Introduction to

Hardware-oriented Numerics

(for PDEs)

S. Turek + Feast Group

M. Altieri, Chr. Becker, S. Buijssen, S. Kilian, D. Kuzmin, ...

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'

$\Downarrow$

Aim: Improvement of efficiency via:

1. 'Optimal' **numerical complexity**

   $\rightarrow$ 'Less' number of grid points/#dofs (?)

   $\rightarrow$ 'Less' MG/DD iteration steps (?)

2. 'Optimal' use of **efficient** 'sub-components'

   $\rightarrow$ Reduction to 'simpler' subproblems (?)

Trends in Processor Technology:

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

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

$\Downarrow$

National Technology Roadmap for Semiconductors						
Year	1997	**1999**	2003	2006	2009	**2012**
Local clock (MHz)	750	**1250**	2100	3500	6000	**10K**
Transistors/chip	11M	**21M**	76M	200M	520M	**1.4B**

$\Downarrow$

1. 1 Billion Transistors ($\times$ **100 !**)
2. 10 Ghz clock rate ($\times$ **10 !**)

$\Downarrow$

1 PC 'faster' than **complete** *CRAY T3E today !*

Questions:

'Can we use this enormous computing power?'

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

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

Error Control/Adaptivity!

Iterative (parallel) Solvers!

Decoupling Strategies!

$\Updownarrow$

Not **data processing**, *but* **data moving** *is costly!*

Employ **cache-oriented** *techniques!*

Key Tools: *'Data Locality/Pipelining'!*

'Counterexample': Poisson Problem in 3D

- $100 \times 100 \times 100$ grid points $\Longrightarrow$ **Problem size:** $N = 10^6$
- **Complexity of GE:** $N^{7/3} \approx 10^{14}$ FLOPs

100 sec on a 1 TFLOP/s computer

- **Complexity of (opt.) MG:** $1,000N \approx 10^9$ FLOPs

100 sec on a 10 (!!!) MFLOP/s computer

- $1,000^3$ grid points $\Longrightarrow$ **Problem size:** $N = 10^9$
- **Complexity of GE:** $N^{7/3} \approx 10^{21}$ FLOPs

10^6 **sec on a 1 P(!!!)FLOP/s computer**

- **Complexity of (opt.) MG:** $1,000N \approx 10^{12}$ FLOPs

1,000 sec on a 1 (!!!) GFLOP/s computer

$\Downarrow$

Improvements by modern Numerics !??

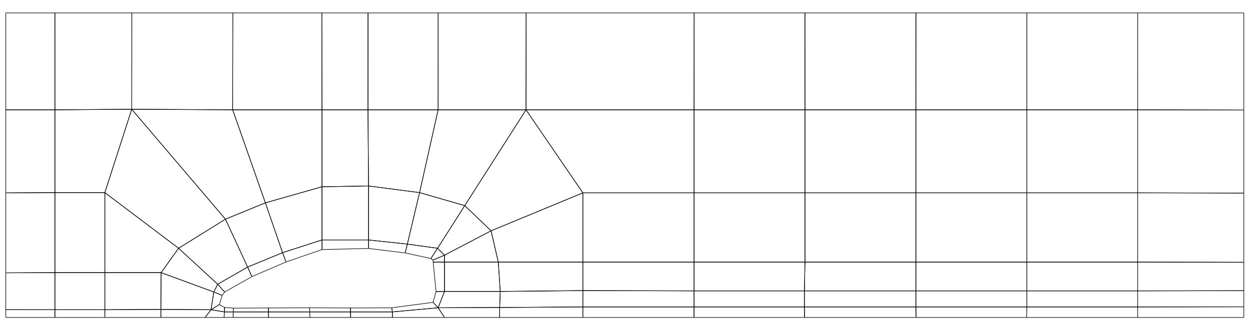
Main components in iterative schemes:

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 ?**

$\Downarrow$

Sparse MV multiplication in **FEATFLOW**



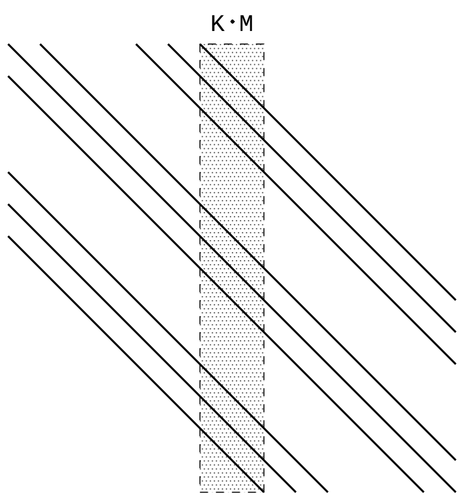
Computer	#Unknowns	**CM**	**TL**	**STO**
SUN E450 ($\sim$ 250 MFLOP/s)	13,688	22	20	**19**
	54,256	17	15	**13**
	216,032	16	14	**6**
	862,144	16	15	**4**

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 !!!

