Trends in processor technology
and their impact on
Numerics for PDE’s

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

http://gaia.iwr.uni-heidelberg.de/~ture
http://www.iwr.uni-heidelberg.de/~featflow

October 1999: Universität Dortmund
Two main topics:

‘A posteriori (error) control of FEM/FV discretizations with adaptive meshing strategies’

‘(Iterative) Solution strategies for huge systems of equations’

• Is error control with adaptive meshing necessary?
  → YES !!!!

• Does error control lead to more efficient simulation tools?
  → ??  ⇒ ‘real life’ applications (CFD ???)

• Do adaptive meshes lead to more efficient simulations?
  → ?  ⇒ RAM ???
  → ?? ??  ⇒ CPU ???

↑

• Robust and efficient (complete) solvers !
• Modern processor technology !!!
Main components in iterative schemes:

**Matrix-Vector applications (MV)**

- Krylov-space methods, Multigrid, etc.
- **Defect calculations**, smoothing, step-length control, etc.
- Sometimes consuming (at least) 60 - 90% of CPU time

---

**MV techniques (Storage/Application):**

1. **Sparse MV**
   - 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

2. **(Sparse) Banded MV**
   - Typical for **FD** codes on ‘**tensorproduct meshes**’
   - Storage of ‘**non-zero**’ elements in **bands/diagonals**
   - MV multiplication ‘**bandwise**’ (and ‘**window-oriented**’)
   - **FEAST !!!**
Results on general unstructured meshes

- **Sparse MV multiplication in FEATFLOW (F77!!!)**

<table>
<thead>
<tr>
<th>Computer</th>
<th>#Unknowns</th>
<th>CM</th>
<th>TL</th>
<th>STO</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUN E450</td>
<td>13,688</td>
<td>22</td>
<td>20</td>
<td>19</td>
</tr>
<tr>
<td>(~ 250 MFLOP/s)</td>
<td>54,256</td>
<td>17</td>
<td>15</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td>216,032</td>
<td>16</td>
<td>14</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>862,144</td>
<td>16</td>
<td>15</td>
<td>4</td>
</tr>
</tbody>
</table>

- **STAR-CD, 500,000 hexaheders** (by DaimlerChrysler)
- **SGI Origin2000 (6 processors), 6.5 h (CPU)**

\[ \uparrow \]

‘Optimal’ multigrid \( \sim 0.1 \text{ sec/subproblem} \)???

<table>
<thead>
<tr>
<th>Quantity</th>
<th>Experiment</th>
<th>Simulation</th>
<th>Difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Drag (‘( c_w )’)</td>
<td>0.165</td>
<td>0.317</td>
<td>92 %</td>
</tr>
<tr>
<td>Lift (‘( c_a )’)</td>
<td>-0.083</td>
<td>-0.127</td>
<td>53 %</td>
</tr>
</tbody>
</table>
First conclusions for sparse MV:

Different numbering strategies can lead to:

- identical numerical results and work (arithm. operations, memory accesses)!
- huge differences in elapsed CPU time!

Sparse MV techniques are ‘slow’ and depend on:

- problem size!
- ‘amount’ and ‘kind’ (?) of data access!

Sparse MV techniques are basis for:

- Most available commercial codes!
- Most recent research software projects!
Results on locally structured meshes

