# Whole program generation for Ocean Models

Prof. Dr. Harald Köstler [a], Sebastian Kuckuk [a], Sara Faghih-Naini [a,b], Daniel Zint [a], Vadym Aizinger [b], Roberto Grosso [a]

[a] Friedrich-Alexander-Universität Erlangen-Nürnberg, [b] Universität Bayreuth
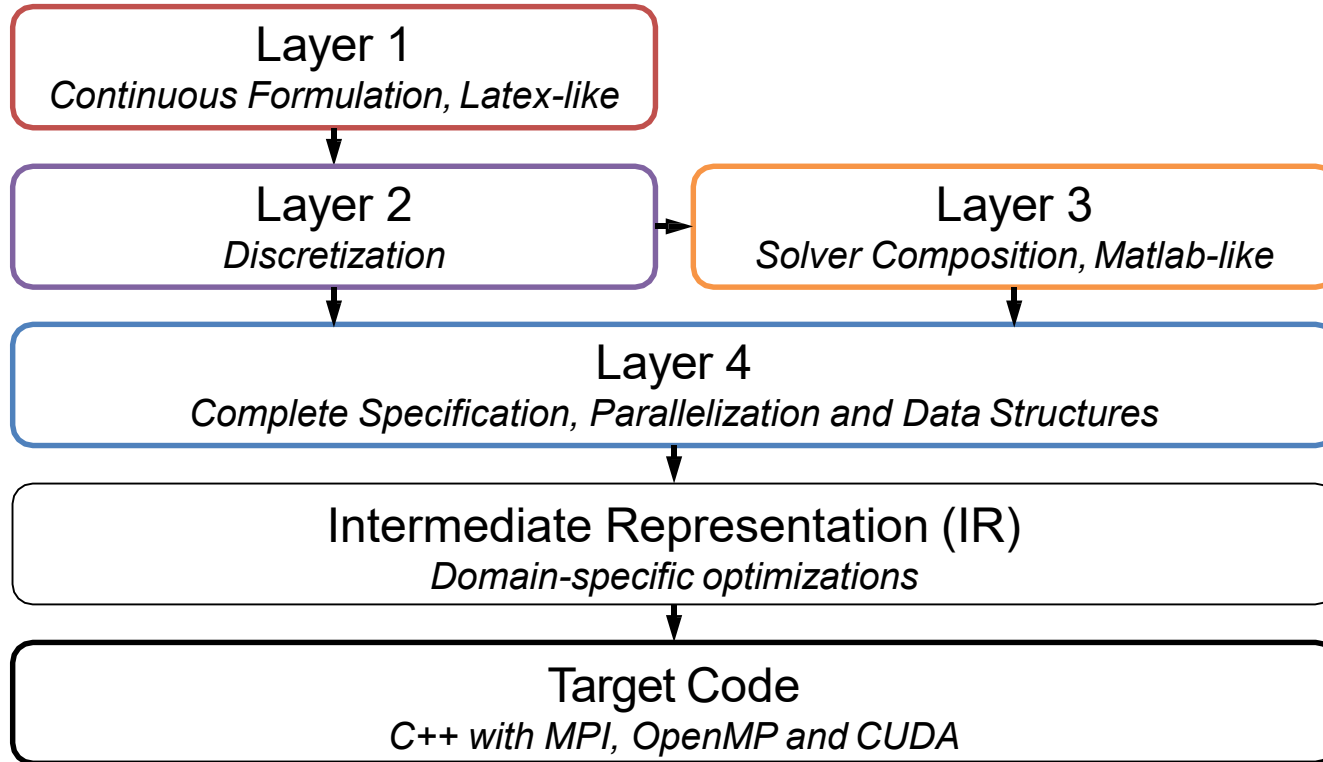
Research Vision

- **Automize the HPC software development process**
- **Increase**
  - *Performance*: by smart numerical algorithms and performance engineering
  - *Portability*: by using domain-specific compilers or code transforms to produce concrete implementations
  - *Productivity*: by working on different layers of abstract descriptions and separation of concerns

Problem Model

Exploration Validation

Insight Design
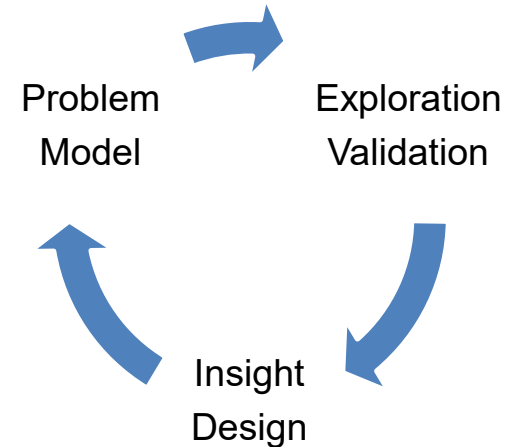
The full program generation ExaStencils approach (for geometric multigrid solvers)

Workflow (DFG SPPEXA 2013-2019)

Layer 1
Continuous Formulation, Latex-like

Layer 2
Discretization

Layer 3
Solver Composition, Matlab-like

Layer 4
Complete Specification, Parallelization and Data Structures

Intermediate Representation (IR)
Domain-specific optimizations

Target Code
C++ with MPI, OpenMP and CUDA
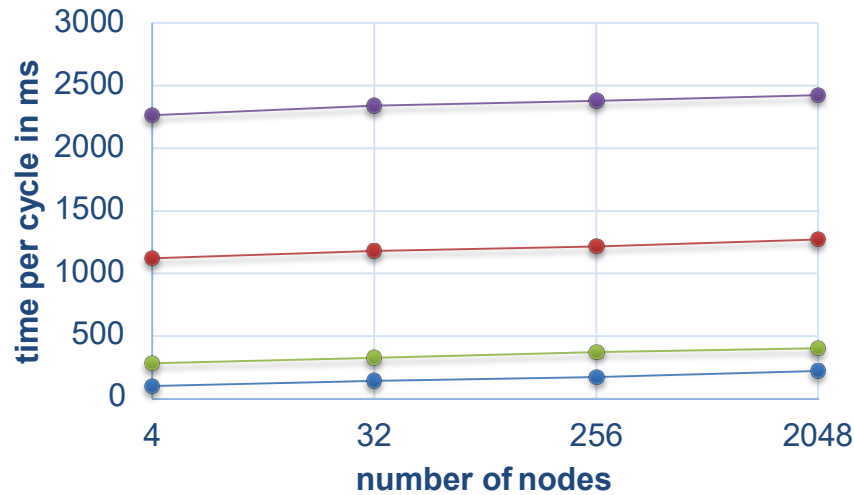
ExaStencils approach

