Hardware-oriented Numerics

for PDEs

Consequences for

Incompressible Flow Solvers

S. Turek + FEAST GROUP

M. Altieri, Chr. Becker, S. Buijssen, S. Kilian, D. Kuzmin, F. Platte
J. Hron, P. Kotalik, W. Decheng, A. Ouazzi, R. Schmachtel, L. Rivkind...

Institut für Angewandte Mathematik & Numerik
Universität Dortmund

`http://www.featflow.de`

Trends in Numerics for PDEs:

'A posteriori error control/adaptive meshing'

'Iterative (parallel) solution strategies'

'Operator-splitting for coupled problems'

Reduction of numerical complexity !!!

Trends in Processor Technology:

'Enormous improvements in **Processing Data***'*

'Much lower advances in **Moving Data***'*

1 PC in 10 years $\approx$ 1 CRAY T3E today !!!

Questions:

'Can we use this enormous computing power ?'

'Particularly: *For Numerics for PDEs ???'*

'If **not**, *how to achieve a significant percentage ?'*

Main components in iterative schemes:

Matrix-Vector applications (MV)

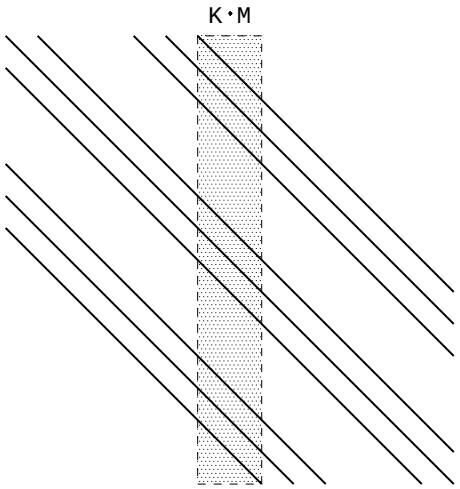
- Krylov-space methods, **Multigrid**, etc.
- **Smoothing**, defect calculations, step-length control, etc.
- Often consuming (at least) **60 - 90%** of CPU time
- **How many MFLOP/s are realistic ?**

Sparse Matrix-Vector techniques

```
      DO 10 IROW=1,N
      DO 10 ICOL=KLD(IROW),KLD(IROW+1)-1
10     Y(IROW)=DA(ICOL)*X(KCOL(ICOL))+Y(IROW)
```

→ Standard technique in most **FEM** or **FV** codes
→ Storage of '**non-zero**' matrix elements only (*CSR*, *CSC*, ...)
→ Access via **index vectors, linked lists, pointers**, etc
→ Comparable with '**indexed DAXPY**'

Sparse Banded MV techniques



→ Typical for **FD** codes on '**tensorproduct meshes**'
→ Storage of '**non-zero**' elements in **bands/diagonals**
→ MV multiplication '**bandwise**' and '**window-oriented**'

Results on locally structured meshes

2D case	N	DAXPY-I	SBB-V	SBB-C	**MG-V**	**MG-C**
DEC 21264	65^2	205 (178)	538	795	370	452
(667 MHz)	257^2	224 (110)	358	1010	314	487
'**ES40**'	1025^2	**78 (11)**	**158**	**813**	**185**	**401**
HITACHI	65^2	173 (82)	238	391	191	266
(375 MHz)	257^2	143 (29)	243	388	198	260
'**SR8000**'	1025^2	**144 (7)**	**226**	**390**	**200**	**267**
AMD K7	65^2	203 (195)	101	556	122	355
(850 MHz)	257^2	29 (27)	78	241	72	166
'**ATHLON**'	1025^2	**31 (10)**	**64**	**236**	**58**	**126**

Sparse **MV techniques (DAXPY-I)**

- MFLOP/s rates far away from '**Peak Performance**'
- Depending on **problem size + numbering**
- PC partially **faster** than processors in 'supercomputers' !!!

Sparse Banded **MV techniques (SBB)**

- 'Supercomputing' power gets visible (up to **1 GFLOP/s**)
- FEM-Simulation for **complex domains**???

FEAST project

Implementation techniques and
Numerics (!) adapted to **hardware !!!**

$\Uparrow$

Precise knowledge of **processor** characteristics
for different tasks in **MG** for **FEM** spaces:

(Collection of) **FEAST INDICES**

$\Downarrow$

1) Get the optimum (from the hardware)!!!

