Mixed-Precision GPU-Multigrid Solvers with Strong Smoothers

Dominik Göddeke

Institut für Angewandte Mathematik (LS3)
TU Dortmund
dominik.goeddeke@math.tu-dortmund.de

ILAS 2011
Mini-Symposium: Parallel Computing in Numerical Linear Algebra
Braunschweig, August 24, 2011
The Big Picture

**Hardware evolution**
- Memory wall: Data movement cost prohibitively expensive
- Power wall: Nuclear power plant for each machine (in the cloud)?
- ILP wall: ‘Automagic’ maximum resource utilisation?
- Memory wall + power wall + ILP wall = brick wall

**Inevitable paradigm shift: Parallelism and heterogeneity**
- In a single chip: singlecore → multicore, manycore, . . .
- In a workstation (cluster node): NUMA, CPUs and GPUs, . . .
- In a big cluster: different nodes, communication characteristics, . . .

**This is our problem as mathematicians**
- Affects standard workstations and even laptops
- Compilers and (most often) libraries don’t hide these issues from us
Consequences for Numerics

**Parallelism is inevitable**
- Impossible to exploit ever increasing peak performance
- Sequential codes even run slower on newer hardware (!)

**Challenges**
- Technical: Compilers can’t solve these problems, libraries are limited
- Numerical: Traditional methods often contrary to hardware trends
- Goal: Redesign existing numerical schemes (and invent new ones) to work well in the fine-grained parallel setting

**GPUs (‘manycore’) are forerunners of this development**
- 10,000s of simultaneously active threads
- Promises of significant speedups
- Focus of this talk: iterative solvers for sparse systems
GPUs and the Memory Wall Problem

Diagram showing the connection between the GPU, CPU, device memory, and host memory. The diagram includes the following key points:

- **GPU**: Contains SIMT-MP units and is connected to the device memory.
- **CPU-Socket**: Contains cores and caches, with shared caches and MCs.
- **Device Memory**: Connected to the GPU via a 20-180 GB/s connection.
- **Host Memory**: Connected to the CPU via a 6-35 GB/s connection.
- **Infiniband (PCIe)**: Connection between device memory and host memory, with a 1-8 GB/s connection.
Mixed Precision Iterative Refinement

Combatting the memory wall problem
Motivation

Switching from double to single precision (DP→SP)

- 2x effective memory bandwidth, 2x effective cache size
- At least 2x compute speed (often 4–12x)

Problem: Condition number

- For all problems in this talk: \( \text{cond}_2(A) \sim h_{\text{min}}^{-2} \)
- Theory for linear system \( Ax = b \)

\[
\text{cond}_2(A) \sim 10^s; \frac{\|A + \delta A\|}{\|A\|}, \frac{\|b + \delta b\|}{\|b\|} \sim 10^{-k} (k > s) \quad \Rightarrow \quad \frac{\|x + \delta x\|}{\|x\|} \sim 10^{s-k}
\]

In our setting

- Truncation error in 7–8th digit increased by \( s \) digits
Numerical Example

**Poisson problem on unit square**

- Simple yet fundamental
- $\text{cond}_2(A) \approx 10^5$ for $L = 10$ (1M bilinear FE, regular grid)
- Condition number usually much higher: anisotropies in grid and operator

<table>
<thead>
<tr>
<th>Level</th>
<th>Data+Comp. in DP $L_2$ Error</th>
<th>Data in SP, Compute in DP $L_2$ Error</th>
<th>Data+Comp. in SP $L_2$ Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>1.1102363E-3 4.00</td>
<td>1.1102371E-3 4.00</td>
<td>1.1111655E-3 4.00</td>
</tr>
<tr>
<td>6</td>
<td>2.7752805E-4 4.00</td>
<td>2.7756739E-4 4.00</td>
<td>2.8704684E-4 3.87</td>
</tr>
<tr>
<td>7</td>
<td>6.9380072E-5 4.00</td>
<td>6.9419428E-5 4.00</td>
<td>1.2881795E-4 2.23</td>
</tr>
<tr>
<td>8</td>
<td>1.7344901E-5 4.00</td>
<td>1.7384278E-5 3.99</td>
<td>4.2133101E-4 0.31</td>
</tr>
<tr>
<td>9</td>
<td>4.3362353E-6 4.00</td>
<td>4.3757082E-6 3.97</td>
<td>2.1034461E-3 0.20</td>
</tr>
<tr>
<td>10</td>
<td>1.0841285E-6 4.00</td>
<td>1.1239630E-6 3.89</td>
<td>8.8208778E-3 0.24</td>
</tr>
</tbody>
</table>

