Azimuthal integration of area-detector data on field-programmable gate arrays

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

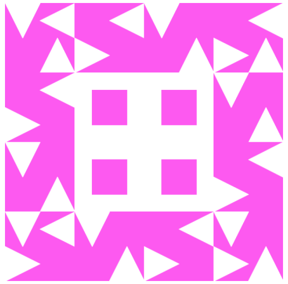
[2] Niels Bohr Institute, København, Denmark

Scientific Computing on FPGAs

How it started at MAX IV

Artur – physicist, CERN, MAX IV network and infrastructure team

Andrii – IT & cloud

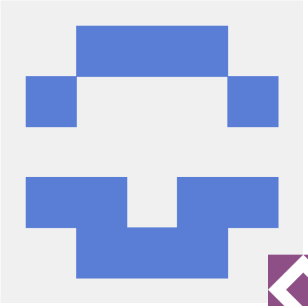


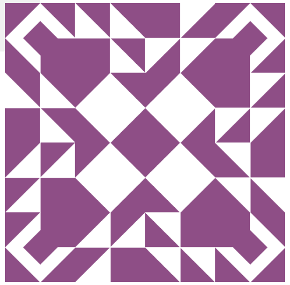
Clemens – MAX IV scientific sw, x-ray scattering sw for CPUs

Zdenek – MAX IV scientific sw, x-ray scattering and imaging

Kenneth – expert in FPGA languages, “inventor” of SME*, assistant prof. at NBI former eSSENCE postdoc at MAX IV

Carl and Brian – both NBI#, Carl finished recently phd in *X-ray imaging applications for food industry* and Brian an e-Science professor at NBI, at Aarhus University now





*SME: Synchronous Message Excange: https://sme-hdl.org
#NBI: Niels Bohr Institute, Copenhagen University

Scientific Computing on FPGAs

Possible applications

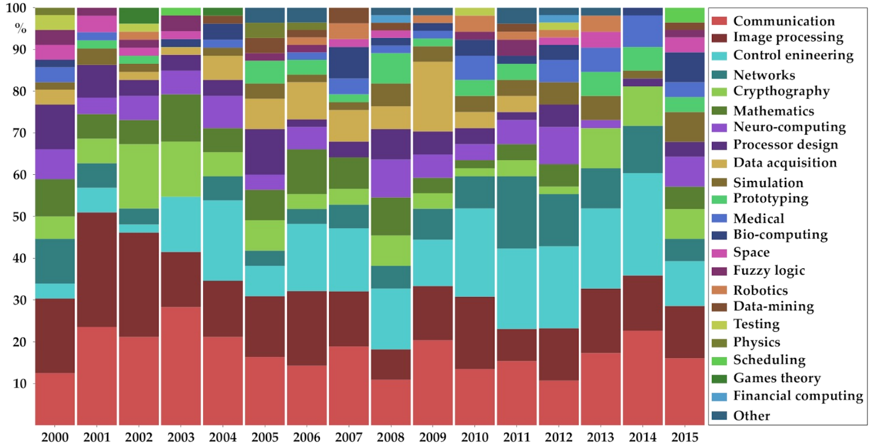
- fast feedback orbit correction
  - real time power spectrum calculation
- image corrections
  - mask
  - flat field
- simple processing
  - image binning
  - roi-counting
- spot finding
- GPU like applications
  - **azimuthal integration**
  - phase retrieval
  - full field tomography reconstructions
- compression
- machine learning (ML) inference
  - image classification with ML & AI
- Frame & event filtering

pros

- real time
- parallel, high performance and throughput
- integrated networking - couple with detectors
- energy effective

cons

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



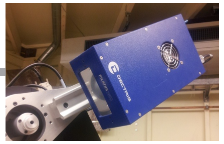
2010–2015: *800 FPGA application reported* by J. Romoth et al. (2017), doi: 10.13140/RG.2.2.16364.56960 + *increasing variety*

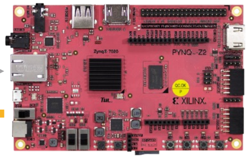
Azimuthal integration on FPGAs

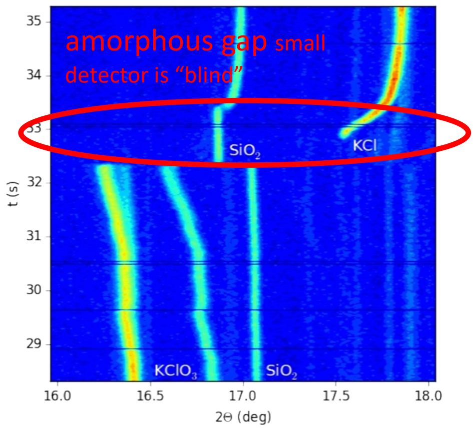
motivation

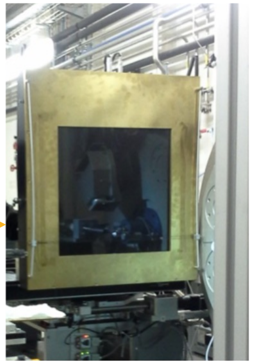
Real-time x-ray probe as trigger for slower detectors