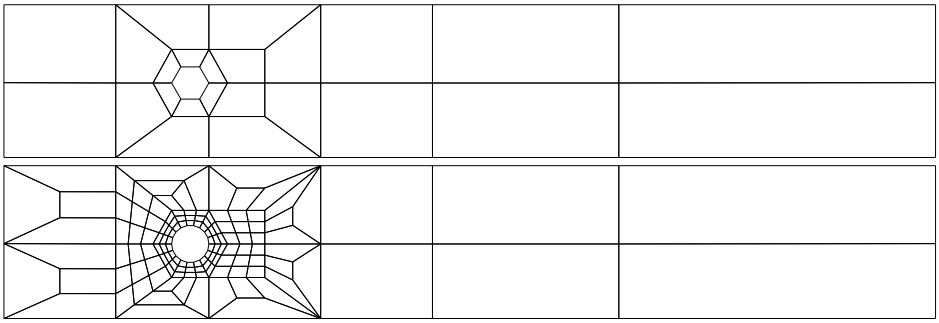
2) Prevent from dramatic performance losses !!!

$\Uparrow$

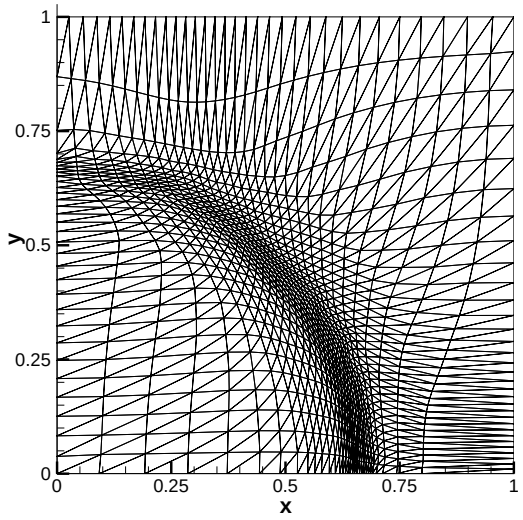
Hardware-oriented Numerics ???

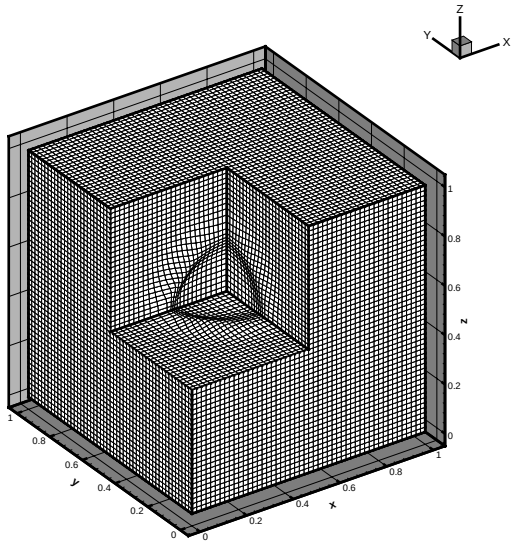
1) Concepts for adaptive meshing

1) macro-oriented adaptivity



2) patchwise adaptivity





3) local adaptivity

$\rightarrow$ hanging nodes, triangles/quads,...

'Exploit the locally nice structures !'

- Local use of efficient SPARSE BANDED BLAS techniques
- Local superconvergence through orthogonal meshes
- Local storage cost small through the use of matrix stencils

$\Uparrow$

Open problems:

1. **'Optimality' of the mesh ?**
   $\rightarrow$ *w.r.t. number of unknowns or CPU time ?*
2. **Error estimators ?**
   $\rightarrow$ *degree of macro-refinement ?*
   $\rightarrow$ *anisotropic refinement ?*
   $\rightarrow$ *h/p-refinement ?*

II) Concepts for iterative (parallel) solvers

1) Standard Multigrid

- parallelization of 'recursive' smoothers (only blockwise) ?
- complicated geometric structures with local anisotropies ?
- **too few arithmetic work vs. data exchange !**

2) Standard Domain Decomposition

- good ratio for communication/arithm. work !
- implementation (overlap, coarse grid problem, **3D**) ?
- **bad convergence behaviour w.r.t. multigrid !**

