Discontinuous Galerkin Shallow-Water Simulations on FPGAs

Tobias Kenter

Adesh Shambhu, Sara Faghih-Naini, Vadym Aizinger

Paderborn University, Germany, Paderborn Center for Parallel Computing

NHR PerfLab Seminar, 19 October 2021

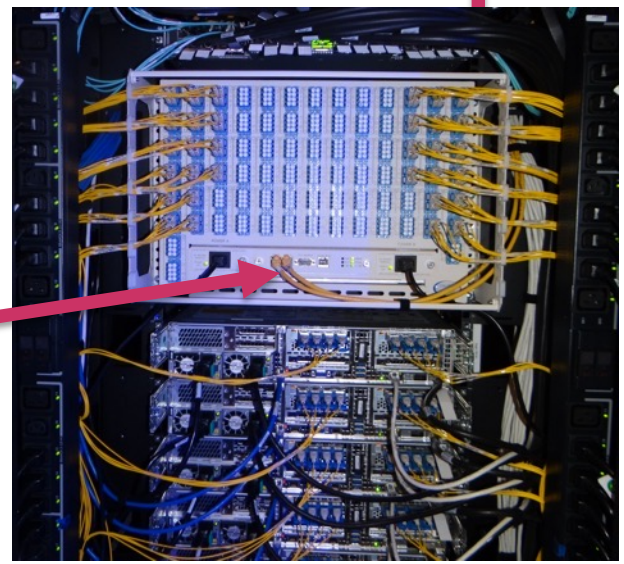
Paderborn Center for Parallel Computing

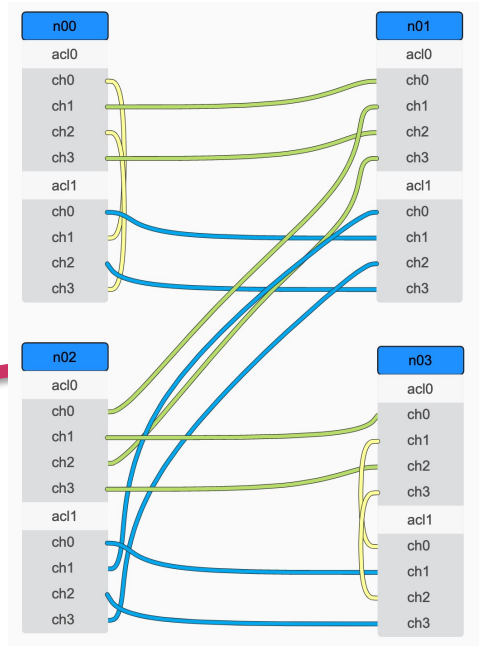
Paderborn Center for Parallel Computing (PC²)

- HPC operations and research
- Noctua System since 2018
  - 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
- 16 FPGA nodes
  - 2 x Intel Stratix 10 GX2800 per node (BittWare 520N boards, PCIe 3.0 x8)
    - 4 x 8GB DDR4 channels per board
  - 4 QSFP28 ports per board
  - configurable point-to-point topologies

```
srun -N4 --fpgalink="ring0"
```

- Successor system 2022





FPGA Plattform

- Intel Stratix 10 GX 2800
  - 5760 DSP blocks (1 single precision FMA/cycle each)
  - 11,721 M20K RAM blocks (20Kb each)
  - 933,120 ALMs: control, addresses, all non-FP arithmetic
  - 3,732,480 registers: form pipeline stages

- Bittware 520N card
  - PCIe Gen3 x8 (x16)
  - 4 * 8GB DDR4

- Intel FPGA SDK for OpenCL
- Intel FPGA Add-on for oneAPI Base Toolkit

This work: Discontinuous Galerkin Shallow-Water Model on FPGA

- Shallow-Water Code
  - Discontinuous Galerkin discretization
  - unstructured mesh
  - polynomial orders 0, 1, 2 viable
- Performance challenges
  - not well-suited for vectorization
    - small inner loops, e.g. 3, 6, 9 iterations
  - indirect and irregular memory access
  - strong scaling, simulation of long time scales

- How can FPGAs help?

[T. Kenter, A. Shambhu, S. Faghih-Naini, V. Aizinger. *Algorithm-Hardware Co-design of a Discontinuous Galerkin Shallow-Water Model for a Dataflow Architecture on FPGA.* PASC'21.]

Mapping Code to FPGA Ressources

- Intel Stratix 10 GX 2800
  - 5760 DSP blocks (1 single precision FMA/cycle each)
  - 11,721 M20K RAM blocks (20Kb each)
  - 933,120 ALMs: control, addresses, all non-FP arithmetic
  - 3,732,480 registers: form pipeline stages

FP-arithmetic

local memory

unrolling creates small vector units

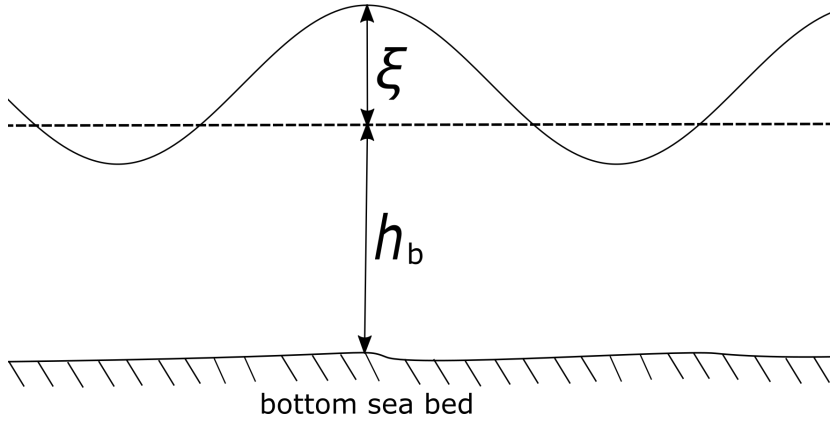
```
523         /* Gradient of tidal potential minus athmospheric pressure */
524         #pragma unroll
525         for (char i = 0; i < d_space; i++) {
526           float temp = G * tip2_l[all_el_info[it].vertex_number[i + 1]] -
527                        pr2_l[all_el_info[it].vertex_number[i + 1]] -
528                        G * tip2_l[all_el_info[it].vertex_number[0]] +
529                        pr2_l[all_el_info[it].vertex_number[0]];
530         #pragma unroll
531           for (char j = 0; j < d_space; j++)
532             tip_pr_grad[j] += all_el_info[it].jacob_phys_to_ref[i][j] * temp;
533         }
```