fast but small detector
with limited Q-rage





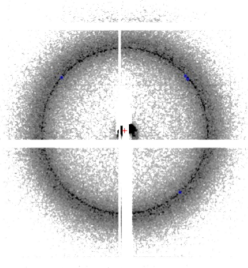




large but slow

binning/histograming/AZINT – important algorithms

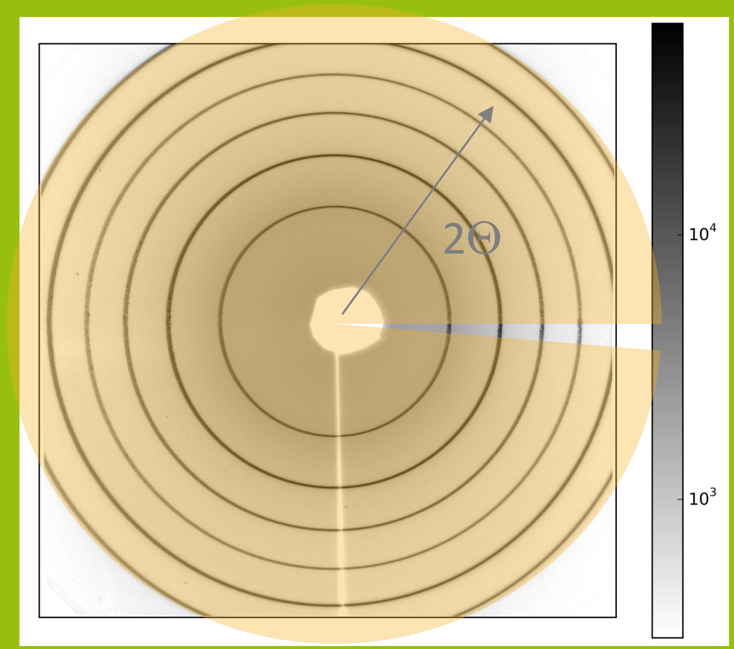
- crystallography:
  - background estimation
  - ice-ring removal
- . . .

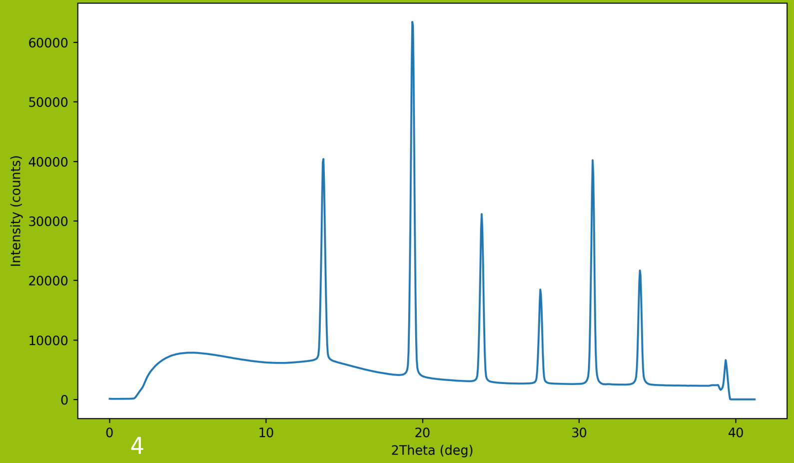


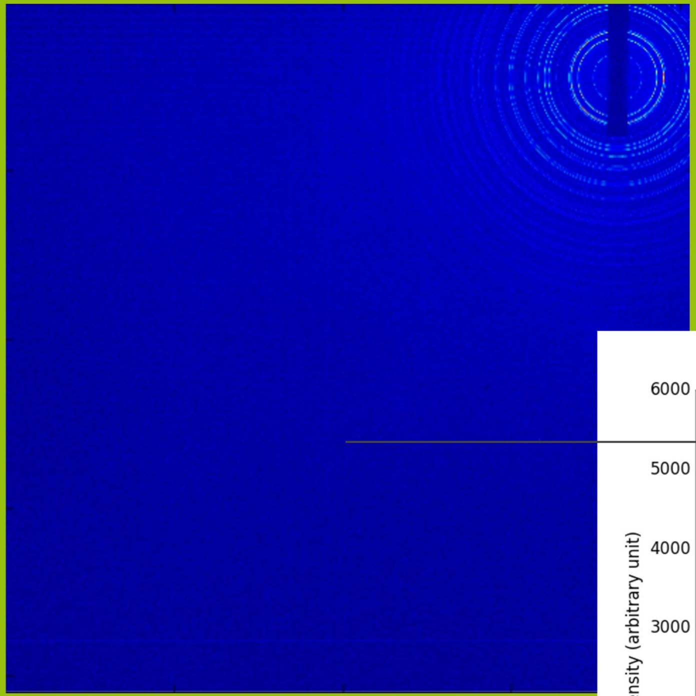
non-synchrotron

- portable instruments
- space applications:
  - limitted bandwidth for data transfer
  - energy efficiency

