A Multi-layered Domain-specific Language for Stencil Computations

Christian Schmitt, Frank Hannig, Jürgen Teich

Hardware/Software Co-Design, University of Erlangen-Nuremberg

Workshop ExaStencils 2014, Dresden, Germany; March 31, 2014
Challenges for Software Development in Computational Science and Engineering

Current Situation

- **Hardware**: Modern HPC clusters are massively parallel
  - Parallel intra-core, intra-node, and inter-node
  - Increasing heterogeneity
- **Applications**: Become more complex with increasing computation power
  - More complex (physical) models
  - Code development in interdisciplinary teams
- **Algorithm**: Multigrid is a general idea
  - Components and parameters depend on type of problem, grid, ...

→ Software development has to address all these issues!
Challenge: The 3 P’s

**Performance**

- Portable: high performance on different target hardware
- Competitive: performance comparable to hand-written code

**Portability**

- Support different target architectures from the same algorithm description
- Support different target languages and technologies from the same algorithm description

**Productivity**

- Algorithm description at a high level
- Hide low-level details from the user
Domain-Specific Language (DSL) Definitions

“Domain-Specific Languages: An Annotated Bibliography”, Arie van Deursen, Paul Klint, und Joost Visser:

“A domain-specific language (DSL) is a programming language or executable specification language that offers, through appropriate notations and abstractions, expressive power focused on, and usually restricted to, a particular problem domain.”

“Domain-Specific Languages“, Martin Fowler:

“Domain-specific language: a computer programming language of limited expressiveness focused on particular domain.”
Advantages of DSLs

Productivity
- Yield *average* programmers from the difficulties of parallel programming
- Priority is given to the development of algorithms and applications, i.e., focus is not on details of a low-level implementation

Performance
- Exploitation of generic parallel execution patterns in a domain at high abstraction level
- Reduced (restricted) expressiveness
- Full extraction of available parallelism
- Make use of knowledge (domain, architecture) for static and dynamic optimizations

Portability and Scalability
- DSL and run-time system should be easily extendable in order to adapt to new architectures
- Applications (DSL codes) should remain the same
- Allows hardware engineers to introduce innovations without worrying about portability
Goals of a DSL

Domain-specific: provide only expressions relevant to the topic

Orthogonal: one single way of specifying something

Expressive and compact: describe relevant constructs with few statements

Abstract: work on a high-level point of view

Adaptable: support complex things

Adoptable: employ terms and concepts of the domain

Regular: all terms should follow the same syntax and ideas

Well-defined: non-ambiguous and easy to understand

→ Talk about what should be computed, not how.
   (declarative vs. imperative)
Two Approaches to Creating DSLs

Internal / embedded DSLs

- Utilize a general-purpose programming language (host language)
- Extension or restriction of the host language (or both at the same time)
- Extensions possible in form of libraries (e.g., data types, objects, methods), annotations, macros, etc.
- Same syntax as host language and usually the same compiler or interpreter

External DSLs

- Completely new defined programming language
- More flexible and expressive than internal DSLs
- Syntax and semantics defined freely, but often related to existing languages
- Higher design effort, but supporting tools exist
- Potential to create a powerful semantic model as intermediate representation (IR)
Motivation & Introduction to DSLs

Language Design Guidelines

The ExaStencils DSL
Language Design Guidelines
Language Design Guidelines

It’s all about simplicity!

Language Design Guidelines

- Identify users
- Identify uses
- Appropriate representation
- Use domain concepts
- Re-use existing languages
- Focus on essentials
- Avoid redundancy

- Language purpose
- Language realization
- Language content

- Design Guidelines

- Concrete syntax
- Be descriptive
- Avoid redundancy

- Abstract syntax
- Align abstract & concrete syntax

- Enable modularity
- Stay consistent

---

The ExaStencils DSL
Goals of the ExaStencils DSL

Three different types of users with individual expectations:

Natural scientists
- Care about **effectivity** of solving the problem
- Little knowledge of underlying methods
- No requirements to override compiler-chosen decisions

Mathematicians
- Care about **applicability** and **convergence** of mathematical models
- Little interest in implementation specifics
- Want to extend/override compiler-chosen decisions with custom multigrid components

Computer scientists
- Care about **performance and software engineering** approach
- Little interest in the mathematical problem
- Have to understand of the compiler and its decision process
ExaStencils DSL

- Multi-layered structure
- Top-down approach: From abstract to concrete
- Very few mandatory specifications on one layer → room for SPL-based decisions on lower layers
- Decisions may be taken by user via specification in DSL
- Mandatory specifications include
  - Shape and size of computational domain
  - Continuous equation
- External DSL
  - Better reflection of extensive ExaStencils approach
  - Enables greater flexibility of different layers
  - Eases tailoring of DSL layers to users
ExaStencils: Multi-layered DSL Structure

Different layers of DSL tailored towards different users and knowledge:

1. **abstract**
   - Continuous Domain & Continuous Model

2. **Discrete Domain & Discrete Model**

3. **Algorithmic Components & Parameters**

4. **concrete**
   - Complete Program Specification

Hardware Description
ExaStencils: 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

Natural scientists

Hardware Description
ExaStencils: Multi-layered DSL Structure

Different layers of DSL tailored towards different users and knowledge:

- Continuous Domain & Continuous Model
- Discrete Domain & Discrete Model
- Algorithmic Components & Parameters
- Complete Program Specification

Matematicians

Hardware Description
ExaStencils: Multi-layered DSL Structure

Different layers of DSL tailored towards different users and knowledge:

- Continuous Domain & Continuous Model
- Discrete Domain & Discrete Model
- Algorithmic Components & Parameters
- Complete Program Specification

