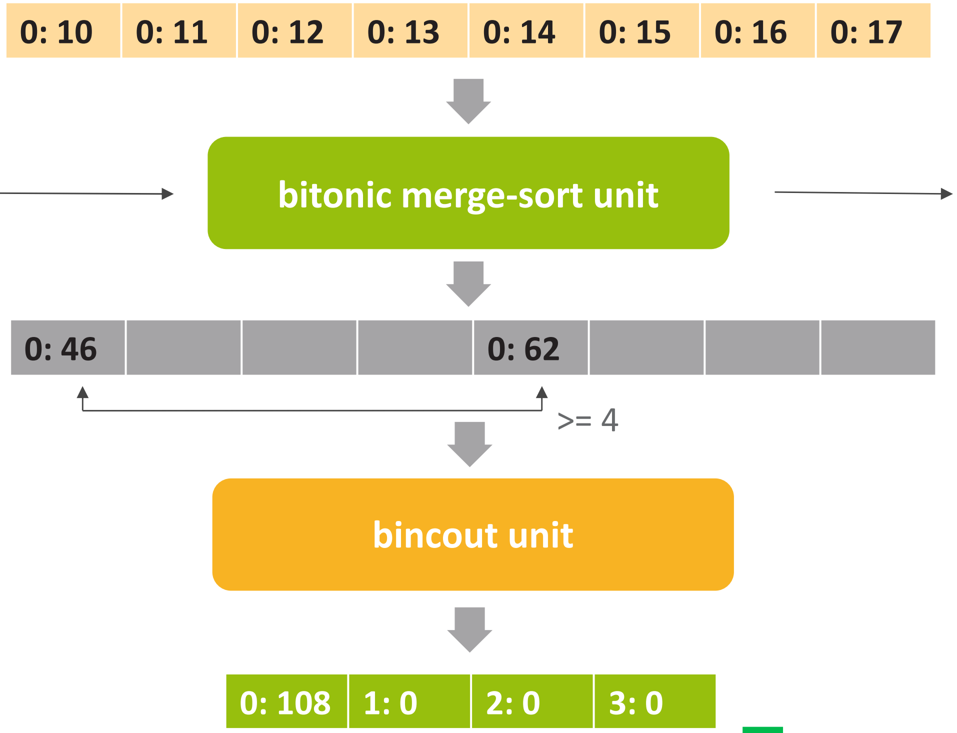
Basic algorithm on FPGAs

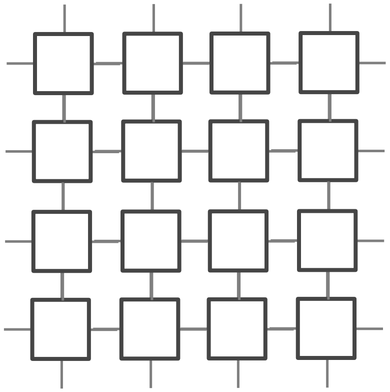
bincount + bisort & resort for floating point data

Bitonic sorter



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





can be expressed as systolic array

very suitable for FPGA

bincount on FPGAs

tools

- highl level tools
  - **Synchronous Message Exchange** (SME): github.com/kenkendk/sme
    - *Kenneth Skovhede* & *Carl Johansen* (Niels Bohr Institute)
    - C# -> SMEIL (SME intermediate language) -> VHDL -> bitstream
    - vendor agnostic: from (100$) Zynq SoC for PYNQ to large Ultrascale+ or Stratix-10 boards
  - **OpenCL**
    - Intel (former Altera) & Bittware boards with Aria-10, Stratix-10
    - in production at MAX IV
  - Xilinx **High Level Synthesis** (HLS)
    - "C with pragmas"
    - collaboration with PSI for JungFRAU detector
    - not AZINT & bincount yet
- Python - orchestration on host, tests

