Friedrich-Alexander-Universität
Technische Fakultät

Code Generation Technology for HPC Codes: Case Studies on waLBerla and HyTeG

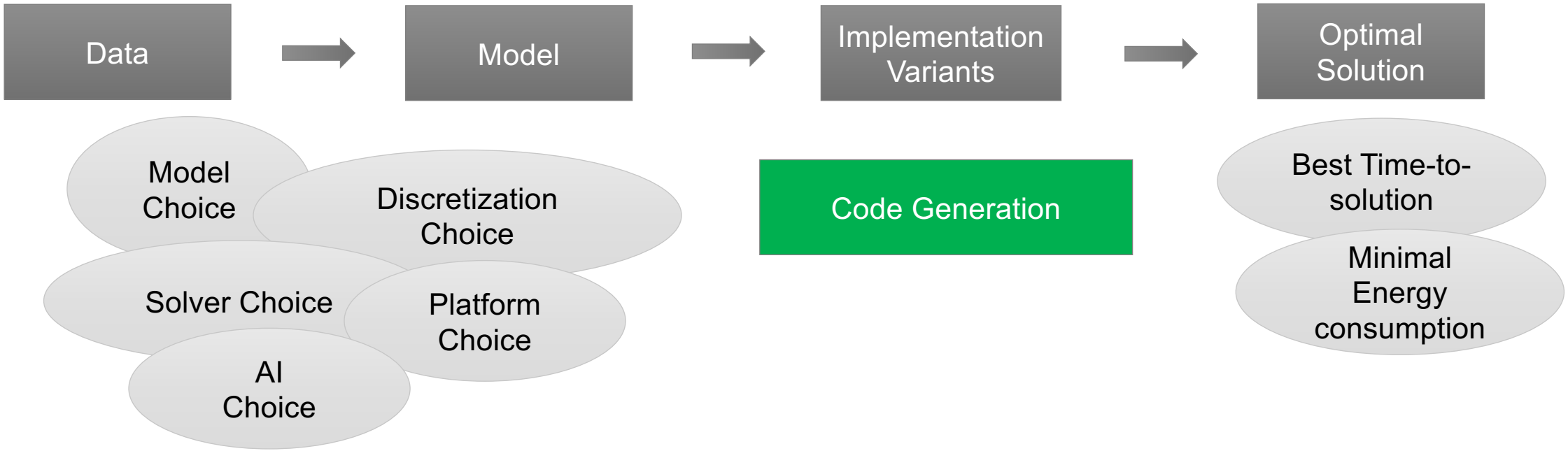
Harald Köstler[1,2], Ulrich Rüde, Markus Holzer, Fabian Böhm, Daniel Bauer, Richard Angersbach, Sebastian Kuckuk, Sara Faghih-Naini, Florian Schornbaum, Martin Bauer, Nils Kohl, Daniel Zint, Christoph Schwarzmeier, Dominik Thönnes, Samuel Kemmler, et al

1 Lehrstuhl für Systemsimulation (LSS) – Informatik X
2 Zentrum für Nationales Hochleistungsrechnen Erlangen (NHR@FAU)
Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU)

Introduction

- LSS develops open-source massively parallel software frameworks
- Special focus on (block)structured grids
- Several (multi-physics) application fields
- 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, using modern software engineering techniques

Large search space for optimal solution

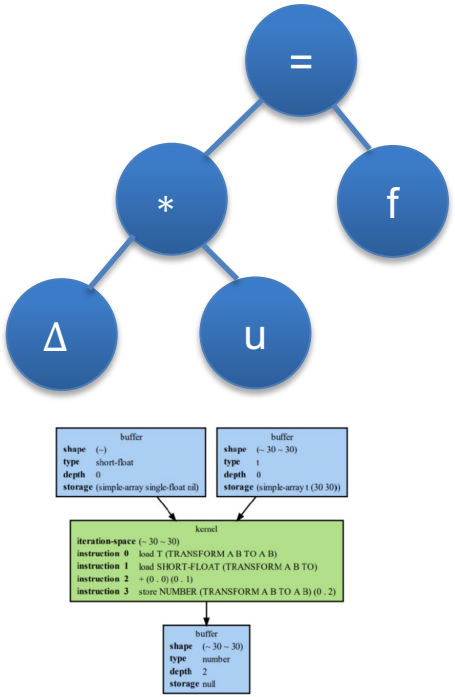


Definition: Code generation in a strict sense is used to describe the process of producing code that can be executed on a certain platform from an abstract representation

Why? Because programming is time-consuming. A domain-specific compiler should do the job.

Code Generation Process

Tree representation



Mathematical representation

$$\Delta u = f$$

Code representation 1

Field u,f
Laplace(u) = f

Code representation 2

mov EAX, [ebp+8]
add EAX, EBX

Abstract equations:
symbolic algebra

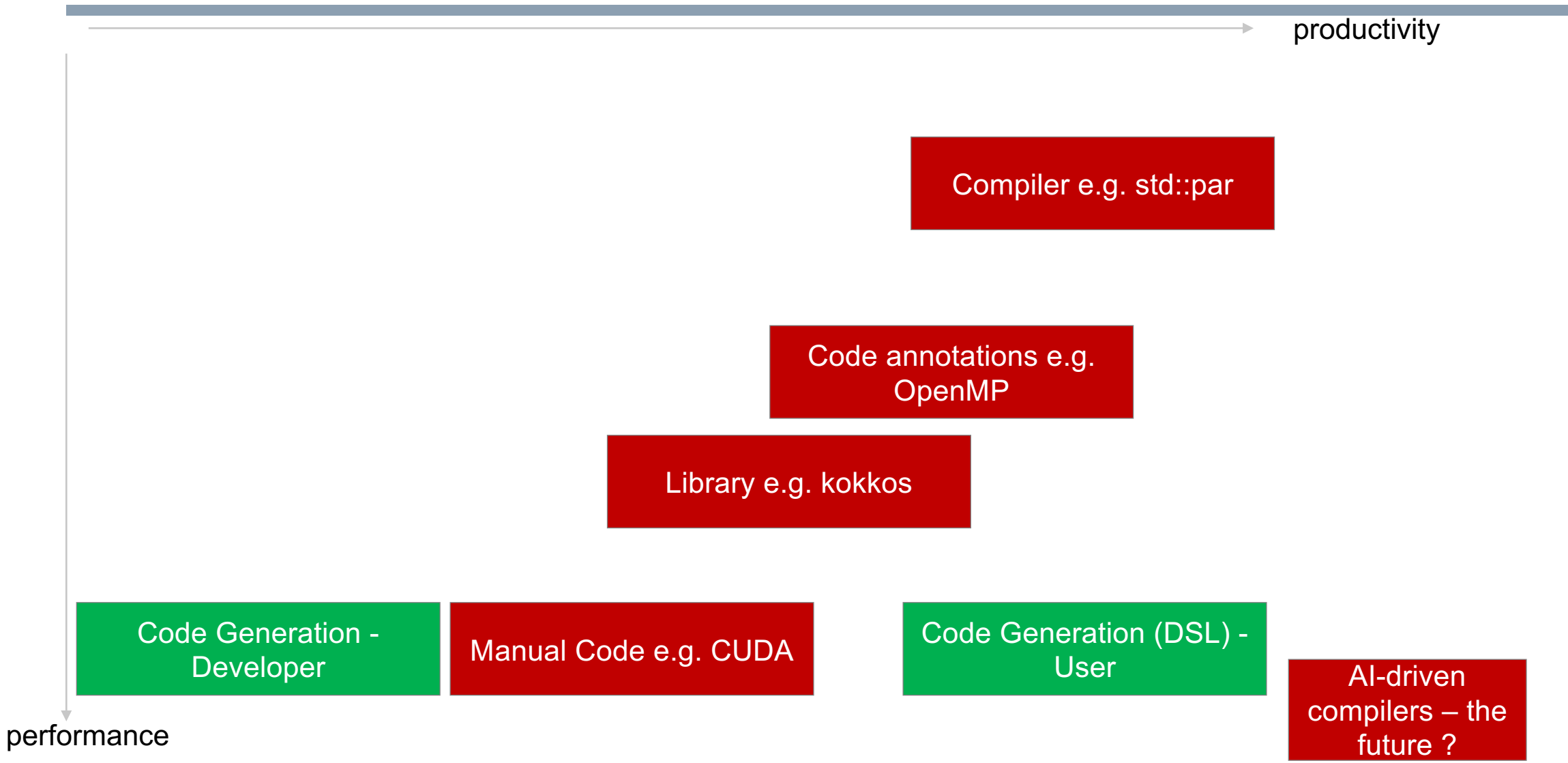
Abstract code descriptions:
includes data structures and numerical calculations

Low-level code:
architecture-specific machine code

Implement transforms to switch between different intermediate representations

Alternative approaches for portability

Discussion



Multiphysics Simulations – Complexity

- Simulation frameworks are inherently **domain-specific**
- Put multiple simulation codes into **one program**
- Different domains
  - **Algorithms**, **discretizations** and **data structures**
  - Discretizations space and time **scales**
  - **Target platforms** for execution

➢ Highly specialized and complex coupling codes

Challenge: software for multi-domain problems

Whole Program Generation - ExaStencils

ExaStencils

Whole-program Code Generation

ExaStencils

https://www.exastencils.fau.de/

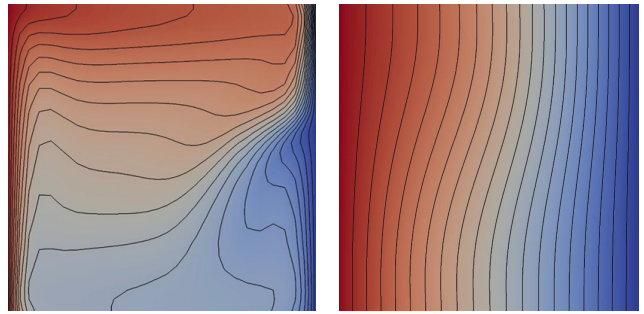
- External, hierarchical DSL ExaSlang
- Whole-program code generator written in Scala
- Focus on geometric multigrid solvers on structured grids
- Backends to CUDA, OpenCL, C++ with OpenMP, MPI

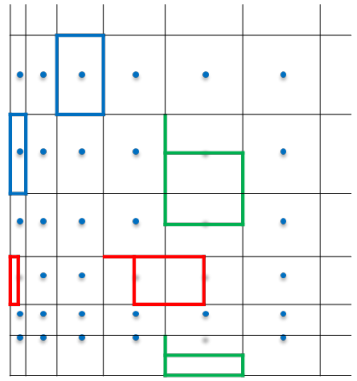
Ch. Lengauer et al. *ExaStencils: Advanced multigrid solver generation.* Software for Exascale Computing-SPPEXA, 405-452, 2020

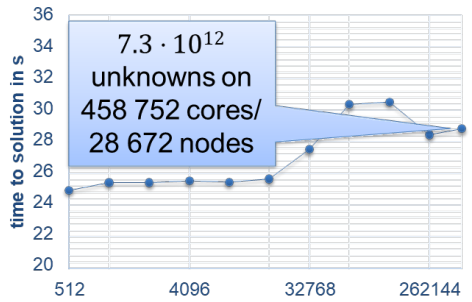
Example: Non-Newtonian and non-isothermal fluids

$$\frac{\partial \boldsymbol{u}}{\partial t} + (\boldsymbol{u} \cdot \nabla)\,\boldsymbol{u} - \nu \nabla^2 \boldsymbol{u} + \frac{1}{\rho}\nabla p = \boldsymbol{f}$$
$$\nabla \cdot \boldsymbol{u} = 0$$

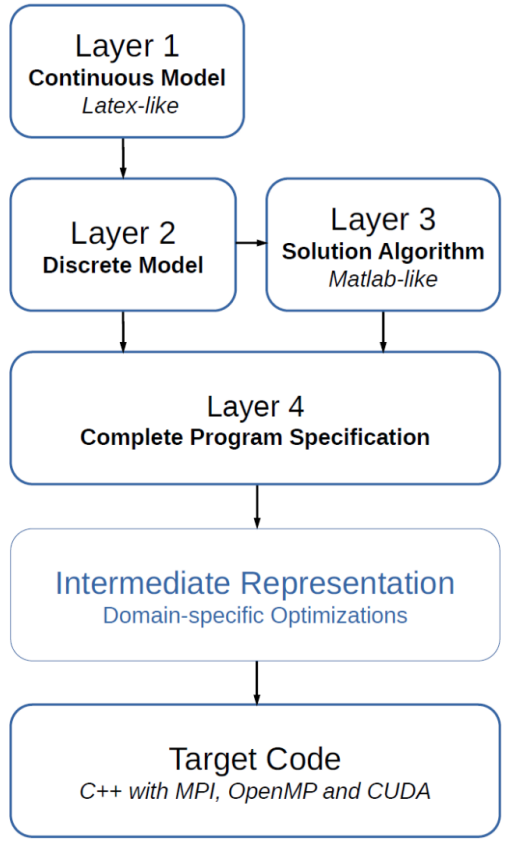
- Finite volumes
- Non-uniform staggered grids
- SIMPLE algorithm







Original workflow:



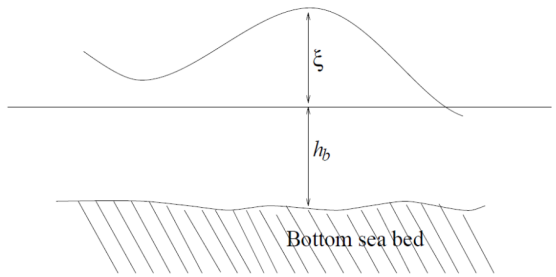
Code Generation for Shallow Water Equations

