FEM Simulations of Incompressible Flow using AD in the PDE Framework Peano

Hans-Joachim Bungartz, Tobias Neckel

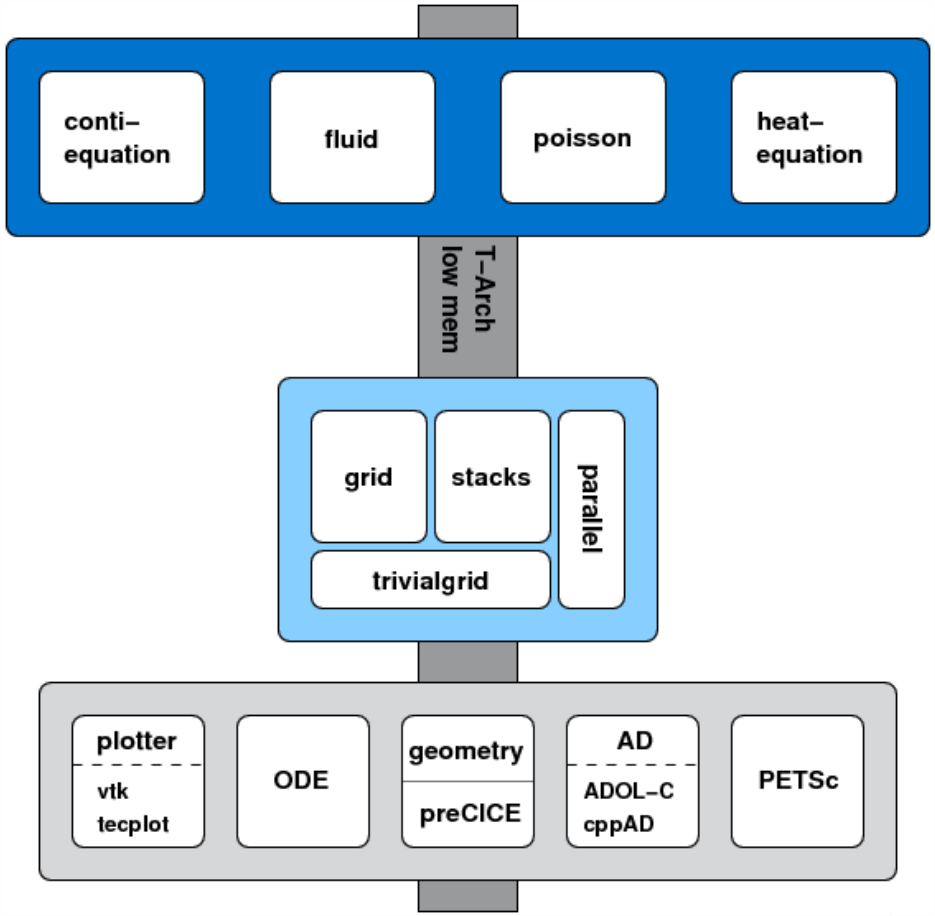
Scientific Computing in Computer Science,
Fakultät für Informatik
TU München
Germany

FEM Symposium 2009, Oberwiesenthal, September 30, 2009

Outline

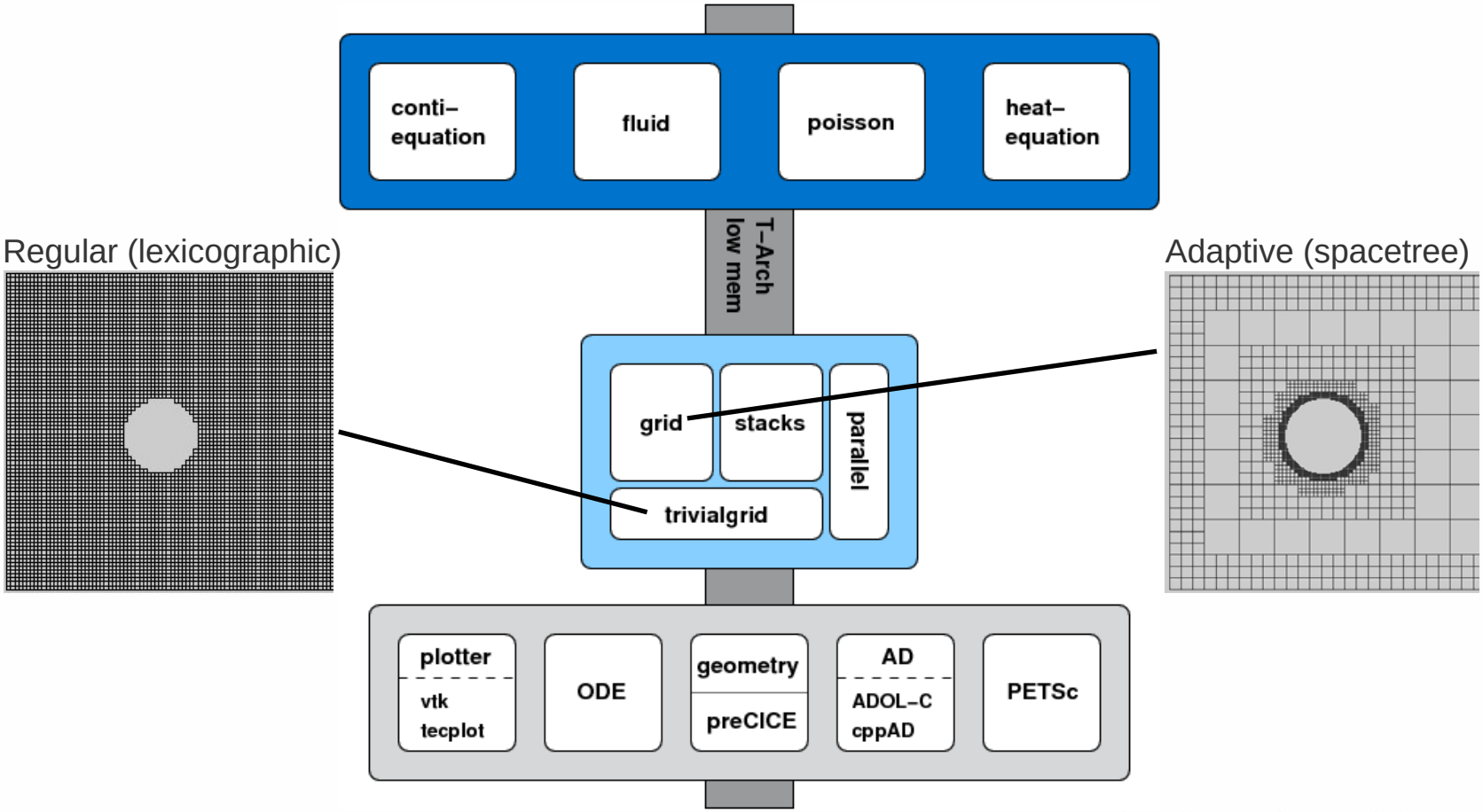
- The PDE Framework Peano
- Different Approaches for Jacobians
  - Finite Differences
  - Analytic Differentiation
  - Automatic Differentiation
- Numerical Results
- Outlook

