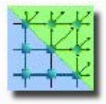
ExaStencils

http://www.exastencils.org/

Jürgen Teich
Frank Hannig
Christian Schmitt

Ulrich Rüde
Harald Köstler
Sebastian Kuckuk

Matthias Bolten
Lisa Claus
Hannah Rittich

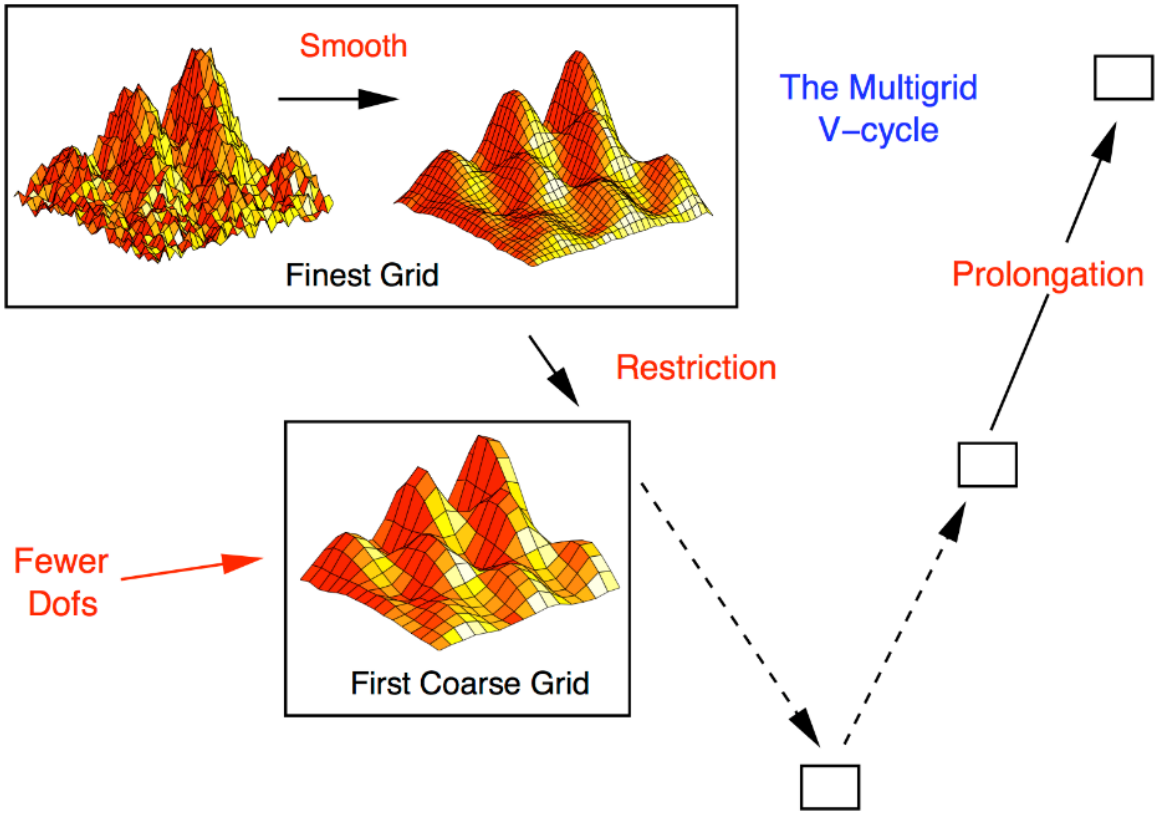


Christian Lengauer
Armin Größlinger
Stefan Kronawitter

Sven Apel
Alexander Grebhahn

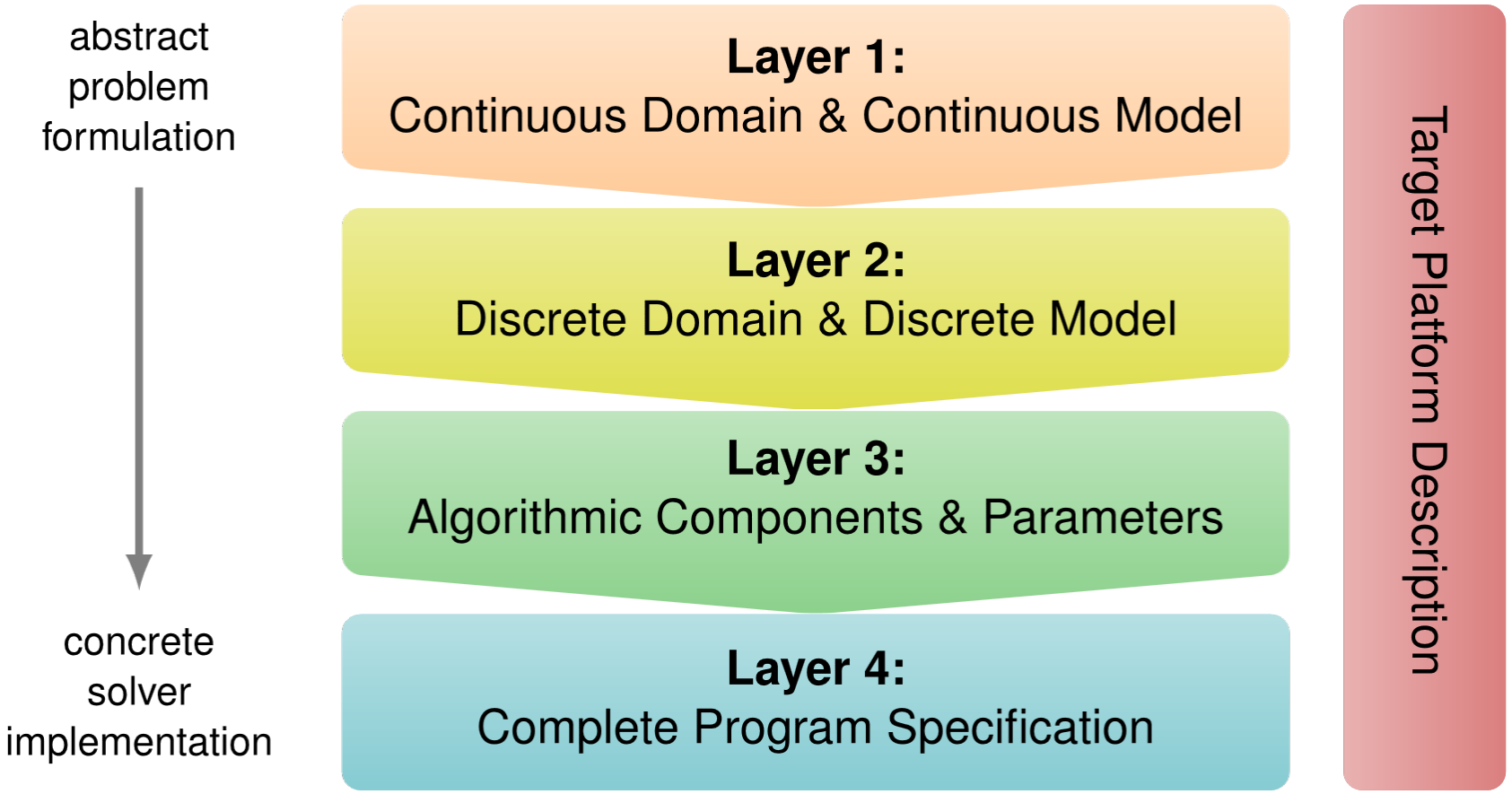
Shigeru Chiba

Stencil Domain: Multigrid



- Elliptic PDEs and systems thereof
- Discretization using finite differences or volumes
- Patch-based domains

Domain-Specific Stencil Language ExaSlang



ExaSlang Layers

- **Layer 1 (continuous)**

  Support of Unicode and LaTeX symbols in a continuous problem definition.
  Optional specification of discretization and solver options used to auto-generate lower layers.
  Support for automatic finite difference discretization of operators.

- **Layer 2 (discrete)**

  Discretized functions are fields (data type, grid location), tied to a domain.
  Geometric information as „virtual fields", resolved to constants or field accesses.
  (Discretized) Operators as stencils or stencil templates.

- **Layer 3 (solver)**

  Specification of a solver for the discrete problem, either by hand or set up automatically.
  Support of a Matlab-like syntax.

- **Layer 4 (application)**

  Tuning of communication patterns. Specification of the main application, I/O, performance evaluation and visualization.

2D Poisson: Layer 1

```
/// inline knowledge

Knowledge { dimensionality = 2