Block-structured grids and symbolic algebra

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

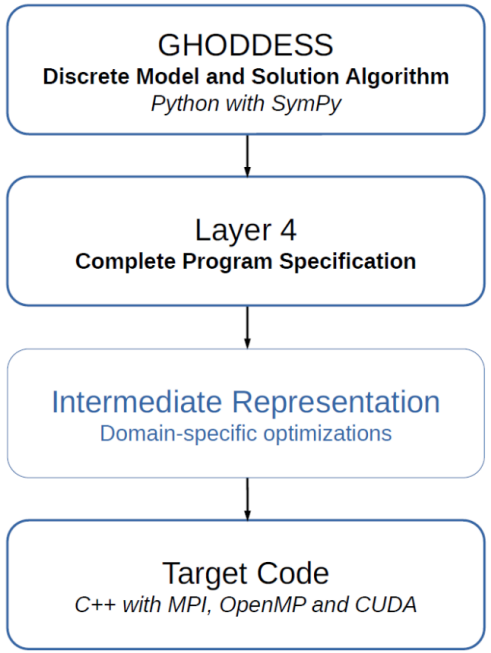
- Optimizations: buffering of geometric information, projection, algorithmic variations
- 2D shallow water equations using discontinuous Galerkin

$$\frac{\partial \xi}{\partial t} + \nabla \cdot \boldsymbol{U} = 0$$
$$\frac{\partial \boldsymbol{U}}{\partial t} + \nabla \cdot (\boldsymbol{U}\boldsymbol{U}^T/H) + \tau_{bf}\boldsymbol{U} + f_c \boldsymbol{k} \times \boldsymbol{U} + gH\nabla\xi = \boldsymbol{F}$$



Definition of elevation and bathymetry

Adapted workflow:



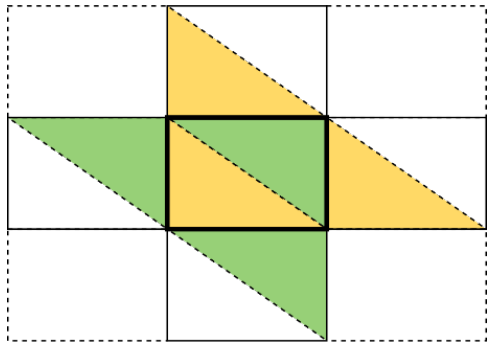
S. Faghih-Naini et al. *Quadrature-free discontinuous Galerkin method with code generation features for shallow water equations on automatically generated block-structured meshes*. Advances in Water Resources 138, 2020

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

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

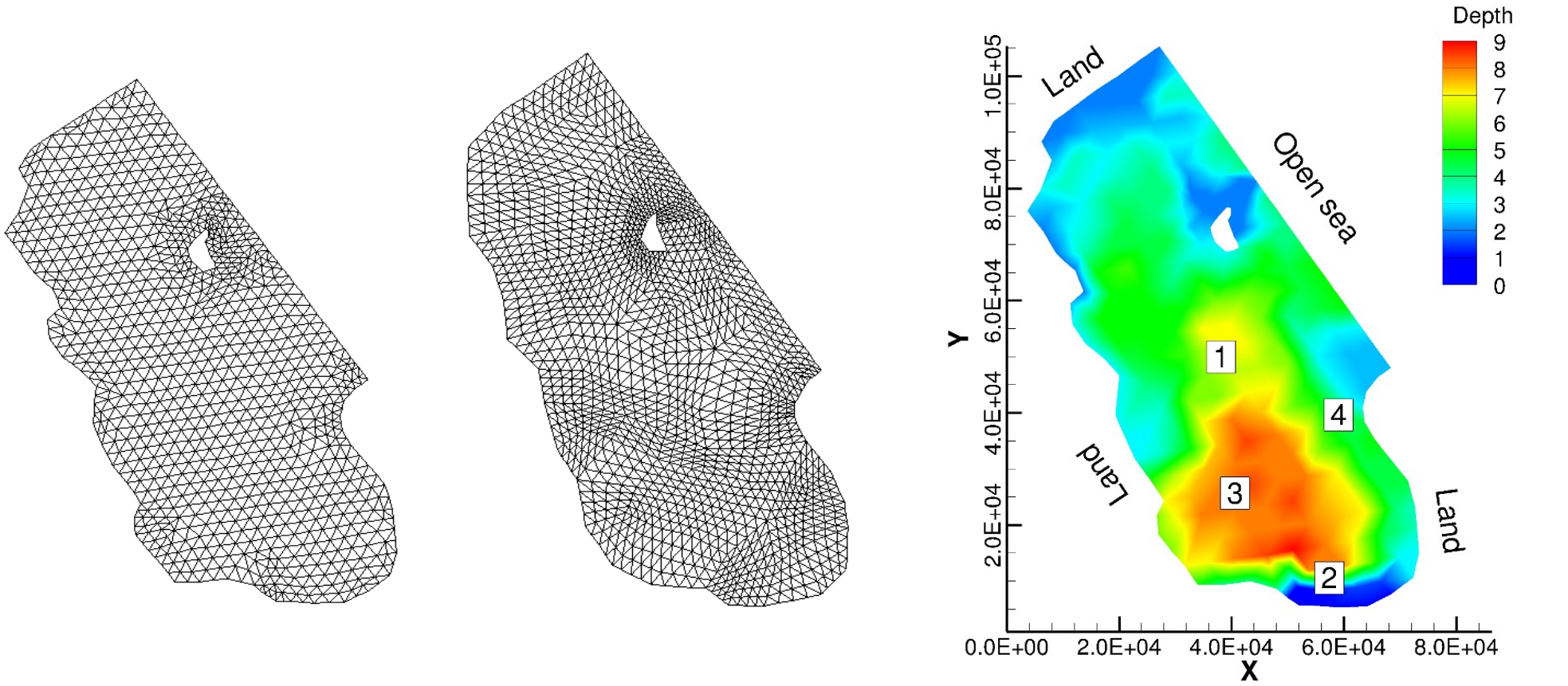


is translated to

```
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**

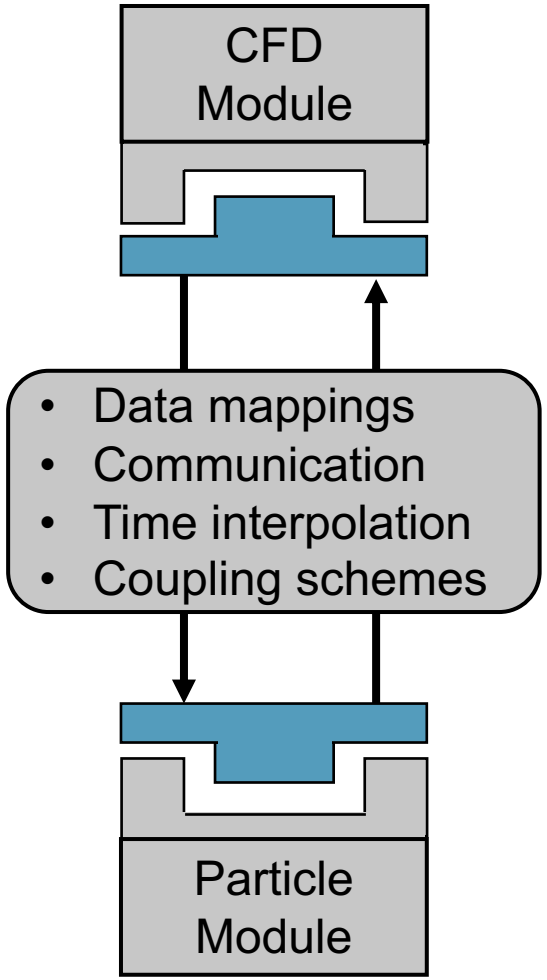
Tidal flow at Bahamas



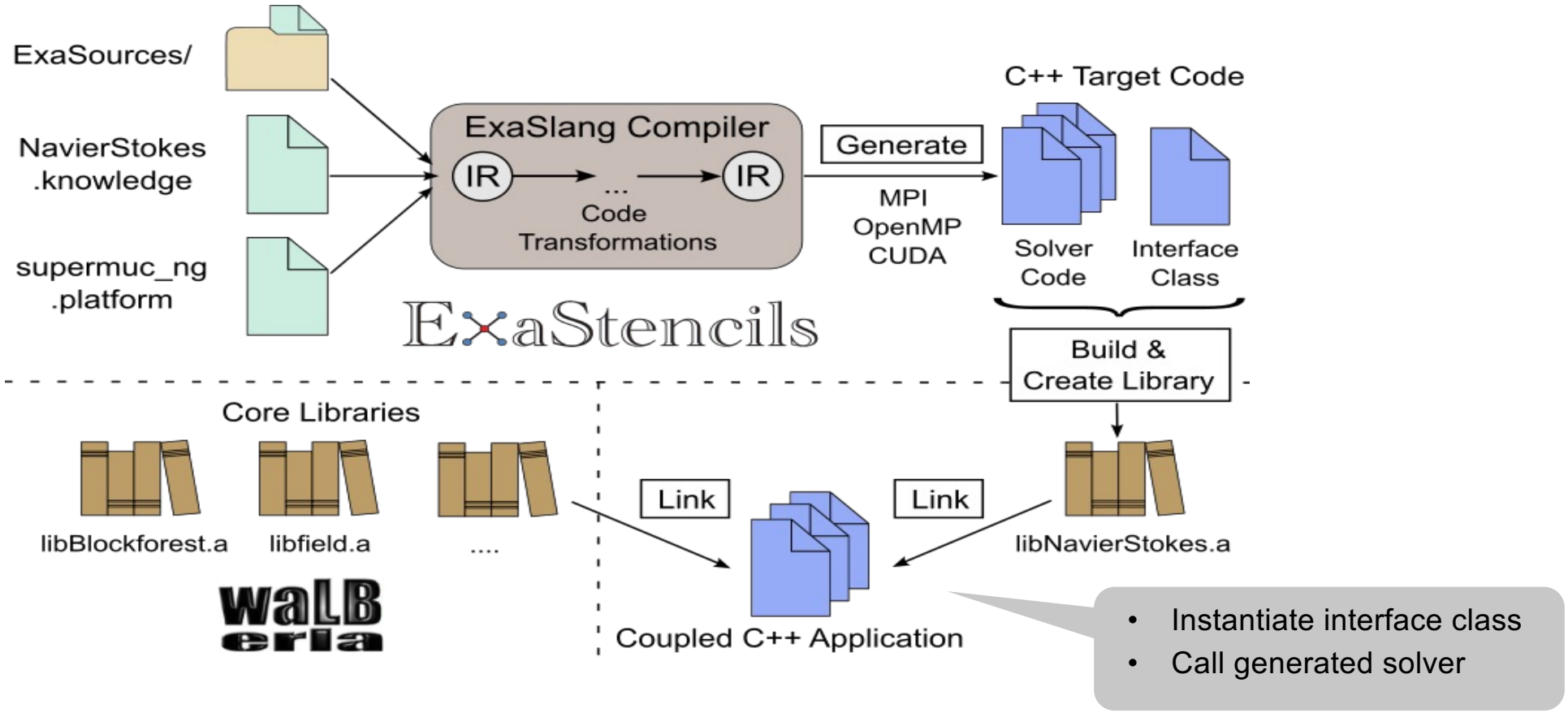
Original UTBEST unstructured mesh (left), automatically generated block-structured mesh used by the GHODDESS framework (middle), bathymetry and positions of recording stations (right).

Software Couplings

- **Hand-crafted** coupling codes
  - Often done in ad-hoc fashion
  - Tightly coupled, not very flexible
- Coupling **Libraries**, e.g. preCICE, MUSCLE
  - Minimal invasive, high-level APIs
  - Peer-to-peer, client/server designs
- **Generating** coupling codes
  - High-level description on DSL
  - On-the-fly conversions and data mappings
  - Performance-portable target code



Coupling – whole-solver code generation



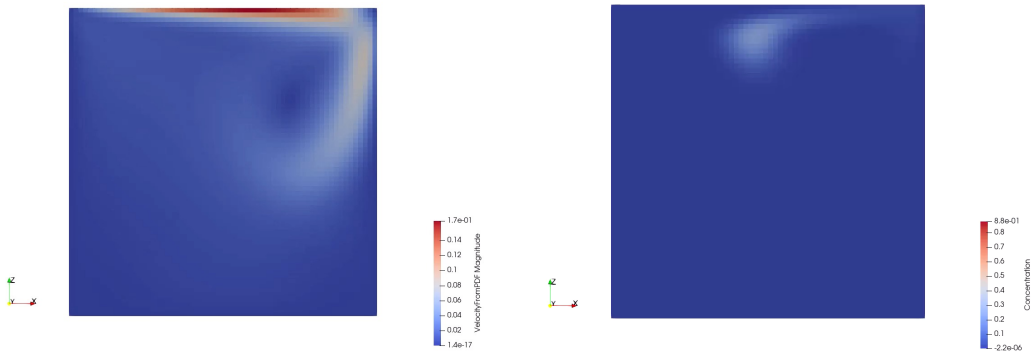
Application Scenarios – Coupled 3D Transport

- Transport of concentrations in fluid
- Advection-diffusion-reaction equation

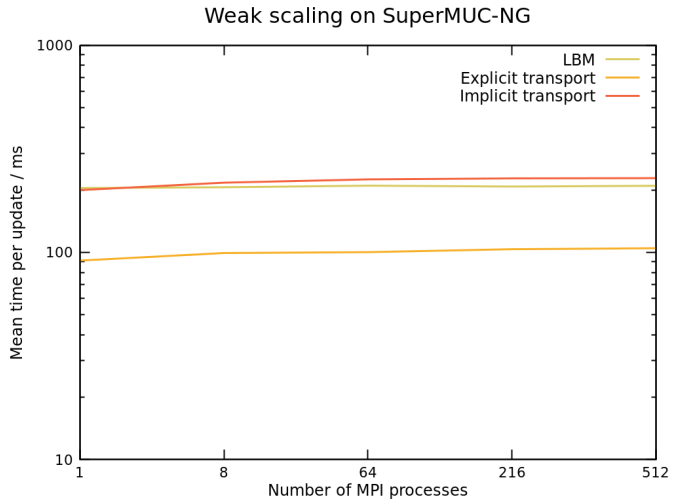
$$\frac{\delta c}{\delta t} = Q + D\nabla^2 c - \mathbf{u} \cdot \nabla c$$

- Euler-Lagrange Coupling
- Two (FVM) variants: **explicit** and **implicit**

