ExaSlang: A Domain-Specific Language for Highly Scalable Multigrid Solvers

Christian Schmitt‡, Sebastian Kuckuk†, Frank Hannig‡, Harald Köstler†, Jürgen Teich‡
‡Hardware/Software Co-Design, †System Simulation,
Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU)
WOLFHPC, New Orleans, LA, USA; November 17, 2014
Motivation

```c
void Smoother() {
    // Code...
}
```

```c
#pragma omp critical
#pragma omp critical
#pragma omp critical
#pragma omp parallel for schedule (static) num_threads (27)
#pragma omp critical
#pragma omp critical

void Smoother() {
    // Code...
}
```

```c
#upper = (iteration Offset End[0] + 7)

reqOutstanding_Send_0[1] = true; } } };

reqOutstanding_Recv_0[2] = true; } } }

reqOutstanding_Send_0[1] = false; } } }

reqOutstanding_Recv_0[0] = false; } } }

reqOutstanding_Send_0[1] = true; } } }

reqOutstanding_Recv_0[2] = true; } } }

reqOutstanding_Recv_0[0] = false; } } }

reqOutstanding_Send_0[1] = true; } } }

reqOutstanding_Recv_0[2] = true; } } }

reqOutstanding_Send_0[1] = false; } } }

reqOutstanding_Recv_0[2] = false; } } }

// ...
```

```c
#upper = std::min((iteration Offset Begin[0]-2+3)-3), (iteration Offset End[0]-7));

slottedFieldDataSolution[slot][5][(((w*19024)+(y*104)+(z*1206))] = buffer_Send_0[0][(((w*9409)+(y*104)+(z*1206))]);

slottedFieldDataSolution[slot][5][(((w*19024)+(y*104)+(z*1206))] = buffer_Send_0[0][(((w*9409)+(y*104)+(z*1206))]);
```

```c
nMax ve0 = nMax sext_pc[0.000000e+00];

nMax ve0 = nMax sext_pc[0.000000e+00];
```

```c
for (int x = iteration Offset Begin[2]; (x < iteration Offset End[2]-7); x += 1) {
    // Code...
}
```

```c
for (int fragmentIdx = 0; fragmentIdx < 1; ++fragmentIdx) {
    // Code...
}
```

```c
for (int w = 0; (w<1); w += 1) {
    for (int z = 1; (z < 98); z += 1) {
        for (int y = 1; (y < 98); y += 1) {
            for (int x = 4; (x < 5); x += 1) {
                // Code...
            }
        }
    }
}
```

```c
for (int y = 1; (y < 98); y += 1) {
    for (int z = 1; (z < 98); z += 1) {
        for (int x = 4; (x < 5); x += 1) {
            // Code...
        }
    }
}
```

```c
for (int x = 1; (x < 100); x += 1) {
    // Code...
}
```

```c
for (int y = 1; (y < 100); y += 1) {
    for (int x = 1; (x < 100); x += 1) {
        // Code...
    }
}
```

```c
for (int z = 1; (z < 100); z += 1) {
    for (int x = 1; (x < 100); x += 1) {
        // Code...
    }
}
```

```c
for (int slot = 0; (slot < nMax slots pd)); slot += 333333e-00;
```

```c
for (int x = iteration Offset Begin[0]-2+3); x < iteration Offset End[0]-7; x += 1) {
    // Code...
}
```

```c
for (int x = iteration Offset Begin[0]-2+3); x < iteration Offset End[0]-7; x += 1) {
    // Code...
}
```

```c
for (int slot = 0; (slot < nMax slots pd)); slot += 333333e-00;
```

```c
for (int fragmentIdx = 0; fragmentIdx < 1; ++fragmentIdx) {
    // Code...
}
```

```c
for (int fragmentIdx = 0; fragmentIdx < 1; ++fragmentIdx) {
    // Code...
}
```

```c
for (int fragmentIdx = 0; fragmentIdx < 1; ++fragmentIdx) {
    // Code...
}
```

```c
for (int fragmentIdx = 0; fragmentIdx < 1; ++fragmentIdx) {
    // Code...
}
```

```c
for (int fragmentIdx = 0; fragmentIdx < 1; ++fragmentIdx) {
    // Code...
}
```

```c
for (int fragmentIdx = 0; fragmentIdx < 1; ++fragmentIdx) {
    // Code...
}
```

```c
for (int fragmentIdx = 0; fragmentIdx < 1; ++fragmentIdx) {
    // Code...
}
```
Motivation

