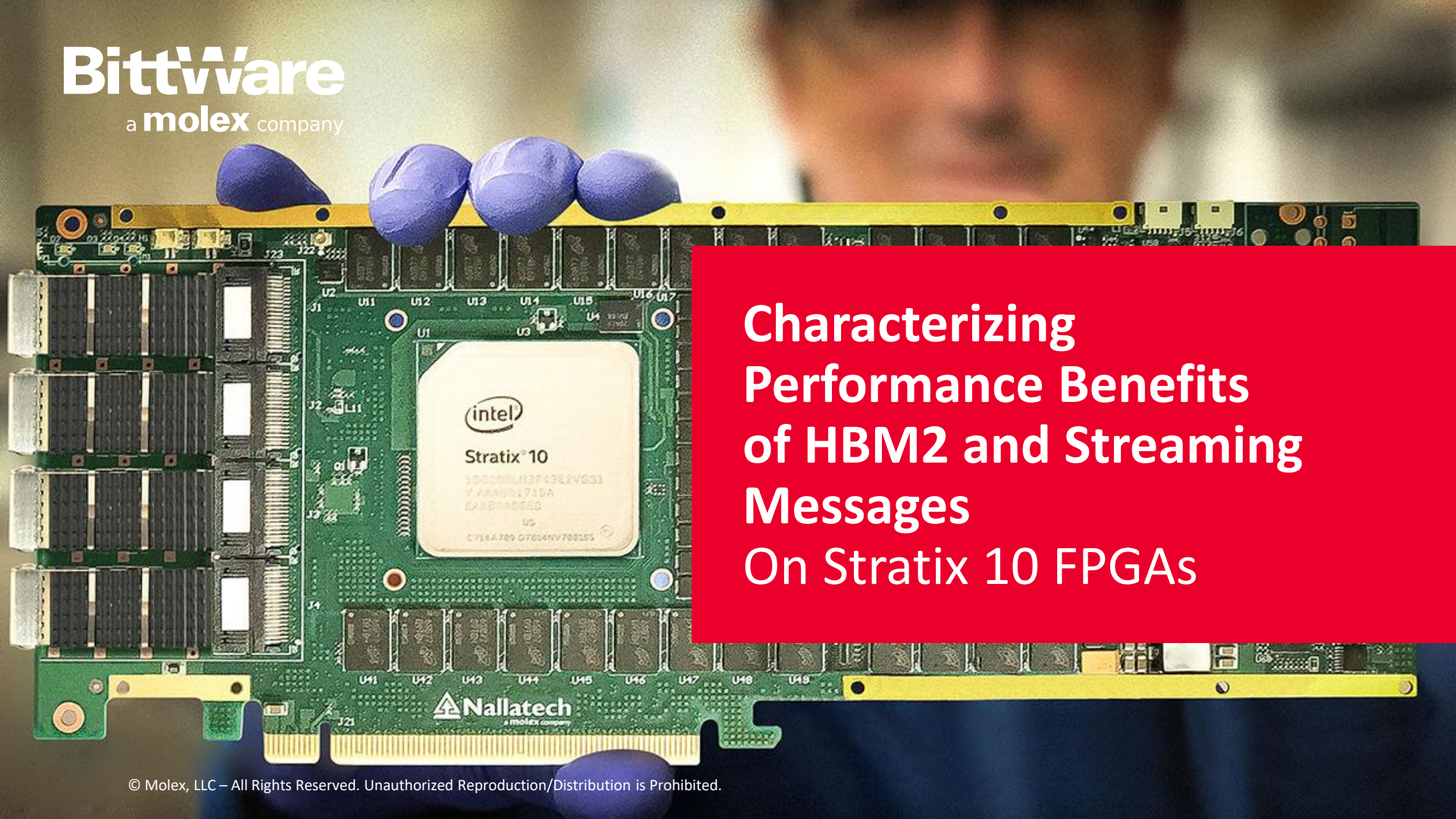
BittWare
a molex company



Characterizing Performance Benefits of HBM2 and Streaming Messages

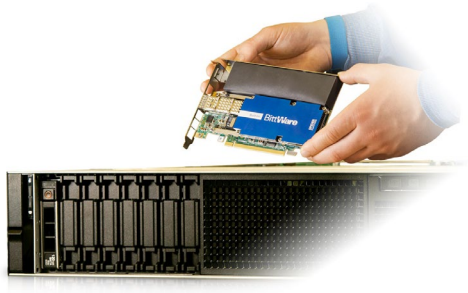
On Stratix 10 FPGAs

BittWare

a molex company

- Part of Molex Datacom & Specialty Solutions BU
- **30 years** FPGA heritage
- Four key segments:
  - **Compute**
  - **Network**
  - **Storage**
  - **Signal Processing**
- Application enablement and benchmarking
- Deliver custom solutions featuring Intel® FPGAs
- Investing in OpenCL BSPs and application-level software/IP to complement HW

