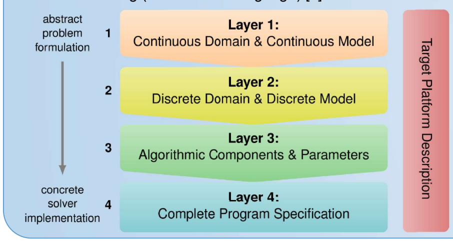
Whole Program Generation for Complex Fluid Flow Solvers

Sebastian Kuckuk, Harald Köstler
Friedrich-Alexander-Universität Erlangen-Nürnberg, Department of Computer Science 10 (System Simulation)

Project ExaStencils

Generation of efficient, robust and exa-scalable geometric multigrid solvers [1] from abstract inputs in our multi-layered DSL ExaSlang (ExaStencils Language) [2]



Stefan Kronawitter
Armin Größlinger
Christian Lengauer

Alexander Grebhahn
Sven Apel

SPPEXA

ExaStencils

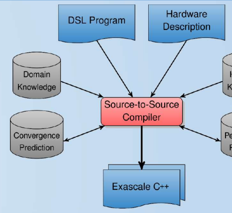
Sebastian Kuckuk
Georg Altmann
Harald Köstler
Ulrich Rüde

Hannah Rittich
Lisa Claus
Matthias Bolten

Christian Schmitt
Frank Hannig
Jürgen Teich

Shigeru Chiba

FAU · UNIVERSITÄT PASSAU · BERGISCHE UNIVERSITÄT WUPPERTAL · UNIKASSEL VERSITÄT · 東京大学 THE UNIVERSITY OF TOKYO



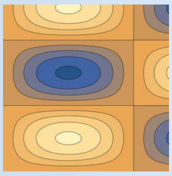
- Modular code generation framework based on fine-grained transformations
- Automatic parallelization via MPI/OMP [3] and/or CUDA
- Automatic low-level optimizations (polyhedral loop transformations, vectorization, CSE, APC, etc.) [4]
- Interfacing with performance prediction models

Problem Specification

We consider the following problems:

(1) a steady-state Stokes flow with the (known) solution:

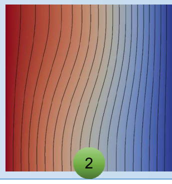
$$\begin{aligned} u &= -4\cos(4z) \\ v &= 8\cos(8x) \\ w &= -2\cos(2y) \\ p &= \sin(4x)\sin(8y)\sin(2z) \end{aligned}$$

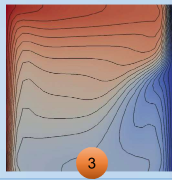


$$\begin{aligned} -\Delta u + \frac{\partial u}{\partial x} &= f_u \\ -\Delta v + \frac{\partial v}{\partial y} &= f_v \\ -\Delta w + \frac{\partial w}{\partial z} &= f_z \\ \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z} &= f_p \end{aligned}$$

(2) a non-isothermal/ non-Newtonian fluid flow following the specifications in [6]
we choose a natural convection test-case (a flow is induced by heating one wall and cooling another)

(3) a non-isothermal/ Newtonian fluid flow, equivalent to (2) disregarding the non-Newtonian properties





Discretization

```
Stencil Au@all { /* stencil coefficients are not stored */
  [ 1,  0,  0] => integrateOverXStaggeredEastFace   ( -1.0 )
                  / vf_cellWidth_x@[1, 0, 0]
  [-1,  0,  0] => integrateOverXStaggeredWestFace   ( -1.0 )
                  / vf_cellWidth_x@[0, 0, 0]
  [ 0,  1,  0] => integrateOverXStaggeredNorthFace  ( -1.0 )
                  / vf_stagCVWidth_y@[0, 1, 0]
  [ 0, -1,  0] => integrateOverXStaggeredSouthFace  ( -1.0 )
                  / vf_stagCVWidth_y@[0, 0, 0]
  [ 0,  0,  1] => integrateOverXStaggeredTopFace    ( -1.0 )
                  / vf_stagCVWidth_z@[0, 0, 1]
  [ 0,  0, -1] => integrateOverXStaggeredWestBottom ( -1.0 )
                  / vf_stagCVWidth_z@[0, 0, 0]

  [ 0,  0,  0] => /* negative sum of other entries */
}
```

(1)

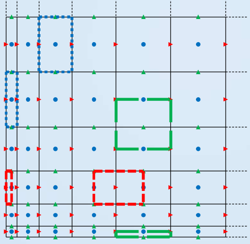
```
Stencil Cu@all { /* dx from xStag to Cell */
  [ 0,  0,  0] => -vf_cellWidth_y * vf_cellWidth_z
  [ 1,  0,  0] =>  vf_cellWidth_y * vf_cellWidth_z
}
```

