UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft

High-Performance Computing with FPGAs

Christian Plessl

Paderborn University, Germany
Paderborn Center for Parallel Computing

Paderborn Center for Parallel Computing

IXPUG Workshop – ISC'19 Frankfurt – 20 June 2019

Outline

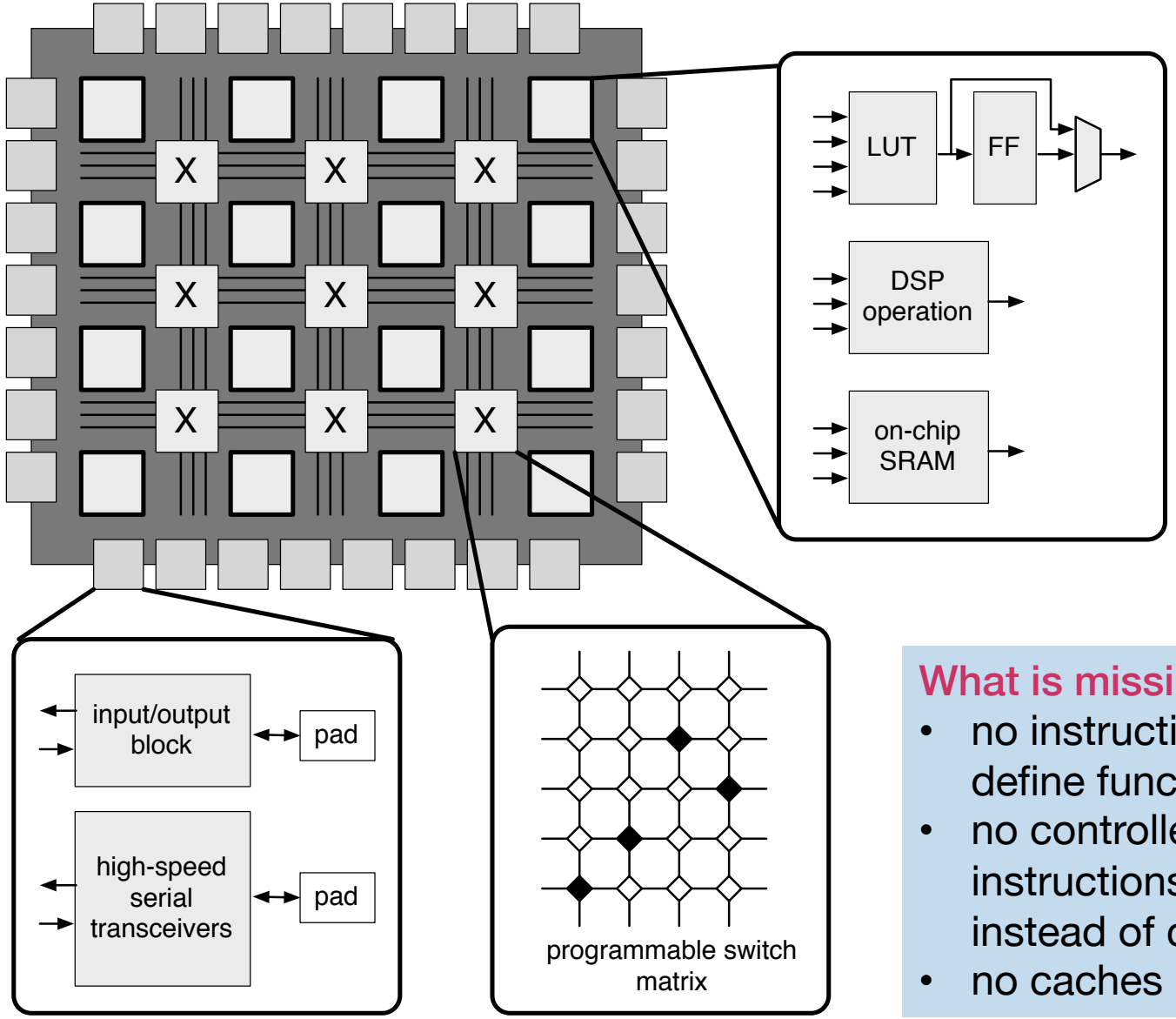
- FPGA Fundamentals
  - architecture
  - execution model
- State of the FPGA Union
- Accelerating scientific applications with FPGAs
  - Discontinuous Galerkin PDE solver for Maxwell's Equations
  - Kernel analysis
  - Implementation with OpenCL and High-Level Synthesis
  - Results
- Scaling FPGA Applications
- Conclusions

FPGA Fundamentals

Noctua Cluster at Paderborn Center for Parallel Computing

- Cray CS500 Cluster System
- 256 CPU nodes
  - 2 x Intel Xeon Skylake Gold 6148, 2 x 20 Cores, 2.4GHz
  - 192 GB RAM
- 100 Gbps Intel Omni-Path network
- 700 TB Cray ClusterStor L300N storage system
- 16 FPGA nodes
  - 2 x Intel Stratix 10 GX2800 (BittWare 520N boards)
    PCIe 3.0 x16, 4 x 8GB DDR4 channels
  - per board 4 QSFP28 ports
  - currently one of worldwide largest academic HPC systems with modern FPGAs

Field Programmable Gate Array (FPGA) – Basic Architecture



millions

thousands

thousands

hundreds

dozens

What is missing in contrast to CPU

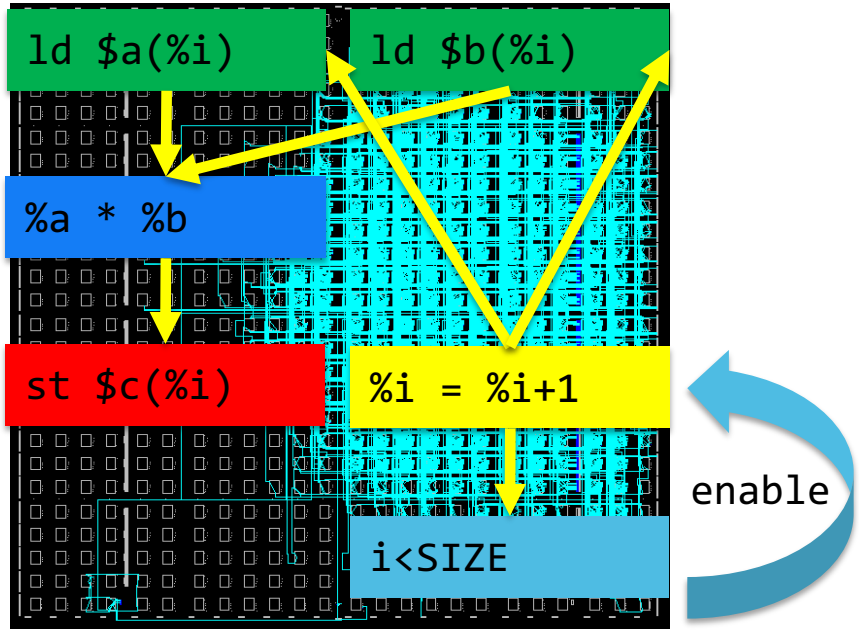
- no instructions, configuration bits define function of each component
- no controllers, no sequence of instructions (computing in space instead of computing in time)
- no caches

Execution Model: Temporal vs. Spatial Computing

```
for (int i = 0; i < SIZE; i++){
    c[i] = a[i] * b[i];
}
```

```
loop:
  ld %a $a(%i)
  ld %b $b(%i)
  %c = %a * %b
  st $c(%i) %c
  %i = %i + 1
  branch i<SIZE: loop
```

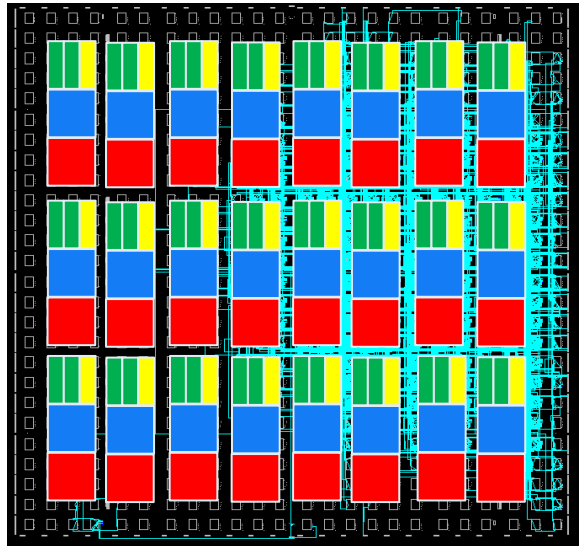
Execution on CPU: repeated sequence of instructions (time-multiplexing of ALU, computing in time)

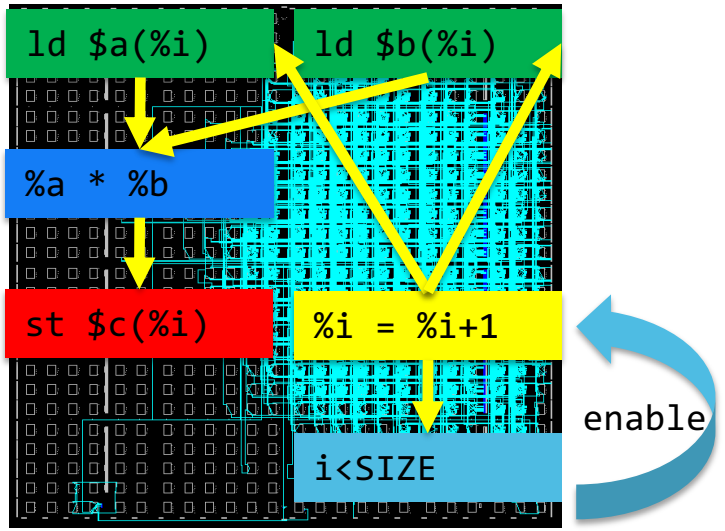


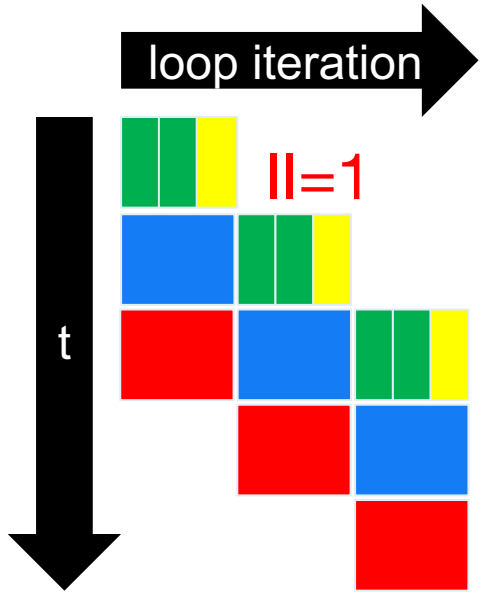
Execution on FPGA: spatial mapping of data-path and control (computing in space)