FPGA Boards



Integrated Servers

Customization

Application Enablement

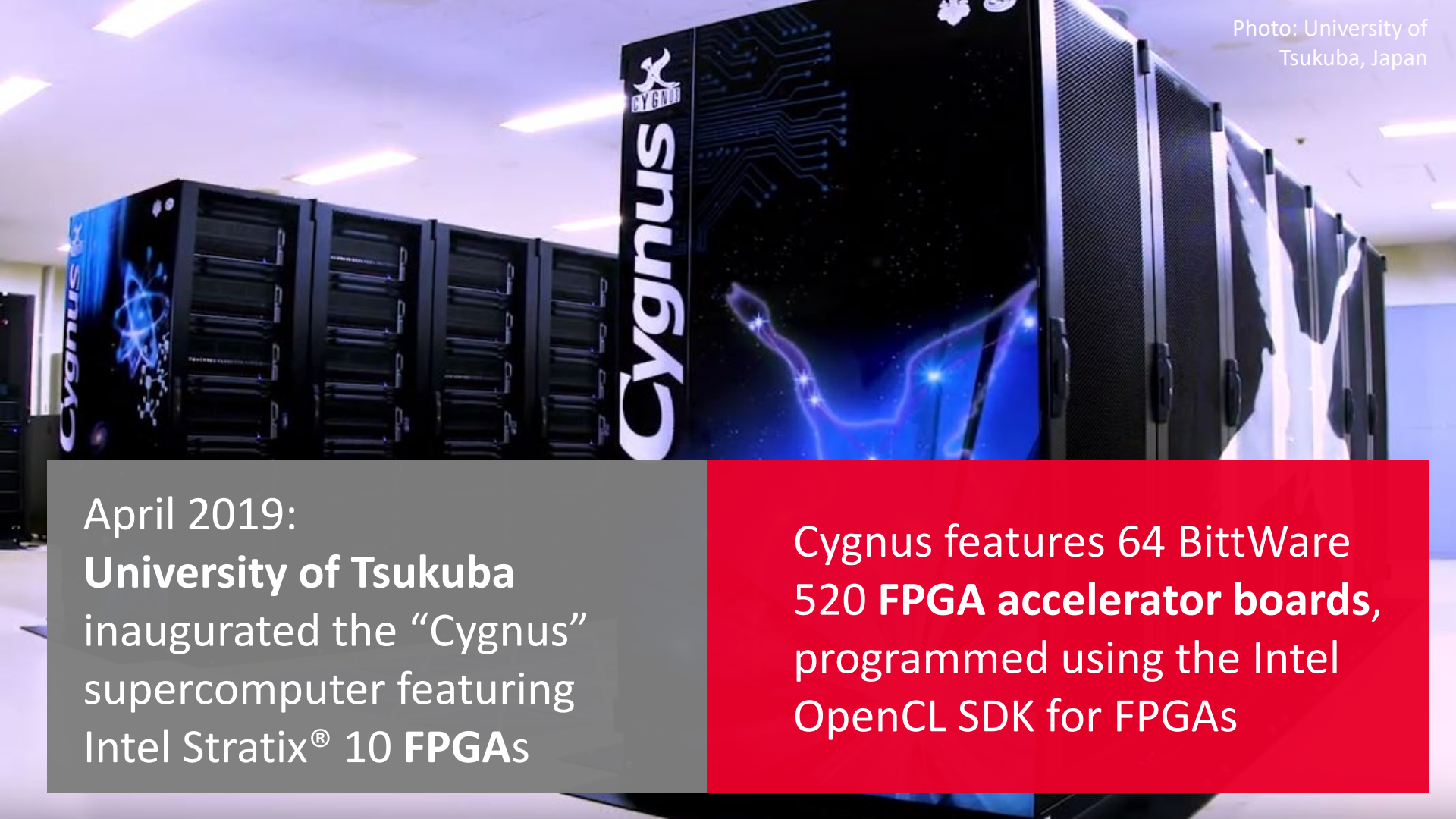


Photo: University of Tsukuba, Japan

April 2019:
University of Tsukuba inaugurated the “Cygnus” supercomputer featuring Intel Stratix® 10 **FPGA**s

Cygnus features 64 BittWare 520 **FPGA accelerator boards**, programmed using the Intel OpenCL SDK for FPGAs