⇒ Single precision insufficient for moderate problem sizes already
Iterative refinement

- Established algorithm to provably guarantee accuracy of computed results (within given precision)
  - High precision: \( \mathbf{d} = \mathbf{b} - \mathbf{A}\mathbf{x} \) (cheap)
  - Low precision: \( \mathbf{c} = \mathbf{A}^{-1}\mathbf{d} \) (expensive)
  - High precision: \( \mathbf{x} = \mathbf{x} + \mathbf{c} \) (cheap) and iterate (expensive?)

- Convergence to high precision accuracy if \( \mathbf{A} \) ‘not too ill-conditioned’

- Theory: Number of iterations linear in \( \log(\text{cond}_2(\mathbf{A})) \) and \( \log(\varepsilon_{\text{high}}/\varepsilon_{\text{low}}) \)

New idea (hardware-oriented numerics)

- Use this algorithm to improve time to solution and thus efficiency of linear system solvers

- Goal: Result accuracy of high precision with speed of low precision floating point format
Iterative Refinement for Large Sparse Systems

**Refinement procedure not immediately applicable**
- ‘Exact’ solution using ‘sparse LU’ techniques too expensive
- Convergence of iterative methods not guaranteed in single precision

**Solution**
- Interpretation as a preconditioned mixed precision defect correction iteration
  \[
  x^{(k+1)}_{DP} = x^{(k)}_{DP} + C^{-1}_{SP}(b_{DP} - A_{DP}x^{(k)}_{DP})
  \]
- Preconditioner \( C_{SP} \) in single precision: ‘Gain digit(s)’ or 1-3 MG cycles instead of exact solution

**Results (MG and Krylov for Poisson problem)**
- Speedup at least 1.7x (often more) without loss in accuracy
- Asymptotic optimal speedup is 2x (bandwidth limited)
Grid- and Matrix Structures

Flexibility $\leftrightarrow$ Performance
Grid- and Matrix Structures

**General sparse matrices (on unstructured grids)**
- CSR (and variants): general data structure for arbitrary grids
- Maximum flexibility, but during SpMV
  - Indirect, irregular memory accesses
  - Index overhead reduces already low arithm. intensity further
- Performance depends on nonzero pattern (mesh numbering)

**Structured matrices**
- Example: structured grids, suitable numbering ⇒ band matrices
- Important: no stencils, fully variable coefficients
- Direct regular memory accesses (fast), mesh-independent performance
- Structure exploitation in the design of MG components (later)
Approach in FEAST

**Combination of respective advantages**

- Global macro-mesh: unstructured, flexible
- Local micro-meshes: structured (logical TP-structure), fast
- Important: structured \(\neq\) cartesian meshes!
- Reduce numerical linear algebra to sequences of operations on structured data (maximise locality)
- Developed for larger clusters, but generally useful
Example

Poisson on unstructured domain

- Nehalem vs. GT200, $\approx 2$M bilinear FE, MG-JAC solver
- Unstructured formats highly numbering-dependent
- Multicore 2–3x over singlecore, GPU 8–12x over multicore
- Banded format (here: 8 ‘blocks’) 2–3x faster than best unstructured layout and predictably on par with multicore
Parallelising Inherently Sequential Operations

Multigrid with strong smoothers
Lots of parallelism available in inherently sequential operations
Test case: anisotropic diffusion in generalised Poisson problem

- $-\text{div} \ (G \ \nabla u) = f$ on unit square (one FEAST patch)
- $G = I$: standard Poisson problem, $G \neq I$: arbitrarily challenging
- Example: $G$ introduces anisotropic diffusion along some vector field

Only multigrid with a strong smoother is competitive
Gauß-Seidel Smoother

Disclaimer: Not necessarily a good smoother, but a good didactical example.