Optimizations: Pipelining and Data Path Replication

- Primary goal: Use functional units every cycle
- Initiation Intervall (II)
  - describes pipeline fill rate
  - number of clock cycles until new input values can be accepted (target: II = 1)
- Replicate datapath
  - >3900 instances for Stratix 10 GX2880
  - saturate bandwidth







State of the FPGA Union

Highly Capable FPGA Devices

Example: Intel Stratix 10 GX2800 FPGA

- > 900,000 configurable logic blocks
  - up to 4 Boolean functions of 8 inputs
- 5760 hardened arithmetic units (DSP)
  - fixed point and IEEE 754 SP floating-point
- > 11,000 independent SRAM blocks
  - width/depth/ports highly configurable
- integrated DDR4-2666 memory controllers
- 96 serial transceivers, up to 28.3 Gbps
- typically about 300-600MHz
- power consumption 50-225W

100 TERRA-OPS

10 single-precision TFLOPS

20 TB/s internal SRAM bandwidth (full duplex)

300 TB/s communication bandwidth (full duplex)

up to 80 GFLOPS/W

Increasingly Productive FPGA Tools

- Traditional EDA Software
  - hardware synthesis from VHDL, Verilog
  - simulators, place and route

mature but much too cumbersome for general HPC

- General high-level synthesis tools
  - generation of complete accelerators or components from OpenCL or C/C++ with annotations (Intel OpenCL SDK for FPGAs, Xilinx SDAccel)

higher productivity and increasingly good results

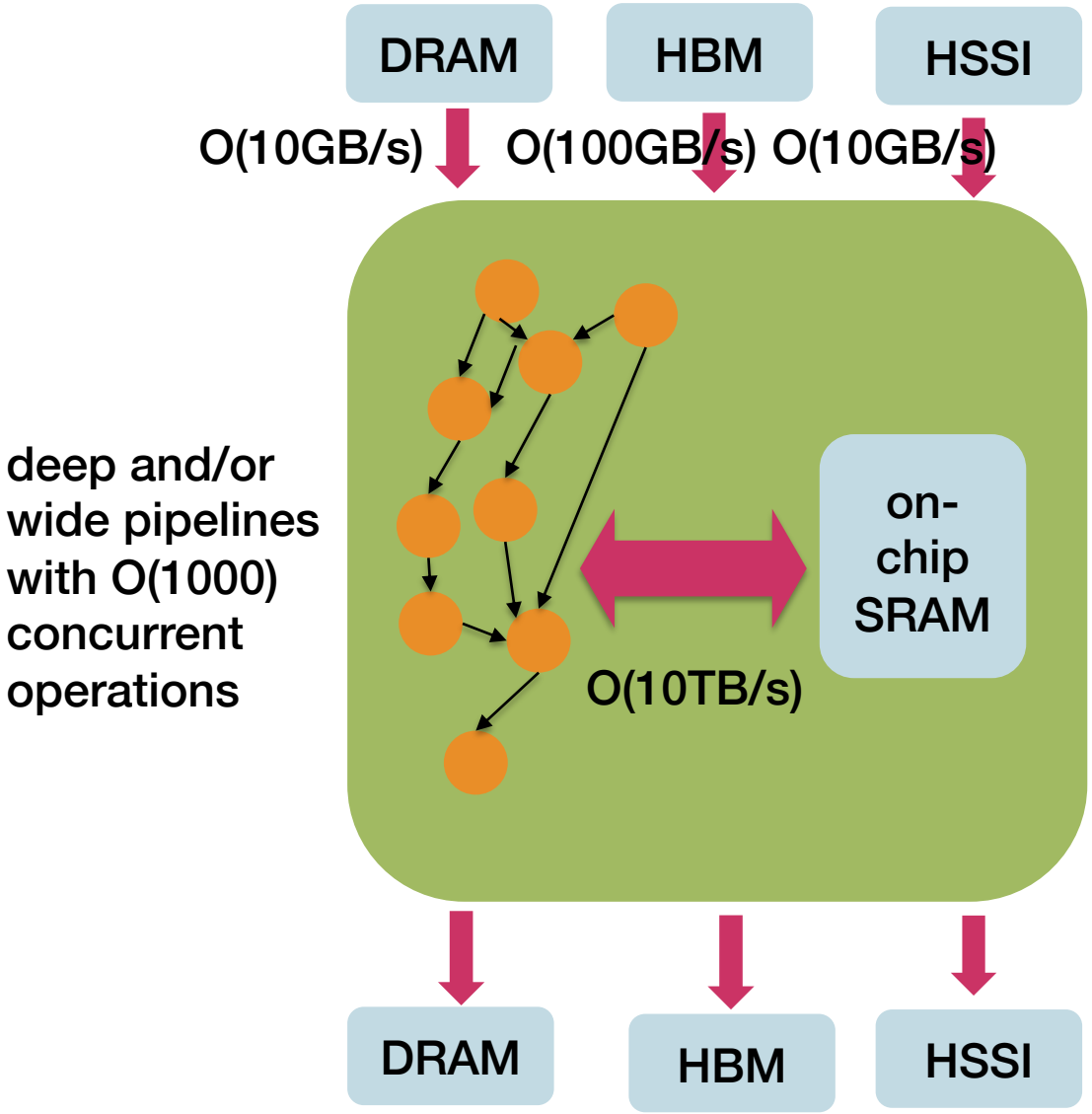
- Domain-specific tools for important niches
  - networking (P4)
  - digital signal processing (MATLAB/Simulink toolboxes)
  - deep learning inference (Xilinx xDNN, Intel DLA)

even higher productivity but narrow scope

- Libraries and ready-to-use applications

almost inexistent for HPC

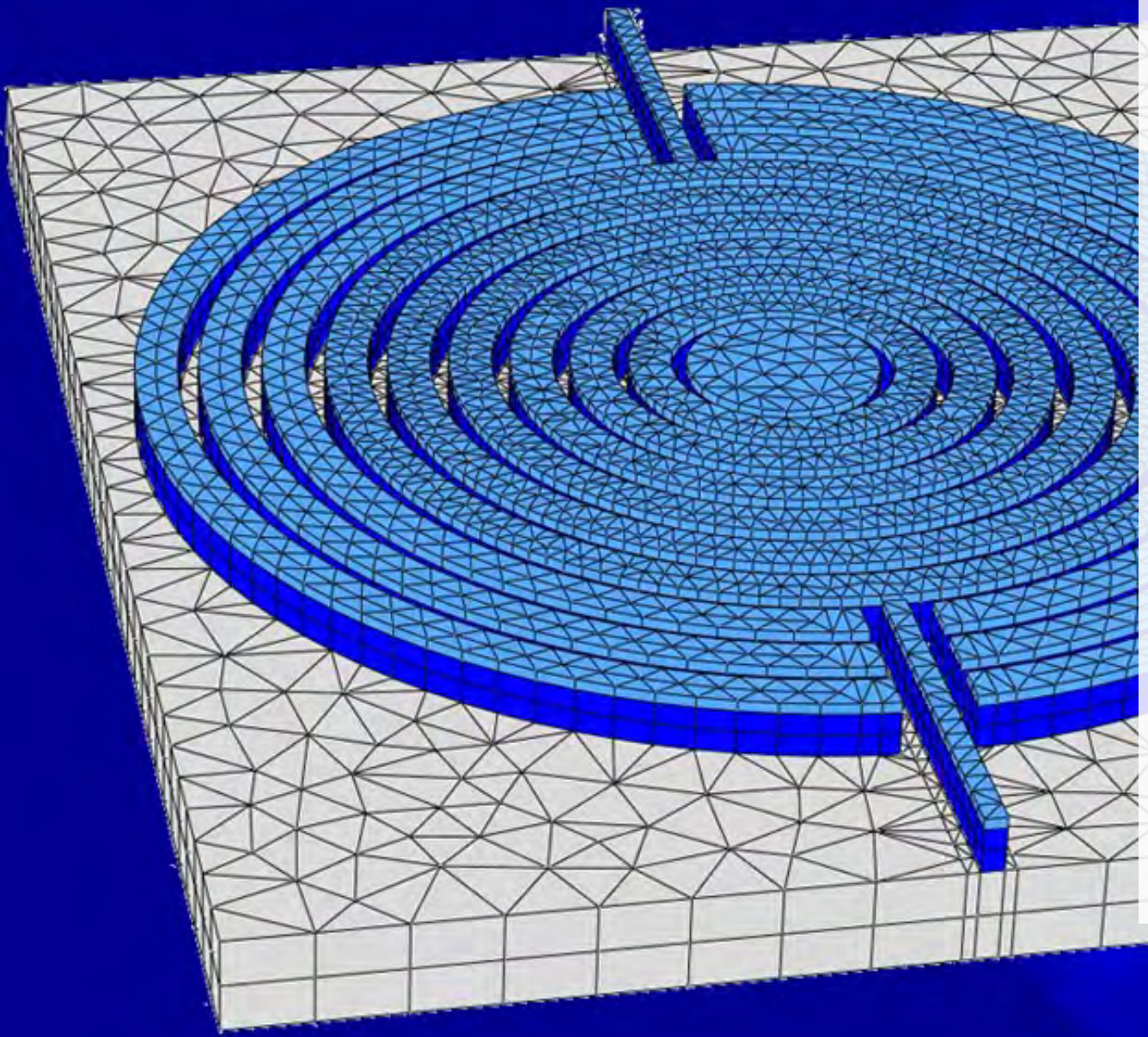
Opportunities for FPGAs



- Compute-bound applications
  - customization of operations and data formats
  - new methods considering FPGA architecture
- Memory-bound applications
  - unrolling and data flow computing with very deep pipelines
  - application-specific, distributed memory architectures
- Latency-bound applications
  - speculative or redundant execution
- I/O-bound applications
  - on-board network interfaces
  - direct FPGA-to-FPGA communication

Accelerating Scientific Applications with FPGAs

Computational Nanophotonics



- Simulation of light in nano-structured materials
  - antennas, polarization filters, wave guides
  - collaboration with Prof. Jens Förstner, Paderborn University
- Discontinuous Galerkin solver for Maxwell equations

Kenter et. al: **OpenCL-based FPGA design to accelerate the nodal Discontinuous Galerkin method for unstructured meshes.** *Proc. Int. Symp. on Field-Programmable Custom Computing Machines (FCCM)*. Apr. 2018.