language: OpenCL/C
code is hw agnostic
bitonic sort: 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);
```

bincount on FPGAs

few technicalities: memory bandwidth

- raw image data: 2 bytes / per clock cycle / pipeline
  - source
    - A. on-board serial or Ethernet I/O channels
    - B. host or on-board (DDR) memory (typically 10 MB per image)
- control (metadata) - sparse matrix data
  - target bin position (irow): 2 bytes
  - pixel index (icol): 4 bytes
  - correction: 4 bytes (float)
  - weights: 4 bytes (float)
  - source
    - A. calculated on on-the-fly on FPGA (limitations)
    - B. on-board (DDR) memory (typically 100 MB per image)
  - identical for multiple images (in the most important application cases)

2 bytes data
vs. 14 bytes metadata
!!!!

solutions

A. calculate geometry and corrections on-the-fly
B. hw with many (>=32) memory channels, e.g. expensive HBM2 chips
C. process multiple images simultaneously

Performance figures

AZINT bincount – on Intel FPGAs with OpenCL

	385A	520N-MX	comment
size	medium	large	
FPGA	Aria 10 GX	Stratix 10 MX	Bittware / Nallatech
process	20 nm	14 nm	
memory	2xDDR3	2xHBM2	
QSPF	2x10/40 Gbs	4x100 Gbs	
framework	OpenCL	OpenCL	
processing pipelines	32	32	
ALUTs utilization	**45%**	**40%**	
RAMs utilization	60%	25%	fp32 (fp64 possible), 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)

Next

and future steps

- Summary
  - AZINT and bincount are very suitable for FPGAs
  - FPGAs are very “predicable” for data processing (n-pixels per clock cycle)
  - a lot of new literature and code about HLS or OneAPI and exciting hw available
- Next
  - more materials in public repo: gitlab.com/MAXIV-SCISW/compute-fpgas/bincount
  - optimizations for Intel FPGAs: 60% is on f_{max}
- Future
  - HLS for Xilinx FPGAs: both large and small FPGAs including fancy Zynq SoC for PYNQ
  - OneAPI for Intel
  - floating point for SME
  - contributions most welcomed

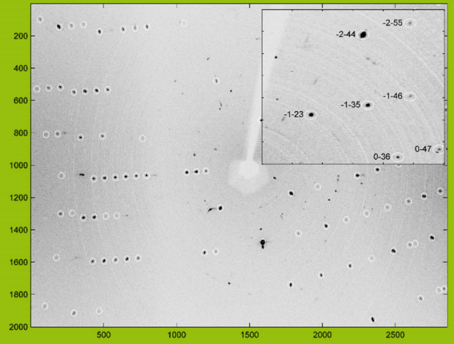
PYNQ

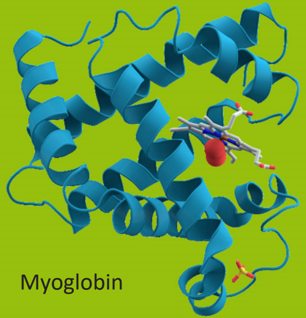
Similar activities

elsewhere

- *Filip Leonarski et al.* (PSI)
  Fast and accurate data collection for macromolecular crystallography using the JUNGFRAU detector
  - IBM OpenCAPI framework & Xilinx Vivado HLS
  - doi: 10.1038/s41592-018-0143-7
- *Maxime Martelli et al.* (CNRS, Paris)
  3D Tomography Back-Projection Parallelization on Intel FPGAs Using OpenCL
  - Intel OpenCL & Aria-10
  - doi: 10.1007/s11265-018-1403-6
- non-accelerator example:
  - X-ray detectors at Cubesat nanosatellites

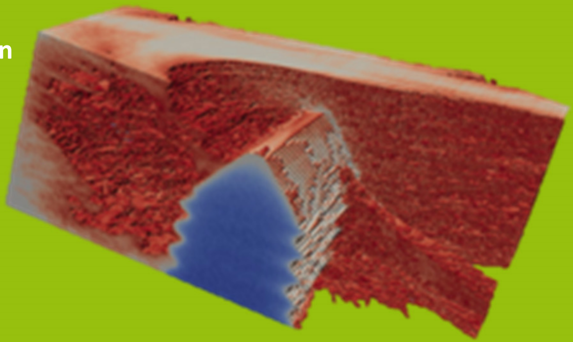
Spot-finding





Myoglobin

Tomography reconstruction

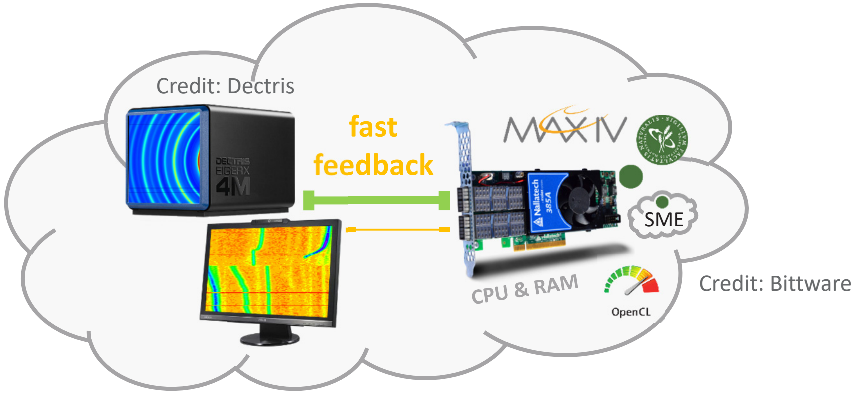


X-ray detector at nanosatellite

Credit: vzlusat2.cz and
Rigaku Innovative Technologies Europe

References

- Synchronous Message Exchange (SME): github.com/kenkendk/sme
- SME-Binning: github.com/bh107/SME-Binning
- AZINT & bincount: gitlab.com/MAXIV-SCISW/compute-fpgas/bincount
- email: zdenek.matej(a)maxiv.lu.se



Thank you for your attention