$\Downarrow$

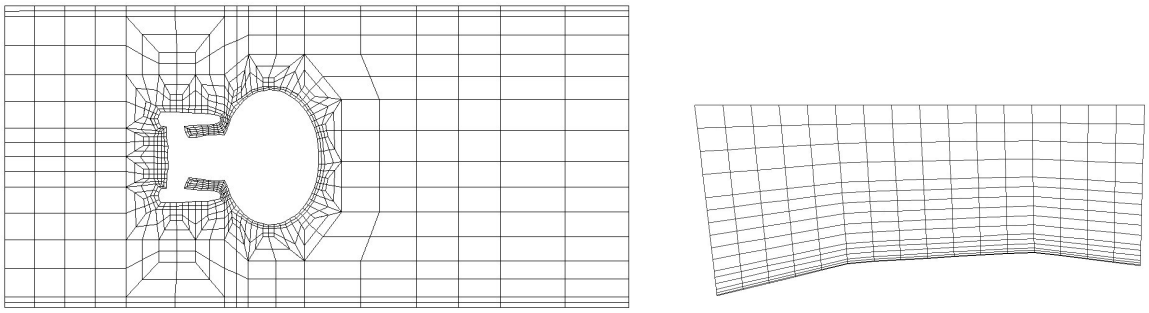
1) + 2) = ScaRC

Hide recursively all anisotropies in more 'local' units! (**robustness**)

Perform all Linear Algebra tasks on 'local' units only! (**efficiency**)

Example: Realization of ScaRC in Feast

2D decomposition and zoomed (macro) element (LEVEL 3) with locally anisotropic refinement towards the wall

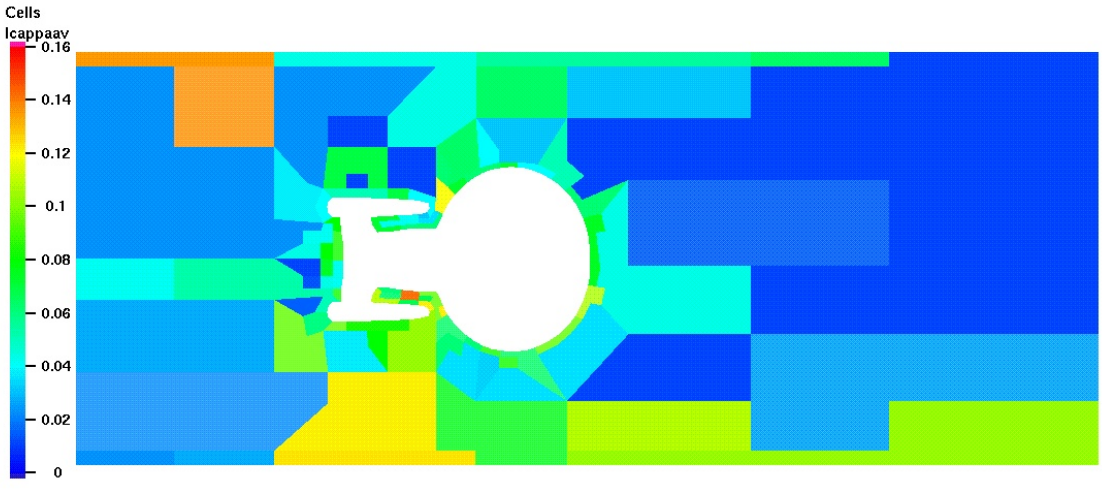


ScaRC-CG solver (smoothing steps: 1 global ScaRC ; 1 local 'MG-TriGS') for locally (an)isotropic refinement

Global (parallel) convergence rates

$\#NEQ$	Dirichlet 'Velocity'		Neumann 'Pressure'	
	$AR \approx 10$	$AR \approx 10^6$	$AR \approx 10$	$AR \approx 10^6$
210,944	0.17 (**8**)	0.18 (**8**)	0.21 (**9**)	0.15 (**8**)
843,776	0.17 (**8**)	0.17 (**8**)	0.20 (**9**)	0.17 (**8**)
3,375,104	0.18 (**9**)	0.19 (**9**)	0.22 (**10**)	0.22 (**10**)
13,500,416	0.19 (**9**)	0.18 (**9**)	0.23 (**10**)	0.23 (**10**)

Local convergence rates (for $AR \approx 10^6$)



III) MPSC: Incompressible Flow Solvers

Results with FeatFlow

Computer	total time	'memory intensive'	'floating point'
IBM SP2 (160 MHz)	**2748**	1587 (**58%**)	1161 (**42%**)
PC PII (400 MHz)	**4927**	2401 (**49%**)	2526 (**51%**)
IBM SP2 (66 MHz)	**5970**	3824 (**64%**)	2146 (**36%**)

- **'Discrete Projection':** Burgers + Pressure Poisson
- Streamline-Diffusion FEM discretization in 3D
- Implicit adaptive time stepping
- *Faster solvers useless !*

How to increase the 'floating point' intensive parts ?