Discontinuous Galerkin (DG) Method

- Numerical method for solving (partial) differential equations
  - From continuous field equations…
  - *Here: Maxwell's equations*
    - *with electric field E, magnetic field M, material constants ε, μ*

$$\frac{\partial H}{\partial t} = -\frac{1}{\mu}\nabla\times E - \frac{1}{\mu}(M_{source} + \sigma^* H)$$

$$\frac{\partial E}{\partial t} = -\frac{1}{\epsilon}\nabla\times H - \frac{1}{\epsilon}(J_{source} + \sigma E)$$

  - … to a discrete system
    - over finite elements k, polynomial basis coefficients ψ

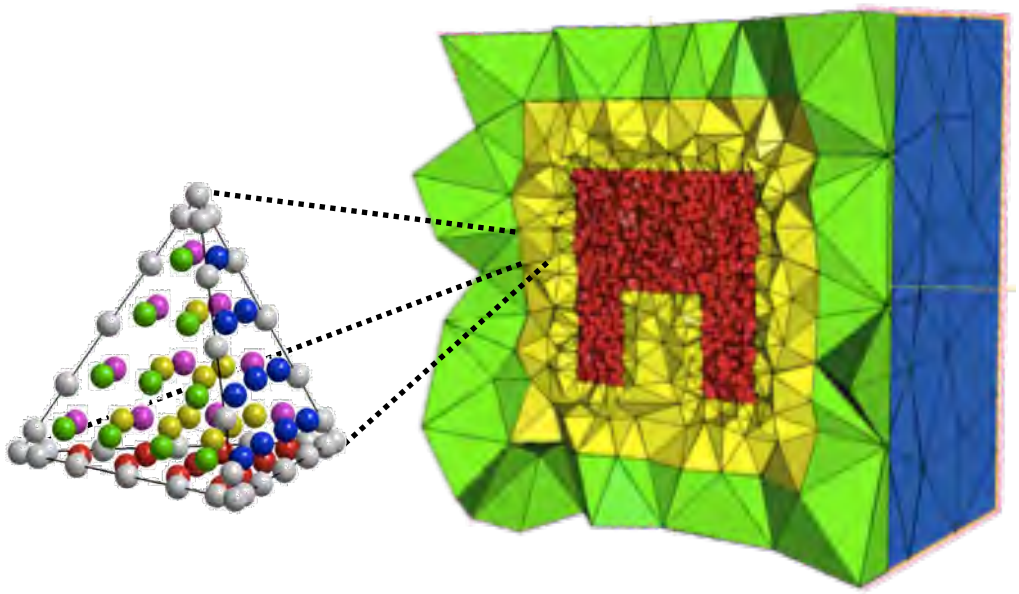
$$H_h^k = \sum_{n=1}^{N} \hat{H}_n^k \, \psi_n \,, \qquad E_h^k = \sum_{n=1}^{N} \hat{E}_n^k \, \psi_n$$

(only field approximation, Maxwell's equations not shown here)

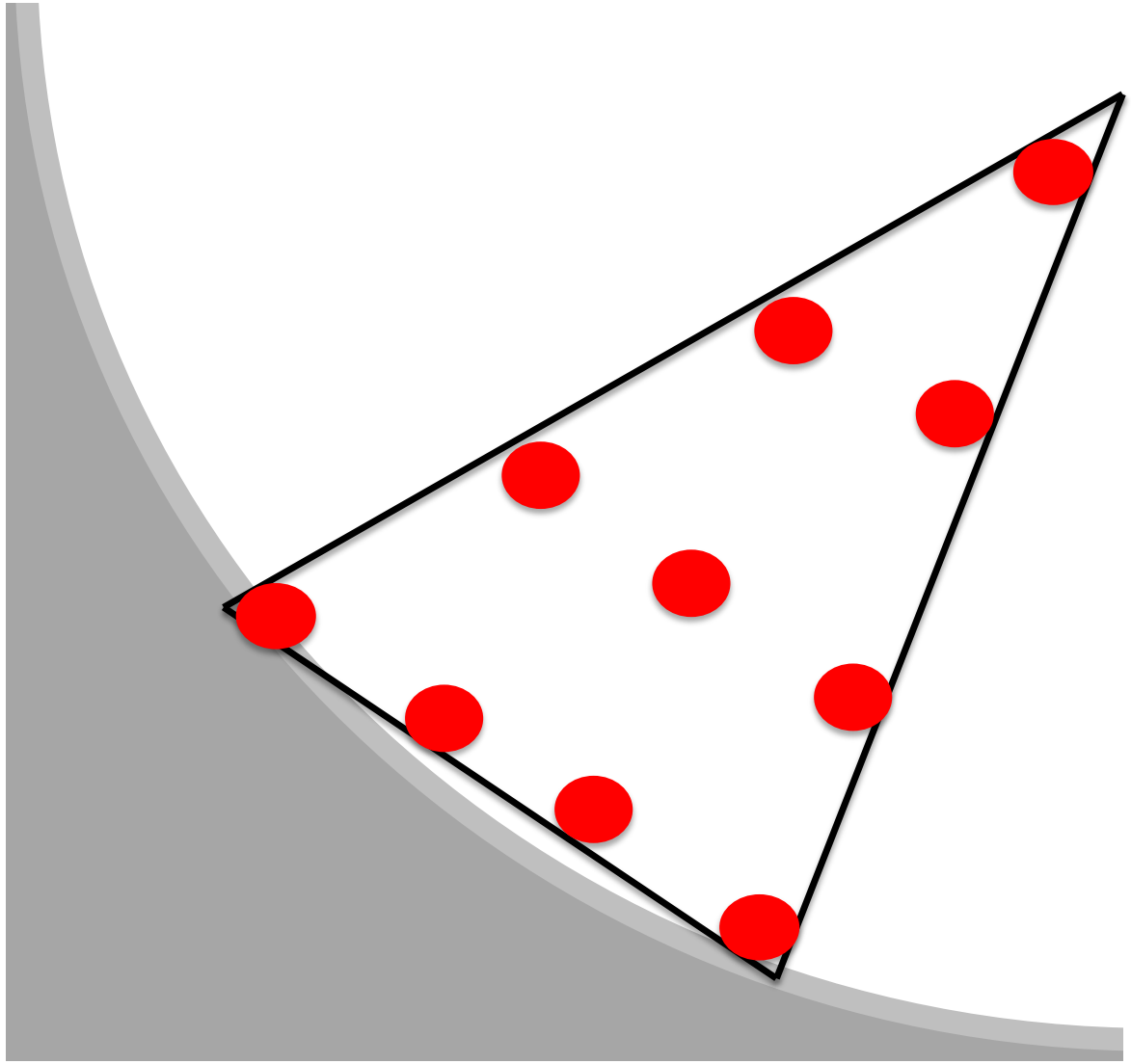
  - … to a computer program using linear algebra

DG Method Properties

- Nodal Discontinuous Galerkin Time Domain Method
  - state-of-the-art method
  - high numerical quality, provable stability
  - unstructured 3D mesh
    - adapted to material boundaries and regions of interest
    - particularly suitable for non-linear materials and multi-physics
    - no global stiffness matrix required
  - variable polynomial order
    - controls communication/computation-ratio
    - excellent parallel scaling
    - impacts arithmetic intensity



Unstructured Meshes, Nodal Representation

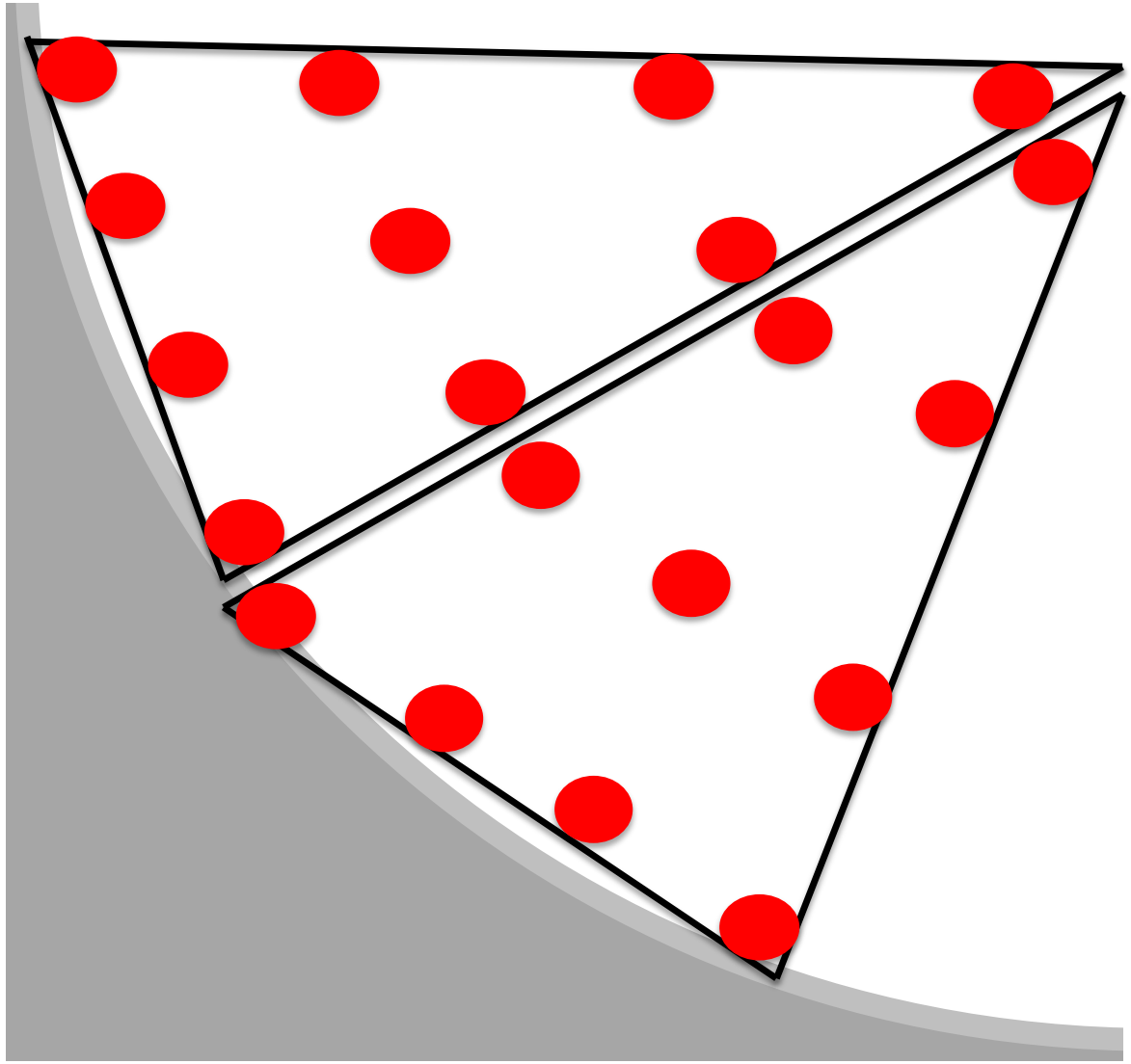


- over finite elements k
- polynomial basis coefficients ψ

$$H_h^k = \sum_{n=1}^{N} \hat{H}_n^k \, \psi_n$$

- element k
- N nodal points
  - depending on polynomial order
    - here: p=3
  - for each point
    - expansion coefficient $\hat{H}_n^k$
    - polynomial basis coefficient ψ