- External DSL
- Hierarchical DSL
- Code generator written in Scala
- Focus on geometric multigrid solvers on structured grids
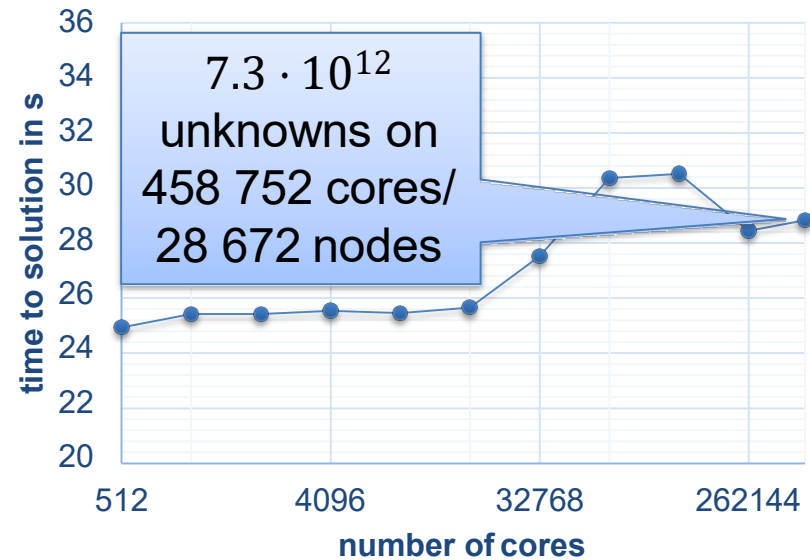- Backends to CUDA, OpenCL, C++ with OpenMP, MPI
- Whole-program generation

Does it run well on supercomputers?

- Pure MPI & CUDA

- Hybrid MPI/ OpenMP

$$7.3 \cdot 10^{12}$$
unknowns on 458 752 cores/ 28 672 nodes

CC - GPU   CC - CPU   VC - GPU   VC - CPU

The full simulation generation ExaStencils approach (for block-structured grids and DG)

Mathematical model for the 2D Shallow Water Equations (SWE)

$$\frac{\partial \xi}{\partial t} + \nabla \cdot (\boldsymbol{U}) = 0 \tag{1}$$

$$\frac{\partial (\boldsymbol{U})}{\partial t} + \nabla \cdot ((\boldsymbol{U})(\boldsymbol{U})^T / H) + \tau_{bf}\boldsymbol{U} + f_c \boldsymbol{k} \times \boldsymbol{U} + gH\nabla\xi = \boldsymbol{F} \tag{2}$$

ξ : elevation of the free water surface

$H = h_b + \xi$: total fluid depth

h_b : bathymetric depth

$\boldsymbol{U} = (U, V)^T$: depth integrated horizontal velocity field

f_c : Coriolis parameter

$\boldsymbol{k}$: local vertical vector

g : gravitational acceleration

τ_{bf} : bottom friction coefficient

$\boldsymbol{F}$: forcing term
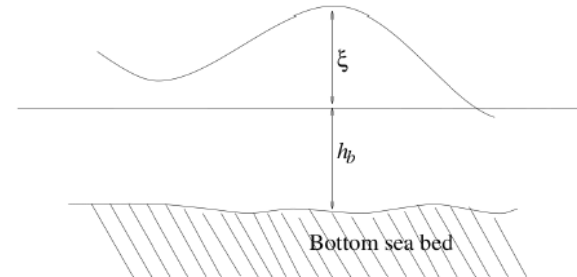
Figure: Definition of elevation and bathymetry

Aizinger, Dawson. *A discontinuous Galerkin method for two-dimensional flow and transport in shallow water*. Advances in Water Resources 25.1, 2002.

