Performance Engineering in the ExaStencils project

Harald Köstler
FAU Erlangen-Nürnberg
23.06.2016, ISC Frankfurt

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG
TECHNISCHE FAKULTÄT

Introduction

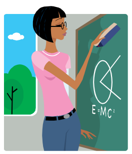
Application-driven Projects

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG
TECHNISCHE FAKULTÄT

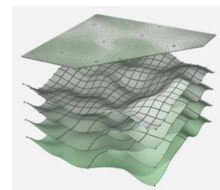
User from application field ⇨ Description of application

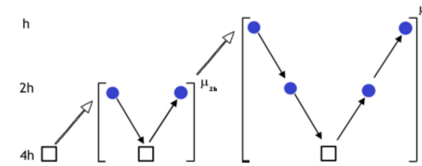
$$\phi^{(k)}_{\mathrm{MGM}}(\mathbf{x}_k, \mathbf{b}_k) = (\phi^{(k)}_{\mathrm{S}})^{\nu_2}((\phi^{(k)}_{\mathrm{S}})^{\nu_1}(\mathbf{x}_k, \mathbf{b}_k)$$
$$+ P_k((\phi^{(k-1)}_{\mathrm{MGM}})^{\gamma}(\mathbf{0}, R_k(\mathbf{b}_k - A_k\mathbf{x}_k))), \mathbf{b}_k)$$

⇩



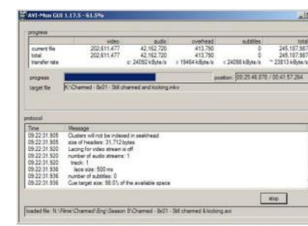
Mathematician ⇨ Solution method





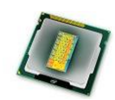
⇩

```
void GaussSeidel_rb(int lev, Array <double> *Sol, Array<double> *RHS) {
  int offset;

  #pragma omp parallel for private(offset)
  for (int i=1; i<Sol[lev].nrows()-1; i++) {
    offset = (i % 2 == 0 ? 2 : 1);
    for (int j=offset; j<Sol[lev].ncols()-1; j+=2) {
      Sol[lev] (i, j) = double(0.25)*(RHS[lev] (i, j) + Sol[lev] (i+1, j)
          + Sol[lev] (i-1, j) + Sol[lev] (i, j+1) + Sol[lev] (i, j-1));
    }
  }

  #pragma omp parallel for private(offset)
  for ( int i=1; i<Sol[lev].nrows()-1; i++ ) {
    offset = (i % 2 == 0 ? 1 : 2);
    for (int j=offset; j<Sol[lev].ncols()-1; j+=2) {
      Sol[lev] (i, j) = double(0.25)*(RHS[lev] (i, j) + Sol[lev] (i+1, j)
          + Sol[lev] (i-1, j) + Sol[lev] (i, j+1) + Sol[lev] (i, j-1));
    }
  }
}
```



Software specialist ⇨ Parallel implementation and framework

⇩



Hardware specialist ⇨ Efficient implementation on specific hardware

Programming Approaches in CSE

- One problem – one code
  - Everything is implemented from scratch or one uses common libraries
  - Can be easily specialized and therefore optimized
  - Good to test new algorithms and implementations
- One library – several problems
  - Higher maintenance and user support effort
  - Hard to fit all users needs and achieve optimal performance
  - A whole community can benefit from it
  - Standard for production codes
- One language – (hopefully) most problems
  - Design requires very high effort
  - Problem-specific optimizations possible
  - Current research

The ExaStencils Project

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

TECHNISCHE FAKULTÄT

Project ExaStencils is funded by the German Research Foundation (DFG) as part of the Priority Program 1648 (Software for Exascale Computing) -> http://www.exastencils.org

- Sebastian Kuckuk
- Harald Köstler
- Ulrich Rüde

SPPEXA

ExaStencils

- Alexander Grebhahn
- Sven Apel

Software Product-Line Group

- Christian Schmitt
- Frank Hannig
- Jürgen Teich

- Hannah Rittich
- Matthias Bolten

- Stefan Kronawitter
- Armin Größlinger
- Christian Lengauer

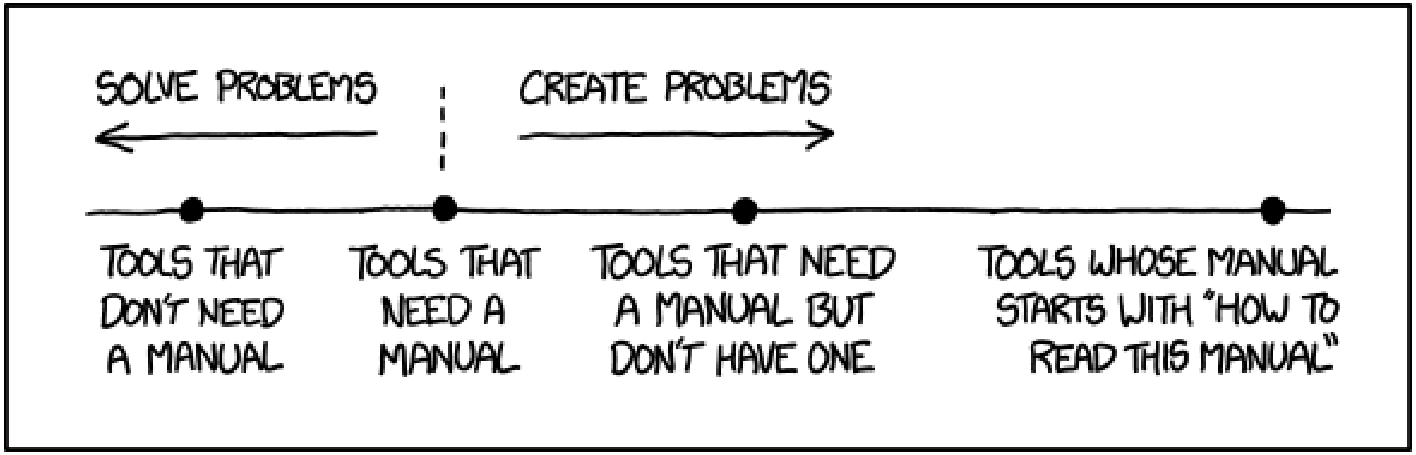
FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG

BERGISCHE UNIVERSITÄT WUPPERTAL

UNIVERSITÄT PASSAU

Domain-specific Languages

- **It's all about simplicity!**



XKCDManuals

Two Approaches to Create DSLs

FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG
TECHNISCHE FAKULTÄT

Internal / embedded domain-specific languages (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

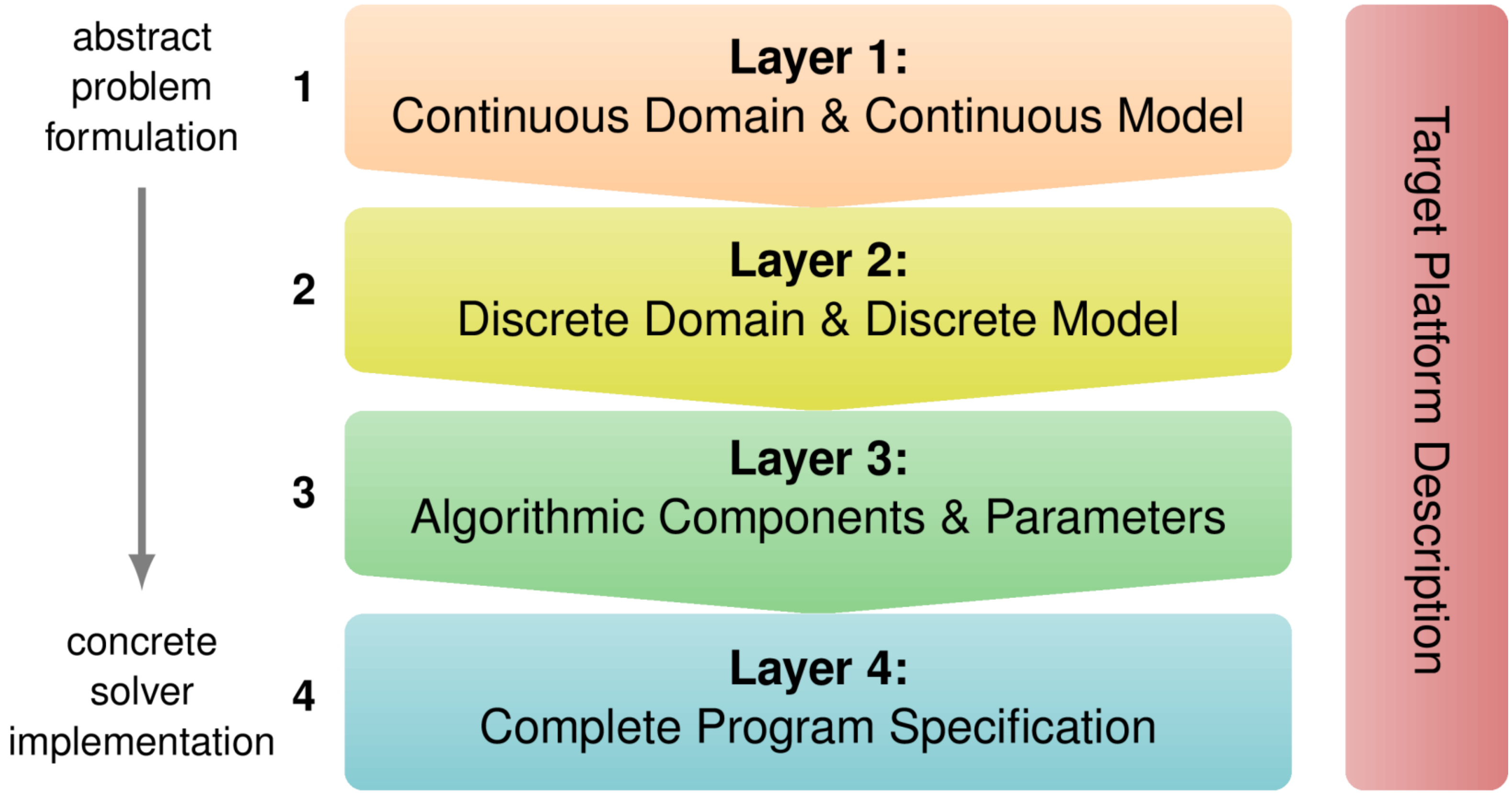
- Completely newly defined programming language
- More flexible and expressive than an internal DSLs
- Syntax and semantics defined freely, but often related to existing languages
- Higher design effort, but supporting tools exist
- Potential to create a powerful semantic model as intermediate representation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