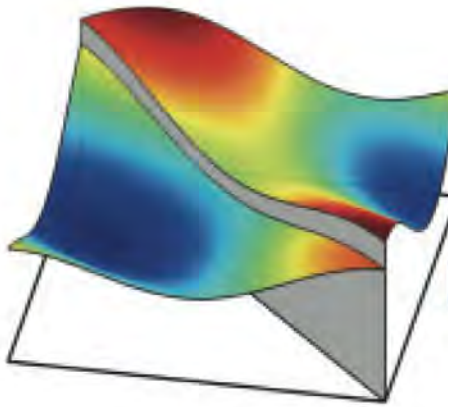
Discontinuous Galerkin Method



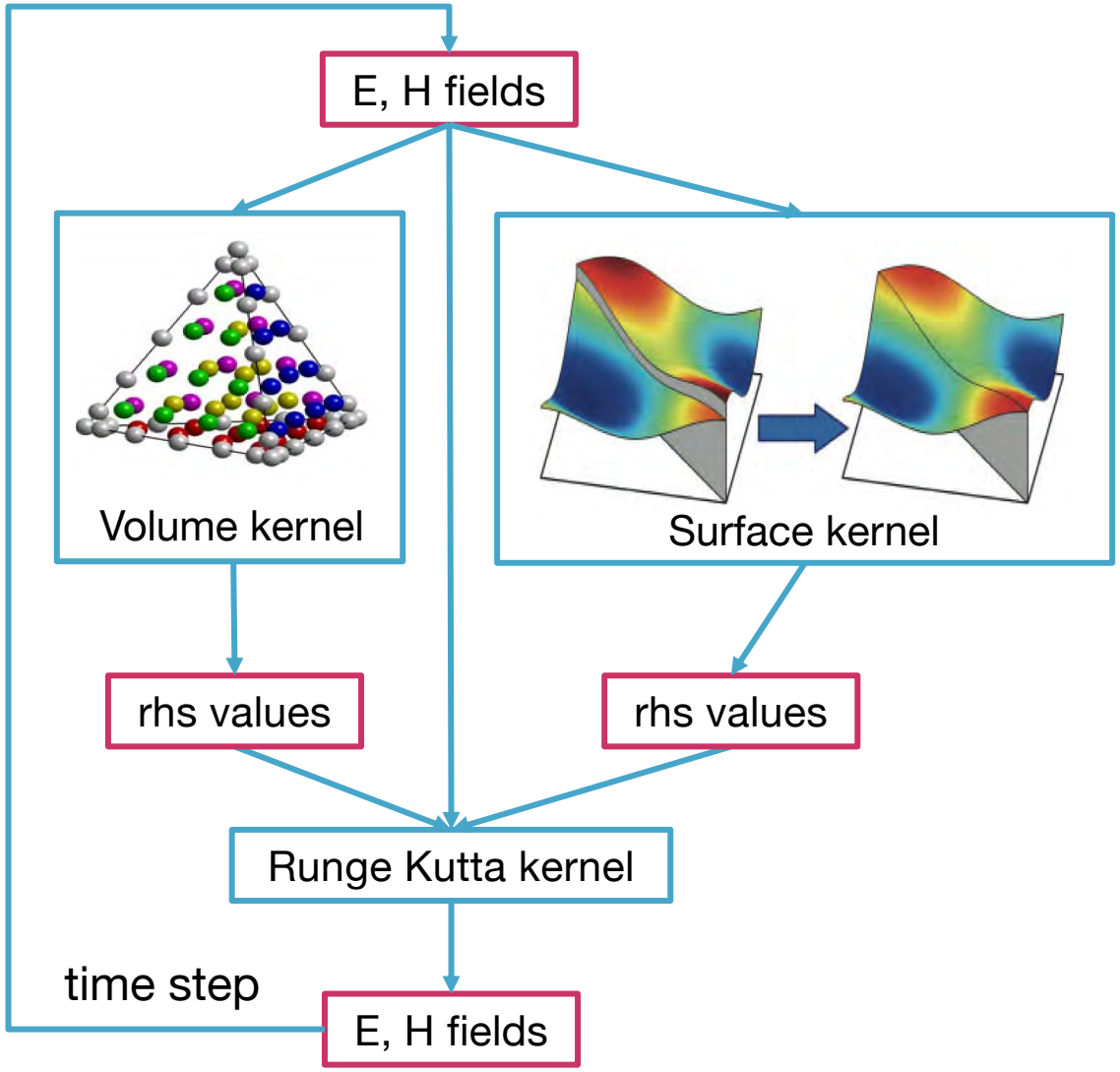
- over finite elements k
- polynomial basis coefficients ψ

$$H_h^k = \sum_{n=1}^{N} \hat{H}_n^k \, \psi_n$$

- Nodal points at edges
  - two copies
  - same coordinates
  - different coefficients $\hat{H}_n^k,\ \psi_n$



Application Structure

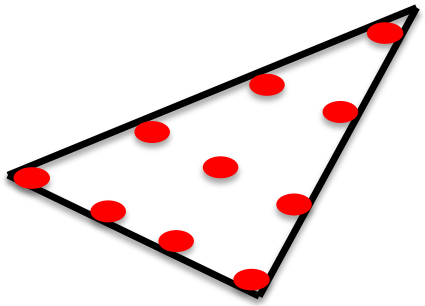


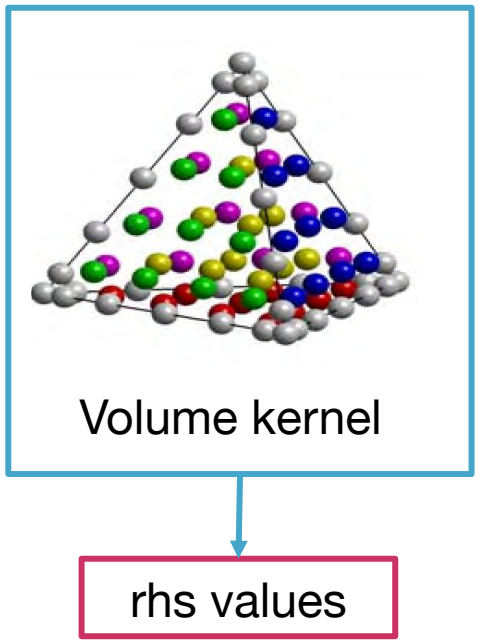
- Implementation based on MIDG2 code by Tim Warburton
- Algorithm divided in three kernels running on FPGA
  - Volume kernel
  - Surface kernel
  - Runge-Kutta kernel
- Method works uses tetrahedral meshes as elements
  - E and H field is defined at nodal points in volume and at surface
  - typ. mesh sizes 10^3–10^6 elements

T. Warburton: MIDG2 – https://github.com/tcew/MIDG2

DG Solver Kernel Analysis

Volume Kernel





Input: Field values $\mathbf{E}_{k,n}$, $\mathbf{H}_{k,n}$, polynomial base coefficients $\mathbf{D}_{n,m}$, geometry coefficients $\mathbf{G}_k$
Output: Local update terms $\mathbf{rhsE}_{k,n}$, $\mathbf{rhsH}_{k,n}$

```
1  foreach element k ∈ 1..K do
       // Prefetch all nodes of k first
2      foreach node n ∈ 1..N do
3          ldE = 0
4          ldH = 0
5          foreach node m ∈ 1..N do
6              ldE += D_{n,m} ⊗ E_{k,m}
7              ldH += D_{n,m} ⊗ H_{k,m}
8          end
9          rhsE_{k,n} = G_k ⊗ ldH
10         rhsH_{k,n} = G_k ⊗ ldE
11     end
12 end
```

Volume Kernel – Arithmetic Intensity

Input: Field values $\mathbf{E}_{k,n}$, $\mathbf{H}_{k,n}$, polynomial base coefficients $\mathbf{D}_{n,m}$, geometry coefficients $\mathbf{G}_k$
Output: Local update terms $\mathbf{rhsE}_{k,n}$, $\mathbf{rhsH}_{k,n}$

1 **foreach** *element* $k \in 1..K$ **do**
  // Prefetch all nodes of k first
2 **foreach** *node* $n \in 1..N$ **do**
3 $\mathbf{ldE} = 0$
4 $\mathbf{ldH} = 0$
5 **foreach** *node* $m \in 1..N$ **do**
6 $\mathbf{ldE} \mathrel{+}= \mathbf{D}_{n,m} \otimes \mathbf{E}_{k,m}$
7 $\mathbf{ldH} \mathrel{+}= \mathbf{D}_{n,m} \otimes \mathbf{H}_{k,m}$
8 **end**
9 $\mathbf{rhsE}_{k,n} = \mathbf{G}_k \otimes \mathbf{ldH}$
10 $\mathbf{rhsH}_{k,n} = \mathbf{G}_k \otimes \mathbf{ldE}$
11 **end**
12 **end**