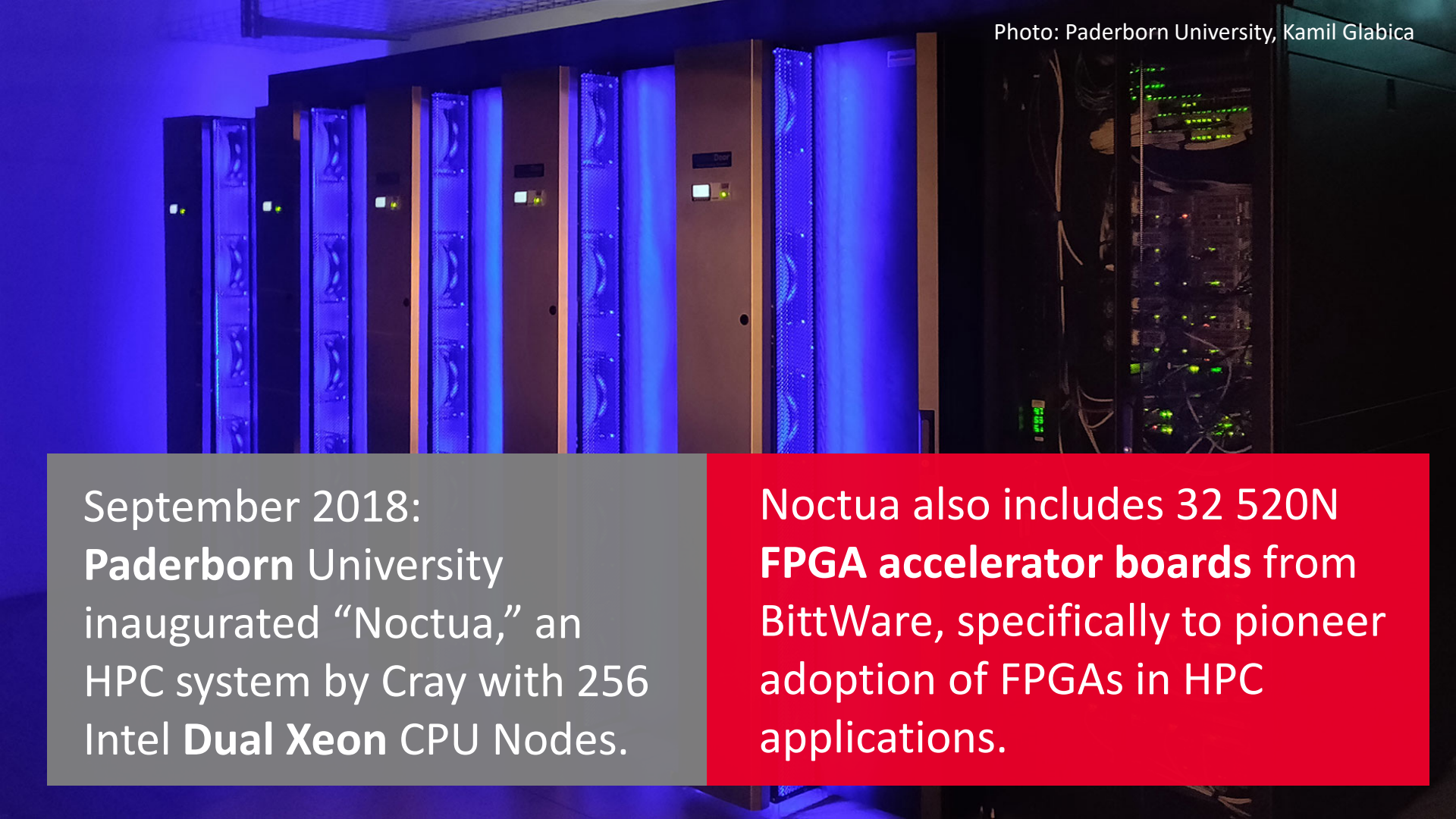


Photo: Paderborn University, Kamil Glabica

September 2018:
Paderborn University inaugurated “Noctua,” an HPC system by Cray with 256 Intel **Dual Xeon** CPU Nodes.

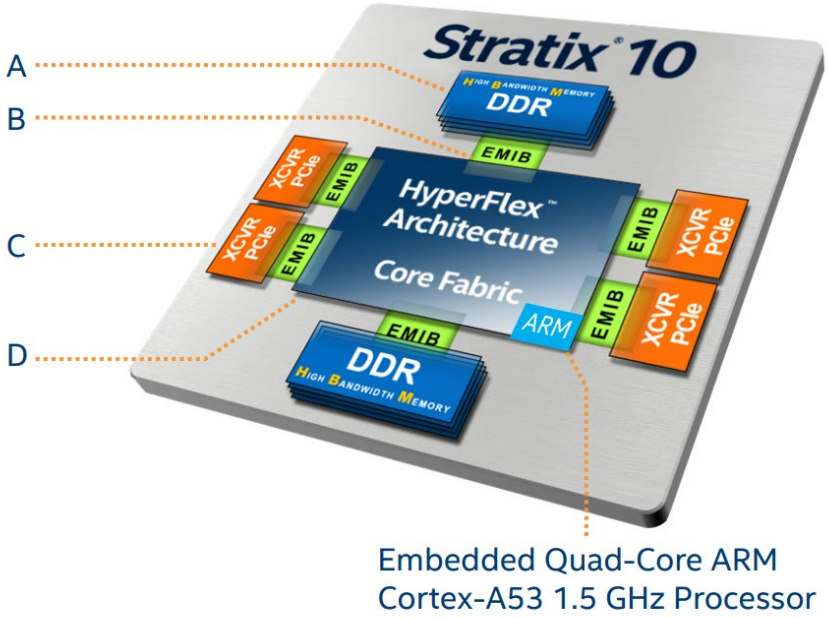
Noctua also includes 32 520N **FPGA accelerator boards** from BittWare, specifically to pioneer adoption of FPGAs in HPC applications.

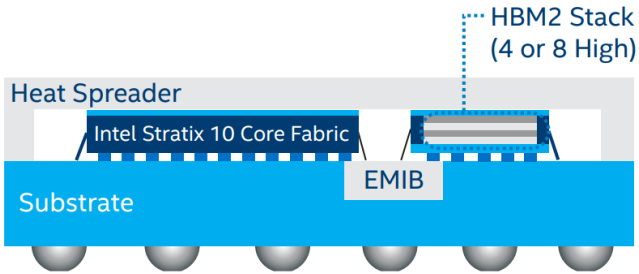
Application Enablement

OpenCL

- Analyze applications at a system level
- Identify where FPGAs provide value
- Generate paper study to estimate potential performance improvements
- Port code and optimize
- Benchmark vs. competing solutions
- Optimize source code executing on hardware
- Deliver of full turnkey solution (cloud/on-premise)
- Make customer self-sufficient (tools, training)

About HBM2 on Stratix 10





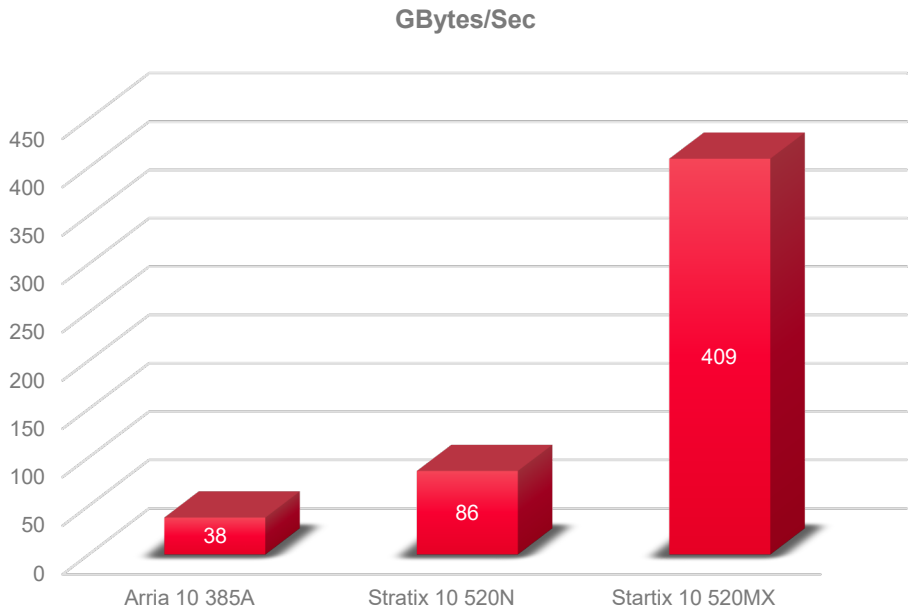
BittWare
a molex company

Characterizing the performance benefits of HBM2

- What FPGA applications benefit from increased external memory bandwidth, but are not suitable for other high bandwidth devices such as GPUs?
- Possible Answers?
  - Problems with unusual data access patterns that break cache structure of other technologies
  - Problems that use unusual data types, e.g. reduced precision, posits, etc