```
loop over c {
  c<next> = c + ( dt / vf_cellVolume ) *
          ( Q
          - D * Laplace * c
          - ( integrateOverEastFace ( c * u ) +
              integrateOverWestFace ( c * u ) + ...
            )
          )
}
```



Re ≈ 800

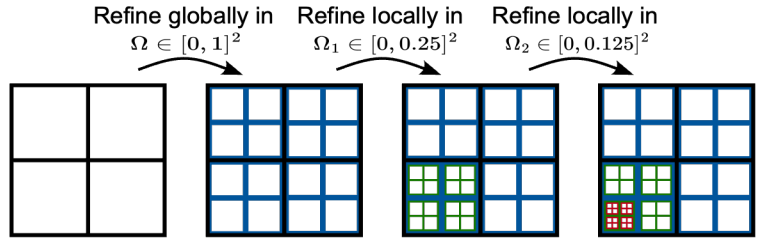


Multigrid With Local Refinement

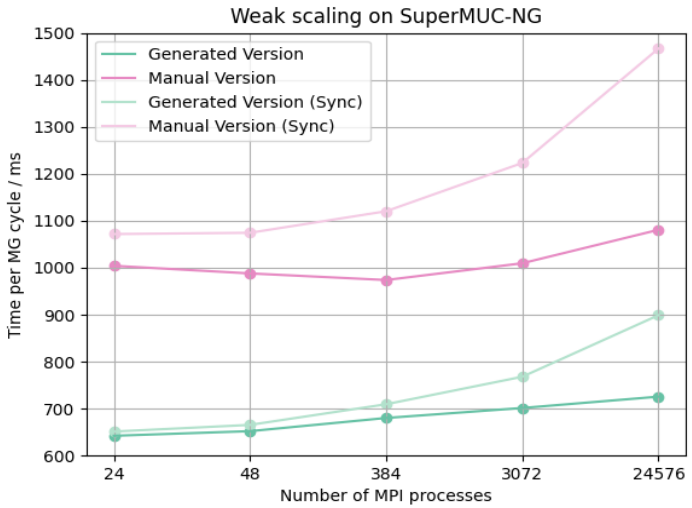
ExaStencils

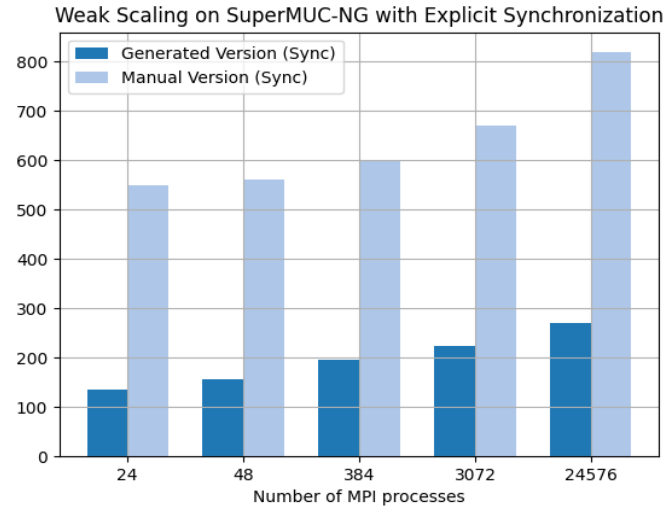
LSS FAU

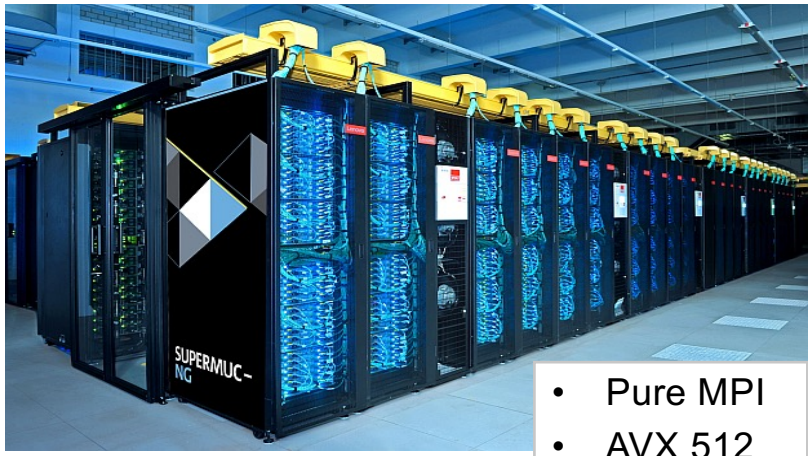
- Software Coupling with waLBerla provides automatic
  - Mesh refinement support for waLBerla‘s octree
  - Operator re-discretization
  - Generation of fused communication and interpolation
- **Comparison:** manual vs generated communication




```
Stencil GradientX {
  [0, 0, 0] => -1.0 / vf_cellWidth_x,
  [1, 0, 0] =>  1.0 / vf_cellWidth_x
}
```







- Pure MPI
- AVX 512

R. Angersbach et al. Code generation for octree-based multigrid solvers with fused higher-order interpolation and communication. *European Conf. on Parallel Processing*. Springer, 2024.

Operator generation - HyTeG

Code Generation and Performance Engineering for Matrix-Free Finite Element Methods on Hybrid Tetrahedral Grids

Fabian Böhm, Daniel Bauer, Nils Kohl, Christie Alappat,
Dominik Thönnes, Marcus Mohr, Harald Köstler, Ulrich Rüde

Accepted for publication in
SIAM Scientific Computing
https://arxiv.org/abs/2404.08371

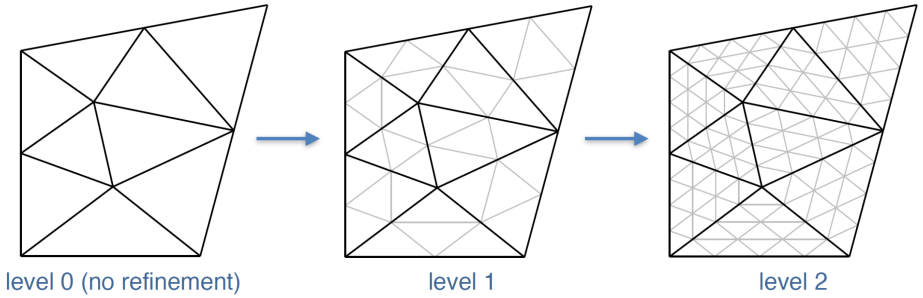
F. Böhm, F. et al (2025). Code Generation and Performance Engineering for Matrix-Free Finite Element Methods on Hybrid Tetrahedral Grids. *SIAM Journal on Scientific Computing*, *47*(1), B131-B159.

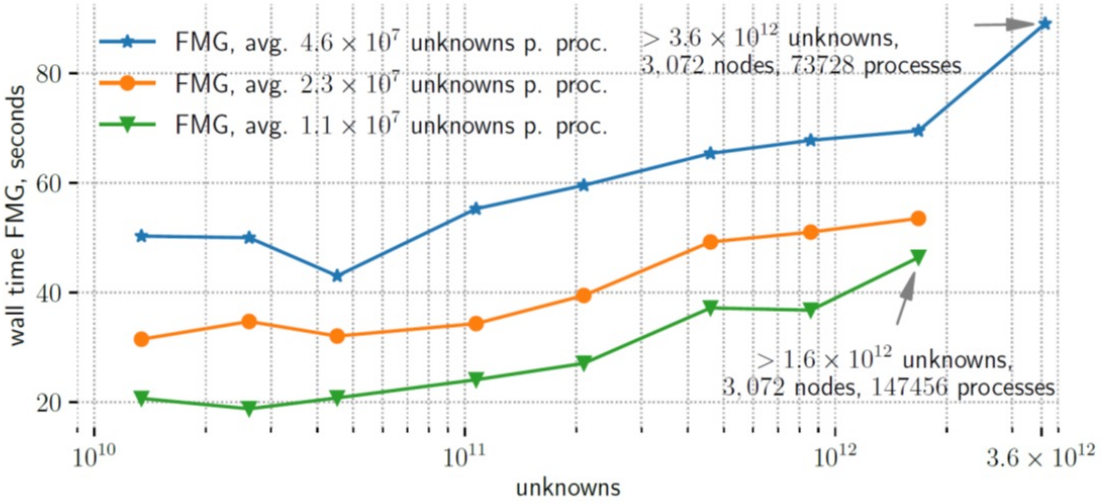
HyTeG

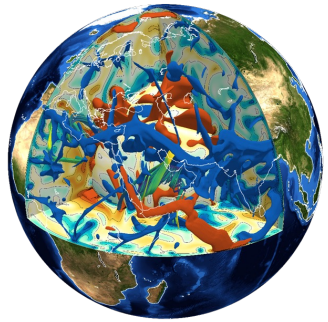
https://terraneo.fau.de/

Hybrid Tetrahedral Grids (HyTeG)

- Uniform refinement of unstructured tetrahedral grids
- “Automatic” multigrid hierarchy
- matrix-free, stencil-based kernels for finite element discretizations
- Open-source C++ software, MPI parallelization for CPU clusters







Earth mantle circulation

N. Kohl, U. Rüde. Textbook efficiency: massively parallel matrix-free multigrid for the Stokes system. SIAM Journal on Scientific Computing 44.2 (2022): C124-C155.

Large-Scale Finite Elements: Whole-Planet Mantle Convection

Governing Strong Equation: Stokes

$$\frac{\delta \boldsymbol{u}}{\delta t} - \nabla \cdot (\mu(\boldsymbol{x}, t, \boldsymbol{u})\epsilon(\boldsymbol{u})) + \nabla p = f(\boldsymbol{x}), \text{in } \Omega$$

$$\nabla \cdot \boldsymbol{u} = 0$$

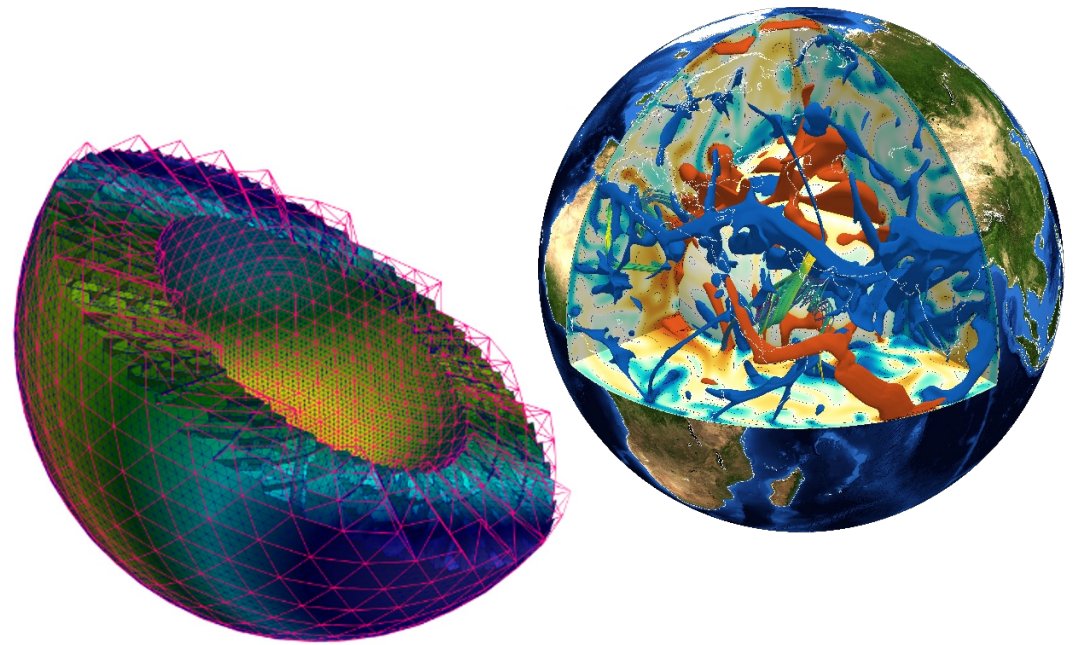
Weak Formulation

$$\text{Find } v \in V, \text{ such that}$$

$$\underbrace{\int_\Omega G(\boldsymbol{x}, v, w) d\boldsymbol{x}}_{=:a(v,w)} = b(w), \quad \forall w \in W$$

Linear System

$$A\boldsymbol{U} = \boldsymbol{F}$$



Domain: Planet Earth

- Spatial resolution: 1 km, 10^{12} DoFs
- Storage for global matrix: $7 \cdot 10^{12} \cdot 8\text{B} = 56\ \text{TB}$[1]

[1] Gmeiner et al., “A quantitative performance study for Stokes solvers at the extreme scale,”, 2016

Matrix-free finite element method

Find $v \in V$, such that

$$\underbrace{\int_\Omega G(\boldsymbol{x}, v, w)\,\mathrm{d}\boldsymbol{x}}_{a(v,w)} = b(w), \quad \forall w \in W\,. \tag{1}$$

FE discretization (finite basis with compact support)

$$\Omega = \bigcup_{T\in\mathcal{T}} T, \quad v = \sum_{j\in\mathcal{J}} v_j\phi_j, \quad w = \sum_{i\in\mathcal{I}} w_i\psi_i \tag{2}$$

$$A\boldsymbol{u} = \boldsymbol{f}, \quad \text{with } a_{ij} = a(\phi_j, \psi_i), f_i = b(\psi_i)\,. \tag{3}$$

Traditional approach: assemble and solve

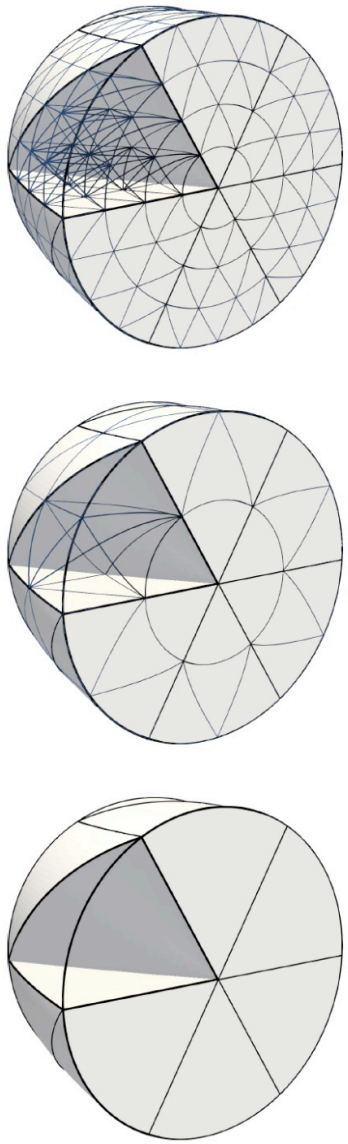
$$\begin{aligned}(A\boldsymbol{u})_i &= \boldsymbol{a}_i^T \cdot \boldsymbol{u} \\ &= \sum_{j\in\mathcal{J}} \underbrace{\sum_{T\in\mathcal{T}} \int_T G(\boldsymbol{x}, \phi_j, \psi_i)\,\mathrm{d}\boldsymbol{x}}_{a_{ij}}\, u_j\end{aligned}$$

Matrix-free: assemble on-the-fly

$$\begin{aligned}(A\boldsymbol{u})_i &= (\sum_{T\in\mathcal{T}} P_T A_T R_T \boldsymbol{u})_i \\ &= \sum_{T\in\mathcal{T}} \sum_{j\in\mathcal{J}_T} \underbrace{\int_T G(\boldsymbol{x}, \phi_j, \psi_i)\,\mathrm{d}\boldsymbol{x}}_{a_{ij}^T}\, u_j\end{aligned}$$

Requires information about the mesh at solve time.

Hierarchical Hybrid Grids



- Only coarsest mesh is stored in memory
- Flexible topology
- Indirection-free consecutive memory accesses
- Multigrid hierarchy by design
- Assign blocks to processes / no intra-macro parallelism

Why Code Generation?

Weak forms: $a(\boldsymbol{v}, \boldsymbol{w}) = \ldots$

- $\int_\Omega k(\boldsymbol{x}) \nabla v \cdot \nabla w$
- $\int_\Omega (\alpha(\boldsymbol{x}) \nabla \times \boldsymbol{v} \cdot \nabla \times \boldsymbol{w} + \beta(\boldsymbol{x}) \boldsymbol{v} \cdot \boldsymbol{w})$

...

Discretization Spaces

- Lagrangian $\mathcal{P}_k$
- Nedelec $\mathcal{ND}_1$
- Discontinuous Lagrangian $\mathcal{P}_{-k}$
- Enriched Galerkin $\mathcal{EG}_0$

Precisions

Half, Single, Double

Optimizations

- Cache-locality improving
- Redundant computation elimination
- Speed-up computations

Architectures

CPU vectorization registers

Aim of the present work:

Achieve **widely applicable, high node-level performance**

+ scalability by means of **code generation**

Element loop

▷ loop over all elements $T \in \mathcal{T}$

foreach *elementType* **do**
  for $z = 0, \ldots, n_z$ **do**
    for $y = 0, \ldots, n_y(z)$ **do**
      for $x = 0, \ldots, n_x(z, y)$ **do**

        ▷ compute A_T (using quadrature)
        $a_{ij} \leftarrow 0$
        for $q = 0, \ldots, n_q$ **do** ▷ quadrature rule
          $J \leftarrow$ computeJacobian(elementType)
          $a_{ij} \leftarrow a_{ij} + w_q \cdot (\ldots)$ ▷ bilinear form
        end

        $\mathrm{dst}_T \leftarrow \mathrm{dst}_T + A_T \mathrm{src}_T$ ▷ kernel operation
      end
    end
  end
end

HyTeG (HOG) Operator Generator

Inputs to HOG

define performance properties

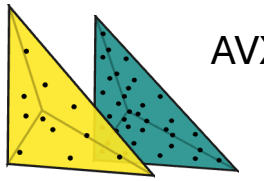
$$\int_\Omega G(\boldsymbol{x}, \boldsymbol{v}, \boldsymbol{w}) d\boldsymbol{x}$$ Weak form

$\mathcal{P}_2$ Function space

AVX Architecture

Quadrature

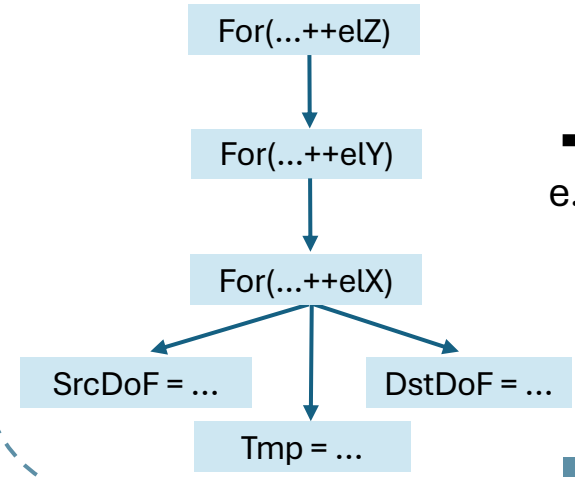
Precision



double, float, half

Abstract syntax tree

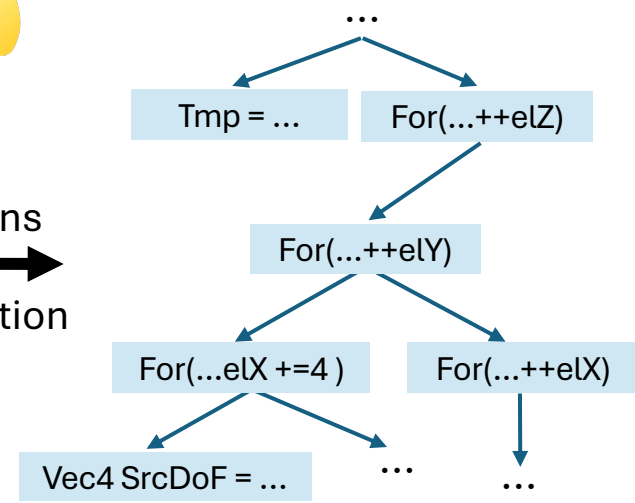
Provides option to transform the program



Tailored optimizations

e.g. vectorization

Transformed AST



Python Code Generator

Print sweep

Output: C++ code

Scalable, node-level optimized, matrix-free FE operator

C++ Framework

HYTEG

Hybrid Tetrahedral Grids

