

# Steady-state Flow Simulations using Exact Jacobians on Cartesian Grids

Hans-Joachim Bungartz, Miriam Mehl, Tobias Neckel

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

Africomp09, Sun City, January 07, 2009

# Outline

- Introduction
- Different Approaches for Jacobians
  - Finite Differences
  - Automatic Differentiation
  - Analytic Differentiation
- Numerical Results
- Outlook

# Introduction

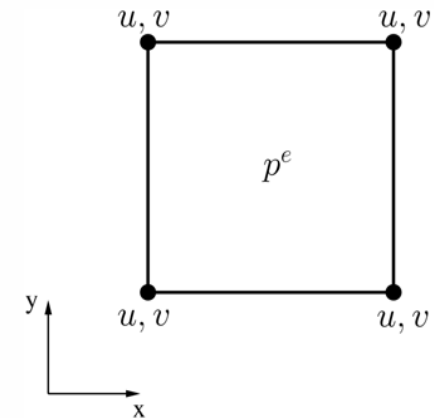
- Incompressible Navier-Stokes equations (NSE)

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

- Discretisation in Peano framework

- Low-order FEM (Q1P0)
- Steady-state: preparation for implicit time integration
- Discrete NSE:

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



# Introduction

- 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)

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

- Initial value
  - Crucial for convergence
  - Pseudo-timestepping (semi-implicit, PPE)

# Introduction

- 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)

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

- Initial value
  - Crucial for convergence
  - Pseudo-timestepping (semi-implicit, PPE)

# Jacobian - Finite Differences

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

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

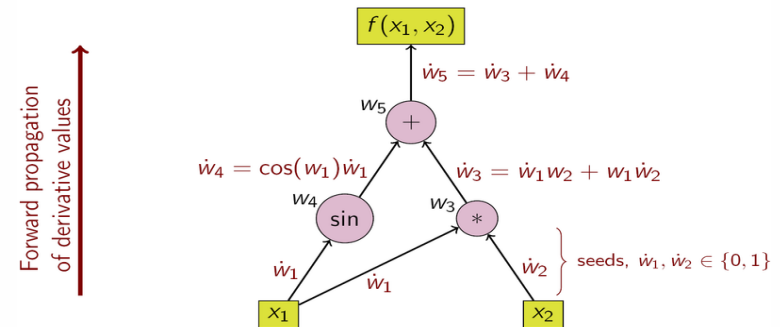
$$\varepsilon = \begin{cases} 10^{-8} \cdot u_i & \text{if } |u_i| > 10^{-8} \\ 10^{-8} \cdot 10^{-8} \cdot \text{sign}(u_i) & \text{otherwise} \end{cases}$$

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

# Jacobian - 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](http://www.wikipedia.org)

- Backward accumulation
- Mixed
- Realisation in AD tools
  - Source code transformation (function.cpp  $\rightarrow$  diff\_function.cpp)
  - Operator overloading (special data type)

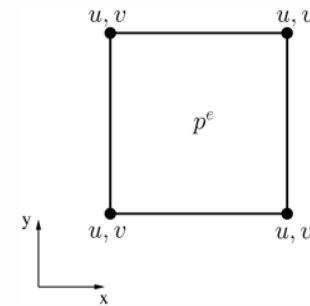
# Jacobian - Automatic Differentiation

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
<b>ADOL-C</b>	<b>Operator overloading (forward,backward)</b>	<b>Aug 2006</b>	<b>all OS supported good documentation many features (Jacobian, sparsity) active community</b>	-	<b>by TU Dresden</b>
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](http://www.autodiff.org)