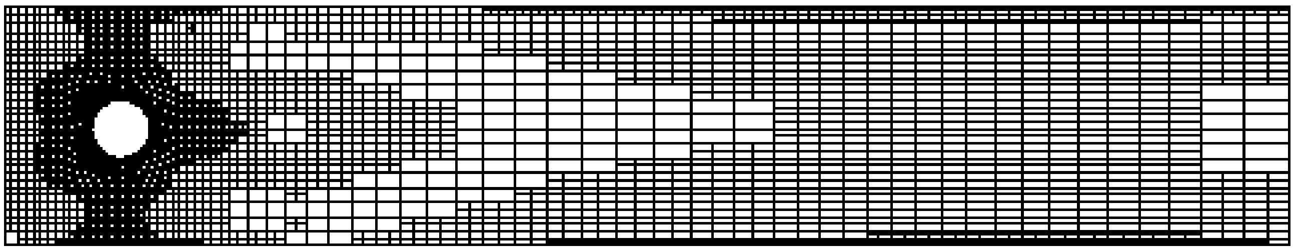
$\Uparrow$

Hardware-oriented Numerics???

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 !'

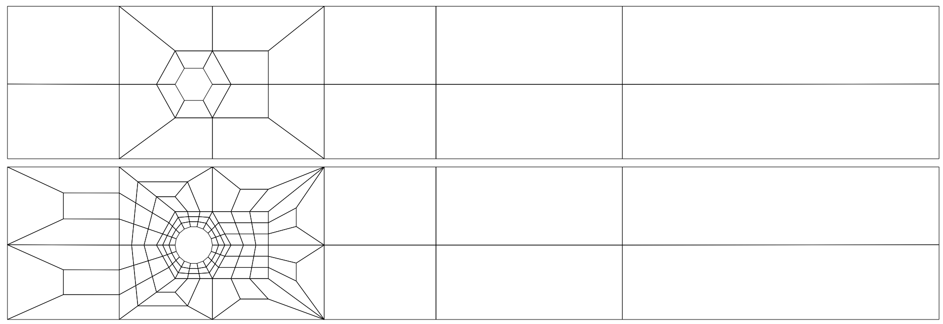
3) *Navier-Stokes solvers: Full Galerkin MPSC*

Develop Navier-Stokes solvers with <u>*more*</u> *'arithmetic'*
than 'memory intensive' operations

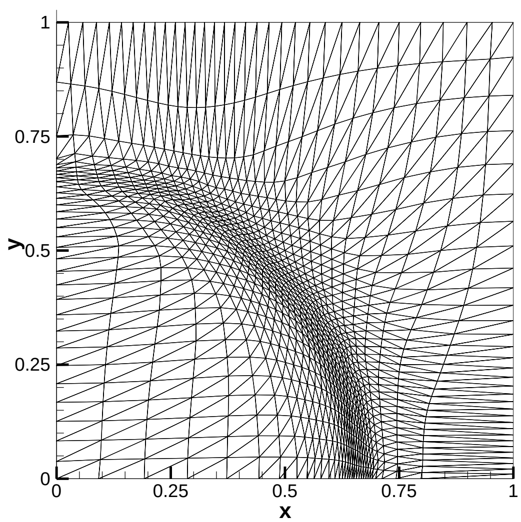
'Exploit locally regular structures !!!'