Evaluation on GPU cluster Piz Daint

- Finite Volumes on a uniform grid is perfectly suited for ExaSlang

Legend: computation, communication
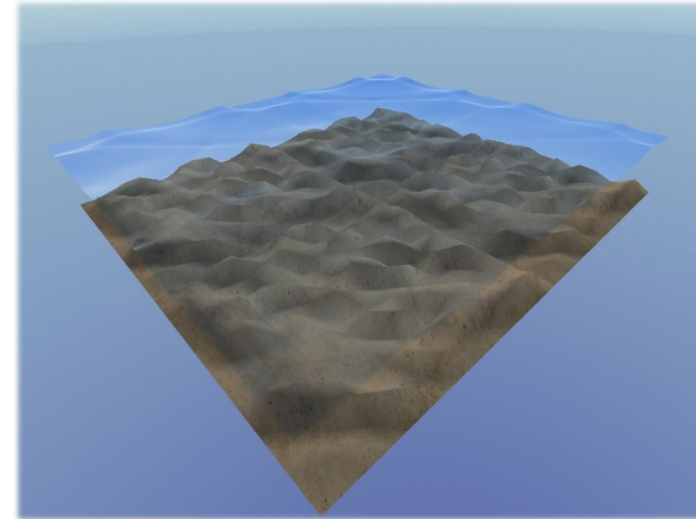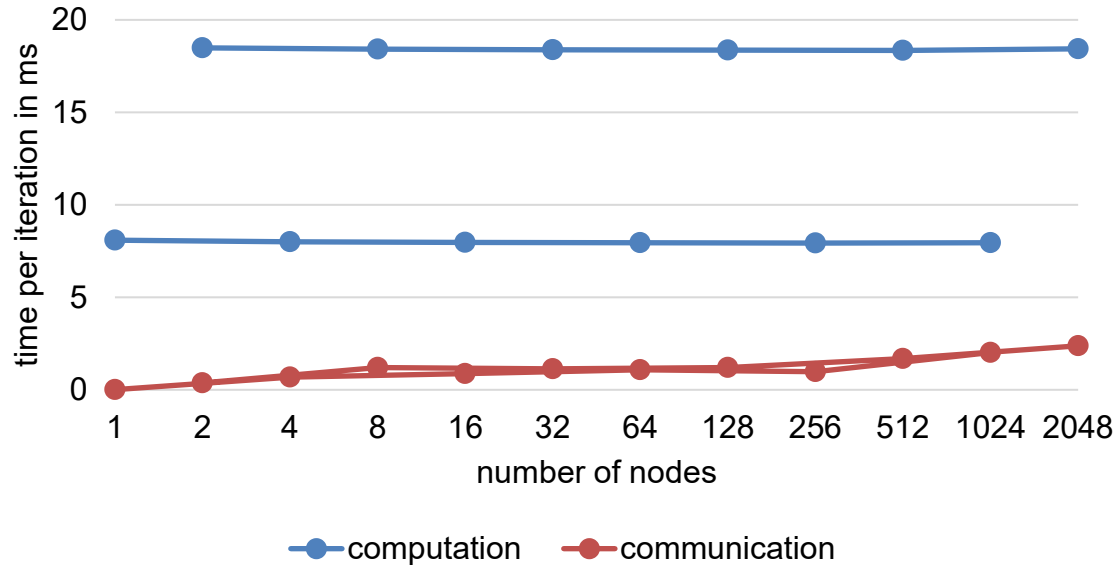
Kuckuk, Köstler. *Whole Program Generation of Massively Parallel Shallow Water Equation Solvers*. CLUSTER'18
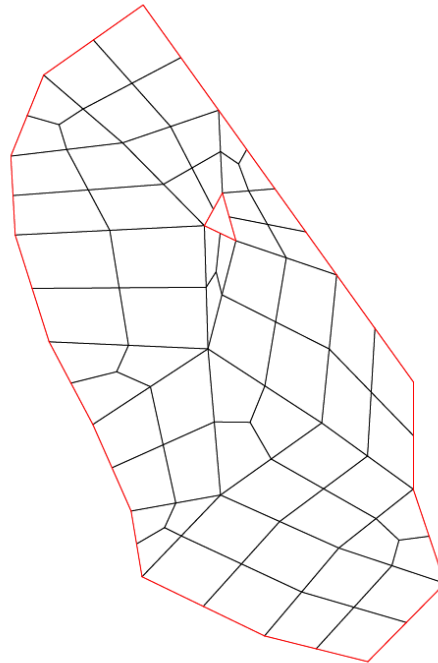
Extended Toolchain (DFG project 2016-2020)
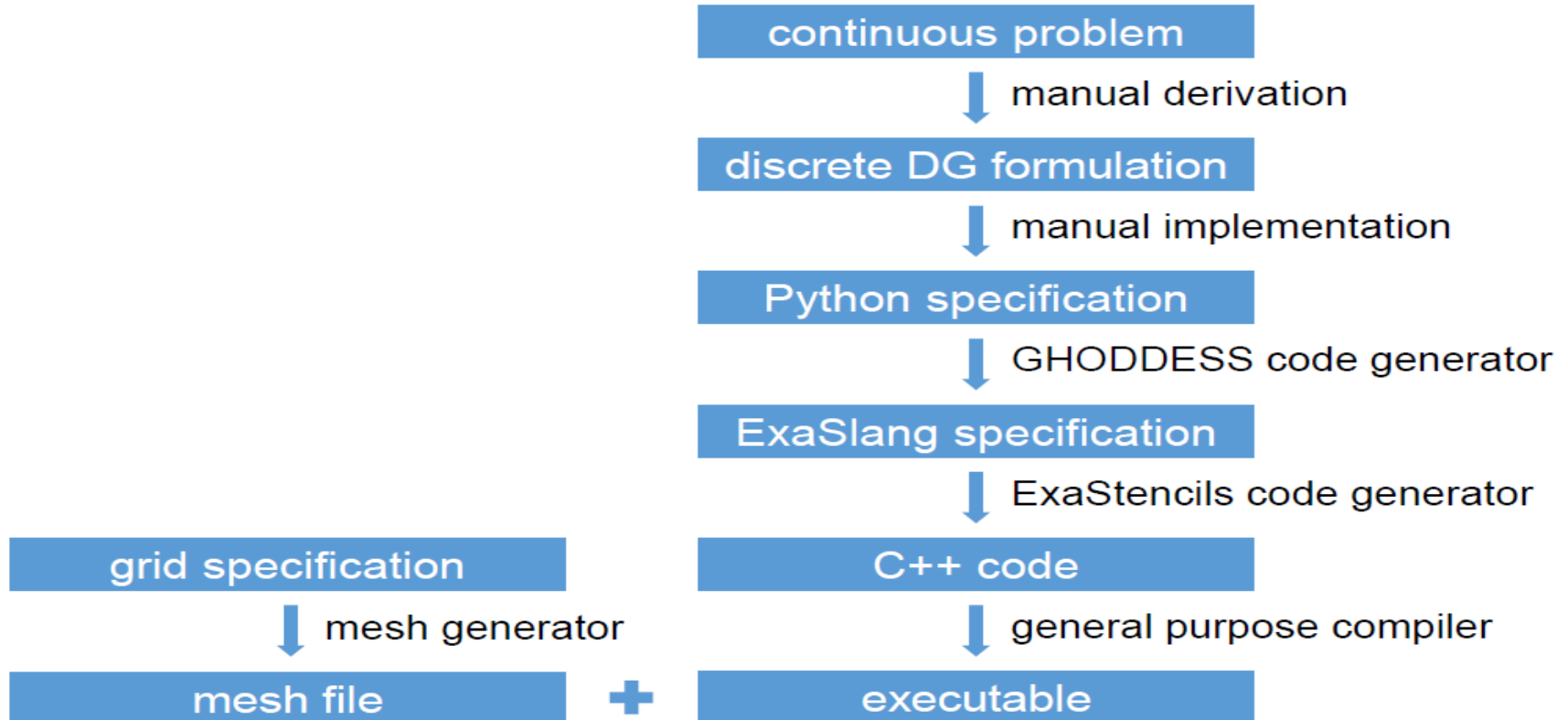
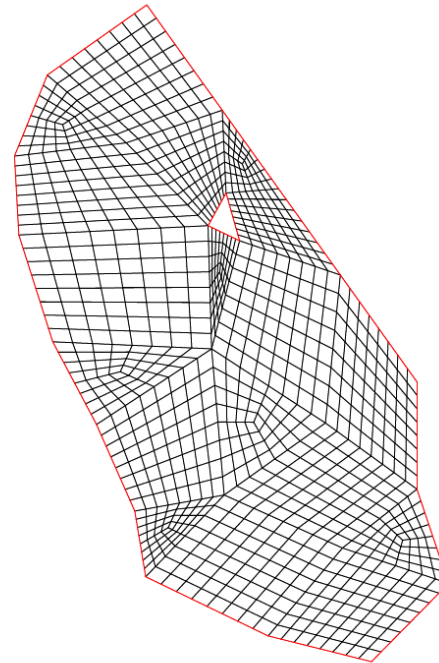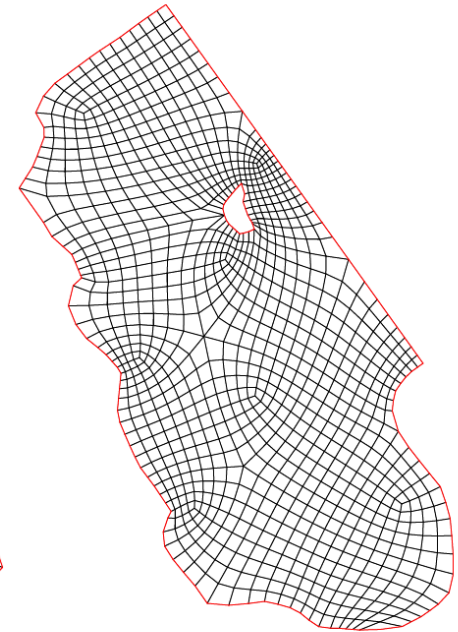# Block-structured grids in ExaStencils