Shallow Water DG Code

Shallow Water Equations

- 2D shallow water equations (SWE) (derived from the Navier-Stokes equations)

  - $\partial_t \xi + \nabla \cdot \mathbf{u} = 0$
  - $\partial_t \mathbf{u} + \nabla \cdot \left(\frac{\mathbf{u} \otimes \mathbf{u}}{\mathrm{H}} \right) + \tau_{bf} \mathbf{u} + f_c \mathbf{k} \times \mathbf{u} + \mathrm{g} \mathrm{H} \nabla \xi = \mathbf{F}$



with unknowns

ξ : elevation of free water surface, $\mathbf{u} = (U, V)^T$: depth integrated horizontal velocity field

and parameters

h_b: bathymetric depth, $\mathrm{H} = h_b + \xi$: total fluid depth, τ_{bf} : bottom friction coefficient

f_c : Coriolis coefficient, $\mathbf{k}$: unit vertical vector, g : gravitational acceleration

F : forcing term from wind and atmospheric pressure gradient

DG Formulation

- Uses Discontinuous Galerkin method on unstructured triangular meshes

$$\int_{\Omega_i} \partial_t \mathbf{c}_\Delta \boldsymbol{\varphi}\, dx + \int_{\partial\Omega_i} \widehat{\mathrm{A}}(\boldsymbol{c}_\Delta, \boldsymbol{c}_\Delta{}^+, \boldsymbol{n})\, \boldsymbol{\varphi}\, ds - \int_{\Omega_i} \mathrm{A}(\boldsymbol{c}_\Delta) \cdot \nabla\varphi\, dx = \int_{\Omega_i} \boldsymbol{r}(\boldsymbol{c}_\Delta)\, \boldsymbol{\varphi}\, dx$$

Edge kernel — Element kernel

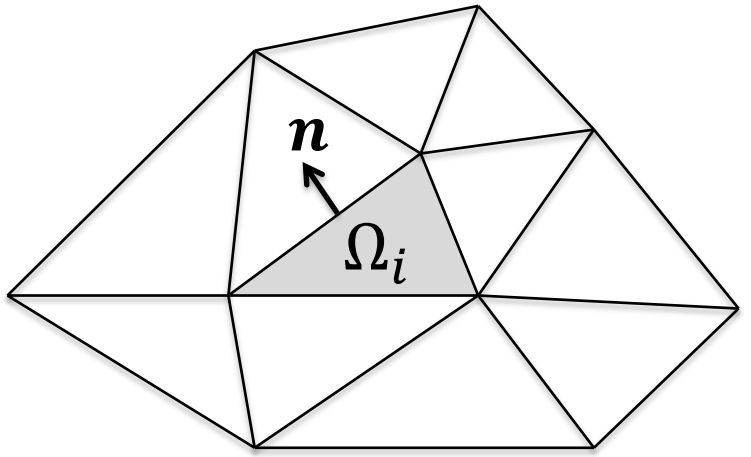
where

$\mathbf{c}_\Delta = (\xi_\Delta, U_\Delta, V_\Delta)^T$: the discrete vector of unknowns restricted to Ω_i,

$\mathbf{c}_\Delta{}^+$: the discrete vector of unknowns restricted to the edge-neighbour of Ω_i,

$\boldsymbol{n}$: the exterior unit normal to $\partial\Omega_i$, $\boldsymbol{\varphi}$: test function

$\widehat{\mathrm{A}}$: numerical flux from Riemann solver (Lax-Friedrichs)



UTBEST Overview

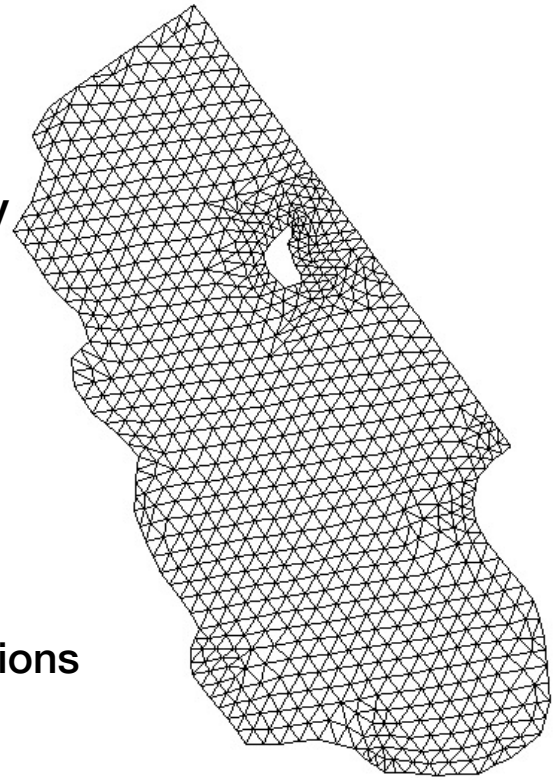
- I/O and grid management: FORTRAN
- DG scheme + computationally intensive parts: C
  - works in single precision
- 3 polynomial DG discretizations
  - piecewise constant (PC), 1 basis function (= cell-centered finite volumes)
  - piecewise linear (PL), 3 basis functions
  - piecewise quadratic (PQ), 6 basis functions
- Integration kernels
  - elements: 1, 4, 9 quadrature points
  - edges: 1, 2, 3 quadrature points
    - Lax-Friedrichs Riemann solver
- Corresponding time discretization
  - Runge-Kutta orders 1, 2, 3