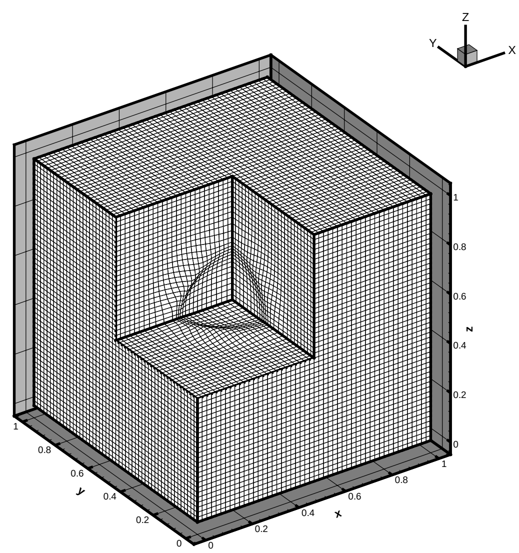
I) Concepts for adaptive meshing

1) macro-oriented adaptivity



2) patchwise adaptivity





3) local adaptivity

→ 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 !**

$\Downarrow$

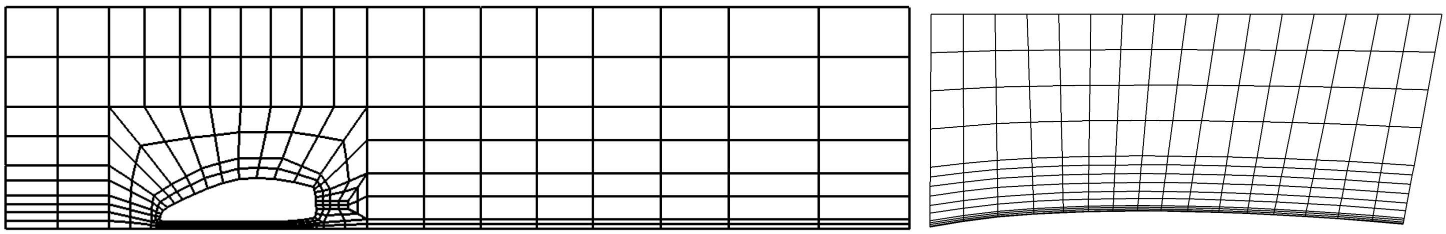
1) + 2) = ScaRC

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

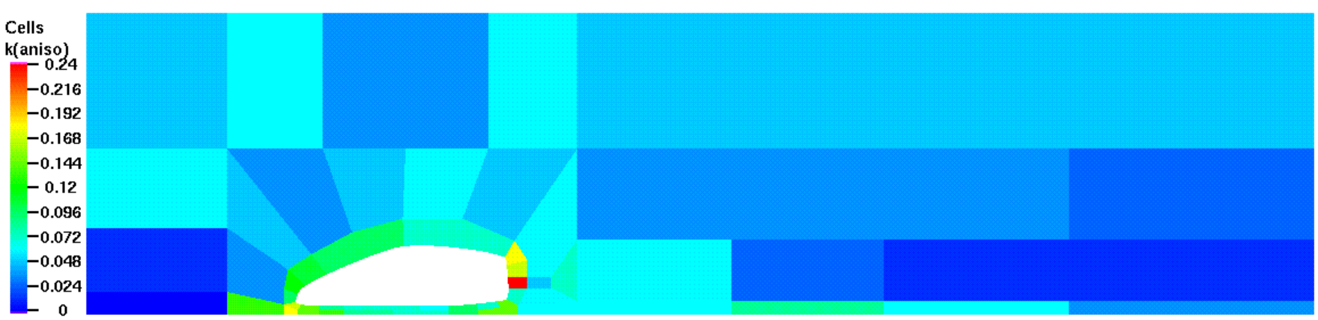


ScaRC-CG solver (smoothing steps: 1 global ScaRC ; 1 local 'MG-TriGS') for locally (an)isotropic refinement

Global (parallel) convergence rates

$\#NEQ$	$AR \approx 10$	$AR \approx 10^6$
$286,720$	0.17	0.20
$1,146,880$	0.17	0.18
$4,587,520$	0.17	0.19

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



‘Exploit the locally nice structures !’

- Local use of efficient Sparse Banded Blas techniques
- Locally different choice of components (V/C, smoothing)
- Local storage cost small through the use of matrix stencils

$\Uparrow$

Open problems:

Dynamic a posteriori Load Balancing ?

Macro-refinement vs. number of macros ?

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!*

$\Downarrow$

How to increase the 'floating point' intensive parts?

Larger time steps!

How???