(a) Triangle mesh   (b) Patches   (c) Refined   (d) BSG

Zint, Grosso. *Generation of Quadrilateral Block Structured Grids on Complex Domains for High Performance Simulation in Ocean Modeling,* 2020.
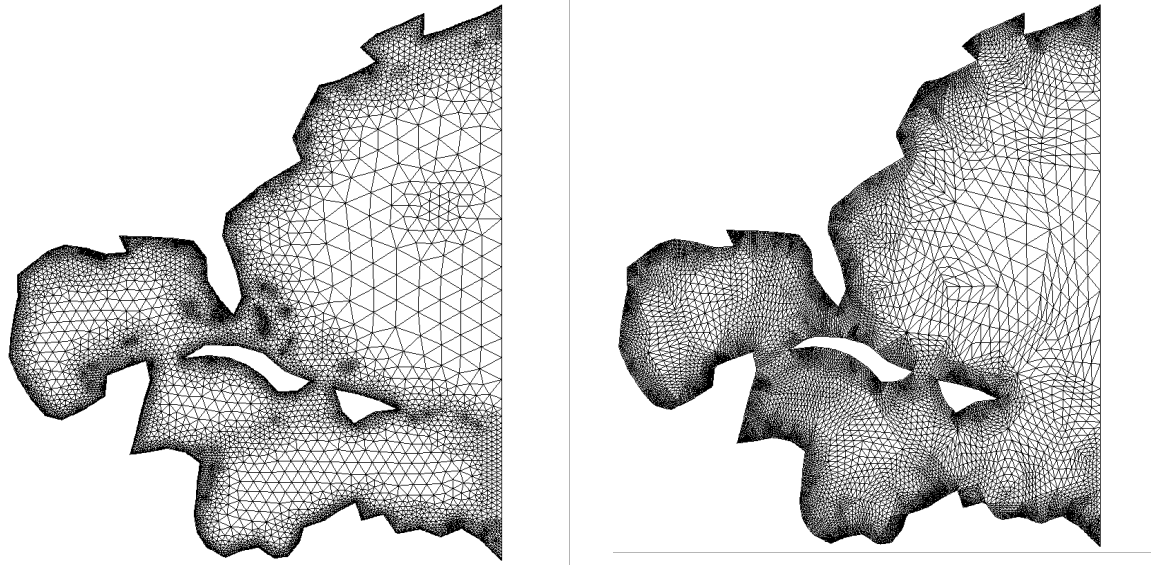
Numerical Results for Tide-driven Flow

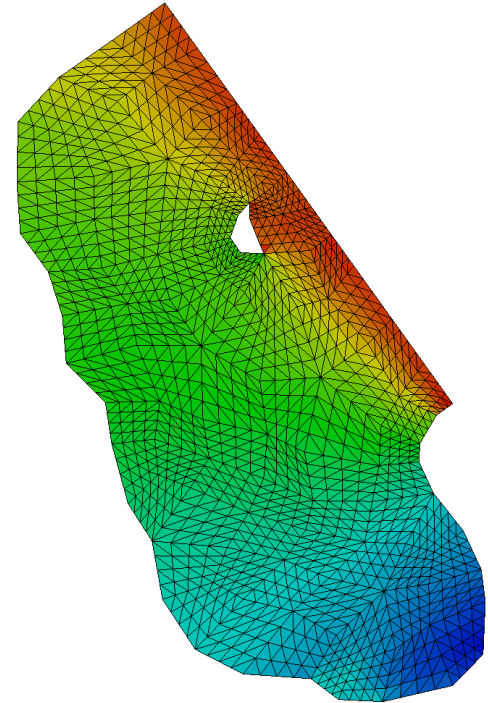**Example**: Flow at the US East coast with tidal forcing at open sea boundary

Finite element mesh with bathymetry for unstructured grid (left) and block-structured grid (right).