Larger time steps !

How ???

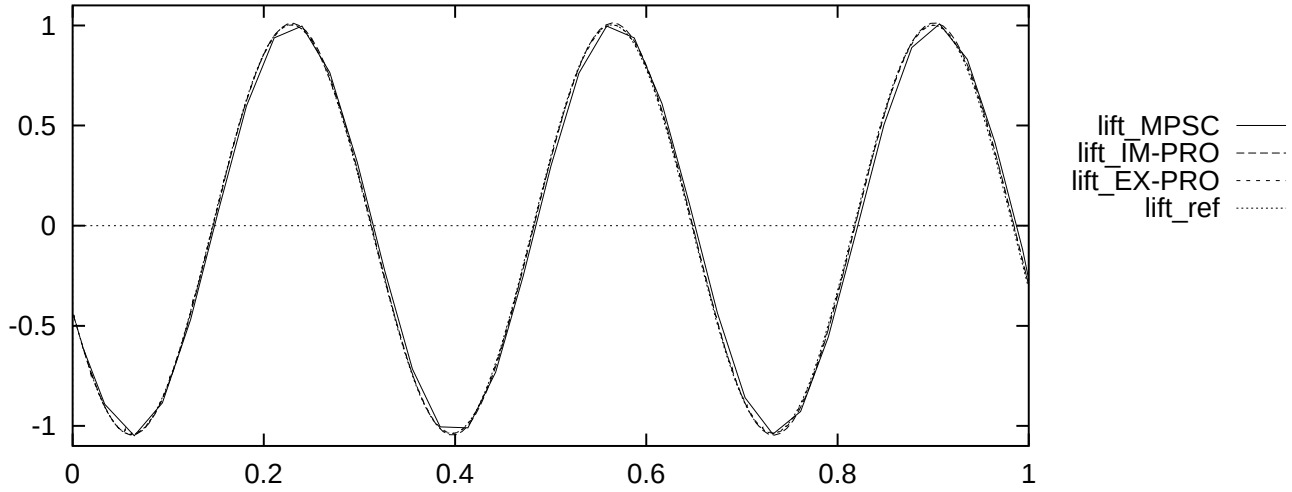
Stronger velocity-pressure coupling:

'Perform more arithmetic work with the once assembled matrices and vectors'!

⇓

Multilevel Pressure Schur Complement

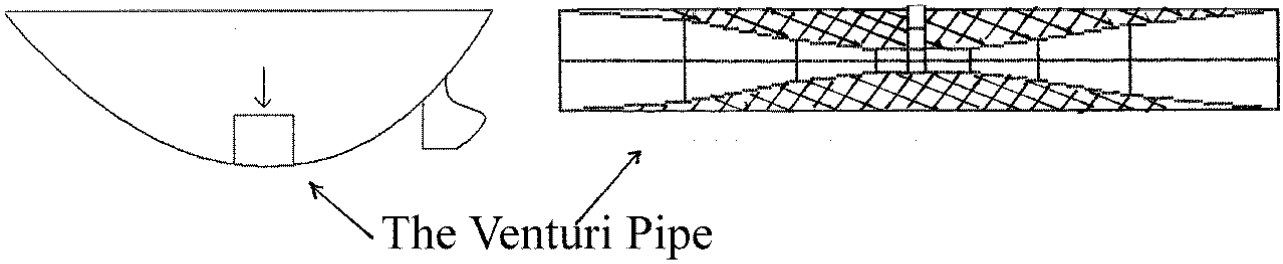
CPU (solvers)	Method	#NT	**Lift** mean	**Lift** peak	**Drag** mean	**Drag** peak
14,358 (**81%**)	fully impl. MPSC	39	1%	1%	0%	2%
42,679 (**51%**)	semi-impl. DPM	165	0%	0%	0%	0%
64,485 (**54%**)	semi-expl. DPM	889	0%	8%	0%	0%



