GPU Acceleration of Unmodified CSM and CFD Solvers

Dominik Göddeke
Sven H.M. Buijssen, Hilmar Wobker and Stefan Turek

Angewandte Mathematik und Numerik
TU Dortmund, Germany
dominik.goeddeke@math.tu-dortmund.de

High Performance Computing & Simulation
Workshop on Architecture-aware Simulation and Computing
Leipzig, June 22, 2009
Scientific computing is in the middle of a paradigm shift

ILP wall        memory wall        characteristic feature size
heat        power consumption        leaking voltage

Hardware evolves towards parallelism and heterogeneity

multicore CPUs    Cell BE processor    GPUs

Emerging manycore architectures

accelerators    algorithm design for 10000s of threads
FEAST – Hardware-oriented Numerics
Mesh structure

Fully adaptive grids
Maximum flexibility
‘Stochastic’ numbering
Unstructured sparse matrices
Indirect addressing, very slow.

Locally structured grids
Logical tensor product
Fixed banded matrix structure
Direct addressing (⇒ fast)
r-adaptivity

Unstructured macro mesh of tensor product subdomains
**ScaRC – Scalable Recursive Clustering**

- Minimal overlap by extended Dirichlet BCs
- Hybrid multilevel domain decomposition method
- Inspired by parallel MG ("best of both worlds")
  - Multiplicative vertically (between levels), global coarse grid problem (MG-like)
  - Additive horizontally: block-Jacobi / Schwarz smoother (DD-like)
- Hide local irregularities by MGs within the Schwarz smoother
- Embed in Krylov to alleviate Block-Jacobi character

**global BiCGStab**
preconditioned by

**global multilevel** (V 1+1)
additively smoothed by

for all $\Omega_i$: **local multigrid**

coarse grid solver: UMFPACK
Multivariate problems

**Block-structured systems**
- Guiding idea: Tune scalar case once per architecture instead of over and over again per application
- Equation-wise ordering of the unknowns
- Block-wise treatment enables multivariate ScaRC solvers

**Examples**
- Linearised elasticity with compressible material (2x2 blocks)
- Saddle point problems: Stokes, linearised elasticity with (nearly) incompressible material, Navier-Stokes with stabilisation (3x3 blocks, three zero Blocks for Stokes)

\[
\begin{pmatrix}
A_{11} & A_{12} \\
A_{21} & A_{22}
\end{pmatrix}
\begin{pmatrix}
u_1 \\
u_2
\end{pmatrix} = f,
\begin{pmatrix}
A_{11} & 0 & B_1 \\
0 & A_{22} & B_2 \\
B_1^T & B_2^T & 0
\end{pmatrix}
\begin{pmatrix}
v_1 \\
v_2 \\
p
\end{pmatrix} = f,
\]

\(A_{11}\) and \(A_{22}\) correspond to scalar (elliptic) operators
\(\Rightarrow\) Tuned linear algebra and tuned solvers

Introduction
FEAST
Co-processor integration
Results
Conclusions
Co-processor integration into FEAST
Bandwidth in a CPU/GPU node

**Introduction**

**FEAST**

**Co-processor integration**

**Results**

**Conclusions**

Diagram showing the bandwidth in a CPU/GPU node with:
- **CPU** containing processing elements and a cache with 40 GB/s bandwidth.
- **Co-processor** with 20-160 GB/s bandwidth.
- **System memory** with 6-15 GB/s bandwidth.
- **Device memory** with 1-8 GB/s bandwidth.
- Infiniband to next node with 1-2 GB/s bandwidth.
Mixed Precision Multigrid

<table>
<thead>
<tr>
<th>Level</th>
<th>Core2Duo (double)</th>
<th>GTX 280 (mixed)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>time(s) MFLOP/s</td>
<td>time(s) MFLOP/s</td>
</tr>
<tr>
<td>7</td>
<td>0.021 1405</td>
<td>0.009 2788</td>
</tr>
<tr>
<td>8</td>
<td>0.094 1114</td>
<td>0.012 8086</td>
</tr>
<tr>
<td>9</td>
<td>0.453 886</td>
<td>0.026 15179</td>
</tr>
<tr>
<td>10</td>
<td>1.962 805</td>
<td>0.073 21406</td>
</tr>
</tbody>
</table>