# Jacobian – Analytic Differentiation

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

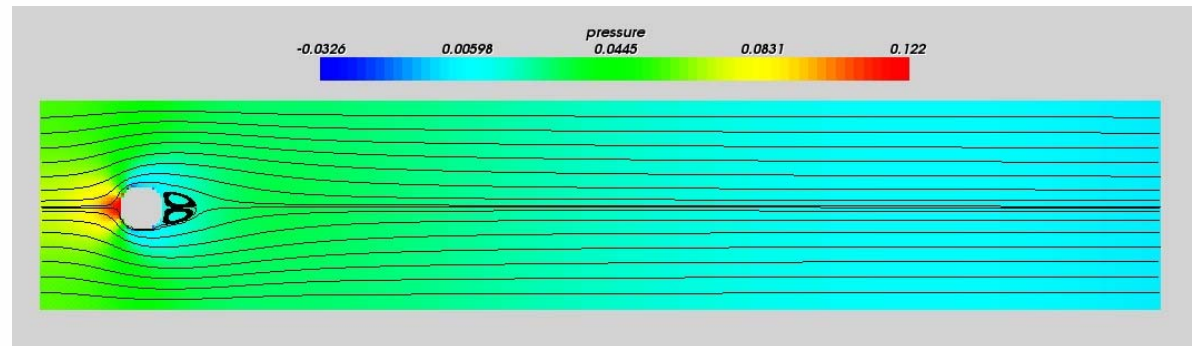


- Block 1: linear
- Block 2: linear
- Block 3: non-linear

$$J^e = \begin{pmatrix} & \text{Block 3} & \text{Block 2} \\ & \frac{\partial F}{\partial \mathbf{u}_h} & \frac{\partial \nabla_h}{\partial p_h} \\ \text{Block 1} & & 0 \\ \frac{\partial \nabla_h}{\partial \mathbf{u}_h} & & \end{pmatrix} \in \mathbf{R}^{9 \times 9}$$

# Numerical Results

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



- PETSc solver
  - Explicit assembly 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 - I

compiled w/o ADOL-C	time total	time $J$	time $B$	lin. it. total	lin. iterations per non-lin. it.	$c_d$ 4.9561	$c_t$ 0.0210
<b>FD</b>							
1 step	53.0	0.8	0.04	1335	209 - 556 - 322 - 191 - 57	4.9561	0.0210
2 steps	21.8	0.8	0.04	231	105 - 65 - 43 - 18	4.9561	0.0210
10 steps	24.0	0.8	0.04	202	92 - 56 - 43 - 11	4.9561	0.0210
<b>ANALYTICAL</b>							
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

**220x41****N=26,406**

## Numerical Results - I

Untuned FD:  
Time J = 1198

compiled w/o ADOL-C	time total	time $J$	time $B$	lin. it. total	lin. iterations per non-lin. it.	$c_d$	$c_t$
FD						4.9561	0.0210
1 step	53.0	0.8	0.04	1335	209 - 556 - 322 - 191 - 57	4.9561	0.0210
2 steps	21.8	0.8	0.04	231	105 - 65 - 43 - 18	4.9561	0.0210
10 steps	24.0	0.8	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

220x41

N=26,406

# Numerical Results - I

compiled w/o ADOL-C	time total	time $J$	time $B$	lin. it. total	lin. iterations per non-lin. it.	$c_d$	$c_t$
<b>FD</b>							
1 step	53.0	0.8	0.04	1335	209 - 556 - 322 - 191 - 57	4.9561	0.0210
2 steps	21.8	0.8	0.04	231	105 - 65 - 43 - 18	4.9561	0.0210
10 steps	24.0	0.8	0.04	202	92 - 56 - 43 - 11	4.9561	0.0210
<b>ANALYTICAL</b>							
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

**220x41****N=26,406**

compiled with ADOL-C	time total	time $J$	time $B$	lin. it. total	lin. iterations per non-lin. it.	$c_d$	$c_t$
<b>FD</b>							
1 step	83.0	6.45	0.21	1337	211 - 556 - 322 - 191 - 57	4.9561	0.02108
2 steps	45.6	6.45	0.21	231	105 - 65 - 43 - 18	4.9561	0.02108
10 steps	49.8	6.45	0.21	202	92 - 56 - 43 - 11	4.9561	0.02108
<b>ANALYTICAL</b>							
1 step	50.3	0.12	0.21	1309	182 - 556 - 323 - 191 - 57	4.9561	0.02108
2 steps	20.5	0.12	0.21	231	105 - 65 - 43 - 18	4.9561	0.02108
10 steps	24.0	0.12	0.21	202	92 - 56 - 43 - 11	4.9561	0.02108
<b>ADOL-C</b>							
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

