New possible use-cases of FPGAs in HEP High-Level-Trigger systems

Christian Färber
CERN Openlab Fellow
LHCb Online group

On behalf of the LHCb Online group and the HTC Collaboration

CERN EP-ESE electronics seminar, Geneva
12.12.2017

HTCC

- High Throughput Computing Collaboration
- Members from Intel® and CERN LHCb/IT
- Test Intel technology for the usage in trigger and data acquisition (TDAQ) systems
- Projects
  - Intel® KNL computing accelerator
  - Intel® Omni-Path Architecture 100 Gbit/s network
  - Intel® Xeon®+FPGA computing accelerator

General HEP Readout Chain

Optical links

Fast networks

Readout electronic for detectors (Custom)

Mainly ASICs
In low rad. areas FPGAs

Distribution of ECS/TFC

Back-end electronics (Custom)

Many FPGAs and CCPCs

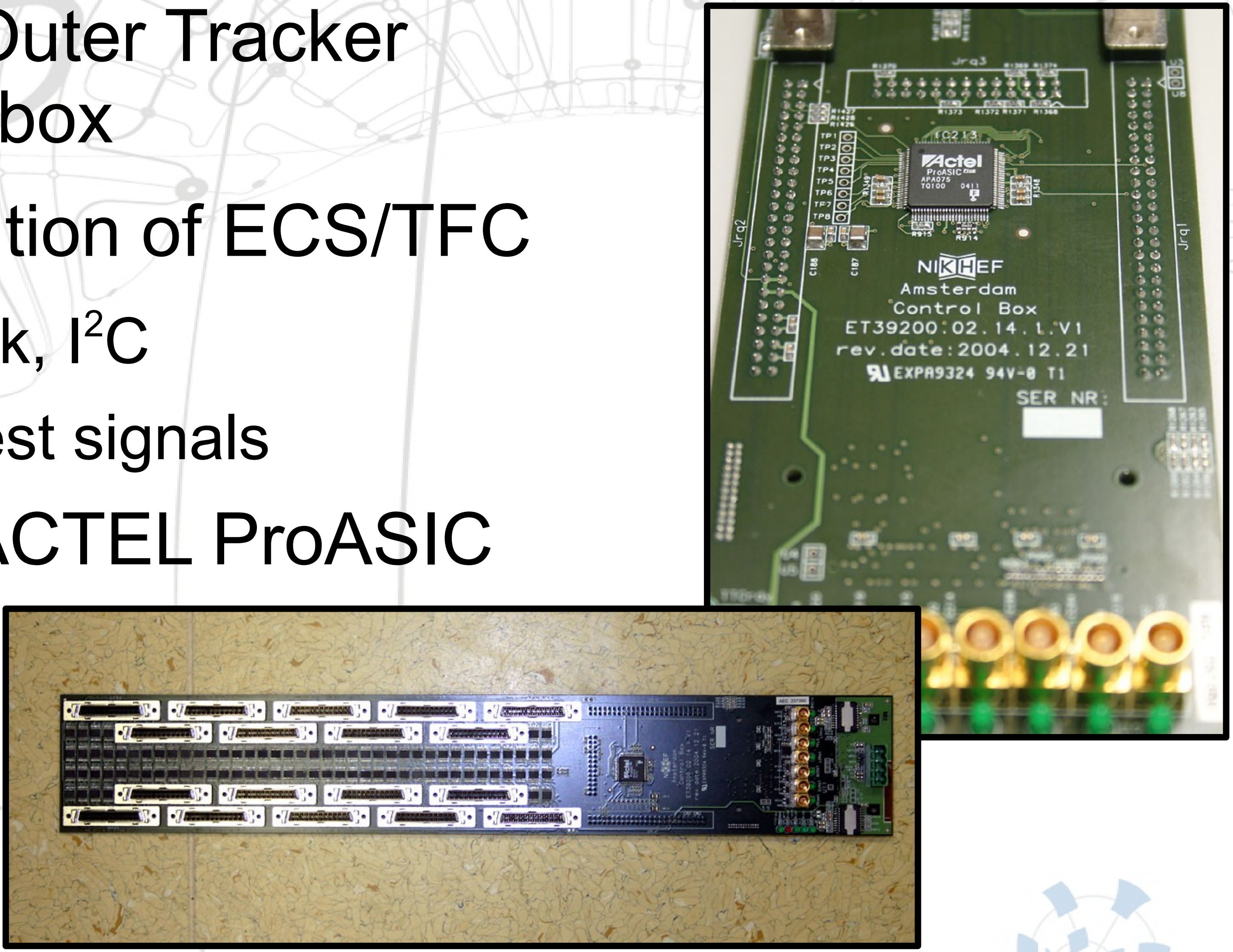
Pre-processing, zero supression, L0 trigger

Computing farms (Commercial)

FPGA usage under investigation for the Event Filter Farms (HLT)!

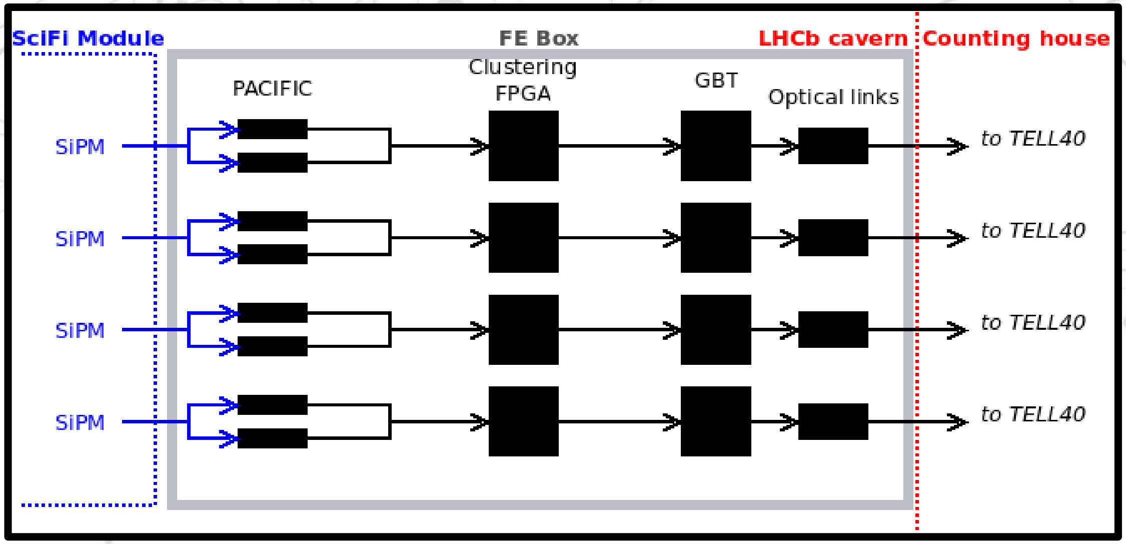
Example: Control Electronics

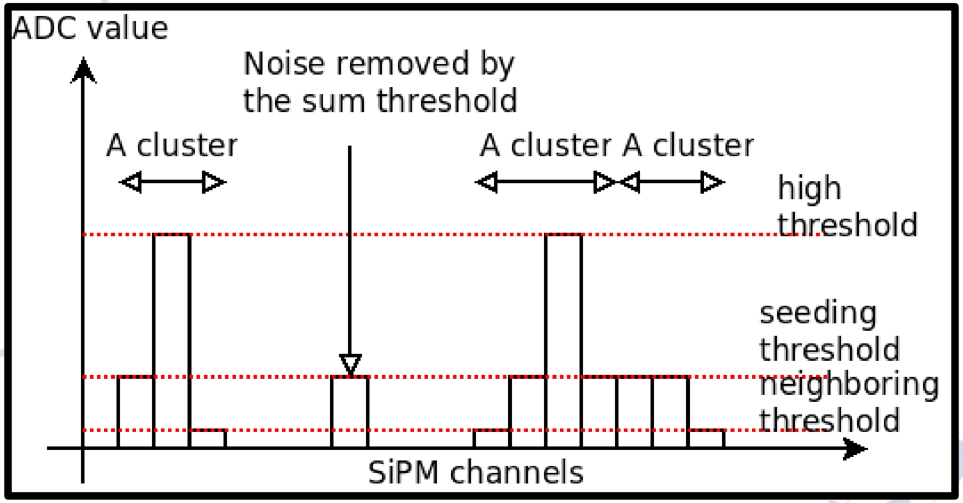
- LHCb Outer Tracker control box
- Distribution of ECS/TFC
  - Clk, I^2C
  - Test signals
- Using ACTEL ProASIC



Example: Readout Electronics

- Future LHCb SciFi readout electronics
- Using Microsemi IGLOO2 120kLE
- Drive data from PACIFIC to GBT
- Clusterization and Zero-suppression

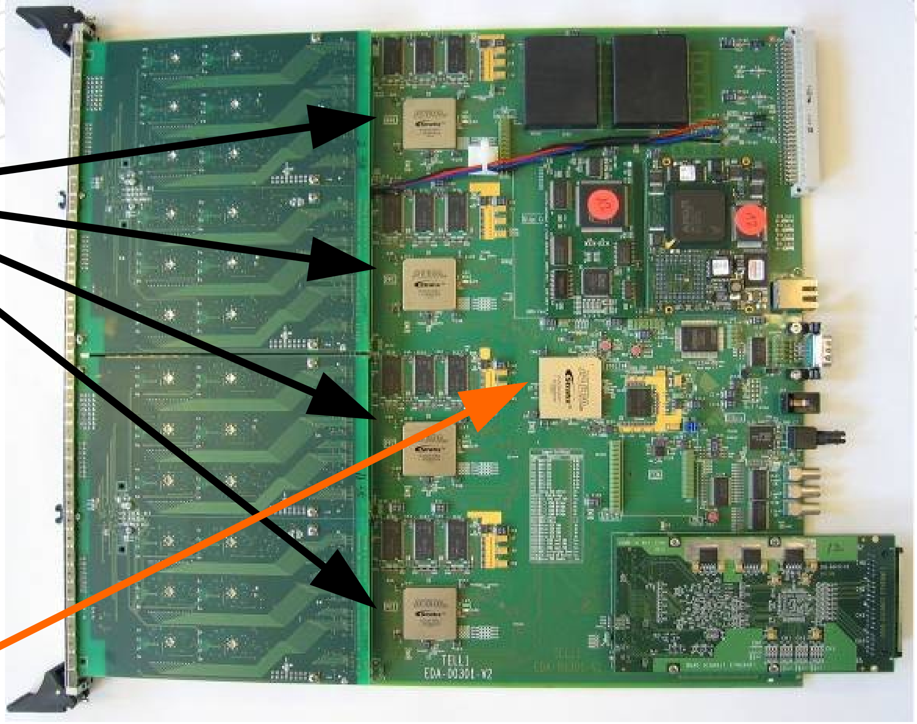




Hervé Chanal
„The readout electronics of the SciFi Tracker for LHCb detector Upgrade“
TWEPP 2015

Example: Back-end Electronics

- LHCb TELL1
- 4x Stratix GX for pre-processing, zero suppression, error checking