            minLevel       = 2
            maxLevel       = 8
          }

/// problem specification

Domain   Ω = (0,1) × (0,1)

Field    f@finest ∈ Ω = 0.0
Field           u ∈ Ω = 0.0

Field    u@finest ∈ ∂Ω = (vf_boundaryCoord_x**2 - vf_boundaryCoord_y**2)
Field    u@(all but finest) ∈ ∂Ω = 0.0

Operator op = -Δ

Equation uEq@finest              op * u = f
// rhs for the lower levels will be injected at solver layer
Equation uEq@(all but finest)  op * u = 0.0
```

2D Poisson: Layer 1

```
// configuration of inter-layer transformations

DiscretizationHints { f on Node
                      u on Node
                      op on Ω
                      uEq

                      // parameters
                      discr_type = "FiniteDifferences"
                    }

SolverHints { generate solver for u in uEq

              // parameters
              solver_targetResReduction = 1e-6
            }

ApplicationHints { // parameters
                   l4_genDefaultApplication = true
                 }
```

2D Poisson: Layer 2

```
/// inline knowledge

Knowledge { dimensionality = 2
            minLevel       = 2
            maxLevel       = 8
          }

/// problem specification

Domain global< [ 0, 0 ] to [ 1, 1 ] >

Field Solution with Real on Node of global = 0.0

Field Solution@finest on boundary =
                  (vf_boundaryCoord_x ** 2 - vf_boundaryCoord_y ** 2)
Field Solution@(all but finest) on boundary = 0.0

Field RHS with Real on Node of global = 0.0
```