⊗ operation: curl-like vector operation

	p=3, N=20	p=4, N=35
#total acc.	3600	11025
#unique	249	429

number of accesses to float variables
→ high reuse, small working sets

#FLOPs per k

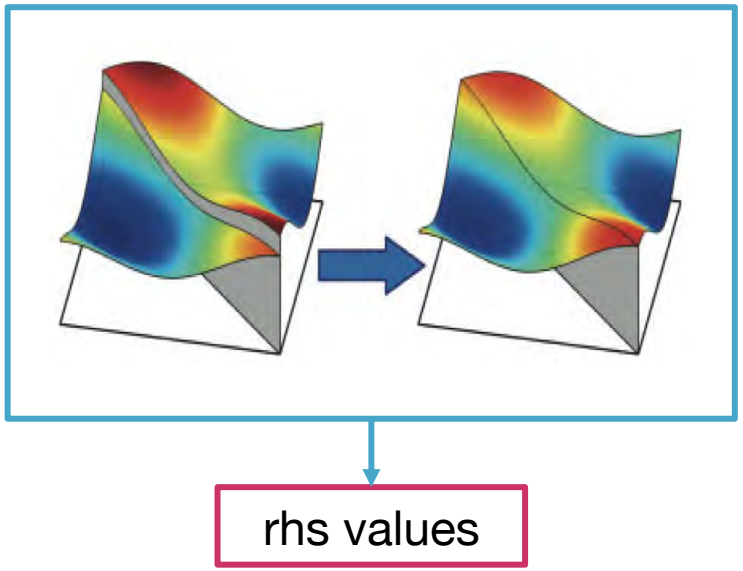
- $36 * N^2$
- $72 * N$

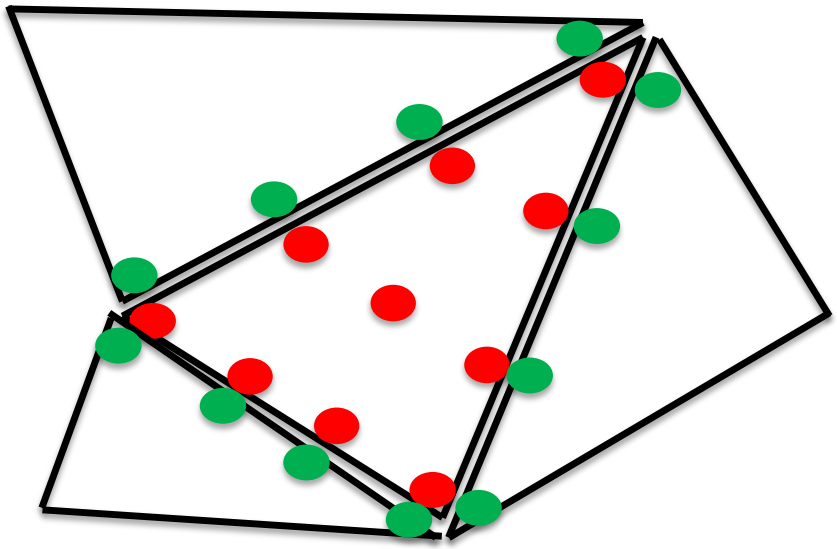
p=3, N=20	p=4, N=35
15840	46620

Arithmetic intensity [FLOPs/byte]

p=3, N=20	p=4, N=35
15.9	27.2

Surface Kernel





Input: Field values $\mathbf{E}_{k,n}$, $\mathbf{H}_{k,n}$, data layout information $localMAP_{k,f,n}$, $otherMAP_{k,f,n}$, surface geometry information $\mathbf{S}_{k,f,n}$, correlation coefficients to other element $\mathbf{LIFT}_{n,f,m}$

Output: Neighborhood update terms $\mathbf{rhsE}_{k,n}$, $\mathbf{rhsH}_{k,n}$

1 **foreach** *element* $k \in 1..K$ **do**
2 **foreach** *surface* $f \in 1..4$ **do**
3 **foreach** *surfacenode* $m \in 1..N_f$ **do**
4 $\mathbf{E}^- = \mathbf{E}[localMAP_{k,f,m}]$
5 $\mathbf{H}^- = \mathbf{H}[localMAP_{k,f,m}]$
6 $\mathbf{E}^+ = \mathbf{E}[otherMAP_{k,f,m}]$
7 $\mathbf{H}^+ = \mathbf{H}[otherMAP_{k,f,m}]$
8 $\Delta\mathbf{E}_{\mathbf{f},\mathbf{m}} = \mathbf{S}_{k,f,m} \otimes (\mathbf{E}^+ - \mathbf{E}^-)$
9 $\Delta\mathbf{H}_{\mathbf{f},\mathbf{m}} = \mathbf{S}_{k,f,m} \otimes (\mathbf{H}^+ - \mathbf{H}^-)$
10 **end**
11 **end**
12 $\mathbf{rhsE}_{k,n} = 0$
13 $\mathbf{rhsH}_{k,n} = 0$
14 **foreach** *node* $n \in 1..N$ **do**
15 **foreach** *surface* $f \in 1..4$ **do**
16 **foreach** *surfacenode* $m \in 1..N_f$ **do**
17 $\mathbf{rhsE}_{k,n}$ += $\mathbf{LIFT}_{n,f,m} \otimes \Delta\mathbf{E}_{\mathbf{f},\mathbf{m}}$
18 $\mathbf{rhsH}_{k,n}$ += $\mathbf{LIFT}_{n,f,m} \otimes \Delta\mathbf{H}_{\mathbf{f},\mathbf{m}}$
19 **end**
20 **end**
21 **end**
22 **end**

Surface Kernel – Arithmetic Intensity

Input: Field values $\mathbf{E}_{k,n}$, $\mathbf{H}_{k,n}$, data layout information $localMAP_{k,f,n}$, $otherMAP_{k,f,n}$, surface geometry information $\mathbf{S}_{k,f,n}$, correlation coefficients to other element $\mathbf{LIFT}_{n,f,m}$
Output: Neighborhood update terms $\mathbf{rhsE}_{k,n}$, $\mathbf{rhsH}_{k,n}$

1 **foreach** *element* $k \in 1..K$ **do**
2 **foreach** *surface* $f \in 1..4$ **do**
3 **foreach** *surfacenode* $m \in 1..N_f$ **do**
4 $\mathbf{E}^- = \mathbf{E}[localMAP_{k,f,m}]$
5 $\mathbf{H}^- = \mathbf{H}[localMAP_{k,f,m}]$
6 $\mathbf{E}^+ = \mathbf{E}[otherMAP_{k,f,m}]$
7 $\mathbf{H}^+ = \mathbf{H}[otherMAP_{k,f,m}]$
8 $\Delta\mathbf{E}_{\mathbf{f},\mathbf{m}} = \mathbf{S}_{k,f,m} \otimes (\mathbf{E}^+ - \mathbf{E}^-)$
9 $\Delta\mathbf{H}_{\mathbf{f},\mathbf{m}} = \mathbf{S}_{k,f,m} \otimes (\mathbf{H}^+ - \mathbf{H}^-)$
10 **end**
11 **end**
12 $\mathbf{rhsE}_{k,n} = \mathbf{0}$
13 $\mathbf{rhsH}_{k,n} = \mathbf{0}$
14 **foreach** *node* $n \in 1..N$ **do**
15 **foreach** *surface* $f \in 1..4$ **do**
16 **foreach** *surfacenode* $m \in 1..N_f$ **do**
17 $\mathbf{rhsE}_{k,n}$ += $\mathbf{LIFT}_{n,f,m} \otimes \Delta\mathbf{E}_{\mathbf{f},\mathbf{m}}$
18 $\mathbf{rhsH}_{k,n}$ += $\mathbf{LIFT}_{n,f,m} \otimes \Delta\mathbf{H}_{\mathbf{f},\mathbf{m}}$
19 **end**
20 **end**
21 **end**
22 **end**

indirect memory access

61 × 4 N_f FLOPS

12 × N × 4 N_f FLOPS

- Data usage
  - LIFT: constant memory
  - $\Delta E, \Delta H$: local buffer inside k
  - #unique float values per k

p=3, N=20	p=4, N=35
880	970

- FLOPs

p=3, N=20	p=4, N=35
12040	28860

- Arithmetic intensity [FLOPs/byte]

p=3, N=20	p=4, N=35
3.4	7.4

FPGA Accelerator Design with High-Level Synthesis from OpenCL

High-Level Synthesis for FPGAs with OpenCL

- Accelerator model: Kernels + host code
- Pipelining
  - pipeline independent work items *or*
  - infer pipeline from (data parallel) loop
  - between kernels: channels (adaptation from OpenCL 2.0 pipes)
- Parallelism
  - work items, work groups
  - vector data types
  - unrolling
  - multiple kernels
- Custom memory structures
  - local memory for variables and fixed-size arrays
  - mapped to FPGA on-chip SRAM blocks or registers
  - independent ports per named array or through partitioning

Parallelism

Input: Field values $\mathbf{E}_{k,n}$, $\mathbf{H}_{k,n}$, polynomial base coefficients $\mathbf{D}_{n,m}$, geometry coefficients $\mathbf{G}_k$
Output: Local update terms $\mathbf{rhsE}_{k,n}$, $\mathbf{rhsH}_{k,n}$

```
1  foreach element k ∈ 1..K do
       // Prefetch all nodes of k first
2      foreach node n ∈ 1..N do
3          ldE = 0
4          ldH = 0
5          foreach node m ∈ 1..N do
6              ldE += D_{n,m} ⊗ E_{k,m}
7              ldH += D_{n,m} ⊗ H_{k,m}
8          end
9          rhsE_{k,n} = G_k ⊗ ldH
10         rhsH_{k,n} = G_k ⊗ ldE
11     end
12 end
```

pipeline

unroll

$36 \times N^2$ FLOPS

$72 \times N$ FLOPS

- Configurable unrolling of inner loops
- Rule of thumb: 1000+ FLOPs/cycle

Input: Field values $\mathbf{E}_{k,n}$, $\mathbf{H}_{k,n}$, data layout information $localMAP_{k,f,n}$, $otherMAP_{k,f,n}$, surface geometry information $\mathbf{S}_{k,f,n}$, correlation coefficients to other element $\mathbf{LIFT}_{n,f,m}$
Output: Neighborhood update terms $\mathbf{rhsE}_{k,n}$, $\mathbf{rhsH}_{k,n}$

```
1  foreach element k ∈ 1..K do
2      foreach surface f ∈ 1..4 do
3          foreach surfacenode m ∈ 1..N_f do
4              E⁻ = E[localMAP_{k,f,m}]
5              H⁻ = H[localMAP_{k,f,m}]
6              E⁺ = E[otherMAP_{k,f,m}]
7              H⁺ = H[otherMAP_{k,f,m}]
8              ΔE_{f,m} = S_{k,f,m} ⊗ (E⁺ − E⁻)
9              ΔH_{f,m} = S_{k,f,m} ⊗ (H⁺ − H⁻)
10         end
11     end
12     rhsE_{k,n} = 0
13     rhsH_{k,n} = 0
14     foreach node n ∈ 1..N do
15         foreach surface f ∈ 1..4 do
16             foreach surfacenode m ∈ 1..N_f do
17                 rhsE_{k,n} += LIFT_{n,f,m} ⊗ ΔE_{f,m}
18                 rhsH_{k,n} += LIFT_{n,f,m} ⊗ ΔH_{f,m}
19             end
20         end
21     end
22 end
```

pipeline

$61 \times 4\ N_f$ FLOPS

unroll

$12 \times N \times 4\ N_f$ FLOPs

Results: Synthesis and Kernel Performance

- Nallatech 385A with Intel Arria 10 1150 GX FPGA
- Intel OpenCL SDK 16.0
- Optimize one design per polynomial order p (and corresponding N)
  - containing all three kernels, RK not shown, fully unrolled for p = 3,4