void Smoother() {
    for (int fragmentIdx = 0; fragmentIdx < 1; ++fragmentIdx) {
        if (isValidForSubdomain) {
            for (int w = 0; (w < 1); w += 1) {
                for (int z = 1; (z < 98); z += 1) {
                    for (int y = 97; (y < 98); y += 1) {
                        for (int x = 4; (x < 101); x += 1) {
                            // Critical section
                            #pragma omp critical
                            buffer_Send_0[w][z][y][x] = sledgedFieldData Solution[w][z][y][x];
                        }
                    }
                }
            }
        }
    }
}

void Send() {
    MPI_Isend(buffer_Send_0[w][z][y][x], 9409, MPI_DOUBLE, neighbor_remoteRank[w][z][y][x],
               (neighbor_fragCommId[w][z][y][x] << 10) + (commId << 20), mpiCommunicator,
               &mpiRequest_Send_0[w][z][y][x]);
    waitForMPIReq(&mpiRequest_Send_0[w][z][y][x]);
    MPI_Irecv(buffer_Recv_0[w][z][y][x], 9409, MPI_DOUBLE, neighbor_remoteRank[w][z][y][x],
               (neighbor_fragCommId[w][z][y][x] << 10) + (commId << 20), mpiCommunicator,
               &mpiRequest_Recv_0[w][z][y][x]);
    waitForMPIReq(&mpiRequest_Recv_0[w][z][y][x]);
    waitForMPIReq(&mpiRequest_Send_0[w][z][y][x]);
    MPI_Irecv(buffer_Recv_0[w][z][y][x], 9409, MPI_DOUBLE, neighbor_remoteRank[w][z][y][x],
               (neighbor_fragCommId[w][z][y][x] << 10) + (commId << 20), mpiCommunicator,
               &mpiRequest_Recv_0[w][z][y][x]);
    waitForMPIReq(&mpiRequest_Recv_0[w][z][y][x]);
    waitForMPIReq(&mpiRequest_Recv_0[w][z][y][x]);
}

__m256d vec10;
__m256d vec9 = _mm256_loadu_pd((&sledgedFieldData Solution[curSlot_Solution[w][z][y][x]]));
__m256d vec8 = _mm256_loadu_pd((&sledgedFieldData Solution[curSlot_Solution[w][z][y][x]]));
__m256d vec7 = _mm256_loadu_pd((&sledgedFieldData Solution[curSlot_Solution[w][z][y][x]]));
__m256d vec6 = _mm256_loadu_pd((&sledgedFieldData Solution[curSlot_Solution[w][z][y][x]]));
__m256d vec4 = _mm256_loadu_pd((&sledgedFieldData Solution[curSlot_Solution[w][z][y][x]]));
__m256d vec2 = _mm256_set1_pd(1.333333e-01);
vec10 = _mm256_add_pd(_mm256_mul_pd(vec0, vec1), _mm256_mul_pd(vec2, _mm256_add_pd(_mm256_add_pd(_mm256_add_pd(vec3, vec4), vec5), vec6), vec7), vec8), vec9);
Motivation

Why not concentrate on the algorithmic description?

```plaintext
Function Smoother () : Unit {
    communicate Solution
    loop over Solution {
        Solution = Solution + 0.8 * (1.0 / diag(Laplace)) * 
                   (RHS - Laplace * Solution)
    }
}
```
**Motivation**

Why not concentrate on the algorithmic description?

```plaintext
Function Smoother () : Unit {
  communicate Solution
  loop over Solution {
    Solution = Solution + 0.8 * (1.0 / diag(Laplace)) * 
                (RHS - Laplace * Solution)
  }
}
```

- **Productivity**
  - Algorithm description at high-level
  - Hide low-level details from programmer

- **Portability**
  - Support different target platforms from the same description
  - Support different target languages from the same description

- **Performance**
  - Portable: high performance on different target platforms
  - Competitive: comparable performance to hand-written code
ExaSlang
ExaSlang

- **ExaStencils language**
- Abstract description for generation of massively parallel geometric multigrid solvers
- Multi-layered structure $\rightarrow$ set of Domain-Specific Languages (DSLs)
- Top-down approach: From abstract to concrete
- Very few mandatory specifications at one layer $\rightarrow$ room for decisions at lower layers based on domain knowledge
- External Domain-Specific Language
  - Better reflection of extensive ExaStencils approach
  - Enables greater flexibility of different layers
  - Eases tailoring of DSL layers to users
  - Enables code generation for large variety of target platforms
Basic Multigrid Ideas

Residual on fine grid

Residual on coarse grid

Smother applied
Basic Multigrid Ideas

Multigrid method

1. Pre-smoothing
2. Calculation of residual
3. Restriction
4. Recursive call(s) or solve (at coarsest level)
5. Prolongation
6. Correction
7. Post-smoothing
ExaSlang: Multi-layered DSL Structure

Different layers of DSL tailored towards different users and knowledge.

abstract problem formulation

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

Target Platform Description
ExaSlang: Multi-layered DSL Structure

Different layers of DSL tailored towards different users and knowledge.