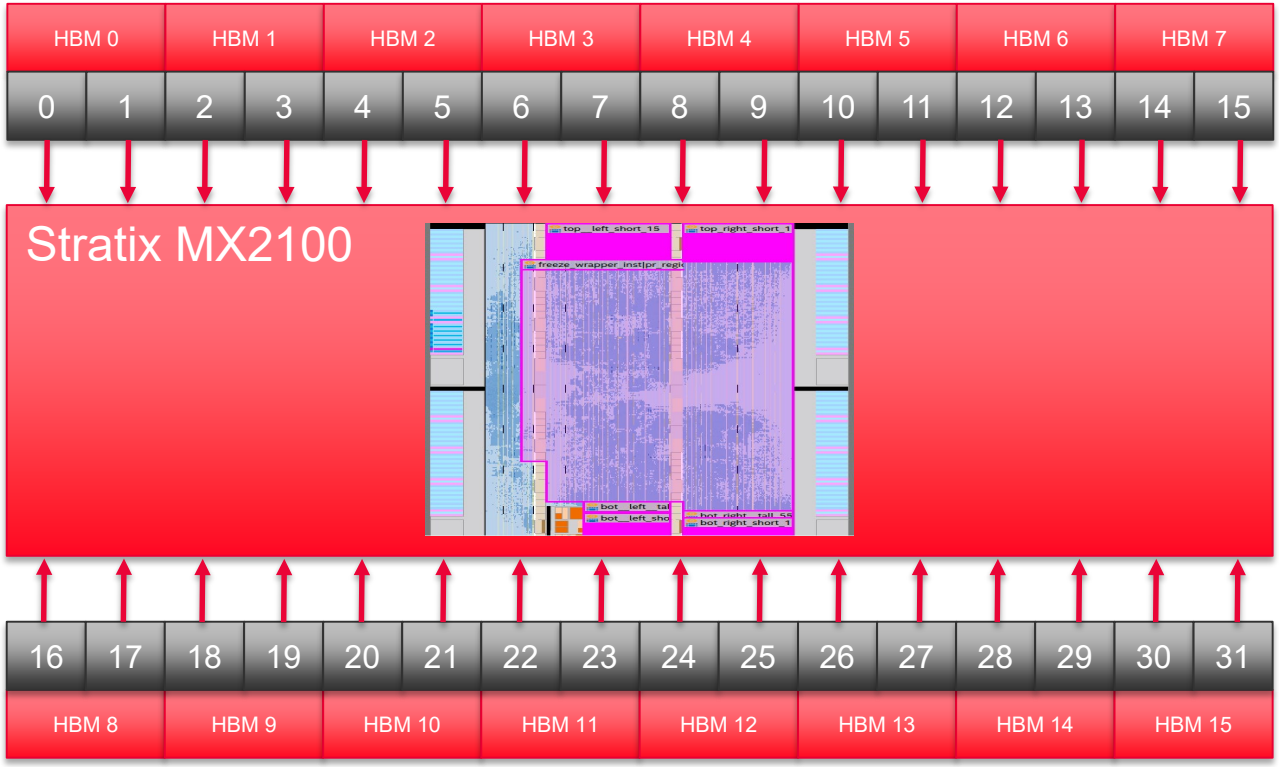
MX HBM FPGA Configuration

- HBM provides a 4x performance boost versus previous technologies
- HBM
  - 16 DDR Banks split into 32 ports
  - 2 pseudo ports for each bank
  - Total bandwidth (-2) 409 GBytes/sec
  - No cross bar between HBMs
    - Can be created by user code at the cost of device resources
  - 16 GBytes of data



BittWare
a molex company

HBM infrastructure on MX



BittWare
a molex company

Achieving highest performance (OpenCL)

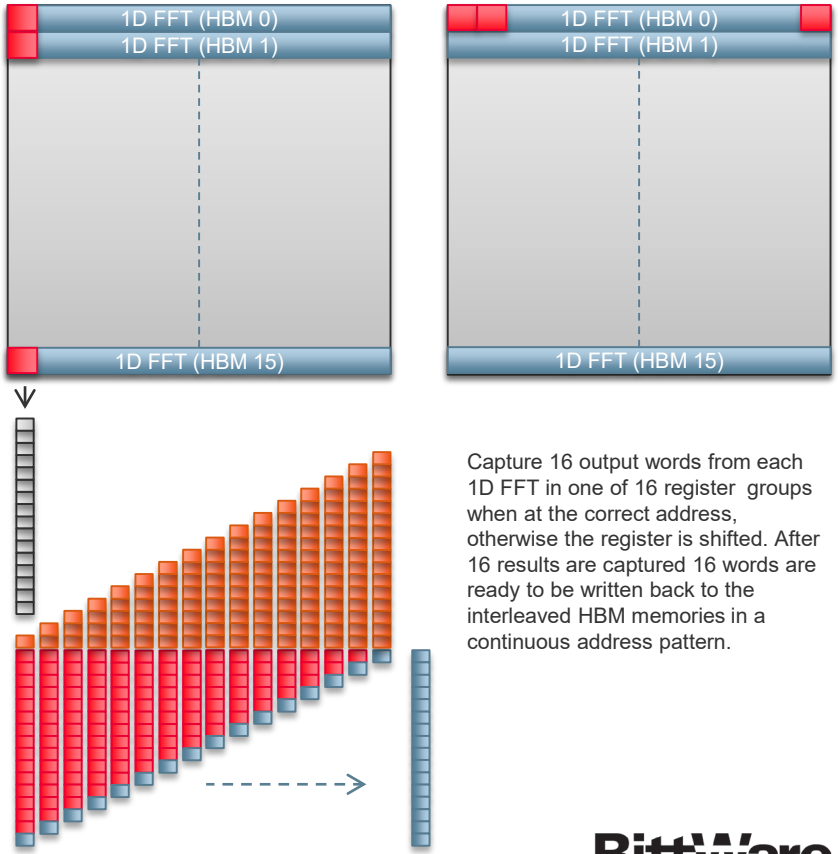
- HBM memory interfaces run at 400 MHz for this device speed grade
  - Kernel clock must 400 MHz or greater to achieve maximum bandwidth
    - Hyper-flex pipelining needs to be enabled
- Memory controllers most efficient when burst 16 or more words
  - 1 word is 32 Bytes
  - True for all DDR memory interfaces

