FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG
FACULTY OF ENGINEERING

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

```
void Smoother() {
for (int fragmentIdx = 0; fragmentIdx < 1; ++fragmentIdx) {if (isValidForSubdomain) {if ((neighbor_isValid[1]&&neighbor_isRemote[1])) {
#pragma omp parallel for schedule(static) num_threads(27)
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 = 100; (x<101); x += 1) {buffer_Send_0[1][(((((w*9409)+x)+y)+(z*97))-198)] = slottedFieldData_Solution[slot][5][((((w*1019304)+x)+(y*104))+(z*10296))];}}}}}}}
for (int fragmentIdx = 0; fragmentIdx < 1; ++fragmentIdx) {if (isValidForSubdomain) {if ((neighbor_isValid[1]&&neighbor_isRemote[1])) {
#pragma omp critical
{ MPI_Isend(buffer_Send_0[1], 9409, MPI_DOUBLE, neighbor_remoteRank[1], (((unsigned int)commId << 20)+((unsigned int)(neighbor_fragCommId[1]) << 10)), mpiCommunicator, &mpiRequest_Send_0[1]); }
reqOutstanding_Send_0[1] = true;}}};
for (int fragmentIdx = 0; fragmentIdx < 1; ++fragmentIdx) {if (isValidForSubdomain) {if ((neighbor_isValid[0]&&neighbor_isRemote[0])) {
#pragma omp critical
{ MPI_Irecv(buffer_Recv_0[0], 9409, MPI_DOUBLE, neighbor_remoteRank[0], (((unsigned int)neighbor_fragCommId[0] << 20)+((unsigned int)(commId) << 10)), mpiCommunicator, &mpiRequest_Recv_0[0]); }
reqOutstanding_Recv_0[0] = true;}}}
for (int fragmentIdx = 0; fragmentIdx < 1; ++fragmentIdx) {if (isValidForSubdomain) {if (reqOutstanding_Recv_0[0]) {
#pragma omp critical
{ waitForMPIReq(&mpiRequest_Recv_0[0]); }
reqOutstanding_Recv_0[0] = false;}}}
for (int fragmentIdx = 0; fragmentIdx < 1; ++fragmentIdx) {if (isValidForSubdomain) {if ((neighbor_isValid[0]&&neighbor_isRemote[0])) {
#pragma omp parallel for schedule(static) num_threads(27)
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) {
slottedFieldData_Solution[slot][5][((((w*1019304)+x)+(y*104))+(z*10296))] = buffer_Recv_0[0][(((((w*9409)+x)+y)+(z*97))-102)];}}}}}}};
for (int fragmentIdx = 0; fragmentIdx < 1; ++fragmentIdx) {if (isValidForSubdomain) {if (reqOutstanding_Send_0[1]) {
#pragma omp critical
{ waitForMPIReq(&mpiRequest_Send_0[1]); }
reqOutstanding_Send_0[1] = false;}}}
for (int fragmentIdx = 0; fragmentIdx < 1; ++fragmentIdx) {if (isValidForSubdomain) {if ((neighbor_isValid[3]&&neighbor_isRemote[3])) {
#pragma omp parallel for schedule(static) num_threads(27)
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) {
buffer_Send_0[3][(((((w*9409)+x)+(y*97))+(z*97))-9510)] = slottedFieldData_Solution[slot][5][((((w*1019304)+x)+(y*104))+(z*10296))];}}}}}}}
for (int fragmentIdx = 0; fragmentIdx < 1; ++fragmentIdx) {if (isValidForSubdomain) {if ((neighbor_isValid[3]&&neighbor_isRemote[3])) {
#pragma omp critical
{ MPI_Isend(buffer_Send_0[3], 9409, MPI_DOUBLE, neighbor_remoteRank[3], (((unsigned int)commId << 20)+((unsigned int)(neighbor_fragCommId[3]) << 10)), mpiCommunicator, &mpiRequest_Send_0[3]); }
reqOutstanding_Send_0[3] = true;}}};
for (int fragmentIdx = 0; fragmentIdx < 1; ++fragmentIdx) {if (isValidForSubdomain) {if ((neighbor_isValid[2]&&neighbor_isRemote[2])) {
#pragma omp critical
{ MPI_Irecv(buffer_Recv_0[2], 9409, MPI_DOUBLE, neighbor_remoteRank[2], (((unsigned int)neighbor_fragCommId[2] << 20)+((unsigned int)(commId) << 10)), mpiCommunicator, &mpiRequest_Recv_0[2]); }
reqOutstanding_Recv_0[2] = true;}}}
for (int fragmentIdx = 0; fragmentIdx < 1; ++fragmentIdx) {if (isValidForSubdomain) {if (reqOutstanding_Recv_0[2]) {
#pragma omp critical
{ waitForMPIReq(&mpiRequest_Recv_0[2]); }