The PDE Framework Peano



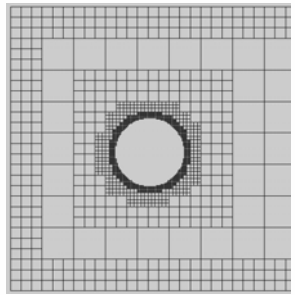
http://www5.in.tum.de/peano/

The PDE Framework Peano



http://www5.in.tum.de/peano/

The PDE Framework Peano



- Cartesian grids (arbitrary dimensions)
- Plug-in concept for applications
- Space-filling curves, spacetrees, and stack data structures
  - Strictly element-wise access
  - Low memory demands
  - Dynamical load balancing
  - Moving geometries, dynamical adaptivity, geometric multigrid
- Software Engineering
  - automatic tests, continuous integration, OO, design patterns, ...
- CFD component
  - Incompressible flow (FEM, IDO)
  - Explicit + implicit time-integration schemes (FE, RK4, BE, (adaptive) TR)

http://www5.in.tum.de/peano/

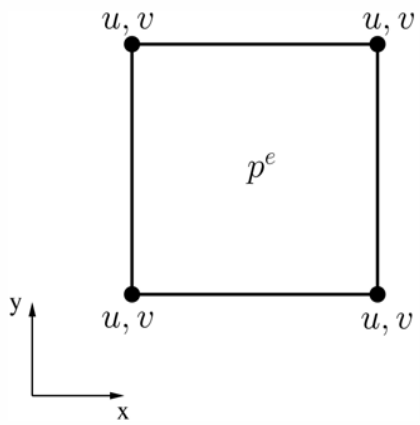
The PDE Framework Peano - CFD

- Incompressible Navier-Stokes equations (NSE)

$$\begin{aligned} \dot{\mathbf{u}} + (\mathbf{u} \cdot \nabla)\mathbf{u} + \frac{1}{\mathrm{Re}} \Delta \mathbf{u} - \nabla p &= \mathbf{0} \\ \nabla \cdot \mathbf{u} &= 0 \end{aligned}$$

- Discretisation
  - Low-order FEM (Q1Q0)
  - Steady-state
  - Discrete NSE:

$$\begin{aligned} C(\mathbf{u}_h)\mathbf{u}_h + D\mathbf{u}_h - M^T p_h &= \mathbf{0} \\ M\mathbf{u}_h &= 0 \end{aligned}$$



The PDE Framework Peano - CFD

- Nonlinear system of equations

$$\begin{aligned} C(\mathbf{u}_h)\mathbf{u}_h + D\mathbf{u}_h - M^T p_h &= \mathbf{0} \\ M\mathbf{u}_h &= 0 \end{aligned} \quad \Leftrightarrow \quad 0 = B(\mathbf{u}_h, p_h) =: B(u) \in \mathbf{R}^N$$

- Newton's method (PETSc)

$$\begin{aligned} k &= 0, 1, \ldots \\ J(u_k) \cdot \Delta u_k &= -B(u_k) \\ u_{k+1} &= u_k + \Delta u_k \end{aligned}$$

The PDE Framework Peano - CFD

- Nonlinear system of equations

$$\begin{aligned} C(\mathbf{u}_h)\mathbf{u}_h + D\mathbf{u}_h - M^T p_h &= \mathbf{0} \\ M\mathbf{u}_h &= 0 \end{aligned} \quad \Leftrightarrow \quad 0 = B(\mathbf{u}_h, p_h) =: B(u) \in \mathbf{R}^N$$

- Newton's method (PETSc)

$$\begin{aligned} k &= 0, 1, \ldots \\ J(u_k) \cdot \Delta u_k &= -B(u_k) \\ u_{k+1} &= u_k + \Delta u_k \end{aligned}$$

- Jacobian via
  - Finite differences (untuned + tuned)
  - Analytical implementation
  - Automatic differentiation

Jacobian - Finite Differences

- FD untuned
  - Use default B evaluation
  - Very (!) slow

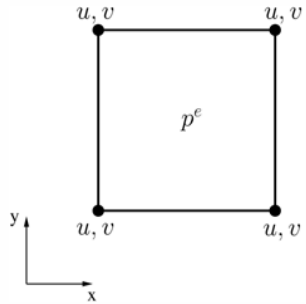
$$J(u,\varepsilon)_i \approx \frac{B(u+\varepsilon\cdot e_i)-B(u)}{\varepsilon}$$

$$\varepsilon = \begin{cases} 10^{-8}\cdot u_i & \text{if } |u_i| > 10^{-8} \\ 10^{-8}\cdot 10^{-8}\cdot sign(u_i) & \text{otherwise} \end{cases}$$

- FD (tuned)
  - Manual implementation (no default B evaluation)
  - Avoid unnecessary operator evaluations (due to stencil)

Jacobian – Analytic Differentiation

$$B\begin{pmatrix}\mathbf{u}_h\\ p_h\end{pmatrix} := \begin{pmatrix}\underbrace{C(\mathbf{u}_h)\,\mathbf{u}_h + D\mathbf{u}_h}_{=:F} + M^T p_h\\ M\mathbf{u}_h\end{pmatrix} = \mathbf{0}$$



- Block 1: linear
- Block 2: linear
- Block 3: non-linear

$$J^e = \left(\begin{array}{c|c} \text{Block 3}\ \frac{\partial F}{\partial \mathbf{u}_h} & \text{Block 2}\ \frac{\partial \nabla_h}{\partial p_h} \\ \hline \text{Block 1}\ \frac{\partial \nabla_h}{\partial \mathbf{u}_h} & 0 \end{array}\right) \in \mathbf{R}^{9\times 9}$$

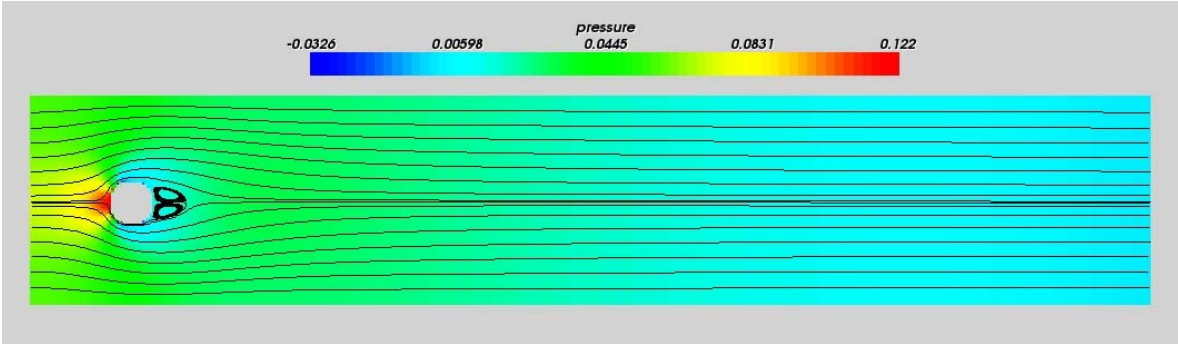
Jacobian - Automatic Differentiation

Supported AD variants in Peano:

- ADOL-C
  - Standard
  - Tapeless
  - One-touch
- cppAD
  - Standard
  - Tuned allocation

Numerical Results

- Benchmark scenario: Laminar flow around a cylinder 1)
  - Lift and drag forces for comparison
  - Regular Cartesian grids
  - Re = 20
  - Steady-state



- PETSc solver
  - Explicit assemblation of J
  - GMRES + ILU
  - rtol = atol = 1e-7

1) S. Turek and M. Schaefer, “Benchmark Computations of Laminar Flow around a Cylinder” Vieweg, Notes on Numerical Fluid Mechanics 52, 1996

Numerical Results - N=26,406

method	time total	time J	time B	lin. it. total	lin. iterations per non-lin. it.	c_d 4.9561	c_l 0.02108
FD							
1 step	53.0	0.80	0.04	1335	209 - 556 - 322 - 191 - 57	4.9561	0.0210
2 steps	21.8	0.80	0.04	231	105 - 65 - 43 - 18	4.9561	0.0210
10 steps	24.0	0.80	0.04	202	92 - 56 - 43 - 11	4.9561	0.0210
ANALYTICAL							
1 step	48.9	0.12	0.04	1309	182 - 556 - 323 - 191 - 57	4.9561	0.0210
2 steps	19.4	0.12	0.04	231	105 - 65 - 43 - 18	4.9561	0.0210
10 steps	21.5	0.12	0.04	202	92 - 56 - 43 - 11	4.9561	0.0210
ADOL-C							
1 step	54.7	0.98	0.21	1309	182 - 556 - 323 - 191 - 57	4.9561	0.02108
2 steps	24.1	0.98	0.21	231	105 - 65 - 43 - 18	4.9561	0.02108
10 steps	27.6	0.98	0.21	202	92 - 56 - 43 - 11	4.9561	0.02108

220x41 cells, N=26,406,

standard ADOL-C

Numerical Results - N=26,406

Untuned FD:

time J = 1198

method	time total	time J	time B	lin. it. total	lin. iterations per non-lin. it.	c_d 4.9561	c_l 0.02108
1 step	53.0	0.80	0.04	1335	209 - 556 - 322 - 191 - 57	4.9561	0.0210
2 steps	21.8	0.80	0.04	231	105 - 65 - 43 - 18	4.9561	0.0210
10 steps	24.0	0.80	0.04	202	92 - 56 - 43 - 11	4.9561	0.0210
ANALYTICAL							
1 step	48.9	0.12	0.04	1309	182 - 556 - 323 - 191 - 57	4.9561	0.0210
2 steps	19.4	0.12	0.04	231	105 - 65 - 43 - 18	4.9561	0.0210
10 steps	21.5	0.12	0.04	202	92 - 56 - 43 - 11	4.9561	0.0210
ADOL-C							
1 step	54.7	0.98	0.21	1309	182 - 556 - 323 - 191 - 57	4.9561	0.02108
2 steps	24.1	0.98	0.21	231	105 - 65 - 43 - 18	4.9561	0.02108
10 steps	27.6	0.98	0.21	202	92 - 56 - 43 - 11	4.9561	0.02108

220x41 cells, N=26,406,

standard ADOL-C

Numerical Results - N=26,406

method	time total	time J	time B	lin. it. total	lin. iterations per non-lin. it.	c_d 4.9561	c_l 0.02108
FD							
1 step	53.0	0.80	0.04	1335	209 - 556 - 322 - 191 - 57	4.9561	0.0210
2 steps	21.8	0.80	0.04	231	105 - 65 - 43 - 18	4.9561	0.0210
10 steps	24.0	0.80	0.04	202	92 - 56 - 43 - 11	4.9561	0.0210
ANALYTICAL							
1 step	48.9	0.12	0.04	1309	182 - 556 - 323 - 191 - 57	4.9561	0.0210
2 steps	19.4	0.12	0.04	231	105 - 65 - 43 - 18	4.9561	0.0210
10 steps	21.5	0.12	0.04	202	92 - 56 - 43 - 11	4.9561	0.0210
ADOL-C							
1 step	54.7	0.98	0.21	1309	182 - 556 - 323 - 191 - 57	4.9561	0.02108
2 steps	24.1	0.98	0.21	231	105 - 65 - 43 - 18	4.9561	0.02108
10 steps	27.6	0.98	0.21	202	92 - 56 - 43 - 11	4.9561	0.02108

