Metaprogramming for CSE
Applications meets performance engineering

PD Dr. Harald Köstler
ISC, Frankfurt 2017
Contents

- Motivation
- ExaStencils Code Generation Framework
  - Applications
  - Performance Engineering
- Next: Metaprogramming
**Code generation and Metaprogramming**

**Definition:** Code generation in a strict sense is used to describe the process of producing code that can be executed on a certain platform from an abstract representation.

**Why?** Because programming is time-consuming.

**Definition:** Metaprogramming is a very general term and can be defined as writing software that can itself produce new software. The final goal is to develop software that can itself solve problems without human interaction.

**Approaches:** Genetic Programming, Deep Learning
CSE optimization problem

- Code generation

- Simulation

- Real World

- Model

- Metaprogramming

- improve

- Data

- Domain knowledge

≈

produce
Metaprogramming group @ LSS

- Code generation
  - Parallel Data Structures
  - Performance engineering
  - GPU, accelerators
- Metaprogramming
  - Lisp, AnyDSL
  - Scala
  - Python, llvm
  - C++
  - Genetic Programming, Deep Learning
- Applications
  - Data analysis, medical image processing
  - Solvers for PDEs, DG discretizations
  - Tsunami simulation, material sciences, CFD
The ExaStencils Framework
Project ExaStencils

- Sebastian Kuckuk
- Georg Altmann
- Harald Köstler
- Ulrich Rüde

- Christian Schmitt
- Frank Hannig
- Jürgen Teich

- Hannah Rittich
- Lisa Claus
- Matthias Bolten

- Alexander Grebhahn
- Sven Apel

- Stefan Kronawitter
- Armin Größlinger
- Christian Lengauer
What is ExaStencils?

- **Scope:**
  - (Geometric) **multigrid solvers** for
  - Finite difference and finite volume discretizations of **elliptic PDEs**
  - On regular (staggered) quad and hex meshes

- **Multi-layered, external DSL** as input, whole program as output
- Transformation-based **code generation framework** written in Scala
- Support for OpenMP, MPI and/or CUDA
- Many **optimizations can be applied automatically** (loop transformations, vectorization, sophisticated CSE, etc.)
- Applications from **CFD**, image processing and quantum chemistry

http://www.exastencils.org/#pubs
We propose a **multilayered, external DSL**

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

---

Workflow

Setup
• User writes DSL code and sets up configuration files

Gener.
• Code generator emits source code in target language and, optionally, a Makefile, project file, job script, etc.

Comp.
• User moves generated code to target machine and uses general purpose compilers

Run
• User executes the code and checks results
/** Layer 2 **/

// domain – information is carried over from L1
Domain global

// fields from L1
Field Solution@all with Real on Node of global = 0.0
Field RHS@all with Real on Node of global = 0.0

// boundary conditions from L1
Field Solution@finest on boundary = vf_boundaryCoord_x**2 - 0.5 * ...
Field Solution@(all but finest) on boundary = 0.0

// (discretized) operators from L1
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)
}
/** Layer 3 **/

/* fields and operators from L2 are progressed automatically */

// new fields
Field Residual from Solution

// override inherited boundary conditions where applicable
override bc for Residual@finest with 0.0

// generate default inter-grid operators for node-based discretizations
Operator RestrictionStencil from default restriction on Node with 'linear'
Operator CorrectionStencil from default prolongation on Node with 'linear'

// create smoother function
Function Smoother@all {
    Val omega : Real = 0.8
    repeat 3 times {
        Solution =
            Solution + omega * diag_inv ( Laplace ) * ( RHS - Laplace * Solution )
    }
}
/** Layer 3 **/

// create v-cycle function
Function VCycle((coarsest + 1) to finest) {
    Smoother ( )

    Residual = RHS - ( Laplace * Solution )
    RHS@coarser = RestrictionStencil * Residual

    Solution@coarser = 0.0
    VCycle@coarser ( )