Extracting peak performance 2D FFT use case

- FFT's are memory bound on standard non HBM device
  - Fully pipelined FFT (1024 tap) requires ~ 0.16 Bytes/Flop/Clock
  - Theoretical peak flop for MX2100 = 3.17 Tflops
  - ~500 GBytes/Sec to saturate all DSP logic
- Perform multiple parallel 1D FFTs
  - Stripe input rows across all available HBMs
  - 16 parallel 1D FFTs each reading and writing 16 Bytes per clock cycle to HBM working on their own row of data

Transpose problem

- Striping memory causes complexities for the transposition part of a multi-dimensional FFT
- 2D FFT requires transpose of rows to columns, however columns are striped across multiple memory ports with no shared connectivity.
- Solution is to create a sliding window to move HBM data from rows to columns
  - Sliding windows are very efficient in FPGAs

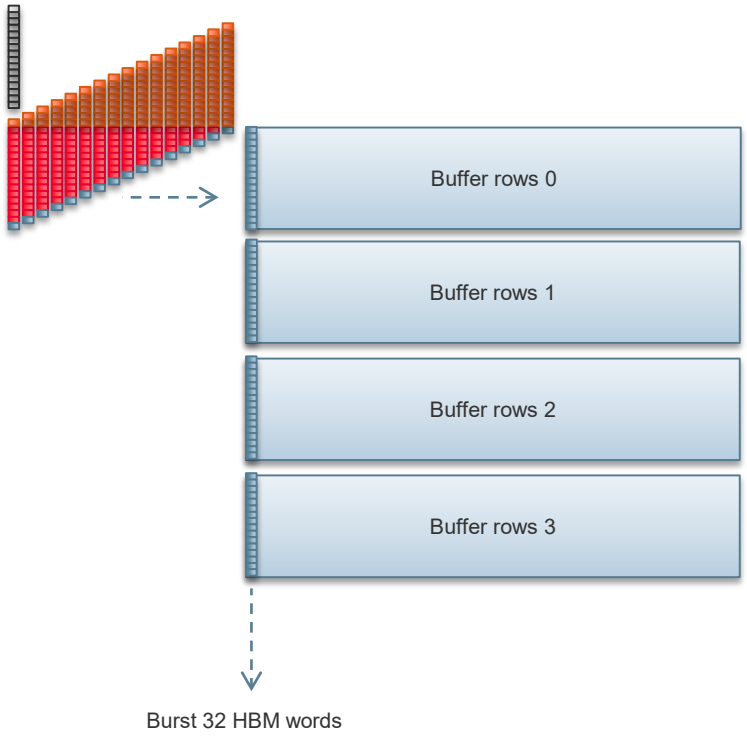


Capture 16 output words from each 1D FFT in one of 16 register groups when at the correct address, otherwise the register is shifted. After 16 results are captured 16 words are ready to be written back to the interleaved HBM memories in a continuous address pattern.

BittWare
a molex company

HBM burst requirements

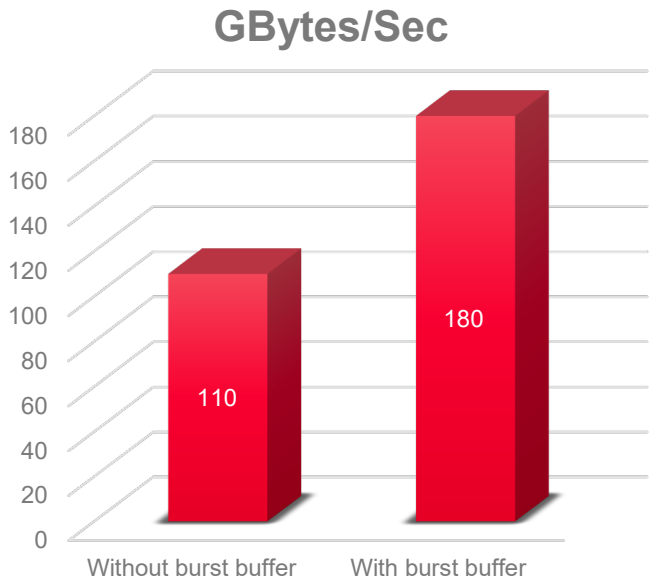
- Use local memories to buffer enough data to enable a burst 16 words
- Requires 4 lots to of 16 outputs generated by the HBMs to be cached locally
  - Requires a double buffer implementation using local M20K memories
  - Transpose output is then 64 complex numbers or 16 HBM words
- HBM performance is as close to 100% as it can be



BittWare
a molex company

Striped HBM Transpose Performance

- HBM bandwidth 180 Gbytes/Sec ~ 90% peak
  - Only half available bandwidth utilised in this example, (beta version of OpenCL BSP)
- FPGA logic used to store enough intermediate results, prevents transpose degrading performance.



BittWare
a molex company

Conclusion

- HBM memory provides a significant performance boost to memory limited FPGA designs
- Using HBM memories requires careful consideration of data access patterns if data is spread across HBMS
- Care needs to be taken to ensure data can be burst in large enough blocks to hit peak performance
  - For applications that are not bandwidth limited, but require access to the whole address space, this will require users to code multiplexing across all 32 ports. This is not trivial to do efficiently

HBM Enabled Applications?