reqOutstanding_Recv_0[2] = false;}}}
// ...
// ...
// ...
for (int fragmentIdx = 0; fragmentIdx < 1; ++fragmentIdx) {if (isValidForSubdomain) {for (int z = iterationOffsetBegin[2]; (z<(iterationOffsetEnd[2]+7)); z += 1) {for (int y = iterationOffsetBegin[1]; (y<(iterationOffsetEnd[1]+7)); y += 1) {
int x = iterationOffsetBegin[0];
int _upper = std::min(((iterationOffsetBegin[0]+3)&(~3)), (iterationOffsetEnd[0]+7));
for (; (x<_upper); x += 1) {
slottedFieldData_Solution[(((curSlot_Solution[1]+1)%2)][1][(((x+(y*12))+(z*108))+124)] = ((slottedFieldData_Solution[(curSlot_Solution[1]%2)][1][(((x+(y*12))+(z*108))+124)]*2.000000e-01)+(((((((fieldData_RHS[1][((x+(y*8))+(z*56))]+slottedFieldData_Solution[(curSlot_Solution[1]%2)][1][(((
x+(y*12))+(z*108))+112)])+slottedFieldData_Solution[(curSlot_Solution[1]%2)][1][(((x+(y*12))+(z*108))+123)])+slottedFieldData_Solution[(curSlot_Solution[1]%2)][1][(((x+(y*12))+(z*108))+125)])+slottedFieldData_Solution[(curSlot_Solution[1]%2)][1][(((x+(y*12))+(z*108))+136)])+
slottedFieldData_Solution[(curSlot_Solution[1]%2)][1][(((x+(y*12))+(z*108))+16)])+slottedFieldData_Solution[(curSlot_Solution[1]%2)][1][(((x+(y*12))+(z*108))+232)])*1.333333e-01));}
__m256d vec0 = _mm256_set1_pd(2.000000e-01);
__m256d vec2 = _mm256_set1_pd(1.333333e-01);
_upper = (iterationOffsetEnd[0]+7);
for (x = ((iterationOffsetBegin[0]+3)&(~3)); (x<(_upper-3)); x += 4) {
__m256d vec1 = _mm256_loadu_pd((&slottedFieldData_Solution[(curSlot_Solution[1]%2)][1][(((x+(y*12))+(z*108))+124)]));
__m256d vec3 = _mm256_loadu_pd((&fieldData_RHS[1][((x+(y*8))+(z*56))]));
__m256d vec4 = _mm256_loadu_pd((&slottedFieldData_Solution[(curSlot_Solution[1]%2)][1][(((x+(y*12))+(z*108))+112)]));
__m256d vec5 = _mm256_loadu_pd((&slottedFieldData_Solution[(curSlot_Solution[1]%2)][1][(((x+(y*12))+(z*108))+123)]));
__m256d vec6 = _mm256_loadu_pd((&slottedFieldData_Solution[(curSlot_Solution[1]%2)][1][(((x+(y*12))+(z*108))+125)]));
__m256d vec7 = _mm256_loadu_pd((&slottedFieldData_Solution[(curSlot_Solution[1]%2)][1][(((x+(y*12))+(z*108))+136)]));
__m256d vec8 = _mm256_loadu_pd((&slottedFieldData_Solution[(curSlot_Solution[1]%2)][1][(((x+(y*12))+(z*108))+16)]));
__m256d vec9 = _mm256_loadu_pd((&slottedFieldData_Solution[(curSlot_Solution[1]%2)][1][(((x+(y*12))+(z*108))+232)]));
__m256d vec10;
vec10 = _mm256_add_pd(_mm256_mul_pd(vec0, vec1), _mm256_mul_pd(vec2, _mm256_add_pd(_mm256_add_pd(_mm256_add_pd(_mm256_add_pd(_mm256_add_pd(_mm256_add_pd(vec3, vec4), vec5), vec6), vec7), vec8), vec9)));
_mm256_storeu_pd((&slottedFieldData_Solution[(((curSlot_Solution[1]+1)%2)][1][(((x+(y*12))+(z*108))+124)]), vec10);}
for (; (x<_upper); x += 1) {slottedFieldData_Solution[(((curSlot_Solution[1]+1)%2)][1][(((x+(y*12))+(z*108))+124)] = ((slottedFieldData_Solution[(curSlot_Solution[1]%2)][1][(((x+(y*12))+(z*108))+124)]*2.000000e-01)+(((((((fieldData_RHS[1][((x+(y*8))+(z*56))]+slottedFieldData_Solution[(
curSlot_Solution[1]%2)][1][(((x+(y*12))+(z*108))+112)])+slottedFieldData_Solution[(curSlot_Solution[1]%2)][1][(((x+(y*12))+(z*108))+123)])+slottedFieldData_Solution[(curSlot_Solution[1]%2)][1][(((x+(y*12))+(z*108))+125)])+slottedFieldData_Solution[(curSlot_Solution[1]%2)][1][(((x+(y*12)
)+(z*108))+136)])+slottedFieldData_Solution[(curSlot_Solution[1]%2)][1][(((x+(y*12))+(z*108))+16)])+slottedFieldData_Solution[(curSlot_Solution[1]%2)][1][(((x+(y*12))+(z*108))+232)])*1.333333e-01));
}}}}
curSlot_Solution[1] = ((curSlot_Solution[1]+1)%2);
}}}
```

