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

SPPEXA Annual Plenary Meeting 2018, Garching, Christian Lengauer
**Stencil Domain: Multigrid**

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

**The Multigrid V-cycle**

**Restriction**

**Prolongation**

**Smooth**

**Finest Grid**

**Fewer Dofs**

**First Coarse Grid**
Domain-Specific Stencil Language ExaSlang

Layer 1: Continuous Domain & Continuous Model
Layer 2: Discrete Domain & Discrete Model
Layer 3: Algorithmic Components & Parameters
Layer 4: Complete Program Specification

abstract problem formulation

concrete solver implementation

Target Platform Description
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 $\Omega = (0,1) \times (0,1)$

Field $f_{\text{finest}} \in \Omega = 0.0$
Field $u \in \Omega = 0.0$

Field $u_{\text{finest}} \in \partial \Omega = (v_f_{\text{boundaryCoord}_x}^{**2} - v_f_{\text{boundaryCoord}_y}^{**2})$
Field $u_{\text{(all but finest)}} \in \partial \Omega = 0.0$

Operator $\text{op} = -\Delta$

Equation $u_{\text{Eq} \text{finest}} \quad \text{op} \ast u = f$
// rhs for the lower levels will be injected at solver layer
Equation $u_{\text{Eq} \text{(all but finest)}} \text{ op} \ast u = 0.0$
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

```exa slang
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
Taming the Variability of the Code-Generator

Sampling  Learning  Performance-Influence Model
Validation of the Machine-Learning Technique

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

<table>
<thead>
<tr>
<th>case study</th>
<th>min</th>
<th>mean</th>
<th>optimum</th>
<th>speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>2d-CC</td>
<td>0.854</td>
<td>5.832</td>
<td>0.599</td>
<td>1.42</td>
</tr>
<tr>
<td>2d-VC</td>
<td>2.070</td>
<td>12.817</td>
<td>1.488</td>
<td>1.39</td>
</tr>
<tr>
<td>3d-CC</td>
<td>1.460</td>
<td>9.431</td>
<td>1.341</td>
<td>1.08</td>
</tr>
<tr>
<td>3d-VC</td>
<td>3.673</td>
<td>17.440</td>
<td>2.966</td>
<td>1.24</td>
</tr>
</tbody>
</table>

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

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

```python
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()
```

PhD Thesis

Rittich, H.

*Extending and Automating Fourier Analysis for Multigrid Methods*, University of Wuppertal, 2017

hrittich.github.io/lfa-lab
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

![Diagram showing performance comparison between baseline and isl heuristics across different filter levels.](image-url)
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**

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

**Ruby**

```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
  ...
}

**Ruby**

```ruby
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:


Others:

