CGMN and CARP-CG on clusters of multi-core CPUs

Jonas Thies

German Aerospace Center (DLR)
Simulation and Software Technology
High Performance Computing group
Jonas.Thies@DLR.de
Outline

Motivation

The CGMN algorithm

Parallelization

Convergence behavior

Performance aspects
Motivation
Overview of this talk

Have you met a linear system that didn’t want to be solved iteratively?

Aims of this talk:

• bring CGMN to your attention
• show two competing parallelization schemes
• discuss implementation aspects
Example application that is ‘hard to precondition’

- Graphene: Carbon atoms in a 2D hexagonal mesh
- Hamiltonian: random diagonal entries $|a_{ii}| < |a_{i\neq j}|$
- symmetric and completely indefinite
- Task: find 10-1000 innermost eigenpairs
Hardware challenges for solvers

- multi-level parallelism
- heterogenous hardware
- complex memory/cache hierarchy
- resilience: fast recovery if a node fails (using some flavor of checkpoint/restart)
Drawbacks of common parallel preconditioners

- Domain decomposition methods (FETI, Schwarz+ILU etc)
  - high memory demands (bandwidth bottleneck)
  - resilience: checkpointing the preconditioner not practical
  - load balancing: only static

- AMG
  - limited to certain problem classes (e.g. elliptic PDEs)
  - setup phase complex and hard to parallelize
  - needs parallel smoother
  - more communication on coarser grids
The CGMN algorithm
Kaczmarz iteration

- SOR on the minimum norm problem (MNP),
  \[ \mathbf{A} \mathbf{A}^T \mathbf{y} = \mathbf{b}, \mathbf{x} = \mathbf{A}^T \mathbf{y}. \]

- Equivalent to Kaczmarz iteration for \( \mathbf{A} \mathbf{x} = \mathbf{b} \) (KACZ)
- Forward + backward KACZ \( \implies \) SSOR on the MNP

\[
\mathbf{x}^{(k+1)} = \mathbf{Q}_{SSOR} \mathbf{x}^{(k)} + \mathbf{R}_{SSOR} \mathbf{b},
\]
with \( \mathbf{Q}_{SSOR} = \mathbf{Q}_1 \mathbf{Q}_2 \cdots \mathbf{Q}_n \mathbf{Q}_{n-1} \cdots \mathbf{Q}_1 \),
\[
\mathbf{Q}_i = \mathbf{I} - \frac{\omega}{\|\mathbf{a}_{i,:}\|_2} \mathbf{a}_{i,:}^H \mathbf{a}_{i,:},
\]
- \( \mathbf{Q}_i \): projections onto \( i \)'th row \( \mathbf{a}_{i,:} \) of \( \mathbf{A} \).
CGMN (Björck & Elfving, 1979)

- CG for \((I - Q_{SSOR})x = R_{SSOR} b\) converges even though the system matrix is only symmetric positive semi-definite.
- implicit SSOR preconditioning
- efficient row-wise formulation
- extremely robust: \(A\) may be non-symmetric, singular, non-square etc.
- row scaling alleviates issue of ‘squared condition number’
Core operation: KACZ sweep (in CRS format)

\[ \text{spMVM, } y \leftarrow A \times x \]

1: \textbf{for} (i=0; i<n; i++) \textbf{do}
2: \hspace{1em} tmp=0
3: \hspace{1em} \textbf{for} (j=rptr[i]; j<rptr[i+1]; j++) \textbf{do}
4: \hspace{2em} tmp+=val[j]*x[col[j]];
5: \hspace{1em} \textbf{end for}
6: \hspace{1em} // non-temporal store
7: \hspace{1em} y[j]=tmp;
8: \textbf{end for}

Kaczmarz update, \( x \leftarrow KACZ(A, x, b, \omega) \)