Computer scientists

Hardware Description
ExaStencils: Multi-layered DSL Structure

Continuous Domain & Continuous Model (Layer 1)

Specification of

- Size and structure of computational domain
- Variables
- Functions and operators (pre-defined functions and operators also available)
- Mathematical problem

```
Domain d = UnitCube

Function f = 0
Unknown solution = 1
Operator Lapl = Laplacian

PDE pde { Lapl(solution) = f }
PDEBC bc { solution = 0 }
Accuracy = 7
```
ExaStencils: Multi-layered DSL Structure

Discrete Domain & Discrete Model (Layer 2)

Discretization of

- Computational domain into fragments (e.g., triangles)
- Variables to fields
  - Specification of data type
  - Selection of discretized location (cell based or node based)

Transformation of energy functional to PDE or weak form

```csharp
Fragment f1 = UnitCube
Discrete_Domain d { ... }
Field<Double, 1>@nodes f
Stencil<Double, 1, 1, FD, 2>@nodes Lapl
// ....
```
ExaStencils: Multi-layered DSL Structure

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

```plaintext
1  Iteration smoother { u = Mu + Nf }
2  Set s1 = [0, 0] - [END, END], [+1, +1]
3
4  u(s1) = A(s1, s1) * u(s1)
5
6  Set s2 = ( [0,0] + [1,1] ), [+2, +2]
7  Set s3 = [0:2, 0:2], [+2, +2]
8  // ...
```
ExaStencils: Multi-layered DSL Structure

Complete Program Specification (Layer 4)

- Complete multigrid V-cycle, or
- Custom cycle types
- Operations depending on the multigrid level
- Loops over computational domain
- Communication and data exchange
- Custom output data formats

```plaintext
1 def V@(1 to 6) () : Unit {
2     repeat up 3 { GaussSeidel
3         @(current) () }
4     Residual@(current) ()
5     Restrict@(current) ()
6     // ...
7 }

8 def Application() : Unit {
9     var res0 : Real = sqrt ( L2Residual @6 () )
10     // ...
11     repeat up 10 {
12         V @6 ()
13     } // ...
```
ExaStencils: Multi-layered DSL Structure

Target Hardware Description

- Availability of compute units (e.g., CPUs, GPUs, FPGAs) and capabilities
- Memory and cache architecture
- Cluster nodes (e.g., number of cores, types of available accelerators, available software)
- Complete cluster (e.g., number of nodes, interconnect topology and characteristics, I/O storage specifications)

```plaintext
Hardware {
  MPI {
    name                  "my cluster"
    id                    HYBRID_CLUSTER
    nodes                 32
    components {         
      XEON_CPU [2]
      GTX_680_GPU [2]
    }
  }
  // ...
```
ExaStencils Concept

- **End-user**
  - Domain expert
  - Mathematician
  - Software specialist
  - Hardware expert

- **DSL program** → Discretization and algorithm selection → Software component selection via SPL → Polyhedral optimization → Code generation → Tuning towards target hardware → Exascale C++
ExaStencils: Multi-layered DSL Structure

- Continuous Domain & Continuous Model
  - discretization
- Discrete Domain & Discrete Model
  - parametrization
- Algorithmic Components & Parameters
  - specification
- Complete Program Specification
  - optimization
- Intermediate Representation (IR)
  - generation
- Exascale C++ Code
ExaStencils Framework

Implementation details

- Implemented in Scala
- Specialized data structures for each DSL layer
- Domain-knowledge modeled as compiler-wide accessible module
- A central instance keeps track of program state changes: `StateManager`
- Functionality separated into namespaces, e.g.,
  - `exastencils.core`: Log functionality, `StateManager`, compiler settings
  - `exastencils.core.collectors`
  - `exastencils.datastructures`: Annotations, program state duplication, `trait Strategy`, `trait Transformation`
  - `exastencils.datastructures.{L1, L2, L3, L4, IR}`
  - `exastencils.parsers`
  - `exastencils.prettyprinting`
ExaStencils Framework

Transformations

- Are grouped together in strategies
- Are atomic – either applied completely or not at all
- Are applied to program state in depth search order
- May be applied to a part of the program state
- Carry an identifier

Strategies

- Are applied by the StateManager
- Carry an identifier
- A standard strategy for sequential execution of transformations is provided
- Custom strategies possible
ExaStencils Framework

Transition between layers

- Transition from one layer to another via transformations
- Step-wise from one layer to the next
- Domain knowledge needed for each transition (e.g., which discretization or which smoother to choose)
- Transitions are atomic – no mix of nodes of different layers

Intermediate Representation (IR) Layer

- Layer in which most optimizations take place
- Progress is made by application of many strategies with a single and narrow purpose
- Polyhedral model to be used for partitioning of computational domain
- Can be prettyprinted
ExaStencils Framework: Example transformations (IR)

```java
var s = Strategy("example standard strategy")

// replace constant '1' under a certain node with '3'
s += Transformation("t1",
    { case x : Constant if(x.Value == 1)
      => Constant(3)
    }, someProgramStatementNode)

// rename all variables to 'j'
s += Transformation("t2",
    { case x : Variable
      => Variable("j", x.Type)
    })

// duplicate all methods
s += Transformation("t3",
    { case x : FunctionStatement
      => List(x, FunctionStatement(
          x.returntype, x.name + "_", x.parameters, x.body))
    })

s.apply // execute transformations sequentially
```
Summary and Outlook

Discussed in this talk

- Definitions of DSLs
- Language Guidelines
- Multi-layered DSL for ExaStencils
- Language examples
- Implementation details

Future work

- Finalization of language specifications on different layers
- Transformations of more abstract layers to lowest layer (Layer 4 (Complete Program Specification))
- Domain knowledge as a base for automated decision making and optimization
Thanks for listening. Questions?