(1) (2) (3)

To obtain the discretized version

$$\begin{bmatrix} A_u & 0 & 0 & B_u \\ 0 & A_v & 0 & B_v \\ 0 & 0 & A_w & B_w \\ C_u & C_v & C_w & 0 \end{bmatrix} \begin{bmatrix} u \\ v \\ w \\ p \end{bmatrix} = \begin{bmatrix} rhs_u \\ rhs_v \\ rhs_w \\ rhs_p \end{bmatrix}$$

We apply a finite volume discretization on non-uniform, axis-aligned, staggered grids

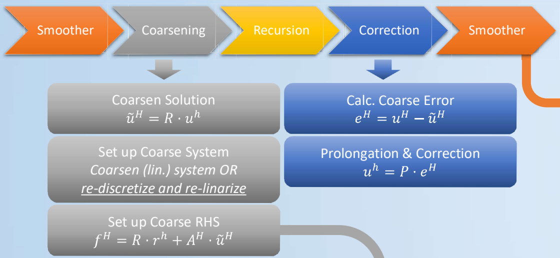


```
Function AssembleAu@all { /* stencil coefficients are stored in a dedicated field */
  loop over Au {
    Val flow_e = integrateOverXStaggeredEastFace ( u * rho )
    Val flow_w = integrateOverXStaggeredWestFace ( u * rho )
    Val diff_e = integrateOverXStaggeredEastFace ( vis ) / vf_cellWidth_x@[ 0,  0,  0]
    Val diff_w = integrateOverXStaggeredWestFace ( vis ) / vf_cellWidth_x@[-1,  0,  0]

    Au:[ 1,  0,  0] = -calc_diflow ( flow_e, diff_e ) + max ( 0.0, -flow_e )
    Au:[-1,  0,  0] = -calc_diflow ( flow_w, diff_w ) + max ( 0.0,  flow_w )

    Val flow_n = integrateOverXStaggeredNorthFace ( v * rho )
    Val flow_s = integrateOverXStaggeredSouthFace ( v * rho )
    Val diff_n = ( integrateOverXStaggeredNorthFace (
                   evalAtXStaggeredNorthFace ( vis, "harmonicMean" ) )
                   / vf_stagCVWidth_y@[ 0,  1,  0] )
    Val diff_s = ( integrateOverXStaggeredSouthFace (
                   evalAtXStaggeredSouthFace ( vis, "harmonicMean" ) )
                   / vf_stagCVWidth_y@[ 0,  0,  0] )

    Au:[ 0,  1,  0] = -calc_diflow ( flow_n, diff_n ) + max ( 0.0, -flow_n )
    Au:[ 0, -1,  0] = -calc_diflow ( flow_s, diff_s ) + max ( 0.0,  flow_s )

    /* other components */
  }}
```

(2) (3)

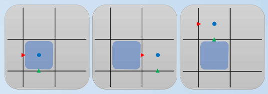
Solver

- A standard geometric multigrid is sufficient for (1)
- A full approximation scheme (FAS) multigrid is required for (2) and (3)
- In all cases, (colored) Vanka smoothers can be employed




```
uApprox@coarser = RestrictionFaceX * u
/* other components */

res_u = rhs_u - Au * u - Bu * p
/* other components */

rhs_u@coarser = ( RestrictionFaceXInteg * res_u
                + Au@coarser * uApprox@coarser
                + Bu@coarser * pApprox@coarser )
/* other components */
```

(2) (3)




```
color with {
  0 == ( i0 + i1 + i2 ) % 2,
  1 == ( i0 + i1 + i2 ) % 2,

  loop over p {
    solve locally relax 0.8 {
      u@[0, 0, 0] => Au@[0, 0, 0] * u@[0, 0, 0] + Bu@[0, 0, 0] * p@[0, 0, 0] == rhs_u@[0, 0, 0]
      u@[1, 0, 0] => Au@[1, 0, 0] * u@[1, 0, 0] + Bu@[1, 0, 0] * p@[1, 0, 0] == rhs_u@[1, 0, 0]

      v@[0, 0, 0] => Av@[0, 0, 0] * v@[0, 0, 0] + Bv@[0, 0, 0] * p@[0, 0, 0] == rhs_v@[0, 0, 0]
      v@[0, 1, 0] => Av@[0, 1, 0] * v@[0, 1, 0] + Bv@[0, 1, 0] * p@[0, 1, 0] == rhs_v@[0, 1, 0]

      /* similar for w */

      p => Cu * u + Cv * v + Cw * w == rhs_p
}}}
```

(1) (2) (3)