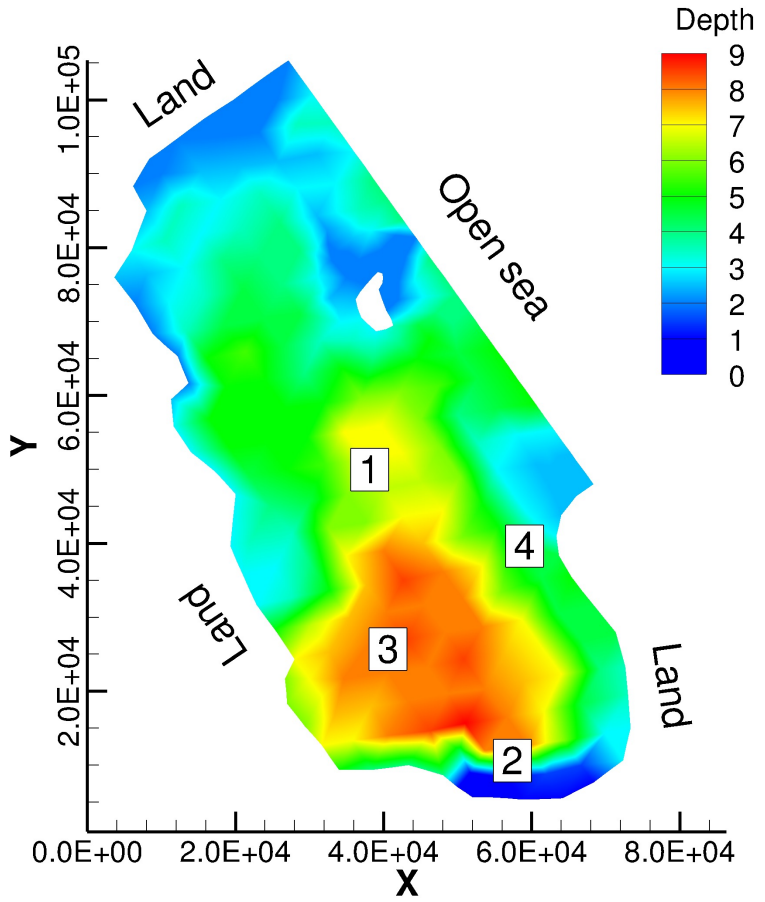
$$\int_{\Omega_i} \partial_t \mathbf{c}_\Delta \boldsymbol{\varphi}\, dx + \underbrace{\int_{\partial\Omega_i} \widehat{\mathrm{A}}(\boldsymbol{c}_\Delta, \boldsymbol{c}_\Delta{}^+, \boldsymbol{n})\, \boldsymbol{\varphi}\, ds}_{\text{Edge kernel}} - \underbrace{\int_{\Omega_i} \mathrm{A}(\boldsymbol{c}_\Delta) \cdot \nabla\varphi\, dx}_{\text{Element kernel}} = \int_{\Omega_i} \boldsymbol{r}(\boldsymbol{c}_\Delta)\, \boldsymbol{\varphi}\, dx$$

[V. Aizinger and C. Dawson. 2002. Adv. in Water Resources 25, 1]

Benchmark Scenario

- Bahamas (Bight of Abaco)
  - unstructured mesh
    - 1696 elements
  - tidal forcing at open sea boundary
  - benchmark runs
    - simulated 1 day
    - time step 5s
    - 17280 steps
  - outputs
    - elevation snapshots
    - full time series at observation stations





bathymetry + observation stations

UTBEST Structure + Execution

```
1:  while t < t_1 do
2:      Loop over Runge–Kutta stages:
3:      for all stages of the Runge–Kutta method do
4:          Element loop:
5:          for all element indices e ∈ {1, ..., E} do
6:              calculate element integrals
7:          end for
8:          Loops over edges of different types:
9:          for all interior edges do
10:             calculate interior edge integrals
11:         end for
12:         for all land edges do
13:             calculate land edge integrals
14:         end for
15:         for all open sea edges do
16:             calculate open sea edge integrals
17:         end for
18:         calculate c_Δ for the next Runge–Kutta stage
19:         perform minimum depth control on c_Δ
20:     end for
21:     t ← t + Δt
22: end while
```

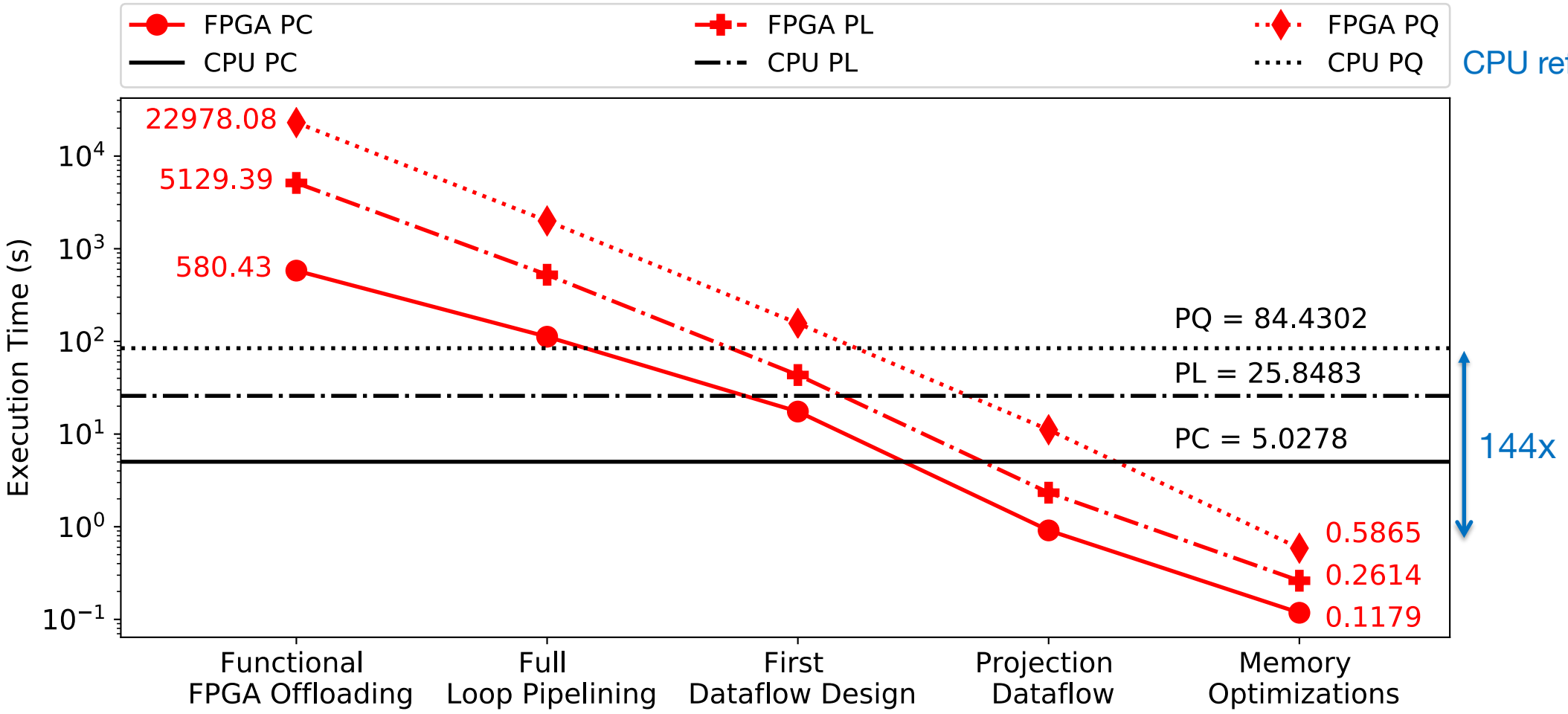
Kernel	Execution time			Perf. [GFLOPs]		
	PC	PL	PQ	PC	PL	PQ
Element	26.9%	38.0%	47.9%	2.27	3.76	4.97
Interior Edge	62.3%	53.2%	44.5%	1.51	2.27	2.97
Land Edge	2.4%	1.8%	1.4%	1.65	2.28	2.85
Sea Edge	2.2%	1.1%	0.6%	0.67	1.30	2.06
Accumulator	2.3%	3.3%	3.9%	2.98	3.92	4.32
Min. Depth	3.8%	2.5%	1.7%	1.40	2.41	3.38
Kernel sum, avg	5.02s	25.8s	84.4s	1.73	2.88	3.98