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

abstract problem formulation

concrete solver implementation
ExaSlang 4: Complete Program Specification

Properties

- Procedural
- Statically typed
- External DSL
- Syntax partly inspired by Scala
ExaSlang 4: Complete Program Specification

Properties

• Procedural
• Statically typed
• External DSL
• Syntax partly inspired by Scala

Specification of

• Operations depending on the multigrid level
• Loops over computational domain
• Communication and data exchange
• Interface to 3rd-party code
Data Types

Simple and aggregate data types

- Real, Integer, String, Boolean
- Complex<Real>, Complex<Integer>

Algorithmic data types

Field

- Correspond to discretized (mathematical) variables
- Communication scheme via Layout
- Specify Slot number for multiple copies

Stencil

- Correspond to discretized (mathematical) operators
- (Nearly) arbitrary expressions possible
Computations

Loop over computational domain split into loop over fragments
- Fragments stem from distribution across different cluster nodes
- Corresponds to global operation
- Optionally: reduction operators

and loop over <field>
- Iteration over parts of fields possible
- Corresponds to local operation
- Optionally: Reduction operators

Function NormResidual @(coarsest and finest) () : Real {
  Variable res : Real = 0
  loop over fragments with reduction(+ : res) {
    loop over Residual @current with reduction(+ : res) {
      res += Residual @current * Residual @current
    }
  }
  return ( sqrt(res) )
}
Level Specifications

Multigrid is inherently hierarchical and recursive

→ We need
  • Multigrid recursion exit condition
  • Access to other levels’ data & functions

→ Additionally, we want
  • Relative addressing
  • Aliases for certain levels
  • Variable definitions per level

Implementation

• Numerical values, e.g., @0 for bottom level
• Aliases, e.g., @all, @current, @coarser, @coarsest
• Simple expressions, e.g., @(coarsest + 1)
• Lists, e.g., @(1, 3, 5)
• Ranges, e.g., @(1 to 5)
Level Specifications: Example

Disjunct function definition

```plaintext
Function VCycle @((coarsest+1) to finest) () : Unit {
    repeat 3 times {
        Smoother @current ()
    }
    UpResidual @current ()
    Restriction @current ()
    SetSolution @coarser (0)
    VCycle @coarser ()
    Correction @current ()
    repeat 3 times {
        Smoother @current ()
    }
}

Function VCycle @coarsest () : Unit {
    /* ... solve directly ... */
}
```
Level Specifications: Example

Disjunct function definition

```haskell
Function VCycle @((coarsest+1) to finest) () : Unit {
    repeat 3 times {
        Smoother @current ()
    }
    UpResidual @current ()
    Restriction @current ()
    SetSolution @coarser (0)
    VCycle @coarser ()
    Correction @current ()
    repeat 3 times {
        Smoother @current ()
    }
}

Function VCycle @coarsest () : Unit {
    /* ... solve directly ... */
}
```
Level Specifications: Example

No disjunction needed due to overloading

```haskell
Function VCycle @(coarsest to finest) () : Unit {
  repeat 3 times {
    Smoother @current ()
  }
  UpResidual @current ()
  Restriction @current ()
  SetSolution @coarser (0)
  VCycle @coarser ()
  Correction @current ()
  repeat 3 times {
    Smoother @current ()
  }
}

Function VCycle @coarsest () : Unit {
  /* ... solve directly ... */
}
```
Level Specifications: Example

Level Specification can be simplified further

```plaintext
Function VCycle @all () : Unit {
  repeat 3 times {
    Smoother @current ()
  }
  UpResidual @current ()
  Restriction @current ()
  SetSolution @coarser (0)
  VCycle @coarser ()
  Correction @current ()
  repeat 3 times {
    Smoother @current ()
  }
}

Function VCycle @coarsest () : Unit {
  /* ... solve directly ... */
}
```

Christian Schmitt | FAU | ExaSlang: A Domain-Specific Language for Highly Scalable Multigrid Solvers
ExaStencils Transformation Framework
ExaStencils Framework