- Poisson on unitsquare, Dirichlet BCs, TP grid, *not a matrix stencil*
- 1M DOF, multigrid, FE-accurate in less than 0.1 seconds!
- Converges against wrong solution in single precision
- 27x faster than CPU, exactly same results as pure double
- 1.7x faster than pure double on GPU
- 8800 GTX (correction loop on CPU): 0.44 seconds on level 10
**Minimally invasive integration**

**global BiCGStab**
preconditioned by
  **global multilevel** \((V^{1+1})\)
  additively smoothed by
  for all \(\Omega_i\): **local multigrid**
coarse grid solver: UMFPACK

All outer work: CPU, double
Local MGs: GPU, single
GPU performs preconditioning
Applicable to many co-processors
Minimally invasive integration

**General approach**

- Balance acceleration potential and integration effort
- Accelerate many different applications built on top of one central FE and solver toolkit
- Diverge code paths as late as possible
- No changes to application code!
- Retain all functionality
- Do not sacrifice accuracy

**Challenges**

- Heterogeneous task assignment to maximise throughput
- Limited device memory (modeled as huge L3 cache)
- Overlapping CPU and GPU computations
- Building dense accelerated clusters
Some results
Linearised elasticity

\[
\begin{pmatrix}
A_{11} & A_{12} \\
A_{21} & A_{22}
\end{pmatrix}
\begin{pmatrix}
u_1 \\
u_2
\end{pmatrix} = f
\]

\[
\begin{pmatrix}
(2\mu + \lambda)\partial_{xx} + \mu\partial_{yy} & (\mu + \lambda)\partial_{xy} \\
(\mu + \lambda)\partial_{yx} & \mu\partial_{xx} + (2\mu + \lambda)\partial_{yy}
\end{pmatrix}
\]

Global multivariate BiCGStab
block-preconditioned by
Global multivariate multilevel (V 1+1)
additively smoothed (block GS) by

for all \(\Omega_i\): solve \(A_{11}c_1 = d_1\) by local scalar multigrid
update RHS: \(d_2 = d_2 - A_{21}c_1\)

for all \(\Omega_i\): solve \(A_{22}c_2 = d_2\) by local scalar multigrid

coarse grid solver: UMFPACK

Introduction
FEAST
Co-processor integration
Results
Conclusions
**Accuracy (I)**

- **Same results for CPU and GPU**
- $L_2$ error against analytically prescribed displacements
- Tests on 32 nodes, 512 M DOF
Cantilever beam, aniso 1:1, 1:4, 1:16
Hard, very ill-conditioned CSM test
CG solver: > 2x iterations per refinement
GPU-ScaRC solver: same results as CPU

<table>
<thead>
<tr>
<th>aniso04</th>
<th>L</th>
<th>Iterations</th>
<th>Volume</th>
<th>y-Displacement</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>CPU</td>
<td>GPU</td>
<td>CPU</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>4</td>
<td>1.6087641E-3</td>
<td>1.6087641E-3</td>
</tr>
<tr>
<td>9</td>
<td>4</td>
<td>4</td>
<td>1.6087641E-3</td>
<td>1.6087641E-3</td>
</tr>
<tr>
<td>10</td>
<td>4.5</td>
<td>4.5</td>
<td>1.6087641E-3</td>
<td>1.6087641E-3</td>
</tr>
</tbody>
</table>

| aniso16 | | | |
|---------|---|------------|--------|----------------|
|         |   | CPU | GPU | CPU | GPU | CPU | GPU |
| 8       | 6 | 6  | 6.7176398E-3 | 6.7176398E-3 | -6.6216232E-2 | -6.6216232E-2 |
| 9       | 6 | 5.5 | 6.7176427E-3 | 6.7176427E-3 | -6.6216551E-2 | -6.6216552E-2 |
| 10      | 5.5 | 5.5 | 6.7176516E-3 | 6.7176516E-3 | -6.6217501E-2 | -6.6217502E-2 |
Outdated cluster, dual Xeon EM64T,
- one NVIDIA Quadro FX 1400 per node (one generation behind the Xeons, 20 GB/s BW)
- Poisson problem (left): up to 1.3 B DOF, 160 nodes
- Elasticity (right): up to 1 B DOF, 128 nodes
16 nodes, Opteron X2 2214,
NVIDIA Quadro FX 5600 (76 GB/s BW), OpenGL
Problem size 128 M DOF
Dualcore 1.6x faster than singlecore
GPU 2.6x faster than singlecore, 1.6x than dual
Acceleration analysis