    Solution += CorrectionStencil * Solution@coarser

    Smoother ( )
}

Function VCycle@coarsest {
    /* CGS */
}
/** Layer 4 **/

// create main function
Function Application ( ) : Unit {
    /* init code */
    Var resStart : Real = NormResidual@finest ( )
    Var curRes : Real = resStart

    repeat until res_0 < 1.0E-5 * resStart_0 {
        VCycle@finest ( )
        curRes = NormResidual@finest ( )
    }

    /* de-init code */
}
loop over $p$ {
    solve locally {
        $u[0, 0, 0] \Rightarrow A_u[0, 0, 0] \ast u[0, 0, 0] = rhs_u[0, 0, 0]$
        $+\ vf\_cellWidth\_y \ast vf\_cellWidth\_z \ast (p[-1, 0, 0] - p[0, 0, 0])$
        $u[1, 0, 0] \Rightarrow A_u[1, 0, 0] \ast u[1, 0, 0] = rhs_u[1, 0, 0]$
        $+\ vf\_cellWidth\_y \ast vf\_cellWidth\_z \ast (p[0, 0, 0] - p[1, 0, 0])$

        $v[0, 0, 0] \Rightarrow A_v[0, 0, 0] \ast v[0, 0, 0] = rhs_v[0, 0, 0]$
        $+\ vf\_cellWidth\_x \ast vf\_cellWidth\_z \ast (p[0, -1, 0] - p[0, 0, 0])$
        $v[0, 1, 0] \Rightarrow ...$

        $w[0, 0, 0] \Rightarrow A_w[0, 0, 0] \ast w[0, 0, 0] = rhs_w[0, 0, 0]$
        $+\ vf\_cellWidth\_x \ast vf\_cellWidth\_y \ast (p[0, 0, -1] - p[0, 0, 0])$
        $w[0, 0, 1] \Rightarrow ...$

        $p[0, 0, 0] \Rightarrow rhs_p ==$
        $\text{integrateOverEastFace} \ (u \ast rho) - \text{integrateOverWestFace} \ (u \ast rho) + \text{integrateOverNorthFace} \ (v \ast rho) - \text{integrateOverSouthFace} \ (v \ast rho) + \text{integrateOverTopFace} \ (w \ast rho) - \text{integrateOverBottomFace} \ (w \ast rho)$
    }
}
Applications: It is all about Grids
Showcase

- Work with Haase (U. Graz) and Vasco (U. Santiago de Chile)
- Simulation of non-Newtonian/non-isothermal fluids
- 3D FV discretization of staggered grids
- Recent extension towards FAS-FMG
- (Automatic) parallelization using OMP/MPI/CUDA

Vasco, Moraga, Haase. Parallel finite volume method simulation of three-dimensional fluid flow and convective heat transfer for viscoplastic non-Newtonian fluids
Kuckuk, Haase, Vasco, Köstler. Towards Generating Efficient Flow Solvers with the ExaStencils Approach
Scope: Towards Ocean Simulation

- Overall goal: simulation of ocean behavior
- Discretization using discontinuous Galerkin (DG) methods
- Performance and scalability

Image credits: NASA Landsat
Hybrid Grids: Unstructured Grids are slow(er)!
Generation of Block-Structured Grids

1. Rough (unstructured) tessellation of the computational domain
Generation of Block-Structured Grids

1. Rough (unstructured) tessellation of the computational domain
2. Uniform subdivision
Generation of Block-Structured Grids

1. Rough (unstructured) tessellation of the computational domain
2. Uniform subdivision
3. Adaptation of vertex positions
1. Rough (unstructured) tessellation of the computational domain
2. Uniform subdivision
3. Adaptation of vertex positions
4. Triangulation
Current State

- What is already possible?
  - Finite volume discretizations for simple model problems on regular, non-uniform patches (cell-centered, 2D)
  - Using the complete ExaStencils pipeline