Profiled on 1 core of Skylake Xeon Gold 6148

FPGA Design Process

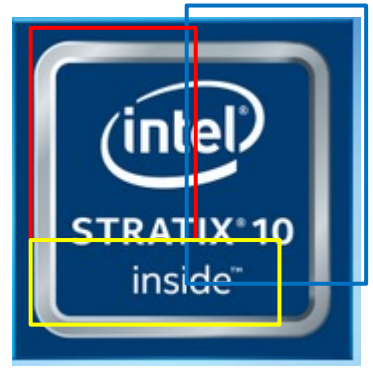
Overview of FPGA Optimization Process

- five main design iterations



FPGA Dataflow Idea 1/2

- C → OpenCL → hardware description
  - create block on FPGA for each kernel
  - pipelining: e.g. process one element per cycle
    - parallel hardware by unrolling of inner loops
    - provide all unknowns from local buffers (prefetching)
- Stream unknowns and updates through kernels
  - task level parallelism
  - keep all unknowns in local buffers



FPGA Dataflow Idea 2/2

```
1: while t < t_1 do
2:    Loop over Runge–Kutta stages:
3:    for all stages of the Runge–Kutta method do
4:       Element loop:
5:       for all element indices e ∈ {1, ..., E} do
6:          calculate element integrals        r ← c_Δ
7:       end for
8:    end for
9:    t ← t + Δt
10: end while
```

c_Δ

```
1: while t < t_1 do
2:    Loop over Runge–Kutta stages:
3:    for all stages of the Runge–Kutta method do
4:       perform minimum depth control on c_Δ
5:    end for
6:    t ← t + Δt
7: end while
```

c_Δ

```
1: while t < t_1 do
2:    Loop over Runge–Kutta stages:
3:    for all stages of the Runge–Kutta method do
4:       Loops over edges of different types:
5:       for all interior edges do
6:          calculate interior edge integrals      r ← c_Δ
7:       end for
8:       for all land edges do
9:          calculate land edge integrals          r ← c_Δ
10:      end for
11:      for all open sea edges do
12:         calculate open sea edge integrals       r ← c_Δ
13:      end for
14:   end for
15:   t ← t + Δt
16: end while
```

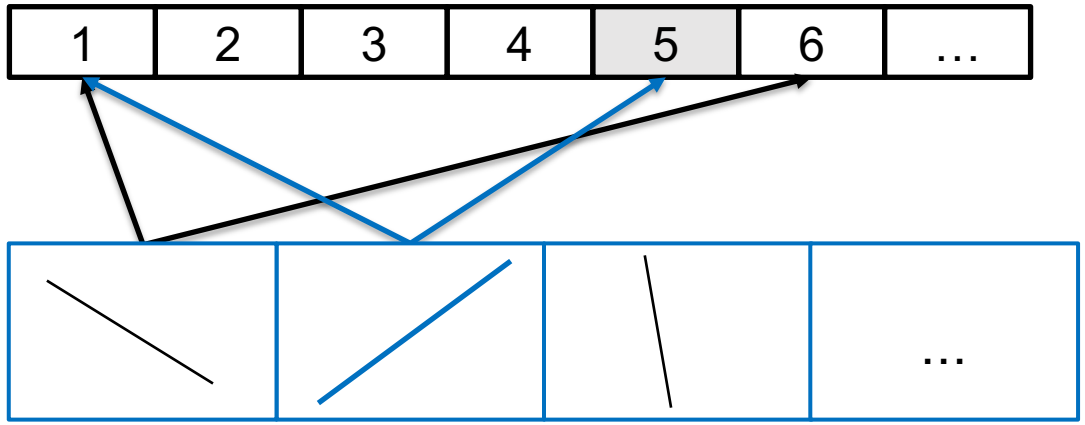
r

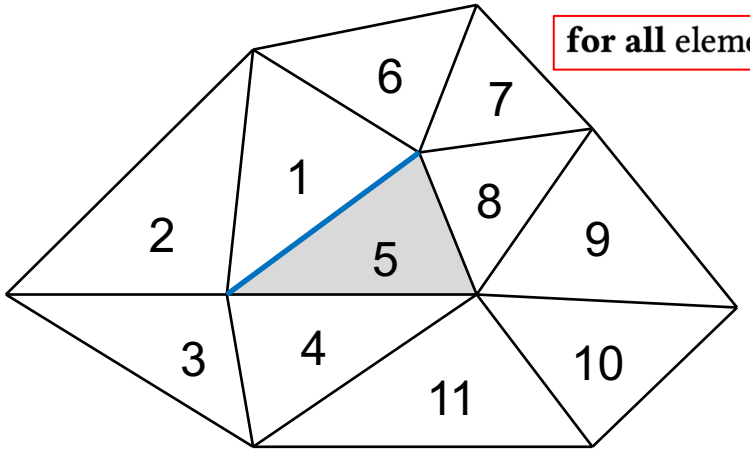
```
1: while t < t_1 do
2:    Loop over Runge–Kutta stages:
3:    for all stages of the Runge–Kutta method do
4:       calculate c_Δ for the next Runge–Kutta stage
5:    end for
6:    t ← t + Δt
7: end while
```