2D Poisson: Layer 2

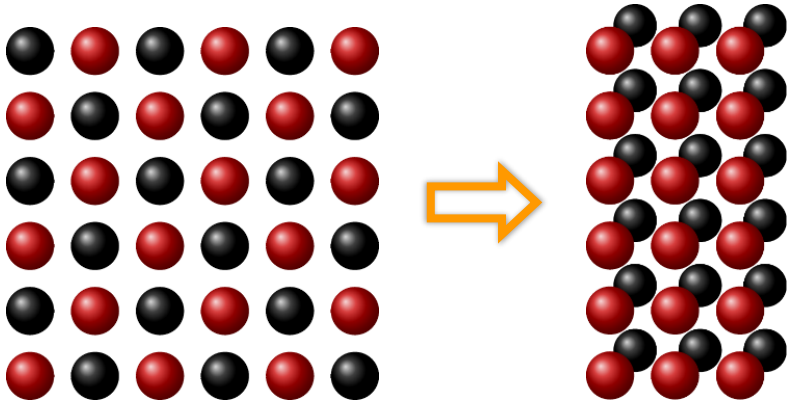
```
Operator Laplace from Stencil {
  [ 0,  0] =>  2.0 / (vf_gridWidth_x ** 2) + 2.0 / (vf_gridWidth_y ** 2)
  [-1,  0] => -1.0 / (vf_gridWidth_x ** 2)
  [ 1,  0] => -1.0 / (vf_gridWidth_x ** 2)
  [ 0, -1] => -1.0 / (vf_gridWidth_y ** 2)
  [ 0,  1] => -1.0 / (vf_gridWidth_y ** 2)
}

Equation solEq@finest {
  Laplace * Solution == RHS
}
Equation solEq@(all but finest) {
  Laplace * Solution == 0.0
}
```

2D Poisson: Layer 3

```
generate solver for Solution in solEq with {
  solver_targetResReduction     = 1e-6
  solver_maxNumIts              = 100

  solver_smoother_jacobiType    = false
  solver_smoother_numPre        = 3
  solver_smoother_numPost       = 3
  solver_smoother_damping       = 0.8
  solver_smoother_coloring      = "red-black"

  solver_cgs                    = "CG"
  solver_cgs_maxNumIts          = 128
  solver_cgs_targetResReduction = 1e-3
}

/// configuration of inter-layer transformations

ApplicationHints {
  // parameters
  l4_genDefaultApplication = true
}
```

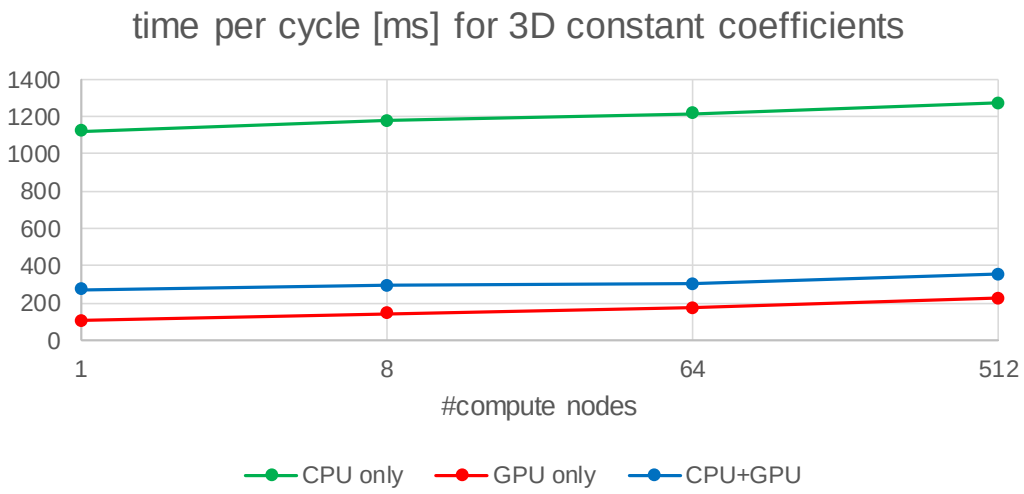
User-Guided Memory Layout Transformation