```
for (int64_t ctr_2 = 0; ctr_2 < mic
for (int64_t ctr_1 = 0; ctr_1 < -ct
for (int64_t ctr_0 = 0; ctr_0 < -ct
{
   const double tmp_WHITE_UP_0 = ma
   const double tmp_WHITE_UP_1 = po
   const double tmp_WHITE_UP_2 = tm
```

Optimizing Cache-Locality

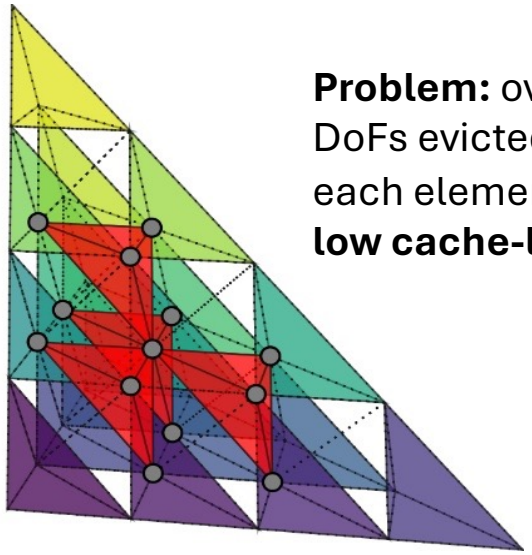
"Sawtooth" Loop Strategy (default)

6 Element loops, one for each orientation of tetrahedra

"Cubes" Loop Strategy (C)

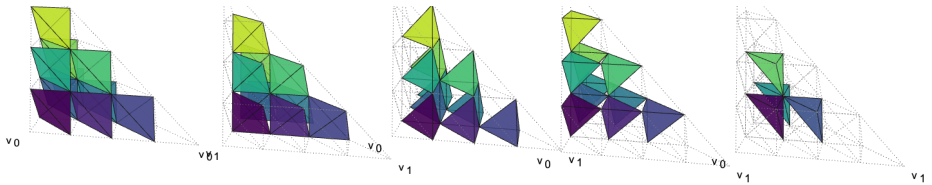
- Single loop, 1 cube of 6 tetrahedra per iteration
- Tetrahedra outside of macro must be skipped

I-UP

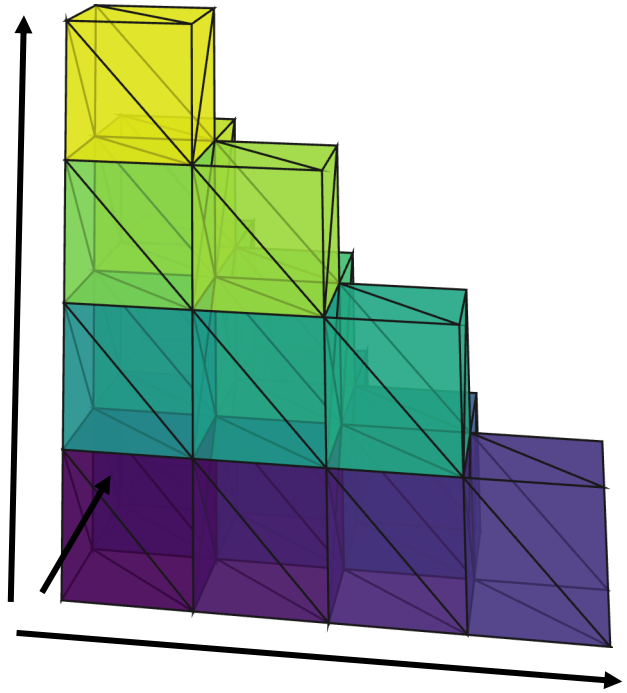


Problem: overlapping DoFs evicted between each element loops, **low cache-locality**

I-DOWN **II-UP** **II-DOWN** **III-UP** **III-DOWN**



Generate fused loops



Optimizing Cache-Locality

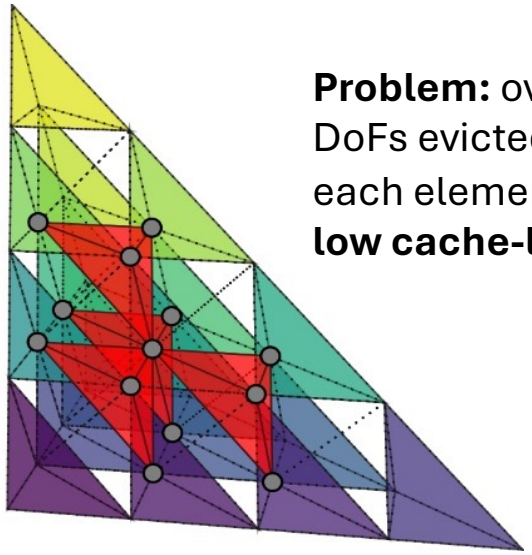
"Sawtooth" Loop Strategy (default)

6 Element loops, one for each orientation of tetrahedra

"Cubes" Loop Strategy (C)

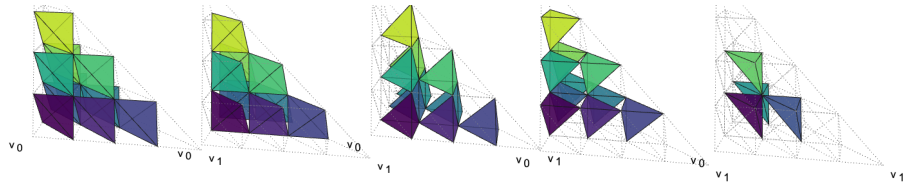
- Single loop, 1 cube of 6 tetrahedra per iteration
- Tetrahedra outside of macro must be skipped

I-UP



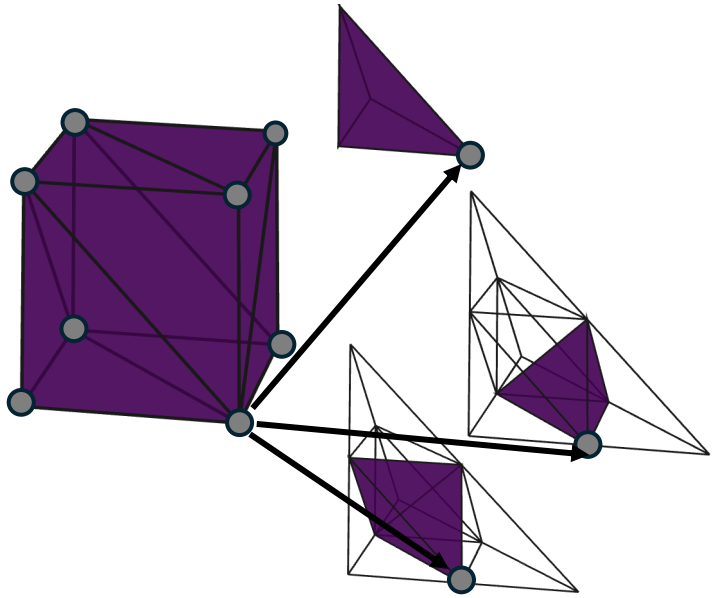
Problem: overlapping DoFs evicted between each element loops, **low cache-locality**

I-DOWN **II-UP** **II-DOWN** **III-UP** **III-DOWN**



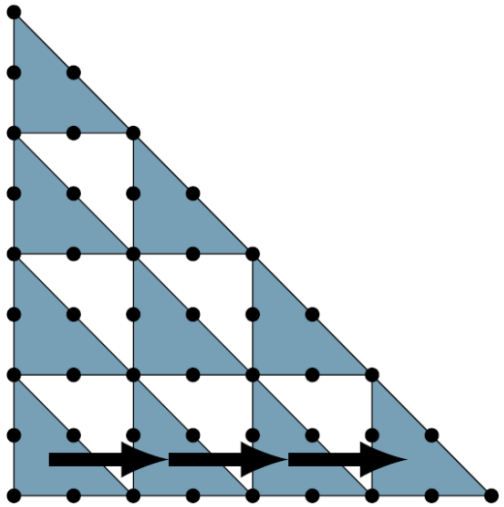
Generate fused loops

Accesses in single cube-iteration, fewer reloads from main memory, **high cache-locality**



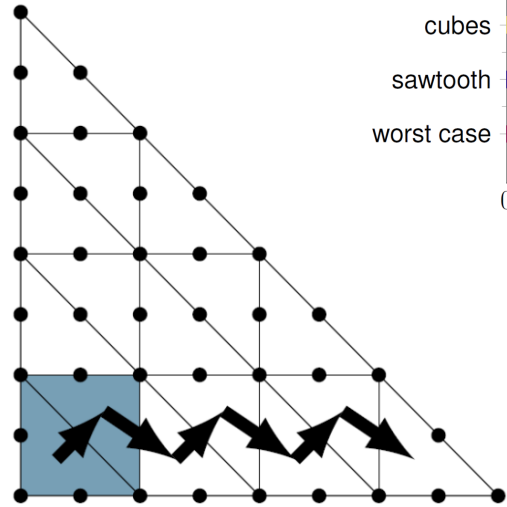
Optimizing Cache-Locality

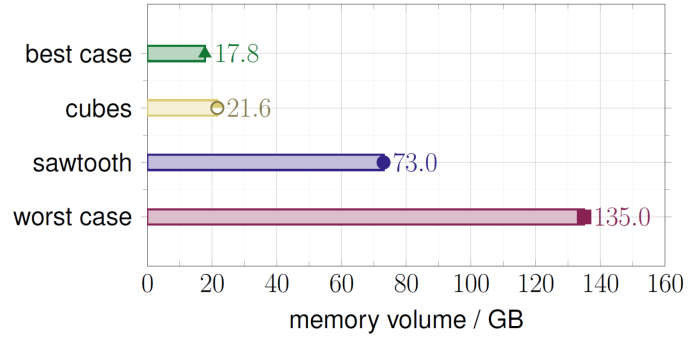
Sawtooth