**ADOL-C  
datatype****(operator  
overloading)**

## Numerical Results - I

220x41

N=26,406

compiled w/o ADOL-C	time total	time $J$	time $B$	lin. it. total	lin. iterations per non-lin. it.	$c_d$	$c_t$
<b>FD</b>							
1 step	53.0	0.8	0.04	1335	209 - 556 - 322 - 191 - 57	4.9561	0.0210
2 steps	21.8	0.8	0.04	231	105 - 65 - 43 - 18	4.9561	0.0210
10 steps	24.0	0.8	0.04	202	92 - 56 - 43 - 11	4.9561	0.0210
<b>ANALYTICAL</b>							
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

compiled with ADOL-C	time total	time $J$	time $B$	lin. it. total	lin. iterations per non-lin. it.	$c_d$	$c_t$
<b>FD</b>							
1 step	83.0	6.45	0.21	1337	211 - 556 - 322 - 191 - 57	4.9561	0.02108
2 steps	45.6	6.45	0.21	231	105 - 65 - 43 - 18	4.9561	0.02108
10 steps	49.8	6.45	0.21	202	92 - 56 - 43 - 11	4.9561	0.02108
<b>ANALYTICAL</b>							
1 step	50.3	0.12	0.21	1309	182 - 556 - 323 - 191 - 57	4.9561	0.02108
2 steps	20.5	0.12	0.21	231	105 - 65 - 43 - 18	4.9561	0.02108
10 steps	24.0	0.12	0.21	202	92 - 56 - 43 - 11	4.9561	0.02108
<b>ADOL-C</b>							
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

## Numerical Results - II

compiled w/o ADOL-C	time total	time $J$	time $B$	lin. it. total	lin. iterations per non-lin. it.	$c_d$	$c_l$
<b>FD</b>						4.7094	0.01107
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
<b>ANALYTICAL</b>							
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

**440x82**  
**N=106,622**

compiled with ADOL-C	time total	time $J$	time $B$	lin. it. total	lin. iterations per non-lin. it.	$c_d$	$c_l$
<b>FD</b>						4.7094	0.01107
2 steps	266.8	25.95	0.84	577	293 - 147 - 113 - 24	4.7094	0.01108
10 steps	274.0	25.95	0.84	500	223 - 147 - 102 - 28	4.7094	0.01108
100 steps	362.9	25.95	0.84	425	178 - 138 - 103 - 6	4.7094	0.01107
<b>ANALYTICAL</b>							
2 steps	165.6	0.49	0.84	590	306 - 147 - 103 - 24	4.7094	0.01108
10 steps	170.7	0.49	0.84	500	223 - 147 - 102 - 28	4.7094	0.01108
100 steps	268.4	0.49	0.84	474	226 - 139 - 104 - 5	4.7094	0.01107
<b>ADOL-C</b>							
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

**ADOL-C**  
**datatype**  
**(operator overloading)**

## Numerical Results - III

compiled with ADOL-C	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

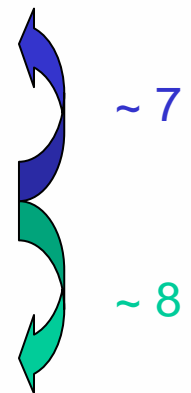
**660x123**  
**N=240,670**

## Numerical Results - III

compiled with ADOL-C	time total	time $J$	time $B$	lin. it total	lin. iterations per non-lin. it.	$c_d$	$c_l$
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**  
**N=240,670**

<b>time J</b>	220	4	440	2.25	660
FD	<b>0.8</b>	4.2	<b>3.4</b>	2.2	<b>7.45</b>
	6.6		7		6.4
ANALYTICAL	<b>0.12</b>	4	<b>0.49</b>	2.4	<b>1.17</b>
	8		7.5		7.5
ADOL-C	<b>0.98</b>	3.8	<b>3.72</b>	2.3	<b>8.74</b>

