Systems of Partial Differential Equations in ExaSlang

C. Schmitt¹, S. Kuckuk¹, F. Hannig¹, J. Teich¹, H. Köstler¹, U. Rüde¹, C. Lengauer²
¹ Friedrich-Alexander University Erlangen-Nürnberg
² University of Passau
SPPEXA Symposium, January 25, 2016
ExaSlang
ExaStencils Language

- Domain-specific language (DSL) for the description of highly scalable geometric multigrid solvers
- External DSL
  - Enables greater flexibility and focus at different layers
  - Enables better tailoring of DSL layers towards user needs
- Multi-layered structure
- Top-down design approach: from abstract to concrete
- Very few mandatory specifications at any layer
  → room for decisions at lower layers
- Decisions may be overridden by user via specification in the DSL
ExaSlang

Hierarchical structure of ExaSlang:

1. Abstract problem formulation
   - Layer 1: Continuous Domain & Continuous Model
2. Concrete solver implementation
   - Layer 2: Discrete Domain & Discrete Model
3. Algorithmic Components & Parameters
   - Layer 3: Algorithmic Components & Parameters
4. Complete Program Specification
   - Layer 4: Complete Program Specification

Target Platform Description


Christian Schmitt | Hardware/Software Co-Design | Systems of Partial Differential Equations in ExaSlang

SPPEXA Symposium 2016
ExaSlang

Algorithmic Components & Parameters (Layer 3)

- Multigrid cycle type
- Multigrid components (e.g., selection of smoother)
- Definition of operations on sets (parts of the computational domain)
- Operations in matrix notation

Complete Program Specification (Layer 4)

- Complete multigrid cycle specification
- Custom cycle types
- Operations depending on the multigrid level
- Loops over computational domain
- Communication and data exchange
Data Types for Systems of Partial Differential Equations
Data Types for Systems of Partial Differential Equations

“Can ExaSlang 4 cover all aspects of our domain?”

Systems of PDEs

- Several values per grid point, e.g., velocity in three dimensions
- Coupled problems, e.g., pressure and temperature in flow simulations
- Computations separated per component possible, but often prohibitive:
  - Numerical stability
  - Convergence rate
  - Smaller time steps for explicit time stepping
  - Higher effort to comply with basic physical principles
  - Decreased programming productivity and code readability

→ Representation of point-local vectors and matrices necessary!
→ One step towards mapping from ExaSlang 3 to ExaSlang 4
Data Types for Systems of Partial Differential Equations

Point-local vectors and matrices in ExaSlang

- New data types: `Vector`, `Matrix`
- Available for stencils, fields, variables, constants
- Standard mathematical operators
- Element-wise mathematical operators
- Matrix inversion

```plaintext
1 Var a : Matrix<Real, 3, 3> = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}}
2 Var b : Real<3, 3> = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}}
3 Var c : Real = {1, 2, 3} * {1, 2, 3}^T
4 Var d : Vector<Real, 3>
5 print("Matrix scaling: ", 7 * b)
6 print("Vector addition: ", {1, 2, 3} + {3, 4, 5})
7 print("Matrix multiplication: ", b * {{1, 2}, {3, 4}, {5, 6}})
8 print("Vector mult.: ", {1, 2, 3}T * {1, 2, 3})
9 print("Element-wise mult.: ", {1, 2, 3} .* {1, 2, 3})
10 print("Inversion: ", inv({{1, 2, 3}, {4, 5, 6}, {7, 8, 9}}))
```
Example Application

Optical Flow

- Approximation of apparent motion in sequence of images
- Computation of optical flow vector \((u, v)\)
- No specialized smoothers needed
- Acceptable convergence rates also without handling as system of PDEs

Theoretical Background

- Partial image derivatives \(l_x := \frac{\partial l}{\partial x}, l_y := \frac{\partial l}{\partial y}, l_t := \frac{\partial l}{\partial t}\)
- Spatio-temporal gradient \(\nabla_\theta l := (l_x, l_y, l_t)^T\)
- Optical flow vector \((u, v) = \left(\frac{dx}{dt}, \frac{dy}{dt}\right)\)
- Solve \(-\alpha \Delta u + l_x(l_x u + l_y v) = -l_x l_t\)
  \(-\alpha \Delta v + l_y(l_x u + l_y v) = -l_y l_t\)
- Trivial extension to 3D
Example Application

Example input images and result for a driving car:

(a) Input image 1  
(b) Input image 2  
(c) Optical flow
Example Application

Snippets of 5-point-stencil definition and Jacobi smoother function

```plaintext
1  Stencil SmootherStencil@all {
2      \[ 0, 0\] \mapsto \{ \{ 4.0 \times \alpha + \text{GradX}@current \times \text{GradX}@current, 
3                       \text{GradX}@current \times \text{GradY}@current \}, 
4      \{ \text{GradX}@current \times \text{GradY}@current, 
5                      4.0 \times \alpha + \text{GradY}@current \times \text{GradY}@current \} \}
6  \[ 1, 0\] \mapsto \{ \{ -1.0, 0.0 \}, \{ 0.0, -1.0 \} \}
7  [-1, 0] \mapsto \{ \{ -1.0, 0.0 \}, \{ 0.0, -1.0 \} \}
8  \[ 0, 1\] \mapsto \{ \{ -1.0, 0.0 \}, \{ 0.0, -1.0 \} \}
9  \[ 0,-1\] \mapsto \{ \{ -1.0, 0.0 \}, \{ 0.0, -1.0 \} \}
10 }

1  Function Smoother@all () : Unit {
2      loop over Flow@current {
3          Flow[next}@current = Flow[active]@current + ( 
4              ( inverse ( diag ( SmootherStencil@current ) ) ) * 
5              ( RHS@current - 
6              SmootherStencil@current \times Flow[active]@current )
7          )
8      }
9  advance Flow@current
10 }
```

