Simulationswerkzeuge
für
Strömungsvorgänge
im
Automobilbau

Projektleiter: **Prof. Dr. R. Rannacher**
Dr. S. Turek

Mitarbeiter: **Chr. Becker**
R. Schmachtel

Institut für Angewandte Mathematik, Universität Heidelberg
Im Neuenheimer Feld 294, 69120 Heidelberg, Germany

`http://www.iwr.uni-heidelberg.de/~featflow`
`http://www.iwr.uni-heidelberg.de/iwr/amj/bmbf/bmbf.html`

Industriepartner: **Dr. M. Wessels**

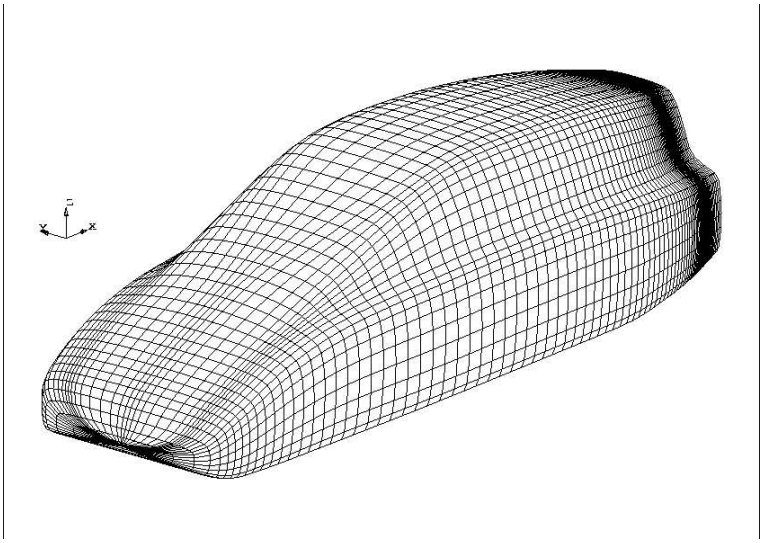
DaimlerChrysler AG, Research and Technology FT1/FB, E222

Anforderungen an Simulationswerkzeuge:

Genauigkeit: (Lokale) Gitterfeinheit, Turbulenzmodellierung

Rechenzeit: Große schlechtkonditionierte Gleichungssysteme

Realität: Komplexe Geometrien, instationäre Strömungen

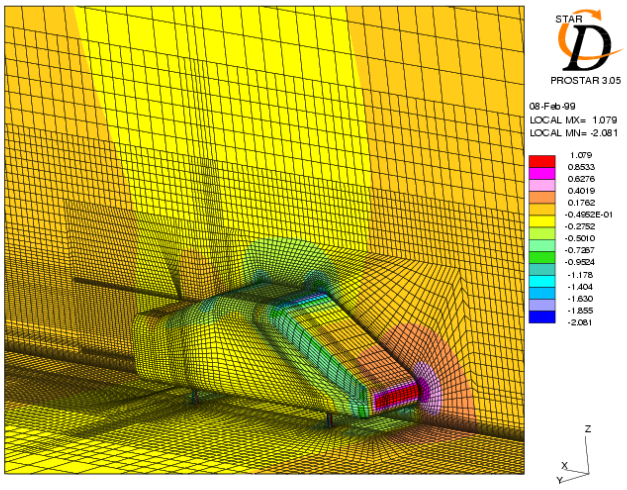


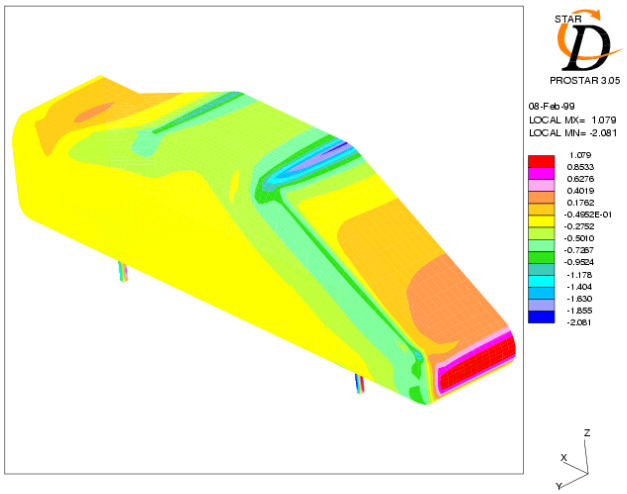
Einsatz in der Praxis:

'10 Millionen Gitterpunkte, Tausende von Zeitschritte!'
(**RAM, CPU**)

Ergebnisse von DaimlerChrysler:

- **STAR-CD**, **500.000 Hexaeder**
- SGI Origin2000 (**6 Prozessoren**), **6.5 Stunden** (CPU)





Größe	Messung	Rechnung	Differenz
Widerstand ('c_w')	0.165	0.317	**92 %**
Auftrieb ('c_a')	-0.083	-0.127	**53 %**

$\Downarrow$

(Relativ) Langsam !!!

'Optimales' Mehrgitter $\sim$ **0.1 sec/Teilproblem**

$+$

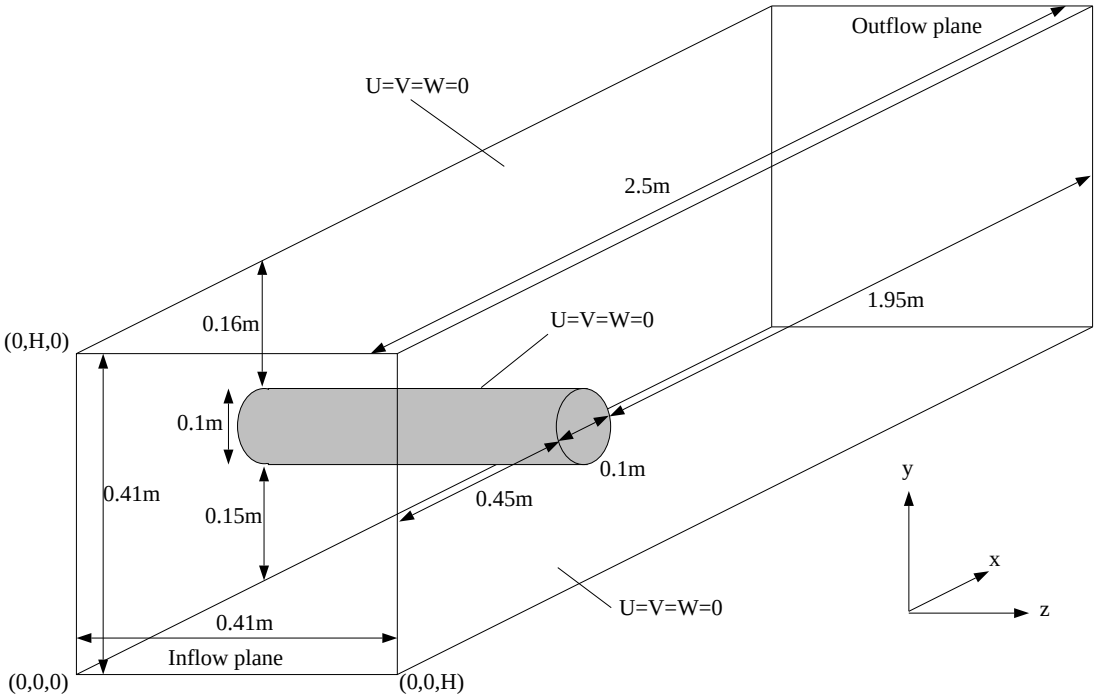
(Lokale) Verfeinerung notwendig !!!

(Mindestens) **2 Millionen Hexaeder** (4-fach)
50 CPU-Stunden (6–8-fach)

Analyse der Problematik

$\Downarrow$

(Laminarer) DFG–Benchmark
(Durst/Krause/R./Schäfer/T.):



Erkenntnisse für laminare Strömungen:

- **Quantitative** Genauigkeit nur mit sehr hoher Gitterfeinheit
- Rechenzeiten im Bereich von **Tagen** auf **'Superrechnern'**

$\Uparrow$

Verbesserung um **Faktor 100 – 1000 (!!!)**

Ansätze zur Verbesserung:

1a) Verbesserung und Kontrolle der **Diskretisierung**

→ weniger **Unbekannte !!!**

1b) Effiziente und robuste **Lösungsverfahren**

→ verbesserte **Konvergenzraten !!!**

2) Verbesserung der **Turbulenzmodellierung**

→ **'Turbulenzfehler' ≈ 'Diskretisierungsfehler' !!!**

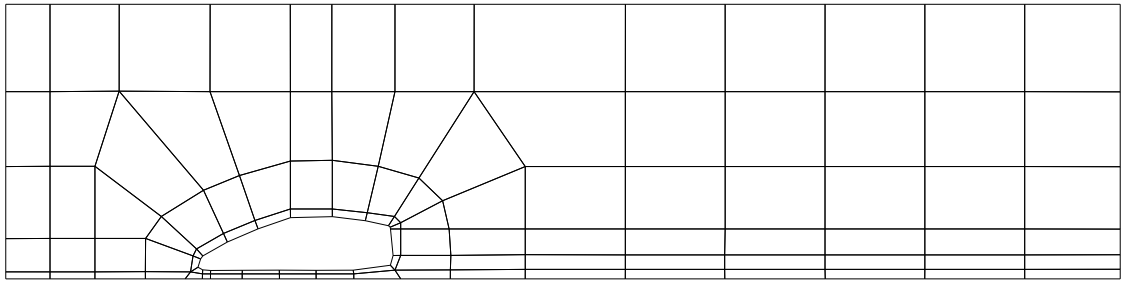
3) Optimierung der **Software**

→ Problem- und Hardware-angepaßte **Implementierung !!!**

Laufzeitprobleme durch Implementierung

$\Downarrow$

SPARSE MV Multiplikation in FEATFLOW



Computer	#Unbekannte	**CM**	**TL**	**STO**
	13 688	22	20	**19**
SUN E450	54 256	17	15	**13**
($\sim$ 250 MFLOP/s)	216 032	16	14	**6**
(*CSR*)	862 144	16	15	**4**

Verschiedene *Numerierungen führen zu* **identischen** *Resultaten (arithm. Operationen, Speicherzugriffe + Ausgabe), bei gleichzeitigen* **signifikanten Unterschieden** *in der Laufzeit!*

$+$

*Wie erstellt man '***schnellere***' Codes* **???**

FEAST Projekt

⇓

1. Schritt: Software-Design

Daten-, Matrix- und Löserstrukturen
Sparse Banded Blas
Feast Indices

⇓

2. Schritt: Angepaßte Numerik

Voll-implizite Galerkin Diskretisierungen
Multilevel Pressure Schur Complement (MPSC)
Anisotrope Gitter + *Adaptivität* + *'Loadbalancing'*

⇓

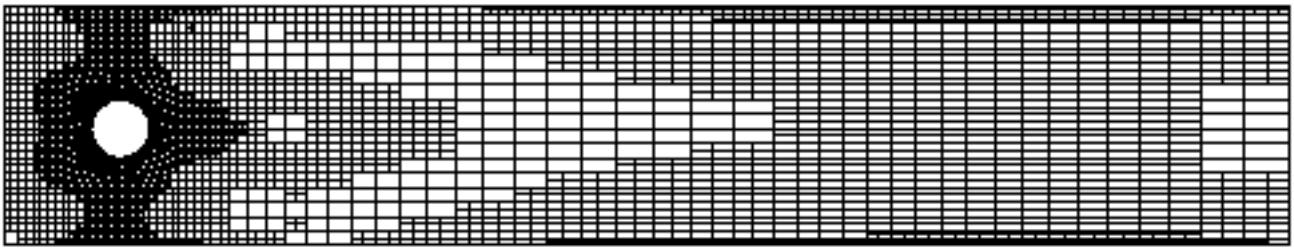
3. Schritt: Industriekooperation

Beschleunigung von 'Fremd'-Software
Eigenentwicklung: **FEATFLOW 2.0**

Grundprinzipien in FEAST:

Kombiniere moderne Numerik (FEM, MG) mit 'High Performance' Lineare Algebra !!!

Nutze 'lokale Strukturiertheit' aus !!!



(Rekursive) 'Divide and Conquer' Strategien

$\longrightarrow$ 'Finde lokal **strukturierte** bzw. anisotrope Anteile!'