r

UTBEST Data Layout

- Unknowns c_Δ associated to elements
  - 3 * [1, 3, 6] depending on polynomial order





for all element indices $e \in \{1, \ldots, E\}$ **do**

- Further structure by references
  - edges to elements
    - "random" access into element array
  - elements to edges
  - …
- Geometry, bathymetry

for all interior edges **do**

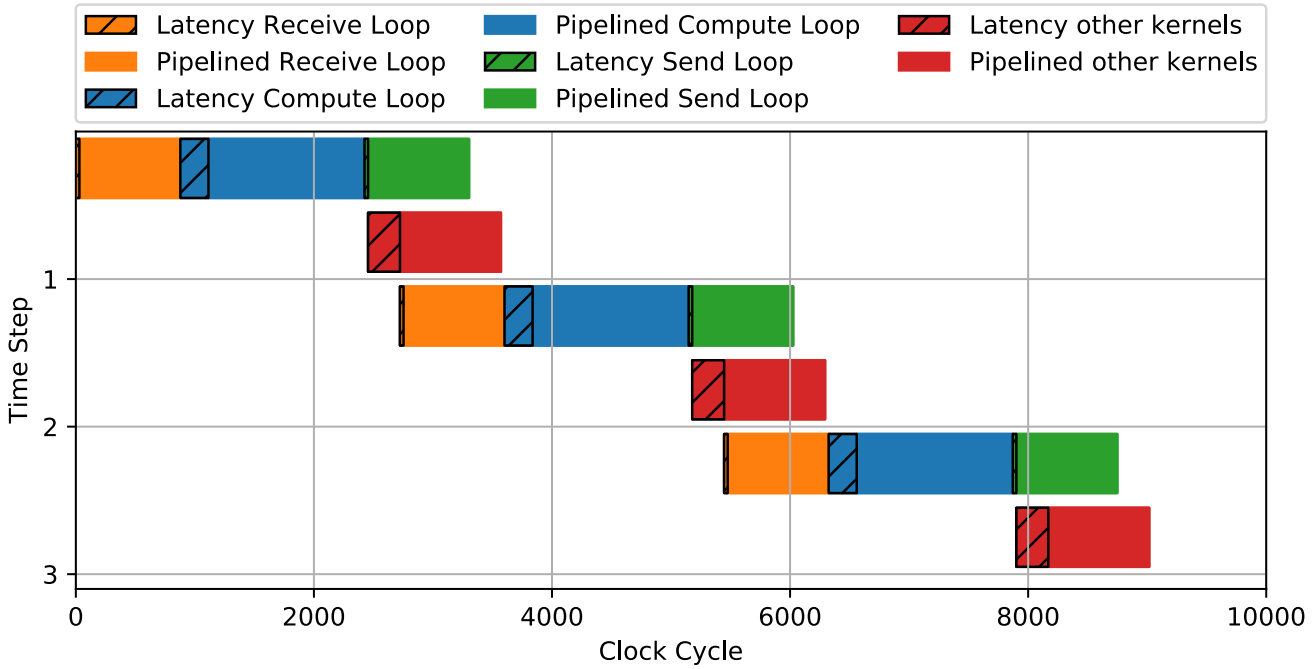
Initial Dataflow around Edge Kernel

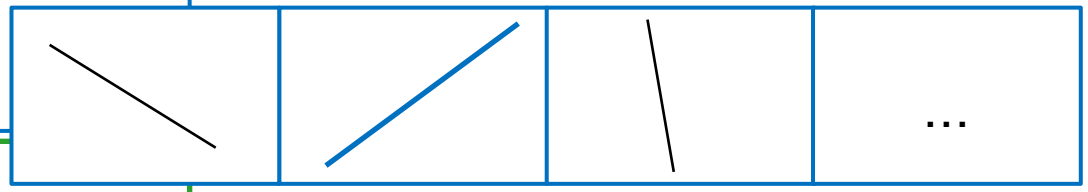
- Three phases required

1	2	3	4	5	6	…

1: *Receive loop:*
2: **for all** element indices $e \in \{1, \ldots, E\}$ **do**
3: receive $\boldsymbol{c}_\Delta$ from minimum depth channel
4: initialize element update term $\boldsymbol{r_{flux}}(\boldsymbol{c}_\Delta) \leftarrow \boldsymbol{0}$
5: **end for**
6: *Compute loop:*
7: **for all** edge indices $d \in \{1, \ldots, D\}$ **do**
8: flux $\boldsymbol{A}(\boldsymbol{c}_\Delta)$ and solution $\boldsymbol{c}_\Delta$ on local element
9: flux $\boldsymbol{A}(\boldsymbol{c}_\Delta^+)$ and solution $\boldsymbol{c}_\Delta^+$ on remote element
10: Riemann flux $\hat{\boldsymbol{A}} \leftarrow \mathit{riemann}(\boldsymbol{A}(\boldsymbol{c}_\Delta), \boldsymbol{A}(\boldsymbol{c}_\Delta^+), \boldsymbol{c}_\Delta, \boldsymbol{c}_\Delta^+)$
11: update $\boldsymbol{r_{flux}}(\boldsymbol{c}_\Delta) \leftarrow \boldsymbol{r_{flux}}(\boldsymbol{c}_\Delta) - \hat{\boldsymbol{A}}$
12: update $\boldsymbol{r_{flux}}(\boldsymbol{c}_\Delta^+) \leftarrow \boldsymbol{r_{flux}}(\boldsymbol{c}_\Delta^+) + \hat{\boldsymbol{A}}$
13: **end for**
14: *Send loop:*
15: **for all** element indices $e \in \{1, \ldots, E\}$ **do**
16: send $\boldsymbol{r_{flux}}(\boldsymbol{c}_\Delta)$ to accumulator channel
17: **end for**

