Efficient Iterative Solution of Large Sparse Linear Systems on a Cluster of Geographically Separated Clusters

Tijmen Collignon

Martin B. van Gijzen

Delft University of Technology
Delft Institute of Applied Mathematics
Delft, the Netherlands

5th International Workshop on Parallel Matrix Algorithms and Applications (PMAA’08)
June, 2008, Neuchâtel, Switzerland
Introduction

The Problem

Problem Setting

Efficient and iterative solution of large sparse linear systems

\[ Ax = b, \quad A \in \mathbb{R}^{n \times n} \]

on large and heterogeneous networks of computers

Properties of Coefficient Matrix \( A \)

- Large problem size \((n > 10^7)\)
- Sparse
- Symmetric positive semi–definite
- Large jumps in coefficient \( \rightarrow \) large condition number
- Finite–difference discretisation of 3D Poisson equation from bubbly flow problem

Properties of Computing Environment

- Large and aggregated
- Geographically separated
- Heterogeneous
- Volatile resources
- Highly expensive synchronisation
The Algorithm

(A)synchronous block Jacobi iteration

The problem

Solve in parallel \(Ax = b\) using \(p\) processes

\[
A = \begin{bmatrix}
A_{11} & A_{12} & \cdots & A_{1p} \\
A_{21} & A_{22} & \cdots & A_{2p} \\
\vdots & \vdots & \ddots & \vdots \\
A_{p1} & A_{p2} & \cdots & A_{pp}
\end{bmatrix}, \quad 
\begin{bmatrix}
x_1 \\
x_2 \\
\vdots \\
x_p
\end{bmatrix}, \quad 
\begin{bmatrix}
b_1 \\
b_2 \\
\vdots \\
b_p
\end{bmatrix}
\]

Block Jacobi iteration

1: Initialize \(x^0\);
2: for \(k = 1, 2, \ldots \) until convergence do
3: for \(i = 1, 2, \ldots, p\) do
4: Approximate/Solve: \(A_{ii}x^k_i = b_i - \sum_{j=1,j\neq i}^{p} A_{ij}x^{k-1}_j\); (in parallel)
5: end for
6: end for
The Algorithm

(A)synchronous block Jacobi iteration

The problem

Solve in parallel \( Ax = b \) using \( p \) processes

\[
A = \begin{bmatrix}
A_{11} & A_{12} & \cdots & A_{1p} \\
A_{21} & A_{22} & \cdots & A_{2p} \\
\vdots & \vdots & \ddots & \vdots \\
A_{p1} & A_{p2} & \cdots & A_{pp}
\end{bmatrix}, \quad x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_p \end{bmatrix}, \quad b = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_p \end{bmatrix}
\]

Asynchronous block Jacobi iteration

1: Initialize \( x^0 \);
2: for \( k = 1, 2, \ldots \) until convergence do
3: \hspace{1em} for \( i = 1, 2, \ldots, p \) do
4: \hspace{2em} Approximate/Solve: \( A_{ii} x_i^{k'} = b_i - \sum_{j=1, j \neq i}^{p} A_{ij} x_j^{k'-1} \); (using most recent \( x \))
5: \hspace{1em} end for
6: end for
Parallel Asynchronous Iterative Methods

Key advantages asynchronous block Jacobi iteration

- No synchronisation points
- Coarse–grain
- Efficient overlap communication/computation
## The Algorithm

### Best of Both Worlds

<table>
<thead>
<tr>
<th>asynchronous block Jacobi iterations</th>
<th>synchronous Krylov methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>coarse–grain (+)</td>
<td></td>
</tr>
<tr>
<td>no synchronisation points (+)</td>
<td></td>
</tr>
<tr>
<td>Grid computing (+)</td>
<td></td>
</tr>
<tr>
<td>may diverge (−)</td>
<td></td>
</tr>
<tr>
<td>slow (−)</td>
<td></td>
</tr>
</tbody>
</table>

...
## The Algorithm

### Best of Both Worlds

<table>
<thead>
<tr>
<th>asynchronous block Jacobi iterations</th>
<th>synchronous Krylov methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>coarse–grain (+)</td>
<td>fine–grain, due to inner products (−)</td>
</tr>
<tr>
<td>no synchronisation points (+)</td>
<td>synchronous (−)</td>
</tr>
<tr>
<td>Grid computing (+)</td>
<td>Cluster computing (+)</td>
</tr>
<tr>
<td>may diverge (−)</td>
<td>always convergent, in theory (+)</td>
</tr>
<tr>
<td>slow (−)</td>
<td>fast (+)</td>
</tr>
</tbody>
</table>

...
## The Algorithm

### Best of Both Worlds

<table>
<thead>
<tr>
<th>asynchronous block Jacobi iterations</th>
<th>synchronous Krylov methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>coarse–grain (+)</td>
<td>fine–grain, due to inner products (−)</td>
</tr>
<tr>
<td>no synchronisation points (+)</td>
<td>synchronous (−)</td>
</tr>
<tr>
<td>Grid computing (+)</td>
<td>Cluster computing (+)</td>
</tr>
<tr>
<td>may diverge (−)</td>
<td>always convergent, in theory (+)</td>
</tr>
<tr>
<td>slow (−)</td>
<td>fast (+)</td>
</tr>
<tr>
<td>⋮</td>
<td>⋮</td>
</tr>
<tr>
<td>inner (preconditioning) iteration</td>
<td>outer (flexible) iteration</td>
</tr>
</tbody>
</table>
The Algorithm

