Towards Generating Flow Solvers with the ExaStencils Approach

Sebastian Kuckuk
FAU Erlangen-Nürnberg
14.07.2015, University of Graz
Outline
Outline

- Scope and Motivation
- Project ExaStencils
- Work in Graz
- Conclusion & Future Work
Scope and Motivation
PDEs

- Goal: Solve a partial differential equation approximately by solving a discretized form

\[ \Delta u = f \quad \text{in } \Omega \]
\[ u = 0 \quad \text{in } \partial \Omega \]
\[ A u_h = f_h \]

Optical Flow (LIC Visualization)

Particles in Flow
Multigrid

- Smoothing of High Frequency Errors
- Coarsened Representation of Low Frequency Errors
Multigrid V-Cycle

Smoothing

Restriction

Prolongation & Correction

Coarse Grid Solving
Computational Grids

- Regular grids

- Block-Structured grids -> deferred for now
Main Focus: HPC

- Highly optimized and highly scalable geometric multigrid solvers
- ‘Traditional’ supercomputers like JUQUEEN and SuperMUC
- Alternative architectures like Piz Daint and Mont Blanc
Problems to Deal With

- Composition of optimal solvers requires choosing
  - MG components: Cycle Type? Which smoother(s)? Which coarse grid solver? Which inter-Grid operators?
  - MG parameters: Relaxation? Number of smoothing steps? Other component dependent parameters?
  - …
Problems to Deal With

- Composition of optimal solvers requires choosing
  - MG components: …
  - MG parameters: …
  - Applied Optimizations: …

- But the choice relies heavily on various influences
  - **Hardware**: CPU, GPU or both? Number of nodes, sockets and cores? Cache characteristics? Network characteristics?
  - **Software**: MPI, OpenMP or both? CUDA or OpenCL? Which versions?
  - **Problem description**: Which PDE? Which boundary conditions?
  - **Discretization**: Finite Differences, Finite Elements or Finite Volumes?
  - **Domain**: Uniform or block-structured? How to partition?
  - …
Possible Solutions

● What do we want?
  ● Efficient and robust multi-grid solvers
  ● Performance portability
  ● Easy to adapt to new hardware
  ● Easy to extend
  ● …

● Solutions?
  ● Optimizing by hand?
  ● Extensive Libraries?
  ● Auto-Tuning?
  ● Code generation?
Project ExaStencils
Project ExaStencils

- Sebastian Kuckuk
- Harald Köstler
- Ulrich Rüde
- Alexander Grebhahn
- Sven Apel
- Christian Schmitt
- Frank Hannig
- Jürgen Teich
- Hannah Rittich
- Matthias Bolten
- Stefan Kronawitter
- Armin Größlinger
- Christian Lengauer
ExaStencils Workflow

End-user

Domain expert

Mathematician

Software specialist

Hardware expert

DSL program

Discretization and algorithm selection

Selection of algorithmic components & parameter settings

Polyhedral optimization

Code generation

Tuning towards target hardware

ExaStencils Compiler

Exascale C++
Our Layered DSL ExaSlang

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

abstract problem formulation

concrete solver implementation

Target Platform Description
ExaSlang 4 Example

Function JacSmother@((coarsest + 1) to finest) ( ) : Unit {
    communicate ghost of Solution[active]@current
    loop over Solution@current {
        Solution[nextSlot]@current = Solution[active]@current +
            (( ( 1.0 / diag ( Laplace@current ) ) * OMEGA ) * (RHS@current
                - Laplace@current * Solution[active]@current )))
    }
    advance Solution@current
}

• Concepts:
  • Leveled functions, fields and stencils
  • Intuitive stencils-field operations
  • Slotting mechanism
  • Communication management
The ExaStencils Code Generation Framework

- Implemented workflow
Example Transformations

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

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

s += Transformation("eval adds", {
    case AdditionExpression (l : IntConstant, r : IntConstant) => IntConstant (l.value + r.value)
})

s. apply // execute transformations sequentially
```
Low-Level Optimizations in the IR

- Address pre-calculation
- Arithmetic simplifications
- Vectorization (SSE3, AVX, AVX2, QPX, NEON)
- Loop unrolling
  - Also enables further optimizations, e.g. eliminating modulo accesses
- Polyhedron model extraction and optimization
  - Sophisticated dependency analysis
  - Advanced dead code elimination
  - Increased memory coalescence and/or parallelism
  - Automatic tiling
  - Code optimization by elimination of conditionals (-> RBGS)
Polyhedron Model Example