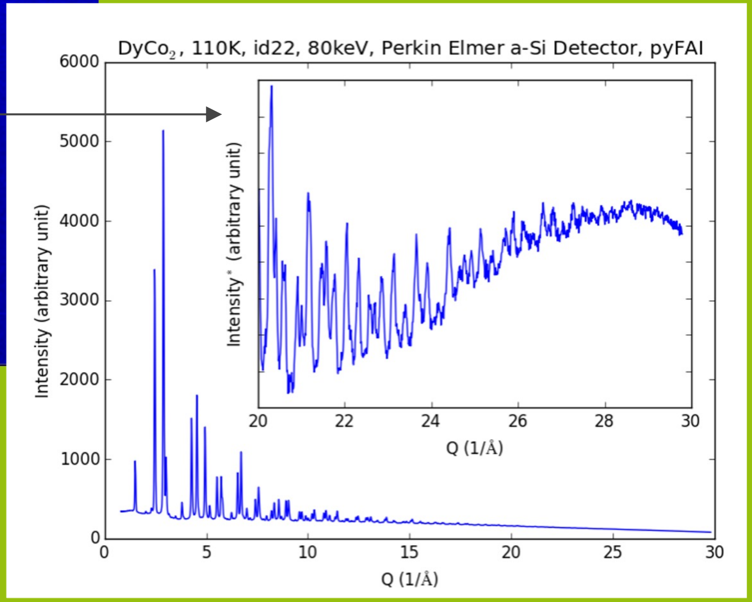
Azimuthal integration







Signal to noise improvement



Data reduction

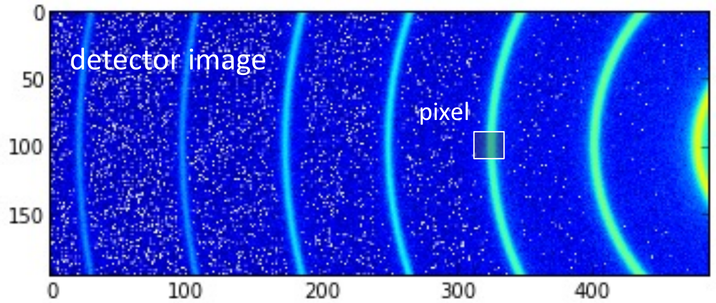
1 Mpix -> 10k bins
factor: 1000x

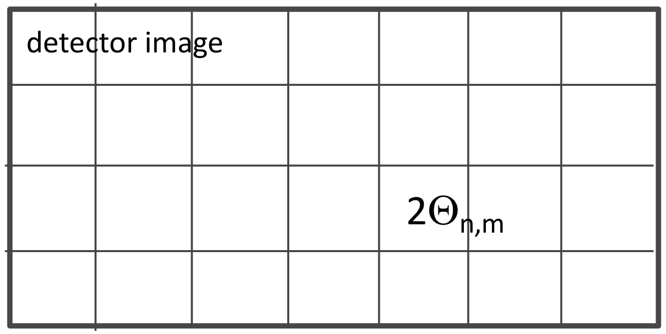
SAXS, XRD, XRD-microscopy, MX calibration