⇓

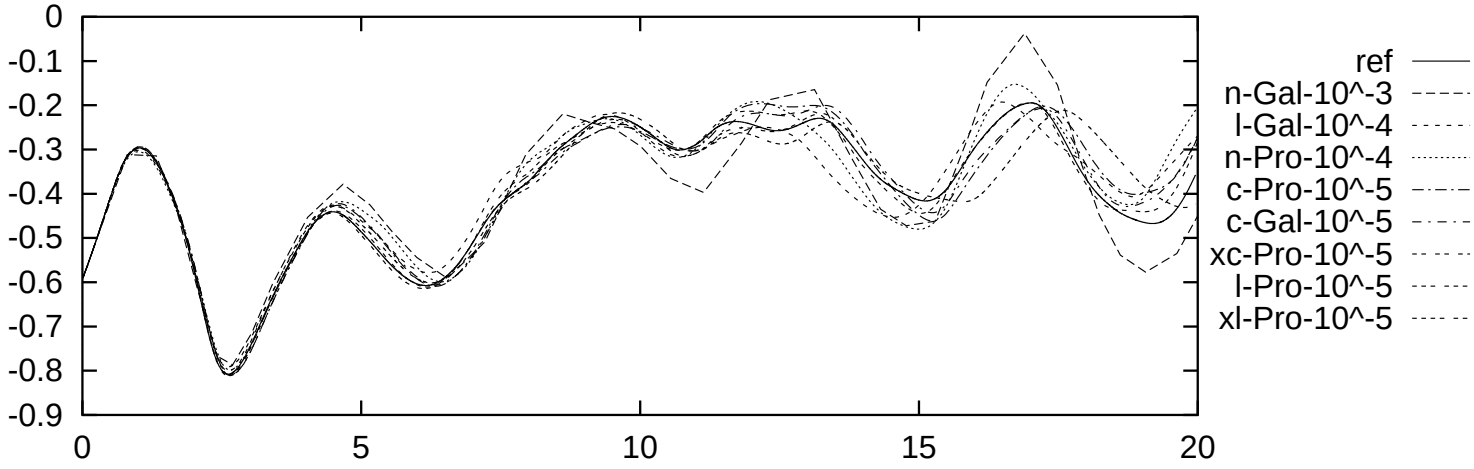
'Ready' for Sparse Banded Blas
'Ready' for Galerkin-type error control

Test: Flow through Venturi Pipe



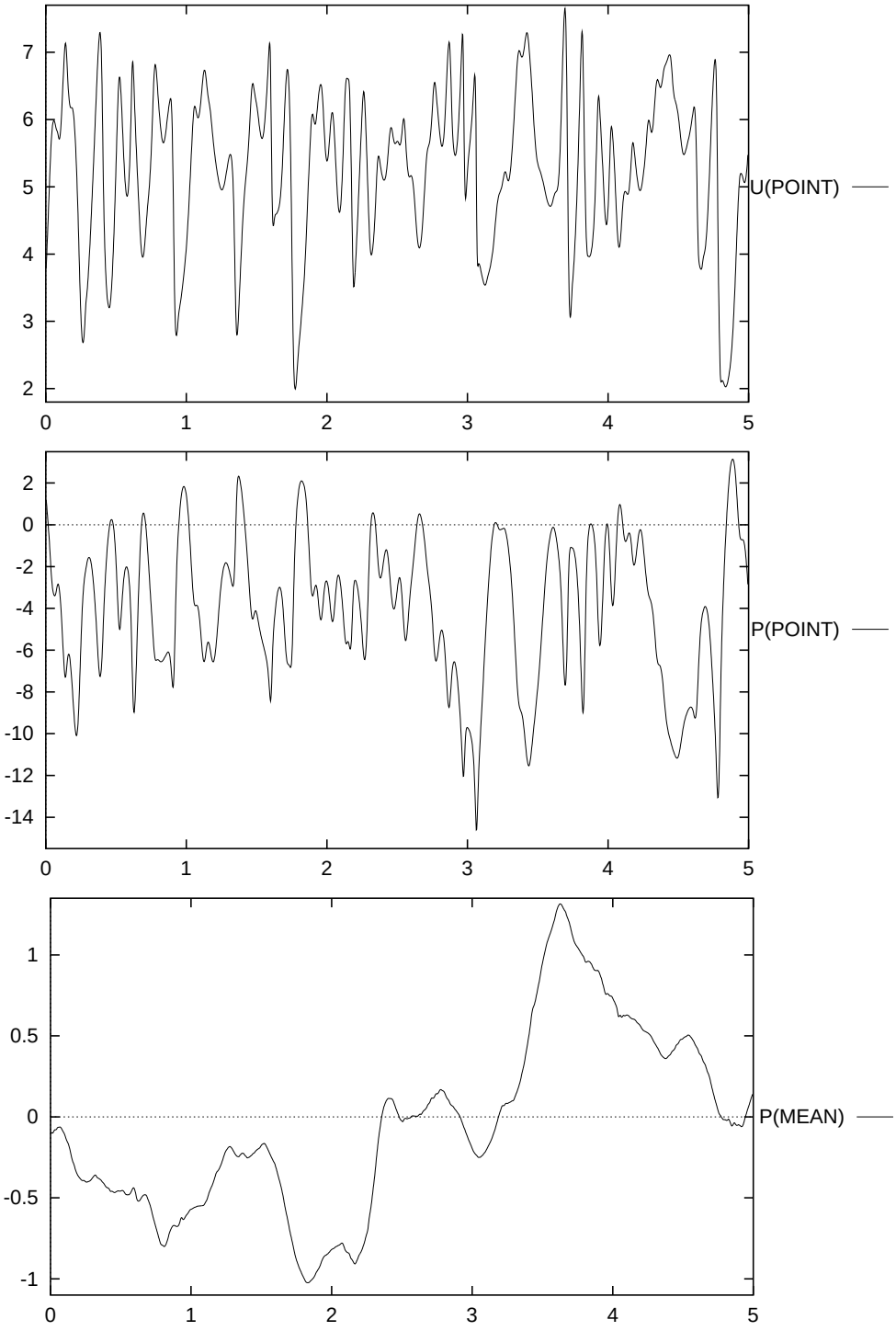
Adaptive time step control for **velocity** (global, l_2) and **flux** (through upper device **!!!**)