Christian Schmitt | Hardware/Software Co-Design | Systems of Partial Differential Equations in ExaSlang
Snippets of 5-point-stencil definition and Jacobi smoother function

1. **Stencil** SmootherStencil@all {
2.   [ 0, 0] => { { 4.0 * alpha + GradX@current * GradX@current, 
3.       GradX@current * GradY@current }, 
4.       { GradX@current * GradY@current, 
5.       4.0 * alpha + GradY@current * GradY@current } }
6.   [ 1, 0] => { { -1.0, 0.0 }, { 0.0, -1.0 } }
7.   [-1, 0] => { { -1.0, 0.0 }, { 0.0, -1.0 } }
8.   [ 0, 1] => { { -1.0, 0.0 }, { 0.0, -1.0 } }
9.   [ 0,-1] => { { -1.0, 0.0 }, { 0.0, -1.0 } }
}

Vector expressions in stencil definition

1. **Function** Smoother@all () : Unit {
2.   loop over Flow@current {
3.     Flow[next]@current = Flow[active]@current + ( 
4.       ( inverse ( diag ( SmootherStencil@current ) ) ) * 
5.       ( RHS@current - 
6.         SmootherStencil@current * Flow[active]@current ) 
7.     )
8.   }
9.   advance Flow@current
10. }

Christian Schmitt | Hardware/Software Co-Design | Systems of Partial Differential Equations in ExaSlang

SPPEXA Symposium 2016
Example Application

Snippets of 5-point-stencil definition and Jacobi smoother function

```c
1 Stencil SmootherStencil@all {
2   [ 0, 0] => { { 4.0 * alpha + GradX@current * GradX@current, 
3                  GradX@current * GradY@current },
4                   { GradX@current * GradY@current, 
5                  4.0 * alpha + GradY@current * GradY@current } } 
6   [ 1, 0] => { { -1.0, 0.0 }, { 0.0, -1.0 } }
7   [-1, 0] => { { -1.0, 0.0 }, { 0.0, -1.0 } }
8   [ 0, 1] => { { -1.0, 0.0 }, { 0.0, -1.0 } }
9   [ 0,-1] => { { -1.0, 0.0 }, { 0.0, -1.0 } }
10 }
```

```c
1 Function Smoother@all () : Unit {
2   loop over Flow@current {
3       Flow[next]@current = Flow[active]@current + (    
4          ( inverse ( diag ( SmootherStencil@current ) ) ) * 
5             ( RHS@current - 
6                SmootherStencil@current * Flow[active]@current ) 
7         ) 
8   }
9   advance Flow@current
10 }
```

Automatic inversion of $2 \times 2$ central weight matrix
Example Application

Comparison of program length in lines of code (LoC) for complete optical flow application

<table>
<thead>
<tr>
<th></th>
<th>Scalar</th>
<th>Vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>2D Jacobi</td>
<td>288</td>
<td>242</td>
</tr>
<tr>
<td>2D RBGS</td>
<td>298</td>
<td>247</td>
</tr>
<tr>
<td>3D Jacobi</td>
<td>408</td>
<td>297</td>
</tr>
<tr>
<td>3D RBGS</td>
<td>423</td>
<td>303</td>
</tr>
</tbody>
</table>
Example Application

Comparison of program length in lines of code (LoC) for complete optical flow application

- 2D Jacobi: 288 (Scalar), 242 (Vector)
- 2D RBGS: 297 (Scalar), 298 (Vector)
- 3D Jacobi: 408 (Scalar), 297 (Vector)
- 3D RBGS: 423 (Scalar), 303 (Vector)

16%–28% lines saved for a simple program for Vector over Scalar
Conclusions and Outlook
Conclusions and Outlook

Conclusions

• New data types in ExaSlang 4 for systems of PDEs: local vectors and matrices
• Easy mapping of coupled problems to code
• Easy compliance of simulation with laws of physics
• Increase productivity
• Improve mapping from ExaSlang 3 to ExaSlang 4

Future Work

• Support for block smoothers
• Consider new data types for polyhedral optimization
• Implement more applications, e.g., incompressible Navier-Stokes solver
Systems of Partial Differential Equations in ExaSlang
Christian Schmitt
christian.j.schmitt@fau.de

Thanks for listening. Questions?

ExaStencils
Advanced Stencil Code Engineering
http://www.exastencils.org
ExaStencils is funded by the German Research Foundation (DFG) as part of the Priority Program 1648 (Software for Exascale Computing).