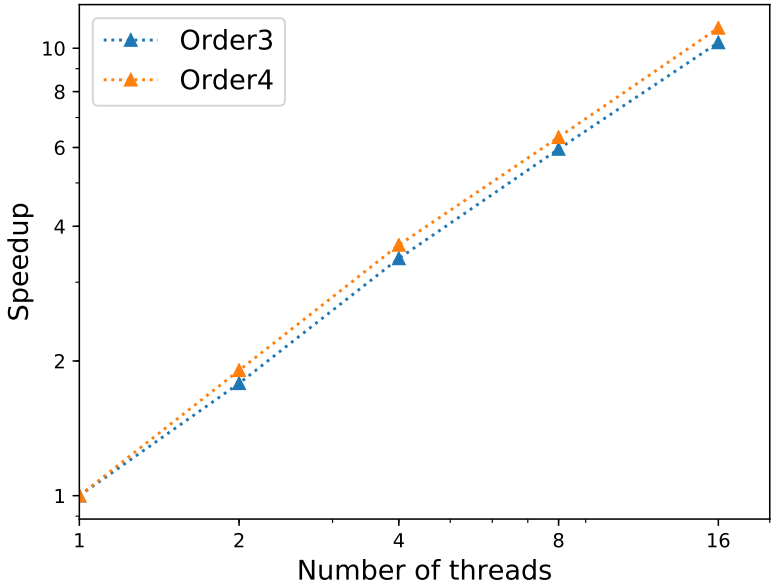
	Volume Kernel					**Surface Kernel**					**Synthesis results**			
p	unr.	FLOPs /cycle	GFLOPs /s	occ [%]	BW [GB/s]	unr.	FLOPs /cycle	GFLOPs /s	occ [%]	BW [GB/s]	Freq [MHz]	Logic [%]	DSP [%]	RAM [%]
3	**20**	792	103	64	6.1	20	301	35	58	**16.3**	202	32	42	42
4	**35**	1332	**164**	66	6.1	20	301	42	75	12.3	187	35	**60**	46
5	14	576	99	86	2.3	28	397	60	75	11.6	201	37	39	61
6	12	504	74	83	1.2	56	913	**102**	95	13.4	176	38	48	**72**

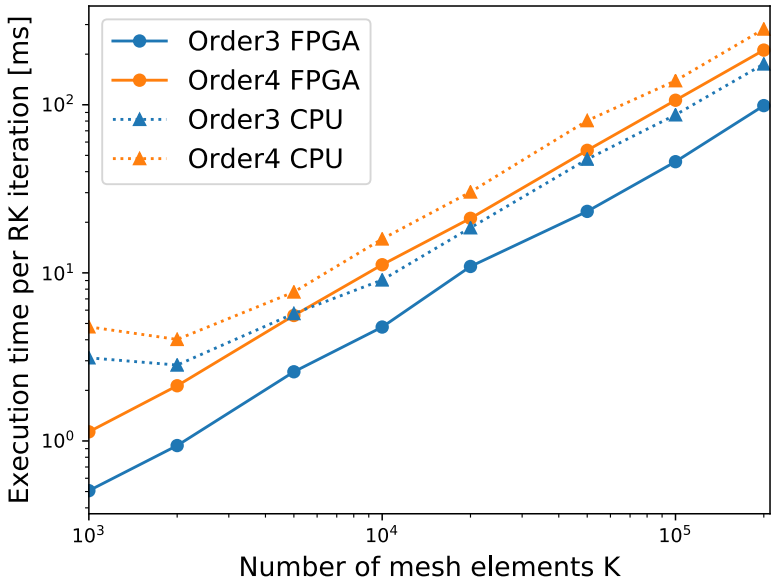
Results: Performance and Scalability

- CPU Reference
  - 2 x Xeon E5-2670v1, 2.60GHz
  - 2 x 8 = 16 cores
  - OpenMP and vectorized

- FPGA implementation outperforms dual-socket Xeon by ~2x
  - >100 GFLOP/s kernel designs
  - using local RAM as buffers and constant memory
  - achieving high off-chip bandwidth through decoupled access

- Atypical application for FPGA, but works very well



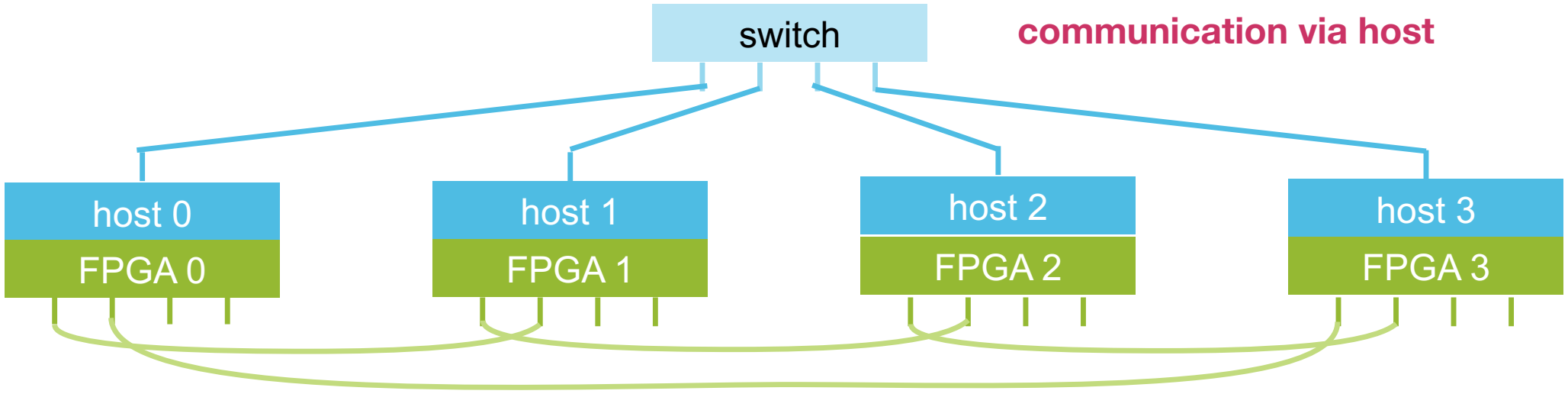


[Kenter et al., OpenCL-based FPGA Design to Accelerate the Nodal Discontinuous Galerkin Method for Unstructured Meshes, FCCM'18]

Scaling FPGA Applications

Parallel Computing with FPGAs

- HPC requires node-level parallelism, but FPGA research focuses on single node
- Requirement for acceptance of FPGAs in general HPC applications
  - allocate required number of host systems and FPGAs from cluster
  - support different tool/BSP version with automated provisioning of bitstream and drivers
  - abstract logical view on communication from physical implementation / placement
- Communication via host vs. direct communication between FPGAs



Host vs. Direct Communication

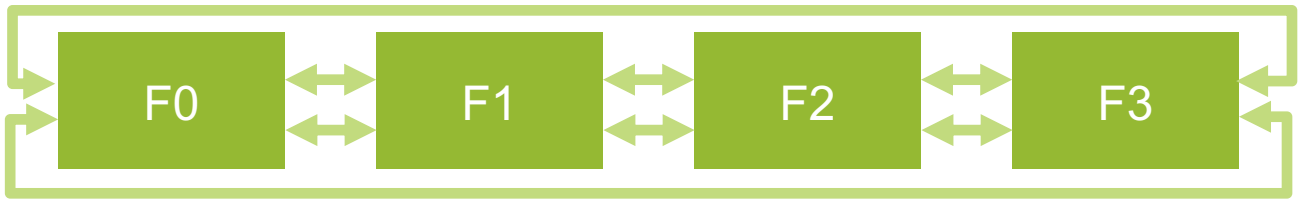
Host communication

- Pro
  - mature communication libraries, in particular MPI
  - natural match for porting existing HPC applications
  - leverage commodity HPC networks (OmniPath, Infiniband)
  - high scalability and flexibility: packet switching and routing
- Con
  - relies on bulk communication, i.e. buffering: introduces latency
  - packetization, SW protocol layers introduce overheads

Direct P2P communication

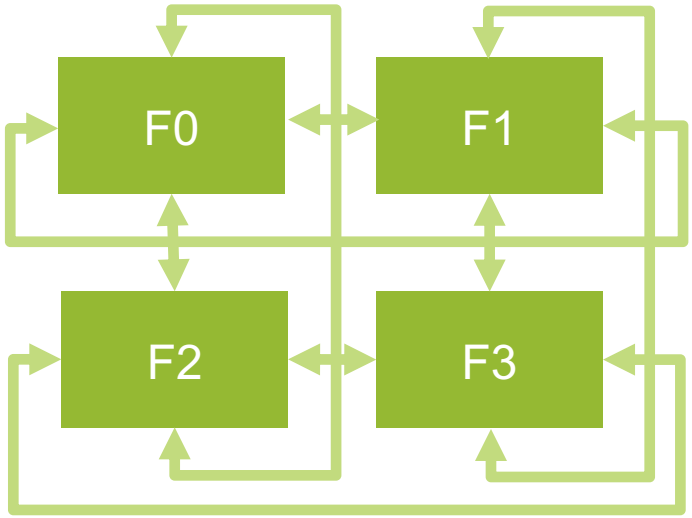
- Pro
  - no SW protocol overhead
  - streaming communication no buffering
  - natural match for OpenCL channel communication model
  - customization of communication topology to application
- Con
  - proprietary serial communication
  - no routing, direct point-to-point links
  - no established SW libraries for CPUs