Application	Complex Access Patterns	Bit manipulation or unusual data types
Multi-dimensional FFT	✓	
Compression		✓
Cryptography		✓
Bioinformatics		✓
Finite element stencil	✓	

Who Am I?

Tiziano De Matteis

- Ph.D. and PostDoc at **University of Pisa** (Italy)
- Currently, PostDoc Researcher in the **Scalable Parallel Computing Lab** (ETH, Zurich)

My principal research interests:

- FPGAs for HPC: tools and libraries for improving HPC programming productivity;
- Parallel Data Stream Processing;
- Energy Awareness in Parallel Computing;

BittWare
a molex company

Streaming Message Interface

- Modern FPGA Chips have high-performance serial link network connections;
- Necessary for adoption in data center and super-computers;
- **Distributed Memory Programming** on **Reconfigurable Hardware** needed to scale to multi-node.

When FPGAs are deployed in a distributed setting, communication is typically handled either by going through the host machine or by streaming across fixed device-to-device connections

github.com/spcl/smi

At SPCL (ETH Zurich) we designed Streaming Messages:

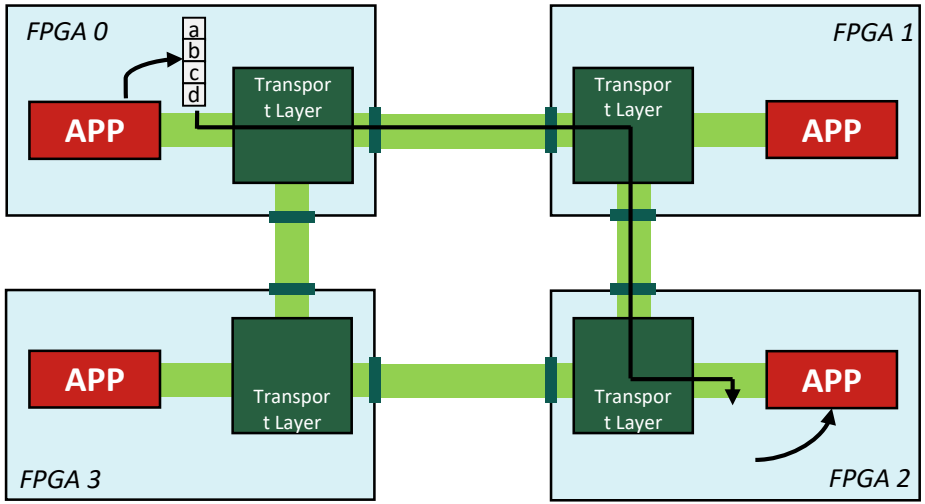
- a distributed memory programming model for FPGAs that unifies message passing and hardware programming (i.e., pipelined codes) with HLS;
- an interface (SMI), an HLS communication interface specification for programming streaming messages in distributed memory multi-FPGA systems

T. De Matteis, J. de Fine Licht, J. Beránek, T. Hoefler. “Streaming Message Interface: High-Performance Distributed Memory Programming on Reconfigurable Hardware”. SC’19

Existing communication models: Message Passing

With Message Passing, *ranks* use local buffers to send and receive information from other pairs

```
for (int i = 0; i < N; i++)
    buffer[i] = compute(data[i]);
SendMessage(buffer, N, my_rank + 2);
```



Flexible: End-points are specified dynamically

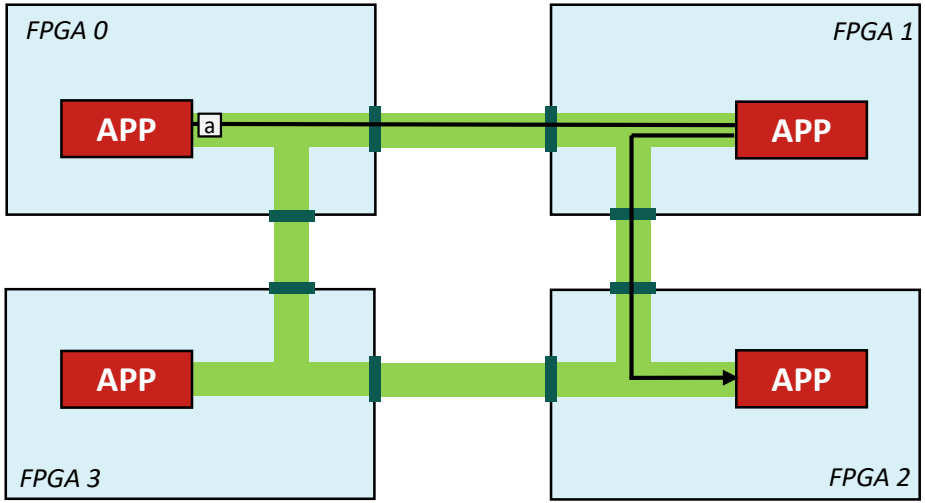
Bad match for HLS programming model:

- relies on bulk transfers;
- (potentially dynamically sized) buffers required to store messages.

Existing communication models: Streaming

Data is streamed across an inter-FPGA in a pipelined fashion

```
// Channel fixed in the architecture
for (int i = 0; i < N; i++)
   stream.Push(compute(data[i]));
```



Communication model **fits** the HLS programming model

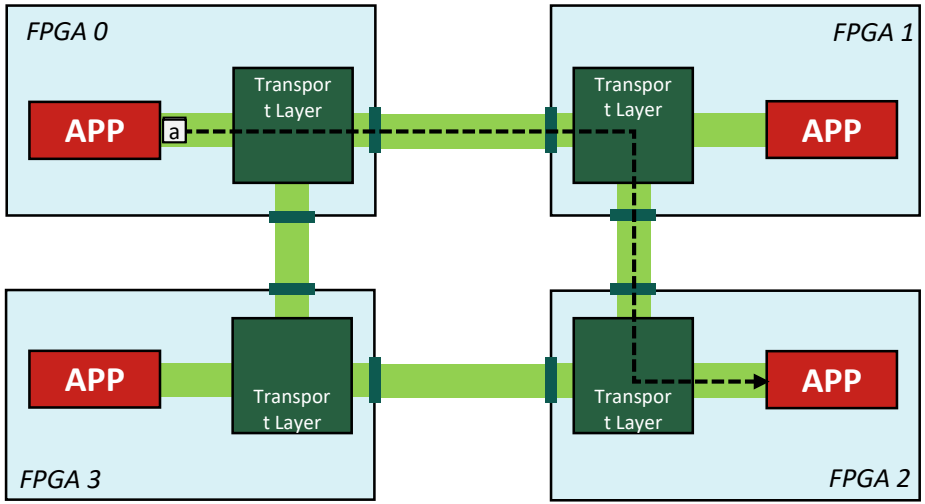
Inflexible, the user must:

- construct the exact path between end-points;
- handle all the forwarding logic.