Reuter, Aizinger, Köstler. *A multi-platform scaling study for an OpenMP parallelization of a discontinuous Galerkin ocean model.* Computers & Fluids, 2015.
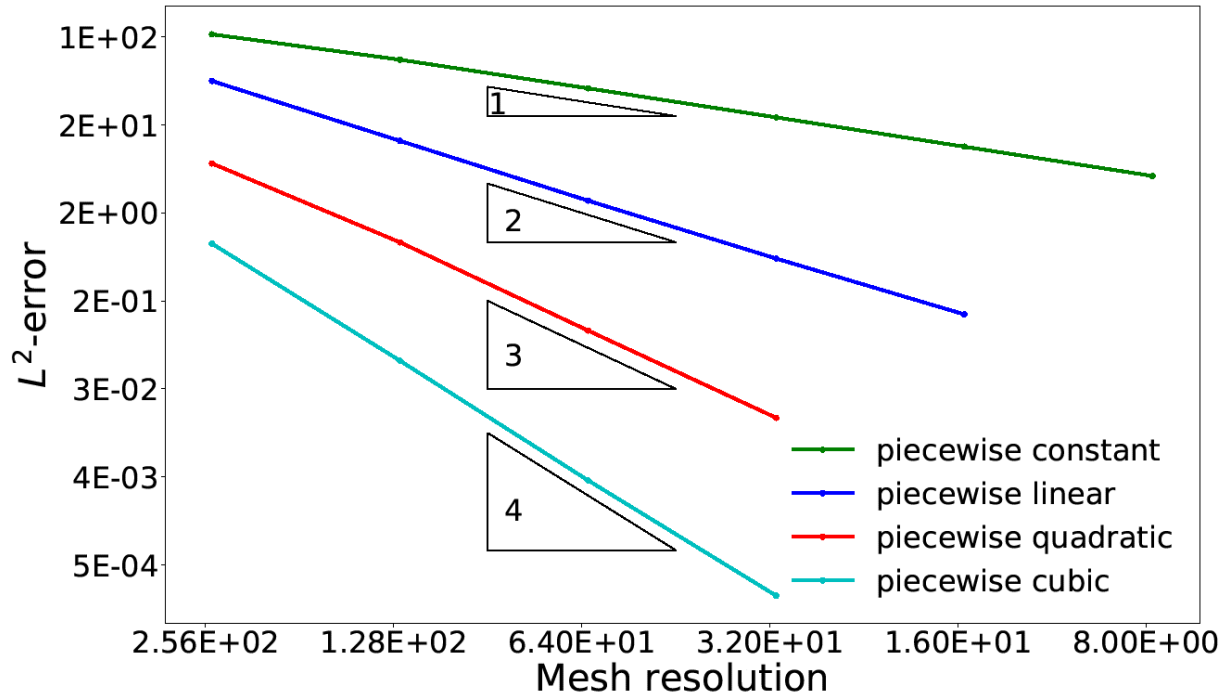
Required extensions

- Block-structured grids

  - Extension of the generator

- Higher order discretization: discontinuous Galerkin (DG)

  - Implementation as Python front end

$$P_2 \begin{cases} P_1 \begin{cases} P_0 \begin{cases} \widehat{\varphi}_1(\widehat{\boldsymbol{x}}) = \sqrt{2} \\ \\ \widehat{\varphi}_2(\widehat{\boldsymbol{x}}) = 2 - 6\hat{x}_1 \\ \widehat{\varphi}_3(\widehat{\boldsymbol{x}}) = 2\sqrt{3}(1 - \hat{x}_1 - 2\hat{x}_2) \\ \widehat{\varphi}_4(\widehat{\boldsymbol{x}}) = \sqrt{6}\big((10\hat{x}_1 - 8)\hat{x}_1 + 1\big) \\ \widehat{\varphi}_5(\widehat{\boldsymbol{x}}) = \sqrt{3}\big((5\hat{x}_1 - 4)\hat{x}_1 + (-15\hat{x}_2 + 12)\hat{x}_2 - 1\big) \\ \widehat{\varphi}_6(\widehat{\boldsymbol{x}}) = 3\sqrt{5}\big((3\hat{x}_1 + 8\hat{x}_2 - 4)\hat{x}_1 + (3\hat{x}_2 - 4)\hat{x}_2 + 1\big) \end{cases} \end{cases} \end{cases}$$

Faghih-Naini, Kuckuk, Aizinger, Zint, Grosso, Köstler. *Towards whole program generation of quadrature-free discontinuous Galerkin methods for the shallow water equations.* Advances in Water Resources, 2020.

Convergence results for different orders

Mapping Math to Python

GHODDESS (Generation of Higher-Order Discretizations Deployed as ExaSlang Specifications)

► Uses Python library sympy[2] (analytical differentiation and integral evaluation)

► Contains classes representing triangles and data fields

► Supports quadrilateral grids only, where each element is divided into two differently oriented triangles in order to obtain a triangular grid

