A Common Automatic Code Generator for a Wide Range of Stencil Codes

H. Köstler

Workshop ExaStencils 2014, Dresden
A unique, tool-assisted, domain-specific co-design approach for the class of stencil codes

http://www.exastencils.org/
Problems in High Performance Computing

- **Hardware**: Modern HPC platforms are massively parallel
  - Intra-core, intra-node, and inter-node

- **Software**: CSE applications become more complex with increasing computational power
  - More complex models
  - Code development in interdisciplinary teams

- **Algorithm**: Class of different algorithms grows, many of them are just a general idea (like multigrid)
  - Components and parameters depend on grid, type of problem, …
High Performance Computing: Applications

- real-time imaging
  e.g. medicine
- Large-scale simulation
  e.g. multi-physics
Parallel Multigrid Solver Performance

- Tianhe-2 (estimated)
- BlueGene/Q
- BlueGene/P
- IBM System x iDataPlex
- SGI Altix 4700
- HP Proliant (Nvidia Tesla M2050)
- Hitachi SR8000-F1
- Cray T3D
- Nvidia GTX 480
- Intel iPSC/2
- CDC Cyber 205
- Caltech Mark II Hypercube

Year:
- 1975
- 1980
- 1985
- 1990
- 1995
- 2000
- 2005
- 2010
- 2015

Values:
- 1,00E+14
- 1,00E+13
- 1,00E+12
- 1,00E+11
- 1,00E+10
- 1,00E+09
- 1,00E+08
- 1,00E+07
- 1,00E+06
- 1,00E+05
- 1,00E+04

Unbeknown to s
Unbeknown

Proposed: Domain-driven Projects

Users from different application fields → Description of application in domain specific language → Domain expert

Mathematician → Automatic selection of algorithmic components

Software specialist → Code generation for specific application

Hardware specialist → Automatic tuning on specific hardware

Domain knowledge

PDE \{ \text{Operators::Laplacian(Data::solution) = Data::rhs} \}
Aspects

- Code generation for specific HPC applications
- Domain-specific language design
- Domain-specific knowledge representation and optimization
- Efficient Algorithms
- Parallelization
- Performance Tuning
DSL Scheme

Layer 1
- Computational Domain
- Continuous Model

Layer 2
- Discrete Domain
- Discrete Model

Layer 3
- Algorithmic Parameters and Components
- Application Settings

Layer 4
- Pseudo Code
## DSL Scope

<table>
<thead>
<tr>
<th></th>
<th>2D</th>
<th>3D</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dimension</td>
<td>2D</td>
<td>3D</td>
</tr>
<tr>
<td>Domain</td>
<td>UnitSquare</td>
<td>UnitCube</td>
</tr>
<tr>
<td>Solution</td>
<td>Scalar</td>
<td>Vector</td>
</tr>
<tr>
<td>Operator</td>
<td>Stencil</td>
<td>weak form</td>
</tr>
<tr>
<td>Boundary</td>
<td>Dirichlet</td>
<td>Neumann</td>
</tr>
<tr>
<td>Architecture</td>
<td>CPU</td>
<td>GPU</td>
</tr>
<tr>
<td>Languages</td>
<td>C++</td>
<td>Scala</td>
</tr>
<tr>
<td>Parallelization</td>
<td>MPI</td>
<td>OpenMP</td>
</tr>
<tr>
<td>Discrete Domain</td>
<td>regular</td>
<td></td>
</tr>
<tr>
<td>Grid</td>
<td>nodebased</td>
<td>cellbased</td>
</tr>
<tr>
<td>Operator Discretization</td>
<td>FD</td>
<td>FE</td>
</tr>
<tr>
<td>Data Types</td>
<td>float</td>
<td>double</td>
</tr>
<tr>
<td>Discretization Accuracy</td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>
High Dynamic Range compression

**DSL EXAMPLE**
What is Multigrid?

- 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 \( O(N) \) in number of unknowns \( N \)
Problem Description (continuous)

**Domain** \( \omega = [0,1] \times [0,2] \)

\[
f : \omega \rightarrow \mathbb{R}^{1}
\]

\[
u : \omega \rightarrow \mathbb{R}^{1}
\]

\[
\text{Laplacian : ( } \omega \rightarrow \mathbb{R}^{1} \text{ ) } \rightarrow ( \omega \rightarrow \mathbb{R}^{1} \text{ )}
\]

\[
\text{Laplacian} = dx^2 + dy^2
\]

\[
\text{pde : } \text{Laplacian} [ u ] = f \text{ in } \omega
\]

\[
\text{bc : } u = 0 \text{ in } \partial \omega
\]
Generation of discrete problem

- Domain-specific knowledge
  - Discretization methods FD, FV, FE
  - Types of operators supported result in sparse matrices
- Domain-specific optimization chooses type of discretization and e.g. concrete data types
- Description is parsed, an abstract syntax tree is constructed and then transformed into a discrete representation of the problem
- Code generation framework is implemented in Scala language
Fragments \( f_1 = \text{Regular}_\text{Square} \)

**Discrete_Domain** \( \omega \) levels 10 {

\[
\begin{align*}
\text{xsize} \; [0] &= 1024 \\
\text{ysize} \; [0] &= 1024 \\
\text{xsize} \; [l+1] &= \text{xsize} \; [l] / 2 \\
\text{ysize} \; [l+1] &= \text{ysize} \; [l] / 2 \\
\end{align*}
\]

Field<Double>@nodes f
Field<Double>@nodes u
StencilMatrix<Double,FD,2>@nodes Laplacian
Algorithm 1 Recursive V-cycle: $u_h^{(k+1)} = V_h(u_h^{(k)}, A^h, f^h, v_1, v_2)$