1: \textbf{for} (i=0; i<n; i++) \textbf{do}
2: \hspace{1em} tmp=-b[i]; \hspace{0.5em} // b!=0 only in 1st iteration
3: \hspace{1em} nrm=0;
4: \hspace{1em} \textbf{for} (j=rptr[i]; j<rptr[i+1]; j++) \textbf{do}
5: \hspace{2em} tmp+=val[j]*x[col[j]];
6: \hspace{2em} nrm+=val[j]*val[j];
7: \hspace{1em} \textbf{end for}
8: \hspace{1em} // update x
9: \hspace{1em} tmp*=omega/nrm;
10: \hspace{1em} \textbf{for} (j=rptr[i]; j<rptr[i+1]; j++) \textbf{do}
11: \hspace{2em} x[cols[j]]-=tmp*val[j];
12: \hspace{1em} \textbf{end for}
13: \textbf{end for}
Parallelization
Multi-Coloring (MC)

- requires “distance 2” coloring
- software: ColPack
  
  [Link to software](http://cscapes.cs.purdue.edu/coloringpage/software.htm)
Component-Averaged Row Projection (CARP)

- Gordon & Gordon, 2005
- Kaczmarz locally
- write to halo
- exchange and average
equiv. to KACZ on a superspace of $\mathbb{R}^n$
Hybrid method: MC_CARP-CG

- global MC would require...
  - an extremely scalable coloring method
  - very well-balanced colors
  - many global sync-points (ca. 15 colors in our examples)

- global CARP requires ...
  - large number of MPI procs
  - increasing amount of ‘interior halo elements’
  - non-trivial implementation on GPU and Xeon Phi

Idea: node-local MC with MPI-based CARP between the nodes
Convergence behavior
Experimental setup

- Machine: Intel Xeon “Ivy Bridge”
- 10 cores/socket, 2 sockets/node
- InfiniBand between nodes

Test cases: conv. dominated PDE, Anderson localization
- 3D 7-point stencil
- octree ordering
- suitable boundary conditions
Application 1: convection dominated flow

\[-(e^{-xyz} U_x)_x - (e^{xyz} U_y)_y - (e^{(1-x)(1-y)(1-z)} U_z)_z\]

\[+ \frac{40}{\delta x} \sin(\pi y) U_x + \frac{2}{\delta y} \sin(\pi z) U_y + \frac{2}{\delta z} \sin(\pi x) U_z = F\]
Application 2: Anderson localization

- 7-point stencil
- diag: random numbers from $[-\frac{L}{2}, \frac{L}{2}]$

Off-diagonal elements: -1

Small complex diagonal shift ($10^{-2}i$)

$L = 16.5$

$L = 1.0$
CARP-CG Convergence for increasing problem size

Application 1

![Bar chart for Application 1](chart)

- ω = 1.0
- ω = 1.5

Application 2

![Bar chart for Application 2](chart)

- Grid size: 64, 128, 256, 512
Performance aspects
Performance on a multi-core CPU

- memory bandwidth limits performance of spMV and KACZ
- ordering causes scattered access to x
Performance on a multi-core CPU

- memory bandwidth limits performance of spMV and KACZ
- ordering causes scattered access to \( x \)
- false sharing prevents socket scaling
Performance on a multi-core CPU

- memory bandwidth limits performance of spMV and KACZ
- ordering causes scattered access to x
- false sharing prevents socket scaling
- more optimizations possible but non-trivial
Performance hazards of multi-coloring approach
Weak scaling (8 cores/socket, $64^3$ unknowns/core)
Summary

- CGMN is a useful method for matrices with small diagonal entries
- also useful for e.g. Helmholtz equations
- runs as fast as unpreconditioned CG on one CPU core
- parallelization schemes
  - distance-2 coloring bad for performance and overrelaxation
  - CARP gives very effective domain decomposition
  - but with quite some memory overhead
  - hybrid MC_CARP may be a good choice, e.g. for block methods
Acknowledgement and references

DFG project ESSEX
(Equipping Sparse Solvers for the EXa-scale)