Motivation

```
void Smoother() {
for (int fragmentIdx = 0; fragmentIdx < 1; ++fragmentIdx) {if (isValidForSubdomain) {if ((neighbor_isValid[1]&&neighbor_isRemote[1])) {
#pragma omp parallel for schedule(static) num_threads(27)
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 = 100; (x<101); x += 1) {buffer_Send_0[1][(((((w*9409)+x)+y)+(z*97))-198)] = slottedFieldData_Solution[slot][5][((((w*1019304)+x)+(y*104))+(z*10296))];}}}}}}}
for (int fragmentIdx = 0; fragmentIdx < 1; ++fragmentIdx) {if (isValidForSubdomain) {if ((neighbor_isValid[1]&&neighbor_isRemote[1])) {
#pragma omp critical
{ MPI_Isend(buffer_Send_0[1], 9409, MPI_DOUBLE, neighbor_remoteRank[1], (((unsigned int)commId << 20)+((unsigned int)(neighbor_fragCommId[1]) << 10)), mpiCommunicator, &mpiRequest_Send_0[1]); }
reqOutstanding_Send_0[1] = true;}}};
for (int fragmentIdx = 0; fragmentIdx < 1; ++fragmentIdx) {if (isValidForSubdomain) {if ((neighbor_isValid[0]&&neighbor_isRemote[0])) {
#pragma omp critical
{ MPI_Irecv(buffer_Recv_0[0], 9409, MPI_DOUBLE, neighbor_remoteRank[0], (((unsigned int)neighbor_fragCommId[0] << 20)+((unsigned int)(commId) << 10)), mpiCommunicator, &mpiRequest_Recv_0[0]); }
reqOutstanding_Recv_0[0] = true;}}}
for (int fragmentIdx = 0; fragmentIdx < 1; ++fragmentIdx) {if (isValidForSubdomain) {if (reqOutstanding_Recv_0[0]) {
#pragma omp critical
{ waitForMPIReq(&mpiRequest_Recv_0[0]); }
reqOutstanding_Recv_0[0] = false;}}}
for (int fragmentIdx = 0; fragmentIdx < 1; ++fragmentIdx) {if (isValidForSubdomain) {if ((neighbor_isValid[0]&&neighbor_isRemote[0])) {
#pragma omp parallel for schedule(static) num_threads(27)
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) {
slottedFieldData_Solution[slot][5][((((w*1019304)+x)+(y*104))+(z*10296))] = buffer_Recv_0[0][(((((w*9409)+x)+y)+(z*97))-102)];}}}}}}};
for (int fragmentIdx = 0; fragmentIdx < 1; ++fragmentIdx) {if (isValidForSubdomain) {if (reqOutstanding_Send_0[1]) {
#pragma omp critical
{ waitForMPIReq(&mpiRequest_Send_0[1]); }
reqOutstanding_Send_0[1] = false;}}}
for (int fragmentIdx = 0; fragmentIdx < 1; ++fragmentIdx) {if (isValidForSubdomain) {if ((neighbor_isValid[3]&&neighbor_isRemote[3])) {
#pragma omp parallel for schedule(static) num_threads(27)
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) {
buffer_Send_0[3][((((((w*9409)+x)+(y*97))+(z*97))-9510)] = slottedFieldData_Solution[slot][5][((((w*1019304)+x)+(y*104))+(z*10296))];}}}}}}}
for (int fragmentIdx = 0; fragmentIdx < 1; ++fragmentIdx) {if (isValidForSubdomain) {if ((neighbor_isValid[3]&&neighbor_isRemote[3])) {
#pragma omp critical
{ MPI_Isend(buffer_Send_0[3], 9409, MPI_DOUBLE, neighbor_remoteRank[3], (((unsigned int)commId << 20)+((unsigned int)(neighbor_fragCommId[3]) << 10)), mpiCommunicator, &mpiRequest_Send_0[3]); }
reqOutstanding_Send_0[3] = true;}}};
for (int fragmentIdx = 0; fragmentIdx < 1; ++fragmentIdx) {if (isValidForSubdomain) {if ((neighbor_isValid[2]&&neighbor_isRemote[2])) {
#pragma omp critical
{ MPI_Irecv(buffer_Recv_0[2], 9409, MPI_DOUBLE, neighbor_remoteRank[2], (((unsigned int)neighbor_fragCommId[2] << 20)+((unsigned int)(commId) << 10)), mpiCommunicator, &mpiRequest_Recv_0[2]); }
reqOutstanding_Recv_0[2] = true;}}}
for (int fragmentIdx = 0; fragmentIdx < 1; ++fragmentIdx) {if (isValidForSubdomain) {if (reqOutstanding_Recv_0[2]) {
#pragma omp critical
{ waitForMPIReq(&mpiRequest_Recv_0[2]); }
reqOutstanding_Recv_0[2] = false;}}}
//...........................................................................................................................
//...........................................................................................................................
//...........................................................................................................................
for (int fragmentIdx = 0; fragmentIdx < 1; ++fragmentIdx) {if (isValidForSubdomain) {for (int z = iterationOffsetBegin[2]; (z<(iterationOffsetEnd[2]+7)); z += 1) {for (int y = iterationOffsetBegin[1]; (y<(iterationOffsetEnd[1]+7)); y += 1) {
int x = iterationOffsetBegin[0];
int _upper = std::min(((iterationOffsetBegin[0]+3)&(~3)), (iterationOffsetEnd[0]+7));
for (; (x<_upper); x += 1) {
slottedFieldData_Solution[((curSlot_Solution[1]+1)%2)][1][((((x+(y*12))+(z*108))+124)] = ((slottedFieldData_Solution[(curSlot_Solution[1]%2)][1][((((x+(y*12))+(z*108))+124)]*2.000000e-01)+(((((((fieldData_RHS[1][(((x+(y*8))+(z*56))]+slottedFieldData_Solution[(curSlot_Solution[1]%2)][1][(((
x+(y*12))+(z*108))+112)])+slottedFieldData_Solution[(curSlot_Solution[1]%2)][1][((((x+(y*12))+(z*108))+123)])+slottedFieldData_Solution[(curSlot_Solution[1]%2)][1][((((x+(y*12))+(z*108))+125)])+slottedFieldData_Solution[(curSlot_Solution[1]%2)][1][((((x+(y*12))+(z*108))+136)])+