1: if coarsest level then
2: solve $A^h u^h = f^h$ by a (parallel) direct solver or by CG iterations
3: else
4: $\tilde{u}_h^{(k)} = L_h^{v_1}(u_h^{(k)}, A^h, f^h)$ \{presmoothing\}
5: $r^h = f^h - A^h \tilde{u}_h^{(k)}$ \{compute residual\}
6: $r^H = Rr^h$ \{restrict residual\}
7: $e^H = V_H(0, A^H, r^H, v_1, v_2)$ \{recursion\}
8: $\tilde{u}_h^{(k)} = \tilde{u}_h^{(k)} + P e^H$ \{prolongate error and do coarse grid correction\}
9: $u_h^{(k+1)} = L_h^{v_2}(\tilde{u}_h^{(k)}, A^h, f^h)$ \{postsmoothing\}
10: end if
VECTOR res SIZE u
MATRIX N = inverse(diag(Laplacian [l]) -lower(Laplacian [l]))
MATRIX M = I – inverse(N)*Laplacian [l]
RESTRMATRIX R of u = 2  // order
SET s = {[0,0]} {[+=1, +=1]} // iteration: {start}, {increment}
SET sred = {[0,0] + [1,1]} {[+=2, +=2]}
SET sblack = {[0,1] + [1,0]} {[+=2, +=2]}

ITERATION rbgs : smred [l] ~ smblack [l]
ITERATION prolong : \( u[l] = u[l] + ((\text{transp}(R)) \cdot u[l+1]) \) order s
ITERATION residual : \( \text{res}[l] = f[l] - (\text{Laplacian}[l] \cdot u[l]) \) order s
ITERATION restrict : \( f[l+1] = (R \cdot \text{res}[l]) \) order s
ITERATION cycle : \( (\text{rbgs}[l])^8 \mid l == 7 \)

\( (((((\text{rbgs}[l])^1) \sim \text{residual}[l]) \sim (\text{restrict}[l] \sim ((\text{cycle}[l+1])^1))) \sim (\text{prolong}[l] \sim ((\text{rbgs}[l])^1)) \mid l != 7 \)
mgcomponents {
    solver = multigrid
}

mgparameter {
    iters = 10
}
Program Specification Structure (Layer 4)

Definitions
- Fields: platform spec. inf., e.g. memory layout
- Stencils: C-like indices

Functions with platform spec. annotations
- generated from Layers 1-3
- helper functions
- external libraries

Application
def \textit{gpu restrict\_at\_current\_level} () : \textbf{Unit}
{
    \textbf{loop innerpoints order s}
    f [1] = R \times res [0]
    \textbf{next }
}
def cpu_cycle ( lev: Int ) : Unit
{
    if (lev == 9) {
        repeat up ncoarse
            rbgs ( lev)
        next
    } else {
        repeat up npvae
            rbgs (lev)
        next
        residual (lev)
        restrict (lev+1, f [(lev+1)], Res[lev])
        set (lev+1, solution [lev+1], 0)
        cycle (lev+1)
        prolong (lev, solution[lev], solution [lev+1])
        repeat up npost
            rbgs (lev)
        next
    } }
def cpu Application( ) : Unit
{
    decl res0 : Double = L2Residual ( 0 )
    decl res : Double = res0
    decl resold : Double = 0
    print ( 'startingres' res0 )
    repeat up 5
        resold = res
        cycle_0 ()
        res = L2Residual ( 0 )
        print ( 'Residual:' res 'residual reduction:' (res0/res) )
    next
}
Hardware $cpu$ {
    bandwidth = 40
    peak = 30
    cores = 4
}

Node $n$ {
    sockets = 2
}
Idea:

Use nonlinear anisotropic diffusion process to denoise the image $u^0$ in the domain $\Omega$, i.e. solve the time-dependent PDE

\[
\text{div}(g\nabla u) = \frac{\partial u}{\partial t} \quad \text{in } \Omega \times T
\]

\[
\langle g\nabla u, n \rangle = 0 \quad \text{on } \partial\Omega \times T
\]

\[
u(x,0) = u^0(x) \quad \text{in } \Omega
\]
## Runtime Results for Different Problems

Sizes 4096x4096 resp. 256x256x256

<table>
<thead>
<tr>
<th></th>
<th>cpu</th>
<th>gpu</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>Double</td>
<td>Float</td>
</tr>
<tr>
<td><strong>Laplace</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2D</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Jacobi</td>
<td>546</td>
<td>1.17</td>
</tr>
<tr>
<td>GaussSeidel</td>
<td>453</td>
<td>1.16</td>
</tr>
<tr>
<td>3D</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Jacobi</td>
<td>608</td>
<td>1.08</td>
</tr>
<tr>
<td>GaussSeidel</td>
<td>608</td>
<td>1.08</td>
</tr>
<tr>
<td><strong>Complex Diffusion</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2D</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Jacobi</td>
<td>9235</td>
<td>1.02</td>
</tr>
<tr>
<td>GaussSeidel</td>
<td>7504</td>
<td>1.11</td>
</tr>
<tr>
<td>3D</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Jacobi</td>
<td>8799</td>
<td>1.15</td>
</tr>
<tr>
<td>GaussSeidel</td>
<td>9048</td>
<td>1.80</td>
</tr>
</tbody>
</table>

**ms** speedup, speedup, speedup, speedup, speedup, speedup
Future Work

- HPC applications
  - Geophysics
  - Quantum Chemistry

- Biggest Issues
  - Performance Optimization
  - Feature Selection via DSL
Acknowledgements

• Funded by
  • Bundesministerium für Bildung und Forschung

• KONWIHR. Bavarian project

• DFG SPP 1648/1 – Software for Exascale Computing

http://www.exastencils.org/
THANK YOU!