ExaSlang

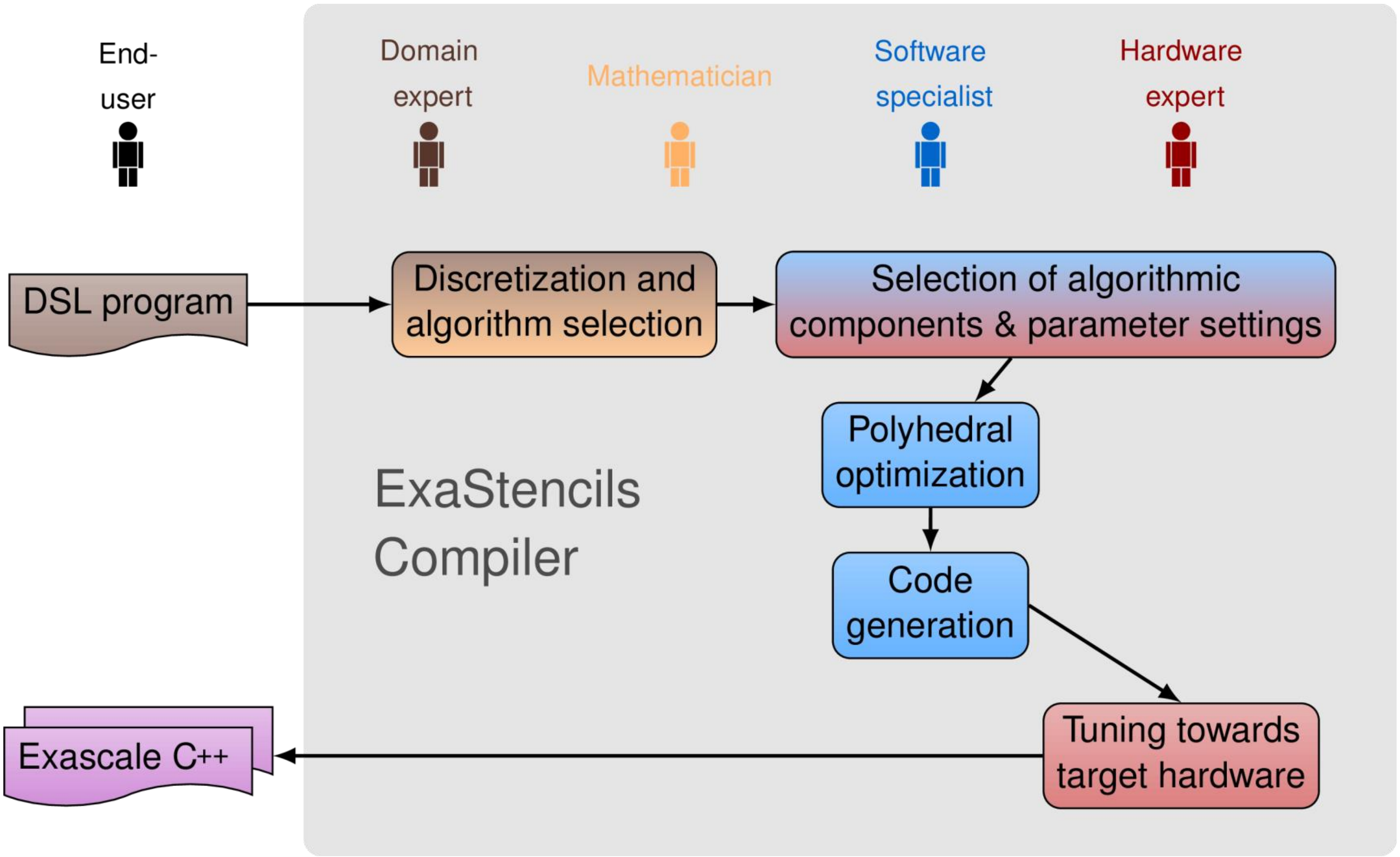
- **ExaS**tencils **lang**uage
- Abstract description for generation of massively parallel geometric multigrid solvers
- Multi-layered structure → hierarchy of DSLs
- Top-down approach: from abstract to concrete
- Very few mandatory specifications at one layer
  → room for decisions at lower layers based on domain knowledge
- External domain-specific language
  - better reflection of extensive ExaStencils approach
  - enables greater flexibility of different layers
  - eases tailoring of DSL layers to users
  - enables code generation for large variety of target platforms
- Parsing and code transformation framework implemented in Scala[1]

[1] **SKKHT14iccsa**

Our Layered DSL ExaSlang



ExaStencils Workflow



Example Transformations


```
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
```

Domain: Multigrid Algorithms

FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG
TECHNISCHE FAKULTÄT

Goal: solve $A^h u^h = f^h$ using a hierarchy of grids

Relax on $A^h u^h = f^h$

Residual $r^h = f^h - A^h u^h$

Restrict $r^H = I_h^H r^h$

Correct $u^h \leftarrow u^h + e^h$

Interpolate $e^h = I_H^h e^H$

Multigrid uses coarse grids
to accomplish the inevitable global data
exchange in the most efficient way possible

...

...

Low-level Optimization

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

TECHNISCHE FAKULTÄT

Low-Level Optimizations in the IR

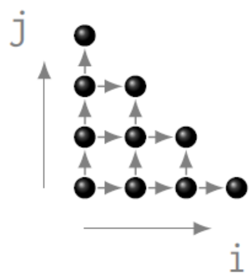
- Address pre-calculation
- Arithmetic simplifications
- Vectorization (SSE3, AVX, AVX2, QPX, NEON)
- Loop unrolling
  - Also enables further optimizations, e.g. eliminating modulo accesses
- Polyhedron model extraction and optimization
  - Sophisticated dependency analysis
  - Advanced dead code elimination
  - 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];
```

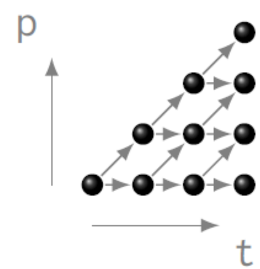
```
for ( int t = 1; t <= n; ++t)
#pragma omp parallel for
    for ( int p = 1; p <= t; ++p )
        a[t-p+1][p] = ...;
```

Iteration domain



Transformation

$$t = i + j - 1$$
$$p = j$$



$$1 \leq i \leq n$$
$$1 \leq j \leq n - i + 1$$

$$1 \leq t \leq n$$
$$1 \leq p \leq t$$

Dependences

$$(i,j) \to (i+1,j)$$
$$(i,j) \to (i,j+1)$$

$$(t,p) \to (t+1,p)$$
$$(t,p) \to (t+1,p+1)$$

ExaSlang 4 Example


```
Function JacSmoother@((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

Resulting Code (w/o basic Opt)

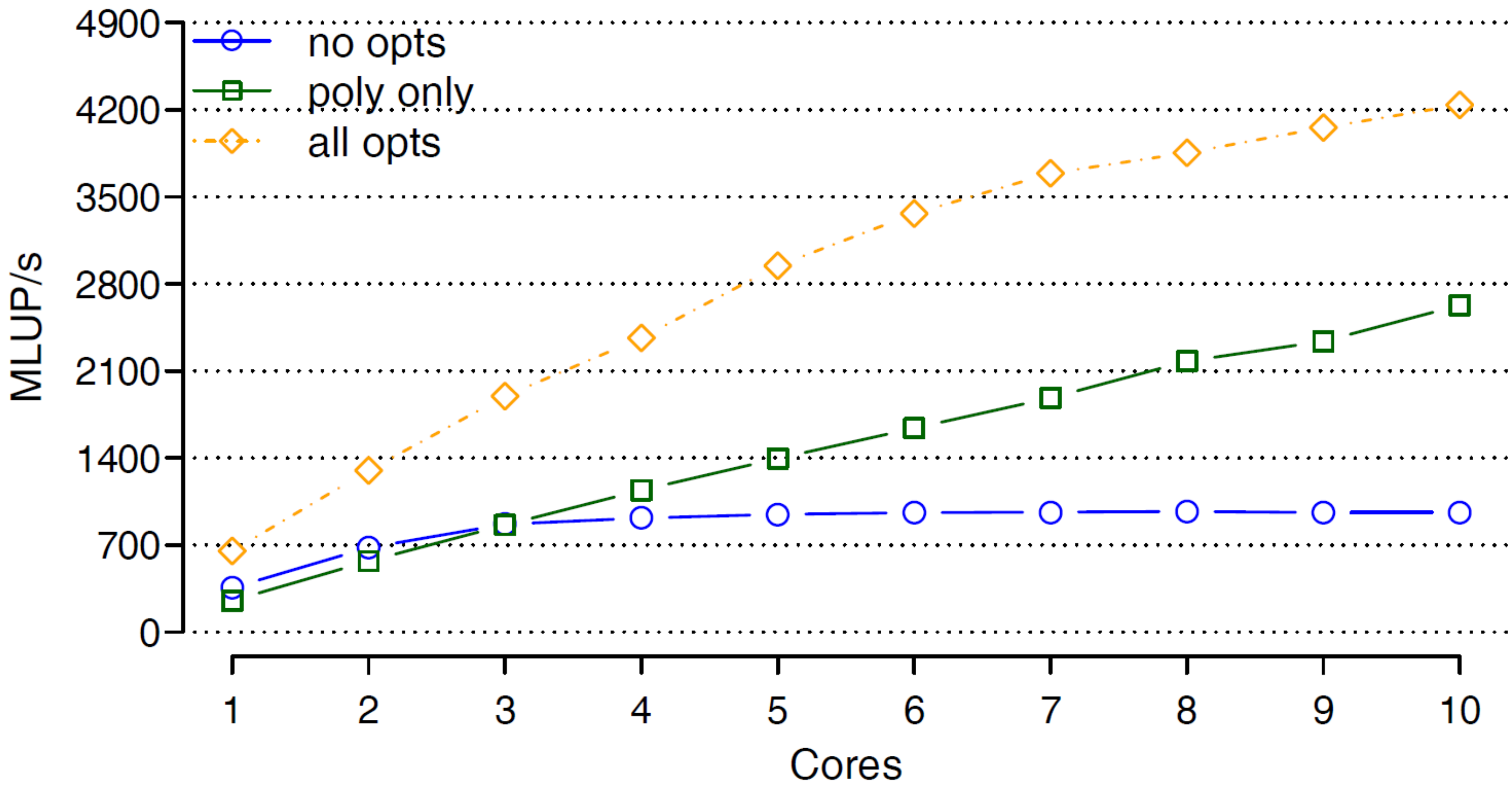

```
#include "MultiGrid/MultiGrid.h"
void Smoother_4() {
  exchsolutionData_4(0);
#pragma omp parallel for schedule(static) num_threads(8)
  for (int fragmentIdx = 0; fragmentIdx < 8; ++fragmentIdx) {
    if (isValidForSubdomain[fragmentIdx][0]) {
      for (int y = iterationOffsetBegin[fragmentIdx][0][1];
               y < (iterationOffsetEnd[fragmentIdx][0][1]+17); y +=1)
        for (int x = iterationOffsetBegin[fragmentIdx][0][0];
                 x < (iterationOffsetEnd[fragmentIdx][0][0]+17); x +=1)
          slottedFieldData_Solution[1][fragmentIdx][4][(((y*19)+19)+(x+1))] =
         (slottedFieldData_Solution[0][fragmentIdx][4][(((y*19)+19)+(x+1))]
         +(((1.0e+00/fieldData_LaplCoeff[fragmentIdx][4][((y*17)+x)])*8.0e-01)
         *(fieldData_RHS[fragmentIdx][4][((y*17)+x)]
         -(((((fieldData_LaplCoeff[fragmentIdx][4][((y*17)+x)]
         *slottedFieldData_Solution[0][fragmentIdx][4][(((y*19)+19)+(x+1))])
         +(fieldData_LaplCoeff[fragmentIdx][4][(((y*17)+289)+x)]
         *slottedFieldData_Solution[0][fragmentIdx][4][(((y*19)+19)+(x+2))]))
         +(fieldData_LaplCoeff[fragmentIdx][4][(((y*17)+578)+x)]
         *slottedFieldData_Solution[0][fragmentIdx][4][(((y*19)+19)+x)]))
         +(fieldData_LaplCoeff[fragmentIdx][4][(((y*17)+867)+x)]
         *slottedFieldData_Solution[0][fragmentIdx][4][(((y*19)+38)+(x+1))]))
         +(fieldData_LaplCoeff[fragmentIdx][4][(((y*17)+1156)+x)]
         *slottedFieldData_Solution[0][fragmentIdx][4][((y*19)+(x+1))])))));
...
```

Resulting Code (w/ basic Opt)

```
#include "MultiGrid/MultiGrid.h"
void Smoother_4() {
  exchsolutionData_4(0);
#pragma omp parallel for schedule(static) num_threads(8)
  for (int fragmentIdx = 0; fragmentIdx < 8; ++fragmentIdx) {
    if (isValidForSubdomain[fragmentIdx][0]) {
      for (int c0 = iterationOffsetBegin[fragmentIdx][0][1];
                (c0<=(iterationOffsetEnd[fragmentIdx][0][1]+16)); c0 = (c0+1))
        double* slottedFieldData_Solution_1_fragmentIdx_4_p1 =
         &(slottedFieldData_Solution[1][fragmentIdx][4][(19*c0)]);
        double* fieldData_RHS_fragmentIdx_4_p1 = &(fieldData_RHS[fragmentIdx][4][(17*c0)]);
        double* slottedFieldData_Solution_0_fragmentIdx_4_p1 = …
        double* fieldData_LaplCoeff_fragmentIdx_4_p1 = …
        for (int c1 = iterationOffsetBegin[fragmentIdx][0][0];
                  (c1<=(iterationOffsetEnd[fragmentIdx][0][0]+16)); c1 = (c1+1)) {
         slottedFieldData_Solution_1_fragmentIdx_4_p1[(c1+20)] =
         (slottedFieldData_Solution_0_fragmentIdx_4_p1[(c1+20)]
         +(((1.0e+00/fieldData_LaplCoeff_fragmentIdx_4_p1[c1])*8.0e01)*(fieldData_RHS_fragmentIdx_4_p
         1[c1] -
         (((((fieldData_LaplCoeff_fragmentIdx_4_p1[c1]*slottedFieldData_Solution_0_fragmentIdx_4_p1[(c1
         +20)])+(fieldData_LaplCoeff_fragmentIdx_4_p1[(c1+289)]*slottedFieldData_Solution_0_fragmentId
         x_4_p1[(c1+21)]))+(fieldData_LaplCoeff_fragmentIdx_4_p1[(c1+578)]*slottedFieldData_Solution_0
         _fragmentIdx_4_p1[(c1+19)]))+(fieldData_LaplCoeff_fragmentIdx_4_p1[(c1+867)]*slottedFieldData
         _Solution_0_fragmentIdx_4_p1[(c1+39)]))+(fieldData_LaplCoeff_fragmentIdx_4_p1[(c1+1156)]*slott
         edFieldData_Solution_0_fragmentIdx_4_p1[(c1+1)])))));
…
```