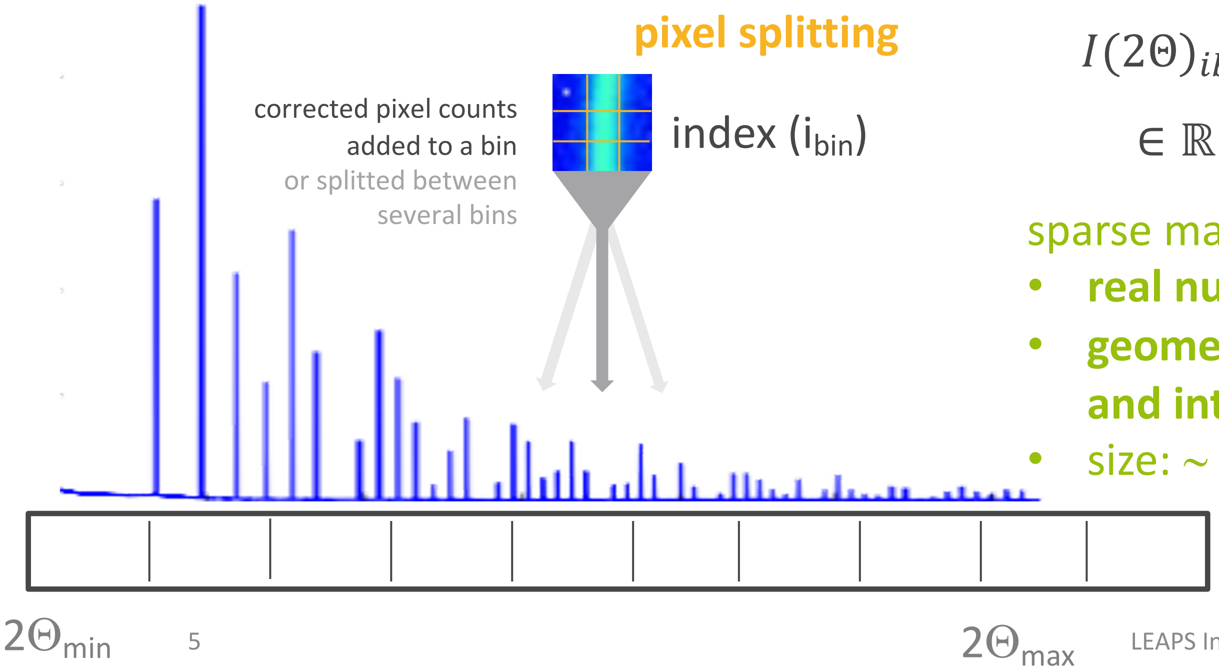
Azimuthal integration

procedure

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







$$I(2\Theta)_{ibin} = \sum_{ipix=0}^{N_{pix}} A_{ibin,ipix} \; Img_{ipix}$$

$\in \mathbb{R}$ $\quad \in \mathbb{R}$ $\quad \in \mathbb{Z}$

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**

fast with FPGA

Existing solutions

CPUs and GPUs

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

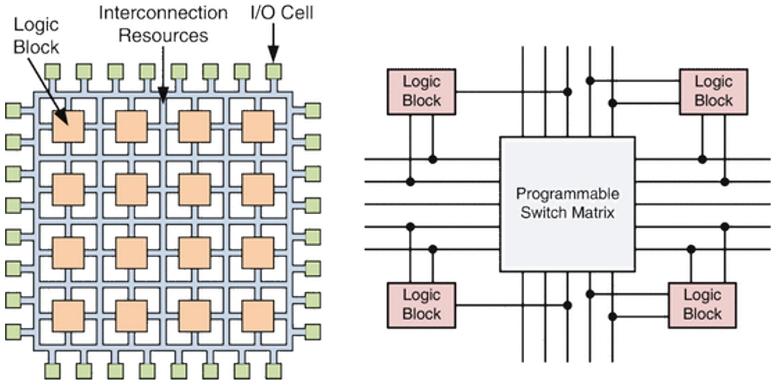
architecture	AZINT rate (Gpix/s)
1 cpu core	0.06
K80 gpu	0.8 – 2
P100, V100 gpu	2.8 – 6
A100 gpu	15
H100 gpu	3x ?

Matlab sparse matrix test

```
>> simple_gpu
0.1383 secs per 500 of 4.0 Mpix (naive GPU)
3614 fps (4.0M), 14457.1 Mpix/s
fx >>
```

Field Programmable Gate Arrays

- arrays of logic gates
  - O(5)M 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
- programming in specialized languages, adapted to electronics development (Verilog, VHDL)
  - IP cores can be purchased from third-party vendors
- ideal for bit-level data manipulation
- FPGA boards: integrated network interfaces

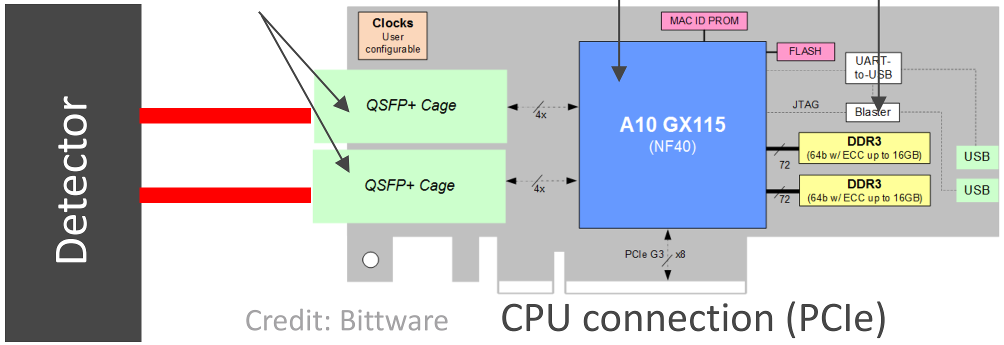


Credit: Kovačec, doi: 10.1007/978-3-319-14346-0_40

network interfaces

FPGA chip

memory



Credit: Bittware

CPU connection (PCIe)

FPGA vs GPU parallelism

Input pixel buffer: | 1 | 2 | 3 | 4 |

a*a + 1

FPGA

space →

time ↓	Load	a*a	a*a+1	Store
t_1	1	-	-	-
t_2	2	1*1	-	-
t_3	3	2*2	1*1+1	-
t_4	4	3*3	2*2+1	2
t_5	-	4*4	3*3+1	5

1 pix/clock

GPU

instruct →	$proc_1$	$proc_2$	$proc_3$	$proc_4$
t_1: Load	1	2	3	4
t_2: a*a	1*1	2*2	3*3	4*4
t_3: a*a+1	1*1+1	2*2+1	3*3+1	4*4+1
t_4: Store	2	5	10	17

4 pix /4 clocks = 1 pix/clock

Programming FPGAs for scientific data processing

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"
    - JungFRAU-Joch by F. Leonarski (PSI)
    - not AZINT & bincount yet
- Python - orchestration on host, tests

OpenCL

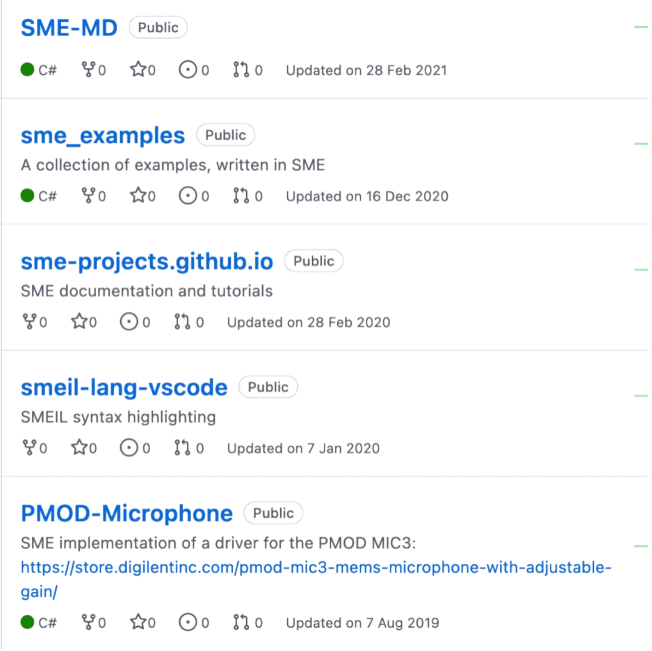
language: OpenCL/C
code is hw agnostic
core: 36 lines
host: PyOpenCL