```
1 foreach elementType do
2 |  for y = 0, ..., n_y do
3 |  |  for x = 0, ..., n_x(y) do
4 |  |  |  ⋮
5 |  |  end
6 |  end
7 end
```

Cubes






```
1 for y = 0, ..., n_y do
2 |  for x = 0, ..., n_x(y) do
3 |  |  foreach elementType do
4 |  |  |  ⋮
5 |  |  end
6 |  end
7 end
```

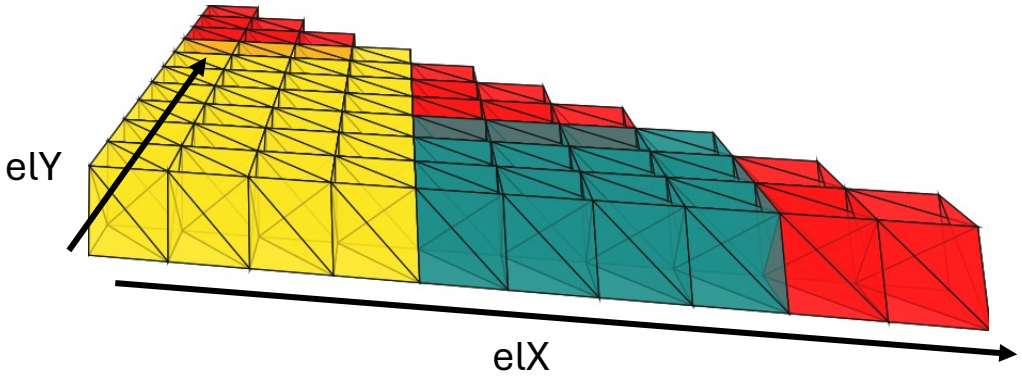
Optimizing Computations: Inter-Element Vectorization (V)

Parallel Assembly

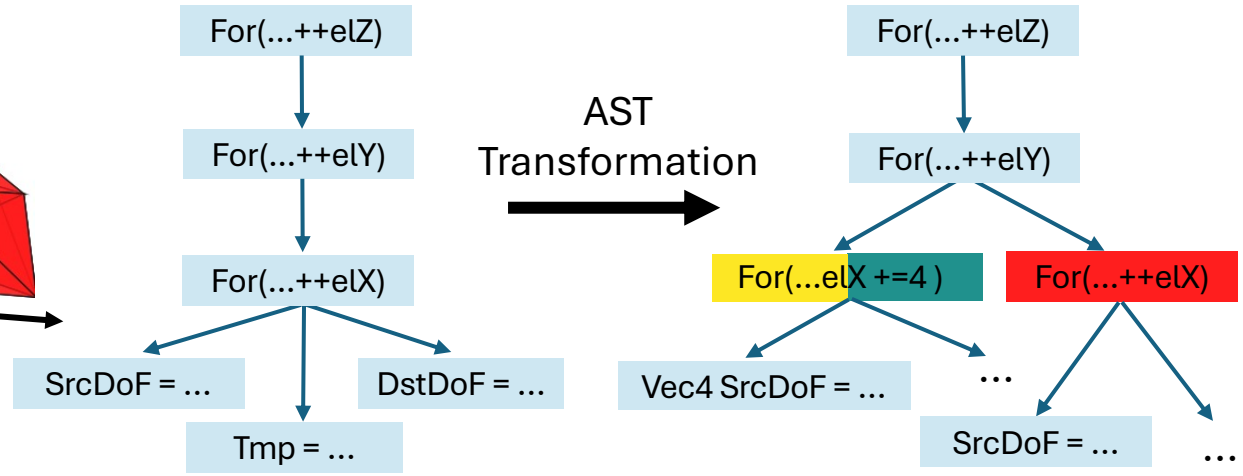
- Assemble on 4 (AVX2) or 8 (AVX512) elements/cubes simultaneously (yellow and teal with AVX2)
- Remainder loop (red)

AST Transformation

- Split x-loop during generation time into vectorized and remainder loop
- AST-nodes in vectorized loop replaced by Intel SIMD intrinsics



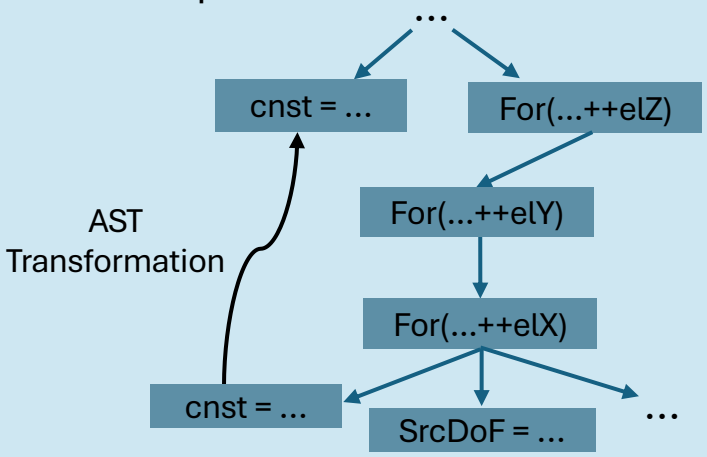
Vectorized cubes loop



Optimizing Computations: Redundancy Reduction

Loop Invariants (I)

- Traverse AST, check expressions for **dependency on loop counters**
- If no dependency is found, **move expression ahead** of the loop



Tabulation (T)

- Factors independent of the current micro tetrahedron **are precomputed and stored in tables**
- For differential operator $-\nabla \cdot (k(\boldsymbol{x})\nabla)$:

$$\int_T k(\boldsymbol{x})\nabla\phi_j(\boldsymbol{x}) \cdot \nabla\psi_i(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x}$$

$$= \int_{\hat{T}} k(\hat{\boldsymbol{x}})|\det J|J^{-T}\nabla\hat{\phi}_j(\hat{\boldsymbol{x}}) \cdot J^{-T}\nabla\hat{\psi}_i(\hat{\boldsymbol{x}})\,\mathrm{d}\hat{\boldsymbol{x}}$$

$$= \sum_q w_q k(\hat{\boldsymbol{x}}_q) \underbrace{|\det J|J^{-T}\nabla\hat{\phi}_j(\hat{\boldsymbol{x}}_q) \cdot J^{-T}\nabla\hat{\psi}_i(\hat{\boldsymbol{x}}_q)}_{\text{spatially constant}}$$

$$= \sum_q w_q k(\hat{\boldsymbol{x}}_q) \cdot \mathrm{table}(i, j, q, \mathrm{elementType})$$

Performance Analysis

Measurements

- Fritz Supercomputer at NHR@FAU
- Matrix-vector multiplication (without communication)
- Single socket: Intel Xeon Platinum 8360Y (”Ice Lake”)
- 36 cores per socket
- LIKWID performance monitoring and benchmarking suite

Approach:

Generated optimizations

- Symmetry (S)
- Inter-element vectorization (V)
- Loop invariants (I)
- Cubes loop strategy (C)
- Under-integration (U)
- Fused quadrature loops (fQ)
- Tabulation (T)

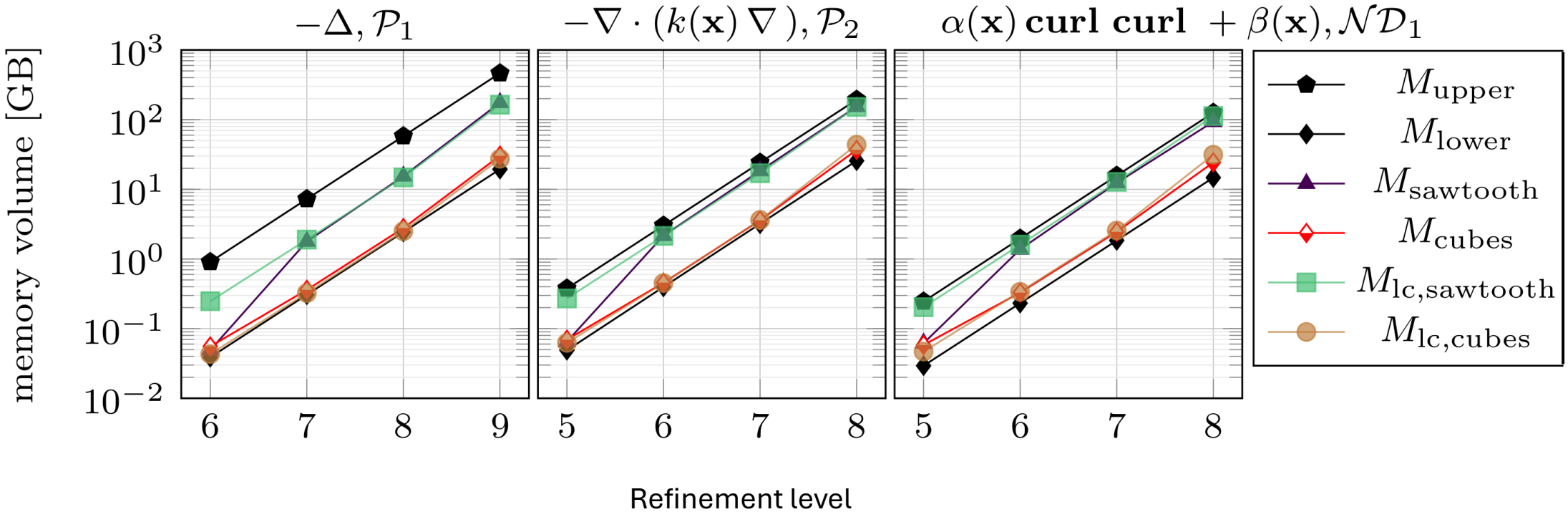
Optimization search

- Generate all combinations
- Determine the **set of most effective optimizations**

Optimization path

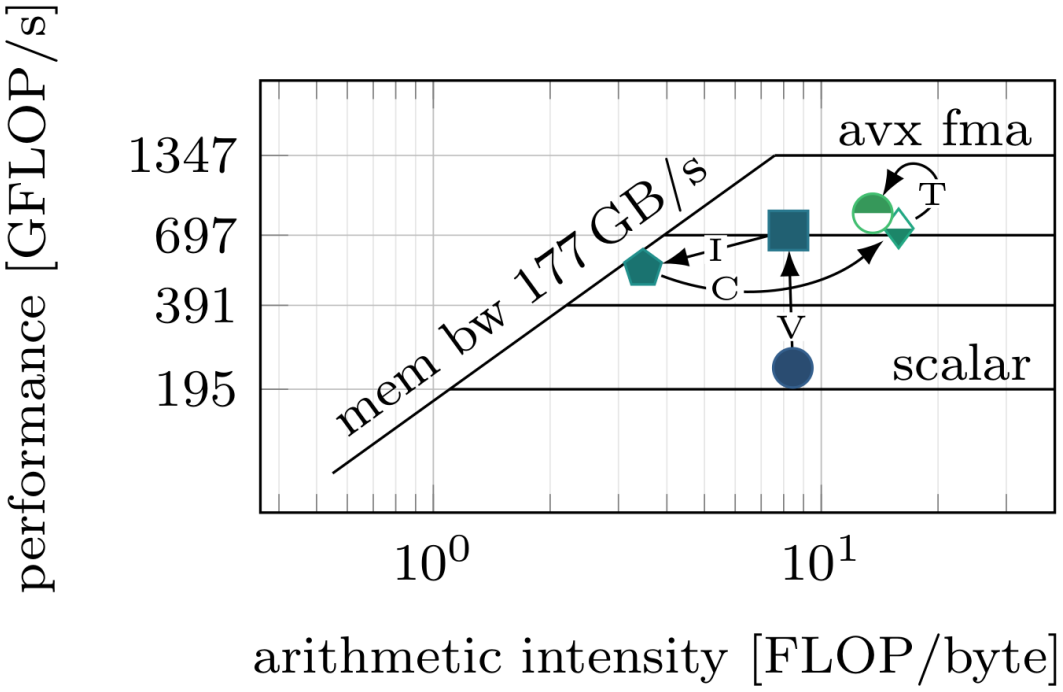
For the fastest operator: Roofline analysis of optimization path

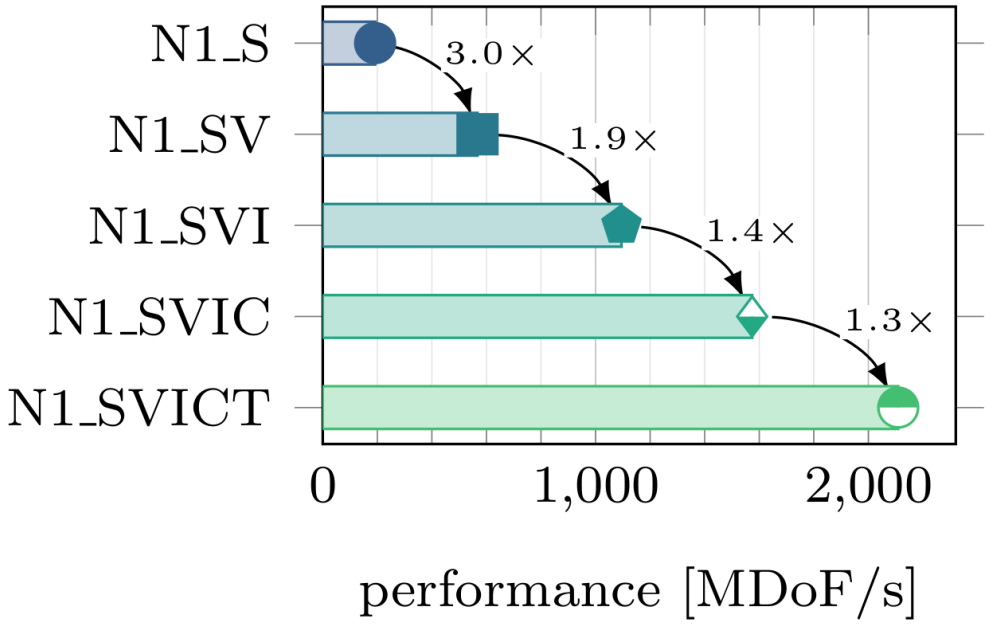
Optimizing Cache-Locality



Optimization Path: N1

Operator N1: $\int_\Omega (\alpha(\boldsymbol{x}) \nabla \times \boldsymbol{v} \cdot \nabla \times \boldsymbol{w} + \beta(\boldsymbol{x}) \boldsymbol{v} \cdot \boldsymbol{w}), \mathcal{ND}_1$

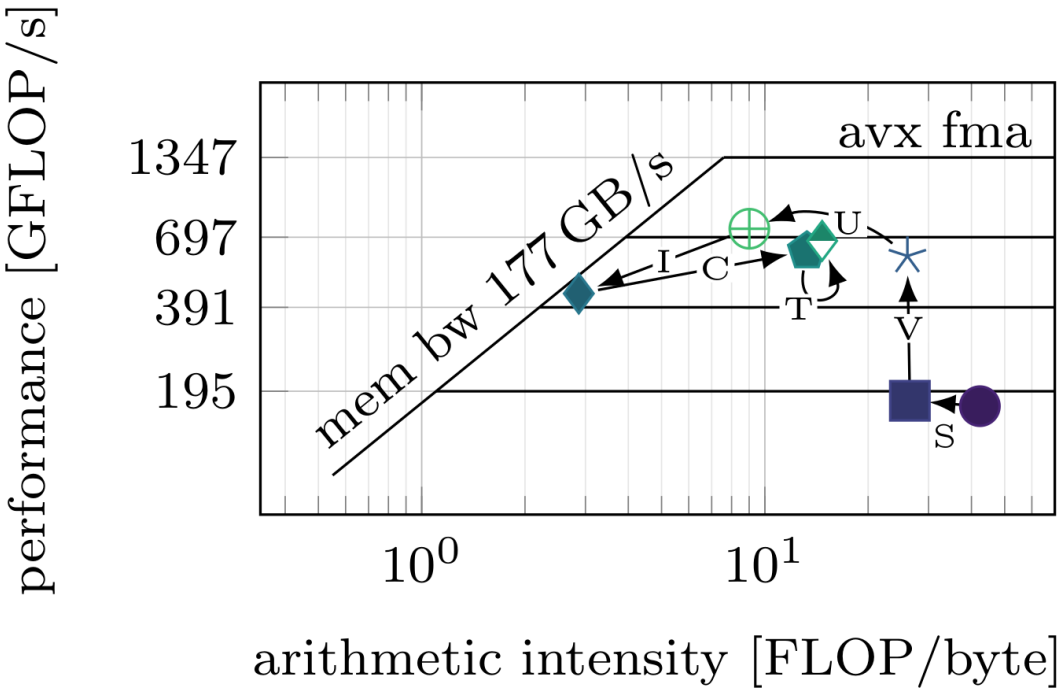


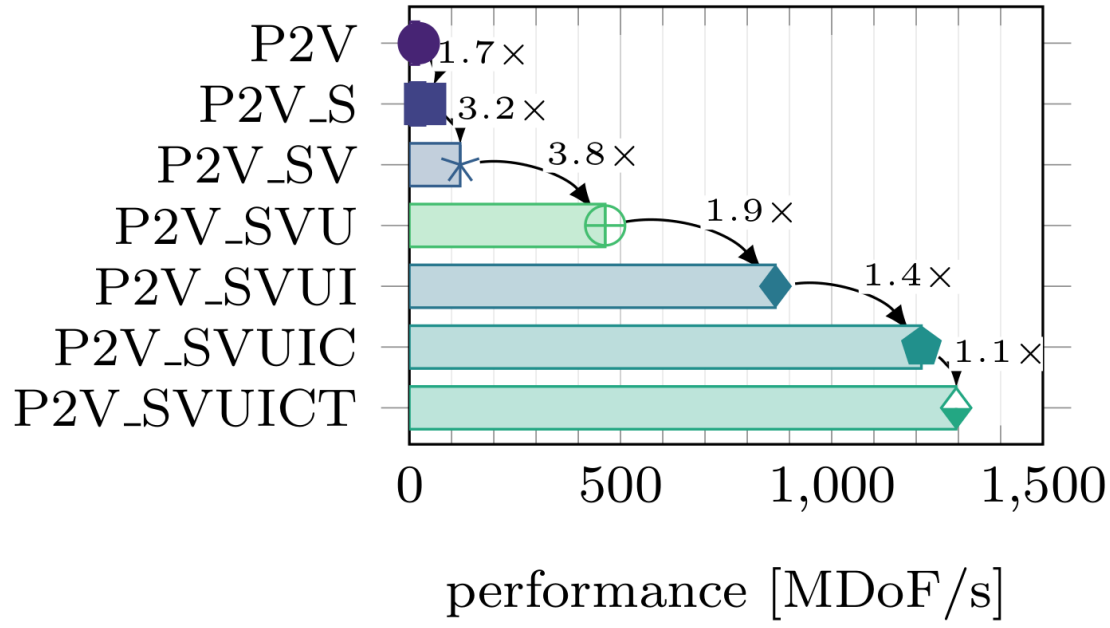


- Vectorization: steep **performance boost**
- Loop invariants: **arithmetic intensity decreases**
- Cubes loop: cache-locality improves**, arithmetic intensity increases**
- **11x accumulated speed-up, 62% peak, >2GDoF/s**

Optimization Path: P2V

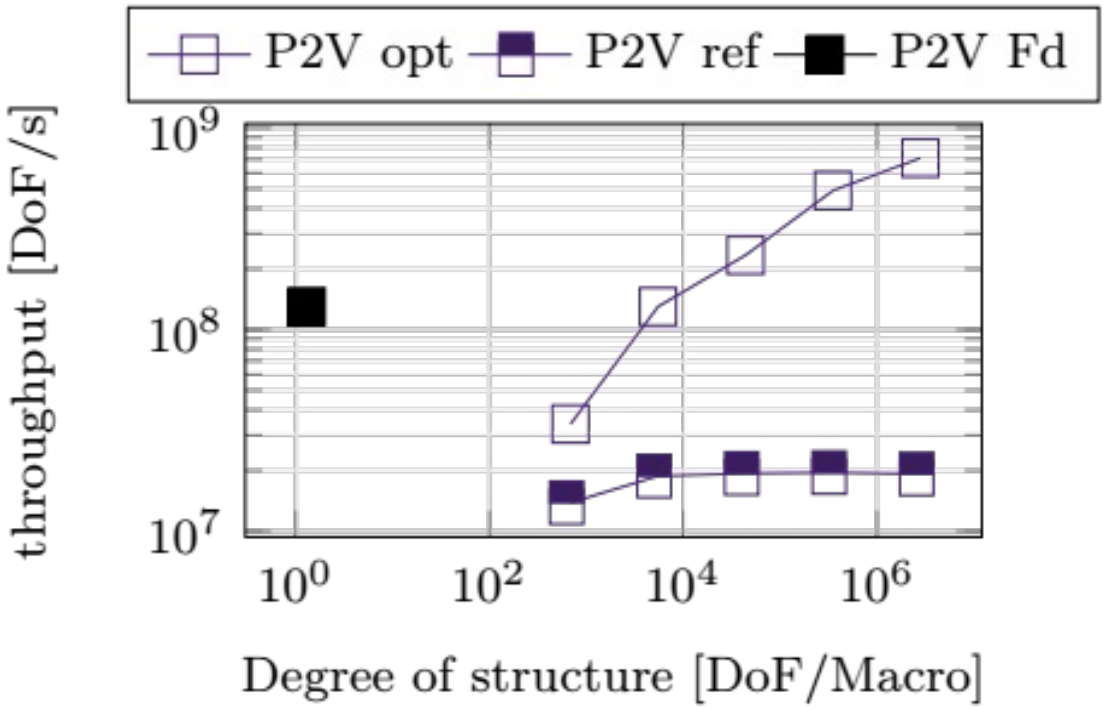
Operator P2V: $\int_\Omega k(\boldsymbol{x}) \nabla v \cdot \nabla w, \mathcal{P}_2$

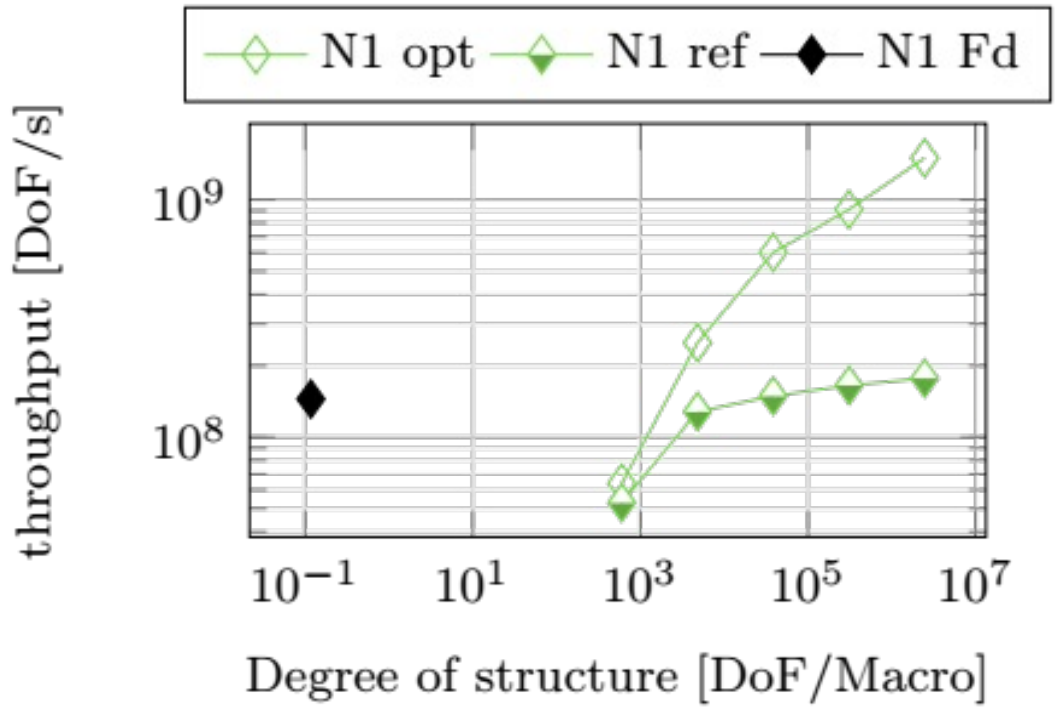




- Starting point: already compute-bound
- Series of opts reducing arithmetic intensity
- Compute-intense **P2V becomes memory-bound** with P2V_SVUI
- Cubes loop applicable -> more speed-up
- **58x accumulated speed-up, 50% peak, 1.4 GDoF/s**

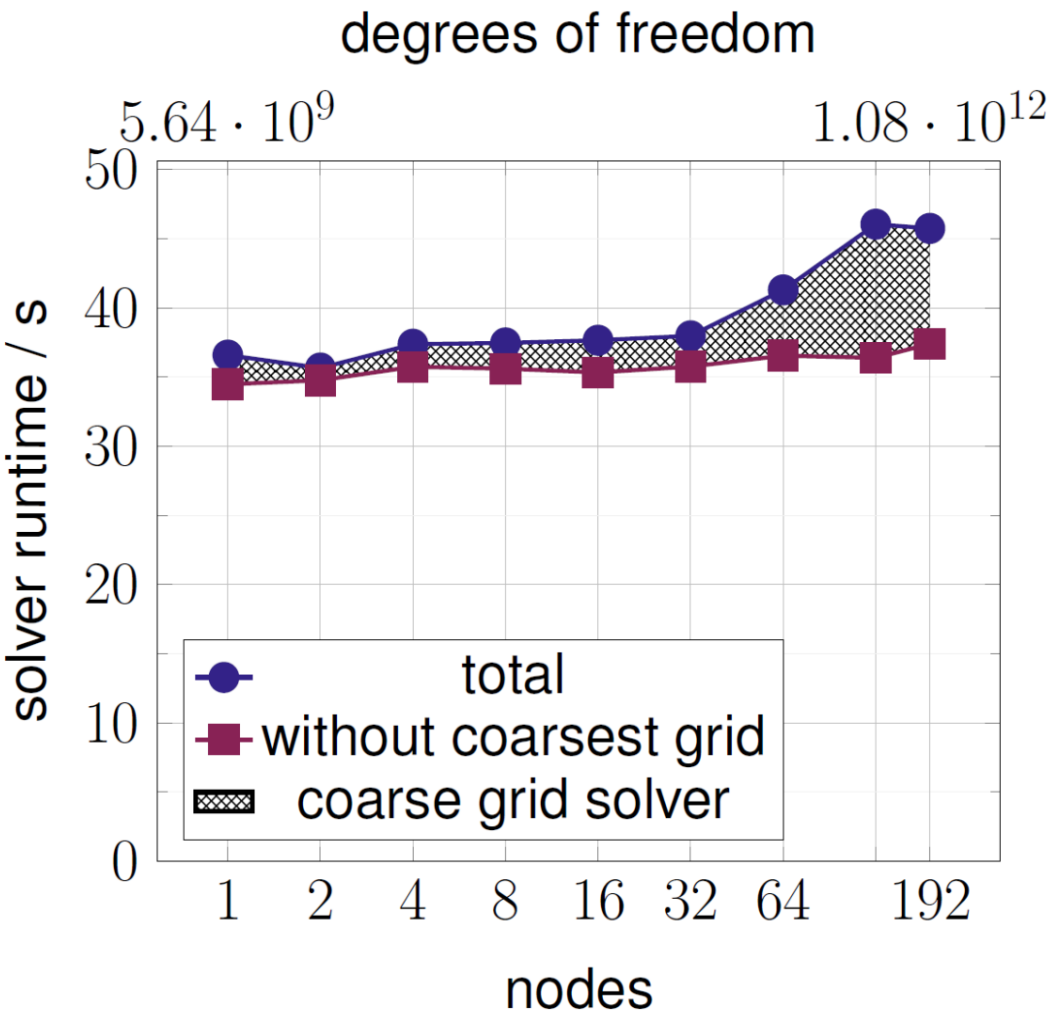
Comparison with Firedrake





- Ref: reference, Opt: generated+optimized, Fd: Firedrake