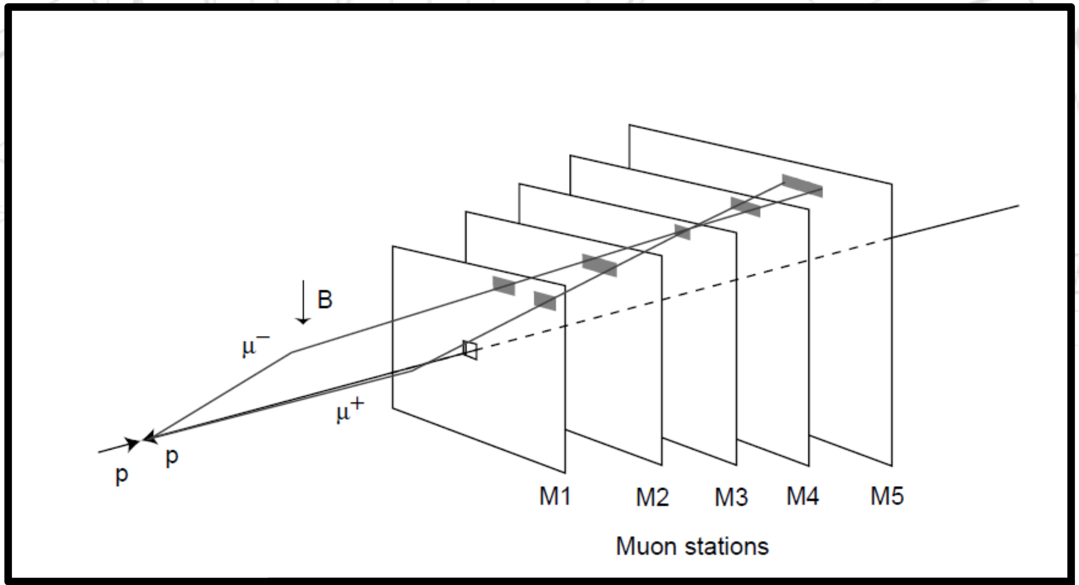


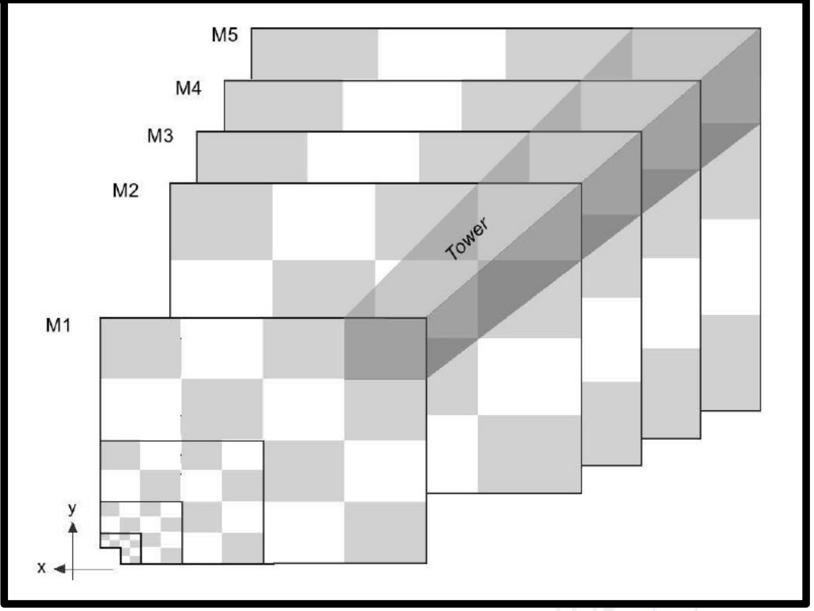
Custom electronics

- 1x Stratix GX for event building, data flow monitoring and preparing data packets (4x 1GBit/s Ethernet)

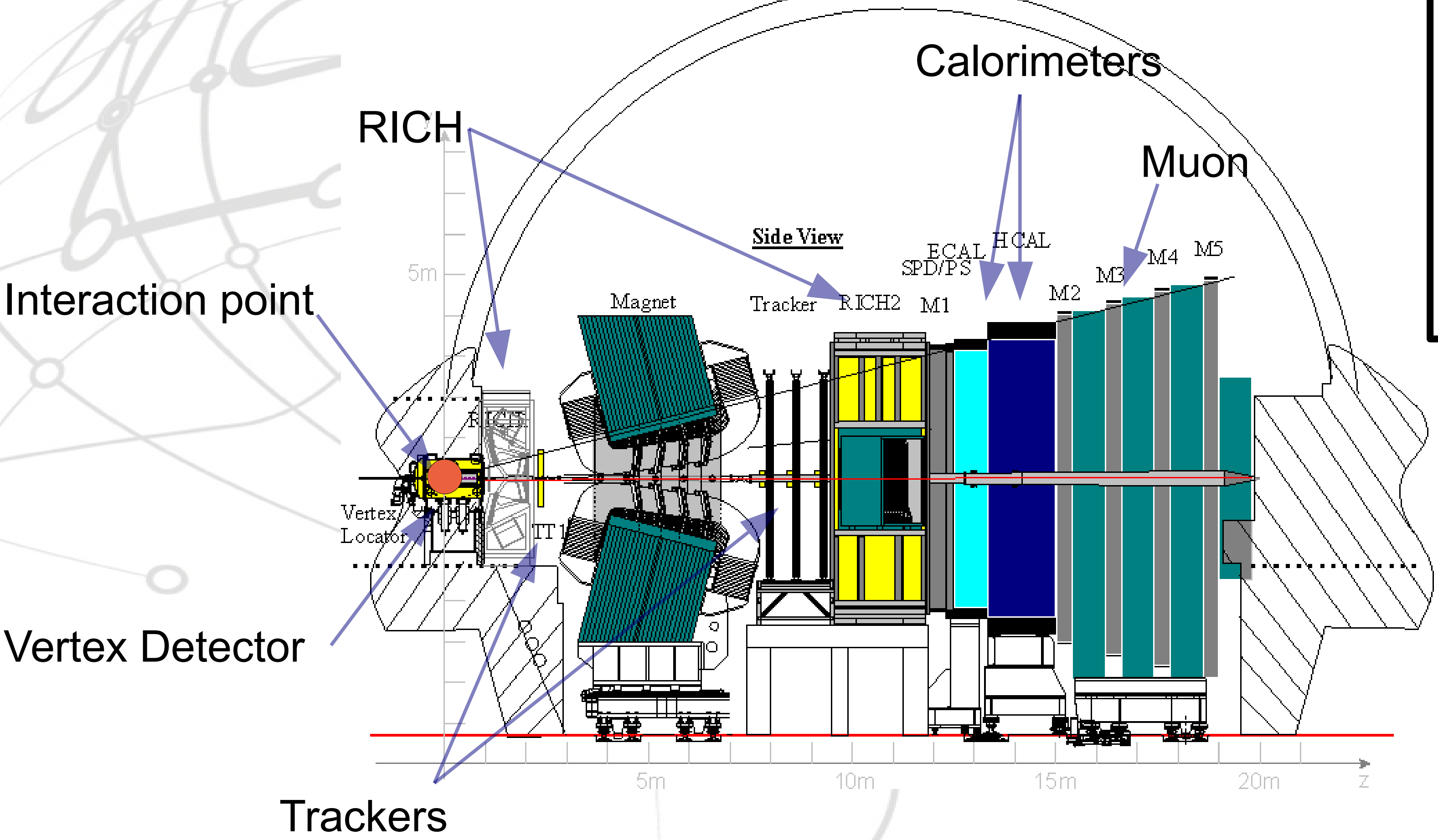
Example: Trigger Electronics

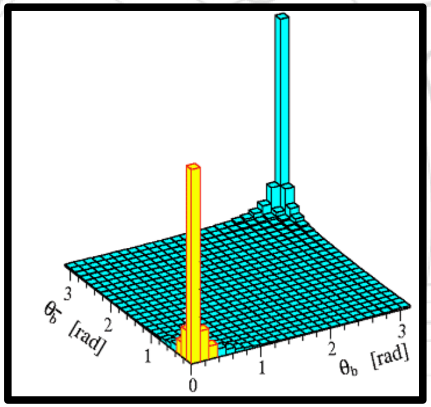
- LHCb L0 Muon trigger, searches for the 2 highest trans. momentum muons
- Receiving 130GB/s
- Every 25ns
- Max. latency 1.2µs !
- Using 248 Stratix GX
- Running 18432 tracking algorithms parallel





Detector Example: LHCb



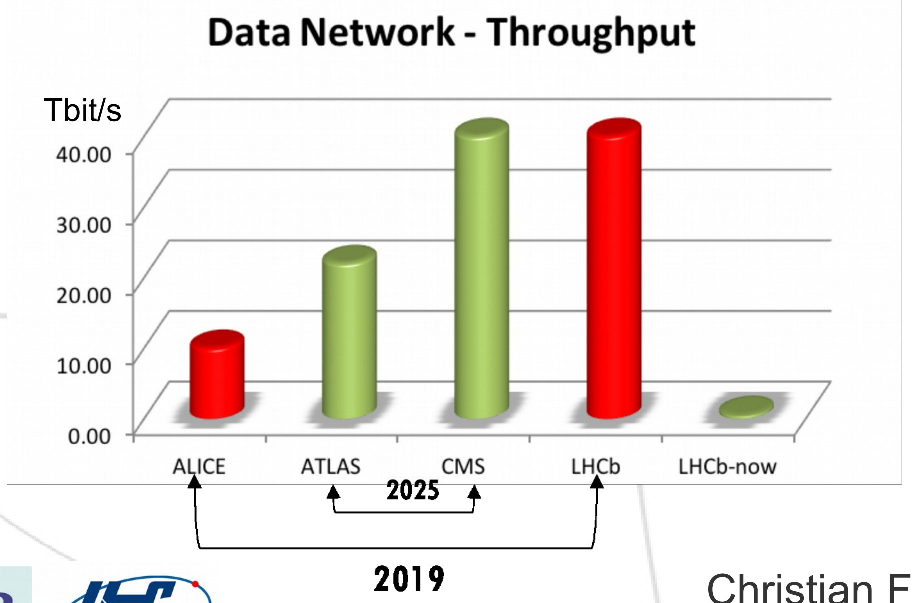


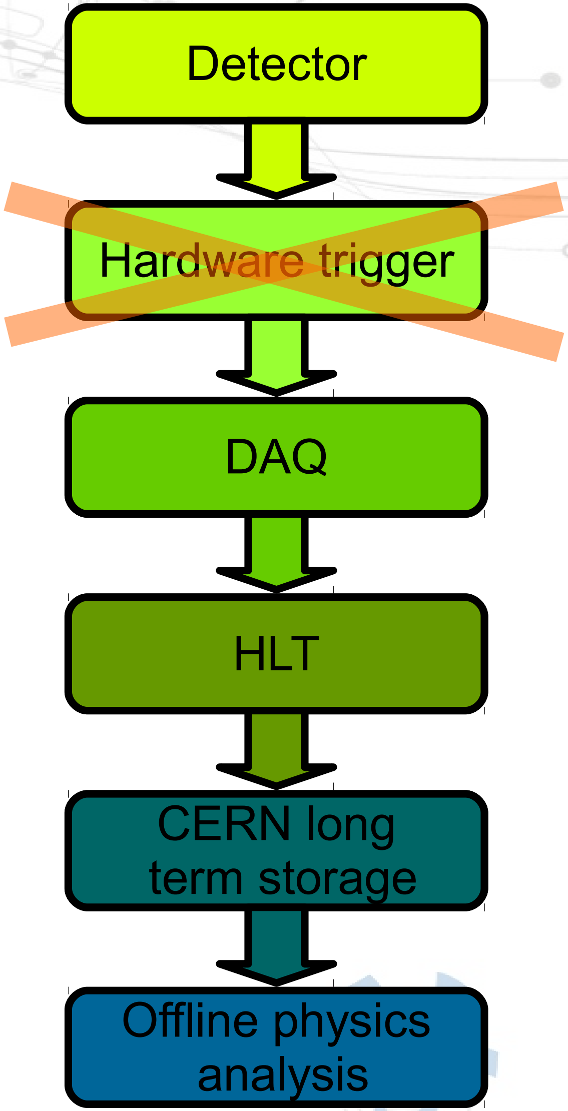
- Single-arm spectrometer designed to search new physics through measuring CP violation and rare decays of heavy flavour mesons.