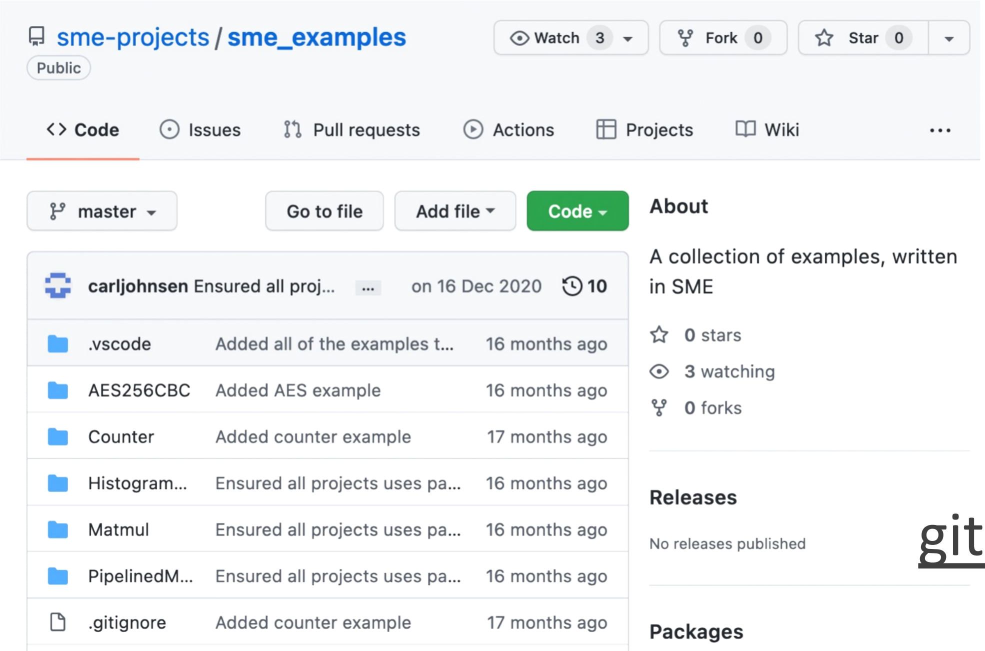
```
/* --- run bitonic merge sort network */
#pragma unroll
for(short row=BN_LEN-1; row>=0; --row) { // program
    #pragma unroll
    for(short col=0; col<BN_N; ++col) {
        // read the program
        const bool ishigh = (bool)_bp[row][col][0];
        if(ishigh) { // only one (high) needs to do 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);
```

Synchronous Message Exchange (SME)

sme-hdl.org

- *Kenneth Skovhede* & *Carl Johansen* (Niels Bohr Institute)
- programming model based on *Communicating sequential processes* (CSP)
- C#/CPP/Python -> SMEIL (SME intermediate language) -> VHDL -> bitstream
- vendor agnostic: from (100$) Zynq SoC to large boards





github.com/sme-projects

Basic algorithm on FPGAs

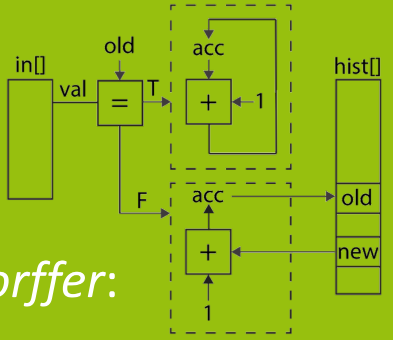
bincount - first implementation

- 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-cycle** per processing unit

```
for(i=0; i<N; i++) {
  pos_new = positions[i];
  val_new = values[i];
  if (pos_new != pos_old) {
    mem[pos_old] = acc;
    acc = mem[pos_new];
  }
  acc += val_new;
}
```