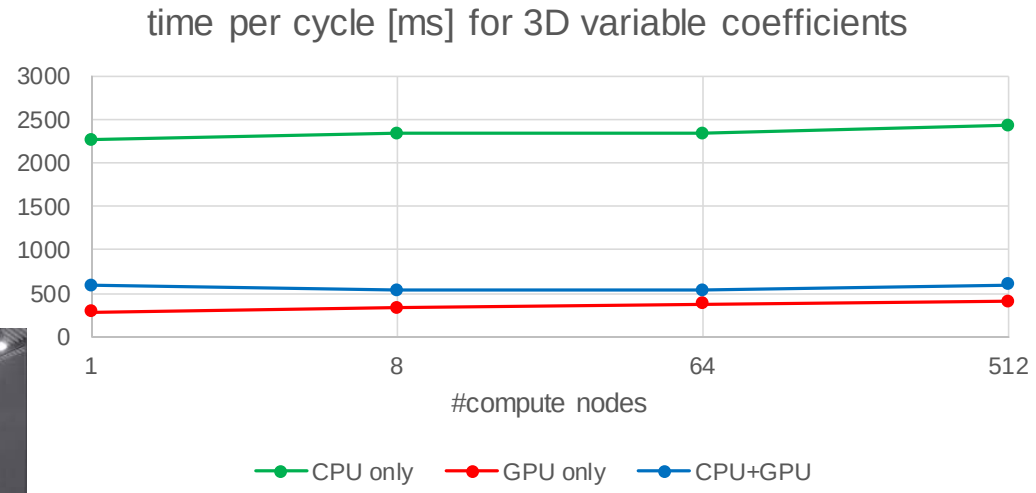
```
// color splitting for smoother
LayoutTransformation {
  transform Solution@all and RHS@all
    with [i0, i1] => [i0/2, i1, (i0+i1) % 2]
}

// Red-Black Gauss-Seidel smoother
Function Smoother@(all but coarsest) {
  color with {
    (i0+i1) % 2,
    loop over Solution {
      Solution = Solution + 0.8 / diag(Laplace) * (RHS - Laplace * Solution)
    }
  }
}
```



Platform Variability

- (The same) ExaSlang input can be mapped to different hardware platforms
  - ”classical” CPUs: x86, PowerPC, ARM
  - GPUs
  - FPGAs
  - ARM
  - and blends of these
- (Automatic) Parallelization using
  - MPI
  - OpenMP (on CPUs)
  - CUDA (on GPUs)
  - ...and combinations
- Scaling experiments on
  - Piz Daint (Lugano)
  - TSUBAME 3.0 (Tokyo)
  - JUQUEEN (Jülich)





Problem Variability

- Beyond Poisson's equation
  - Stokes
  - Navier-Stokes
  - Image processing
  - Non-Newtonian fluids
  - ...
- Each with specialized
  - Grids (may be non-uniform, non-axisparallel, staggered)
  - Discretizations and boundary treatment
  - Solvers (e.g. block smoothers, non-linear multigrid, etc.)

A Block Smoother for Stokes

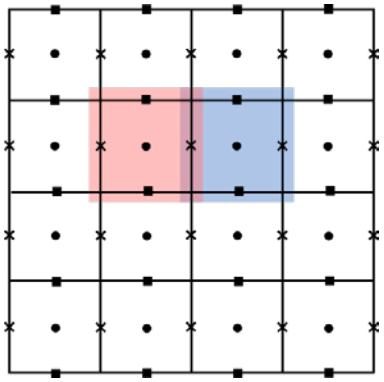
```
loop over p {
 solve locally {
  u@[0, 0] => rhs_u@[0, 0] ==
    Laplace * u@[0, 0] + dxLeft  * p@[0, 0]
  u@[1, 0] => rhs_u@[1, 0] ==
    Laplace * u@[1, 0] + dxLeft  * p@[1, 0]
  …
  p@[0, 0] => rhs_p@[0, 0] ==
    dxRight * u@[0, 0] + dyRight * v@[0, 0]
 }
}
```

ExaSlang 4

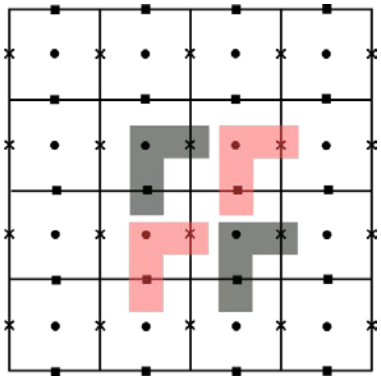
Smoothers for the Stokes Equation

Stokes equations:

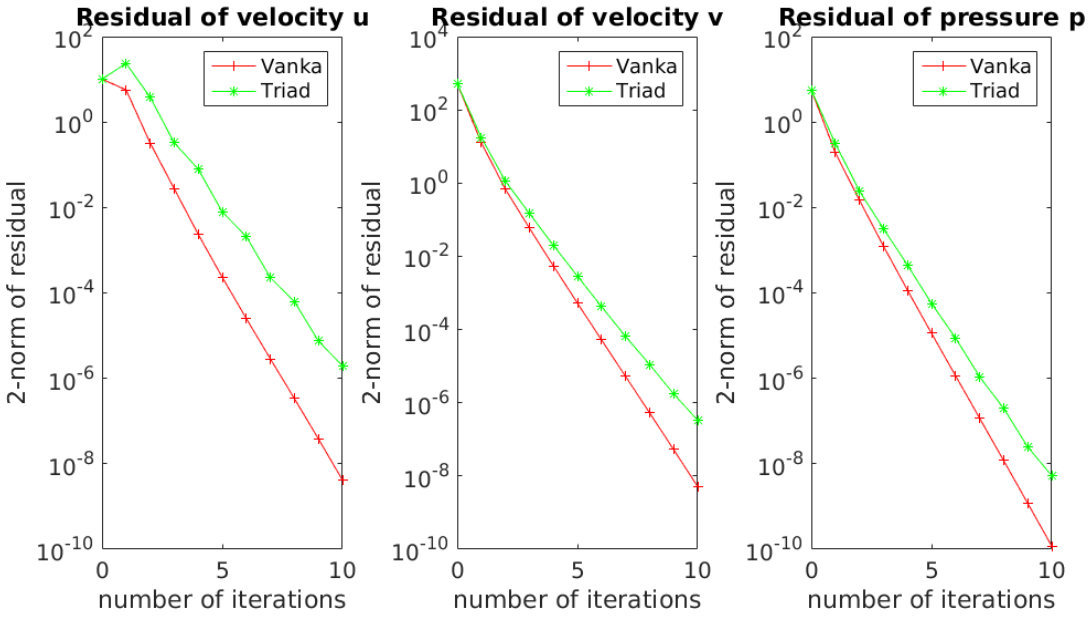
$$\Delta u + \nabla p = f$$
$$\nabla \cdot u = 0$$



Overlapping smoother
- expensive
- parallelization non-trivial



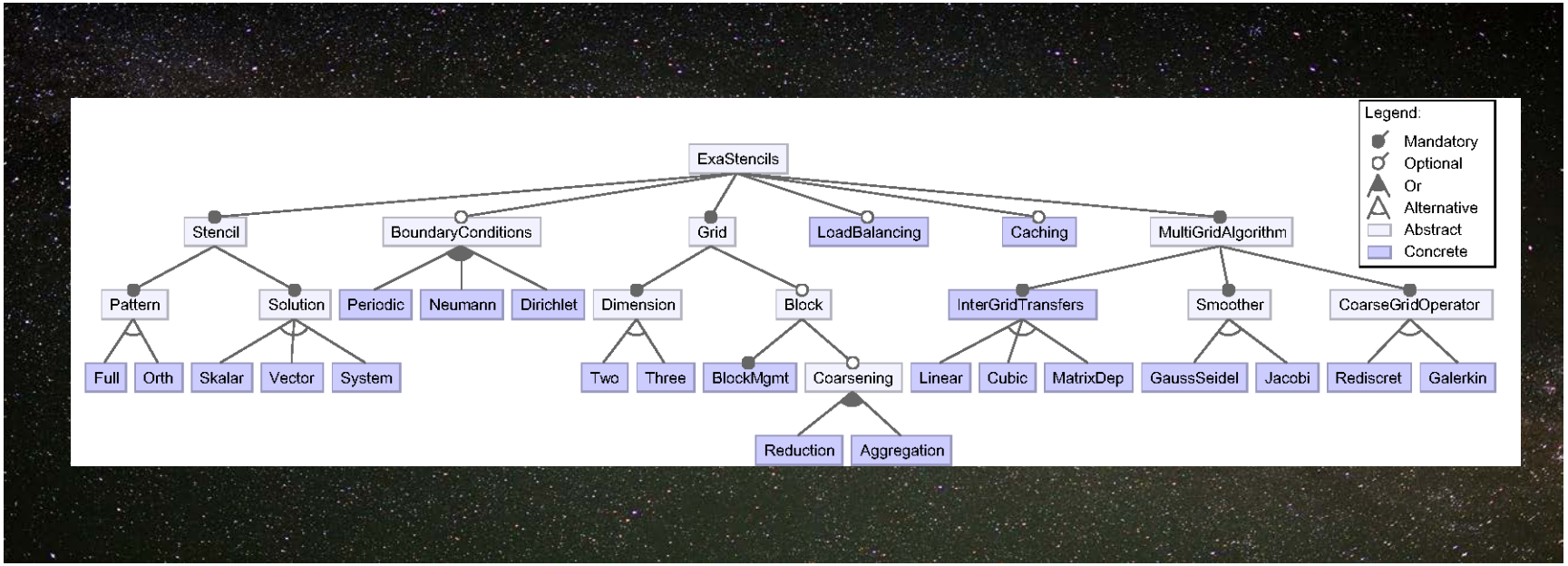
Triad-shape smoother
- cheap
- parallelization straight-forward



Both smoothers are implemented in ExaSlang

ExaStencils

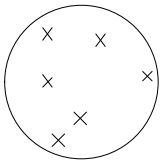
Taming the Variability of the Code-Generator