220x41 cells, N=26,406,

standard ADOL-C

Numerical Results - N=106,622

method	time total	time J	time B	lin. it total	lin. iterations per non-lin. it.	c_d 4.7094	c_l 0.01107
FD							
2 steps	177.3	3.44	0.14	606	324 - 146 - 112 - 24	4.7094	0.01108
10 steps	173.4	3.44	0.14	500	223 - 147 - 102 - 28	4.7094	0.01108
100 steps	201.9	3.44	0.14	425	178 - 138 - 103 - 6	4.7094	0.01107
ANALYTICAL							
2 steps	162.9	0.49	0.14	590	306 - 147 - 103 - 24	4.7094	0.01108
10 steps	160.9	0.49	0.14	500	223 - 147 - 102 - 28	4.7094	0.01108
100 steps	194.1	0.49	0.14	474	226 - 139 - 104 - 5	4.7094	0.01107
ADOL-C							
2 steps	178.6	3.72	0.84	590	306 - 147 - 103 - 24	4.7094	0.01108
10 steps	184.8	3.72	0.84	500	223 - 147 - 102 - 28	4.7094	0.01108
100 steps	280.2	3.72	0.84	474	226 - 139 - 104 - 5	4.7094	0.01107

440x82 cells, N=106,622,

standard ADOL-C

Numerical Results - N=240,670

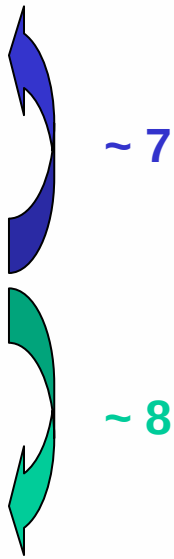
method	time total	time J	time B	lin. it total	lin. iterations per non-lin. it.	c_d 4.6112	c_l 0.00525
FD 2 steps	1310.1	7.45	0.31	3733	2876 - 475 - 328 - 54	4.6112	0.00525
ANALYTICAL 2 steps	1292.3	1.17	0.31	3849	2959 - 473 - 327 - 90	4.6112	0.00526
ADOL-C 2 steps	1318.7	8.74	1.95	3812	2956 - 474 - 328 - 54	4.6112	0.00526

660x123 cells, N=240,670,

standard ADOL-C

Numerical Results - Survey

time J	220	4	440	2.25	660
FD	**0.80**	4.2	**3.44**	2.2	**7.45**
	6.6		7.0		6.4
ANALYTICAL	**0.12**	4.1	**0.49**	2.4	**1.17**
	8.2		7.6		7.5
ADOL-C	**0.98**	3.8	**3.72**	2.3	**8.74**



Numerical Results – AD Tuning

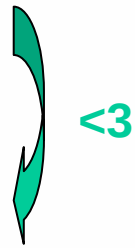
method		runtime for J (normalised to ANALYTICAL 220)		
		220×41 $N = 26,406$	440×82 $N = 106,622$	660×123 $N = 240,670$
FD		5.4	22.2	50.1
ANALYTICAL		1.0	4.0	9.1
ADOL-C	standard	8.2	31.0	72.8
	tapeless	15.9	64.5	144.7
	one-touch	2.7	10.8	24.8
cppAD	standard	18.2	71.1	435.5
	tuned allocation	42.8	193.2	436.6

Template-based evaluation

- No source code changes for different AD tools
- Particular AD datatype in 1 method only!

Numerical Results – AD Tuning

method		runtime for J (normalised to ANALYTICAL 220)		
		220×41 $N = 26,406$	440×82 $N = 106,622$	660×123 $N = 240,670$
FD		5.4	22.2	50.1
ANALYTICAL		1.0	4.0	9.1
ADOL-C	standard	8.2	31.0	72.8
	tapeless	15.9	64.5	144.7
	one-touch	2.7	10.8	24.8
cppAD	standard	18.2	71.1	435.5
	tuned allocation	42.8	193.2	436.6



Template-based evaluation

- No source code changes for different AD tools
- Particular AD datatype in 1 method only!

Jacobian - ADOL-C one-touch

```
void peano::fluid::CalculateJacobian::calculateAdolCContributions(
  const double     uInput[8],
  const double&    p
) {
  if (!_hasTapeAlreadyBeenUsed) { //record tape first time
    // creating calculation arrays and variables
    std::vector<adouble> u(9), div_result(1), F_result(8), pressure_result(8);
    ...
    trace_on(1);// ##############################
    for (int i=0; i<8; i++) u[i] <<= uInput[i];
    u[8] <<= p;
    // collecting contributions
    _calculatePressureGradient.accumulatePressureGradientValues(h, u[8], pressure_result);
    _calculateF.accumulateFGenericValues(u, F_result);
    _calculateDivergence.calculateCellDivergenceCorrectionRightHandSide(u, div_result);
    // setting up B
    div_result[0] >>= last_result;
    ...
    trace_off();// ##############################
    _hasTapeAlreadyBeenUsed = true;
  }

  double** jacobmatrix = new double ...;
  double *working_data = new double[9]; working_data = uInput; ...
  jacobian(1,9,9,working_data,jacobmatrix);

  //clear local data
  ...
}
```

Jacobian - ADOL-C one-touch

```
void peano::fluid::CalculateJacobian::calculateAdolCContributions(
  const double     uInput[8],
  const double&    p
) {
  if (!_hasTapeAlreadyBeenUsed) { //record tape first time
    // creating calculation arrays and variables
    std::vector<adouble> u(9), div_result(1), F_result(8), pressure_result(8);
    ...
    trace_on(1);// ##############################
    for (int i=0; i<8; i++) u[i] <<= uInput[i];
    u[8] <<= p;
    // collecting contributions
    _calculatePressureGradient.accumulatePressureGradientValues(h, u[8], pressure_result);
    _calculateF.accumulateFGenericValues(u, F_result);
    _calculateDivergence.calculateCellDivergenceCorrectionRightHandSide(u, div_result);
    // setting up B
    div_result[0] >>= last_result;
    ...
    trace_off();// ##############################
    _hasTapeAlreadyBeenUsed = true;
  }

  double** jacobmatrix = new double ...;
  double *working_data = new double[9]; working_data = uInput; ...
  jacobian(1,9,9,working_data,jacobmatrix);

  //clear local data
  ...
}
```