**Sequential algorithm**

- Forward elimination, sequential dependencies between matrix rows
- Illustrative: coupling to the left and bottom

**1st idea: classical wavefront-parallelisation (exact)**

![Wavefront diagram]

- Pro: always works to resolve *explicit* dependencies
- Con: irregular parallelism and access patterns, implementable?
2nd idea: decouple dependencies via multicolouring (inexact)

- Jacobi (red) – coupling to left (green) – coupling to bottom (blue) – coupling to left and bottom (yellow)

Analysis

- Parallel efficiency: 4 sweeps with $\approx \frac{N}{4}$ parallel work each
- Regular data access, but checkerboard pattern challenging for SIMD/GPUs due to strided access
- Numerical efficiency: sequential coupling only in last sweep
3rd idea: multicolouring = renumbering

- After decoupling: ‘standard’ update (left+bottom) is suboptimal
- Does not include all already available results

Recoupling: Jacobi (red) – coupling to left and right (green) – top and bottom (blue) – all 8 neighbours (yellow)

- More computations than standard decoupling
- Experiments: convergence rates of sequential variant recovered (in absence of preferred direction)
Tridiagonal Smoother (Line Relaxation)

**Starting point**
- Good for ‘line-wise’ anisotropies
- ‘Alternating Direction Implicit (ADI)’ technique alternates rows and columns
- CPU implementation: Thomas-Algorithm (inherently sequential)

**Observations**
- One independent tridiagonal system per mesh row
- ⇒ top-level parallelisation across mesh rows
- Implicit coupling: wavefront and colouring techniques not applicable
Tridiagonal Smoother (Line Relaxation)

**Cyclic reduction for tridiagonal systems**

- Exact, stable (w/o pivoting) and cost-efficient
- Problem: classical formulation parallelises computation but not memory accesses on GPUs (bank conflicts in shared memory)
- Developed a better formulation, 2-4x faster
- Index nightmare, general idea: recursive padding between odd and even indices on all levels
Starting point

- CPU implementation: shift previous row to RHS and solve remaining tridiagonal system with Thomas-Algorithm
- Combined with ADI, this is the best general smoother (we know) for this matrix structure

Observations and implementation

- Difference to tridiagonal solvers: mesh rows depend sequentially on each other
- Use colouring ($\#c \geq 2$) to decouple the dependencies between rows (more colours = more similar to sequential variant)
**Test problem:** generalised Poisson with anisotropic diffusion

- Total efficiency: time per unknown per digit (µs)
- Mixed precision iterative refinement multigrid solver

---

**Evaluation: Total Efficiency on CPU and GPU**

- **CPU**
  - GS
  - ADITRIDI
  - ADITRIGS

- **GPU**
  - GS(4C)
  - GS(FULL)
  - ADITRIDI
  - ADITRIGS(2C)
Summary: smoother parallelisation

- Factor 10-30 (dep. on precision and smoother selection) speedup over already highly tuned CPU implementation
- Same functionality on CPU and GPU
- Balancing of numerical and parallel efficiency (hardware-oriented numerics)
Extension to Heterogeneous Clusters
FEAST on Heterogeneous Clusters

pressure + isolines
(elevation plot)

magnitude of velocity + coarse grid

a)  
b)  
c)  
d)
Summary and Conclusions
Summary

**Hardware**
- Paradigm shift: Heterogeneity, parallelism and specialisation
- Locality and parallelism on many levels
  - In one GPU (fine-granular)
  - In a compute node between heterogeneous resources (medium-granular)
  - In big clusters between compute nodes (coarse-granular)

**Hardware-oriented numerics**
- Design new numerical methods ‘matching’ the hardware
- Only way to achieve future-proof continuous scaling
Acknowledgements

Collaborative work with

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

Supported by

- DFG: TU 102/22-1, TU 102/22-2
- BMBF: HPC Software für skalierbare Parallelrechner: SKALB project 01IH08003D

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