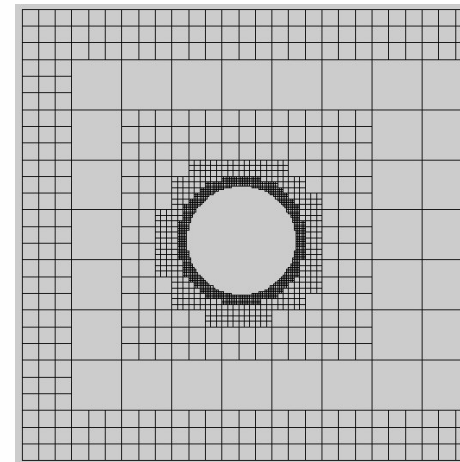


## Conclusion

Criterion	FD	ADOLC	ANALYTICAL
Easy to implement	0	0	0
No code duplication	-	+	-
No manual choice of $\varepsilon$	-	+	+
Less non-lin. its	0	+?	+?
Less linear its	0	+?	+?
Runtime Jacobian	-	-	+
Runtime B (+ pseudo-timestepping)	+	--	+

# Outlook

- ADOL-C tapeless
  - Faster
  - N fixed at compile time
- Matrix-free
  - All 3 approaches for Jacobian usable
  - Element-wise matrix-vector multiplication
  - No preconditioning → use built-in geometric multigrid when available
- 3D
  - Implementation completed (all 3 approaches)
  - Convergence problems (ILU)
- Implicit time integration
  - Euler
  - Trapezoidal rule
- Adaptive grids



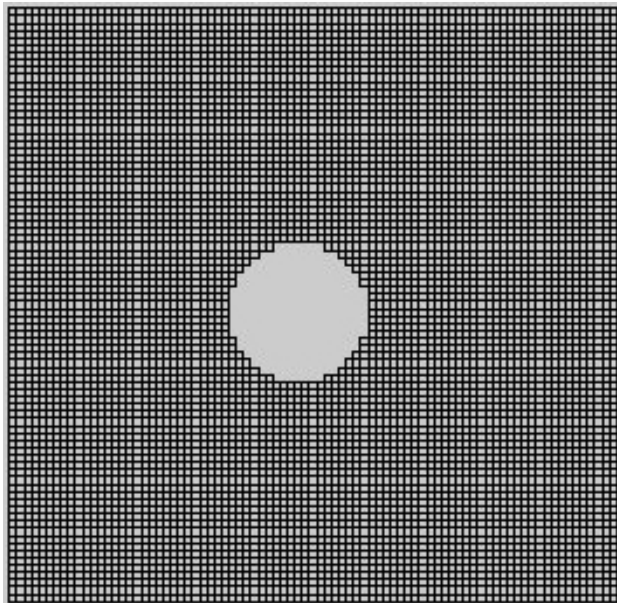
**Thanks for your attention!**



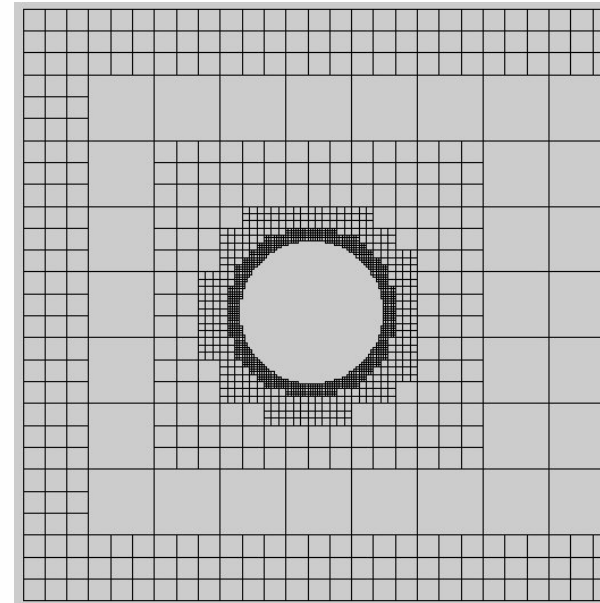
## Backup - Peano

- Framework for different applications (Poisson, NSE, etc.): “plug-in concept”
- Cartesian grids + cell-wise operator evaluation