- What is missing?
  - Suitable abstractions and language extensions for DG
  - Extension to prisms
  - (Specialized) communication routines for triangle and prism data
Towards Performance Engineering
Error Types

Real World

Model

Model Error

Discretization Error

Roundoff Error

Discrete Model

Discrete Solution

Algebraic Error
Algorithmic Performance Engineering

Real World

Model Choice

Model

Discretization Choice

Discrete Model

Platform Choice

Solver Choice

Discrete Solution
Optimization Problem

- For each choice you require a valid performance model that estimates the runtime for your settings!
- Therefore there is a need to automatize the creation of performance models
Performance Modeling

- We aim for an automatic derivation of performance models
- Prototype implementation (optimistic):

For each function

Estimate performance of each statement and sum up

*Kernel:* Roofline
- Count FLOPs & assume vectorization
- Div. by clock rate

*Max*
- Count memory accesses & assume blocking
- Div. by memory bandwidth

*Function call:* recursive estimation

*MPI/CUDA data transfer:* open (e.g. latency + BW)

- Ongoing work: extension to ECM
Runtimes 3D Variable Coefficients

**CPU** dual socket Xeon E5620, 2.4 GHz clock, 4 cores / 8 hw-threads


**L3 Cache** 12 MB smart cache

**Main Memory** 24 GB

**Interconnect** 2x QPI 5.86 GT/s
Towards Metaprogramming
What is the problem?

- Goal: Solve partial differential equation

\[
\Delta u = f \quad \text{in } \Omega \\
u = 0 \quad \text{on } \partial \Omega
\]

- After discretization one requires an efficient iterative solver for sparse systems

\[
Au_h = f_h
\]

- Multigrid solver has complexity \( O(N) \)
Metaprogramming

Mathematical representation

\[ \Delta u = f \]

Code representation

Field u,f;
Laplace(u) = f;

Tree representation

Transforms
Idea: Generate solver automatically

- Linear system of interest as $Ax = b$, where $A$ is the system matrix, $x$ is the unknown vector and $b$ is the right hand side vector.
- Denote the iteration matrix with $M$ and it is represented as a tree.
- The approximated solution in the $n+1$ iteration is obtained using the old solution and the iteration matrix using the relation $x_{n+1} = Mx_n$.
- A tree in any stage of the framework is a valid expression based on the defined matrix operations i.e. the nodes of the tree are selected such that the generated tree is consistent in terms of dimensions and the generated expressions are secured to be computable.
Genetic Programming Setup

- **Terminal Set**: It is composed of arbitrary symbolic matrices.
  - $A$, $b$, diagonal of $A$, inverse of the diagonal of $A$, $A$ minus its diagonal, lower triangular part of $A$, inverse of the lower triangular part of $A$ and upper triangular (1st diagonal) part of $A$.

- **Operation Set**:
  - The operation set consists of addition, subtraction and multiplication.

- **Fitness Evaluation**:
  - The symbolic matrix corresponding to the genetic expression of an individual is simplified using the symbolic Math Toolbox of MATLAB and then evaluated to get the numerical version of the iteration matrix.
  - Spectral radius $r$ is used to rank individuals, fitness function $f$ for $r < 1$ is defined $f(r) = 1/(1 - r^2)$ and for $r > 1$ as $f(r) = a_0 + a_1 r$. 
Examples of individuals of iteration matrix

**Mathematical representation**

\[ M_0 = \text{inv}(\text{diag}(A)) \ast b \]
\[ M_1 = A \ast b - \text{diag}(A) \]