► Example: First part of element integral for ξ

$$\sum_{i=1}^{K(k)} \frac{c_{ei}^2}{|\det(\boldsymbol{B}_e)|} \left(B_{2,2}^e \int_{\hat{\Omega}} \frac{\partial \hat{\varphi}_p}{\partial \hat{x}_1} \hat{\varphi}_i \mathrm{d}\hat{x} - B_{2,1}^e \int_{\hat{\Omega}} \frac{\partial \hat{\varphi}_p}{\partial \hat{x}_2} \hat{\varphi}_i \mathrm{d}\hat{x} \right)$$
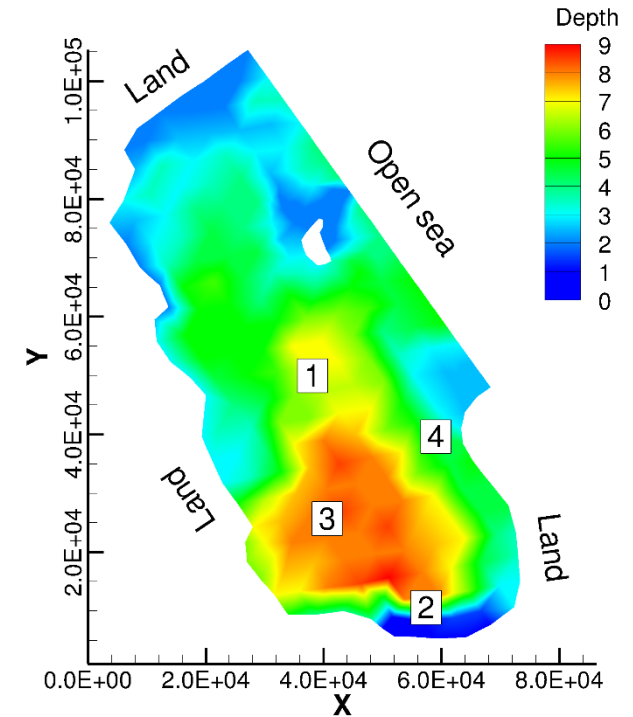
is translated to

```
sum( tri.b[1, 1] * cu(tri.orientation, k)
    * integrate_over_tri(basis.phi[k] * basis.phi_dx[p])
  - tri.b[1, 0] * cu(tri.orientation, k)
    * integrate_over_tri(basis.phi[k] * basis.phi_dy[p])
  for k in range(d)) * tri.det_b_inv
```
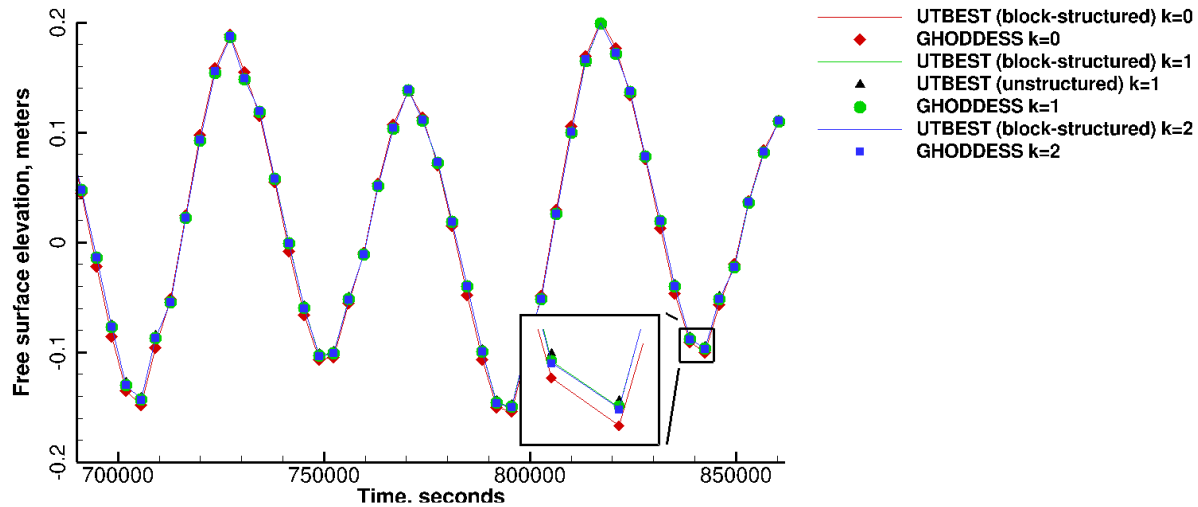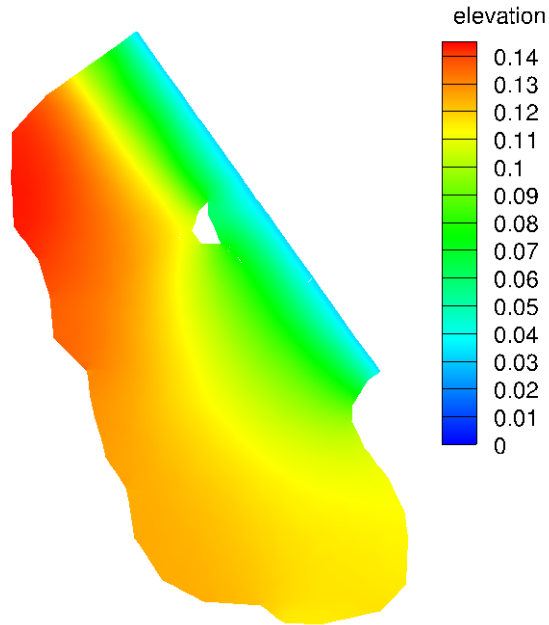
[2]**https://www.sympy.org**
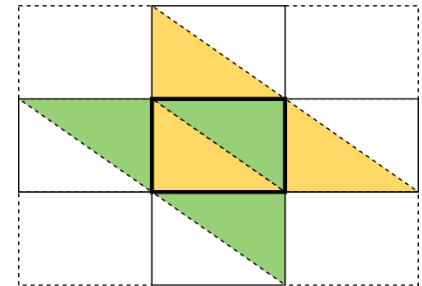
Tidal flow at Bahamas
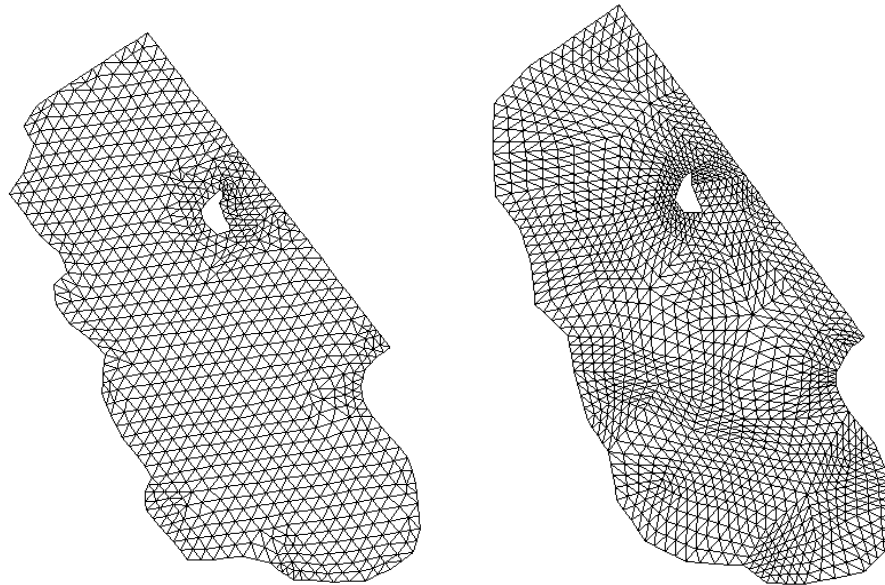
Original UTBEST unstructured mesh (left), automatically generated block-structured mesh used by the GHODDESS framework (middle), bathymetry and positions of recording stations (right).

Numerical results for Bahamas

Runtimes per time step in ms for different scenarios

k	# elements	UTBEST (1)	GHODDESS (1)	GHODDESS (16)	α_{UT}	k	# elements	UTBEST (1)	GHODDESS (1)	GHODDESS (16)	α_{UT}
		Analytical example on square domain with perturbed mesh						Tidal flow around Bahamas			
0	128	4.60E−02	1.94E−02	1.62E−01	2.38	0	2634	4.24E−01	3.29E−01	7.33E−01	1.29
	512	1.79E−01	6.63E−02	2.15E−01	2.69		10496	1.94E+00	8.65E−01	8.13E−01	2.39
	2 048	7.39E−01	2.42E−01	2.60E−01	3.05		41984	1.07E+01	2.95E+00	1.20E+00	8.96
	8 192	3.07E+00	9.87E−01	3.95E−01	7.77		167936	6.67E+01	1.30E+01	2.50E+00	26.61
	32 768	1.40E+01	3.93E+00	7.96E−01	17.6	1	2634	2.17E+00	2.10E+00	2.04E+00	1.06
1	128	3.15E−01	7.87E−02	3.19E−01	4.00		10496	1.05E+01	4.64E+00	2.35E+00	4.46
	515	1.23E+00	2.86E−01	4.11E−01	4.30		41984	5.72E+01	1.99E+01	4.03E+00	14.2
	2 048	4.96E+00	1.12E+00	6.21E−01	7.99		167936	3.21E+02	9.92E+01	1.62E+01	19.8
	8 192	2.00E+01	5.06E+00	1.14E+00	17.6	2	2634	1.15E+01	2.34E+01	6.08E+00	1.90
2	128	1.06E+00	5.84E−01	8.13E−01	1.31		10496	1.26E+02	4.21E+01	8.08E+00	15.6
	512	4.20E+00	2.09E+00	9.41E−01	3.85		41984	7.29E+02	1.60E+02	2.18E+01	33.5
	2 048	1.67E+01	8.93E+00	1.80E+00	8.15		167936	1.48E+03	8.11E+02	1.27E+02	11.6