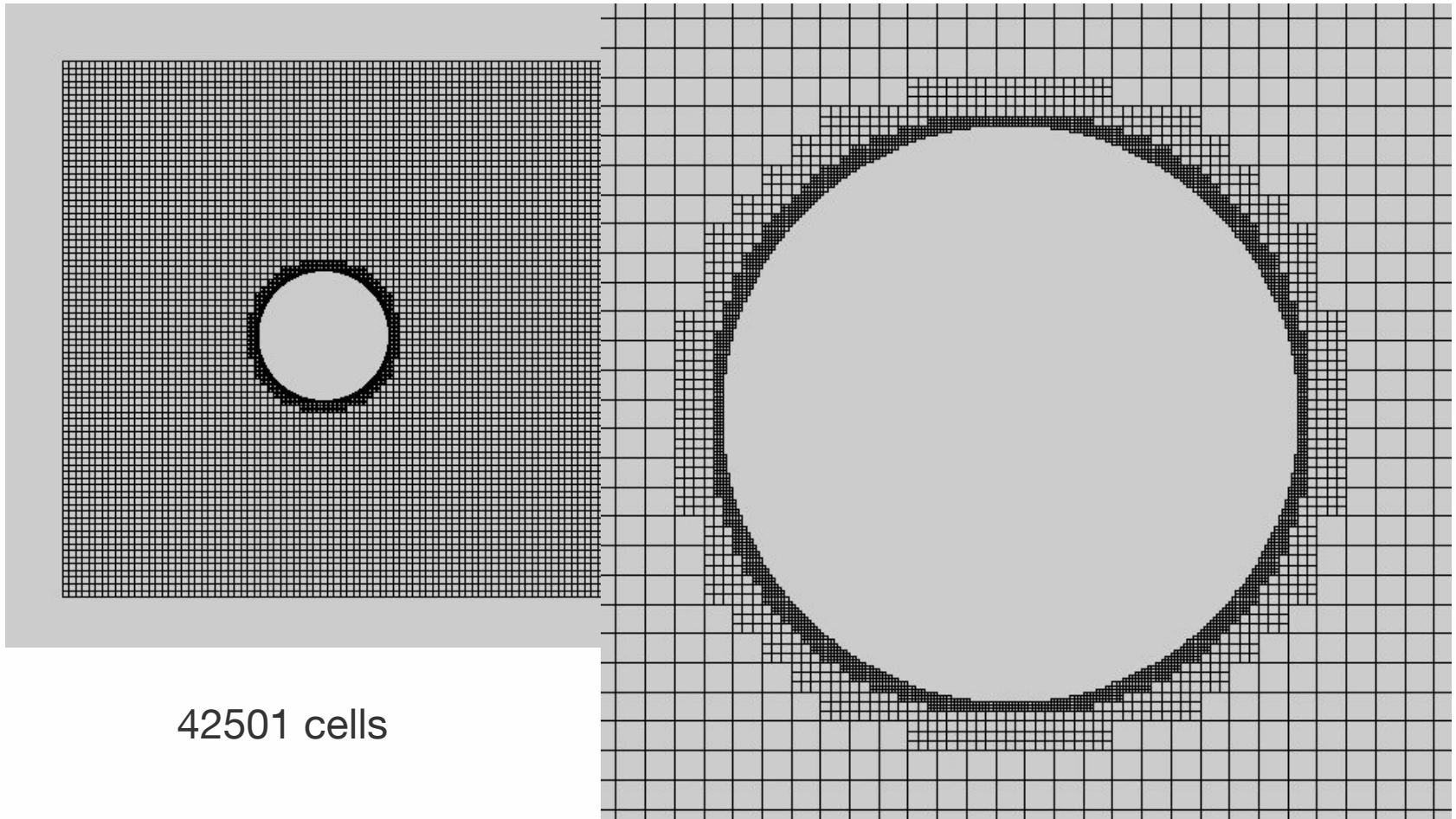
Regular (lexicographic)