Example Topologies



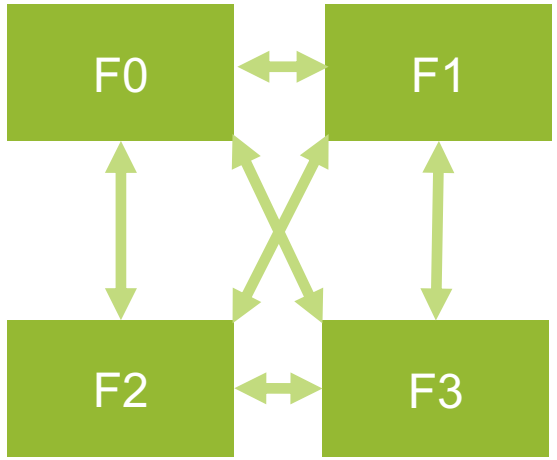
ring

e.g. distributed force computation for n-body problems



2D torus

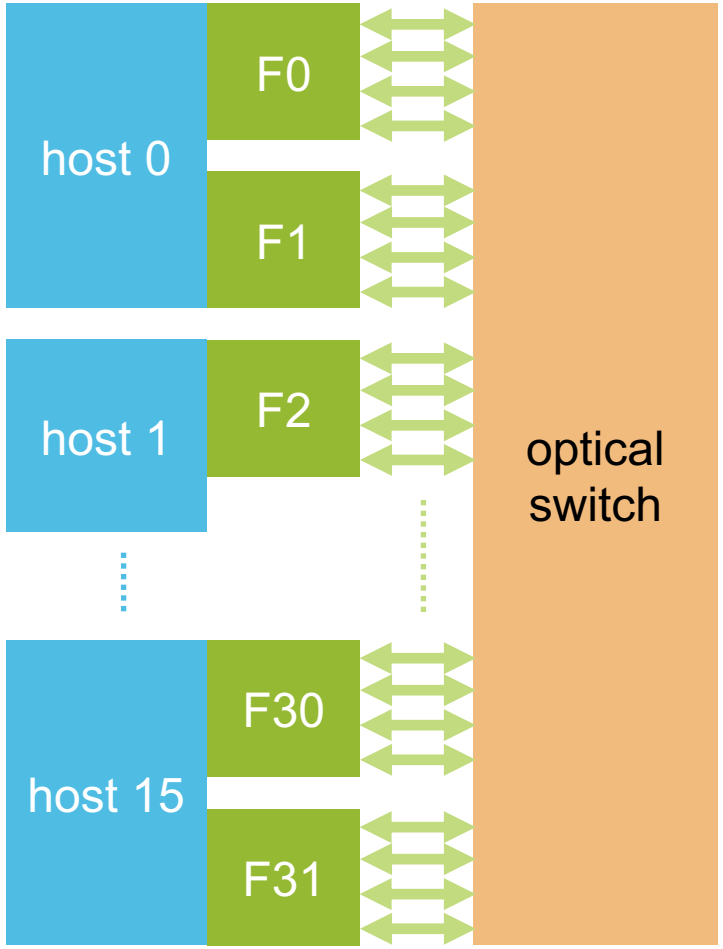
halo exchange, e.g. simulation of periodic structures, structured meshes



fully connected

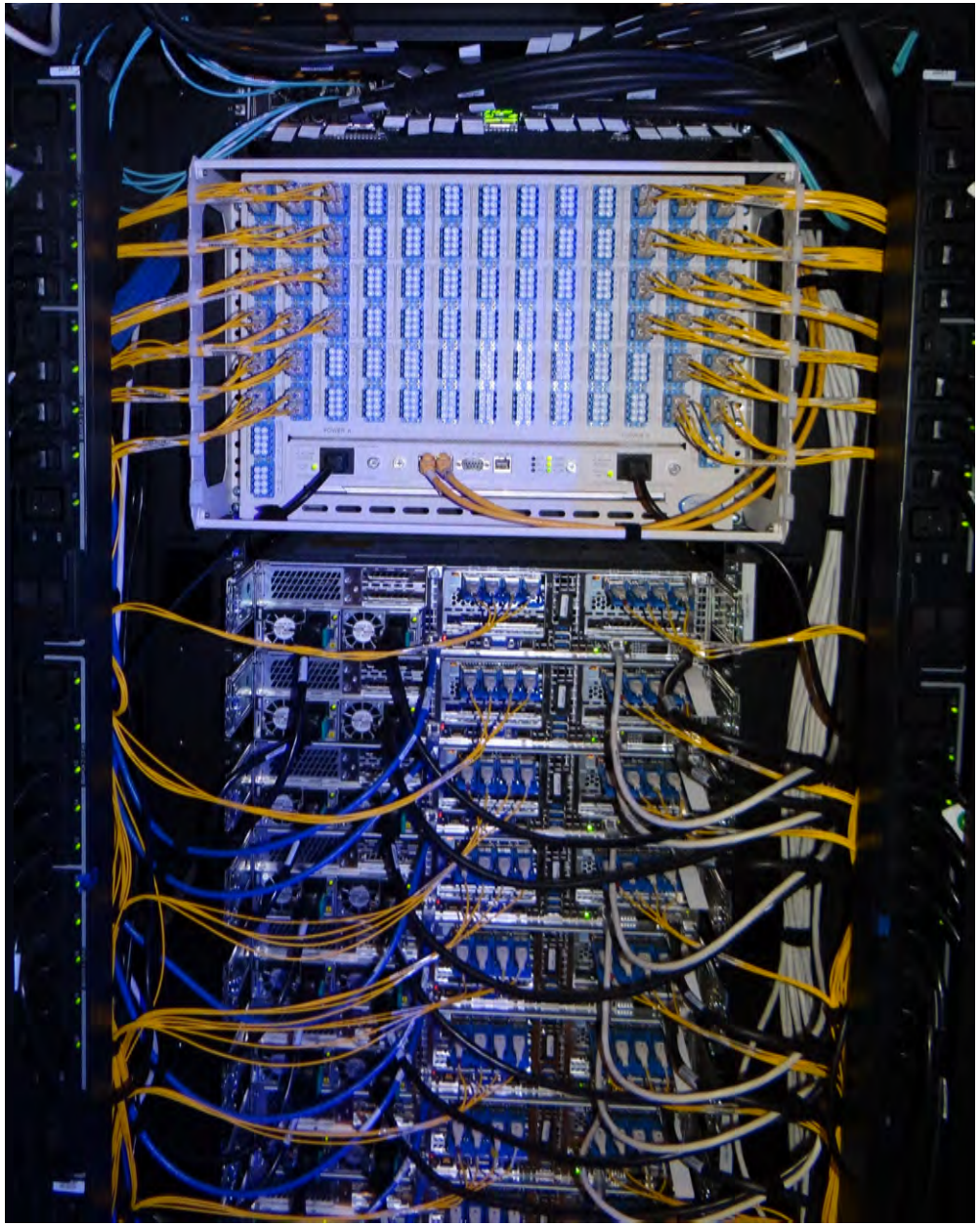
all-to-all communication, e.g. 3D FFT, unstructured meshes

Optical Switch for FPGAs in Noctua



- Protocol-agnostic optical switch (Calinet S320)
  - 3D MEMS technology
    - low latency (<50ns)
    - channel configuration (<50ms)
  - full crossbar connectivity, i.e. congestion-free routing of arbitrary topologies with full bandwidth
- Software configurable using RESTful API, OpenFlow, ...

Optical Switch for FPGAs in Noctua (2)



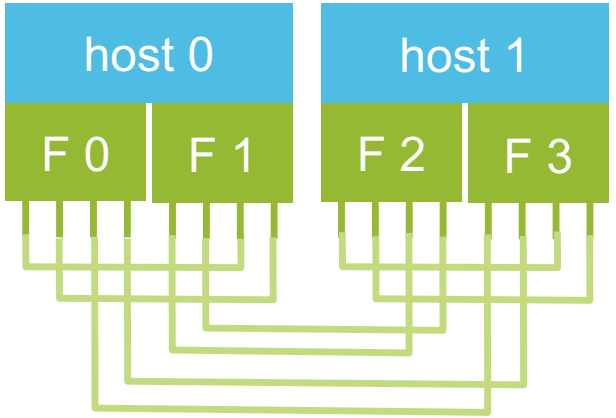
Optical Switch for FPGAs in Noctua (3)

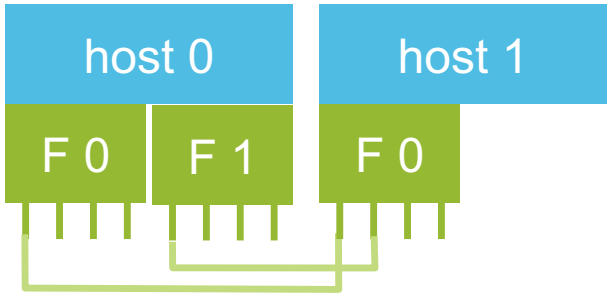
- Switch configuration integrated with SLURM workload manager
  - job submission specifies number of FPGAs and topology
  - SLRUM bootstraps MPI application and configures switch with physical according to topology
  - support predefined (ring, torus, clique, ...) and custom topologies
- Examples
  - 4 FPGAs with torus topology

```
srun –N4 --fpgalink= "ring"
```

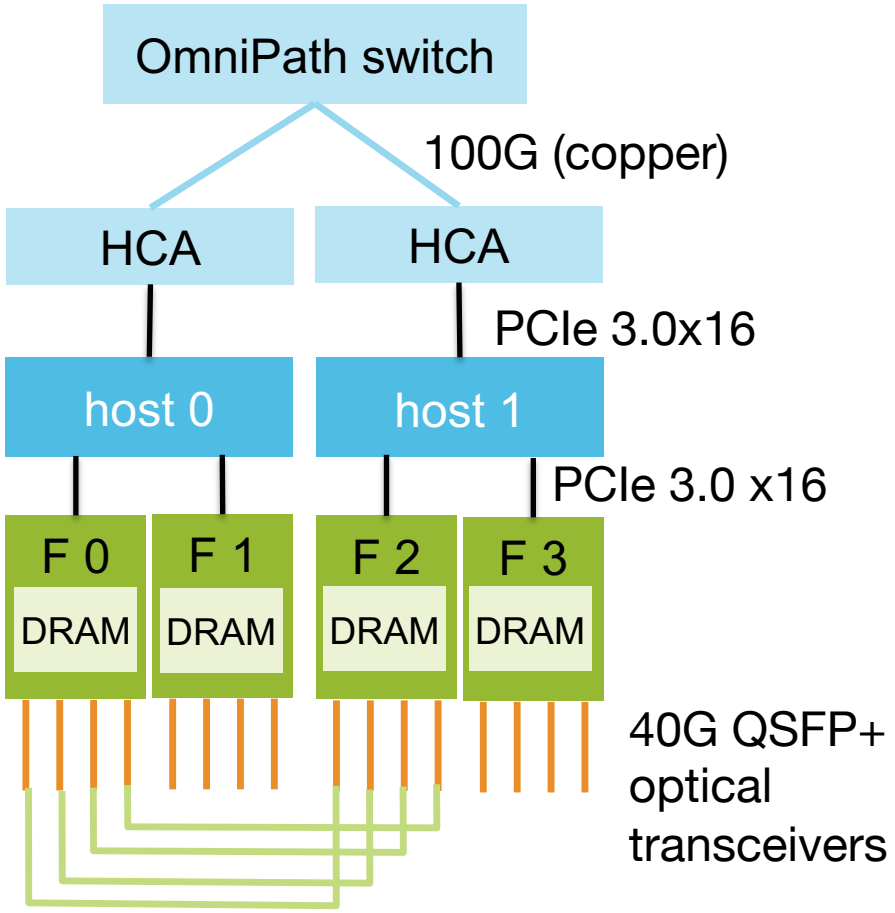
  - custom topologies (tree of 3 FPGAs)

```
srun –N3 \
--fpgalink="n00:acl0:ch0-n01:acl0:ch0" \
--fpgalink="n00:acl1:ch0-n01:acl0:ch1"
```