- 40 MHz proton proton collisions
- Trigger with 1 MHz, upgrade to 40 MHz
- Bandwidth after upgrade up to 40 Tbit/s

Future Challenges

- Higher luminosity from LHC
- Upgraded sub-detector Front-Ends
- Removal of hardware trigger
- Software trigger has to handle
  - Larger event size (50 KB to 100 KB)
  - Larger event rate (1 MHz to 40 MHz)



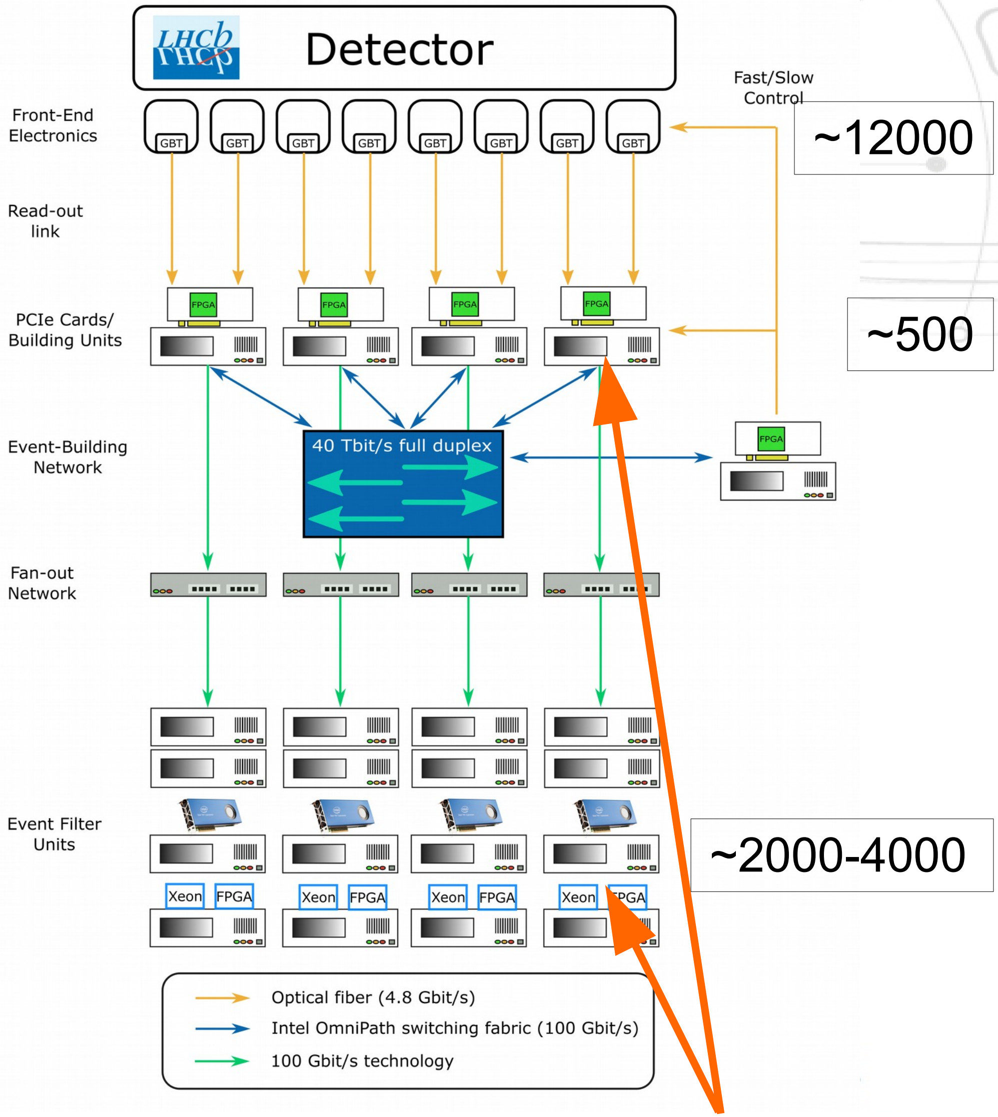


Upgrade Readout Schematic

- Raw data input ~ 40 Tbit/s
- EFF needs fast processing of trigger algorithms, different technologies are explored.

- Test FPGA compute accelerators for usage in:
  - Event building
    - Decompressing and re-formatting packed binary data from detector
  - Event filtering
    - Tracking
    - Particle identification

- Compare with: GPUs, Intel® Xeon Phi^{TM} and other compute accelerators



Which technologies?

FPGAs as Compute Accelerators

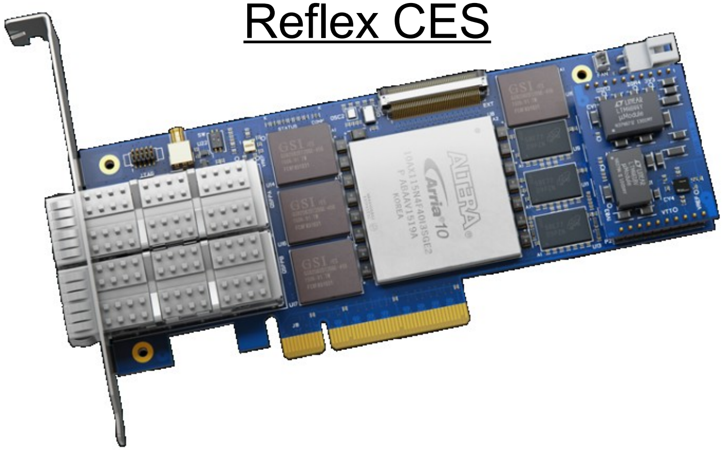
- Microsoft Catapult and Bing
  - Improve performance, reduce power consumption

- Reduce the number of von Neumann abstraction layers
  - Bit level operations
- Power only logic cells and registers needed
- Current test devices in LHCb
  - Nallatech PCIe with OpenCL
  - Intel® Xeon®+FPGA

FPGA compute accelerators

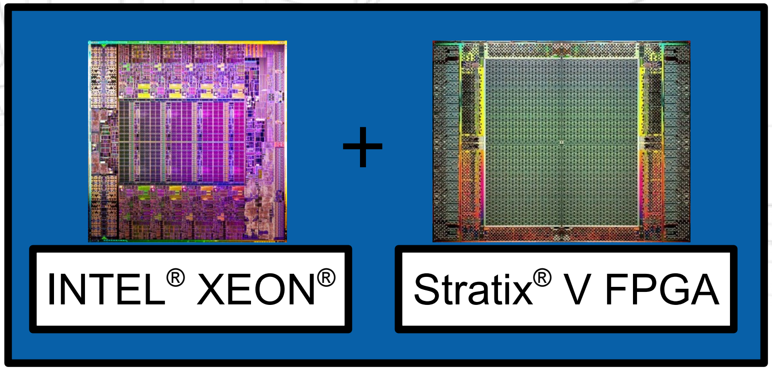
- Typical PCIe 3.0 card with high performance FPGA
  - NIC or GPU size
- On board memory
  e.g. 16 GB DDR4
- Some cards have also network e.g. QSFP 10/40 GbE,…
  - More flexible than GPUs
- Programming in OpenCL
  - OpenCL compiler → HDL
- Power consumption below GPU, price higher than GPU
- Use cases: Machine Learning,
  Gene Sequencing,
  Real-time Network Analytics
-

Reflex CES



Nallatech

Intel® Xeon®+FPGA with Stratix® V FPGA



- Two socket system:

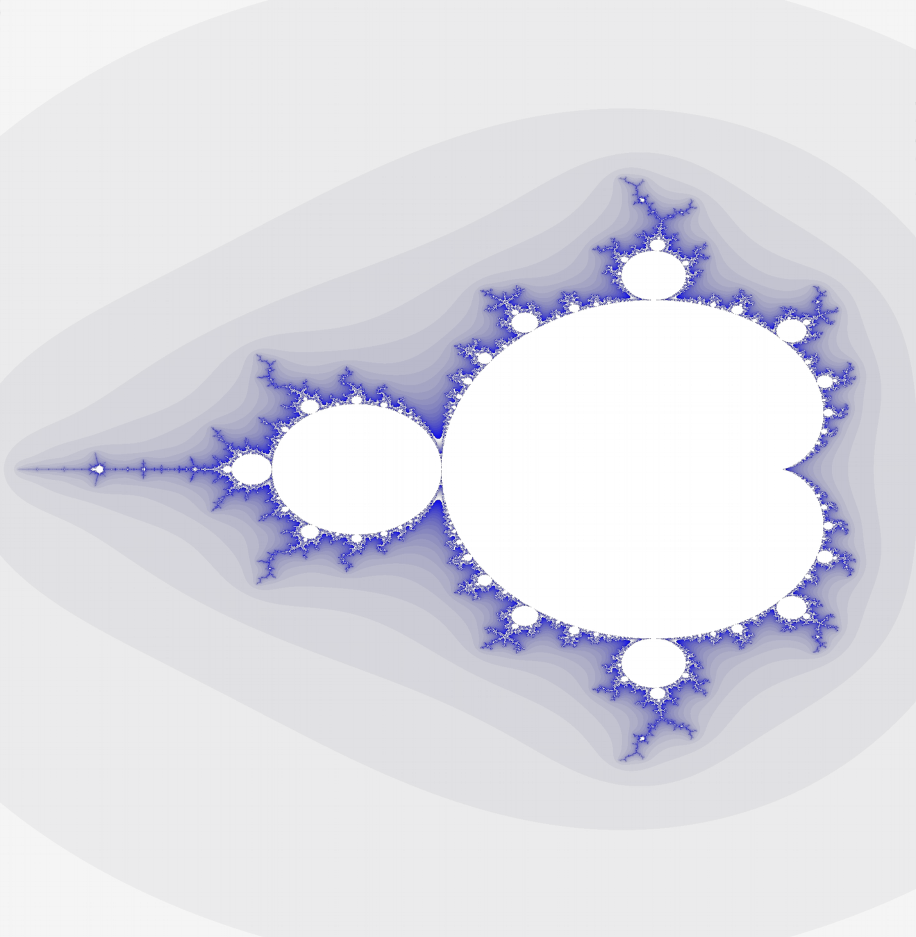
  First: Intel® Xeon® E5-2680 v2

  Second: Intel® Stratix® V GX A7 FPGA

  - 234'720 ALMs, 940'000 Registers, 256 DSPs
- Host Interface: high-bandwidth and low latency
- Memory: Cache-coherent access to main memory
- Programming model: Verilog and OpenCL

Mandelbrot on Intel® Xeon®+FPGA

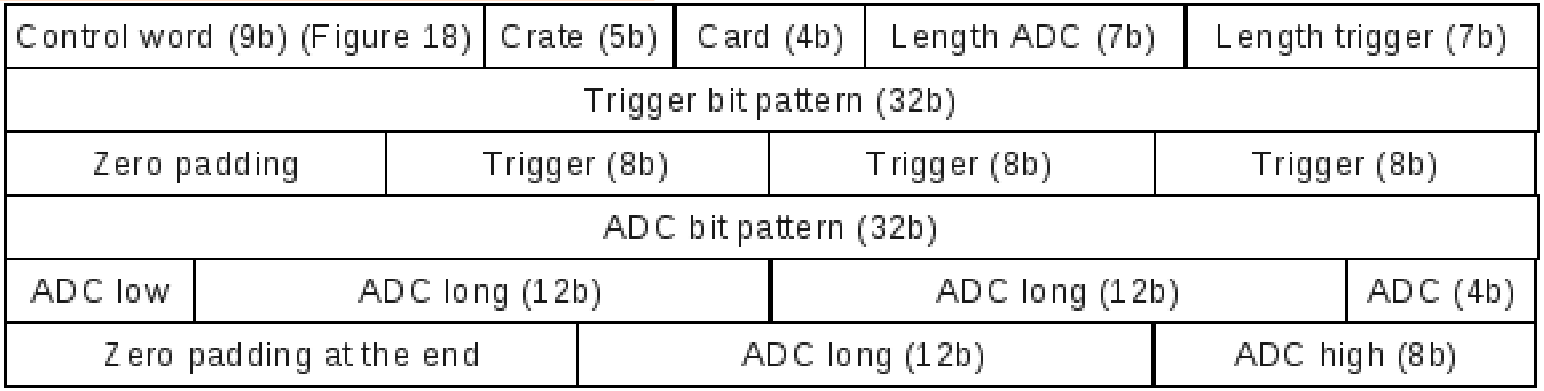
- Mandelbrot with floating point precision
  - Implemented 22 fpMandel pipelines running at 200 MHz, each handles 16 pixels in parallel (total: 352 pixels)
  - FPGA is x12 faster than Intel® Xeon® running 20 threads in parallel
  - Used 72/256 DSPs
  - Reuse of data on FPGA high



Test case: LHCb Calorimeter Raw Data Decoding

- Two types of calorimeters in LHCb: ECAL/HCAL
- 32 ADC channels for each FEB of 238 FEBs
- Raw data format:
  - ADC data is sent using 4 bits or 12 bits
  - A 32 bit word stores information about which channel has short/long decoding