**Tree representation**

Apply mutation and cross over
1D Example

\[ A = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix}, \quad b = \begin{bmatrix} 4 \\ 0 \\ -2 \end{bmatrix} \quad \text{and} \quad x = \begin{bmatrix} 2.5 \\ 1.0 \\ -0.5 \end{bmatrix} \]

\[ M_1 = \frac{a_{22}a_{33} - a_{11}a_{12}a_{21}a_{32} + a_{11}a_{13}a_{21}a_{32} - a_{11}a_{13}a_{22}a_{31}}{a_{11}a_{22}a_{33}}, \]
\[ a_{12}a_{33} - a_{13}a_{22} \\ a_{11}a_{22}a_{33} \]
\[ M_2 = \left[ -\frac{a_{23}a_{21}a_{22}^2 - a_{21}a_{22}a_{23}a_{32} + a_{21}a_{33} - a_{22}a_{23}a_{32}}{a_{11}a_{22}a_{33}}, \frac{a_{23}}{a_{22}a_{33}} \right], \quad M_3 = \left[ \frac{a_{21}a_{32} - a_{22}a_{21}}{a_{11}a_{22}a_{33}^2}, -\frac{a_{22}}{a_{22}a_{33}^2}, \frac{1}{a_{33}^2} \right] \]
2D Poisson Example

\[ A = \begin{bmatrix} 4 & -1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 4 & -1 & 0 & -1 & 0 & 0 & 0 & -1 \\ 0 & -1 & 4 & 0 & 0 & -1 & 0 & 0 & 0 \\ -1 & 0 & 0 & 4 & -1 & 0 & -1 & 0 & 0 \\ 0 & -1 & 0 & -1 & 4 & -1 & 0 & -1 & 0 \\ 0 & 0 & -1 & 0 & -1 & 4 & 0 & 0 & -1 \\ 0 & 0 & 0 & -1 & 0 & 0 & 4 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & -1 & 4 & -1 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & -1 & 4 \end{bmatrix} \]

\[ M_1 = [-6, 0, 0, 0, 0, 0, 0, 0, 0]e^{-2}, \ M_2 = [-9, -6, 0, 0, 0, 0, 0, 0, 0]e^{-2}, \ M_3 = [-3, -9, 0, 0, 0, 0, 0, 0, 0]e^{-2}, \ M_4 = [-9, 0, -6, 0, 0, 0, 0, 0, 0]e^{-2}, \ M_5 = [-5, -9, 0, -9, -6, 0, 0, 0, 0]e^{-2}, \ M_6 = [-2, -5, -9, -2, -9, -6, 0, 0, 0]e^{-2}, \ M_7 = [-3, 0, 0, -10, 0, 0, -6, 0, 0]e^{-2}, \ M_8 = [-2, -3, 0, -5, -10, 0, -10, -6, 0]e^{-2}, \ M_9 = [-1, -2, -3, -2, -5, -10, -3, -10, -6]e^{-2}. \]

Scalable iteration matrices?!
Comparsion of iteration matrices

Optimize pattern for parallelization!
Non-sparse and asymmetric system

$A_1 = [8, 2, 7, 2, 2], A_2 = [2, 7, 5, 3, 2], A_3 = [3, 4, 8, 3, 3], A_4 = [2, 1, 3, 8, 3], A_5 = [2, 3, 5, 1, 8]$ and the column vector $b = [105, 108, 100, 56, 82]$.

$M_1 = \begin{bmatrix} \frac{1}{a_{11}} & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{a_{22}} & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{a_{33}} & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{a_{44}} & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{a_{55}} \end{bmatrix}, M_2 = \begin{bmatrix} \frac{1}{a_{11}} & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{a_{22}} & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{a_{33}} & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{a_{44}} & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{a_{55}} \end{bmatrix}, M_3 = $ 

$M_4 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{a_{44}} & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{a_{55}} \end{bmatrix}, M_5 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{a_{33}} & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{a_{44}} & 0 \\ 0 & 0 & 0 & \frac{1}{a_{55}} & 0 \end{bmatrix}$

A simple preconditioner based on $\text{inv}(A)$!
Acknowledgements

- Funded by
  - Bundesministerium für Bildung und Forschung
  - KONWIHR. Bavarian project
  - DFG SPP 1648/1 – Software for Exascale computing
- Industry
- Supercomputing centers

http://www.exastencils.org/
Thank you for your Attention!

Questions?