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

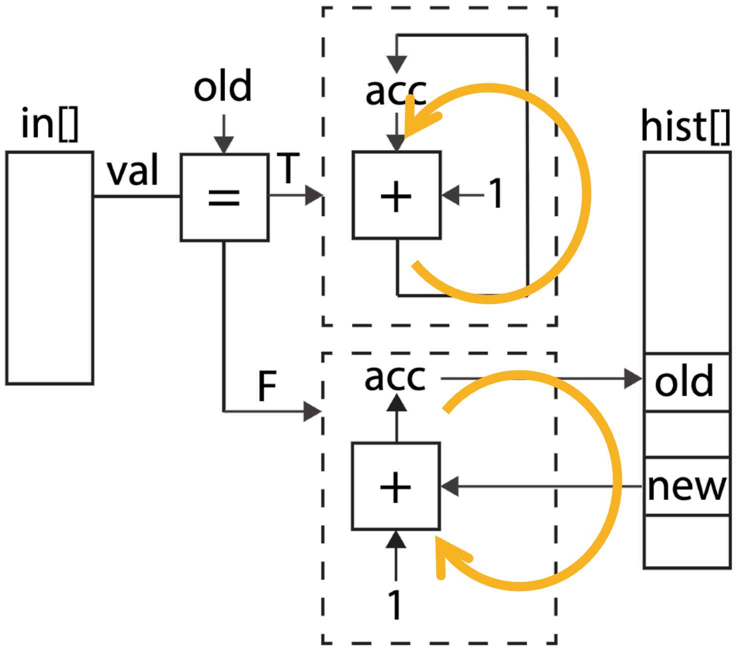
Credit: *The HLS book



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 **pos: val**

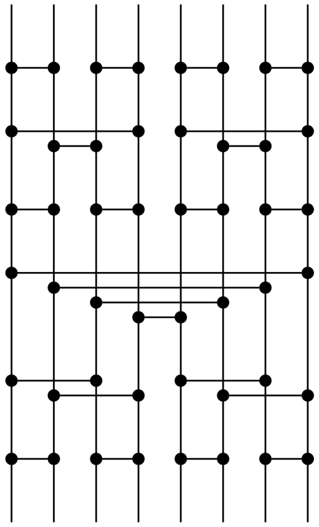
A)	0: 10	1: 11	2: 12	3: 13	0: 10	1: 11	2: 12	3: 13	->	0: 20	1: 22	2: 24	3: 26	✅	
B)	0: 10	1: 11	0: 10	1: 11	2: 12	3: 13	2: 12	3: 13	->	0: 10	1: 11	2: 12	3: 13	❌	sort ?
C)	0: 10	0: 11	0: 12	0: 13	0: 14	0: 15	0: 16	0: 17	->	0: 30	1: 0	2: 0	3: 0	❌	merge ?

2 < 4

Basic algorithm on FPGAs

bincount + bisort & resort for floating point data

Bitonic sorter



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

0: 10	0: 11	0: 12	0: 13	0: 14	0: 15	0: 16	0: 17

bitonic merge-sort unit

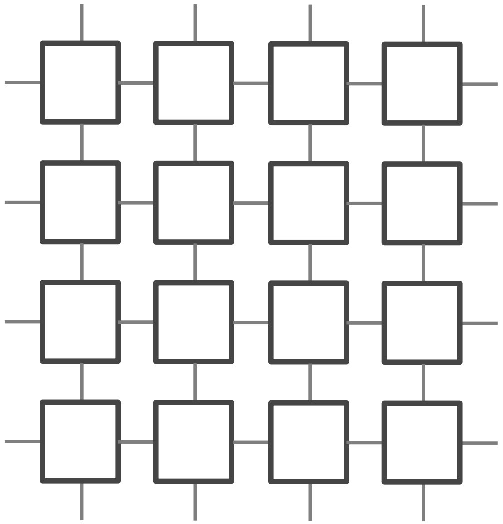
0: 46				0: 62			

>= 4

bincout unit

0: 108	1: 0	2: 0	3: 0

✅



can be expressed as systolic array

very suitable for FPGA

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 common 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

bincount on Intel A10

OpenCL report

32 pipelines

~ 50% utilization

~ 440k ALUTs used
-> 440k / 32 ~ 14000

Reports | Summary | Throughput Analysis | Area Analysis | System Viewers

Summary

Quartus Fit Clock Summary

	Kernel fmax
Frequency (MHz)	205.33333282

Quartus Fit Resource Utilization Summary

	ALMs	FFs	RAMs	DSPs	MLABs
Full design (all kernels)	138198.1	237086	443	291	496
bincount2	132821.8	228958	270	289	428
restrip2	5376.3	8128	173	2	68

Kernel Summary

Kernel Name	Kernel Type	Autorun	Workgroup Size	# Compute Units
bincount2	Single work-item	No	1,1,1	1
restrip2	Single work-item	No	1,1,1	1

Estimated Resource Usage

Kernel Name	ALUTs	FFs	RAMs	DSPs	MLABs
bincount2	316555	419338	271	289	528
restrip2	13360	14172	178	1.5	105
Kernel Subtotal	**329915**	**433510**	**449**	**290**	**633**
Global Interconnect	**3070**	**7569**	**36**	**0**	**0**
Board Interface	**107320**	**214640**	**346**	**72**	**0**
System description ROM	**0**	**67**	**2**	**0**	**0**
Total	**440305 (52%)**	**655786 (38%)**	**833 (31%)**	**362 (24%)**	**633**