Jacobian - ADOL-C one-touch

- Template-based evaluation
- one-touch works (no if etc.)

```
template <class DataType, class DataTypeContainer>
void peano::fluid::CalculatePressureGradient::accumulatePressureGradientValues(
  const Vector&      h,
  const DataType&    p,
  DataTypeContainer& result
) {
  Vector hFactor = _elementMatrices.getHFactorForMOrC(h);

  for (int i=0; i<NO_VERTICES_PER_ELEMENT; i++) {
    for (int d=0; d<DIMENSIONS; d++) {
      result[i*DIMENSIONS + d] = -MElementMatrix[i+(d*NO_VERTICES_PER_ELEMENT)]
                                 * p * hFactor(d) / _rho;
    }
  }
}
```

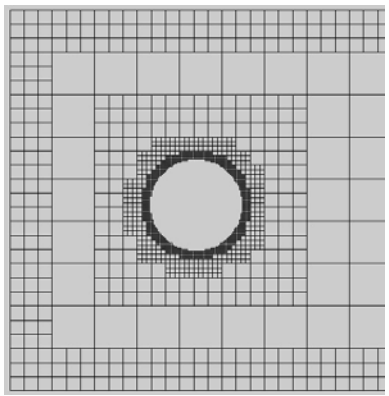
Conclusion

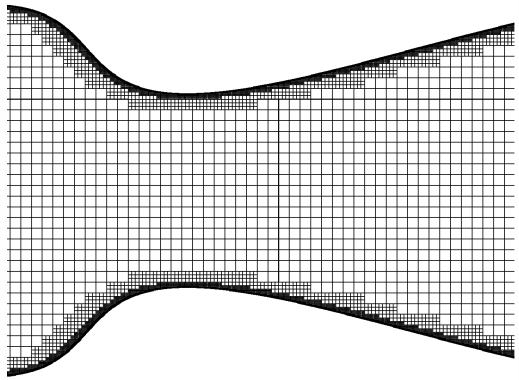
Criterion	FD	ANALYTICAL	AD
Easy to implement	0	0	0
No manual choice of ε	-	+	+
Runtime Jacobian	-	+	0+
Runtime B / pseudo-ts (templated)	+	+	+
Direct usability for other discretisations (without code duplication)	0	-	+

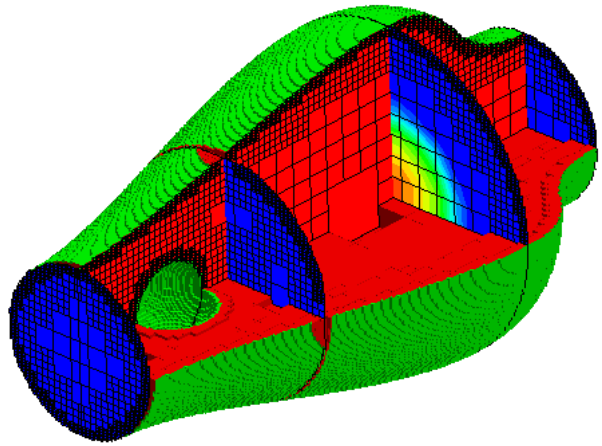
Outlook

- Matrix-free
  - All 3 approaches for Jacobian usable
  - Element-wise matrix-vector multiplication
  - No preconditioning ➔ use built-in geometric multigrid when available