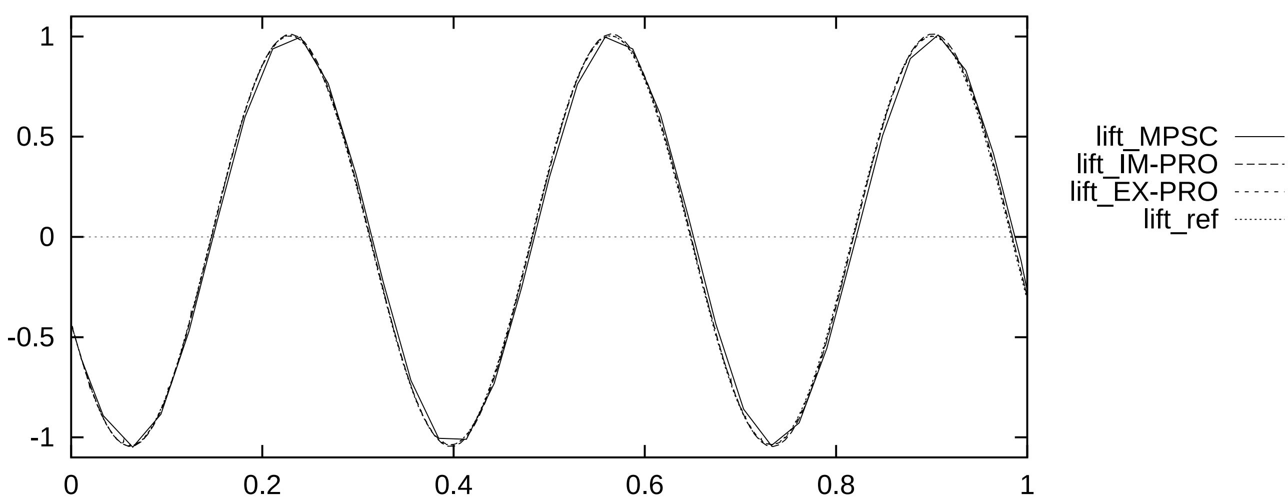
Stronger velocity-pressure coupling:

'Perform more arithmetic work with the <u>once</u> assembled matrices and vectors'!