- Increasing refinement level 3-7, constant problem size
- **low refinement: communication dominates**
- **High refinement**: close to a **magnitude speed-up**

Weak scaling on SuperMUC-NG



- $\int_\Omega (\alpha(\boldsymbol{x})\,\mathbf{curl}\,\boldsymbol{v} \cdot \mathbf{curl}\,\boldsymbol{w} + \beta(\boldsymbol{x})\boldsymbol{v}\cdot\boldsymbol{w})$
- First order, first kind Nédélec $\mathcal{ND}_1$
- 8 refinement levels
- Full multigrid, 1 V(1,1)-cycle per level
- Hiptmair's hybrid smoother[7], Chebyshev order 2 in subspaces
- SuperMUC-NG Phase 2, 112 Cores, 512 GB DDR5 per node[8]

Overall solver performance

$1.1 \cdot 10^6 - 1.4 \cdot 10^6$ DoFs/(core·s)

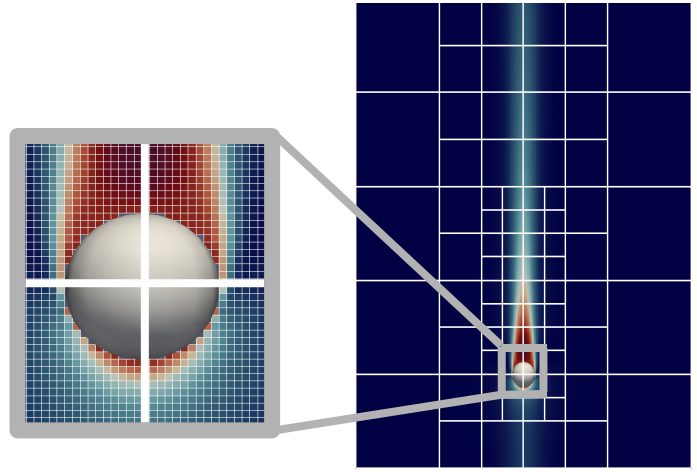
[7] Hiptmair, "Multigrid Method for Maxwell's Equations," 1998
[8] Leibniz-Rechenzentrum (LRZ), *Pilot Operation SuperMUC-NG Phase 2*

LBM and framework-specific code generation-waLBerla

waLBerla framework

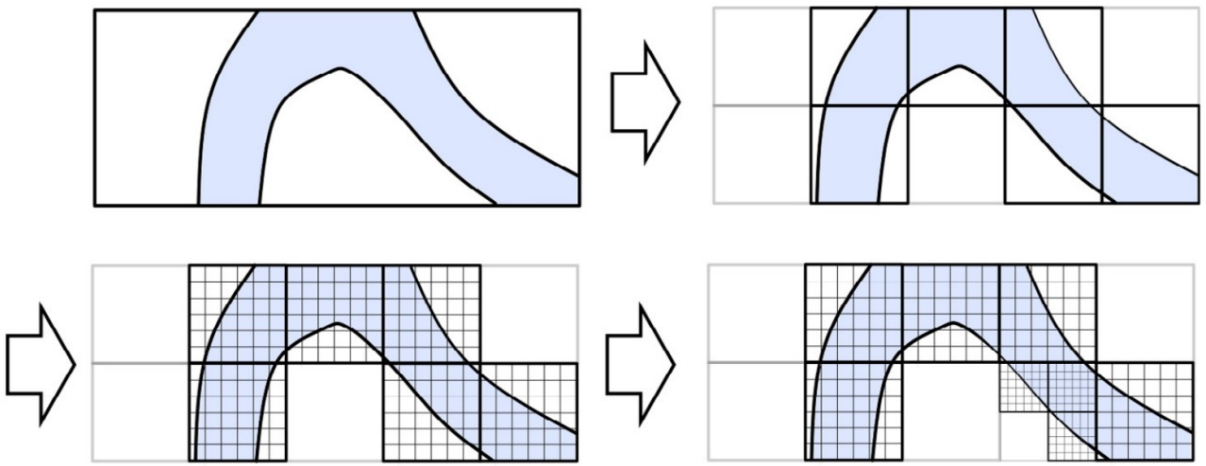
Open Source: https://walberla.net/

- Modular **C++20 multiphysics** framework
- CFD using the lattice Boltzmann method (**LBM**)
- Adaptive mesh refinement and load balancing
- Application scenarios e.g. rigid-particle dynamics, free-surface flows, solidification of material

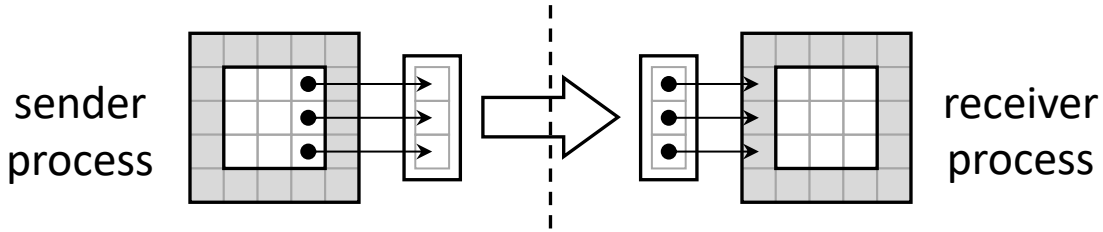


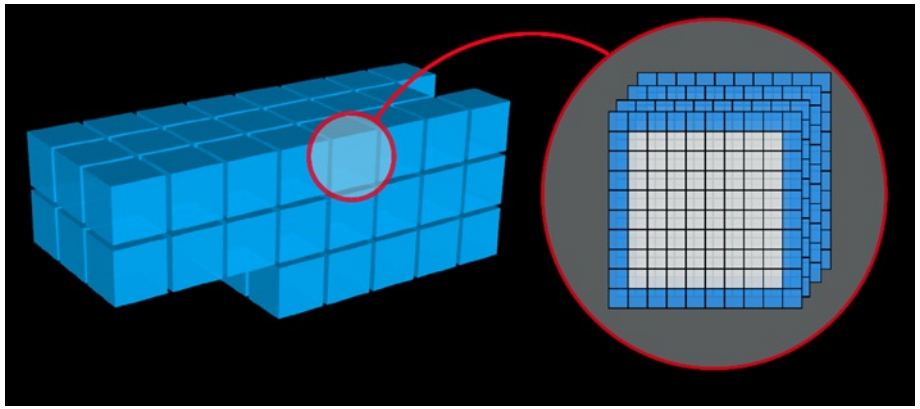
Block-structured domain partitioning

- The concept of block forests
  - Regularly partitioned subdomains act as root nodes for an octree
  - Leaf nodes of the refined forest of octrees make up final simulation subdomains (blocks)
  - The blocks can readily be distributed among available MPI processes



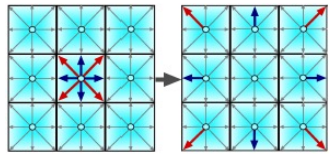
- Communication: distributed-memory systems
  - MPI-based halo exchange
  - support for asynchronous communication: overlapping communication and computation
  - GPU and CPU





M. Bauer, et al. *waLBerla: A block-structured high-performance framework for multiphysics simulations.* Computers & Mathematics with Applications, 81, 478-501, 2021.

Code Generation for Lattice Boltzmann Method



Python script

Field Assignments

pystencils

IR

C++ CUDA

waLBerla C++ Framework

Kernels

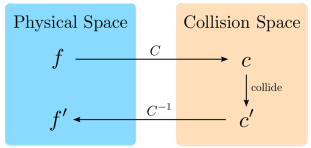
SymPy

LBM Update Rule

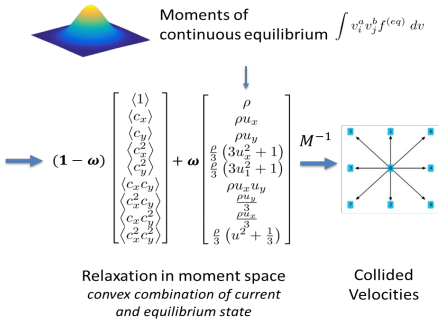
BCs

...

$$f_q(\mathbf{x} + \mathbf{c}_q \delta t, t + \delta t) = K\left(f_q(\mathbf{x}, t)\right)$$



$$c := C(f)$$
$$c^{(eq)} := C\left(f^{(eq)}\right)$$
$$c' := (\mathbb{1} - S)c + Sc^{(eq)}$$
$$S := diag(\omega_1, \omega_2, ..., \omega_q)$$

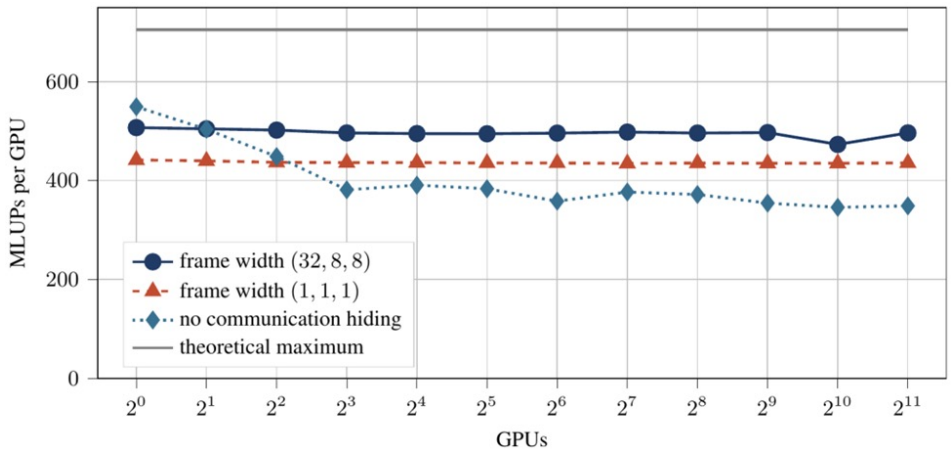


Discrete Velocities

Moment vector

Relaxation in moment space
convex combination of current and equilibrium state

Collided Velocities



Bauer, M., Köstler, H., Rüde, U. (2021). lbmpy : Automatic code generation for efficient parallel lattice Boltzmann methods. Journal of Computational Science, 49.

Code Generation Parameter Space

Models / Features

- Different Stencils (2D and 3D)
- Moment-based methods (MRT)
  - Efficient SRT and TRT implementations
  - Moment basis construction
  - Various equilibria
  - Forcing approaches
- Different collision space: cumulant method
- Entropic stabilization
- Locally varying relaxation rates e.g. to include turbulence models
- Coupling of multiple kernels

Hardware / Optimization

- GPU support
- Vectorization (AVX2, AVX512, QPX, SVE)
- Inner loop splitting to improve prefetching due to lower number of load/store streams
- Sparse (list-based) kernels for domains with many boundary cells
- Data layout: simple two grid stream-collide, AA pattern, EsoTwist

Kernel Generation

Purely symbolic description of a problem in SymPy