- Adaptive grids



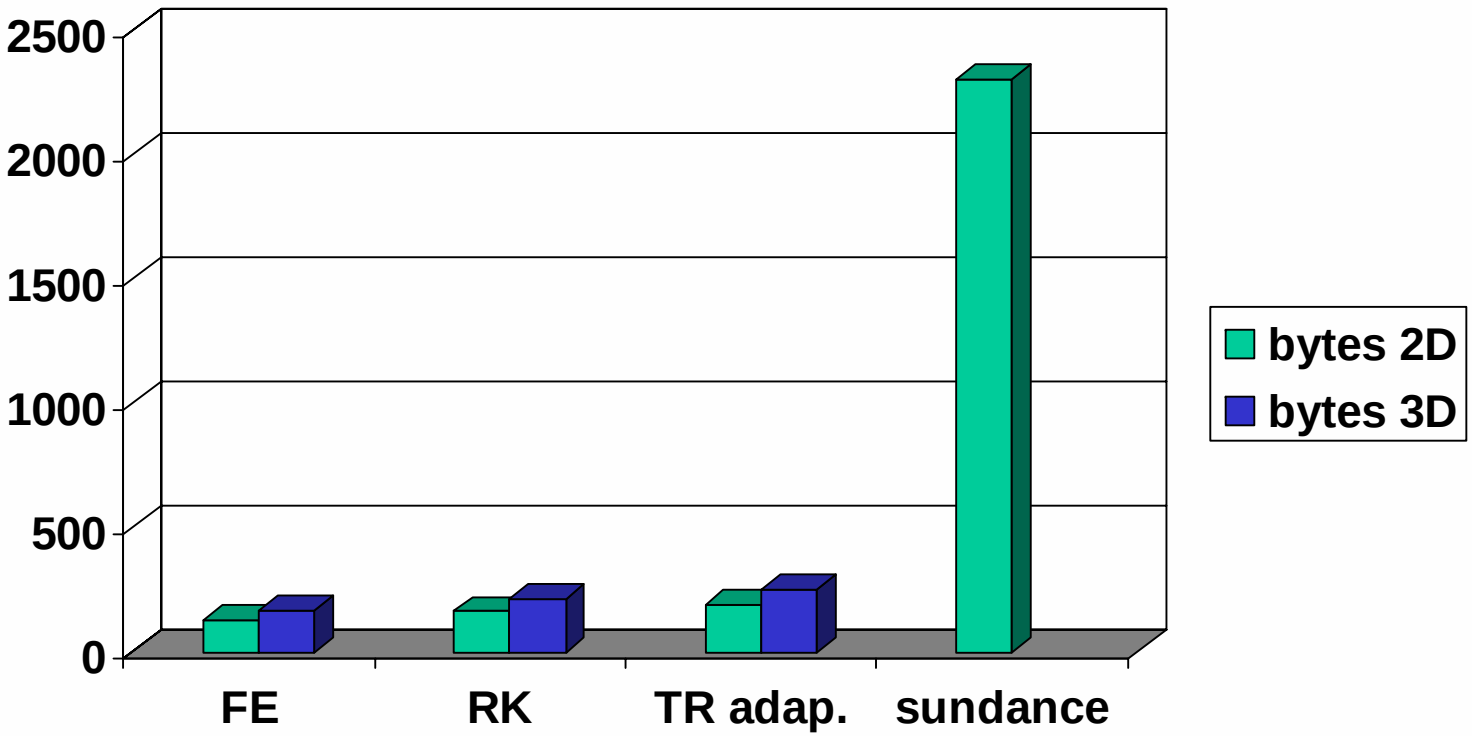




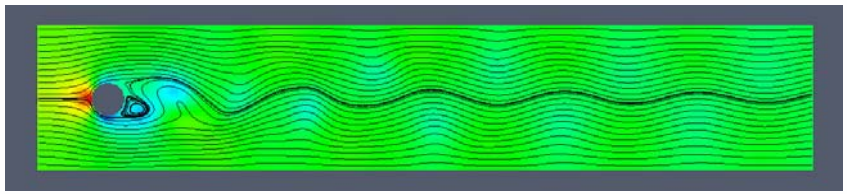
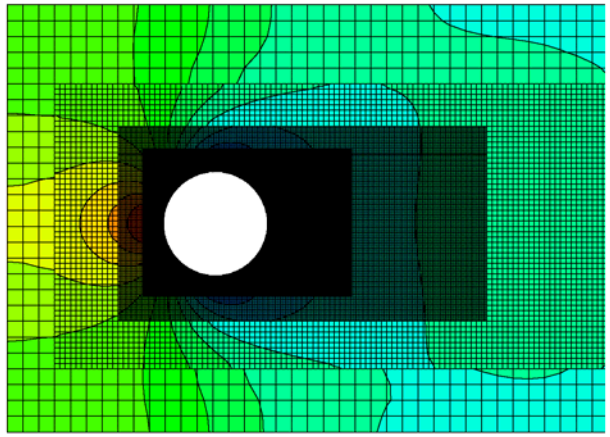
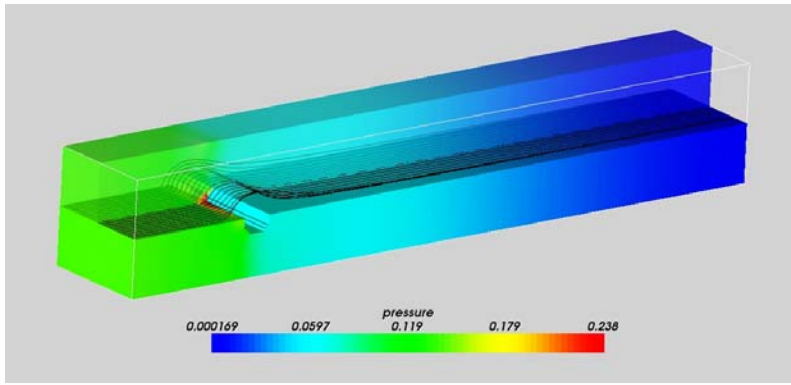
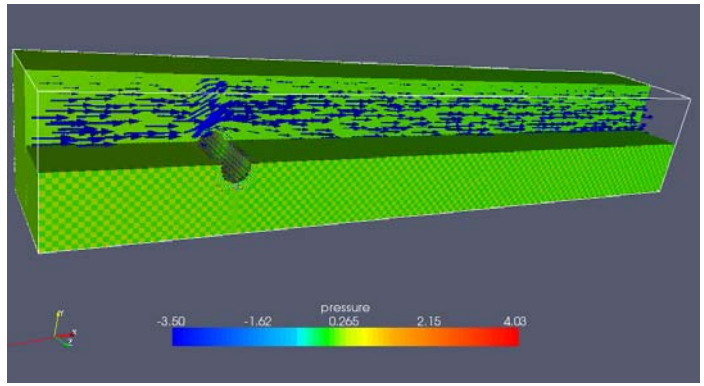
Thanks for your attention!

Backup I

Low memory requirements (FEM + adap.):

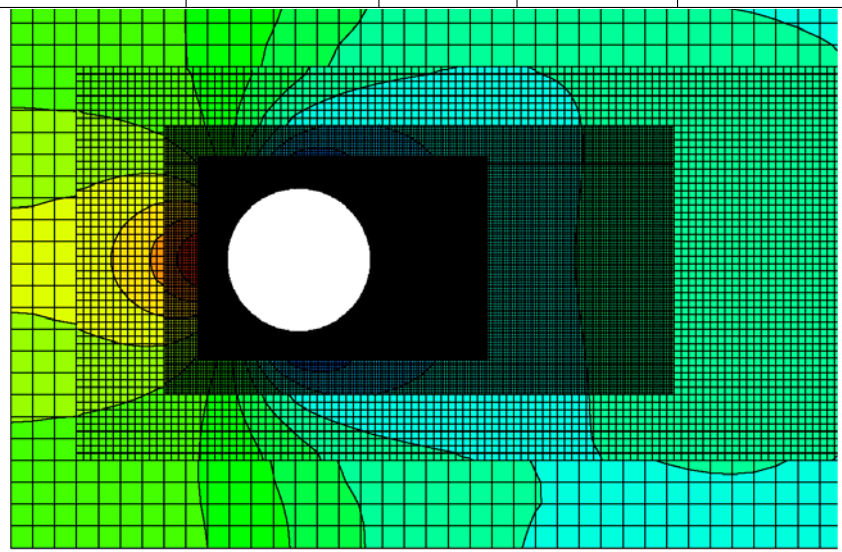


Backup II - Numerical Results (FEM)