Adaptive (spacetree)

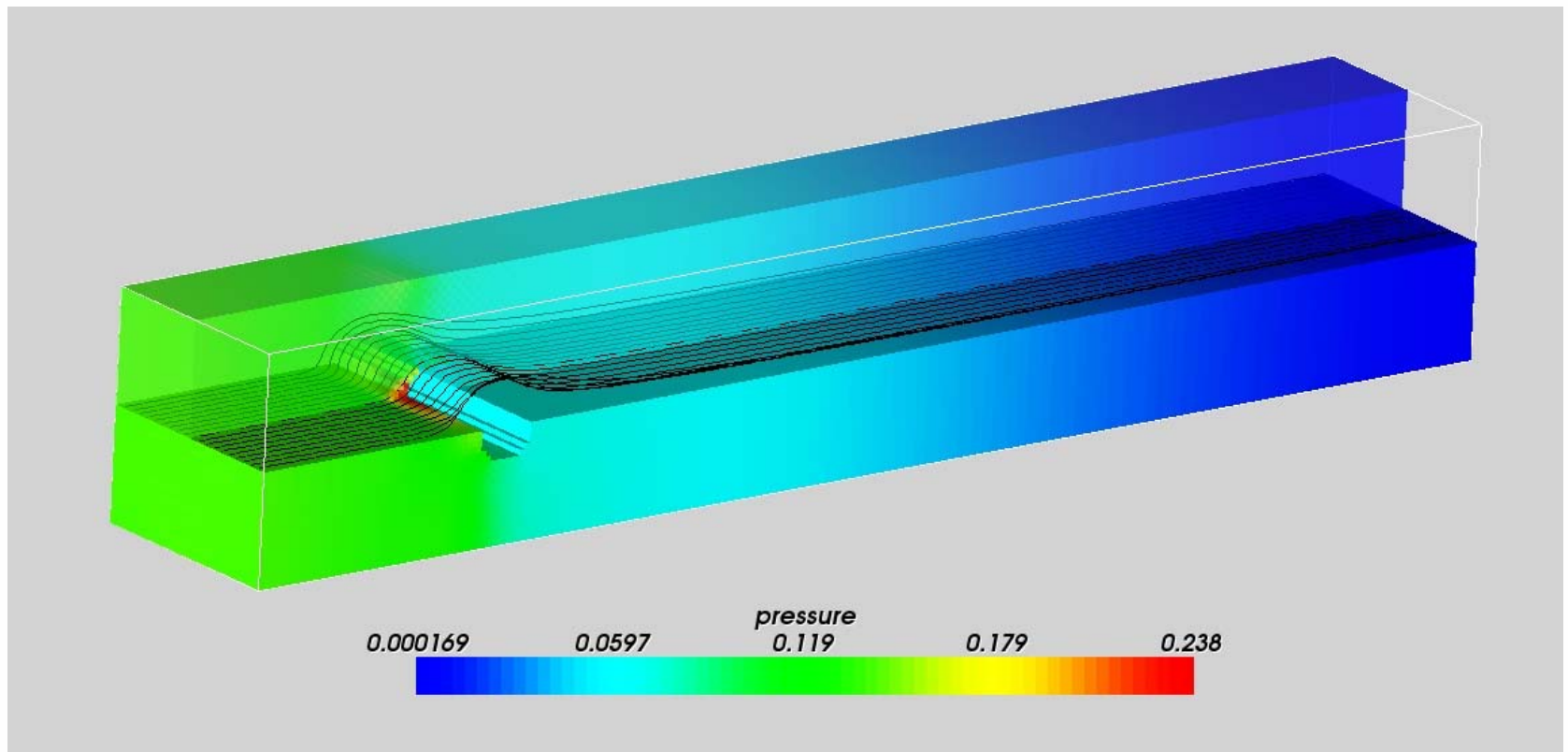


## Backup – Adaptive Grid



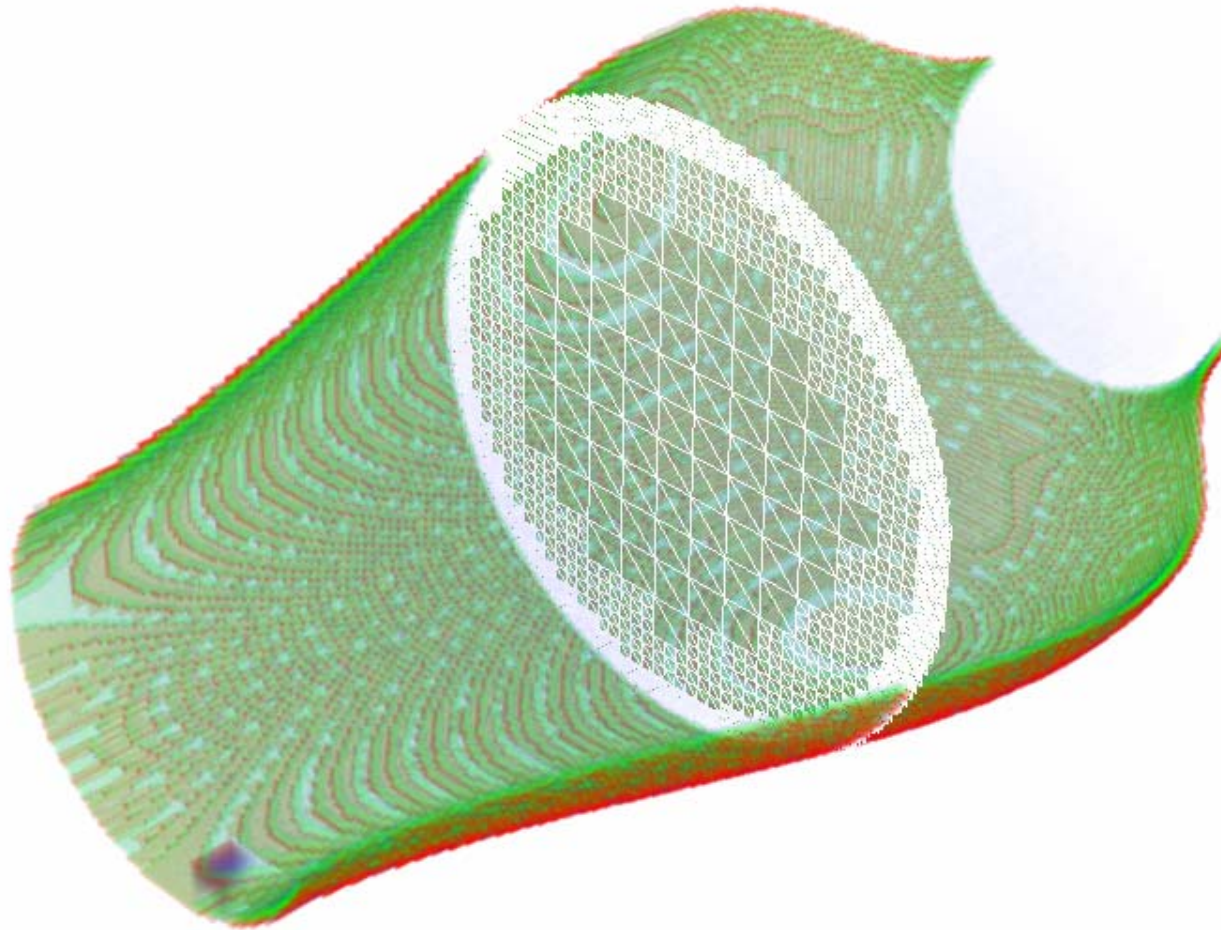
42501 cells

## Backup II – Cylinder Benchmark 3D



416642 cells

## Backup III – 3D adaptive Drift-Ratchet



# Backup IV – Peano Architecture