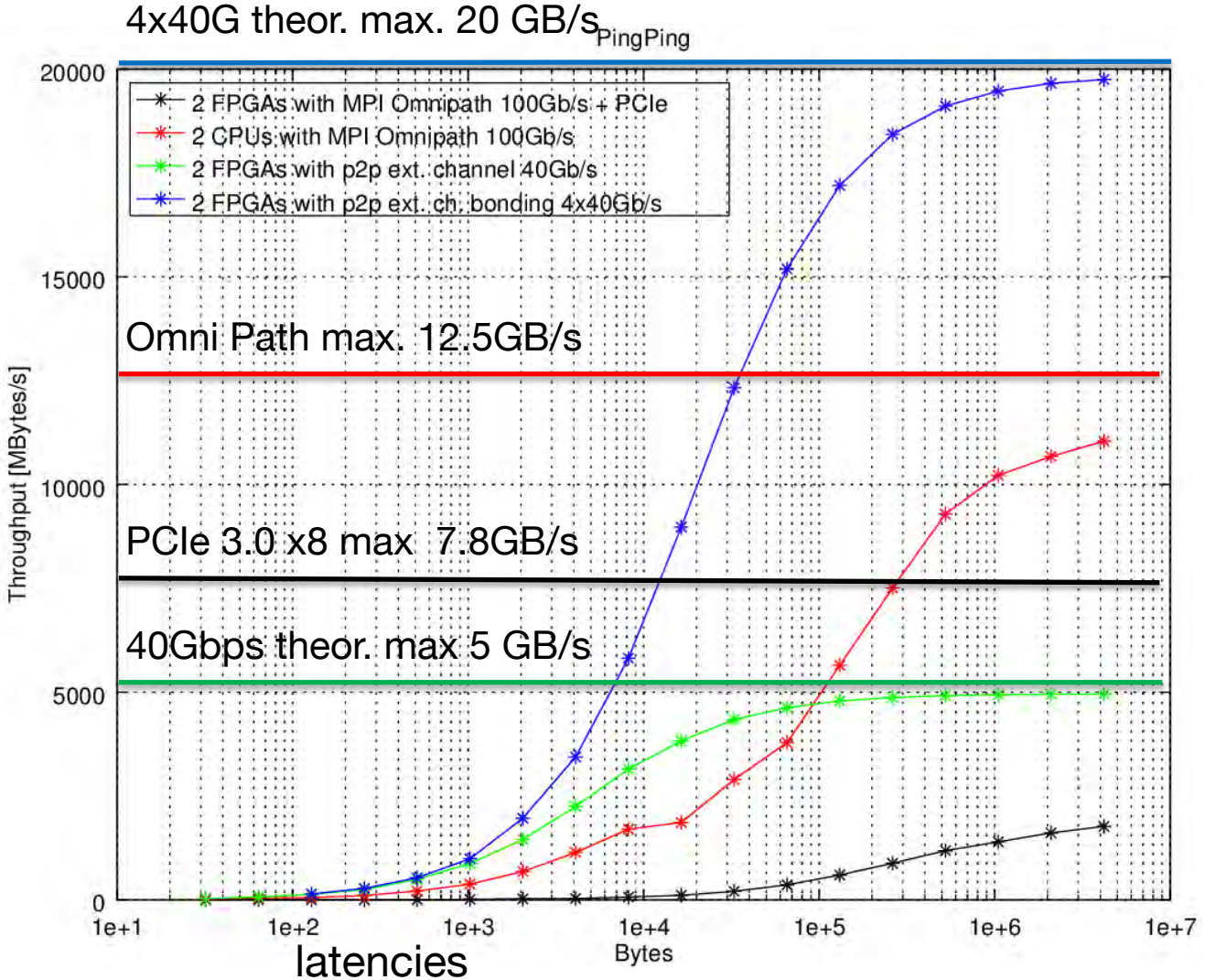


Performance Characterization



Current OpenCL BSP limitations

- FPGA card PCIe limited to x8



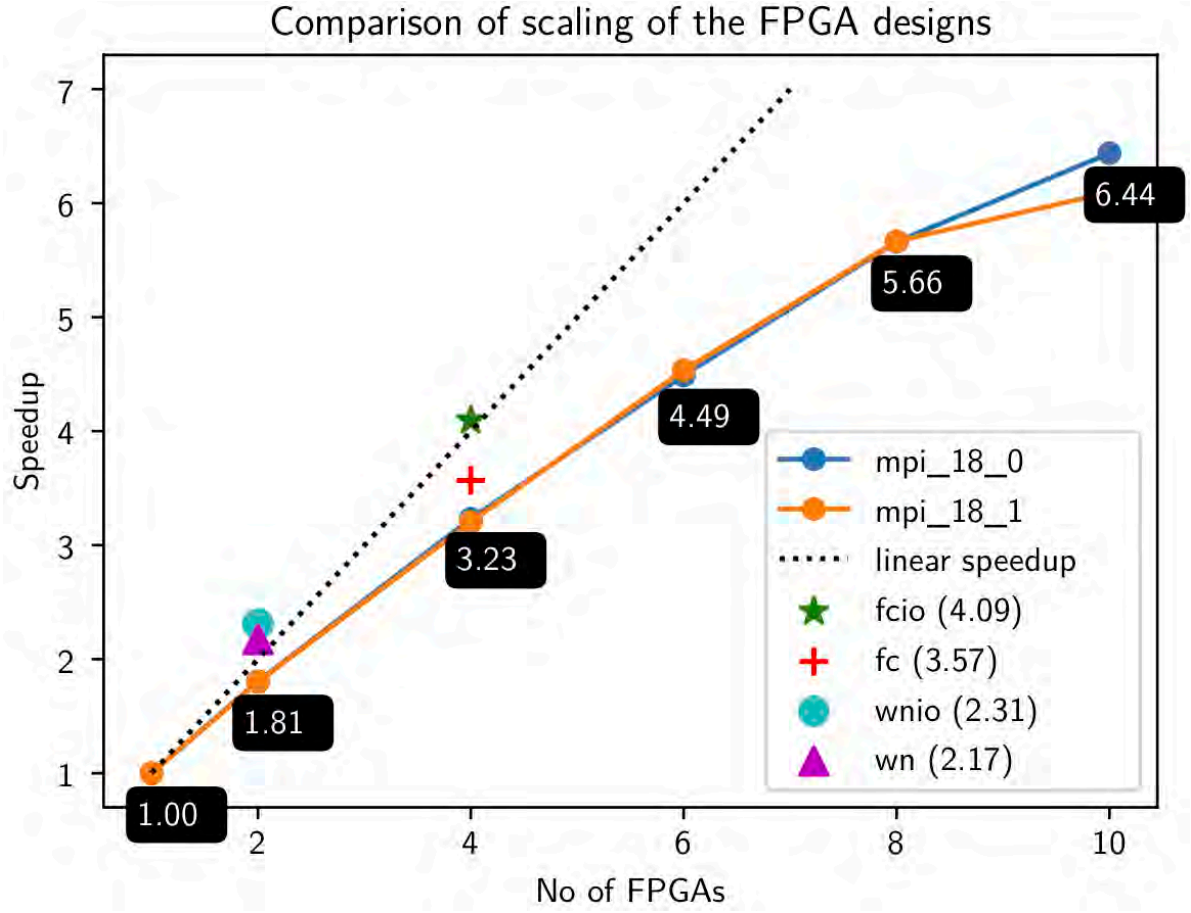
latencies

- CPU: 1µs (OmniPath, same switch)
- FPGA 0.588µs (direct connection)

MIDG2* with Direct Communication (Early Results)

Third phase

- MIDG2* application ported to run completely on FPGA
  - MPI only used in setup phase (partition)
  - all communication during simulation with direct FPGA-to-FPGA links
  - host only used to synchronize kernel memory access
    - *wnio, fcio*
  - with FPGA-only workaround: *wn, fc*
- Much better strong scaling
- Evaluation predates installation of optical switch
  - tested only with up to 4 FPGAs



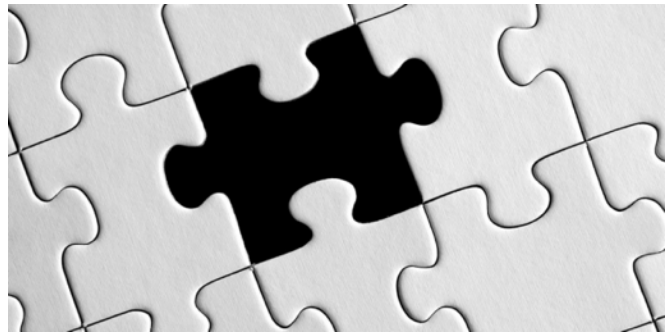
strong scaling using MPI vs. direct communication

Gaurav K. Singh: Adding point-to-point communication between FPGAs to an accelerator for the Discontinuous Galerkin method. Master's thesis, Paderborn University May 2019.

Conclusion

Roadmap to Establish FPGAs in HPC

- Hardware
  - move FPGAs from lab to HPC production systems ✓
  - improve system integration (RDMA, shared memory)
- Software
  - (open source) HPC applications and libraries using FPGAs
  - HPC-specific development tools & runtime environment (✓)
  - benchmarks
- Community
  - currently: FPGA ∩ HPC ≈ ∅
  - workshops, conferences, journals, user meetings (WIP)
- Developer training
  - materials addressing HPC developers and CSE
  - best practices and design patterns



Conclusions

- Demonstrated benefit of FPGAs for algorithms working on unstructured grids
- State-of-the-art FPGAs provide a lot of computation and communication performance
- Using high-level synthesis FPGAs can be programmed without expert knowledge in digital design
- Intersection of FPGA and HPC communities still very small
- We are open for collaborations!

Contact

Christian Plessl
Paderborn University
christian.plessl@uni-paderborn.de

Twitter: @plessl @pc2_upb

http://pc2.uni-paderborn.de

Paderborn Center for Parallel Computing