1	2	3	4	5	6	…

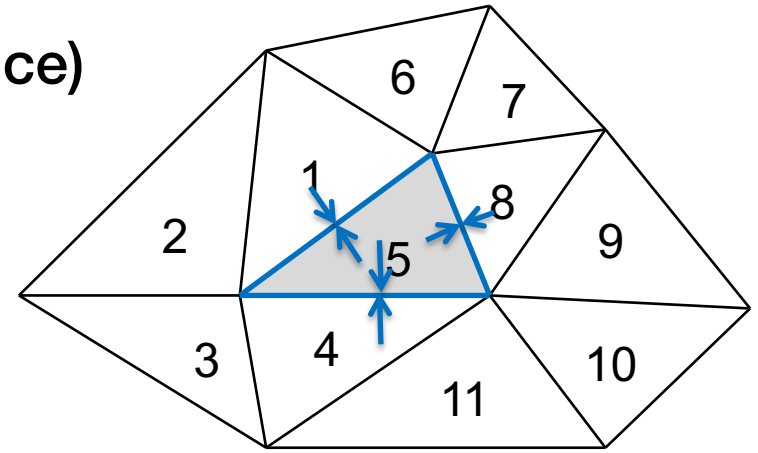




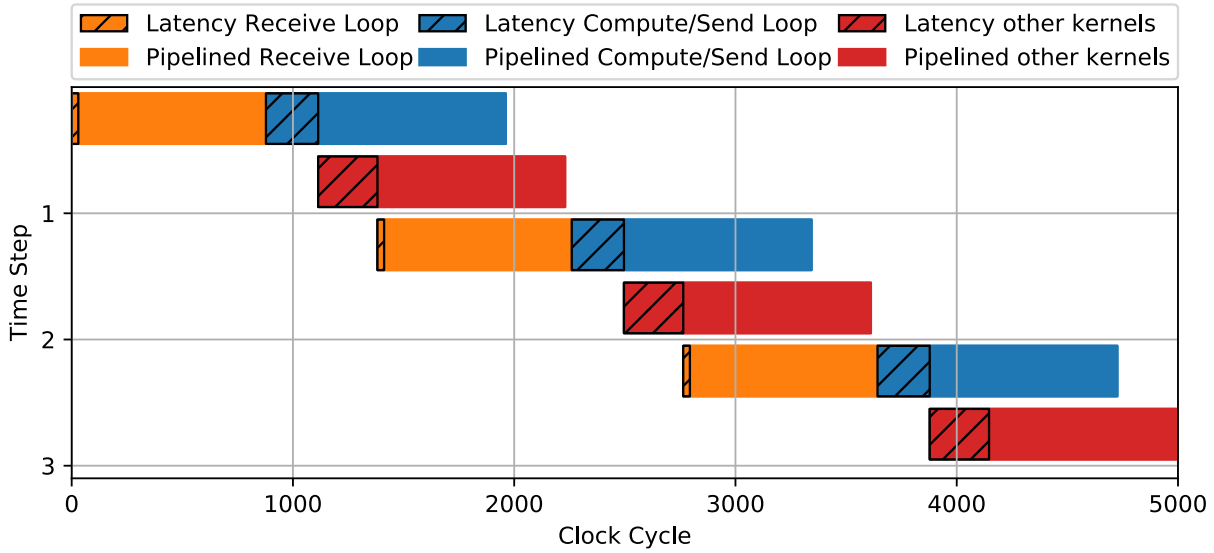
- poor utilization (either edge or element kernel active)
- more edges than elements

Projection Approach

- Run edge kernel in order of elements (each edge twice)
- Reduce redundant computation
  - 3 edge integrals per element, each
    - 2 projections to edge ←
    - 1 Riemann flux
  - perform projections already with element kernel




```
1:  Receive loop:
2:  for all element e ∈ {1, ..., E} do
3:      for all edges belonging to Ω_e do
4:          receive projection p_Δ from element channel   ←
5:      end for
6:  end for
7:  Compute and send loop:
8:  for all element e ∈ {1, ..., E} do
9:      initialize flux update term r_flux(c_Δ) ← 0
10:     Loop over three edges per element:
11:     for all edges belonging to Ω_e do
12:         Riemann flux Â ← riemann(A(p_Δ), A(p_Δ^+), p_Δ, p_Δ^+)
            (Eq. 10)
13:         update r_flux(c_Δ) ← r_flux(c_Δ) + Â
14:     end for
15:     send r_flux(c_Δ) to accumulator channel
16: end for
```



FPGA Results

Parallelism and Synthesis Results

- Parallel and pipelined operations
  - 1 element integral + projections per cycle
  - 3 edge integrals per cycle
  - accumulation + min. depth 1 element / cycle

- DSPs (and logic) for arithmetic
- Fit data into block RAMs (8953 available)
  - edge kernel: multiple copies for projection
  - larger mesh requires more space
  - higher order requires more space

FLOPs per element or edge

Order	elements	pro-jection	edges	accum.	min. depth	sum
PC	106	27	$3 \cdot 87$	15	3	412
PL	634	162	$3 \cdot 210$	90	9	1525
PQ	2295	486	$3 \cdot 396$	270	18	4256
PQ CPU	2286	$3/2 \cdot 873$		162	54	3811.5

Synthesis Results

Order	max. elements	Logic Slices	Block RAMs	DSPs	Frequency [MHz]
PC	2048	23%	1923	457	354.17
	4096	24%	2805	457	341.66
	8192	25%	4569	457	312.50
	16384	26%	8083	457	284.38
PL	2048	36%	3037	1194	320.00
	4096	37%	4694	1194	309.37
	8192	39%	7994	1194	285.00
PQ	2048	59%	4924	2773	216.67
	4096	61%	8063	2773	208.33

Performance Model + Example

- Cycles per iteration = #elements + Latency + #external edges
  - e.g. 1696 + 562 + 156 = 2414 cycles for Bahamas benchmark
  - 2414 cycles @ 354.17 MHz = 6.8µs
  - 146715 time steps / s
  - @5s time steps = 8.5 simulated days / s