Node-Level Performance

3D 7-point Jacobi smoother Intel IvyBridge EP



Distributed-memory Parellelization

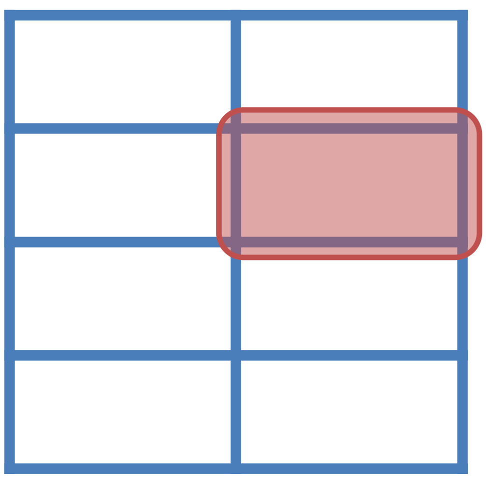
FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

TECHNISCHE FAKULTÄT

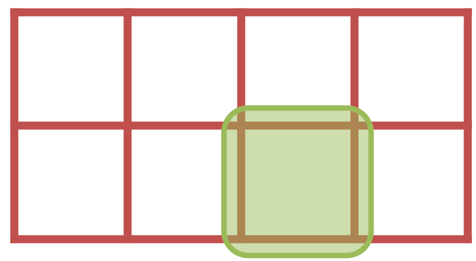
Domain Partitioning

- Easy for regular domains

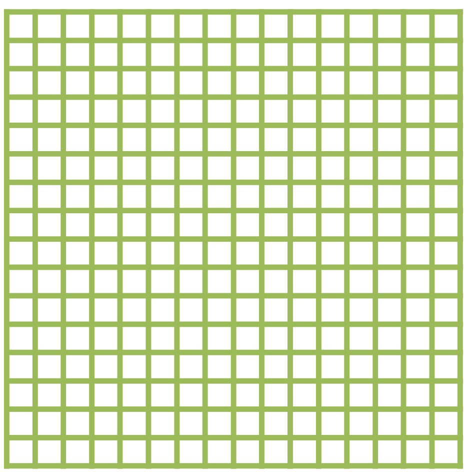
Each **domain** consists
of one or more **blocks**