p-adaptivity (US east coast)
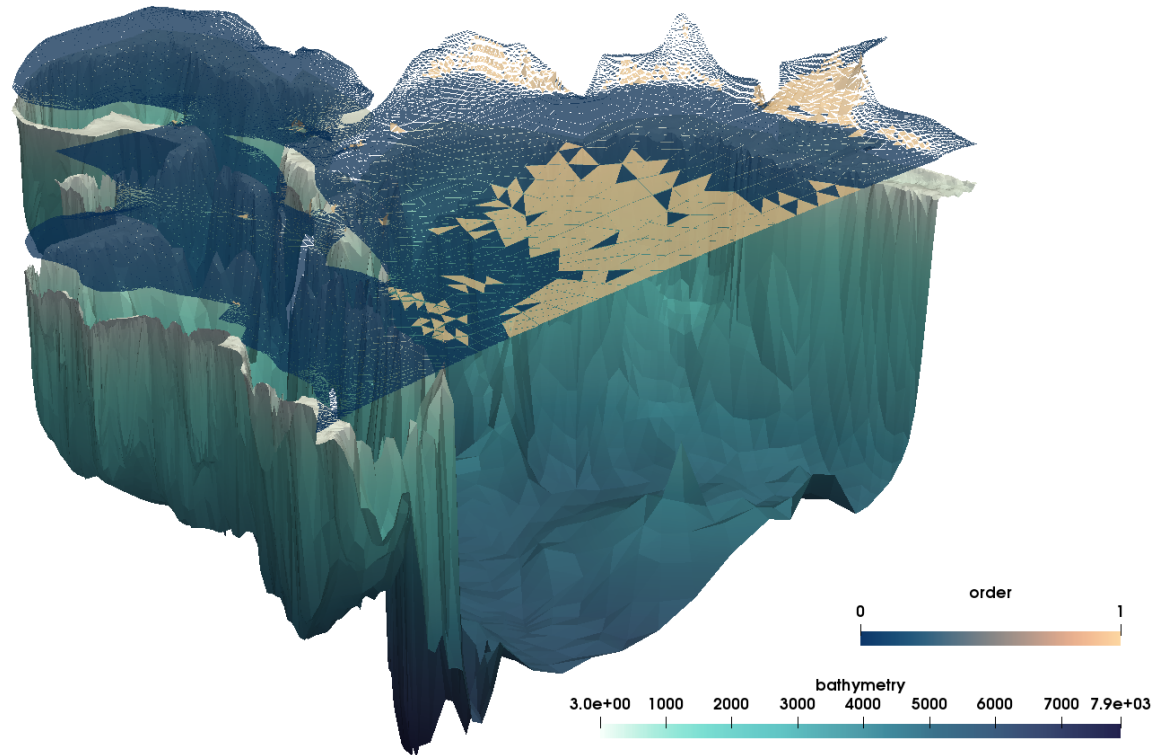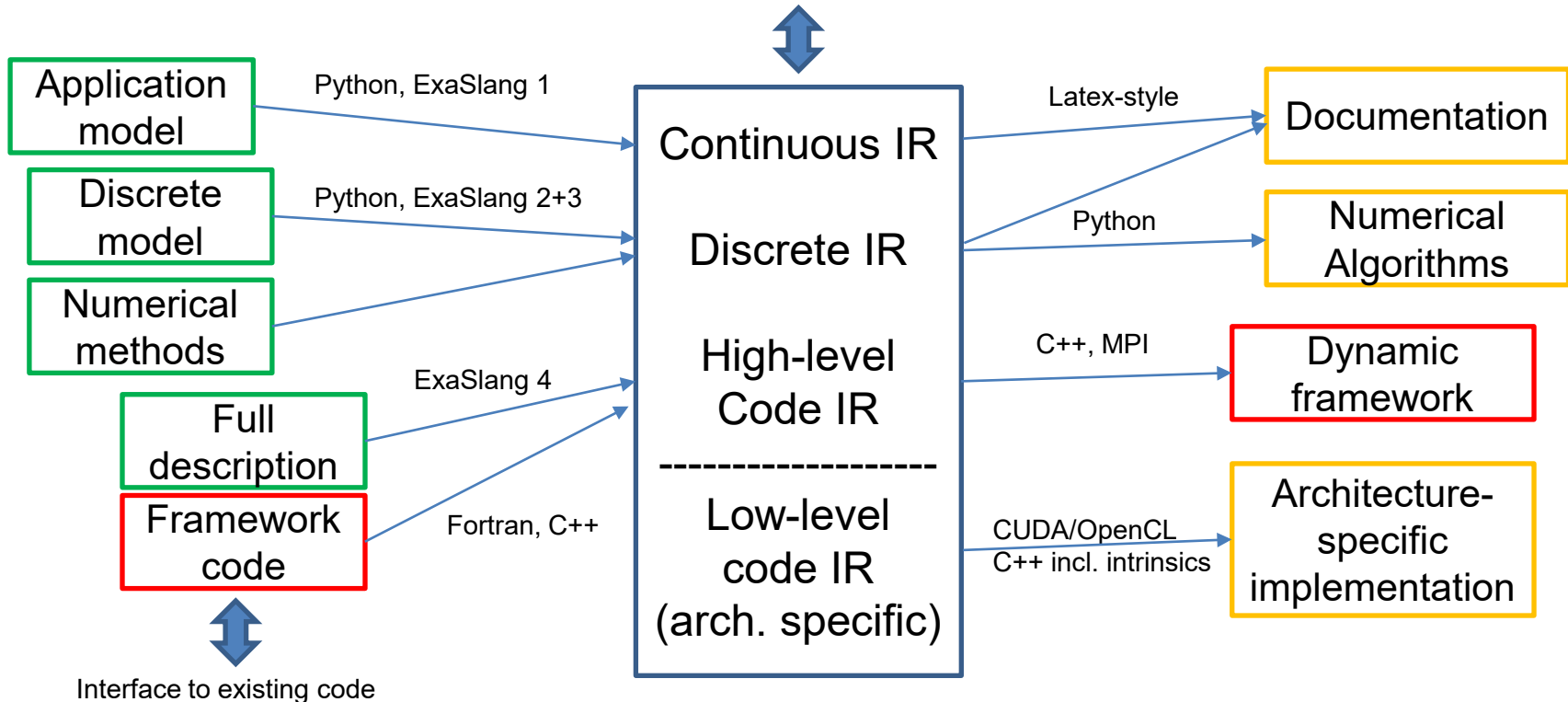
Conclusion