Inner–Outer Method

Flexible CG:
– update solution \( x_{k+1} = x_k + u_k \)
– target precision \( \epsilon_{\text{outer}} \)

Asynchronous block Jacobi iteration:
– evaluate \( u_k = M(r_k) \)
– maximum \( T_{\text{max}} \) seconds

Preconditioned CG:
– local subdomain solves
– independent and inexact
\( A_{22}u = y \)
\( A_{11}u = y \)
\( A_{33}u = y \)
\( A_{ii}u = v \)
Target Hardware. DAS–3: Five Clusters, One System

- VU University (85)
- Leiden University (32)
- Delft University of Technology (68)
- University of Amsterdam (VL-e) (41)
- University of Amsterdam (MultimediaN) (46)
Preliminary Experimental Results (1/3)

Experimental Setup

- Problem size per preconditioning node fixed to 250,000 equations
- Number of nodes \( s \in \{5, 10, 15, \ldots, 70\} \) distributed equally among 5 clusters
- Maximum problem size: 22,000,000
- Devote 120s to each preconditioning step
- 5 truncation vectors
- One (1) bubble at center of unit 3D domain with density jump of \( 10^3 \)
- Sequential outer iteration on single TU Delft cluster node
- Asynchronous vs synchronous preconditioning (purely illustrative)
- CRAC (Communication Routines for Asynchronous Computations) grid middleware
Preliminary Experimental Results (2/3)

Key Observations

- Time per outer iteration dominated by preconditioning
- Limited increase due to sequential outer iteration
- Computing time almost independent of problem size
Highly Unfavourable Conditions

Only mild increase in number of outer iterations, despite:

- Condition number of matrix increases
- Number of subdomains increases/effect of preconditioner deteriorates
- Heterogeneity of environment increases
Main Conclusions and Current Work

Main Conclusions/Contributions

- Mild increase in number of outer iterations for increasing number of nodes/problem size
- Great potential for an efficient iterative algorithm for solving large linear systems in Grid Computing
- CRAC extremely useful for implementing (a)synchronous algorithms
- Valuable real–world and large–scale experimental results

Current Work

- Robustness issues (how accurate/long preconditioning iteration)
- Parallel outer iteration, either using a single cluster or using all five clusters
- Resource–aware load balancing to avoid (potential) desynchronisation
- Extensive comparison studies
Contact Information and Further Reading

Contact me!

- **Email:** tijmen@cwi.nl
- **Website:** http://ta.twi.tudelft.nl/nw/users/collignon/

Acknowledgements

- Stéphane Domas from LIFC for information on CRAC
- GREMLINS team (LIFC and LIP6)
- Netherlands Organisation for Scientific Research (NWO) for the use of the DAS–3
- This project is funded by the Delft Centre for Computational Science and Engineering
DAS–3: five clusters, one system

DAS–3 consists of 272 dual AMD Opteron compute nodes, spread out over five clusters, located at the four universities. The system has been built by ClusterVision. DAS–3 is rather heterogeneous in design:

Overview

<table>
<thead>
<tr>
<th>Cluster</th>
<th>Nodes</th>
<th>Type</th>
<th>Speed</th>
<th>Memory</th>
<th>Storage</th>
<th>HDDs</th>
<th>Network</th>
</tr>
</thead>
<tbody>
<tr>
<td>VU</td>
<td>85</td>
<td>dual</td>
<td>2.4 GHz</td>
<td>4 GB</td>
<td>10 TB</td>
<td>250 GB</td>
<td>Myri-10G/GbE</td>
</tr>
<tr>
<td>LU</td>
<td>32</td>
<td>single</td>
<td>2.6 GHz</td>
<td>4 GB</td>
<td>10 TB</td>
<td>400 GB</td>
<td>Myri-10G/GbE</td>
</tr>
<tr>
<td>UvA(a)</td>
<td>41</td>
<td>dual</td>
<td>2.2 GHz</td>
<td>4 GB</td>
<td>5 TB</td>
<td>250 GB</td>
<td>Myri-10G/GbE</td>
</tr>
<tr>
<td>TUD</td>
<td>68</td>
<td>single</td>
<td>2.4 GHz</td>
<td>4 GB</td>
<td>5 TB</td>
<td>250 GB</td>
<td>GbE</td>
</tr>
<tr>
<td>UvA(b)</td>
<td>46</td>
<td>single</td>
<td>2.4 GHz</td>
<td>4 GB</td>
<td>3 TB</td>
<td>1.5 TB</td>
<td>Myri-10G/GbE</td>
</tr>
<tr>
<td>total</td>
<td>272</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Pros/cons asynchronous iterative method as preconditioner

Some advantages
- increased fault tolerance
- no expensive convergence detection
- preconditioner easy to parallelise in Grid environments
- potential for efficient multi-level preconditioning
- best of both worlds

Some disadvantages
- difficult to determine amount of preconditioning
- many other parameters