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



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



Domain Partitioning

- 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

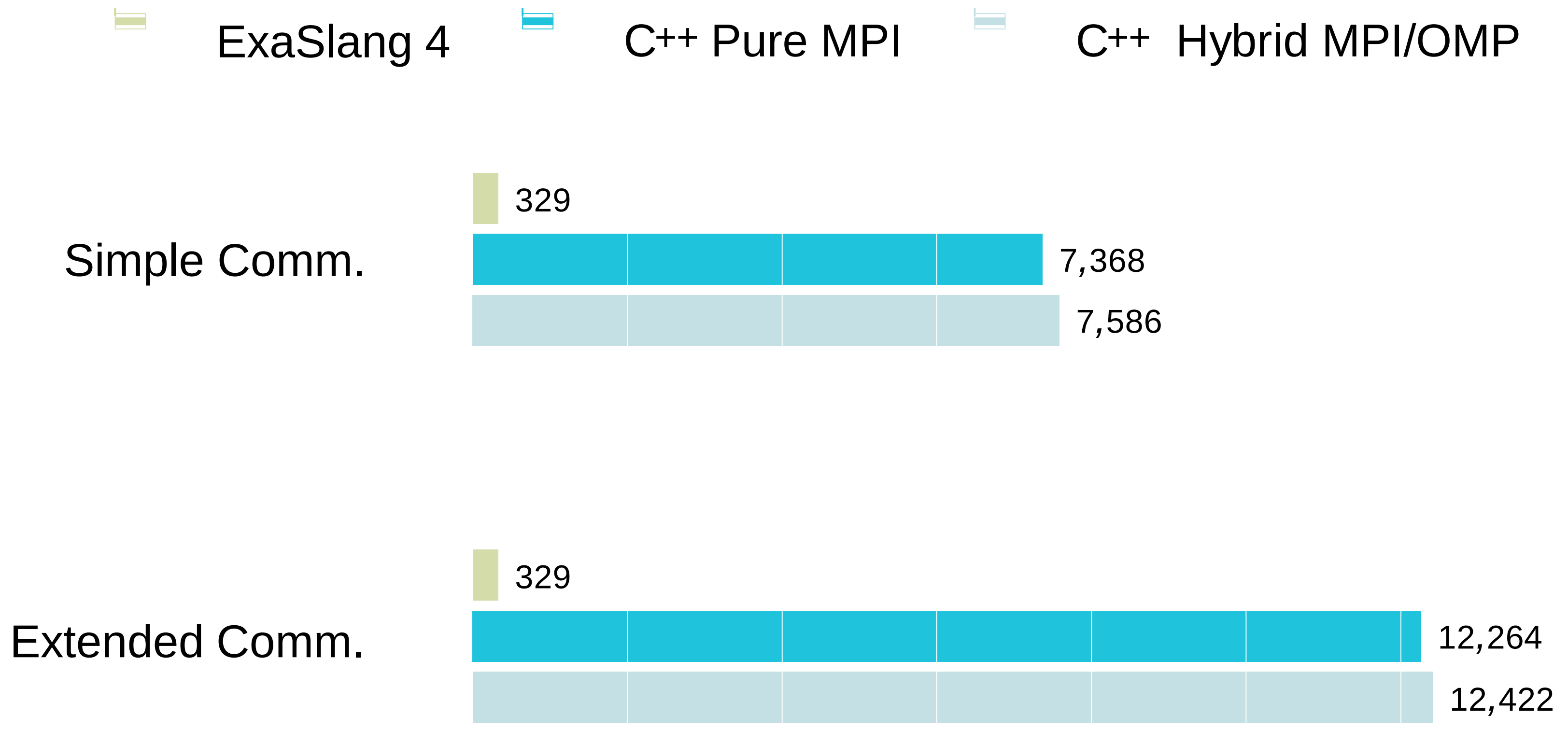
```
/* communicates all applicable layers */
communicate Solution@current
/* communicates only ghost layers */
communicate ghost of Solution[active]@current
/* communicates duplicate and first two ghost layers */
communicate dup, ghost[0, 1] of Solution[active]@current
/* asynchronous communicate */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

Benchmark Problem and System

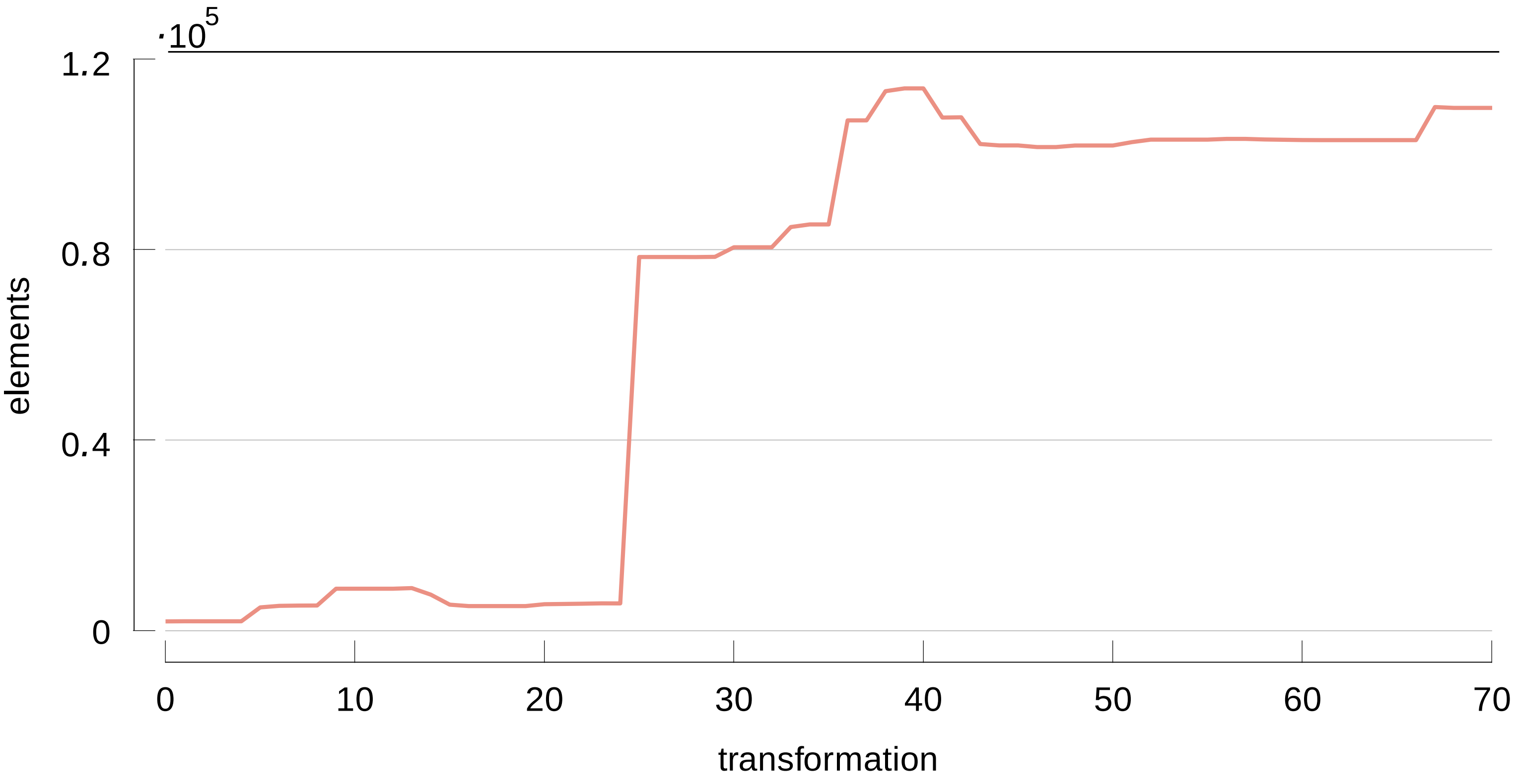
- Target system
  - JUQUEEN supercomputer located in Jülich, Germany
  - 458,752 cores / 28,672 nodes (1.6 GHz, 16 cores each, four-way multithreading)
- Regarded problem
  - 3D finite differences discretization of Poisson's equation ($\Delta \varphi = f$) with Dirichlet boundary conditions
  - V(3,3) cycle, parallel CG as coarse grid solver
  - Jacobi, Gauss-Seidel or red-black Gauss-Seidel smoother
  - pure MPI or hybrid MPI/OMP parallelization
  - 64 threads per node, roughly 10^6 unknowns per core
  - code optimized through polyhedral loop transformations, 2-way unrolling and address precalculation on finer levels as well as custom MPI data types
  - vectorization and blocking are not yet taken into account

Comparison of Lines of Code

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG
TECHNISCHE FAKULTÄT



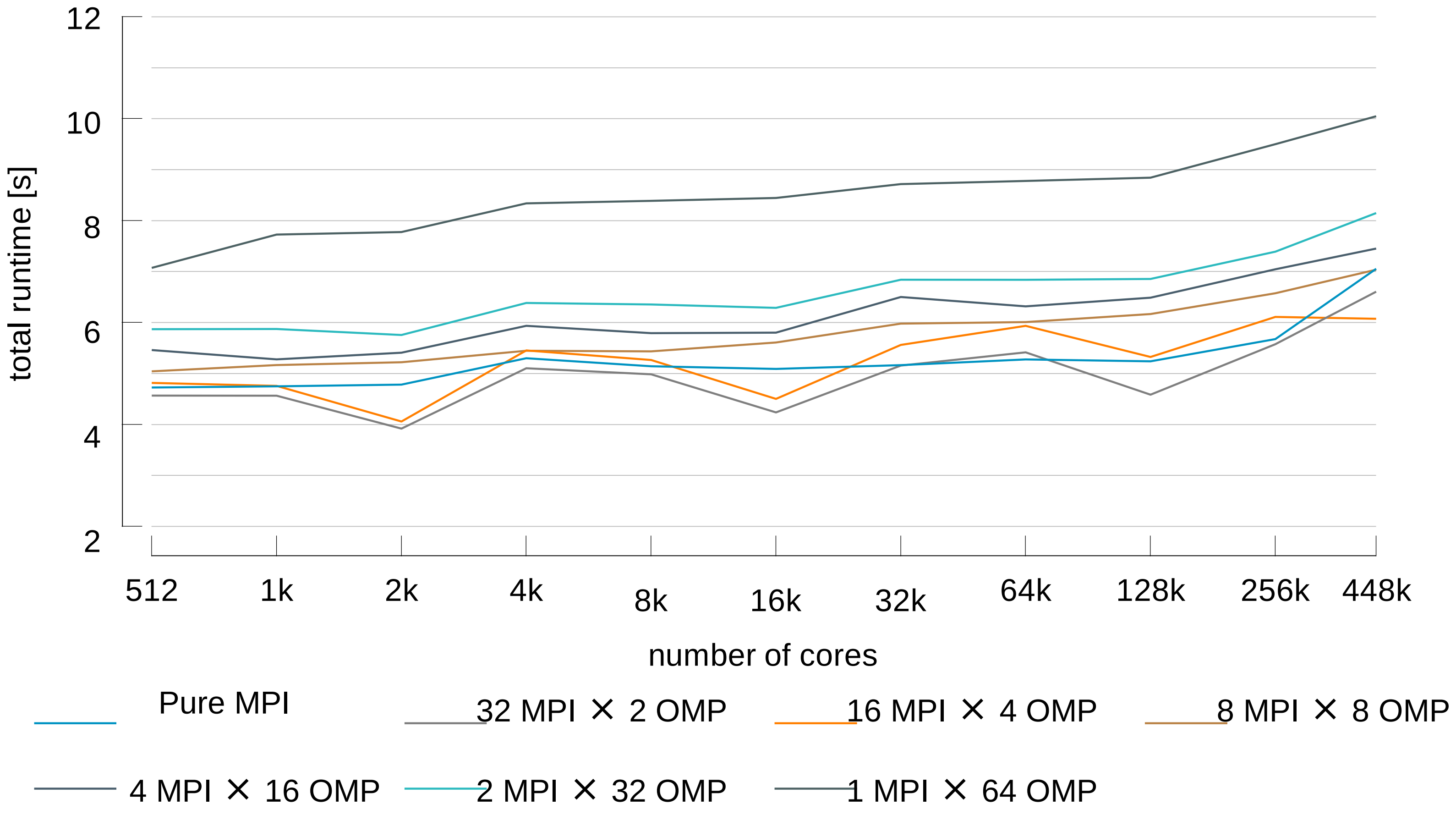
Program Sizes during Transformation



Weak Scalability

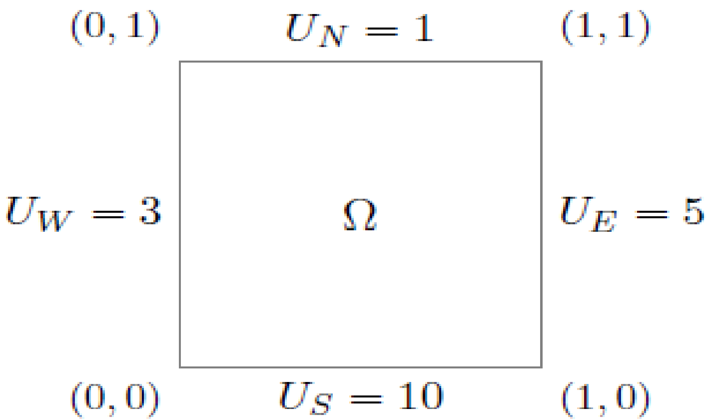
FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG
TECHNISCHE FAKULTÄT

- Mean time per V-cycle
- V(3,3) with Jacobi and CG

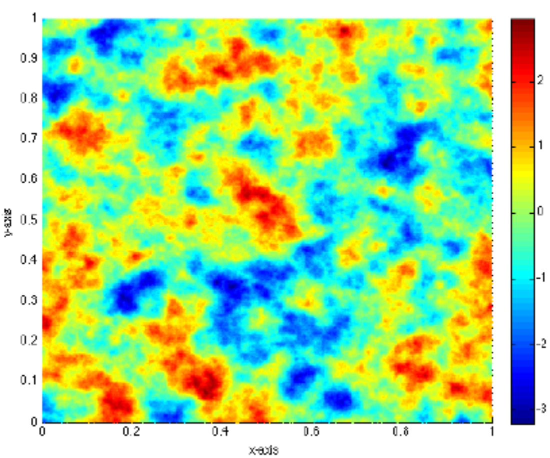


Applications

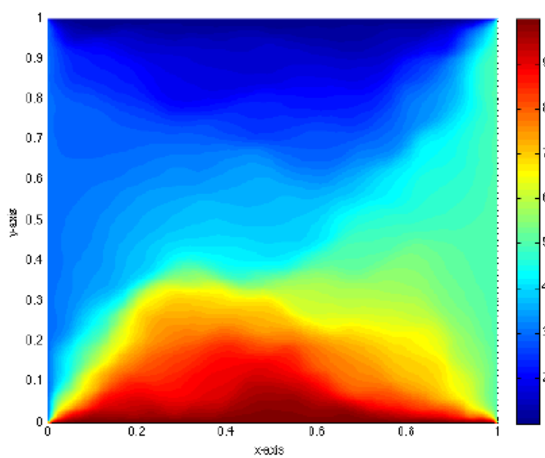
Stochastic variable heat conduction



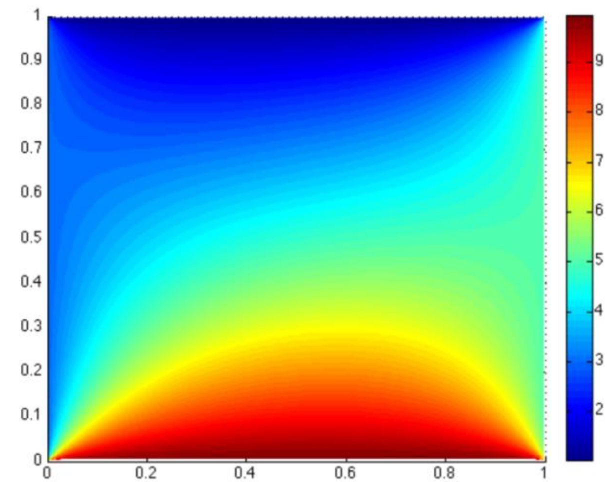
$$\nabla \cdot \left(e^{a(\boldsymbol{x})} \nabla U(\boldsymbol{x})\right) = 0 \qquad \boldsymbol{x} \in \Omega = [0,1] \times [0,1]$$
$$U(0, x_2) = 3 \qquad x_2 \in \partial\Omega_W = [0,1]$$
$$U(1, x_2) = 5 \qquad x_2 \in \partial\Omega_E = [0,1]$$
$$U(x_1, 0) = 10 \qquad x_1 \in \partial\Omega_S = [0,1]$$
$$U(x_1, 1) = 1 \qquad x_1 \in \partial\Omega_N = [0,1]$$



GRMF



solution



Expectation of solution field (100 samples, Monte Carlo simulation)

Loh, *Solving Stochastic PDEs with Approximate Gaussian Markov Random Fields using Different Programming Environments,* 2014.

Non-Newtonian Fluid Flows

- Simulation of non-isothermal/non-Newtonian fluid flows
  - Suspensions of particles or macromolecules
  - E.g. pastes, gels, foams, drilling fluids, food products, blood, etc.
  - Importance in mining, chemical and food industry as well as medical applications

https://www.youtube.com/watch?v=G1Op_1yG6lQ

Differences to Poisson

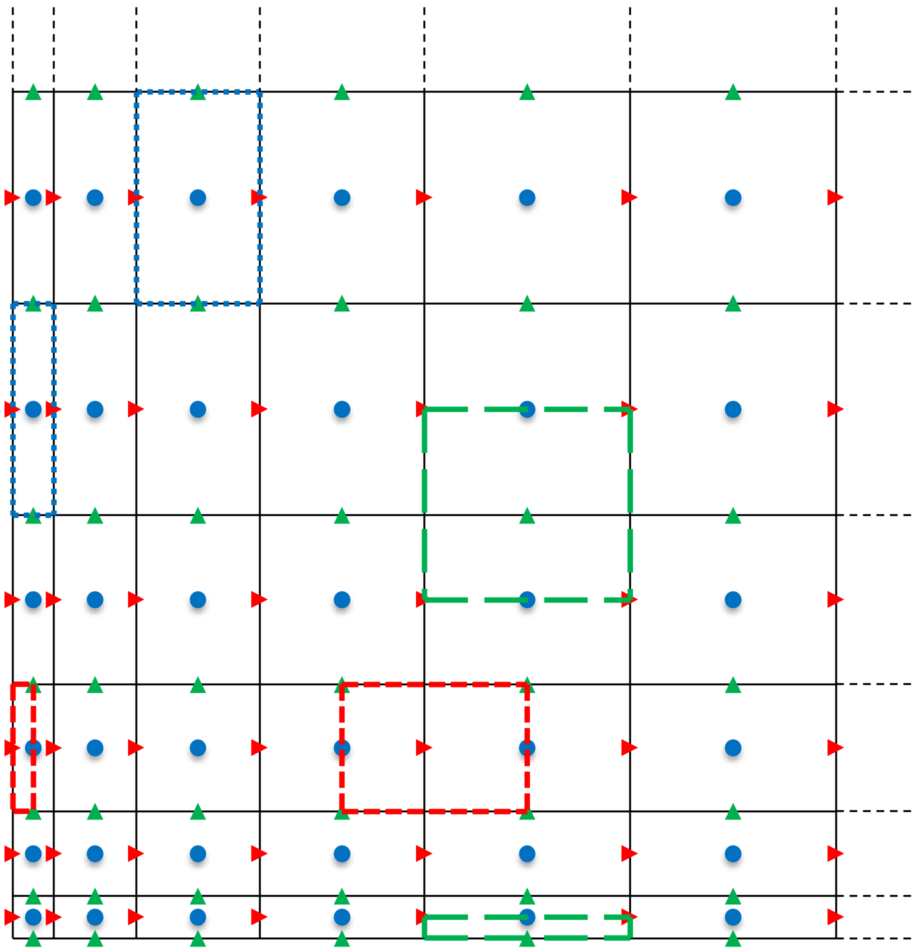
FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG
TECHNISCHE FAKULTÄT

- From model problem to application

Poisson	NNF
$\Delta u = f$	$-\nabla^T(H\nabla\vec{v}) + D(\vec{v}^T \cdot \nabla)\vec{v} + \nabla p + D\begin{bmatrix}0\\\theta\\0\end{bmatrix} = 0$ $\nabla^T\vec{v} = 0$ $-\nabla^T(\nabla\theta) + G \cdot (\vec{\mathrm{v}}^T \cdot \nabla)\theta = 0$
Linear	Non-linear
Scalar PDE (one unknown)	3 velocity components, pressure and temperature with varying localizations
Simple boundary conditions	Mixed boundary conditions
Finite differences	Finite volumes
One grid	Staggered grids
Uniform spacing	Local refinement

Staggered grid

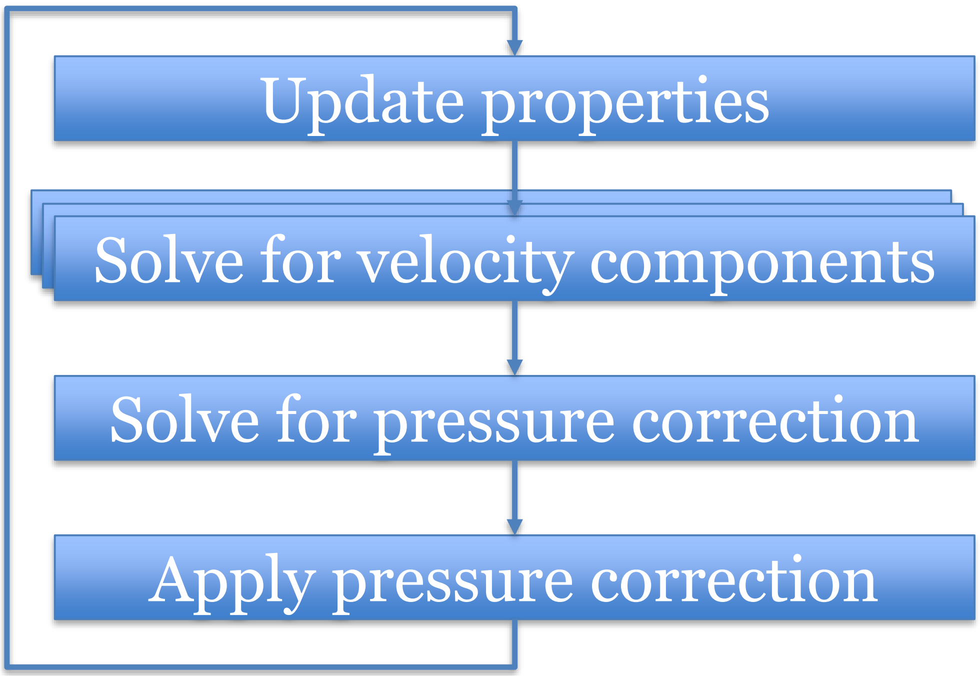
- Finite volume discretization on a staggered grid



- values associated with the cell centers, e.g. p and θ
- values associated with the x-staggered grid, e.g. U
- values associated with the y-staggered grid, e.g. V
- control volumes associated with cell-centered values
- control volumes associated with x-staggered values
- control volumes associated with y-staggered values

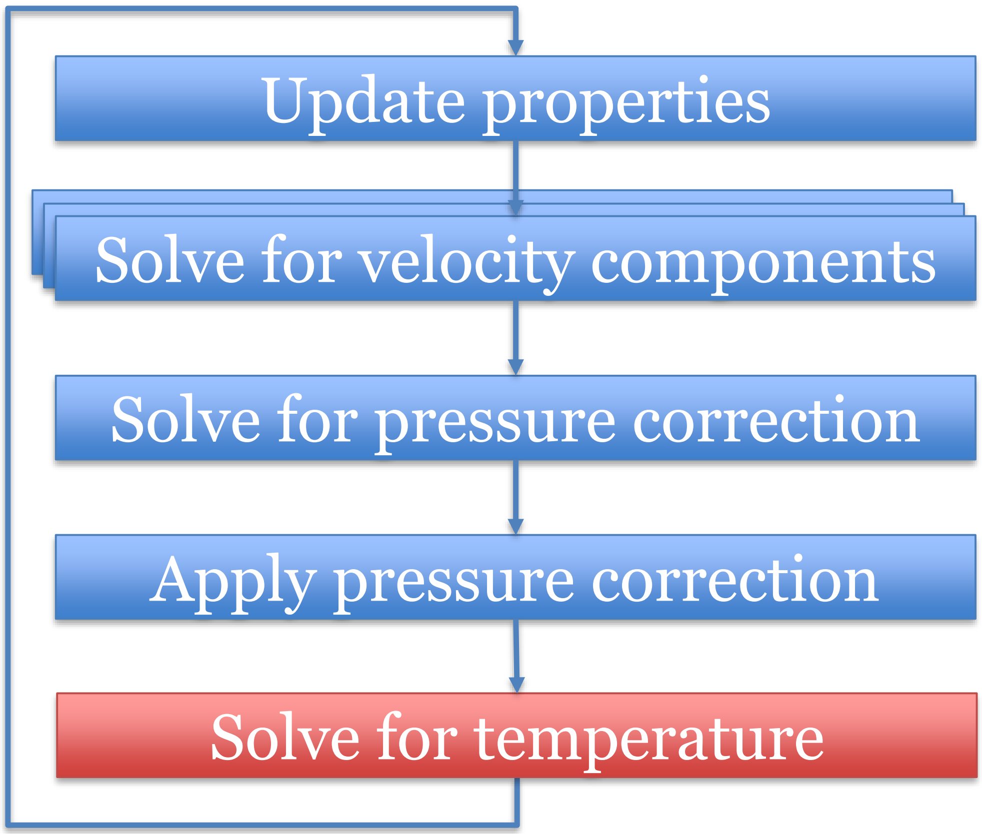
The SIMPLE Algorithm

- **S**emi-**I**mplicit **M**ethod for **P**ressure **L**inked **E**quations
- Concept:



The SIMPLE Algorithm

- Temperature can be added as a separate step



Porting code

- Straight-forward for simple kernels

<pre>subroutine advance_fields ()
!$omp parallel do &
!$omp private(i,j,k) &
!$omp firstprivate(l1,m1,n1) &
!$omp shared(rho,rho0) &
!$omp schedule(static) default(none)
do k=1,n1
  do j=1,m1
    do i=1,l1
      rho0(i,j,k)=rho(i,j,k)
    end do
  end do
end do
!$omp end parallel do</pre>
	<pre>Function AdvanceFields@finest ()
	  : Unit {
	  loop over rho@current {
	    rho[next]@current =
	      rho[active]@current
	  }
	  advance rho@current
	}</pre>

Porting code

- But what about more complicated code? Here we get from this …