$\Downarrow$

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%



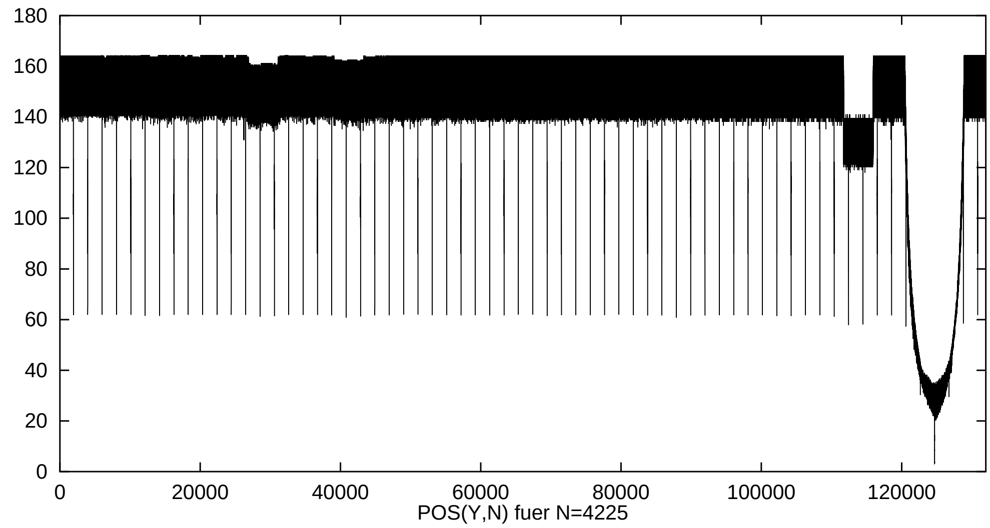
$\Downarrow$

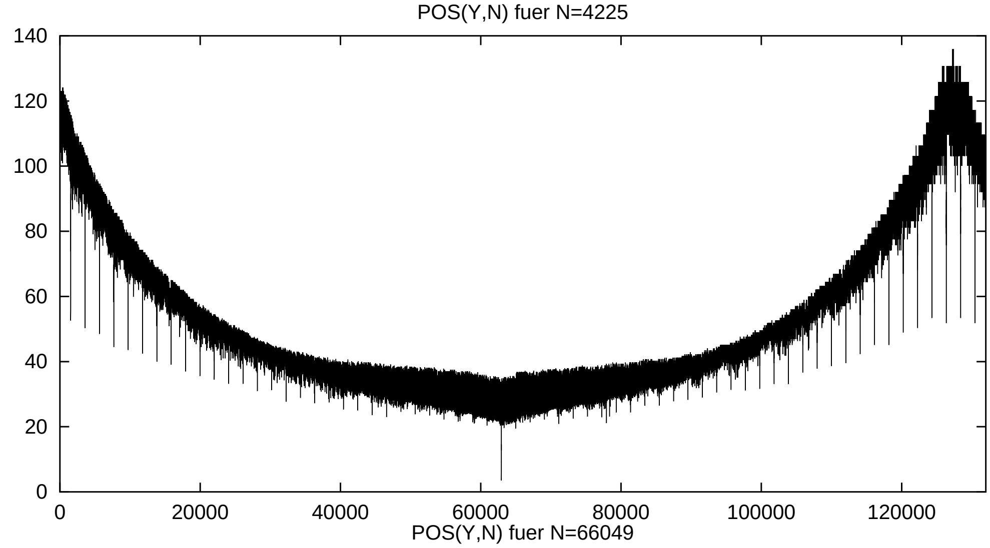
'Ready' for SPARSE BANDED BLAS

IV) Knowledge of Computer Characteristics

Memory management

```
      DOUBLE PRECISION DX(8000000)
      DO 220 I = 1,ITER(J)
220    CALL DAXPY(N(J),ALPHA,DX(1),INCX,DX(1+N(J)+JSTEP),INCX)
```





$\Downarrow$

Cache associativity ???

TRIDI preconditioner (I)

Computer	#unknowns	‘classic’	SBTRI-V	SBTRI-C
IBM RS6K/597	65^2	33	32	33
(160 MHz)	257^2	32	32	33
‘**SP2**’	1025^2	32	32	33

$\Downarrow$

”Division-free” variants

```
      MNUM =M/K ; MSIZE=M*K
      DO 10 I=0,MNUM-1
       DO 100 J=2,MSIZE
100     B(I*MSIZE+J)=B(I*MSIZE+J)-L(I*MSIZE+J-1)*B(I*MSIZE+J-1)
       DO 200 J=1,MSIZE
200     B(I*MSIZE+J)=B(I*MSIZE+J)*D(I*MSIZE+J)
       DO 300 J=MSIZE-1,1,-1
300     B(I*MSIZE+J)=B(I*MSIZE+J)-U(I*MSIZE+J)*B(I*MSIZE+J+1)
10    CONTINUE
```

Computer	#unknowns	‘classic’	SBTRI-V	SBTRI-C
IBM RS6K/597	65^2	144	130	141
(160 MHz)	257^2	80	105	150
‘**P2SC**’	1025^2	80	106	150

$\Downarrow$

Divison vs. Mult./Addition ???

TRIDI preconditioner (II)

Computer	#unknowns	SBTRI-V	MTGS-V	MJAC-V
CRAY T90	65^2	34	195	882
	257^2	34	215	925
'**Vector**'	1025^2	34	230	928

$\Downarrow$

"Cyclic Reduction" variant

Computer	#unknowns	SBTRI-V	MTGS-V	MJAC-V
CRAY T90	65^2	172	186	882
	257^2	201	270	925
'**Vector**'	1025^2	205	419	928

$\Downarrow$

Renaissance of 'old' methods

LINUX (GNU F77) vs. vendor-optimized compilers ???

DEC 21164 PC/LINUX, in Cache!!!

ULTRIX F77: MV-C **830** *MFLOP/s !!!*

EGCS F77: MV-C **196** *MFLOP/s !!!*

F90 vs. F77 compilers ???

SUN ULTRA II, in Cache!!!

F77: MV-C **404** *MFLOP/s !!!*

F90: MV-C **64** *MFLOP/s !!!*

C(++) vs. F77 compilers (SUN E3500) ???

$N = 1025^2$	F77	C
DAXPY-C	30	14
DAXPY-V	16	12
DAXPY-I	**8**	**7**

$N = 65^2$	F77	C
DAXPY-C	228	62
DAXPY-V	166	42
DAXPY-I	**109**	**36**

'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!

$\Downarrow$

Future:

'Hardware-Oriented Numerics'