	regular	adaptive
2D		
3D		

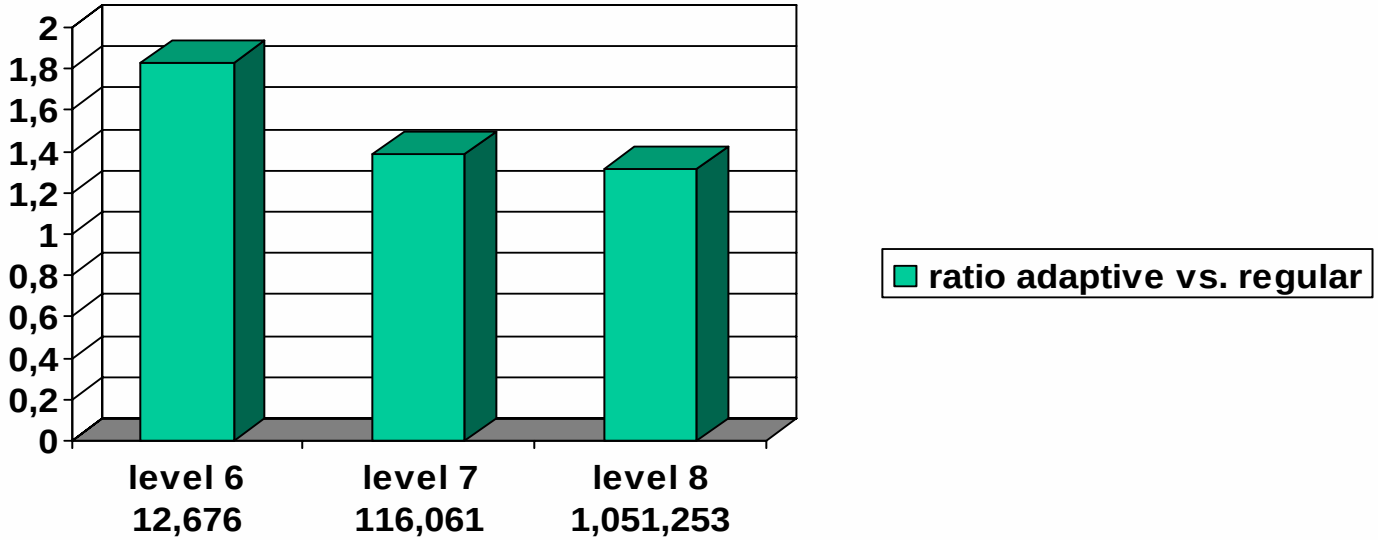
Numerical Results - FEM

max. level	min. level	#total DoF	c_d	c_l	CPU time per time step
8	6 (box)	88857	5.680	0.0150	0.71
9	7	125041	5.591	0.0113	1.03
9	8	1057877	5.561	0.0112	9.46
9	6 (box)	261501	5.586	0.0115	2.46
ref. data		–	5.580	0.0107	–