```
! if not at the boundary
fl = xcvi(i) * v(i,jp,k) * (fy(jp)*rho(i,jp,k) + fym(jp)*rho(i,j,k))
flm = xcvip(im) * v(im,jp,k) * (fy(jp)*rho(im,jp,k) + fym(jp)*rho(im,j,k))
flownu = zcv(k) * (fl+flm)
gm = xcvi(i) * vis(i,j,k) * vis(i,jp,k) / (ycv(j)*vis(i,jp,k) + ycv(jp)*vis(i,j,k) + 1.e-30)
gmm = xcvip(im) * vis(im,j,k) * vis(im,jp,k) /(ycv(j)*vis(im,jp,k) +
ycv(jp)*vis(im,j,k) + 1.e-30)
diff = 2. * zcv(k) * (gm+gmm)
call diflow(flownu,diff,acof)
adc = acof + max(0.,flownu)
anu(i,j,k) = adc - flownu
```

Porting code

FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG
TECHNISCHE FAKULTÄT

- … to this!

```
flownu@current = integrateOverXStaggeredNorthFace (
  v[active]@current * rho[active]@current )

Val diffnu : Real = integrateOverXStaggeredNorthFace (
    evalAtXStaggeredNorthFace ( vis@current, "harmonicMean" ) )
  / vf_stagCVWidth_y@current@[0, 1, 0]