**Speedup analysis**

- Addition of GPUs increases resources
- ⇒ Correct model: strong scalability inside each node
- Accelerable fraction of the elasticity solver: 2/3
- Remaining time spent in MPI and the outer solver

Accelerable fraction $R_{acc}$: 66%
Local speedup $S_{local}$: 9x
Total speedup $S_{total}$: 2.6x
Theoretical limit $S_{max}$: 3x
Stationary Navier-Stokes

\[
\begin{pmatrix}
A_{11} & A_{12} & B_1 \\
A_{21} & A_{22} & B_2 \\
B_1^T & B_2^T & C
\end{pmatrix}
\begin{pmatrix}
u_1 \\
u_2 \\
p
\end{pmatrix}
=
\begin{pmatrix}
f_1 \\
f_2 \\
g
\end{pmatrix}
\]

- 4-node cluster
- Opteron X2 2214
- GeForce 8800 GTX (90 GB/s BW), CUDA
- Driven cavity and channel flow around a cylinder

fixed point iteration
solving linearised subproblems with

- global BiCGStab (reduce initial residual by 1 digit)
- Block-Schurcomplement preconditioner

1) approx. solve for velocities with
- global MG (V_{1+0}), additively smoothed by
  - for all \( \Omega_i \): solve for \( u_1 \) with local MG
  - for all \( \Omega_i \): solve for \( u_2 \) with local MG

2) update RHS:

\[
d_3 = -d_3 + B^T(c_1 \cdot c_2)^T
\]

3) scale \( c_3 = (M_p^L)^{-1}d_3 \)

---

Introduction

FEAST

Co-processor integration

Results

Conclusions
Navier-Stokes results

Speedup analysis

<table>
<thead>
<tr>
<th></th>
<th>$R_{\text{acc}}$</th>
<th>$S_{\text{local}}$</th>
<th>$S_{\text{total}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>L9</td>
<td>L10</td>
<td>L9</td>
</tr>
<tr>
<td>DC Re100</td>
<td>41%</td>
<td>46%</td>
<td>6x</td>
</tr>
<tr>
<td>DC Re250</td>
<td>56%</td>
<td>58%</td>
<td>5.5x</td>
</tr>
<tr>
<td>Channel flow</td>
<td>60%</td>
<td>–</td>
<td>6x</td>
</tr>
</tbody>
</table>

Important consequence: Ratio between assembly and linear solve changes significantly

<table>
<thead>
<tr>
<th></th>
<th>DC Re100</th>
<th>DC Re250</th>
<th>Channel flow</th>
</tr>
</thead>
<tbody>
<tr>
<td>plain</td>
<td>accel.</td>
<td>plain</td>
<td>accel.</td>
</tr>
<tr>
<td>29:71</td>
<td>50:48</td>
<td>11:89</td>
<td>25:75</td>
</tr>
<tr>
<td>13:87</td>
<td>26:74</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Conclusions
Hardware-oriented numerics prevents existing codes being worthless in a few years

Mixed precision schemes exploit the available bandwidth without sacrificing accuracy

GPUs as local preconditioners in a large-scale parallel FEM package

Not limited to GPUs, applicable to all kinds of hardware accelerators

Minimally invasive approach, no changes to application code

Excellent local acceleration, global acceleration limited by ‘sequential’ part

Future work: Design solver schemes with higher acceleration potential without sacrificing numerical efficiency
Acknowledgements

Collaborative work with

FEAST group (TU Dortmund)
Robert Strzodka (Max Planck Institut Informatik)
Jamaludin Mohd-Yusof, Patrick McCormick (Los Alamos National Laboratory)

http://www.mathematik.tu-dortmund.de/~goeddeke

Supported by Deutsche Forschungsgemeinschaft, project TU 102/22-1, TU 102/22-2, TU 102/27-1, TU102/11-3