Numerical Results - Performance

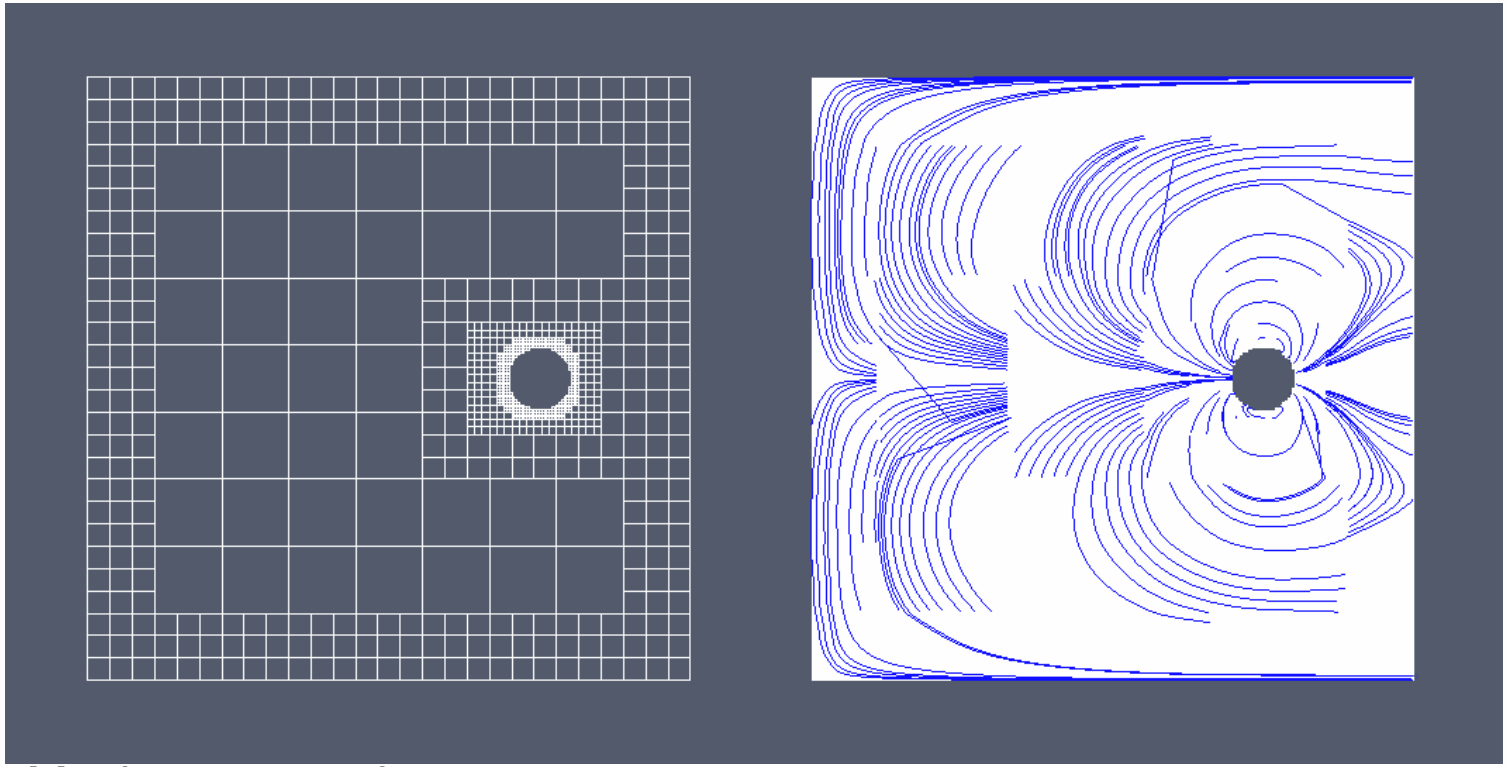
2D FEM



3D FEM — Overhead adaptive vs. regular **< 3%**

2D IDO — Overhead Peano vs. Aoki (regular): **1.3 – 4.4**

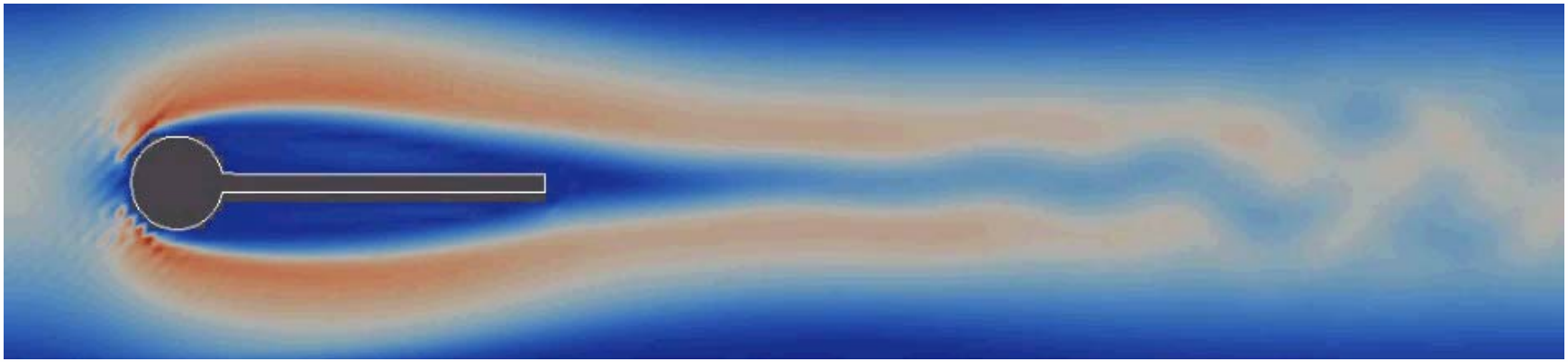
CFD Extensions



- Moving geometries
  - Update of data + grid (regular + adaptive)
  - Divergence correction

joint work with Kristof Unterweger

CFD Extensions



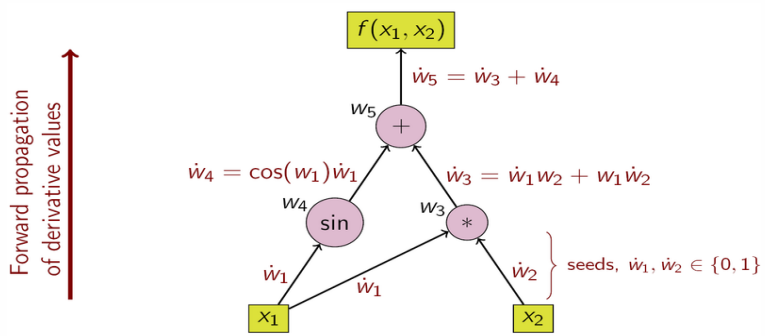
joint work with Janos Benk, Bernhard Gatzhammer, Miriam Mehl, Kristof Unterweger, and Tobias Weinzierl

- Moving geometries
  - Update of data + grid (regular + adaptive)
  - Divergence correction

Backup I - Automatic Differentiation

- Apply chaine rule for derivatives computationally
  - Example: Forward accumulation $f(x_1, x_2) = x_1 \cdot x_2 + \sin(x_1)$

Original code statements	Added statements for derivatives
$w_1 = x_1$	$w'_1 = 1$ (seed)
$w_2 = x_2$	$w'_2 = 0$ (seed)
$w_3 = w_1 w_2$	$w'_3 = w'_1 w_2 + w_1 w'_2 = 1 x_2 + x_1 0 = x_2$
$w_4 = \sin(w_1)$	$w'_4 = \cos(w_1) w'_1 = \cos(x_1) 1$
$w_5 = w_3 + w_4$	$w'_5 = w'_3 + w'_4 = x_2 + \cos(x_1)$



Source: www.wikipedia.org

  - Backward accumulation
  - Mixed
- Realisation in AD tools
  - Source code transformation (function.cpp ➔ diff_function.cpp)
  - Operator overloading (special data type)

Backup I - Automatic Differentiation

Package	Mode	Version	Positive	Negative	misc
ADC	Operator overloading (forward)	unknown	probably efficient	not used widely	commercial
ADIC	Source transformation (forward)	Jun 2005	-	Win32 not supported community dead bad documenation	registration necessary
ADOL-C	**Operator overloading (forward,backward)**	**Aug 2006**	**all OS supported good documentation many features (Jacobian, sparsity) active community**	**-**	**by TU Dresden**
COSY INFINITY	Operator overloading (forward)	2006	all OS supported good documenation suitable for high-dim sparsity features	no community free for private use	registration necessary
CppAD	Operator overloading (forward, backward)	Mar 2008	all OS supproted good documentation integrated speed tests sparsity features	-	-
FAD	Operator overloading (forward)	2002	-	Win32 not supported community dead bad documentation	-
FADBAD++	Operator overloading (forward, backward)	unknown	all OS supported documentation ok	no community little information	-
FFADLib	Operator overloading (forward)	unknown	-	Unix not supported	registration necessary
OpenAD	Source Transformation (forward, backward)	Nov 2006	good documentation	Win32 not supported uses external libs	used by NASA
YAO	unknown (forward, backward)	unknwon	-	Win32 not supported no information	website in French

www.autodiff.org