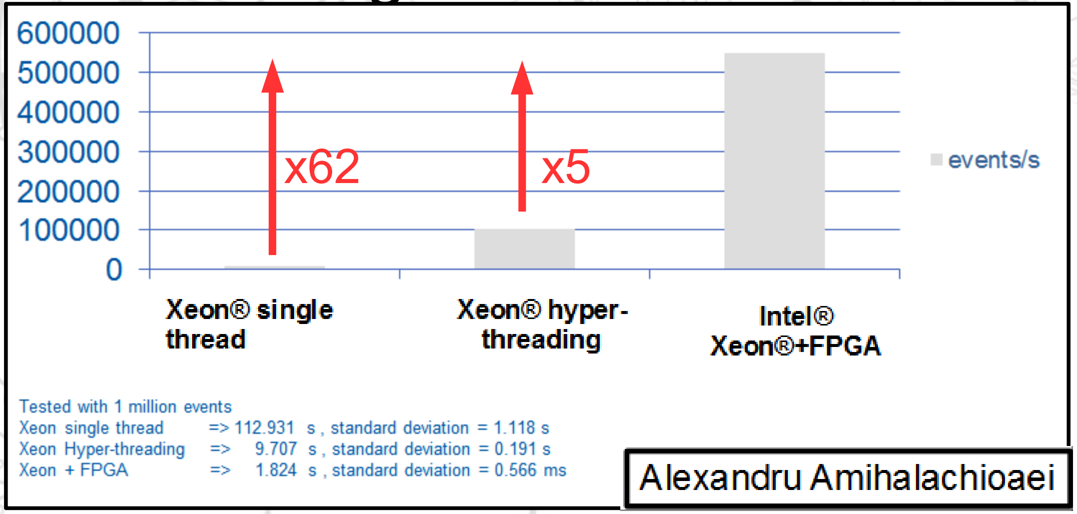
LHCb Calorimeter raw data bank



<table>
<tr><td colspan="3">Control word (9b) (Figure 18)</td><td colspan="2">Crate (5b)</td><td colspan="2">Card (4b)</td><td colspan="2">Length ADC (7b)</td><td colspan="2">Length trigger (7b)</td></tr>
<tr><td colspan="11">Trigger bit pattern (32b)</td></tr>
<tr><td colspan="2">Zero padding</td><td colspan="4">Trigger (8b)</td><td colspan="2">Trigger (8b)</td><td colspan="3">Trigger (8b)</td></tr>
<tr><td colspan="11">ADC bit pattern (32b)</td></tr>
<tr><td>ADC low</td><td colspan="5">ADC long (12b)</td><td colspan="4">ADC long (12b)</td><td>ADC (4b)</td></tr>
<tr><td colspan="4">Zero padding at the end</td><td colspan="4">ADC long (12b)</td><td colspan="3">ADC high (8b)</td></tr>
</table>

Results Calorimeter Raw Data Decoding: Ivy Bridge + StratixV

- On FPGAs the decoding can be realized more efficiently



```
Tested with 1 million events
Xeon single thread      => 112.931  s , standard deviation = 1.118 s
Xeon Hyper-threading    =>    9.707  s , standard deviation = 0.191 s
Xeon + FPGA             =>    1.824  s , standard deviation = 0.566 ms
```

Alexandru Amihalachioaei

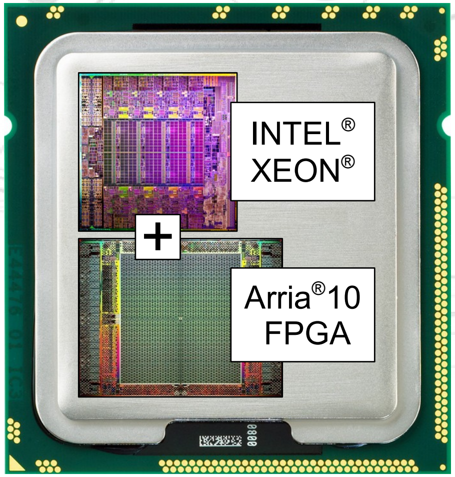
Measured acceleration with Stratix$^{®}$ V: a factor 62

- Bottleneck is bandwidth between CPU and FPGA
  → add more cores, tested BDW + Arria10 GX FPGA

FPGA resources:

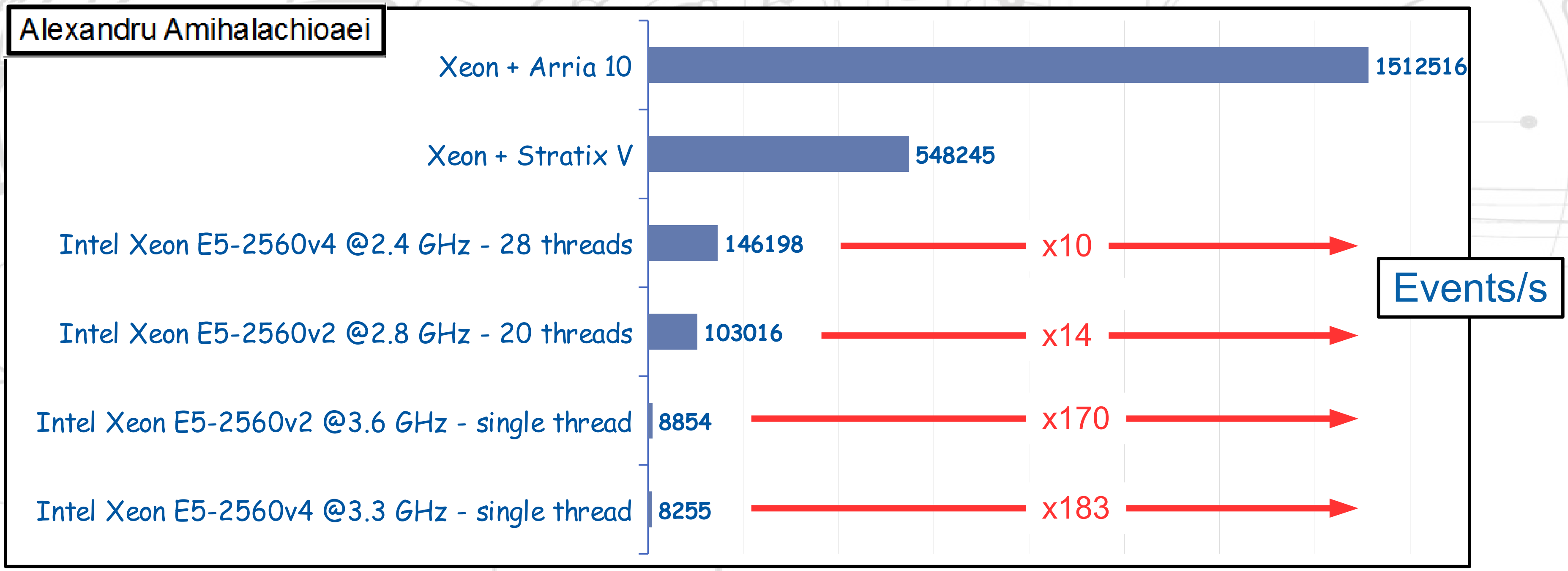
FPGA Resource Type	FPGA Resources used [%]	For Interface used [%]
ALMs	58	30
DSPs	0	0
Registers	15	5

Intel® Xeon®+FPGA with Arria® 10 FPGA



- Multi-chip package including:
  - Intel® Xeon® E5-2600 v4
  - Intel® Arria® 10 GX 1150 FPGA
    - 427'200 ALMs, 1'708'800 Registers, 1'518 DSPs
- Hardened floating point add/mult blocks (HFB)
- Host Interface: Bandwidth target 5x higher than Stratix® V version
- Memory: Cache-coherent access to main memory
- Programming model: Verilog, soon also OpenCL

Results Calorimeter Raw Data Decoding: BDW+Arria10

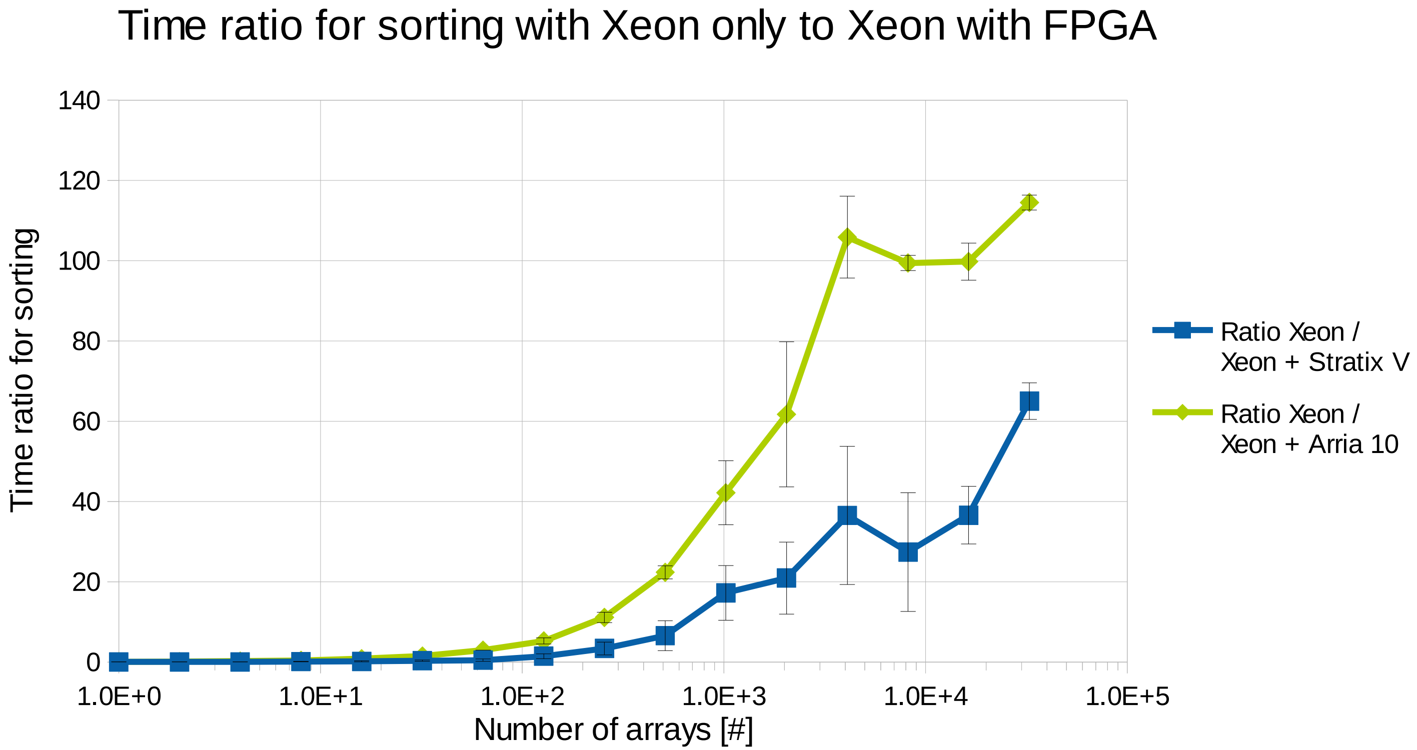


- The higher bandwidth of the newest Intel® Xeon®+FPGA results in an impressive acceleration of a factor 180

FPGA Resource Type	FPGA Resources used [%]	For Interface used [%]
ALMs	57	18
DSPs	0	0
Registers	19	5

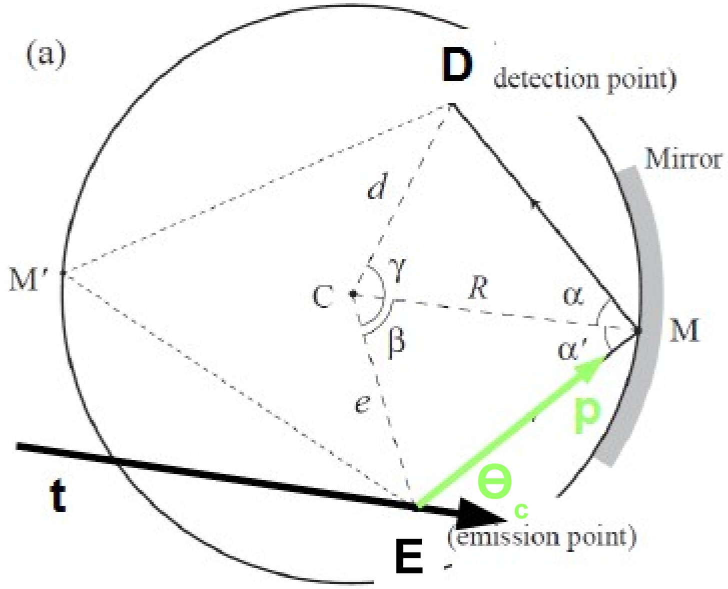
Sorting with Intel® Xeon®+FPGA

- Sorting of INT arrays with 32 elements
  - Implemented pipeline with 32 array stages
  - FPGA sort is up to x117 faster than single Xeon® thread
  - Bandwidth through the FPGA is the bottleneck



Test Case: RICH PID Algorithm

- Calculate Cherenkov angle **Θ_c** for each track **t** and detection point **D**, not a typical FPGA algorithm
- RICH PID is not processed for every event, processing time is too long!