```
1  jacobi
```

$$d_{(0,0)} \leftarrow \frac{f_{(1,0)}}{4} + \frac{f_{(0,1)}}{4} + \frac{f_{(0,-1)}}{4} + \frac{f_{(-1,0)}}{4}$$

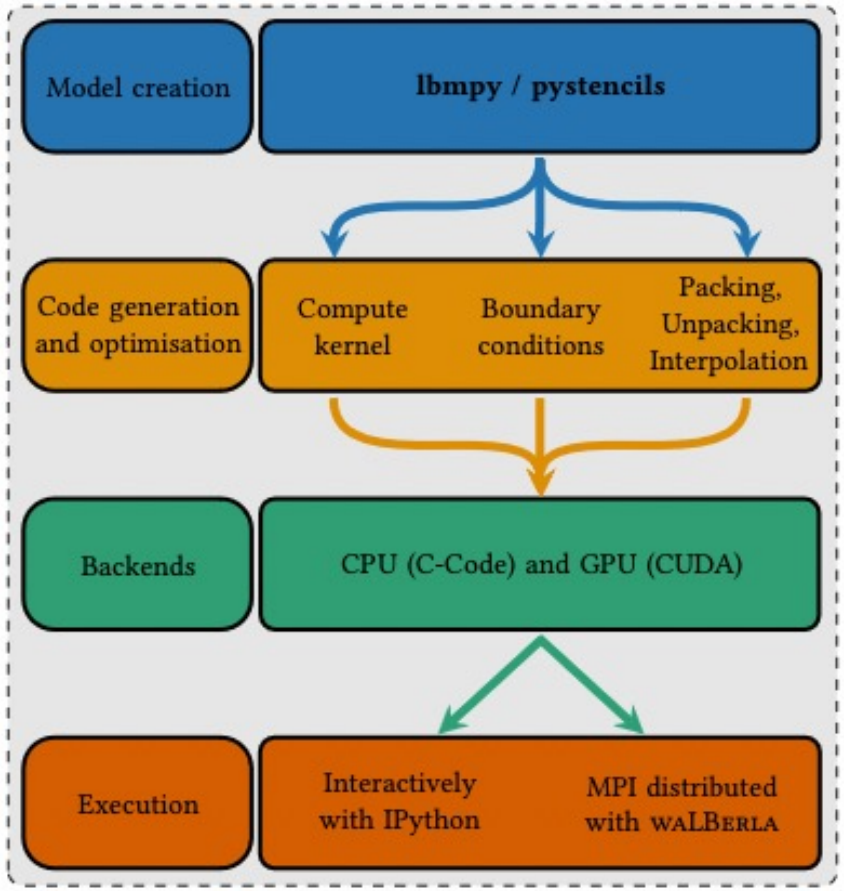
Automatic generation of compute kernel in programming language of lower level (C / CUDA)