slottedFieldData_Solution[(curSlot_Solution[1]%2)][1][((((x+(y*12))+(z*108))+16)])+slottedFieldData_Solution[(curSlot_Solution[1]%2)][1][((((x+(y*12))+(z*108))+232)])*1.333333e-01));}
__m256d vec0 = _mm256_set1_pd(2.000000e-01);
__m256d vec2 = _mm256_set1_pd(1.333333e-01);
_upper = (iterationOffsetEnd[0]+7);
for (x = ((iterationOffsetBegin[0]+3)&(~3)); (x<(_upper-3)); x += 4) {
__m256d vec1 = _mm256_loadu_pd((&slottedFieldData_Solution[(curSlot_Solution[1]%2)][1][((((x+(y*12))+(z*108))+124)]));
__m256d vec3 = _mm256_loadu_pd((&fieldData_RHS[1][(((x+(y*8))+(z*56))]));
__m256d vec4 = _mm256_loadu_pd((&slottedFieldData_Solution[(curSlot_Solution[1]%2)][1][((((x+(y*12))+(z*108))+112)]));
__m256d vec5 = _mm256_loadu_pd((&slottedFieldData_Solution[(curSlot_Solution[1]%2)][1][((((x+(y*12))+(z*108))+123)]));
__m256d vec6 = _mm256_loadu_pd((&slottedFieldData_Solution[(curSlot_Solution[1]%2)][1][((((x+(y*12))+(z*108))+125)]));
__m256d vec7 = _mm256_loadu_pd((&slottedFieldData_Solution[(curSlot_Solution[1]%2)][1][((((x+(y*12))+(z*108))+136)]));
__m256d vec8 = _mm256_loadu_pd((&slottedFieldData_Solution[(curSlot_Solution[1]%2)][1][((((x+(y*12))+(z*108))+16)]));
__m256d vec9 = _mm256_loadu_pd((&slottedFieldData_Solution[(curSlot_Solution[1]%2)][1][((((x+(y*12))+(z*108))+232)]));
__m256d vec10;
vec10 = _mm256_add_pd(_mm256_mul_pd(vec0, vec1), _mm256_mul_pd(vec2, _mm256_add_pd(_mm256_add_pd(_mm256_add_pd(_mm256_add_pd(_mm256_add_pd(_mm256_add_pd(vec3, vec4), vec5), vec6), vec7), vec8), vec9)));
_mm256_storeu_pd((&slottedFieldData_Solution[((curSlot_Solution[1]+1)%2)][1][((((x+(y*12))+(z*108))+124)]), vec10);}
for (; (x<_upper); x += 1) {slottedFieldData_Solution[((curSlot_Solution[1]+1)%2)][1][((((x+(y*12))+(z*108))+124)] = ((slottedFieldData_Solution[(curSlot_Solution[1]%2)][1][((((x+(y*12))+(z*108))+124)]*2.000000e-01)+(((((((fieldData_RHS[1][(((x+(y*8))+(z*56))]+slottedFieldData_Solution[(
curSlot_Solution[1]%2)][1][((((x+(y*12))+(z*108))+112)])+slottedFieldData_Solution[(curSlot_Solution[1]%2)][1][((((x+(y*12))+(z*108))+123)])+slottedFieldData_Solution[(curSlot_Solution[1]%2)][1][((((x+(y*12))+(z*108))+125)])+slottedFieldData_Solution[(curSlot_Solution[1]%2)][1][((((x+(y*12)
)+(z*108))+136)])+slottedFieldData_Solution[(curSlot_Solution[1]%2)][1][((((x+(y*12))+(z*108))+16)])+slottedFieldData_Solution[(curSlot_Solution[1]%2)][1][((((x+(y*12))+(z*108))+232)])*1.333333e-01));
}}}}
curSlot_Solution[1] = ((curSlot_Solution[1]+1)%2);
}}}
```