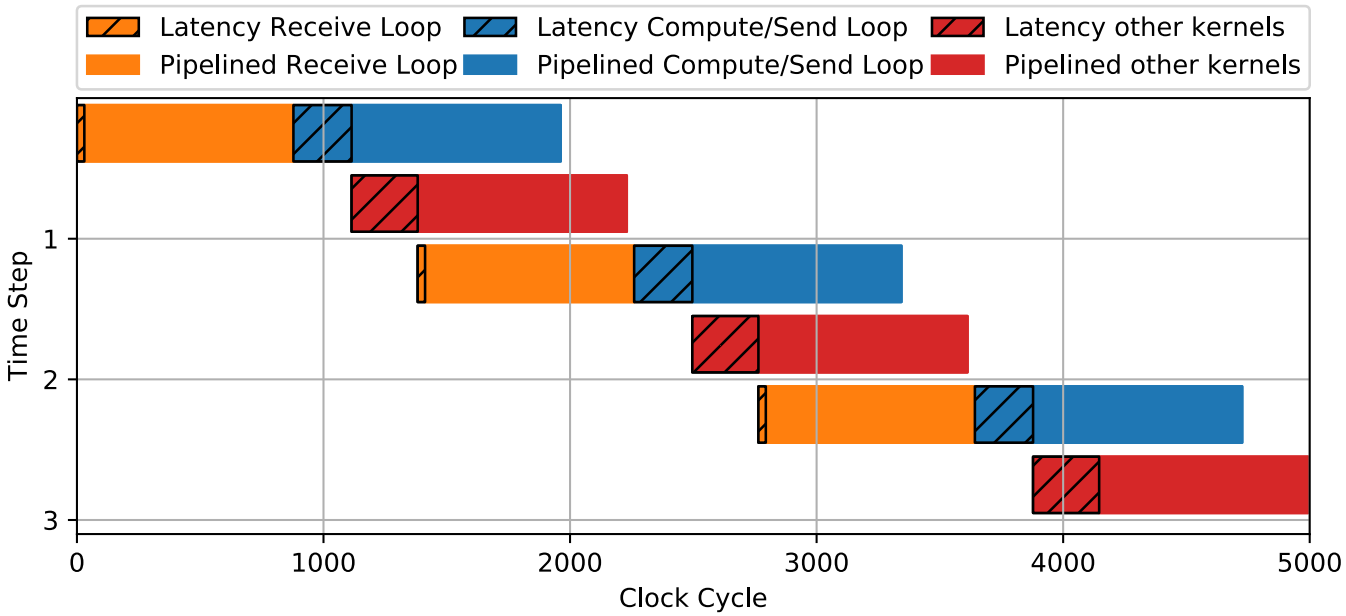


Illustration ~ 1/2 Bahamas, 848 elements

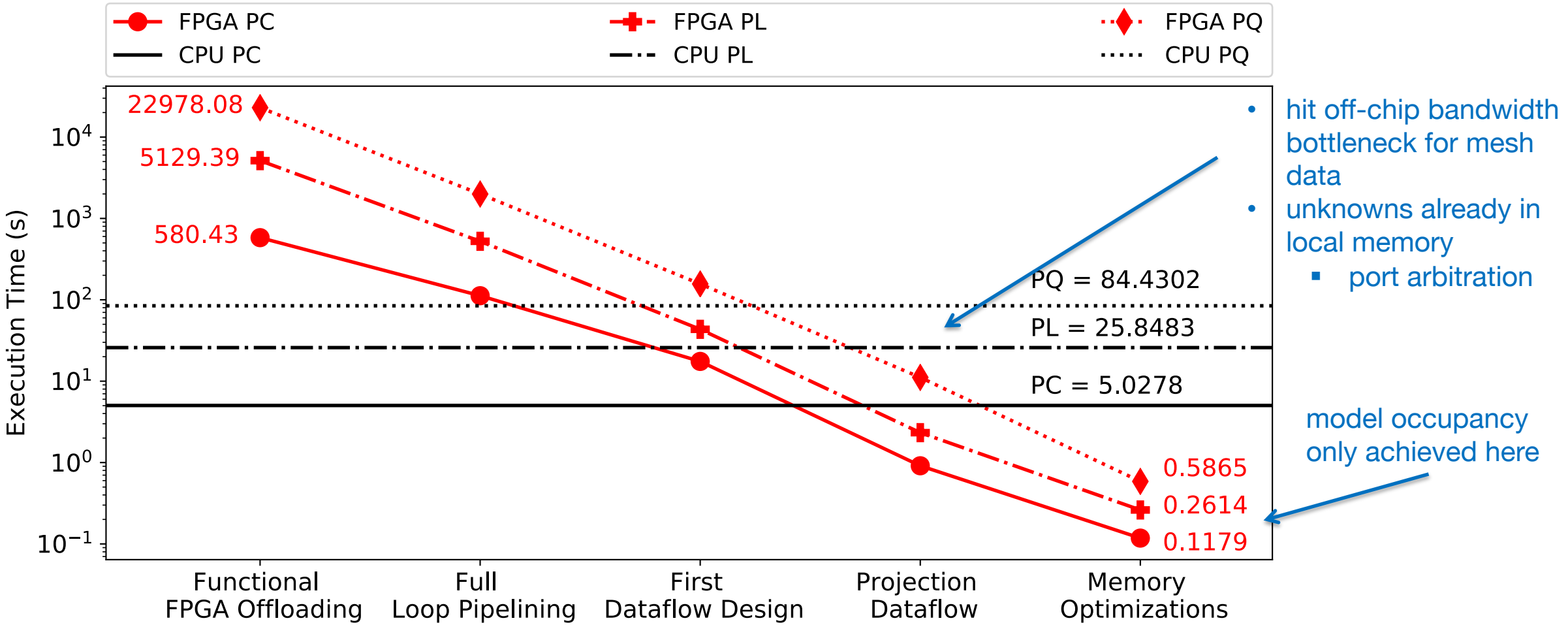
Performance Model vs. Measurements

- cycles per iteration = #elements + Latency + #external edges
- occupancy = #elements / cycles per iteration
- FLOPS = Occupancy * peak FLOPS

Ord.	E	D_{ext}	L	model *occ.*	model GFLOPs	measured GFLOPs	power [W]
PC	1696	156	562	70.3%	102.5	102.4	74.0
	3392	192	562	81.8%	115.2	114.9	74.5
	6784	312	562	88.6%	114.1	113.9	76.0
	13568	384	562	93.5%	109.5	109.3	77.9
PL	1696	156	569	70.1%	341.9	341.8	76.9
	3392	192	569	81.7%	385.3	384.8	78.5
	6784	312	569	88.5%	384.7	384.1	80.3
PQ	1696	156	592	69.4%	639.9	637.9	77.7
	3392	192	592	81.2%	720.2	717.7	78.9