Sampling

Learning

Performance-Influence Model

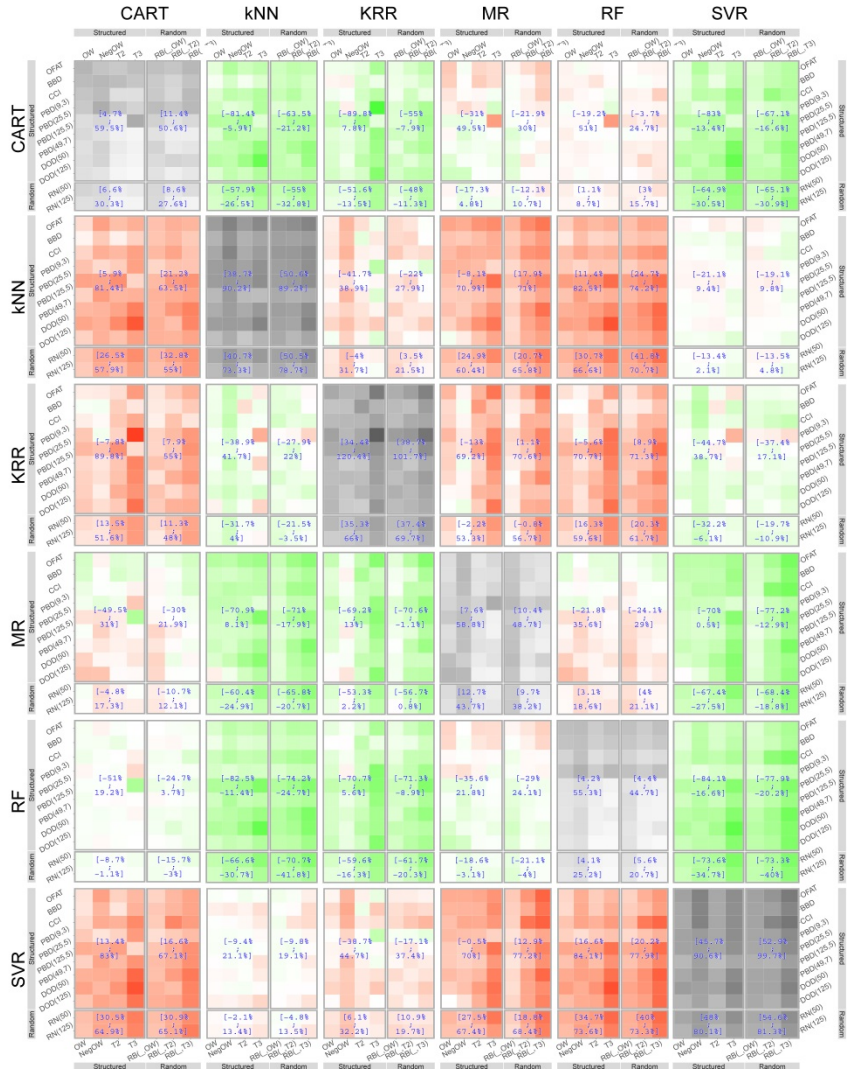


SPL Conqueror

189392.754871324 * root + -5341.1497460973 * cells + -
1097.39668559875 * post + 6852.58846993721 * GradientSolver +
752.542387648181 * BiCGSTABSolver + 1.88645994097499 * cells * post *
post + 547.236997276589 * GradientSolver * pre * pre + -
113.547148735501 * cells * GradientSolver + 0.706383967601375 * cells *
cells * cells + 385.840181761216 * GradientSolver * post * post +
2.72707363219899 * cells * pre + -39.9822314861407 * BiCGSTABSolver *
pre * pre + -11.9701281275527 * cells * GradientSolver * pre * pre + -
8.2288112361045 * cells * GradientSolver * post * post +
112.152829956782 * post * SeqSOR

Validation of the Machine-Learning Technique

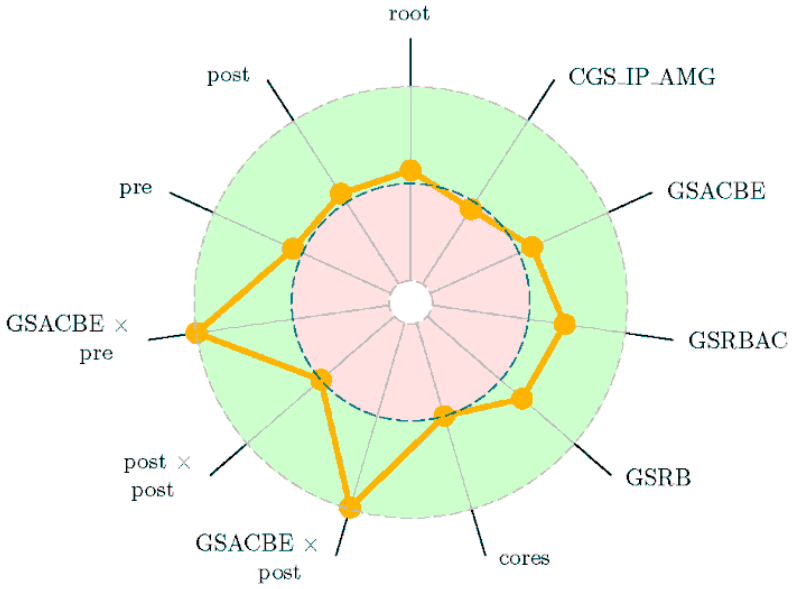
Comparison with other learning techniques



Can we identify performance-optimal parameter settings for different mathematical problems?

case study	min	mean	optimum	speedup
2d-CC	0.854	5.832	0.599	1.42
2d-VC	2.070	12.817	1.488	1.39
3d-CC	1.460	9.431	1.341	1.08
3d-VC	3.673	17.440	2.966	1.24

How can the identified influences be presented to a domain expert?