$\longrightarrow$ Hierarchische Daten-, Löser und **Matrixstrukturen**

$\longrightarrow$ **Parallelisierung + Vektorisierung** direkt integriert

Schnelle Löser auf 'strukturierten' Patches

$\longrightarrow$ Sparse Banded Blas + Feast Indices

$\longrightarrow$ **Hardware-nahe** Implementierung und Speichermanagment

$\longrightarrow$ '**Referenzelement-Löser**' auf Tensorprodukt-Gittern

Sparse **Matrix-Vektor Techniken**

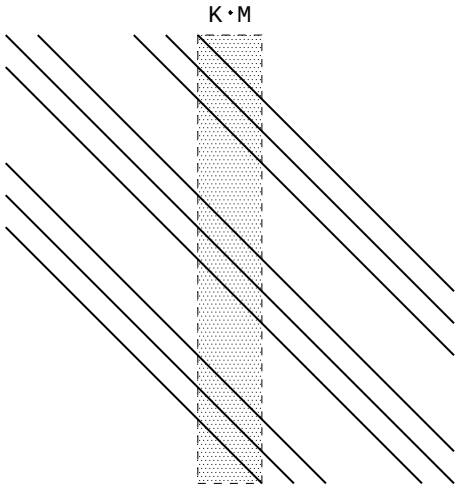
```
      DO 10 IROW=1,N
      DO 10 ICOL=KLD(IROW),KLD(IROW+1)-1
       Y(IROW)=Y(IROW)+A(ICOL)*X(KCOL(ICOL))
10    CONTINUE
```

- Ergebnisse weit weg von '**Peak Performance**'
- Abhängig von **Problemgröße + Numerierung**
- '**alt**' (IBM POWER2) teilweise **schneller** als '**neu**' (IBM P2SC)
- PC teilweise **schneller** als Prozessor in 'Superrechner' !!!

$$\Downarrow$$

Verwende (INTEL) PC's bei **unstrukturierten** *Gittern und* **lokal adaptiven** *Methoden* **!!!**

Sparse Banded Blas **MV Techniken**




```
      DO 10 IM=1,M/K
       DO 100 I=1,K*M
100     Y(I)  =Y(I)  +DD(I)*X(I)+DL(I)*X(I-1)+DU(I)*X(I+1)
       DO 200 I =1,K*M
200     Y(I-M)=Y(I-M)+LD(I)*X(I)+LL(I)*X(I-1)+LU(I)*X(I+1)
       DO 300 I=1,K*M
300     Y(I+M)=Y(I+M)+UD(I)*X(I)+UL(I)*X(I-1)+UU(I)*X(I+1)
10    CONTINUE
```

- **Potential** der 'Superrechner' wird sichtbar (bis 800 MFLOP/s)
- **Warnung:** Tensorprodukt-Gitter **nicht** ausreichend !!!

$\Downarrow$

Prozessoren sind 'sensible' Parallel-Vektor-Superrechner bzgl. **Caching-in + Pipelining !!!**

Sparse Banded Blas vs. Sparse Techniken:

3D case	N	STO	TL	SBB-V	SBB-C	**MG-V**	**MG-C**
DEC 21264	17^3	150	151	446	765	342	500
(500 MHz)	33^3	54	57	240	768	233	474
'**DS20**'	65^3	**24**	**54**	**249**	**713**	**196**	**447**
IBM RS6000/597	17^3	81	81	179	480	171	368
(160 MHz)	33^3	16	55	170	393	152	300
'**SP2**'	65^3	**8**	**14**	**178**	**393**	**150**	**276**
INTEL PII	17^3	28	28	56	183	48	136
(400 MHz)	33^3	24	26	53	139	47	116
'**ALDI**'	65^3	**19**	**23**	**54**	**125**	**45**	**101**

$\Downarrow$

Sehr unterschiedliche MFLOP/s für MV Multiplikation mit **identischer** *Matrix !!!*

$+$

'(Alternierender) Linien-Gauß-Seidel' Glätter für gleichförmige und **anisotrope** *Gitter* **!!!**