Our proposal: Streaming Messages

Traditional, buffered messages are replaced with pipeline-friendly *transient channels.*

```
Channel channel(N, my_rank + 2, 0); // Dynamic
target
for (int i = 0; i < N; i++)
   channel.Push(compute(data[i]));
```



Combines the best of both worlds:

- Channels are **transiently** established, as ranks are specified dynamically
- Data is pushed to the channel during processing in a **pipelined** fashion

Key facts:

- Each channel is identified by a ***port***, used to implements an hardware streaming interface
- All channels can operate in parallel
- Ranks can be programmed **either in a *SPMD* or *MPMD* fashion**

Streaming Message Interface

A communication interface for HLS programs that exposes primitives for both point-to-point and collective communications.

Point-to-Point channels are unidirectional FIFO queues used to send a message between two endpoints:

```
void Rank0(const int N, /* ...args... */) {
  SMI_Channel chs = SMI_Open_send_channel(  // Send to
        N, SMI_INT, 1, 0, SMI_COMM_WORLD); // rank 1

  #pragma ii 1 // Pipelined loop
  for (int i = 0; i < N; i++) {
    int data = /* create or load interesting data */;
    SMI_Push(&chs, &data);
} }
```

```
void Rank1(const int N, /* ...args... */) {
  SMI_Channel chr = SMI_Open_recv_channel(// Receive from
      N, SMI_INT, 0, 0, SMI_COMM_WORLD); // from rank 0
  #pragma ii 1 // Pipelined loop
  for (int i = 0; i < N; i++) {
    int data;
    SMI_Pop(&chr, &data);
    // ...do something useful with data...
} }
```

Streaming Message Interface

A communication interface for HLS programs that exposes primitives for both point-to-point and collective communications.

Collective channels are used to implement collective communications. SMI defines `Bcast`, `Reduce`, `Scatter` and `Gather`

```
void App(int N, int root, SMI_Comm comm, /* ... */) {
  SMI_BChannel chan = SMI_Open_bcast_channel(
          N, SMI_FLOAT, 0, root, comm);
  int my_rank = SMI_Comm_rank(comm);
  for (int i = 0; i < N; i++) {
    int data;
    if (my_rank == root)
      data = /* create or load interesting data */;
    SMI_Bcast(&chan, &data);
    // ...do something useful with data...
} }
```

Data elements are sent in order

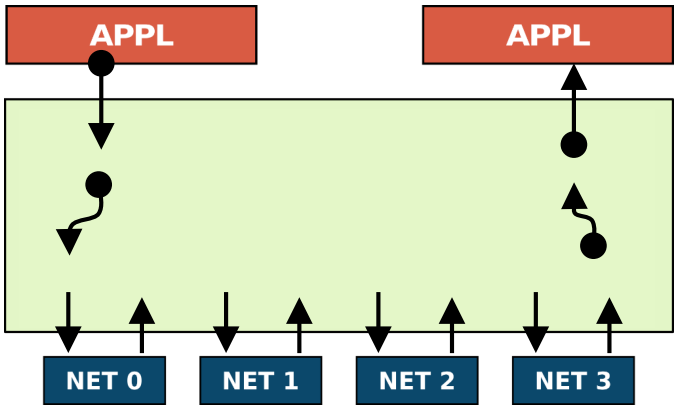
Calls must be pipelined in single clock cycle

Communication is programmed **in the same way** data is normally streamed between intra-FPGA modules

Multiple collectives can execute in **parallel**, provided that they use separate ports

Reference Implementation

We implemented a proof-of-concept HLS-based implementation (targeting Intel FPGA)



Two components:

- **interface** implements the SMI primitives and packs messages in *network packets*
- **transport component** is in charge of routing data between endpoints

Data communications move data through physical connections

- `Port` declared in `Open_channel` primitives are used to lay down the hardware

Each FPGA net. connection is managed by a pair of Communication Kernels (CK)

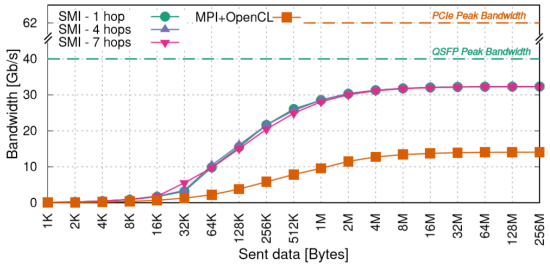
- Each CK has a routing table: **If the network topology changes, we rebuild the routing tables not the entire bitstream**

Key enabler for SMI have been Intel I/O channels and their support in Bittware BSP

Evaluation

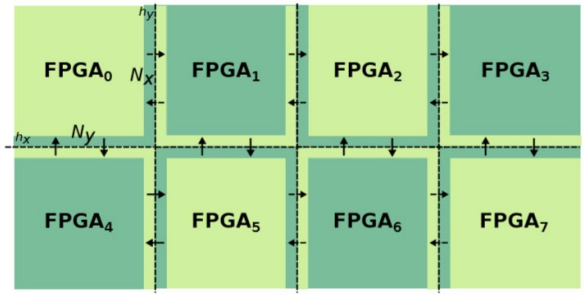
Testbed: 8 Bittware 520N boards (Stratix 10), 2D-Torus, each with 4x 40Gbit/s QSFP, PCI-E 8x