Reference: LHCb Note LHCb-98-040

Calculations:

- solve quartic equation
- cube root
- complex square root
- rotation matrix
- scalar/cross products

Implementation of Cherenkov Angle Reconstruction Stratix® V

- 748 clock cycle long pipeline written in Verilog
  - Additional blocks developed: cube root, complex square root, rot. matrix, cross/scalar product,...
  - Lengthy task in Verilog with all test benches (implementation took 2.5 months)
- Pipeline running with 200 MHz → 5 ns per photon
- FPGA resources:

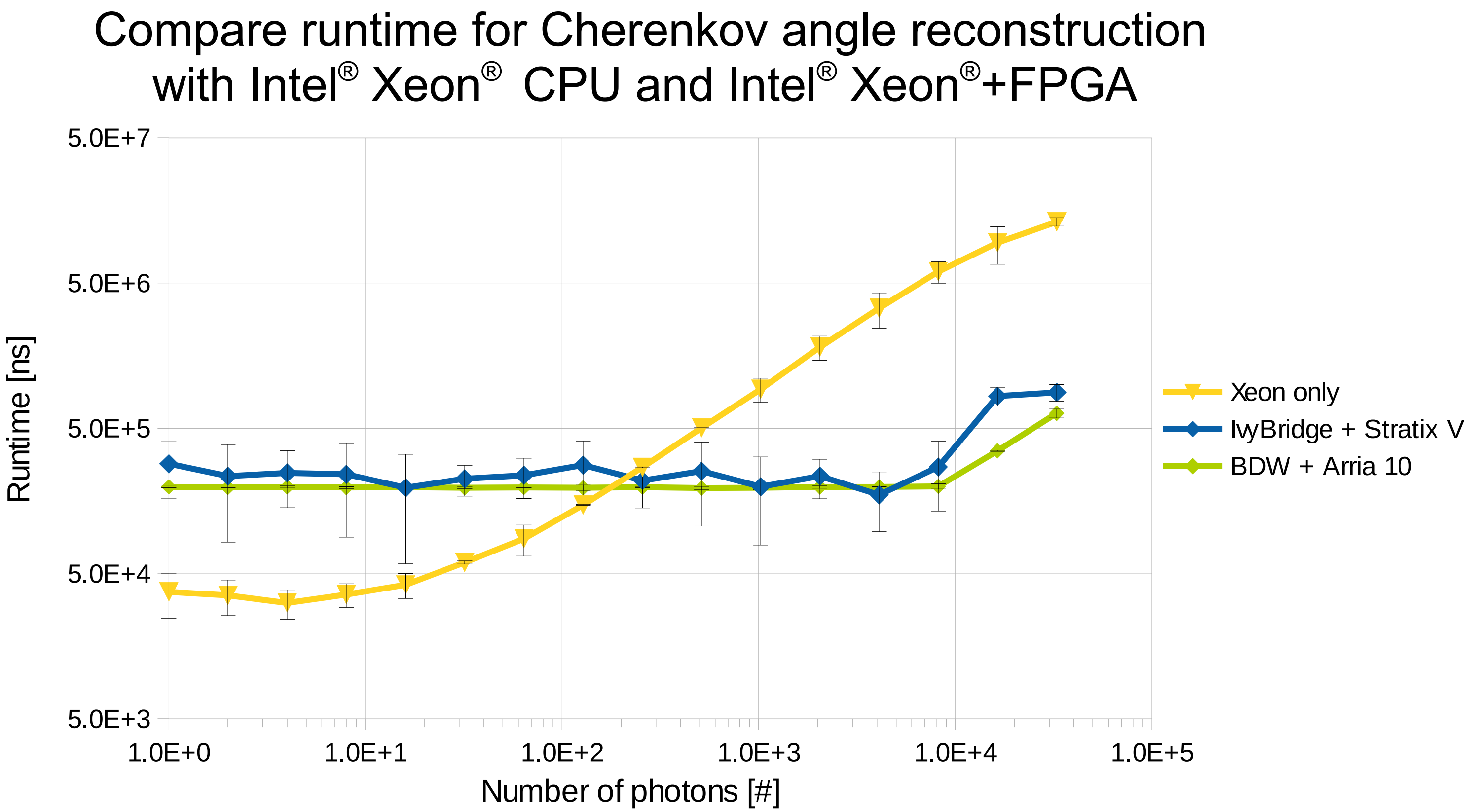
FPGA Resource Type	FPGA Resources used [%]	For Interface used [%]
ALMs	88	30
DSPs	67	0
Registers	48	5

Implementation of Cherenkov Angle Reconstruction Arria® 10

- 259 clock cycle long pipeline written in Verilog
  - Stratix® V blocks ported using HFB: complex square root, rot. matrix, cross/scalar product,...
- Pipeline running with 200 MHz → 5 ns per photon
  - With Arria® 10 GT FPGA 400 MHz possible
- FPGA resources:

FPGA Resource Type	FPGA Resources used [%]	For Interface used [%]
ALMs	32	18
DSPs	15	0
Registers	12	5

Intel® Xeon®+FPGA Results

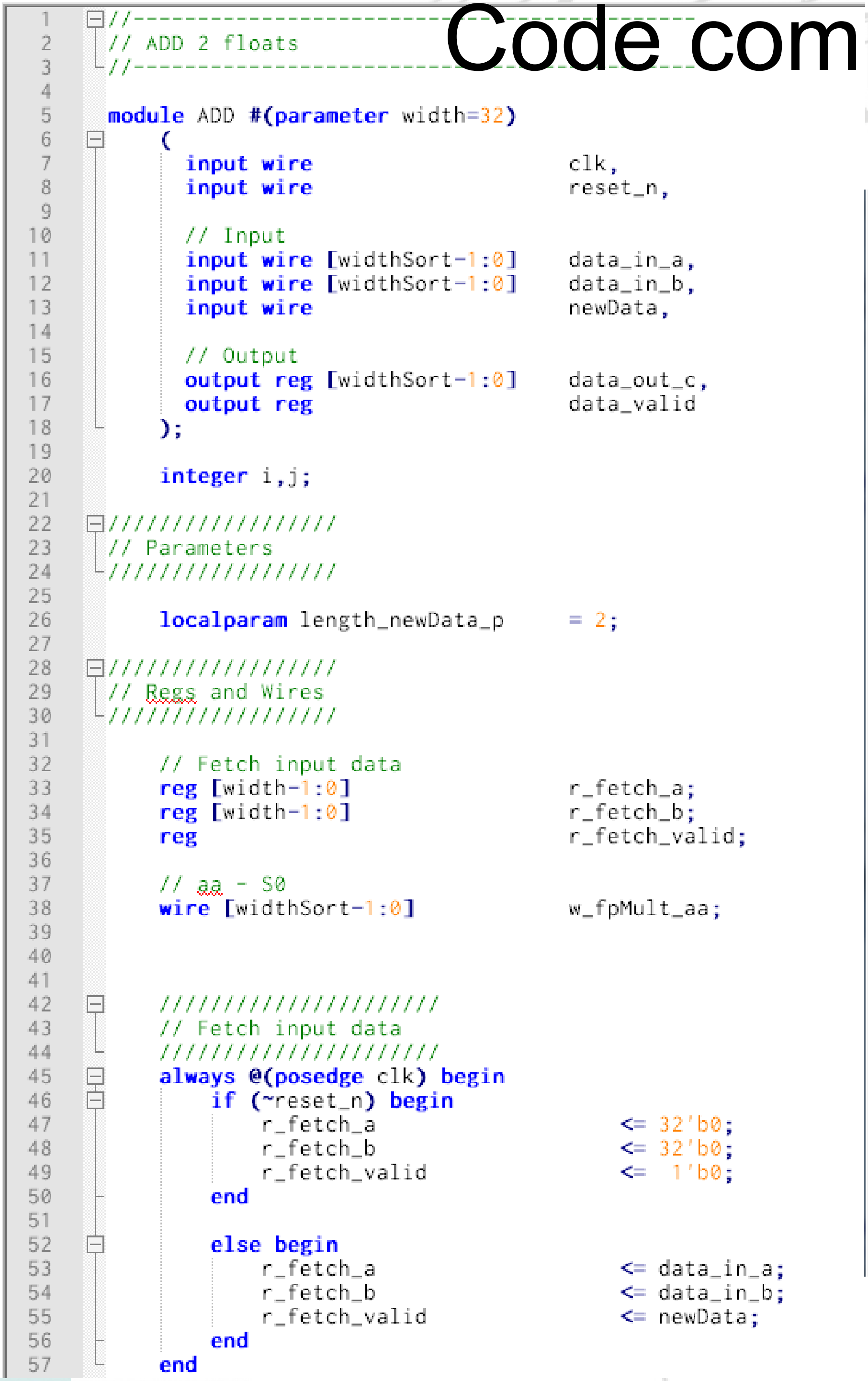


- Acceleration of up to factor 35 with Intel® Xeon®+FPGA
- Theoretical limit of photon pipeline: a factor 64 with respect to single Intel® Xeon® thread, for Arria® 10 a factor ~ 300
- Bottleneck: Data transfer bandwidth to FPGA, caching can improve this, tests ongoing

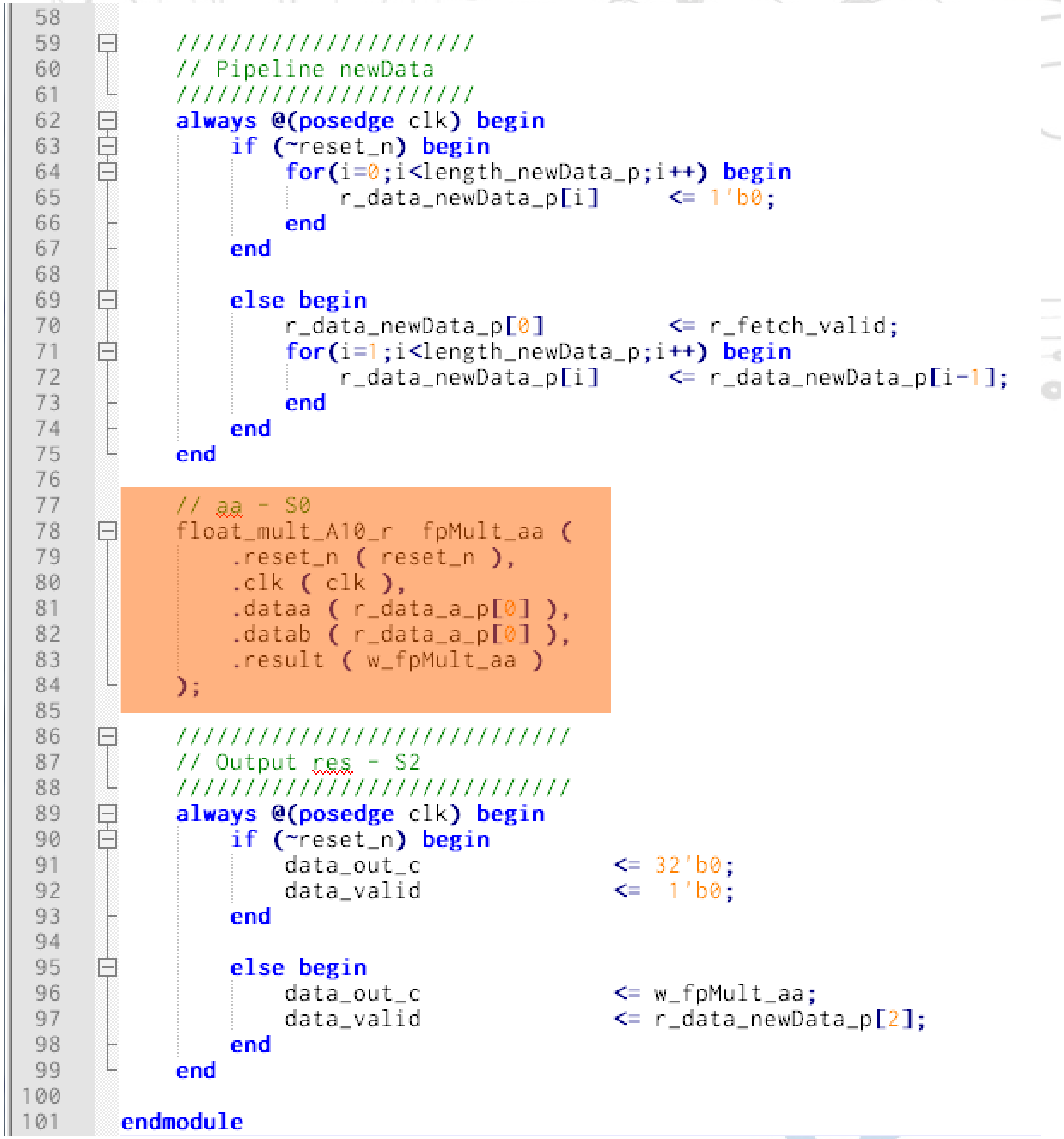
Open Computing Language (OpenCL)