```
1  ps.show_code(ast)
```

```
FUNC_PREFIX void kernel(double * RESTRICT  _data_d, double * RESTRICT const _data_f)
{
   for (int64_t ctr_1 = 1; ctr_1 < 11; ctr_1 += 1)
   {
      double * RESTRICT  _data_d_10 = _data_d + 12*ctr_1;
      double * RESTRICT _data_f_10 = _data_f + 12*ctr_1;
      double * RESTRICT _data_f_11 = _data_f + 12*ctr_1 + 12;
      double * RESTRICT _data_f_1m1 = _data_f + 12*ctr_1 - 12;
      for (int64_t ctr_0 = 1; ctr_0 < 11; ctr_0 += 1)
      {
         _data_d_10[ctr_0] = 0.25*_data_f_10[ctr_0 + 1] + 0.25*_data_f_10[ctr_0 - 1] + 0.25*_data_f_11[ctr_0] + 0.25*
_data_f_1m1[ctr_0];
      }
   }
}
```

1. Mathematical modeling in continuous description of a problem
   - Definition of complex problems in flexible manner
   - Automated parametrization of problem
   - Derivation of discretized problem (index/stencil notation)
2. Description of a problem in a list of symbolic Assignments in index notation
   - Mathematical rewriting to reduce FLOPs
3. Intermediate representation (AST)
   - Transformation in an algorithmic description
   - Introduction of loop nodes, type information, memory access …
4. Code Backends
   - Printing of the AST in a lower level programming language (C or CUDA)
   - Printing of intrinsics for the usage of SIMD on CPU

Code Generation for waLBerla



Generation of:

- Compute kernels for cell updates
- Boundary conditions
- Packing, Unpacking kernels to pack and unpack buffers for MPI communications

Strictly defined API of the printed kernels provides additional advantages like simple embedding in boiler plate codes to combine the generated compute kernels with existing HPC frameworks

Execution of the compute kernels in Python via C-API

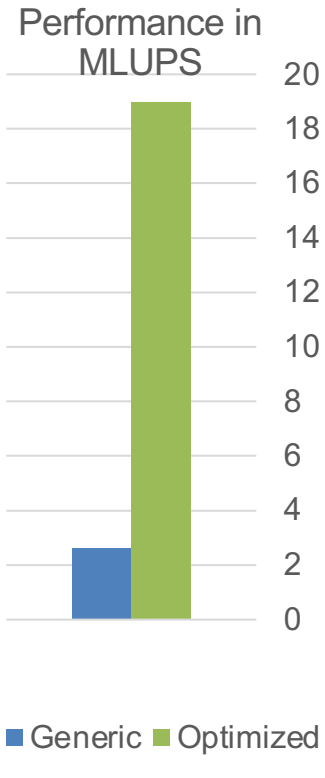
Code generation for LBM kernels

```
real_t velX = 0, velY = 0, velZ = 0, rho = 0;

for( auto d = LM::Stencil::begin(); d !=
LM::Stencil::end(); ++d )
{
  // stream pull
  const real_t pdf = src.neighbor( d.inverseDir(), d );
  dst.getF( d ) = pdf;
  // calculation of velocity and density
  velX += pdf * d.cx();
  velY += pdf * d.cy();
  velZ += pdf * d.cz();
  rho  += pdf;
}
// collide
const real_t vel_sq = 1.5 * ( velX*velX + velY*velY +
velZ*velZ );
for( auto d = LM::Stencil::begin(); d !=
LM::Stencil::end(); ++d )
{
  const real_t vel = d.cx()*velX + d.cy()*velY +
d.cz()*velZ;
  real_t & target = dst.getF(  d );
```

Generic Handwritten C++

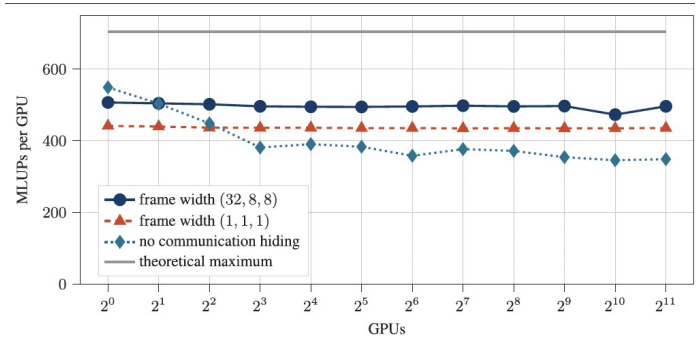
vs.



Specialized Generated C

Results: Multiphase flows

Large scale bubble rise scenario simulated on the Piz Daint supercomputer with several hundred air bubbles.[1]



Weak scaling performance benchmark on the Piz Daint supercomputer.[1]

An example of the bubble propagation through the concentric annular pipe at different timesteps.[2]

Usage of code generation for efficient compute kernels for LBM multiphase flows

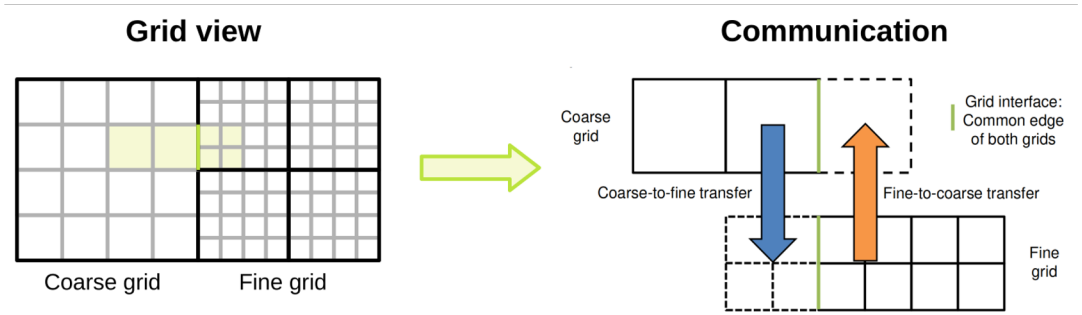
Analysis of physical results and performance

Almost perfect scalability due to code generation for MPI-packing routines

1. M. Holzer, M. Bauer, H. Köstler, et al. “Highly Efficient Lattice Boltzmann Multiphase Simulations of Immiscible Fluids at High-Density Ratios on CPUs and GPUs through Code Generation”. In: The International Journal of High Performance Computing Applications 35.4 (2021). DOI: 10 .1177/10943420211016525.
2. T. Mitchell, M. Holzer, C. Schwarzmeier, et al. “Stability assessment of the phase-field lattice Boltzmann model and its application to Taylor bubbles in annular piping geometries”. In: Physics of Fluids (2021). DOI: 10.1063/5.0061694
3. C. Schwarzmeier, M. Holzer, T. Mitchell, et al. “Comparison of free-surface and conservative Allen-Cahn phase-field lattice Boltzmann method”. In: Journal of Computational Physics (2022). DOI: 10.1016/j.jcp.2022.111753

Recent features for code generation

- lbmpy / pystencils supports:
  - C code (all common CPUs)
  - CUDA (NVIDA GPUs)
  - HIP (AMD GPUs)
  - SYCL prototype (including Intel GPUs)
- New Features
  - Packing / unpacking kernels for non-uniform communication on GPUs
  - reductions incl. optimizations (SIMD vectorization for CPUs, warp reductions NVIDIA GPUs)



- Particle Simulations
  - Ravedutti Lucio Machado, Rafael and Eitzinger, Jan and Köstler, Harald, P4irs: An Intermediate Representation and Compiler for Parallel and Performance-Portable Particle Simulations. Available at SSRN: https://ssrn.com/abstract=4714072

Centre of Excellence in Exascale CFD

Ongoing Work

- Research Software Engineering: Reproducibility
- Strategies for performance portable code

- Mixed-precision
- Surrogate Models and Solvers: Data generation

- Multi-physics applications

Acknowledgements and waLBerla Fundings

- DFG Forschungsgruppe 5134, Erstarrungsrisse beim Laserstrahlschweißen: Hochleistungsrechnen für Hochleistungsprozesse, TP7 Nachhaltiges Daten- und Softwaremanagement für Forschungssoftware zur Simulation von Erstarrungsrissen beim Laserstrahlschweißen DFG Deutsche Forschungsgemeinschaft
- Funded by the European Union. This work has received funding from the European High Performance Computing Joint Undertaking (JU) and Germany under grant agreement number: 101093457, 101093393, 101144014

Co-funded by the European Union

EuroHPC Joint Undertaking

Bundesministerium für Bildung und Forschung

E·C·E

HiDALGO2 CENTRE OF EXCELLENCE

CEEC Centre of Excellence in Exascale CFD

KONWIHR

- Compute resources on supercomputing sites

HLRS

lrz

JÜLICH FORSCHUNGSZENTRUM

CSCS Centro Svizzero di Calcolo Scientifico Swiss National Supercomputing Centre

Vielen Dank
für Ihre Aufmerksamkeit!