OpenMP

MPI

SIMD extensions

Motivation

Why not concentrate on the algorithmic description?

```
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?

```
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

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

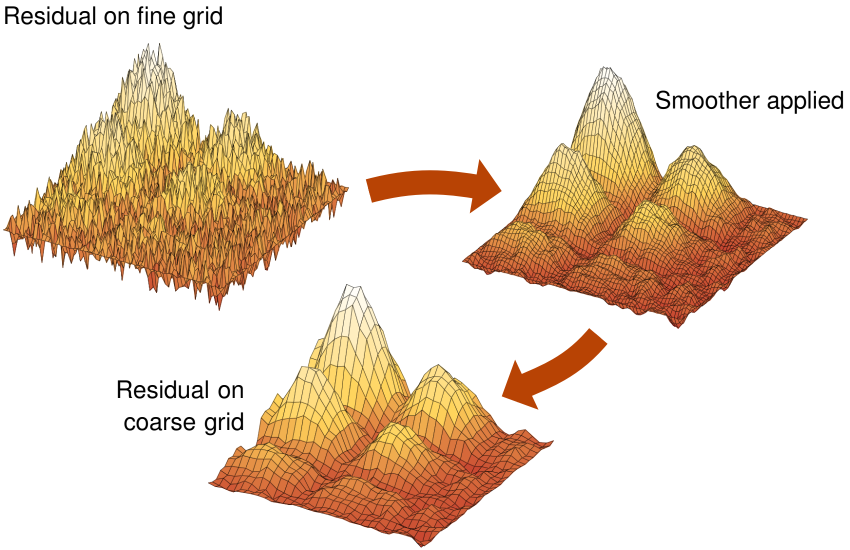
FACULTY OF ENGINEERING

ExaSlang

ExaSlang

- **ExaS**tencils **lang**uage
- Abstract description for generation of massively parallel geometric multigrid solvers
- Multi-layered structure ➔ set of Domain-Specific Languages (DSLs)
- Top-down approach: From abstract to concrete
- Very few mandatory specifications at one layer
  ➔ 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

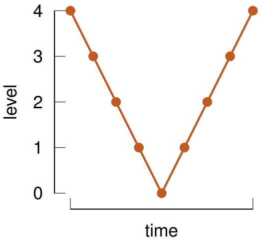
Basic Multigrid Ideas

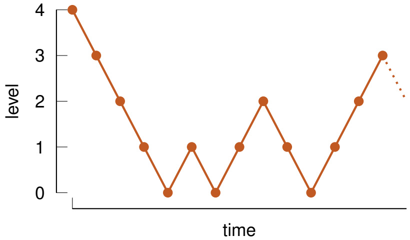


Basic Multigrid Ideas

Multigrid method

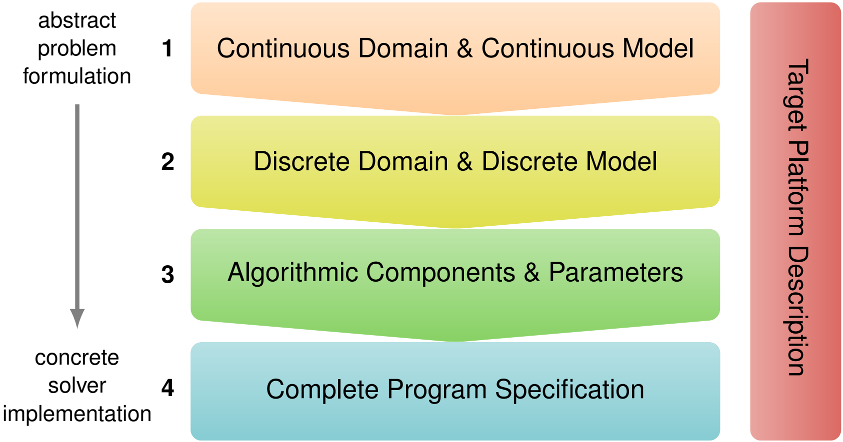
1. Pre-smoothing
2. Calculation of residual
3. Restriction
4. Recursive call(s) or solve
   (at coarsest level)
5. Prolongation
6. Correction
7. Post-smoothing