- Developed by Apple, later Khronos Group, based on C99, first release 2009
- Standard to run code on heterogeneous platforms
  - CPUs, GPUs, FPGAs, …
- Program: Host control, kernel run on GPU,FPGA,…
  - Compiled at run-time
- Memory hierarchy: global (main memory), read-only (for kernel), local (shared by group of PE), per-element private memory
- For FPGA case, BSP needed and synthesis is done in advance (OpenCL kernel → HDL → bitstream)

Code compare Verilog

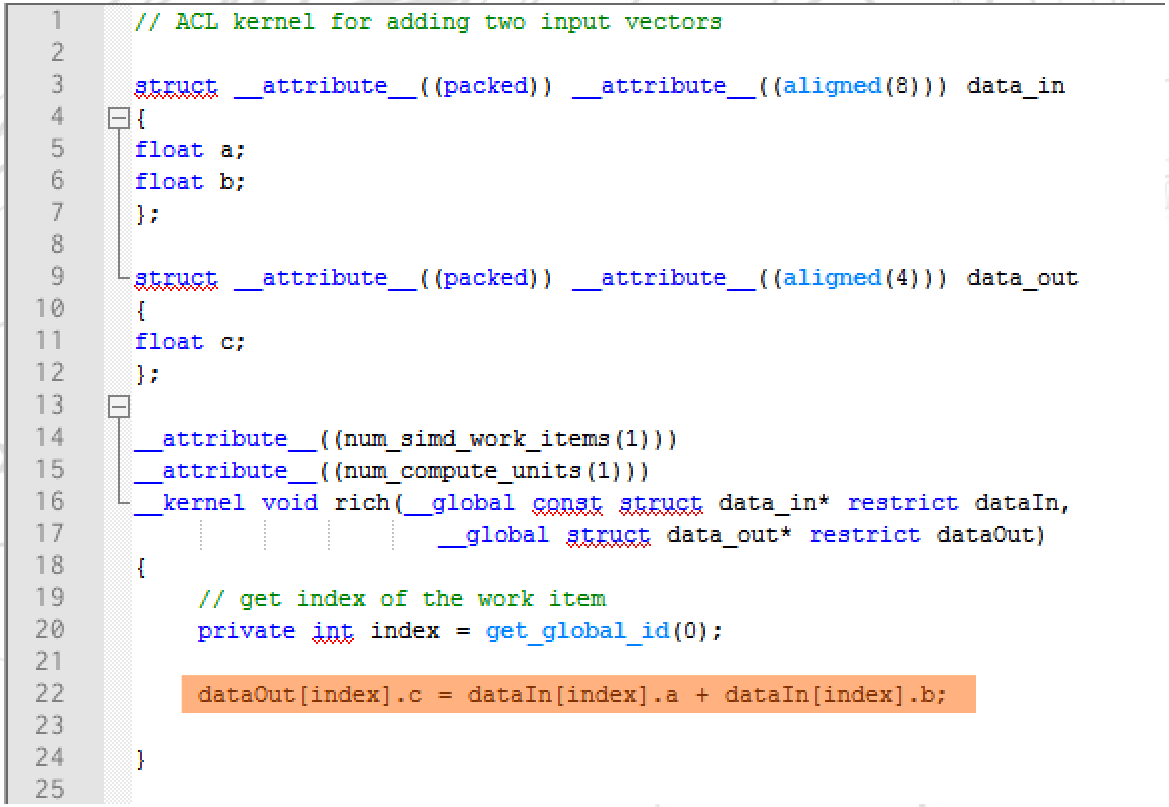


```
//-----------------------------------------------------------
// ADD 2 floats
//-----------------------------------------------------------

module ADD #(parameter width=32)
    (
      input wire                   clk,
      input wire                   reset_n,

      // Input
      input wire [widthSort-1:0]   data_in_a,
      input wire [widthSort-1:0]   data_in_b,
      input wire                   newData,

      // Output
      output reg [widthSort-1:0]   data_out_c,
      output reg                   data_valid
    );

    integer i,j;

//////////////////
// Parameters
//////////////////

    localparam length_newData_p    = 2;

//////////////////
// Regs and Wires
//////////////////

    // Fetch input data
    reg [width-1:0]                r_fetch_a;
    reg [width-1:0]                r_fetch_b;
    reg                            r_fetch_valid;

    // aa - S0
    wire [widthSort-1:0]           w_fpMult_aa;


    //////////////////////
    // Fetch input data
    //////////////////////
    always @(posedge clk) begin
        if (~reset_n) begin
            r_fetch_a                  <= 32'b0;
            r_fetch_b                  <= 32'b0;
            r_fetch_valid              <=  1'b0;
        end

        else begin
            r_fetch_a                  <= data_in_a;
            r_fetch_b                  <= data_in_b;
            r_fetch_valid              <= newData;
        end
    end
```



```
    //////////////////////
    // Pipeline newData
    //////////////////////
    always @(posedge clk) begin
        if (~reset_n) begin
            for(i=0;i<length_newData_p;i++) begin
                r_data_newData_p[i]      <= 1'b0;
            end
        end

        else begin
            r_data_newData_p[0]          <= r_fetch_valid;
            for(i=1;i<length_newData_p;i++) begin
                r_data_newData_p[i]      <= r_data_newData_p[i-1];
            end
        end
    end

    // aa - S0
    float_mult_A10_r  fpMult_aa (
        .reset_n ( reset_n ),
        .clk ( clk ),
        .dataa ( r_data_a_p[0] ),
        .datab ( r_data_a_p[0] ),
        .result ( w_fpMult_aa )
    );

    /////////////////////////////
    // Output res - S2
    /////////////////////////////
    always @(posedge clk) begin
        if (~reset_n) begin
            data_out_c               <= 32'b0;
            data_valid               <=  1'b0;
        end

        else begin
            data_out_c               <= w_fpMult_aa;
            data_valid               <= r_data_newData_p[2];
        end
    end

endmodule
```

Code compare OpenCL



```
// ACL kernel for adding two input vectors

struct __attribute__((packed)) __attribute__((aligned(8))) data_in
{
float a;
float b;
};

struct __attribute__((packed)) __attribute__((aligned(4))) data_out
{
float c;
};

__attribute__((num_simd_work_items(1)))
__attribute__((num_compute_units(1)))
__kernel void rich(__global const struct data_in* restrict dataIn,
                   __global struct data_out* restrict dataOut)
{
    // get index of the work item
    private int index = get_global_id(0);

    dataOut[index].c = dataIn[index].a + dataIn[index].b;

}

```

- No interface to write, using Board Support Package (BSP)
- Using high-level language
- Far less code → easier to develop and to maintain

Compare Verilog - OpenCL

OpenCL

- Development time

2.5 months – 2 weeks

Faster

3400 lines Verilog – 250 lines C

Easier

- Performance

Cube root : x35 – x30

RICH : x35 – x26

Comparable performance

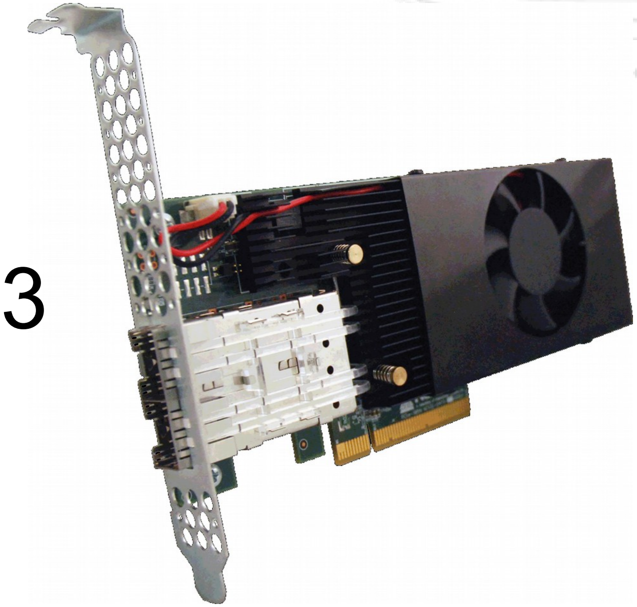
- FPGA resource usage Stratix® V

RICH Kernel	Verilog RTL	OpenCL
FPGA Resource Type	FPGA Resources used [%]	FPGA Resources used [%]
ALMs	88	63
DSPs	67	82
Registers	48	24

Similar resource usage

Nallatech 385 Board

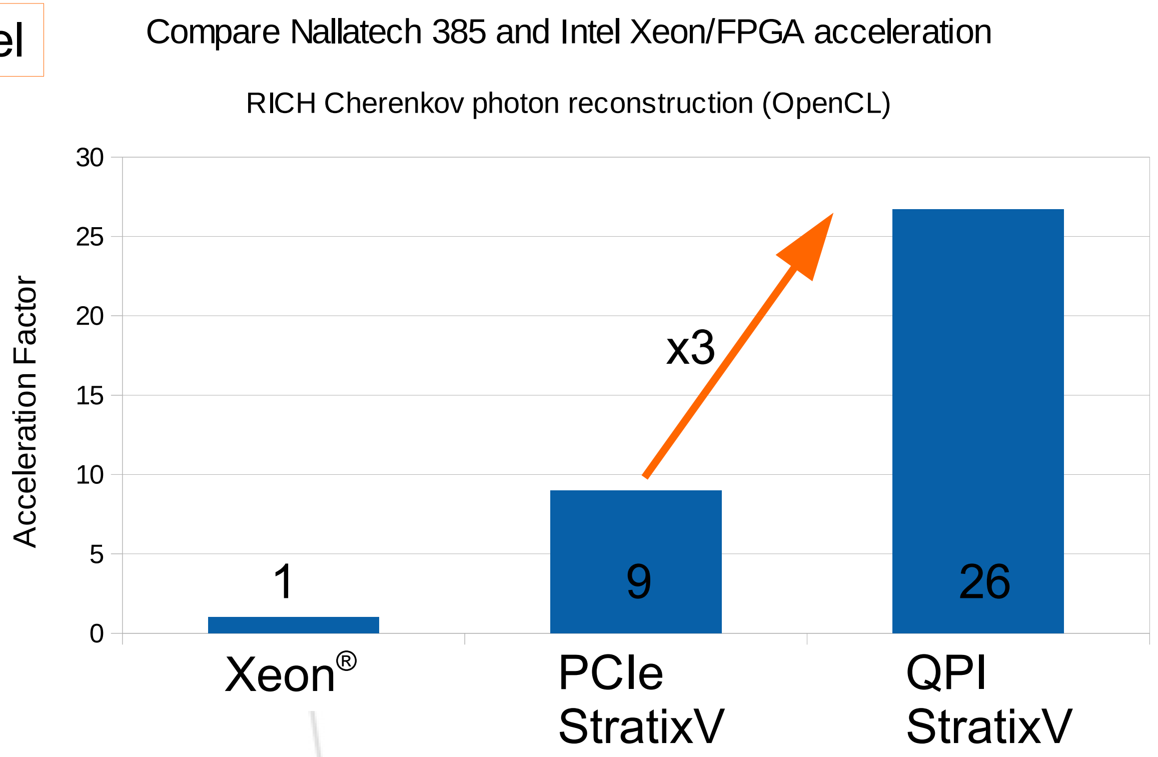
- FPGA: Intel® Stratix® V GX A7
  - 234'720 ALMs, 940'000 Registers
  - 256 DSPs
- Programming model: OpenCL
- Host Interface: 8-lane PCIe Gen3
  - Up to 7.5 GB/s
- Memory: 8 GB DDR3 SDRAM
- Network Enabled with (2) SFP+ 10 GbE ports
- Power usage: ≤ 25 W (GPU up to 300 W)



Compare PCIe – QPI Interconnect

- Nallatech 385 PCIe vs. Intel® Xeon®+FPGA QPI
- Both Intel® Stratix® V A7 FPGA with 256 DSPs
- Programming model: OpenCL
- Reconstruct 1'000'000 photons