ExaStencils

Extending and Automating Local Fourier Analysis (LFA)

- Versatile framework for LFA
  - uses Fourier matrix symbols (FMS)
  - uses periodic stencils
  - FMS closed under many operations
  - flexible software implementation

LFA Lab

hrittich.github.io/lfa-lab

PhD Thesis
Rittich, H.
Extending and Automating Fourier Analysis for Multigrid Methods,
University of Wuppertal, 2017

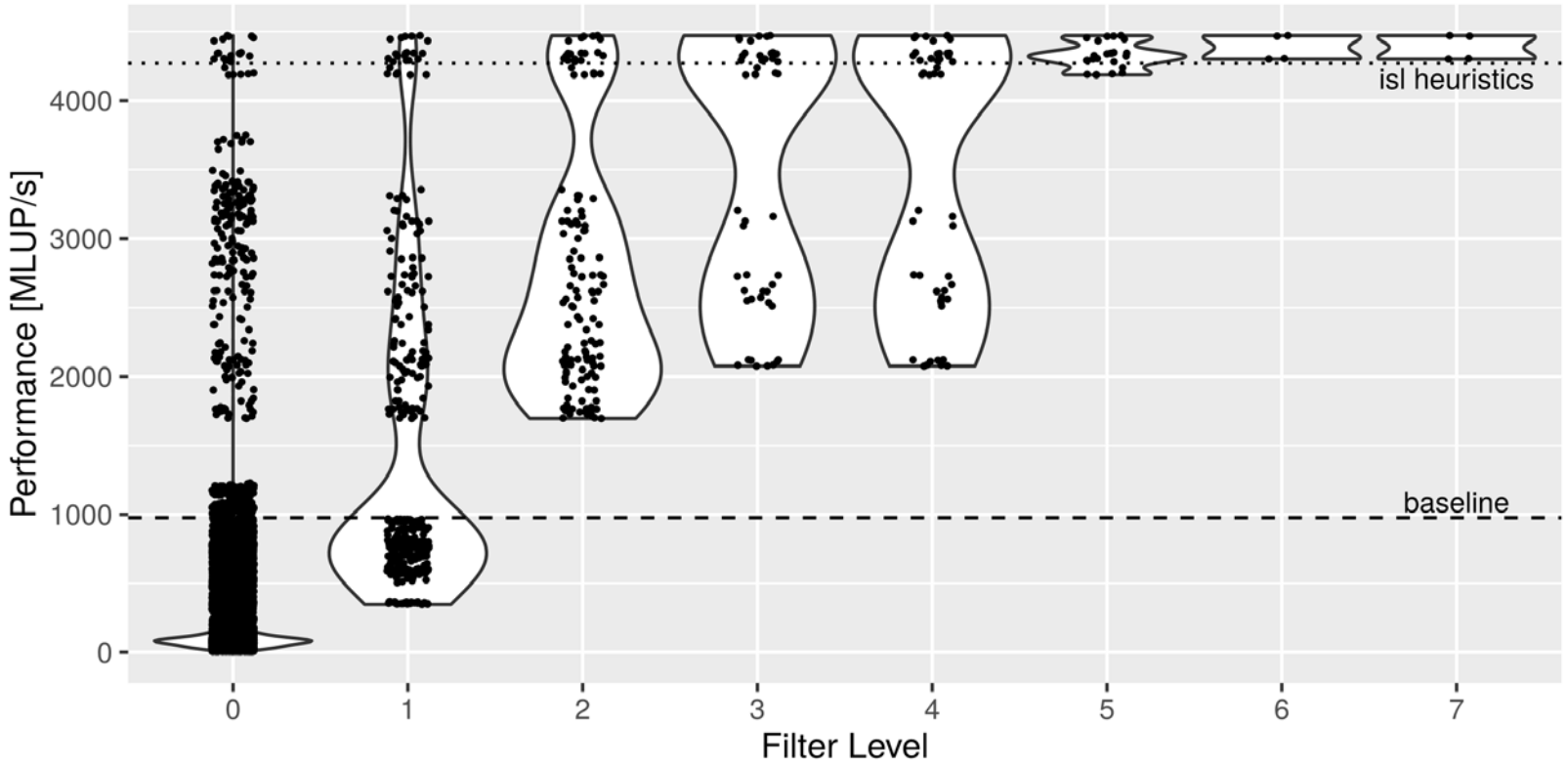
$$E_{Jacobi} = I - \omega D^{-1} A$$

```
from lfa_lab import *
import matplotlib.pyplot as mpp

grid = Grid(2, [1.0/32, 1.0/32])
A = gallery.poisson_2d(grid)
I = operator.identity(grid)
omega = 0.8
E = I - omega * A.diag().inverse() * A

plot.plot_2d(E.symbol())
mpp.show()
```