bincount2_intel.cl

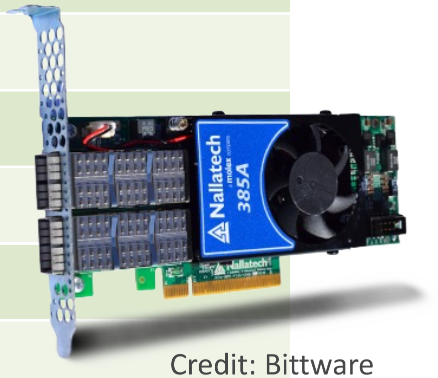
```
#ifndef T_DATA_VAL
    #define T_DATA_VAL  ushort
#endif

#define T_BINNING_VAL   float
#define T_BINNING_NCS   float
#define ULONG_AS_VAL_TYPE(x)  (as_float((uint)(x)))
#define VAL_TYPE_AS_ITYPE(x)  (as_int((x)))
#define ULONG_AS_NCS_TYPE(x)  (as_float((uint)(x)))
#define NCS_TYPE_AS_ITYPE(x)  (as_int((x)))

#include "bisort_prg8.h" /* definition of the bitonic mergesort network */

/* wrapping global net* to net names */
#define BN_N        BISORT_NET8_N
#define BN_LEN      BISORT_NET8_LEN
#define BN_DEPTH    BISORT_NET8_DEPTH
#define _bp         _bisort_prg8

#define UNSEQ_LEN   BN_N    /* unique sequence length */

#ifndef BIN_RANK_PIX
    #define BIN_RANK_PIX 1 /* (BN_N*BIN_RANK_BISORT) */
#endif

#define BIN_RANK_BISORT ((BIN_RANK_PIX+BN_N-1)/BN_N)  /* number of bisort units */

#define NBINS       1024

#define RELAX_LEN   7

/* the largest int16 value is used as a sink position */
#define PIX_POS_SINK    (ushort)(0x7FFF)
/* only upper 15 bins of position are used */
#define PIX_POS_MASK    (ushort)(0x7FFF)
/* lowest bin is flag for a masked position, i.e. all negative positions are masked */
#define PIX_FLG_MASK    (0x8000)
/* get mask for a given position */
#define PIX_MASKED(p)   ((bool)((p & PIX_FLG_MASK)>>15))

#define POS_MASK     0x0000FFFF00000000uL
#define POS_SHIFT    32
#define SET_POS(s,x) (((s) & (~POS_MASK)) | ((((ulong)(x)) & 0xFFFFuL) << POS_SHIFT))
#define GET_POS(s)   ((ushort)(((s) & POS_MASK) >> POS_SHIFT))

#define FLG_MASK     0x0001000000000000uL
```


MAX IV

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 ✅

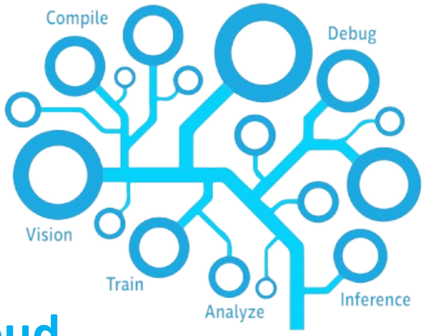


Credit: Bittware

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

Accesibility

lowering the barrier



Credit: Intel

- gitlab.com/MAXIV-SCISW/compute-fpgas/bincount
  - example for A10GX (A10) and D5005 (S10) FPGAs on **Intel DevCloud**
- affordable hardware: **PYNQ platform**

www.pynq.io

- all code in this project:
  - using Python bindings (PyOpenCL)
  - all tests and high-level interface in Python

```
# reshape and reduce output data
dta = np.swapaxes(dta.reshape(buf.shape[0]//32,ntth,32), 1, 2).reshape(buf.shape[0],ntth)[:frame_dta.shape[0],:]
ncs = np.swapaxes(ncs.reshape(buf.shape[0]//32,ntth,32), 1, 2).reshape(buf.shape[0],ntth)[:frame_dta.shape[0],:]

#dta, ncs = bincountx(pos, frame_dta, pix, cor, wgt, nbins=ntth)

[azint-cl] OpenCL will use AOCX binary image: /mxn/home/zdemat/fpga/bincount-fpga/build/p385a_sch_ax115/bincount2_intel.aocx
fpga integration (n=3001), exec. time: 0.181 sec
fpga bincout-intel (0.095 Mpix): 16541.8 fps
[   0.    0. 112. 184. 213.]

4]: t0d = 32.3 # (sec) time of phase transition in diffraction data
# one shoudl read time-step from timestamps/plog, however we know it is 0.020+0.005 s
dtd = 0.020 + 0.005 # sec
# y-time axis
iy = (0.5 + np.arange(frame_dta.shape[0])) * dtd
# show data
```

Compression

Xilinx Vitis Data Compression Library

xilinx.github.io/Vitis_Libraries/data_compression
Apache License, Version 2.0

sources:
- Vitis Library documentation
- https://github.com/lz4/lz4

	perf	f_{max}	resources
VITIS LZ4 compress	290 MB/s	300 MHz	3k LUT, 124 LUTmem, 3.5k REG, 5 BRAM, 6 URAM
VITIS LZ4 decompress	1.8 GB/s	300 MHz	5.5k LUT, 370 LUTmem, 4.8k REG, 4 URAM
XCVU33P (Ultrascale+ FPGA)			440k LUT
LZ4 default decompress	5.0 GB/s	4.9 GHz	Core i7-9700K CPU

- available: lz4, snappy, zlib and zstd
- compression algorithms are often not originally designed for “hardware”
- Outlook
  - 385A is on the edge of memory bandwidth (2xDDR3) and host2device bandwidth
  - adding LZ4 decompression could help with both BW limitations

Similar activities

elsewhere

- *Filip Leonarski et al.* (PSI)