AuStencil@current:[0, 1, 0] = -1.0 * ( calc_diflow ( flownu@current, diffnu )
  + max ( 0.0, flownu@current ) - flownu@current )
```

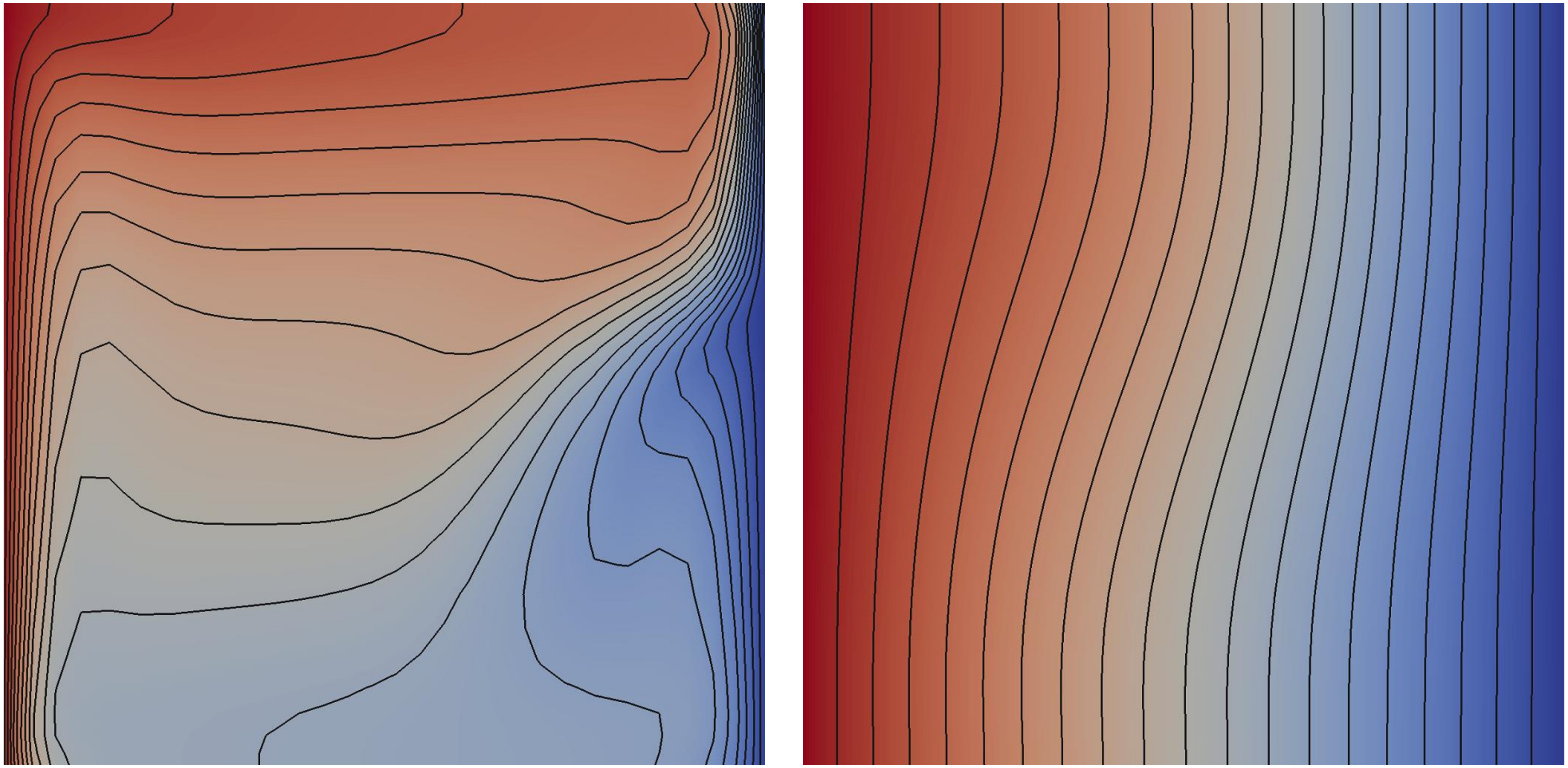
Results

- Natural convection problem
- With and without non-Newtonian model
- Convergence criteria for
  - solving the single components: At least one v-cycle and

$$\|r\| \leq \alpha(1 + \beta\|b\|)$$
$$\|r_{i+1}\| - \|r_i\| < \tilde{\alpha}$$

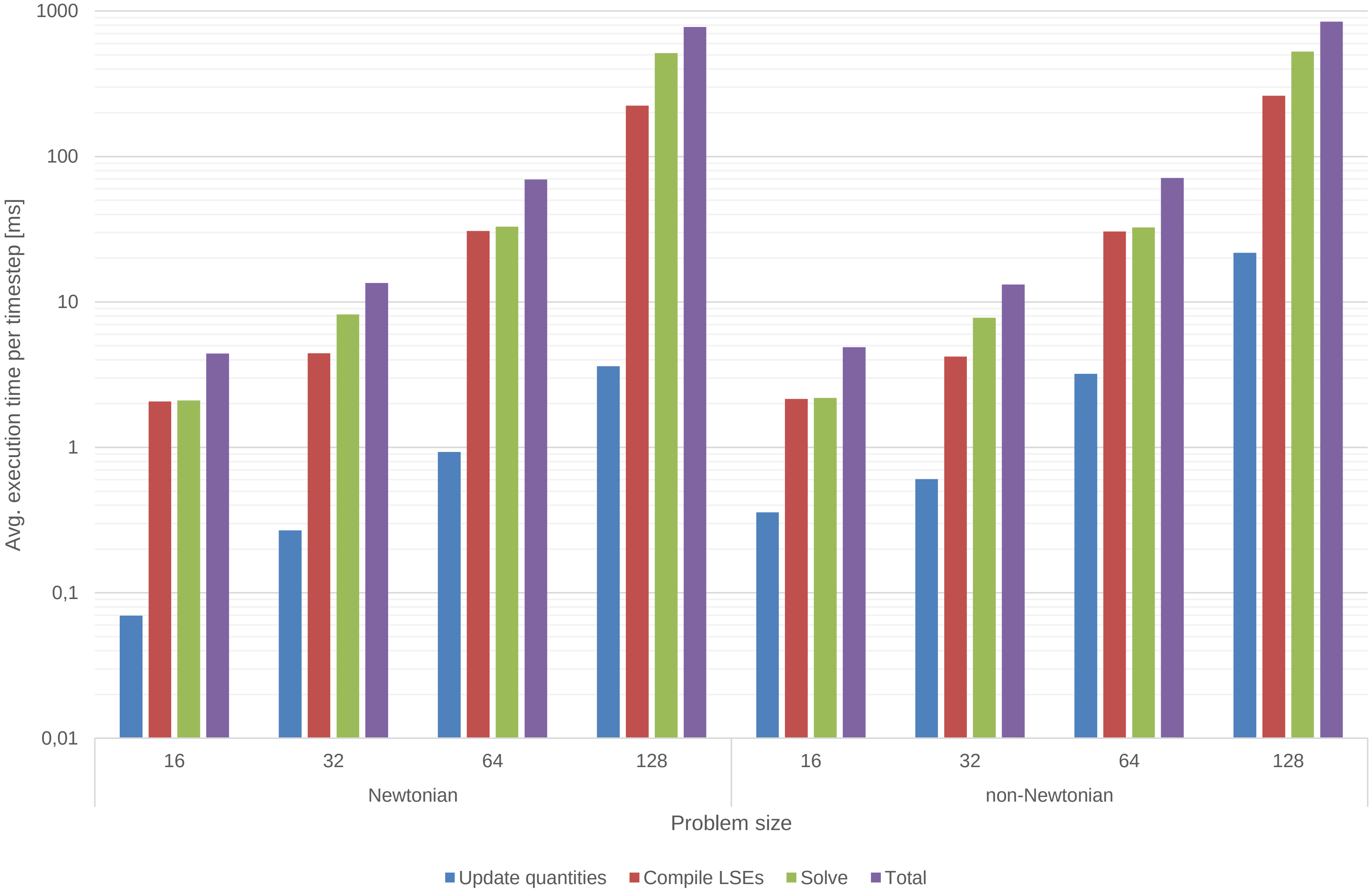
  - SIMPLE: Convergence for all components is achieved *after* updating the LSE but *before* starting the solve routine
- 10,000 timesteps
- One Socket on Emmy (Intel Xeon E5-2660v2) with 20 OpenMP threads

Results



Temperature distribution along slice (z at 50% of the box depth) for the Newtonian and non-Newtonian case

Results



1000
100
10
1
0,1
0,01
Avg. execution time per timestep [ms]
16
32
64
128
16
32
64
128
Newtonian
non-Newtonian
Problem size
Update quantities
Compile LSEs
Solve
Total

Towards Performance Engineering

FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG
TECHNISCHE FAKULTÄT

Assess Quality of Generated Code

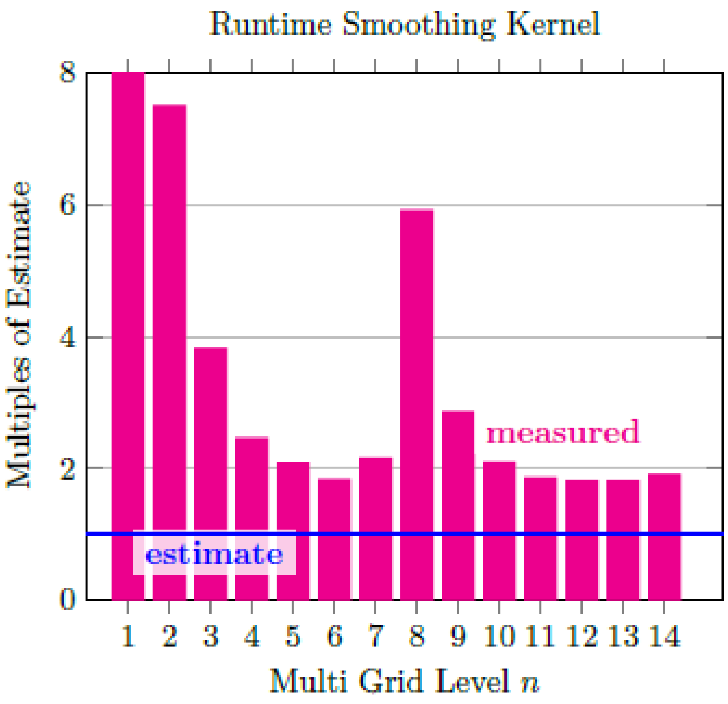
FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG
TECHNISCHE FAKULTÄT

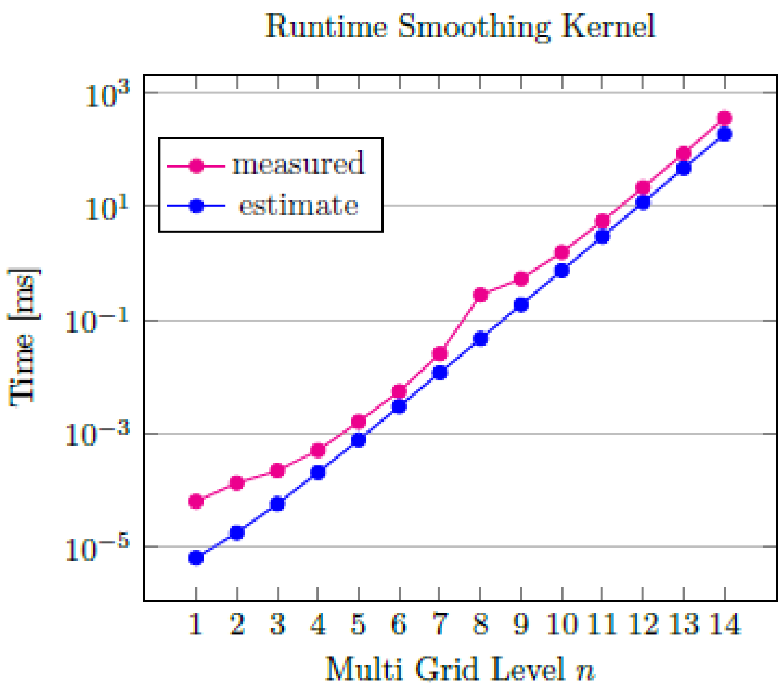
- Typically one cannot check manually how efficient generated code is
- Our solution to this problem is to generate also code for performance measurement
- And to create a performance model within the Scala compiler
- Relative performance would be ok since we want to know which configuration is best
- First results are now available

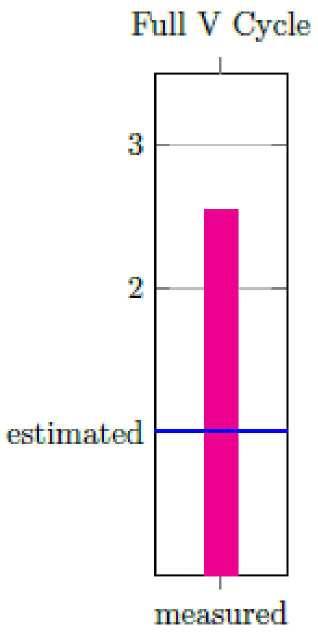
- For each kernel in AST
  - Evaluate kernel costs and save as annotation
    - Do an optimistic[2] estimation of required memory accesses (read/write)
    - Do an optimistic[3] estimation of required FLOPs
    - Apply roofline model using hardware characteristics
- Collect all relevant[1] functions in AST and mark them as unfinished
- While unfinished functions
  - For each unfinished function