Node-Level Optimizations

- Polyhedral transformations (temporal blocking)
  - very large number of ill performing transformations
  - Heuristic filters remove bad transformations

Example: exploration for 3D 7-point Jacobi



Japan: An Embedded DSL for Stencil Computing

- Host language: Ruby
  - Reification and target-code generation at execution time of DSL implementation
  - Subset of ExaSlang 4 features semantics transferred into the Ruby ecosystem
  - λ expression and Ruby's flexible syntax mimics ExaSlang 4 syntax
  - High-performance ExaSlang 4 code can be generated from Ruby

ExaSlang 4

```
Stencil SmootherStencil_u@all {
  [1, 0] => -1.0
  ...
  [0, 0] => 4.0*alpha +
     GradientY@current *
     GradientY@current
}
```

Ruby

```
smootherStencil_u = Stencil.new(@all,
  [1, 0] => -1.0,
  ...
  [0, 0] => ->(){ 4.0*alpha +
    gradientX[@current] *
    gradientX[@current]
})
```

```
Function Smoother@all() : Unit {
  communicate Flow_u[active]@current
  loop over fragments {
    loop over Flow_u[active]@current {
      Flow_u[active]@current = ...
  }
  advance Flow_u@current
  ...
```

```
defun(:smoother, @all, :Unit) do
  communicate_ghost_of
  flow_u[active][@current]
  loop_over(fragments) do
    loop_over(flow_u[@current]) do
      flow_u[:next][@current] = ...
  end end
  advance flow_u[@current]
```

ExaStencils Web Interface

- Purpose:
  - Common problem of new programming languages: entry barrier
    → support tools to help new users
  - Web interface makes ExaStencils technology available without local installation
  - Online code generation supplies C++/CUDA target code for download

- Features:
  - persistent user accounts and project storage
  - projects can be shared between users
  - support of the full ExaSlang hierarchy
  - dual-pane editing mode
  - graphical widgets help with object declaration

- Upcoming:
  - recognition of hand-written mathematical expressions
  - Real-time collaborative editing

Publications 2017

- **Special Issue on SPPEXA Dagstuhl Seminar 15161:**
  - Matthias Bolten, Franz Franchetti, Paul H. J. Kelly, Christian Lengauer, and Marcus Mohr. Algebraic Description and Automatic Generation of Multigrid Methods in SPIRAL. *Concurrency and Computation: Practice & Experience*, 29(17):4105:1–4105:11, Sept. 2017.
  - Alexander Grebhahn, Christian Engwer, Matthias Bolten, and Sven Apel. Variability of Stencil Computations for Porous Media. *Concurrency and Computation: Practice & Experience*, 29(17):4119:1–4119:14, Sept. 2017.
  - Alexander Grebhahn, Carmen Rodrigo, Norbert Siegmund, Francisco J. Gaspar, and Sven Apel. Performance-Influence Models of Multigrid Methods: A Case Study on Triangular Meshes. *Concurrency & Computation: Practice and Experience*, 29(17):4057:1–4057:13, Sept. 2017.
  - Sebastian Kuckuk, Gundolf Haase, Diego Vasco, and Harald Köstler. Towards Generating Efficient Flow Solvers with the ExaStencils Approach. *Concurrency and Computation: Practice & Experience*, 29(17):4062:1–4062:17, Sept. 2017.

- **Others:**
  - Harald Köstler, Christian Schmitt, Sebastian Kuckuk, Stefan Kronawitter, Frank Hannig, Jürgen Teich, Ulrich Rüde, and Christian Lengauer. A Scala Prototype to Generate Multigrid Solver Implementations for Different Problems and Target Multi-Core Platforms. *International Journal of Computational Science and Engineering (IJCSE)*, 14(2):150–163, 2017.
  - Didem Unat, Anshu Dubey, Torsten Hoefler, John Shalf, Mark Abraham, Mauro Bianco, Bradford L. Chamberlain, Romain Cledat, H. Carter Edwards, Hal Finkel, Karl Fürlinger, Frank Hannig, Emmanuel Jeannot, Amir Kamil, Jeff Keasler, Paul H. J. Kelly, Vitus J. Leung, Hatem Ltaief, Naoya Maruyama, Chris J. Newburn, and Miquel Pericás. Trends in Data Locality Abstractions for HPC Systems. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 28(10):3007–3020, Oct. 2017.
  - Norbert Siegmund, Stefan Sobernig, and Sven Apel. Attributed Variability Models: Outside the Comfort Zone. In *Proceedings of the European Software Engineering Conference and the ACM SIGSOFT International Symposium on the Foundations of Software Engineering (ESEC/FSE)*, pages 268–278. ACM Press, Sept. 2017.