Mapping to code

- Preprocessing – apply coloring via duplication
- Set up the (local) LSE
  - Preprocessing – unfold nested expressions
  - Identify unknowns and reorder equations
  - Apply simplifications
- Check for suitability to apply **Schur complement** to accelerate local solve
- Incorporate (inner) boundaries
- Generate code to write back the results

$$\begin{bmatrix} A_{11} & 0 & 0 & B_1 \\ 0 & A_{22} & 0 & B_2 \\ 0 & 0 & A_{33} & B_3 \\ C_1 & C_2 & C_3 & D \end{bmatrix} \begin{bmatrix} U_1 \\ U_2 \\ U_3 \\ V \end{bmatrix} = \begin{bmatrix} F_1 \\ F_2 \\ F_3 \\ G \end{bmatrix}$$

$$S = D - (C_1 A_{11}^{-1} B_1 + C_2 A_{22}^{-1} B_2 + C_3 A_{33}^{-1} B_3)$$

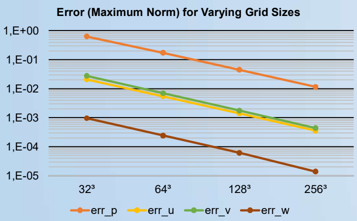
$$\tilde{G} = G - (C_1 A_{11}^{-1} F_1 + C_2 A_{22}^{-1} F_2 + C_3 A_{33}^{-1} F_3)$$

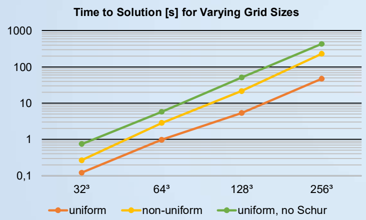
$$V = S^{-1} \tilde{G}$$

$$\begin{aligned} U_1 &= A_{11}^{-1}(F_1 - B_1 V) \\ U_2 &= A_{22}^{-1}(F_2 - B_2 V) \\ U_3 &= A_{33}^{-1}(F_3 - B_3 V) \end{aligned}$$

Results

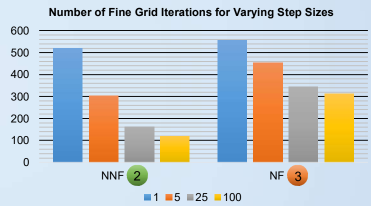
Solves (1) for the steady state using v(4,4)-cycles and a RB Vanka smoother

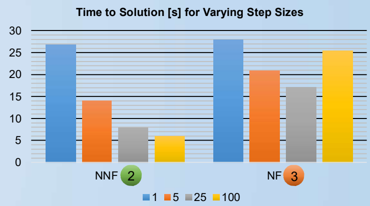




Solves (2) (3) for the first 100s on 32³ cells using

- FMG for the first time step
- RB Vanka as smoother for u,v,w,p
- RB GS as smoother for the temperature
- truncated v((2,5),(2,5))-cycles





Evaluation platform: single Intel Xeon E5-2623 v3 socket with 4 cores @3.0 GHz; no optimizations applied; automatic (OpenMP) parallelization using 4 threads

References

[1] Lengauer C, Apel S, Bolten M, Größlinger A, Hannig F, Köstler H, Rüde U, Teich J, Grebhahn A, Kronawitter S, Kuckuk S, Rittich H, Schmitt C. *ExaStencils: Advanced stencil-code engineering*. Euro-Par 2014: Parallel Processing Workshops, Lecture Notes in Computer Science, vol. 8806, Springer, 2014; 553–564.
[2] Schmitt C, Kuckuk S, Hannig F, Köstler H, Teich J. *ExaSlang: A domain-specific language for highly scalable multigrid solvers*. Proc. of the Fourth International Workshop on Domain-Specific Languages and High-Level Frameworks for High Performance Computing (WOLFHPC), IEEE Computer Society, 2014; 42–51.
[3] Kuckuk S, Köstler H. *Automatic Generation of Massively Parallel Codes from ExaSlang*. Computation, 4(3):Article 27, 2016. Special Issue on High Performance Computing (HPC) Software Design.
[4] Kronawitter S, Lengauer C. *Optimizations applied by the ExaStencils code generator*. Technical Report MIP-1502, Faculty of Informatics and Mathematics, University of Passau 2015.
[5] Schmitt C, Kuckuk S, Hannig F, Teich J, Köstler H, Rüde U, and Lengauer C. *Systems of Partial Differential Equations in ExaSlang*. Software for Exascale Computing – SPPEXA 2013-2015, volume 113 of Lecture Notes in Computational Science and Engineering, Springer, 2016; 47–67
[6] Kuckuk S, Haase G, Vasco DA. *Towards generating efficient flow solvers with the ExaStencils Approach*. Concurrency and Computation: Practice and Experience, 2017. Special Issue on Advanced Stencil-Code Engineering; early view.