Recap FPGA Optimization Process

- five main design iterations

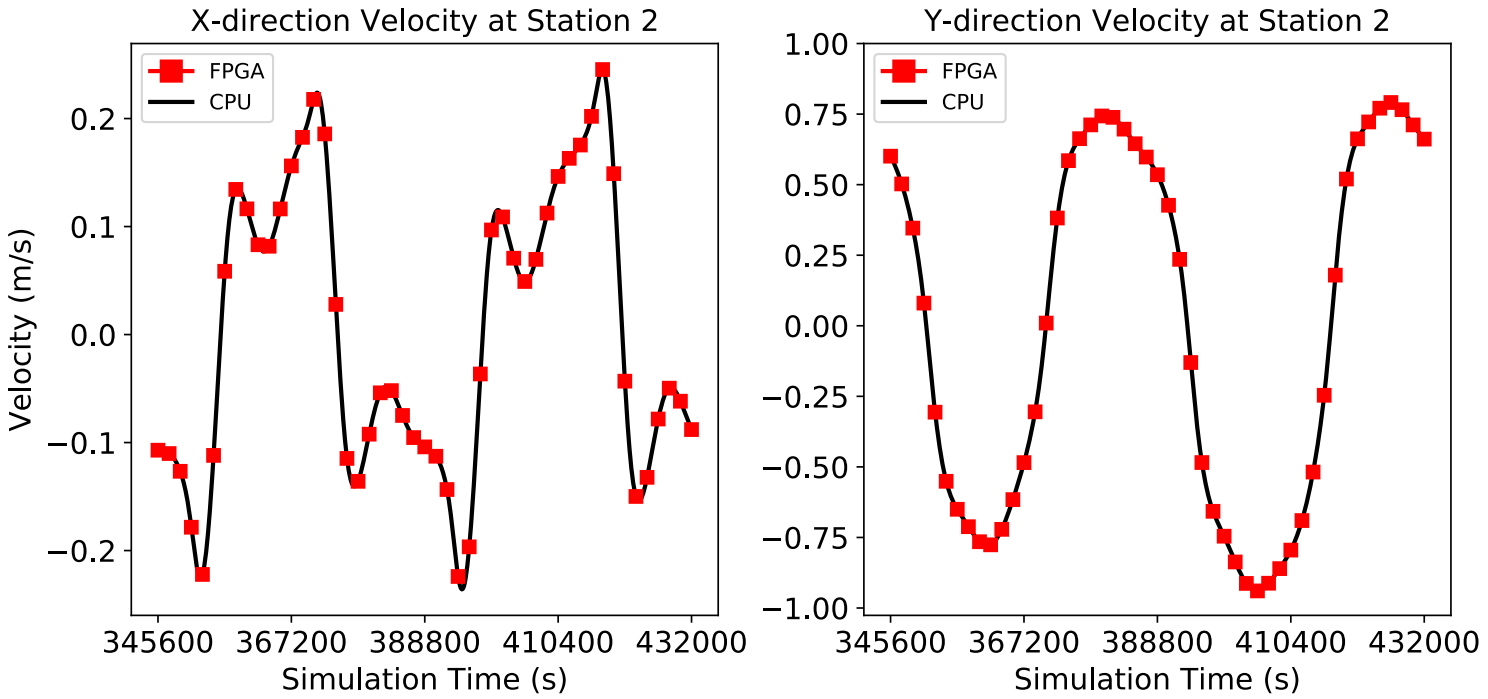


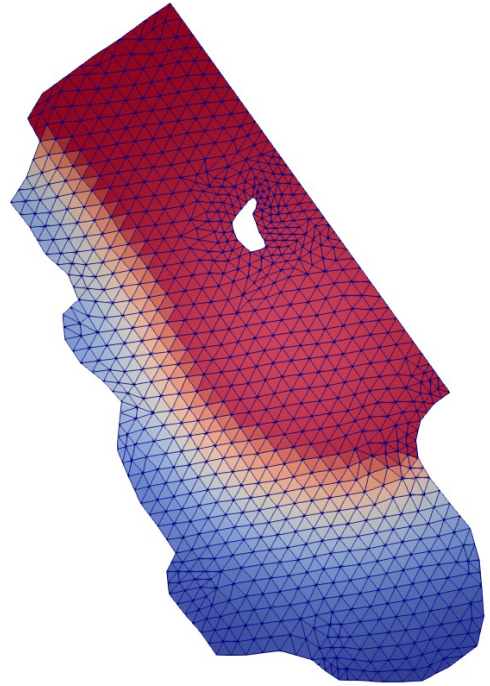
- hit off-chip bandwidth bottleneck for mesh data
- unknowns already in local memory
  - port arbitration

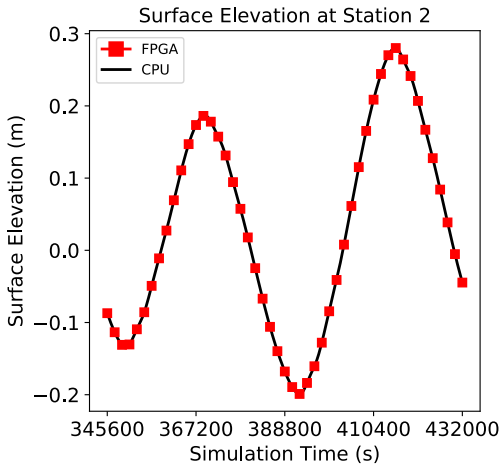
model occupancy only achieved here

Validation

- Time series and elevation maps
  - only minor numeric differences (due to reordering, rounding)







Ongoing and Future Work

- Scaling
  - multiple pipelines per FPGA for PC, PL
  - multiple FPGAs
  - larger meshes with HBM2 and / or temporal blocking
- Further Co-Design
  - block-structured meshes
- Abstractions
  - separation between algorithm and architecture?
- Hybrid execution modes
  - coupling with other models

Summary

- Dataflow architecture on FPGA
  - all kernels in element sequence
  - co-design: projection
- Performance
  - hundreds to thousands operations per cycle
  - up to 720 GFLOPs measured
  - up to 144x speedup over 1 CPU core
  - on small problems