Abstract workflow:

Algorithmic description

\[ \text{parsing} \rightarrow \text{Intermediate representation} \rightarrow \text{prettyprinting} \rightarrow \text{C++ output} \]
ExaStencils Framework

Using a simple 1-step concept, we can do some refinements, e.g.,

```c
loop over Solution {
    // ....
}
```

is processed to

```c
for (int z = start_z; z < stop_z; z += 1) {
    for (int y = start_y; y < stop_y; y += 1) {
        for (int x = start_x; x < stop_x; x += 1) {
            // ....
        }
    }
}
```
ExaStencils Framework

Using a simple 1-step concept, we can do some refinements, e.g.,

```java
loop over Solution {
    // ....
}
```

is processed to

```java
for (int z = start_z; z < stop_z; z += 1) {
    for (int y = start_y; y < stop_y; y += 1) {
        for (int x = start_x; x < stop_x; x += 1) {
            // ....
        }
    }
}
```


→ Very cumbersome with 1-step approach. Need something more flexible!
ExaStencils Framework

Current workflow

1. DSL input (Layer 4) is parsed
2. Parsed input is checked for errors and transformed into the IR
3. Many smaller, specialized transformations are applied
4. C++ output is prettyprinted
ExaStencils Framework

Current workflow

1. DSL input (Layer 4) is parsed
2. Parsed input is checked for errors and transformed into the IR
3. Many smaller, specialized transformations are applied
4. C++ output is prettyprinted

Concepts

- Major program modifications take place only in IR
- IR can be printed to C++ code
- Small transformations can be enabled and arranged according to needs
- Central instance keeps track of generated program: StateManager
- Variant generation by duplicating program at different transformation stages
ExaStencils Framework

Transformations

- Transform program state into another one
- Are applied to program state in depth-first order
- May be applied to only a part of the program state
- Are grouped together in Strategies
ExaStencils Framework

Transformations

• Transform program state into another one
• Are applied to program state in depth-first order
• May be applied to only a part of the program state
• Are grouped together in Strategies

Strategies

• Are applied in transactions
• Standard strategy that linearly executes all transformations is provided
• Custom strategies possible
ExaStencils Framework

Transactions

- Before execution, a snapshot of the program state is made
- May be committed or aborted

Checkpoints

- A copy of program state during compilation
- Restoration of program states
- Acceleration of variant generation for design space exploration
ExaStencils Framework

Example transformations:

```java
var s = DefaultStrategy("example strategy")

// rename a certain stencil
s += Transformation("rename stencil", {
    case x : Stencil if(x.identifier == "foo")
        =>
        {
            if(x.entries.length != 7) error("invalid stencil size")
            x.identifier = "bar"; x
        }
})

// evaluate additions
s += Transformation("eval adds", {
    case AdditionExpression(l : IntegerConstant, r : IntegerConstant)
        => IntegerConstant(l + r)
})

s.apply // execute transformations sequentially
```
ExaStencils Framework

Implemented workflow:

Algorithmic description

- parsing

L4

... IR

IR

prettyprinting

IR

IR

C++ output
First Results
Program Sizes during Transformation

- V(3,3) cycle, Jacobi smoother, CG coarse grid solver
- Hybrid MPI/OpenMP
Generated Lines of Code

<table>
<thead>
<tr>
<th>Method</th>
<th>ExaSlang 4</th>
<th>C++ Pure MPI</th>
<th>C++ Hybrid MPI/OMP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jacobi</td>
<td>244</td>
<td>11,259</td>
<td>13,432</td>
</tr>
<tr>
<td>Gauss-Seidel</td>
<td>236</td>
<td>9,600</td>
<td>11,320</td>
</tr>
<tr>
<td>Red-Black GS</td>
<td>240</td>
<td>9,776</td>
<td>12,887</td>
</tr>
</tbody>
</table>
Weak-Scaling Results

Solution of Poisson’s equation in 3D, V(3,3) cycle, Jacobi, CG
Average time per V-cycle on JUQUEEN

![Graph showing weak-scaling results for different numbers of cores and MPI/OMP combinations.](image-url)
Summary

Presented

- Multi-layered DSL ExaSlang for multigrid-based numerical solvers
- Framework for specification of transformations and code generation
- Generation of highly scalable C++ code

Conclusions

- Code generation is a viable approach to generation of multigrid codes
- Specialized DSLs allow for concise algorithmic descriptions (Productivity)
- Generation of solvers for different target platforms (Portability)
- More work on optimizations needed, but scalability already good (Performance)
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).