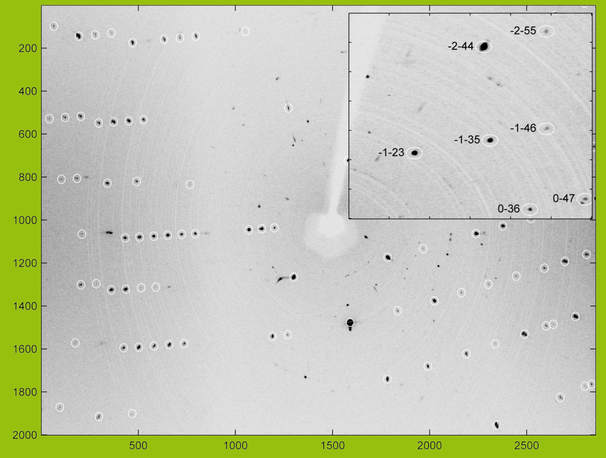
  Fast and accurate data collection for macromolecular crystallography using the JUNGFRAU detector
  - IBM OpenCAPI framework & Xilinx Vivado HLS
  - doi: 10.1038/s41592-018-0143-7

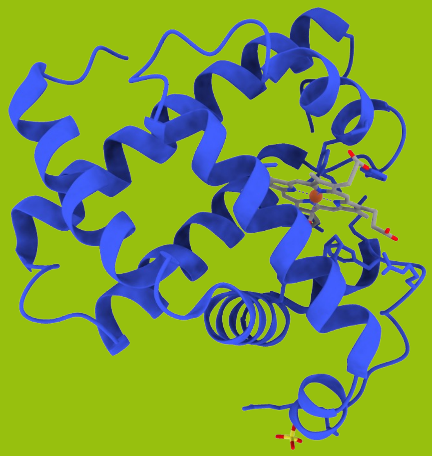
- *Maxime Martelli et al.* (CNRS, Paris)

  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

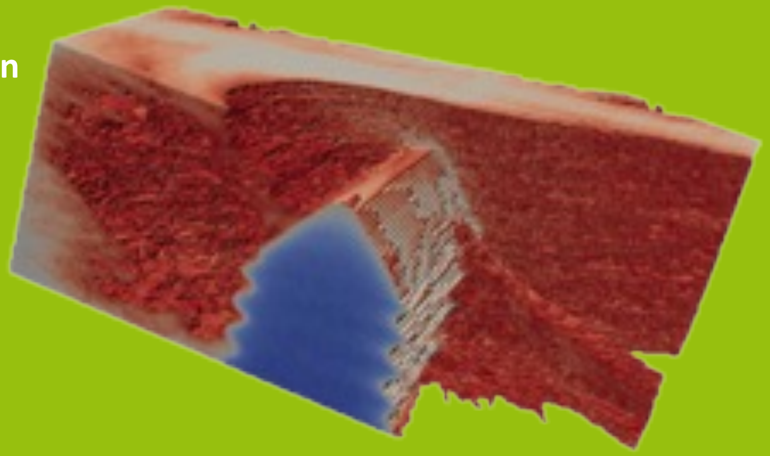
Spot-finding

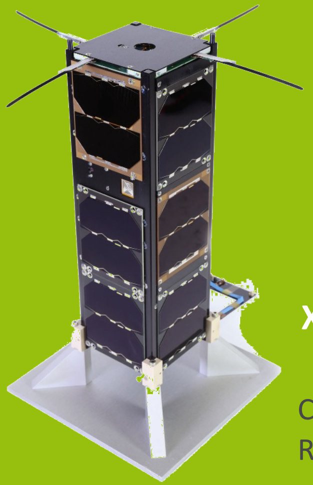




Credit: 2021 joony, Wikimedia

Tomography reconstruction



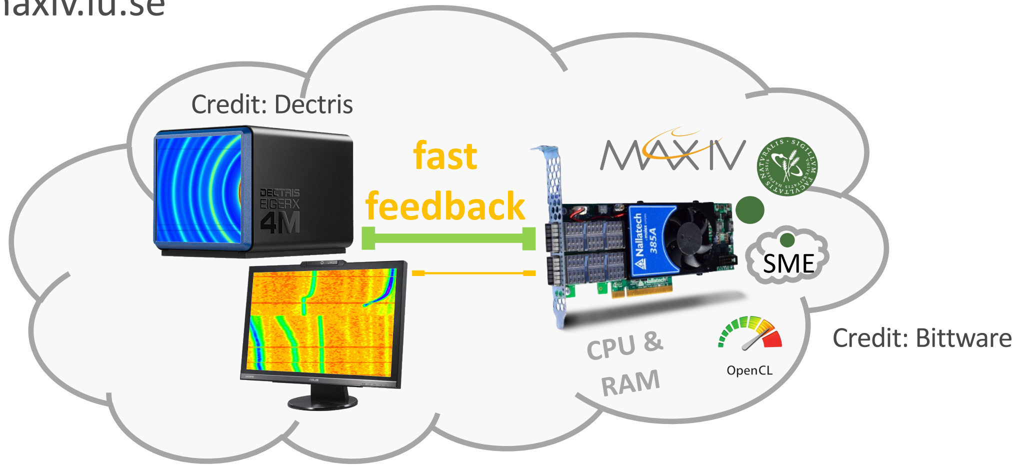


X-ray detector at nanosatellite

Credit: vzlusat2.cz and Rigaku Innovative Technologies Europe

Summary

- **It is possible. We can do AZINT on FPGAs with the standard scientific precision, flexibility and performance higher or comparable to GPUs**
- 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
and the whole team