<table>
<thead>
<tr>
<th>3D case</th>
<th>N</th>
<th>STO</th>
<th>LINE</th>
<th>SBB-V</th>
<th>SBB-C</th>
<th>MG-V</th>
<th>MG-C</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEC 21264</td>
<td>17^3</td>
<td>150</td>
<td>164</td>
<td>446</td>
<td>765</td>
<td>342</td>
<td>500</td>
</tr>
<tr>
<td>(500 MHz)</td>
<td></td>
<td>33^3</td>
<td>54</td>
<td>64</td>
<td>240</td>
<td>768</td>
<td>233</td>
</tr>
<tr>
<td>‘DS20’</td>
<td></td>
<td></td>
<td>24</td>
<td>72</td>
<td>249</td>
<td>713</td>
<td>196</td>
</tr>
<tr>
<td>IBM RS6000/597</td>
<td>17^3</td>
<td>81</td>
<td>86</td>
<td>179</td>
<td>480</td>
<td>171</td>
<td>368</td>
</tr>
<tr>
<td>(160 MHz)</td>
<td></td>
<td>33^3</td>
<td>16</td>
<td>81</td>
<td>170</td>
<td>393</td>
<td>152</td>
</tr>
<tr>
<td>‘SP2’</td>
<td></td>
<td></td>
<td>8</td>
<td>81</td>
<td>178</td>
<td>393</td>
<td>150</td>
</tr>
<tr>
<td>INTEL PII</td>
<td>17^3</td>
<td>28</td>
<td>29</td>
<td>56</td>
<td>183</td>
<td>48</td>
<td>136</td>
</tr>
<tr>
<td>(400 MHz)</td>
<td></td>
<td>33^3</td>
<td>24</td>
<td>29</td>
<td>53</td>
<td>139</td>
<td>47</td>
</tr>
<tr>
<td>‘ALDI’</td>
<td></td>
<td></td>
<td>19</td>
<td>29</td>
<td>54</td>
<td>125</td>
<td>45</td>
</tr>
</tbody>
</table>

**Sparse MV techniques (STO,LINE)**
- MFLOP/s rates far away from ‘Peak Performance’
- Depending on problem size + numbering
- ‘Old’ (IBM PWR2) partially faster then ‘new’ (IBM P2SC)
- PC partially faster than processors in ‘supercomputers’ !!!

**Feast MV techniques (SBB)**
- ‘Supercomputing’ power gets visible (up to 800 MFLOP/s)
- **Warning:** Hard work !!! (→ **Sparse Banded Blas**)
Further conclusions:

‘Most adaptive codes should run on (PENTIUM) PC’s’

‘Processors are ‘sensible’ Parallel-Vector supercomputers w.r.t. caching-in + pipelining’

Evaluation of Soft/Hardware components ???

Adaptivity concepts ???

Realization of complete FEM/FV approaches ???

FEAST project
Expected hardware development:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Transistors/chip</td>
<td>11M</td>
<td>21M</td>
<td>76M</td>
<td>200M</td>
<td>520M</td>
<td>1.4B</td>
</tr>
</tbody>
</table>

↓

1. 1 Billion Transistors ($\times 100$ !)
2. 10 Ghz clock rate ($\times 20$ !)

↓

1 PC ‘faster’ than complete CRAY T3E today!

Memory access ???
Data locality and internal parallelism ???
Vectorization ???
FEAST project

↓

Precise knowledge of processor characteristics

↓

(Collection of) FEAST INDICES

↑

Results for the different tasks in MG/Krylov-space solvers for various FEM/FV discretizations and mesh sizes
MV-MULT with identical matrix !!!

MV/C Level 3 3D

MV/V − Level 3 3D

STO − Level 3 3D
Example: Concepts for adaptive meshing

1) *macro-oriented adaptivity*

→ ‘blind’ (irregular) macro-nodes
→ **conforming** completion

2) *patchwise adaptivity*

→ many (local) logically equivalent **tensorproduct** meshes
→ **moving** mesh points

3) *fully local adaptivity*

→ (local) **unstructured** meshes

‘Exploit the locally nice structures!’
Parallele adaptive Mehrgitterverfahren

Ein objektorientierter Ansatz auf semistrukturierten Gittern

Diplomarbeit

Heiko Lötzbeyer

Aufgabensteller: Prof. Dr. Chr. Zenger
Betreuer: Prof. Dr. U. Rüde

Abgabedatum: 15.12.1996

\[1\text{Institut für Mathematik der Universität Augsburg} \]
Conclusions:

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

→ FORTRAN 90, C++, JAVA ???

Most adaptive FEM codes are NOT slower on PC’s than on ‘High Performance’ workstations or even ‘supercomputers’!

→ However: there is ‘supercomputing power’ available!!!

‘Amount’ and ‘kind’ of memory accesses determines significantly the computational efficiency!

→ modern processors are ‘sensible’ supercomputers!!!

User-defined memory management and optimized implementations ‘w.r.t. hardware’ are absolutely necessary!

→ massive performance losses due to Cache/Heap organization!!!

Modern Numerics has to consider recent and future hardware trends!

→ math. theory for rigorous macro-oriented adaptivity ???
→ improved MG/DD solvers (= SCaRC !!!)
→ algorithmic design: Balance of FLOPs and data access