$$|f(\mathbf{u}_h^k(t_n), p_h^k(t_n)) - f(\mathbf{u}(t_n), p(t_n))| \leq TOL$$



	Meth.	#NT	**Flux**		**Pressure**	
			l_2	mean	l_2	mean
	n-MPSC	**38**	15%	1%	19%	1%
O	l-MPSC	**222**	11%	2%	13%	0%
K	n-DPM	**263**	11%	2%	14%	3%
	l-DPM	**649**	4%	0%	5%	0%
	xl-DPM	**6941**	11%	0%	14%	0%

Test: Perpendicular inflow into pipe

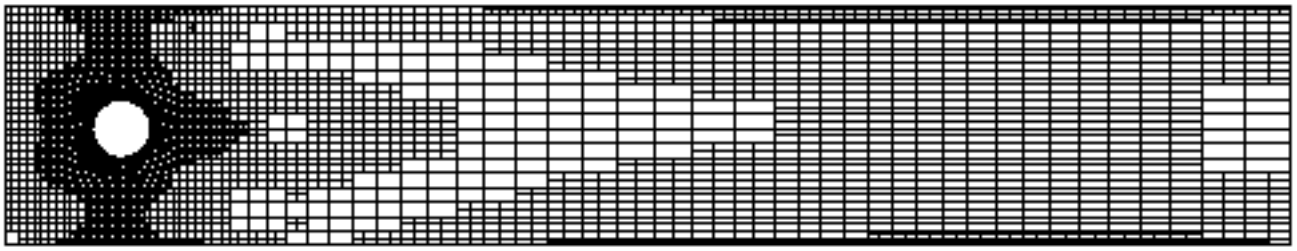


Rigorous adaptive error control of **mean values** ('wall pressure') in **space/time**?!?

Hardware-Oriented Numerics for PDEs:

1) *Patch-oriented adaptivity*

'Many' (local) **tensorproduct grids** (S-B BLAS)
'Few' (local) **unstructured** *grids* (SPARSE)



2) *Generalized MG-DD solver:* ScaRC

Exploit locally 'regular' structures **(efficiency)**
Recursive 'clustering' of anisotropies **(robustness)**

'Strong local solvers improve global convergence !'

3) *Navier-Stokes solvers: Full Galerkin MPSC*

Develop Navier-Stokes solvers with more 'arithmetic' than 'memory intensive' operations

'Exploit locally regular structures !!!'

'My' conclusions:

Everybody can/must test the absolute (!) computational performance!

Compromise between 'flexible/reusable/abstract' and 'machine-dependent/hardware-oriented' implementation styles is required!

Modern Numerics has to consider recent and future hardware trends!

⇓

Future: Huge Potential

'Hardware-Oriented Numerics'