RICH Kernel



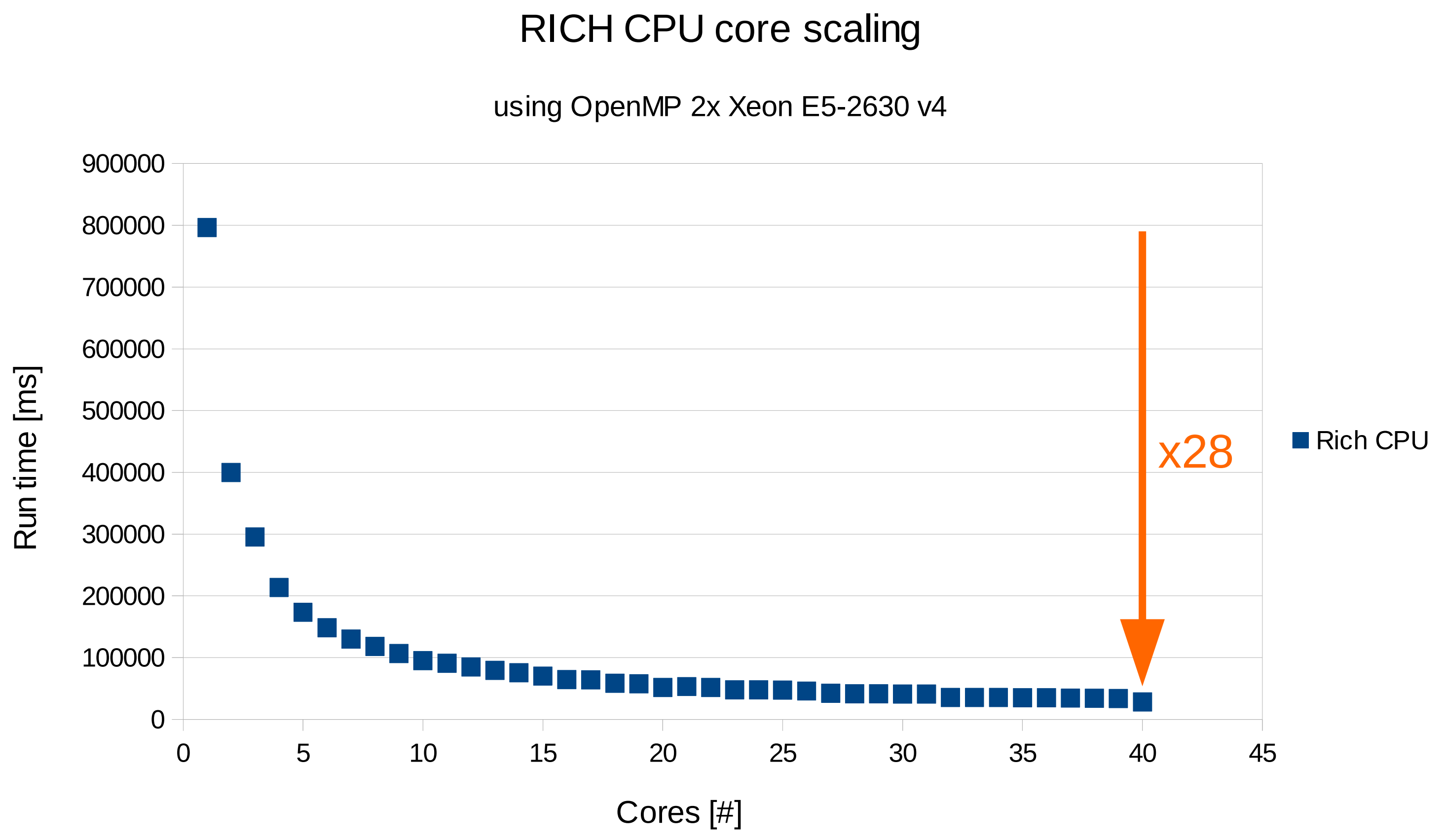
Christian Färber,
CERN EP-ESE electronics seminar, Geneva – 12.12.2017

Nallatech 385A Board

- FPGA: Intel® Arria® 10 GX 1150 FPGA
  - 427'200 ALMs, 1'708'800 Registers
  - 1'518 DSPs
- Programming model: OpenCL
- Host Interface: 8-lane PCIe Gen3
  - Up to 7.9 GB/s
- Memory: 8 GB DDR3 SDRAM
- Network Enabled with (2) QSFP 10/40 GbE ports
- Power usage: full FPGA firmware ~ 40 W

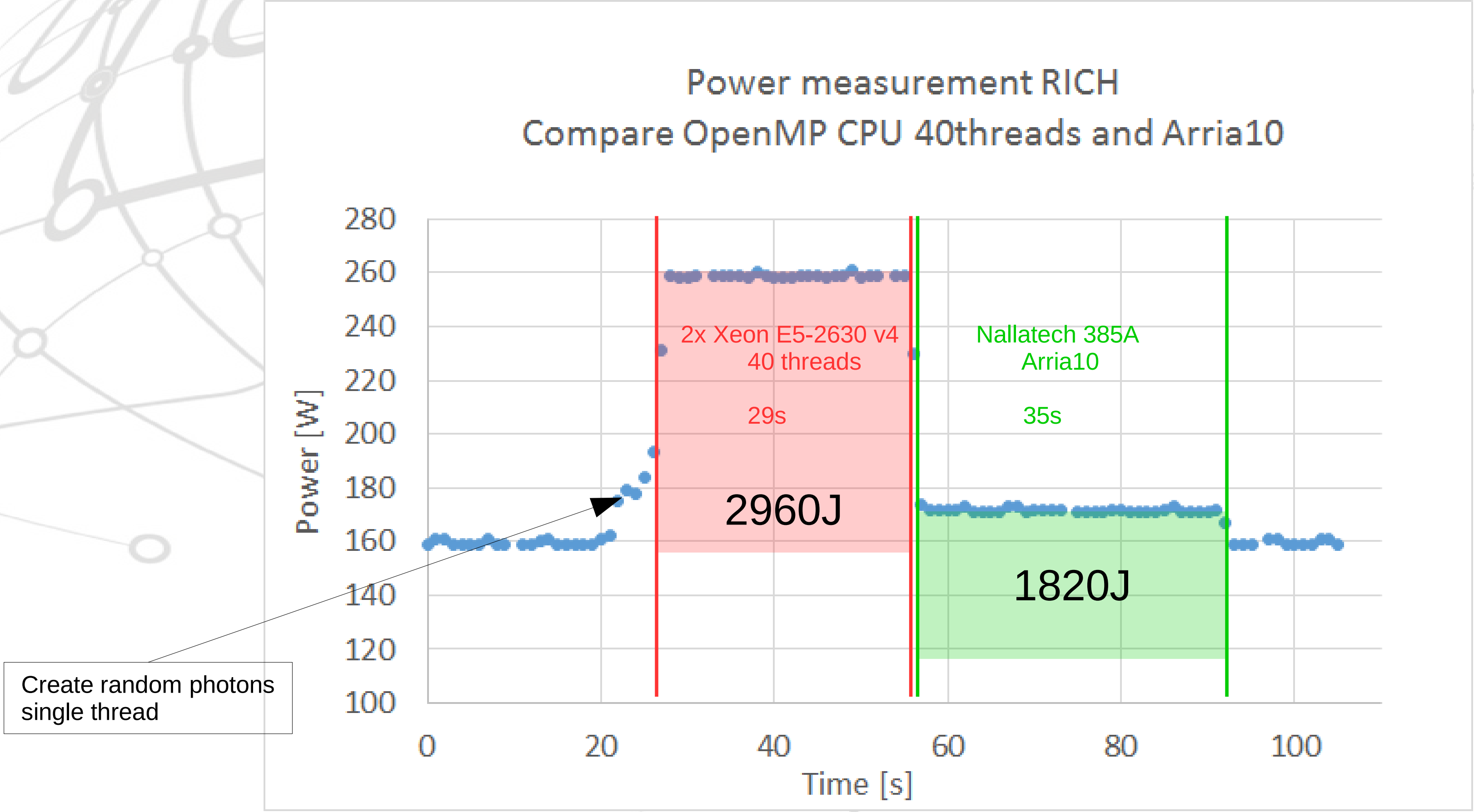
(CERN techlab)

RICH w/o Nallatech 385A faster OpenMP



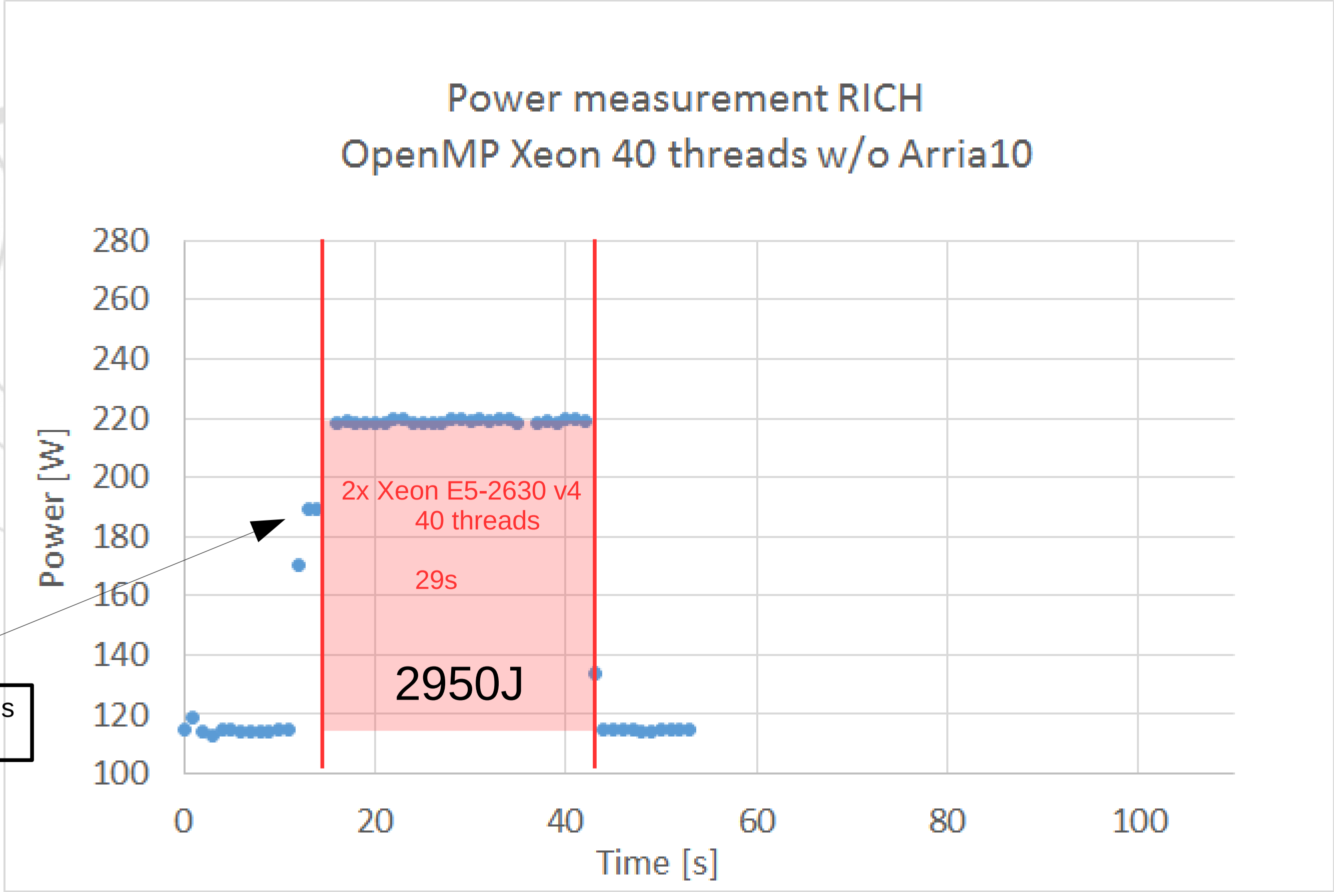
16777216 random photons
Multi loop factor: 160
Used CPU threads: 40