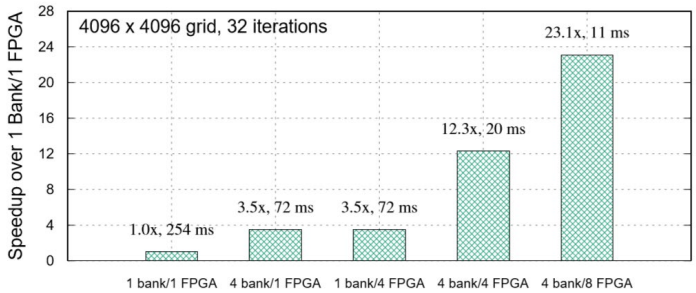
Microbenchmarks: bandwidth/latency over different topology/network distances simply by changing the topology file



MPI+OpenCL	SMI-1	SMI-4	SMI-7
36.61	0.801	2.896	5.103

SPMD program: spatially tiled 2D stencil (**same** bitstream for all the ranks)





We wish to thank the Paderborn Center for Parallel Computing (PC^2) for granting access, support, maintenance, and upgrades on their Noctua multi-FPGAs system.

Stratix 10 brings features like 100G networking and 16GB of on-package **HBM2** memory

HBM2 on 520N-MX

100G Links

OpenCL on FPGAs:

Performance test from CERN on Verilog vs. OpenCL

Faster development — **2.5 months vs. 2 weeks**

Easier development — **3,400 lines vs. 250 lines**

Similar performance — **35x vs. 26-30x acceleration**

OpenCL

OpenCL is also far easier to learn!

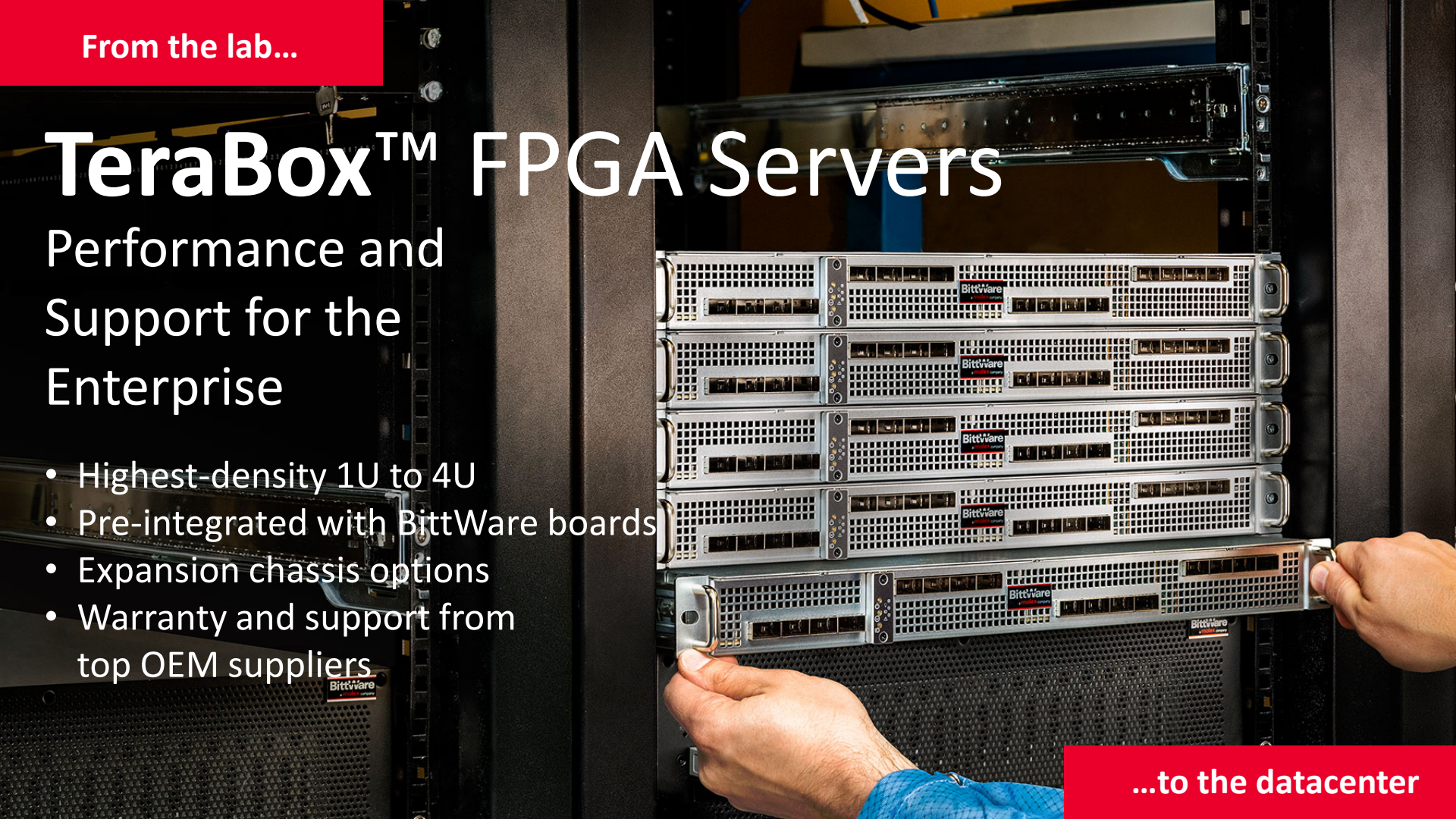
Source: “FPGA Compute Acceleration for High-Throughput Data Processing in High-Energy Physics Experiments,” Christian Färber, CERN Computing Seminar, Geneva 2017

From the lab…

TeraBox™ FPGA Servers

Performance and Support for the Enterprise

- Highest-density 1U to 4U
- Pre-integrated with BittWare boards
- Expansion chassis options
- Warranty and support from top OEM suppliers



…to the datacenter