    - Does function body contain function calls to unprocessed functions?
      - Defer function evaluation
    - Else
      - Evaluate body, save function cost and set function to finished
        - Static loops: multiply # iterations with sum of costs of body
        - Kernels: recall costs from annotation
        - Communication: not modelled at the moment
        - Function calls: recall costs of executing function

[1]omit, e.g., timer functions and error checks
[2]assume perfect spatial blocking / neglect temporal blocking;
[3]assume perfect vectorization

Runtimes 2D Constant Coefficients

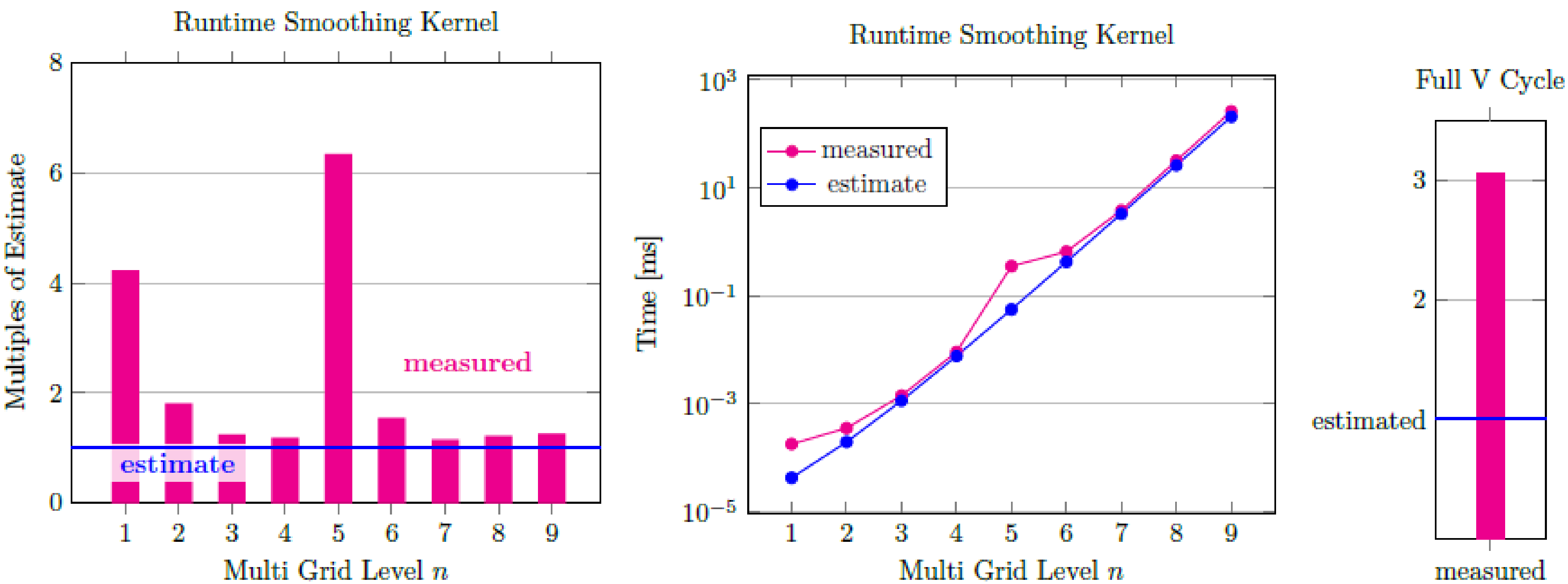






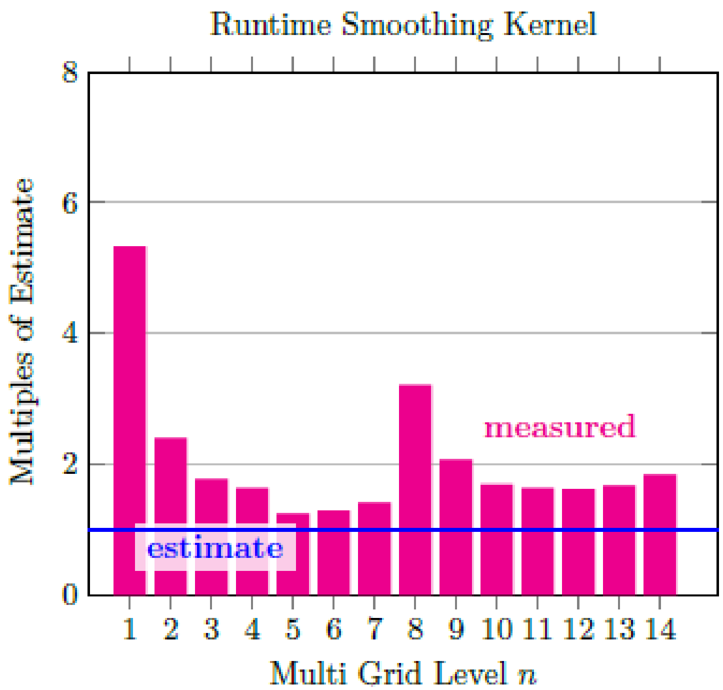
CPU single socket Xeon E3-1275 v5, 3.60 GHz clock, 4 cores / 8 hw-threads
Microarchitecture Skylake (2015) 2
L3 Cache 8 MB
Main Memory 64 GB

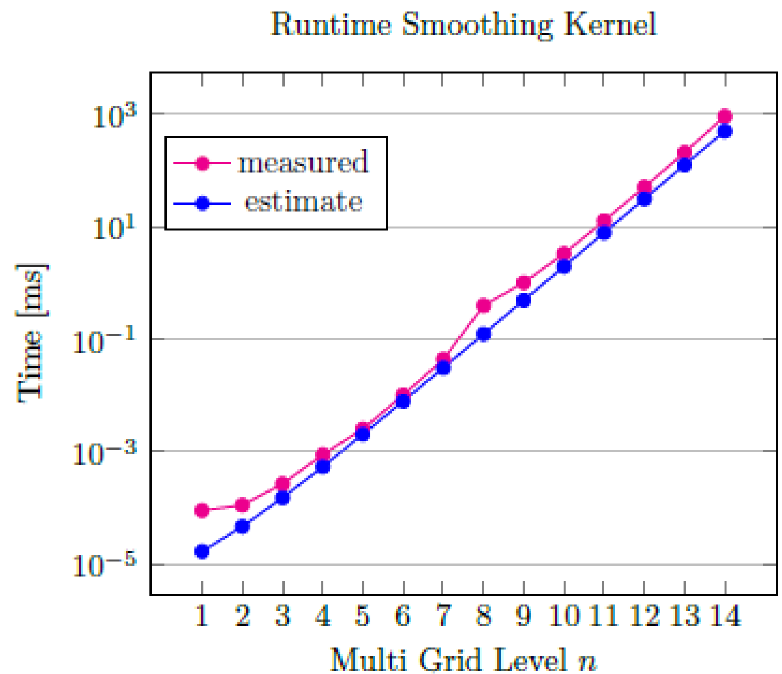
Runtimes 3D Constant Coefficients

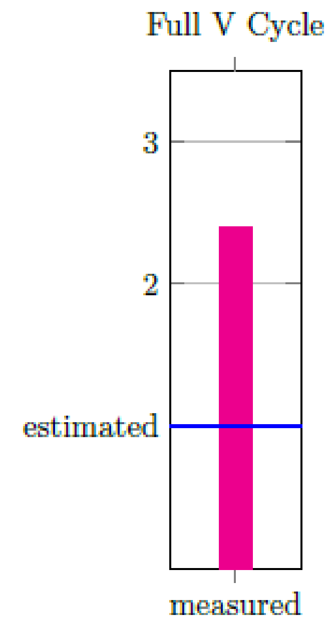


CPU single socket Xeon E3-1275 v5, 3.60 GHz clock, 4 cores / 8 hw-threads
Microarchitecture Skylake (2015) 2
L3 Cache 8 MB
Main Memory 64 GB

Runtimes 2D Variable Coefficients

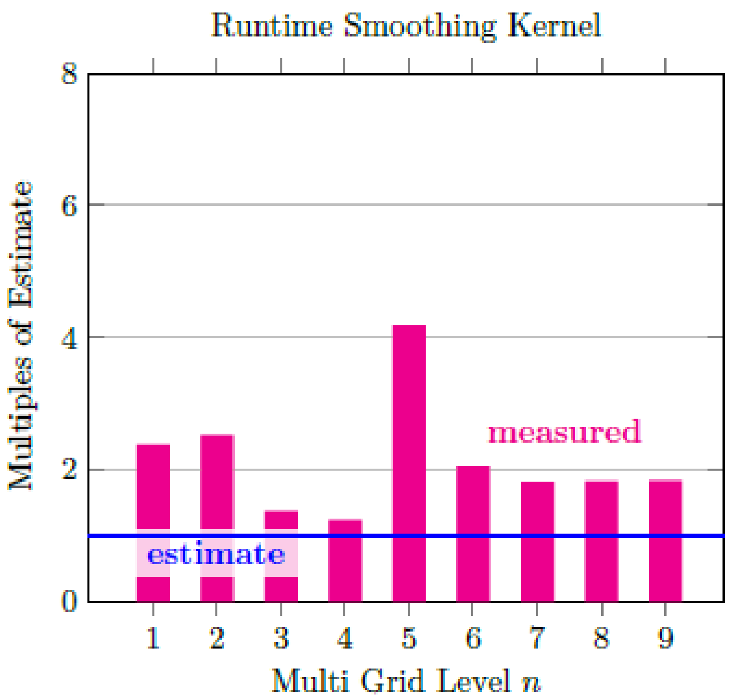


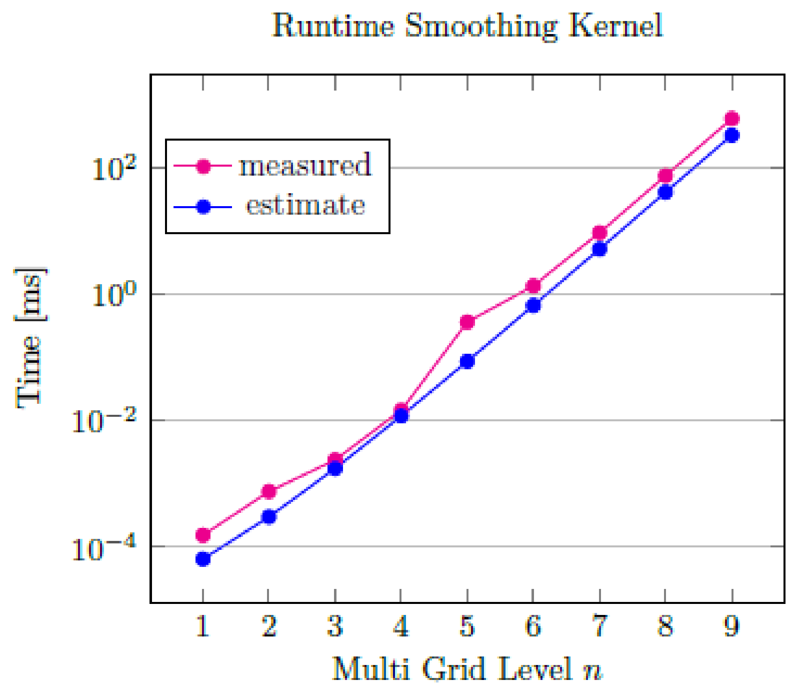


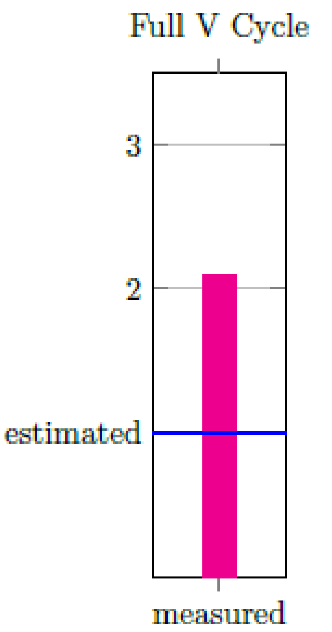


CPU single socket Xeon E3-1275 v5, 3.60 GHz clock, 4 cores / 8 hw-threads
Microarchitecture Skylake (2015) 2
L3 Cache 8 MB
Main Memory 64 GB

Runtimes 3D Variable Coefficients

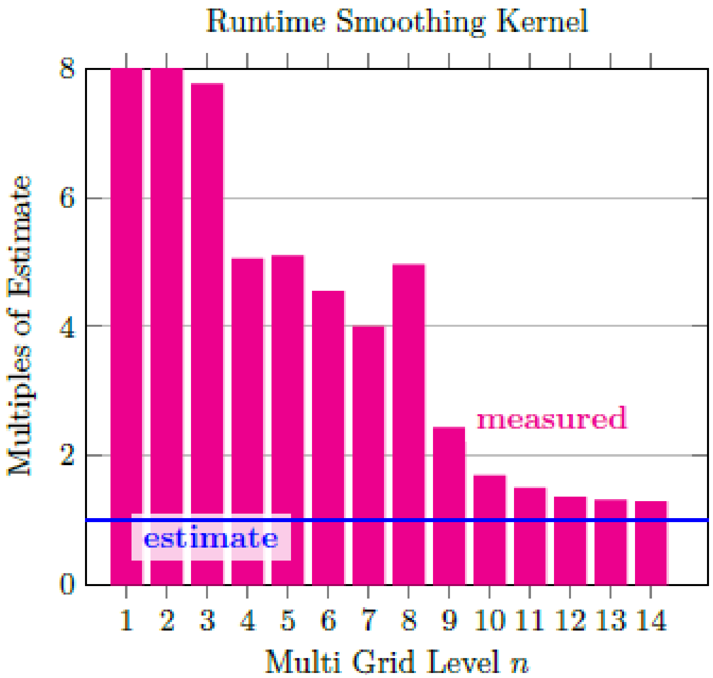


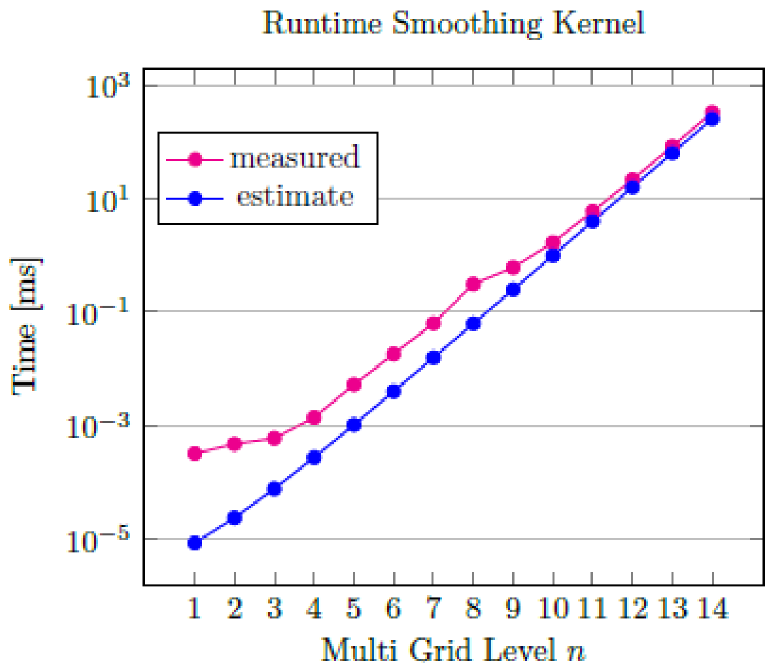


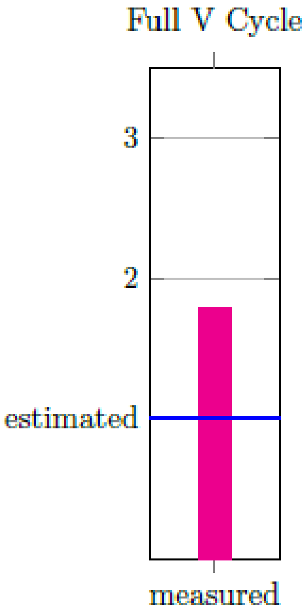


CPU single socket Xeon E3-1275 v5, 3.60 GHz clock, 4 cores / 8 hw-threads
Microarchitecture Skylake (2015) 2
L3 Cache 8 MB
Main Memory 64 GB

Runtimes 2D Constant Coefficients

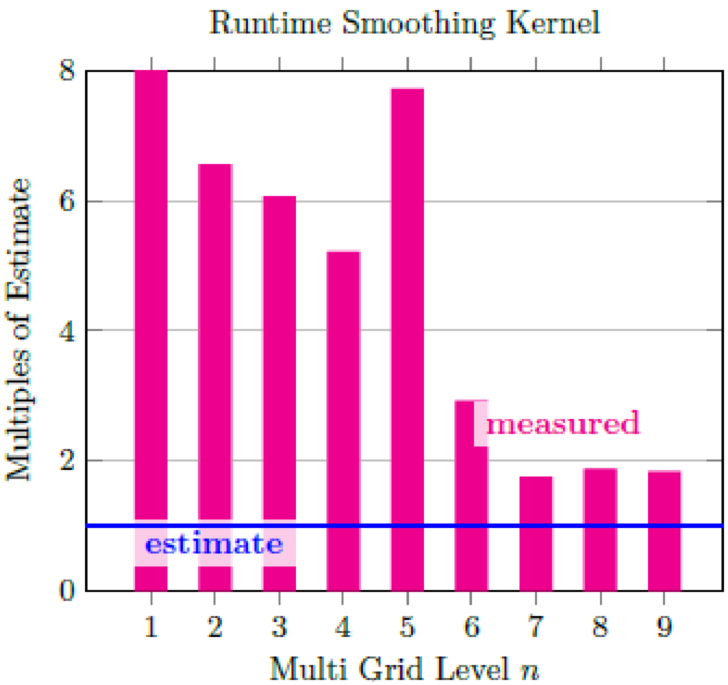


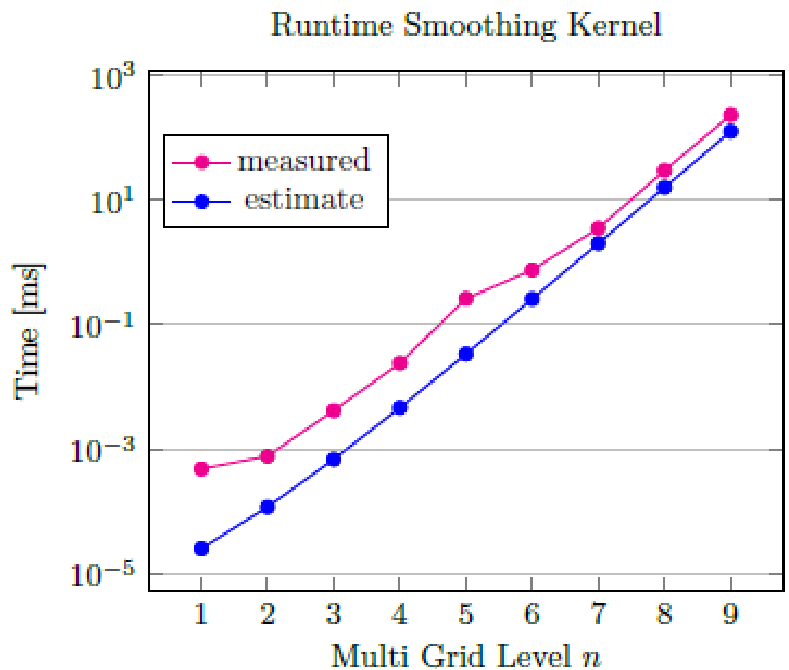


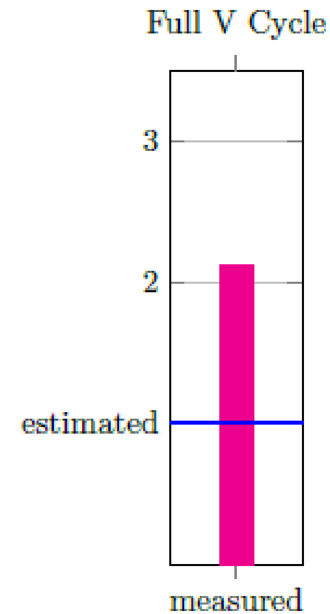


CPU dual socket Xeon E5620, 2.4 GHz clock, 4 cores / 8 hw-threads
Microarchitecture Westmere-EP/Gulftown (2010)
L3 Cache 12 MB smart cache
Main Memory 24 GB