ExaSlang: Multi-layered DSL Structure

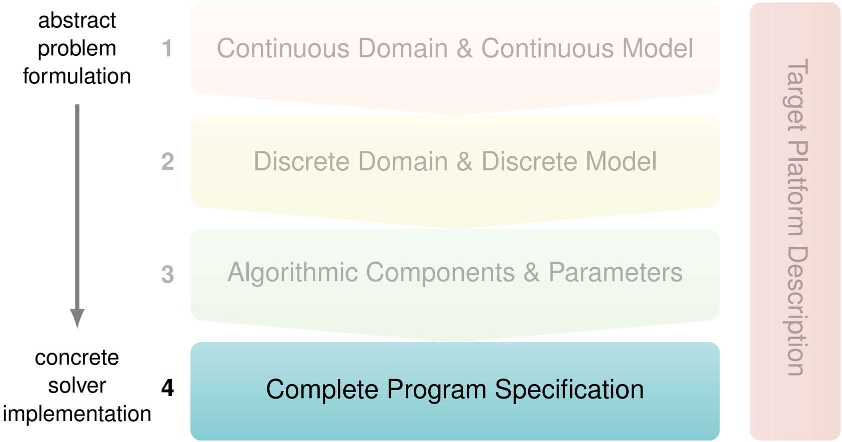
Different layers of DSL tailored towards different users and knowledge.



FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG
FACULTY OF ENGINEERING

ExaSlang: Multi-layered DSL Structure

Different layers of DSL tailored towards different users and knowledge.



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

```
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

```
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

```
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

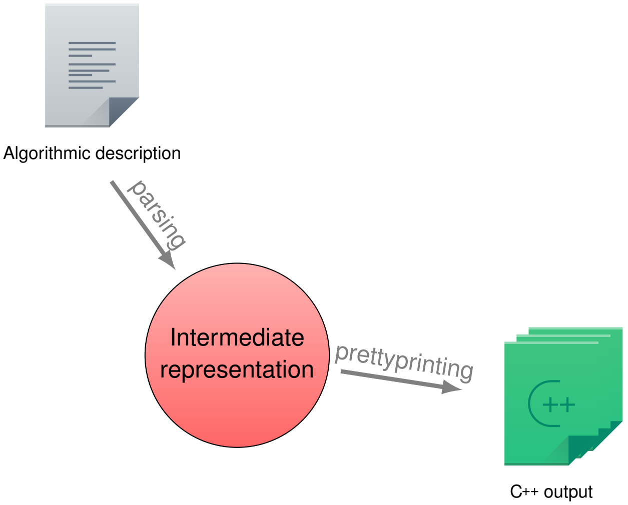
```
Function VCycle @all () : Unit {
    repeat 3 times {
        Smoother @current ()
    }
    UpResidual @current ()
    Restriction @current ()
    SetSolution @coarser (0)
    VCycle @coarser ()
    Correction @current ()
    repeat 3 times {
        Smoother @current ()
    }
}

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

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG
FACULTY OF ENGINEERING

ExaStencils Transformation Framework

ExaStencils Framework

Abstract workflow:



ExaStencils Framework

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

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

is processed to

```
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.,

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

is processed to

```
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) {
            // ....
        }
    }
}
```

But what about the calculations? What about more complex things? Optional code modifications? Parallelization? Vectorization? Blocking? Color splitting?

➔ 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

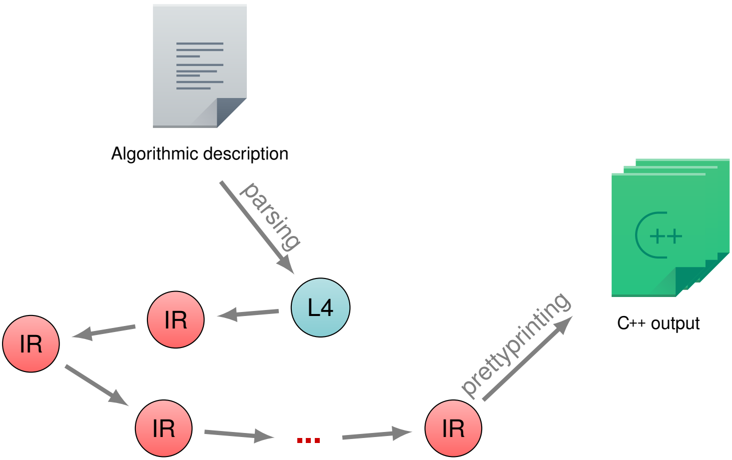
ExaStencils Framework

Example transformations:

```
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

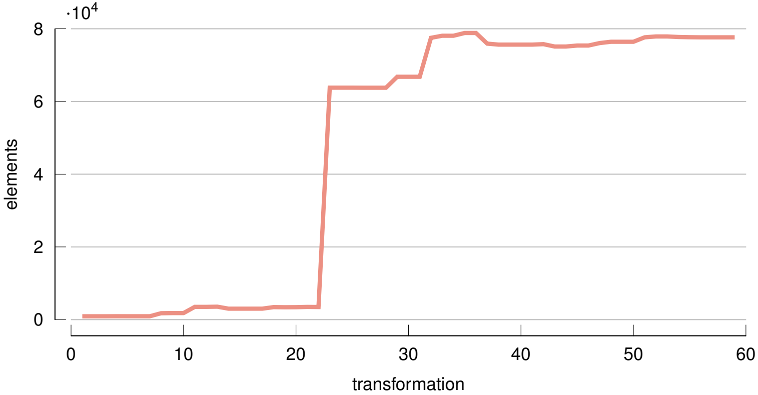
Implemented workflow:



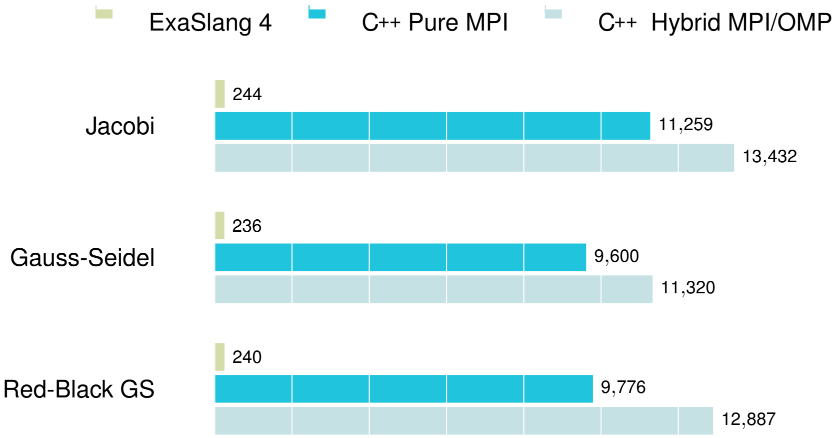
First Results

Program Sizes during Transformation

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

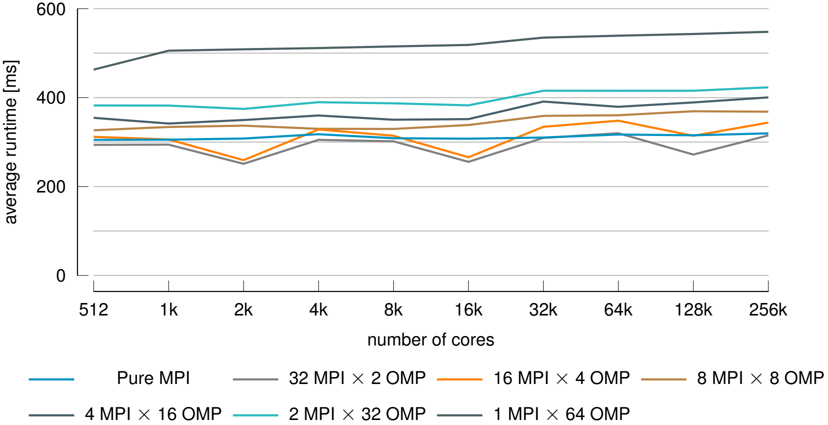


Generated Lines of Code



Weak-Scaling Results

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



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

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).