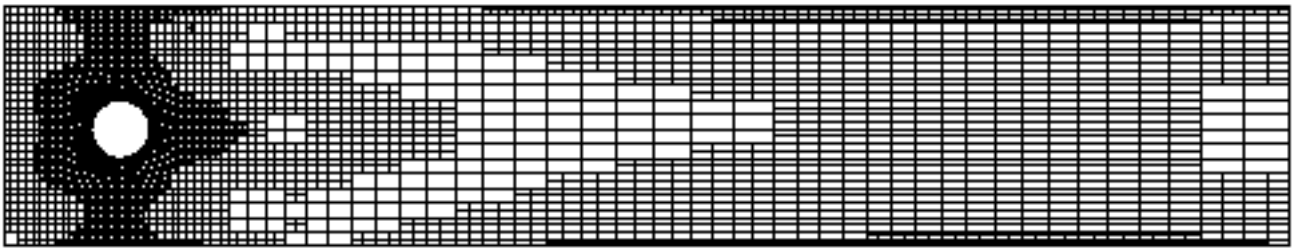
$\Downarrow$

'Optimales' lokales Mehrgitter !!!

eitere numerische Komponenten:

1) *Patch-orientierte Adaptivität*

'Viele' (lokale) **Tensorprodukt-Gitter** (S-B BLAS)
'Wenige' (lokale) **unstrukturierte** *Gitter* (SPARSE)



2) *Verallgemeinerter MG-DD-Löser:* ScaRC

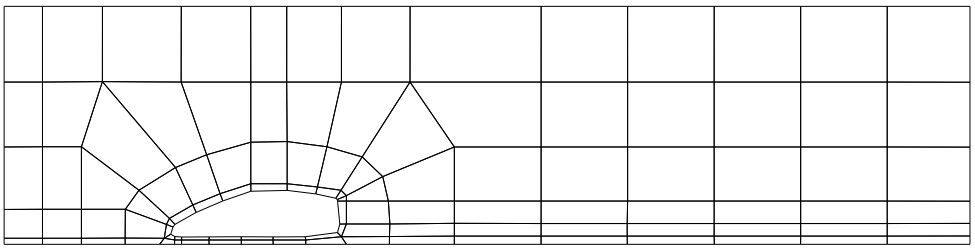
Finde lokale 'reguläre' Strukturen **(Effizienz)**
Rekursives 'Clustering' von Anisotropien **(Robustheit)**
'Starke lokale Löser verbessern globale Konvergenz !!!'

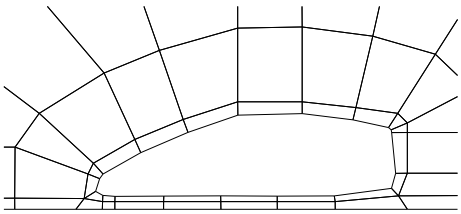
3) *Navier-Stokes Löser: Full Galerkin MPSC*

Entwickle Navier-Stokes Löser mit mehr
'arithmetischen' als 'speicherintensiven' Operationen

'Nutze lokale Strukturiertheit aus !!!'

BMBF-Projekt: Ein schneller Navier–Stokes Löser für die Fahrzeug–**Aerodynamik** (DaimlerChrysler AG)





Paralleles 'two level ScaRC' für das 'Pressure–Poisson' Problem

$+$

Lokale Adaptivität durch Verschieben von Gitterpunkten

$\Downarrow$

'Optimal' bzgl. Robustheit und Effizienz:
1 Stelle pro Iteration **!!!**

	N_l	$ScaRC_g$
$h \approx 10^{-4}$	8	0.09
	16	0.10
	32	0.08
$h \approx 10^{-7}$	8	0.06
	16	0.03
	32	0.02

'Typ I'	N_l	$AdiGS_l$	'Typ II'	$AdiGS_l$
$1:1$	32	0.02	$1:10$	0.03
	64	0.02		0.04
	128	0.02		0.04
$1:10^4$	32	0.02	$1:10^5$	0.09
	64	0.03		0.08
	128	0.03		0.08

Globales (links) und **lokales** (rechts) Konvergenzverhalten

Erwartete Hardware-Entwicklung:

Year of 1st shipment	**1997**	1999	2001	2003	2006	2009	**2012**
Local clock (MHz)	**750**	1250	1500	2100	3500	6000	**10K**
Transistors/chip	**11M**	21M	40M	76M	200M	520M	**1.4B**

Auszüge aus der '1997 Semiconductor Roadmap'

⇓

1 Milliarde Transistoren (**Faktor 100 !**)

10 GHz Taktung (**Faktor 10 !**)

⇓

1 PC 'schneller' als komplette CRAY T3E heute !

Speicherzugriff ???

Datenlokalität und interne Parallelität !!!
Vektorisierung !!!
Speichermanagment vs. Cache-Architekturen !!!

⇕

Numerische Methoden ???