Interconnect 2x QPI 5.86 GT/s

Runtimes 3D Constant Coefficients

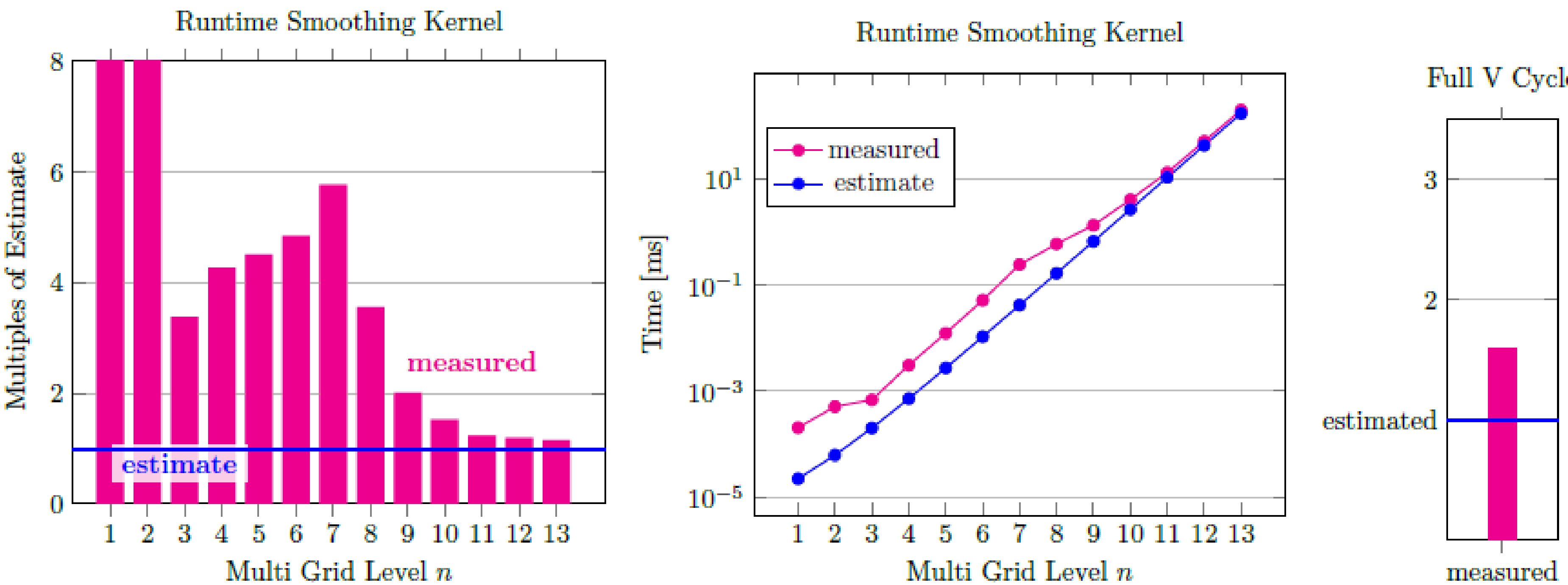






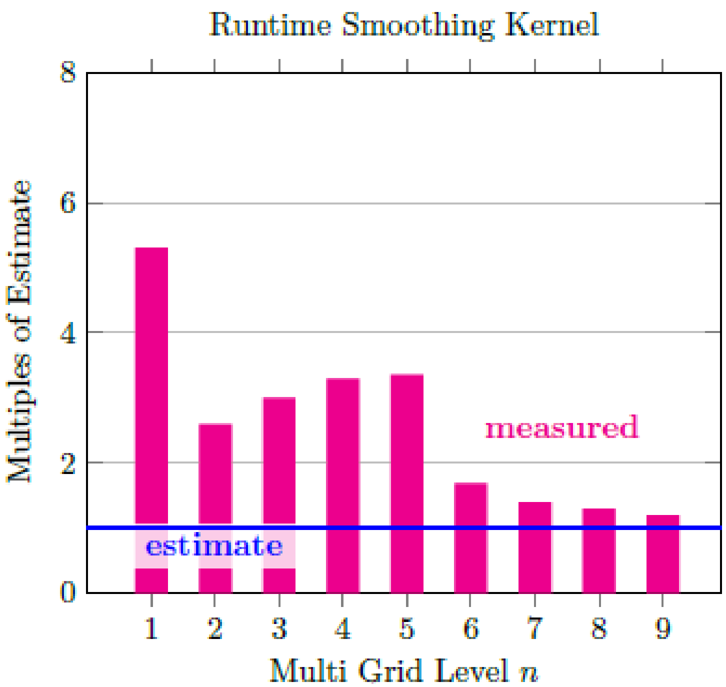
CPU dual socket Xeon E5620, 2.4 GHz clock, 4 cores / 8 hw-threads
Microarchitecture Westmere-EP/Gulftown (2010)
L3 Cache 12 MB smart cache
Main Memory 24 GB
Interconnect 2x QPI 5.86 GT/s

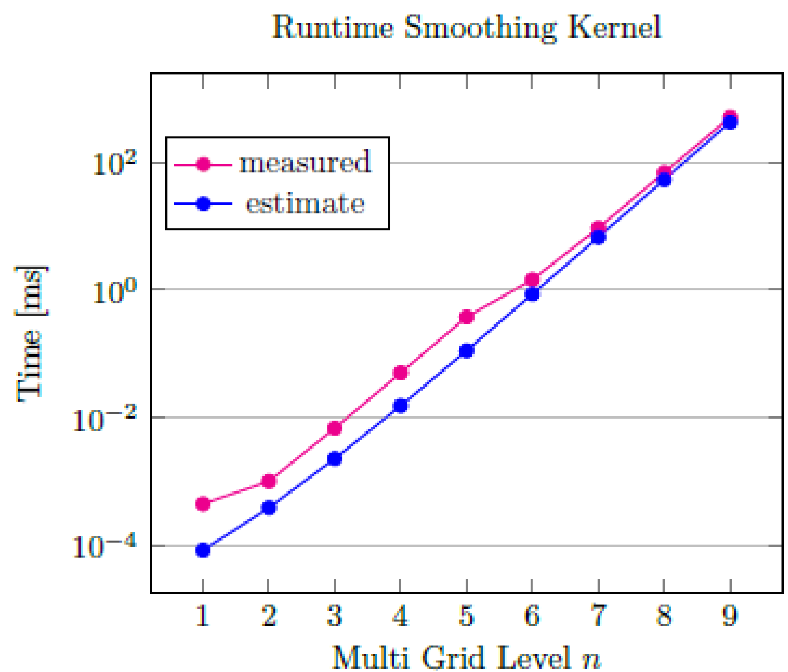
Runtimes 2D Variable Coefficients

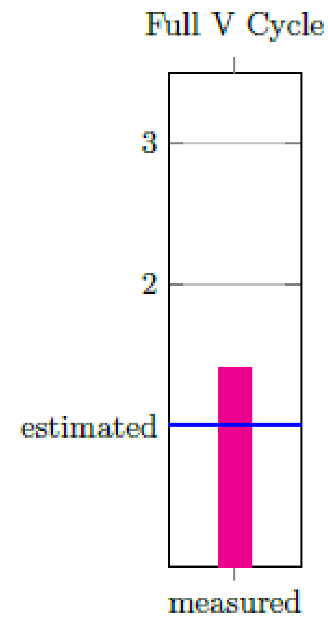


CPU dual socket Xeon E5620, 2.4 GHz clock, 4 cores / 8 hw-threads
Microarchitecture Westmere-EP/Gulftown (2010)
L3 Cache 12 MB smart cache
Main Memory 24 GB
Interconnect 2x QPI 5.86 GT/s

Runtimes 3D Variable Coefficients







CPU dual socket Xeon E5620, 2.4 GHz clock, 4 cores / 8 hw-threads
Microarchitecture Westmere-EP/Gulftown (2010)
L3 Cache 12 MB smart cache
Main Memory 24 GB
Interconnect 2x QPI 5.86 GT/s

Summary and Outlook

- Roughly a factor 2 off for a whole V-cycle is a good starting point
- For small sizes model is not accurate
- Include more advanced performance models
- Connect to performance measurement tools
- Include Autotuning

Acknowledgements

FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG
TECHNISCHE FAKULTÄT

- Funded by
  - Bundesministerium für Bildung und Forschung

    Bundesministerium für Bildung und Forschung

    des Bundesministeriums für Bildung und Forschung von Richtlinien zur Förderung von Forschungsvorhaben auf dem Gebiet "HPC-Software für skalierbare Parallelrechner" im Rahmen des Förderprogramms "IKT 2020 - Forschung für Innovationen"
  - KONWIHR. Bavarian project

    KONWIHR II
  - DFG SPP 1648/1 – Software for Exascale computing

    DFG Deutsche Forschungsgemeinschaft

    ExaStencils

    SPPEXA

    http://www.exastencils.org/
  - Industry

    P&G

    NVIDIA CUDA RESEARCH CENTER

    abberior INSTRUMENTS

    SIEMENS
  - Supercomputing centers

    HLRS

    lrz

    JÜLICH FORSCHUNGSZENTRUM

Thank you for your Attention!

Questions?

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

TECHNISCHE FAKULTÄT