RICH with Nallatech 385A



16777216 random photons
Multi loop factor: 160
Used CPU threads: 40

RICH w/o Nallatech 385A OpenMP



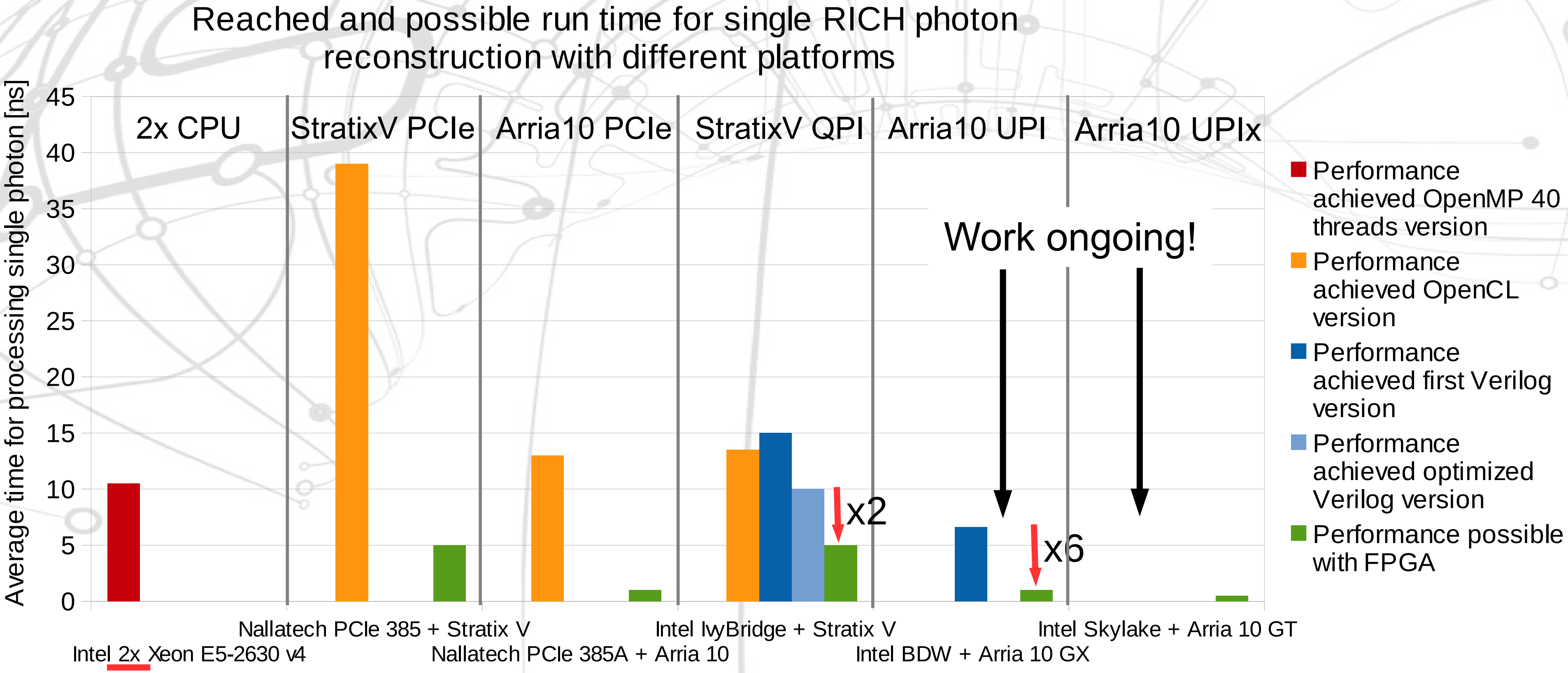
Create random photons
Also with 40 threads

16777216 random photons
Multi loop factor: 160
Used CPU threads: 40

Compare energy consumption

- Processing: 2.7x10^9 photons
  - 2x Xeon® E5-2630 v4 using
    40 threads OpenMP
    no vectorization
    => 29 s x 102 W = 2960 J
  - 1x Arria® 10 GX 1150 GX ↓x1.6
    => 35 s x 52 W = 1820 J
  - FPGA uses 40 W idle + ~12 W single thread
    pushing data into PCIe card
    - Check for better firmware to avoid idle state
    - Use vectorization and OpenCL

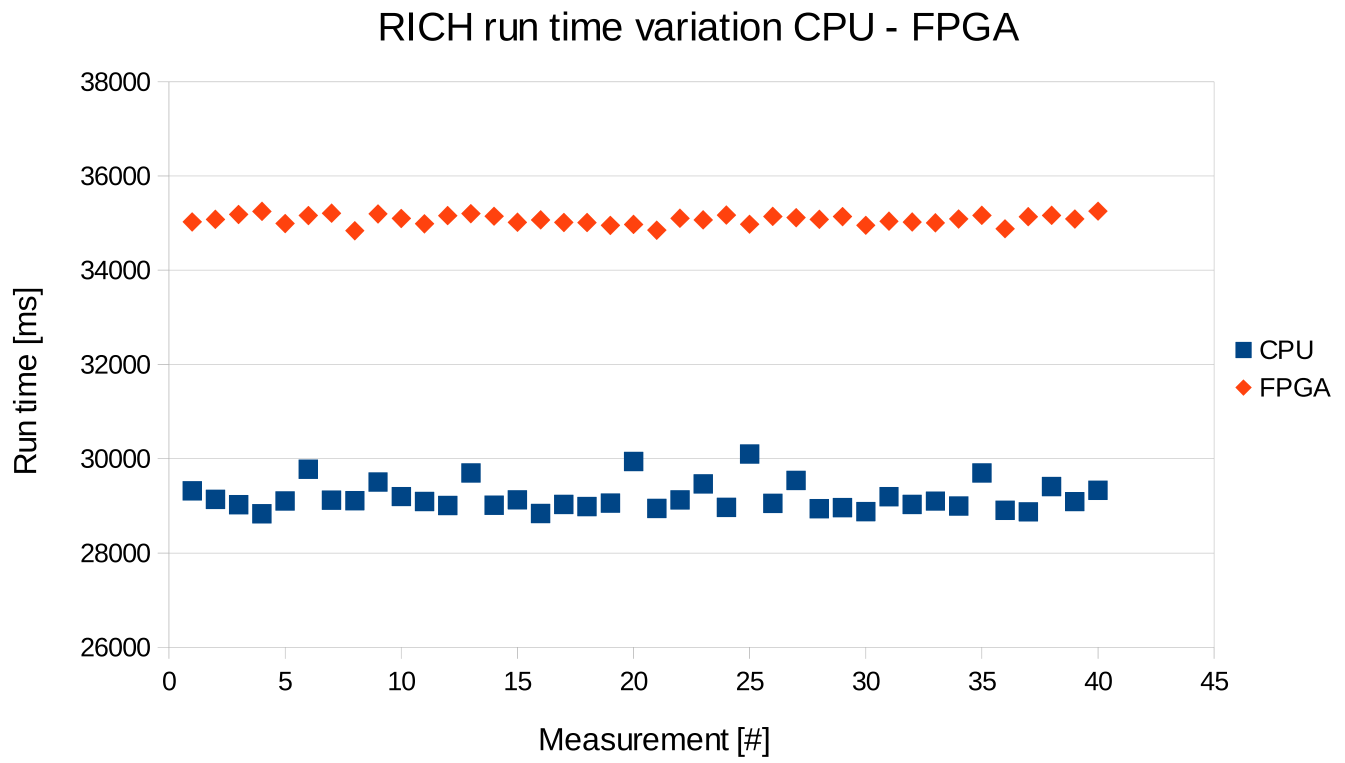
Reached and possible run time for RICH photon reconstruction



- The difference between reached and possible time is due to the limitation by the bandwidth between CPU and FPGA, in both cases the FPGA could process the photons faster. The same case is with the PCIe accelerator, but even worst
- The bandwidth gap could be reduced by caching, for RICH kernel possible
- Between Ivy Bridge and BDW the bandwidth improved by a factor 2

Compare run time variation

- Stddev: CPU : 1.06 %
  FPGA: 0.29 %



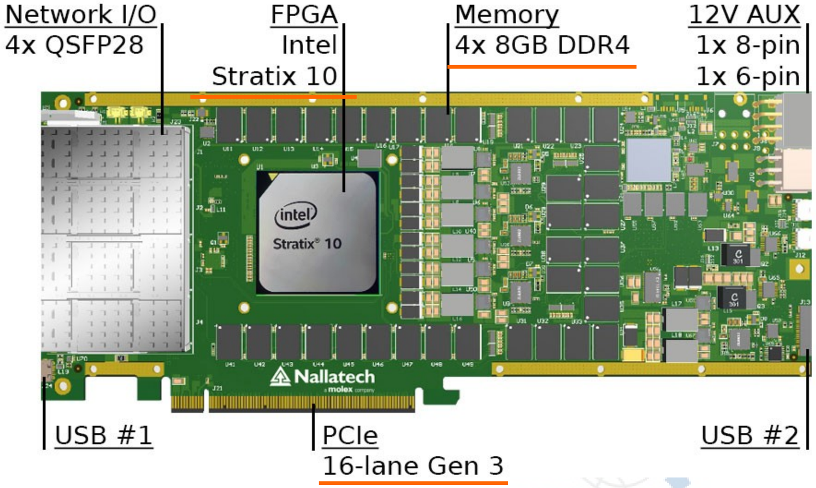
- FPGA runtime is more predictable
- Very important for safety critical control systems

Future Tests

- Implement additional CERN algorithms
  - Tracking - Kalman filter, CNNs
- Compare performance with Intel® Xeon®+FPGA system with Skylake + Arria® 10 FPGA
  - Waiting for missing software and firmware
  - Power measurements
- Compare Verilog vs. OpenCL
- Longterm

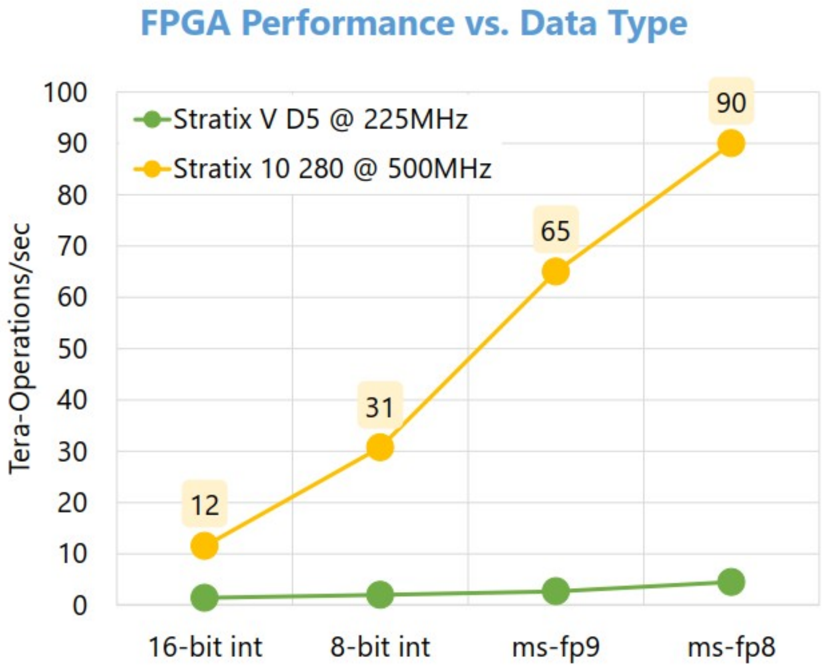
  Measurements of Stratix10 PCIe accelerators and Intel® Xeon® + Stratix10

Nallatech:520 ~10 TFlops



FPGA-based CNN Inference

- For CNN inference single precision is not always needed
- Take advantage of using precision as needed on FPGA
- This increases the operations per second dramatically
- This could be interesting for Monte Carlo production (e.g. Geant V)



Source: FPGA Datacenters - The New Supercomputer, Andrew Putnam – Microsoft Catapult_ACAT_2017_Public

FPGA development

- FPGA potential for general compute acceleration increased a lot with Arria10 and the hardened floating point DSP blocks
  - Future FPGAs will have sev. 10'000 of these DSPs (nowadays already ~6k)
- FPGA transceivers will make huge bandwidth into chip possible, tightly coupled to RAM
- Programming model is changing now to using mostly HLS and OpenCL even for standard FPGA designs
  - Intel recommends to use HLS for Stratix10

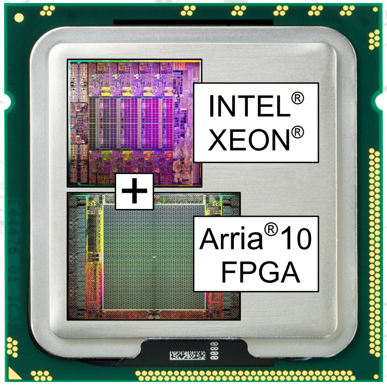
Challenges to use FPGA accelerators

- Compute heavy blocks have to be identified to be ported to the FPGA
- For PCIe accelerators an off-load model is used (larger latency)

  → Intel® Xeon® + FPGA advantage (streaming)
- Kernel size limited by FPGA resources
  - Intel will change programming time from O(s) to O(us) in the future, which makes kernel swapping during runtime practical
-

Summary

- Results are very encouraging to use FPGA acceleration in the HEP field



- Comparing the energy consumption with CPUs show better performance for FPGAs (getting a greener CERN computing ?)
- Programming model with OpenCL very attractive and convenient for HEP field, HLS now also available
- Also other experiments want to test the usage of the Intel® Xeon®+FPGA with Arria10
- High bandwidth interconnect coupled with Arria® 10 FPGA suggests excellent performance per Joule for HEP algorithms! Don’t forget Stratix® 10, … !

Thank you