- The ExaSlang language stack and our code generation framework allows
  - <u>Full program generation</u> from <u>domain-specific</u> representations
  - **If** existing models and numerical methods are implemented or translated to ExaSlang
  - **If** the underlying block-structured grids can be used
  - And is also fast, **if** the required domain-specific optimizations are integrated in the code generator
  - Next: provide interfaces to other technologies and codes!

Towards dynamic HPC software frameworks

Proposed Toolchain (DFG project 2020-)

Interfaces to other (multi-level) IRs

Source	Interface	Central IR	Interface	Output
Application model	Python, ExaSlang 1	**Continuous IR**	Latex-style	Documentation
Discrete model	Python, ExaSlang 2+3	**Discrete IR**	Python	Numerical Algorithms
Numerical methods				
Full description	ExaSlang 4	**High-level Code IR**	C++, MPI	Dynamic framework
Framework code	Fortran, C++	**Low-level code IR (arch. specific)**	CUDA/OpenCL C++ incl. intrinsics	Architecture-specific implementation

Interface to existing code

MITgcm/seaice Coupling

momentum equation

$$m\frac{D\boldsymbol{u}}{Dt} = -mf\boldsymbol{k} \times \boldsymbol{u} + \tau_{air} + \tau_{ocean} - m\nabla\Phi(0) + \nabla \cdot \boldsymbol{\sigma}$$

shear stress terms

$$\tau_{ocean/air} = \rho_{ocean/air} C_{ocean/air} |\mathbf{U}_{ocean/air} - \mathbf{u}| R(\mathbf{U}_{ocean/air} - \mathbf{u})$$
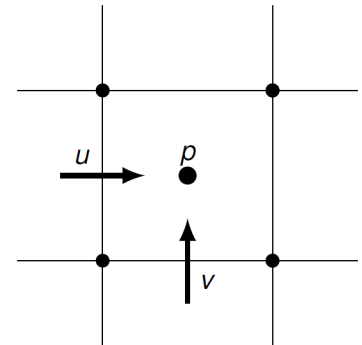
http://mitgcm.org/

Figure: The staggered C-grid used by MITgcm and seaice

Fortran framework integration

- Analog to OpenMP and OpenACC annotations
- ExaSlang 4 code with extra modifiers if necessary
- Steps:
  - parse Fortran code
  - sort and merge individual variable/constant/field declaration statements
  - progress to layer 4 IR
  - convert absolute array index with relative offset
  - insert constants, simplify expression, remove loop variable
  - if annotated replace with given layer 4 code using parser
  - case insensitive name resolution
  - Fortran backend for ExaStencils code generation framework

Example

Abstract formulation

res = Laplace * U
return max(res)

Fortan code

```fortran
!$EXA Function Residual@all() : Double
      Real*8 function Residual()
       implicit none

       Integer NPoints, i, j
       Real*8 norm, U, RHS, RES, C, N, S, W, E
       Parameter(NPoints=42)
       Dimension U(0:NPoints+1, 0:NPoints+1)
       Dimension RHS(NPoints, NPoints)
       Dimension RES(NPoints, NPoints)
       Dimension C(NPoints, NPoints)
       Dimension N(NPoints, NPoints)
       Dimension S(NPoints, NPoints)
       Dimension W(NPoints, NPoints)
       Dimension E(NPoints, NPoints)
       Common /fields/ U, RHS, RES, C, N, S, W, E

       norm = 0
!$EXA loop over RES with reduction(max: norm) collapse(2)
       DO i = 1, NPoints
        DO j = 1, NPoints
         res(i,j) = rhs(i,j)-( C(i,j)*U(i,j) +
     &               N(i,j)*U(i-1,j) + S(i,j)*U(i+1,j) +
     &               W(i,j)*U(i,j-1) + E(i,j)*U(i,j+1) )
          norm = max(abs(res(i,j)), norm)
        ENDDO
       ENDDO

       Residual = norm
      end function
```

generated layer 4 code

```
Function Residual@all : Double {
        Var RESIDUAL : Double
        Var I : Integer
        Var J : Integer
        Var NORM : Double
        Val NPOINTS : Integer = 42

        NORM = 0
        loop over RES with reduction ( max : NORM ) {
          RES@[0, 0] = ( RHS@[0, 0] - ( ( ( ( ( C@[0, 0] * U@[0, 0] ) +
                   ( N@[0, 0] * U@[-1, 0] ) ) + ( S@[0, 0] * U@[1, 0] ) ) +
                   ( W@[0, 0] * U@[0, -1] ) ) + ( E@[0, 0] * U@[0, 1] ) ) )
           NORM = max ( abs ( RES@[0, 0] ), NORM )
        }
        RESIDUAL = NORM
        return RESIDUAL
```

Future Work

- Performance Portability: Sustained performance engineering
- Higher-order DG: Optimize p-adaptivity
- Parallel Data structures: Extend possible grid types
- New Applications: e.g. Tsunami, storm surge (add wetting/drying), particles
- AI Integration: Auto-tuned numerical methods
- Dynamic framework: Code Generation for Automatic Interfaces, MLIR
- Collaborative work with Peano, DUNE and MITgcm
- Connect to other code generation approaches in climate simulation
- Better accessibility – documentation & tutorials

Thank you for your attention!

ExaStencils is now open source:

https://i10git.cs.fau.de/exastencils/release