```
for (int i = 1; i <= n; ++i)
    for (int j = 1; j <= n-i+1; ++j)
        a[i][j] = a[i-1][j] + a[i][j-1];
```

```
for (int t = 1; t <= n; ++t)
    # pragma omp parallel for
    for (int p = 1; p <= t; ++p)
        a[t-p+1][p] = ...;
```

Iteration domain

```
1 ≤ i ≤ n
1 ≤ j ≤ n−i+1
```

Dependences

```
(i,j) → (i+1,j)
(i,j) → (i,j+1)
```

```
(t,p) → (t+1,p)
(t,p) → (t+1,p+1)
```
Domain Partitioning

- Easy for regular domains

Each **domain** consists of one or more **blocks**

Each **block** consists of one or more **fragments**

Each **fragment** consists of several **data points / cells**

- More complicated for HHG
Domain Partitioning vs Parallelization Interface

- Domain partition maps directly to different parallelization interfaces, e.g. MPI and OMP:
  - Each **block** corresponds to one **MPI** rank
  - Each **fragment** corresponds to one **OMP** rank
  - Hybrid **MPI/OMP** corresponds to multiple **blocks** and multiple **fragments** per **block**
  - Alternatively: only one **fragment** per **block** and direct parallelization of kernels with **OMP**

- Easy to map to different interfaces, e.g.
  - PGAS
  - MPI and PGAS
  - MPI and CUDA
Parallelization

- Communication statements are added automatically when transforming Layer 3 to Layer 4 where they may be reviewed or adapted

```c
/* communicates all applicable layers */
communicate Solution@current
/* communicates only ghost layers */
communicate ghost of Solution[active]@current
/* communicates duplicate and first two ghost layers */
communicate dup, ghost[0, 1] of Solution[active]@current
/* asynchronous communicate */
begin communicate Residual@current
//...
finish communicating Residual@current
```
Performance Prediction

● Combination of two powerful techniques: LFA and SPL

● Local Fourier Analysis
  ● Allows effective prediction of convergence rates
  ● And the required number of cycles

● Software Product Lines
  ● Build on variance models
  ● Quantify feature interactions
  ● Predict the runtime per multigrid cycle
Work in Graz
Overall Goal

“Parallel finite volume method simulation of three-dimensional fluid flow and convective heat transfer for viscoplastic non-Newtonian fluids”, D. A. Vasco, N. O. Moraga and G. Haase

- FORTRAN90 Code
- SIMPLE Algorithm
  - Solve linearized velocity components separately
  - Compute and apply pressure correction
- TDMA solvers
 Tasks

- Reproduce results by exchanging old code parts with generated counterparts
- Add geometric multigrid solver(s)
- Introduce distributed memory parallelization
- Extend solver components
- Analyze performance
Preliminary Work @LSS

- Extension of our code generation framework
  - Fortran interfaces
  - Staggered grids
- Implementation of simple test case
  - 2D laminar flow
  - Staggered grid
  - FV discretization
  - SIMPLE algorithm
  - Fully OMP parallel
  - MPI extension is trivial
Current Work on Reference Code

- Currently, only the pressure correction step is fully represented in our DSL, i.e.
  - Stencil compilation
  - Solving of the arising system
  - Correction of physical quantities

- Work on the velocity component updates are underway
  - Parts of the stencils and the RHS are computed in the DSL
  - LSEs are solved with generated (RBGS) solvers
Open Questions/ Issues

- Grid spacing
  - Specification
  - Storage
  - Interpolation/ use as stencil coefficients
- (MPI) parallelization vs interfacing
- Transition to more sophisticated algorithms
  - away from SIMPLE
  - Requires different smoothers, inter-grid operators, stencils, etc.
Conclusion
Conclusion

● ExaStencils framework
  ● Code generation from abstract DSL representations
  ● Application of automatic low-level optimizations
  ● Parallelization and data distribution
  ● Sophisticated performance prediction

● Non-Newtonian fluids as target application
Future Work
Future Work

- Work on ExaSlang Layer 1 – 3
- Extension of the framework, esp. multi-GPU
- Implementation of NNF
- Distributed memory parallelization
- Extension of solver components
Thank you for your Attention!

Questions?