ADAPTING ITERATIVE ALGORITHMS FOR SOLVING LARGE SPARSE LINEAR SYSTEMS FOR EFFICIENT USE ON THE CDC CYBER 205

DAVID R. KINCAID
AND
DAVID M. YOUNG

CENTER FOR NUMERICAL ANALYSIS
UNIVERSITY OF TEXAS AT AUSTIN

AUSTIN, TEXAS
ADAPTING ITERATIVE ALGORITHMS
FOR SOLVING LARGE SPARSE LINEAR SYSTEMS
FOR EFFICIENT USE ON THE CDC CYBER 205

David R. Kincaid
David M. Young

Center for Numerical Analysis
University of Texas at Austin
Austin, TX 78712

* Adapting and designing mathematical software to achieve optimum performance on the CYBER 205 will be discussed

* Comments and observations are made in light of recent work done at the Center for Numerical Analysis on
  - modifying the ITPACK software package
  - writing new software for vector supercomputers
Research goal - develop very efficient vector algorithms and software for solving large sparse linear systems using iterative methods

(older) SCALAR APPROACH - develop algorithms that minimize either number of iterations or arithmetic operations

* Not necessarily the correct approach for vector computers *

(newer) VECTOR APPROACH - avoid operations such as table lookups, indirect addressing, etc. that are inefficient on a vector computer, i.e., non-vectorizable

* Fully vectorizable code may involve more arithmetic operations but can be executed at a very high rate of speed *

* Advances in high performance computers and in computer architecture necessitates additional research in mathematical software to find suitable algorithms for the supercomputers of today and of the future *
THE VECTORIZATION OF THE ITPACK SOFTWARE PACKAGE

Scalar ITPACK:

package for solving large sparse linear systems
7 iterative algorithms available
sparse storage format used
Kincaid, Respess, Young, & Grimes [1982]
ITPACK 2C (ALGORITHM 586) in T.O.M.S.
"Transactions on Mathematical Software"

VECTORIZATION:

- First step: look for obvious vectorization changes since this
  was a large package of over 11,000 lines of code and we did not
  want to completely rewrite it

- Vector ITPACK (standard Fortran version): used a minimum of
  vector syntax available in CYBER 200 Fortran for a portable
  version of Vector ITPACK 2C

- Vector ITPACK (CYBER 205 version): a modified version of
  Vector ITPACK written using CYBER 200 Fortran vector syntax
  where possible
ADOPTING SCALAR ITPACK 2C FOR HIGH PERFORMANCE COMPUTERS

- DO loops which had been unrolled for scalar optimization were not recognized as vectorizable by optimizing vector compilers.

- These loops were rewritten as simple tight DO loops so that they would be executed in vector mode.

- The sparse storage scheme used for the matrix in Scalar ITPACK was row-oriented and inhibited vectorization (The IA-JA-A data structure as in Yale software YSMP used.)

- A column-oriented data structure was used in Vector ITPACK to increase vectorization (The COEF-JCOEF data structure as in Purdue software ELLPACK used.)

- The version of Vector ITPACK specifically for the CYBER 205 was tested on the CYBER 205 at Colorado State University (CSU) and has been added to their Program Library.

- The improvements in time of the vector syntax version over the one in standard Fortran were not as significant as we had anticipated.

- The automatic vectorization available in the CYBER 205 Fortran compiler did a very good job of optimization and vectorization.

Moral: vector syntax best when used in designing and writing new code.
PROBLEM:
\[
\begin{cases}
    u_{xx} + 2u_{yy} = 0 & \text{on } S=(0,1) \times (0,1) \\
    u = 1 + xy & \text{on boundary of } S
\end{cases}
\]

Discretization: standard 5-point finite difference formula

Stopping Criterion: \(5.0 \times 10^{-6}\)

Mesh Sizes: \(1/16; 1/32; 1/64; 1/128; 1/256\)

Number of Unknowns: 225; 961; 3969; 16,129; 65,025

Computer: CSU CYBER 205

CYBER 200 Fortran: Large pages, unsafe vectorization

Scalar ITPACK (unrolled DO-loops & YALE storage used; T.O.M.S. version)

Modified Scalar ITPACK (rolled DO-loops & minor changes: Q8SDOT used)

Vector ITPACK (rolled DO-loops, ELLPACK storage, & CYBER 200 Fortran vector syntax used)
TABLE I: CHANGING SPARSE STORAGE

(Iteration Times in Seconds with H = 1/64)

<table>
<thead>
<tr>
<th>Method</th>
<th>Iterations</th>
<th>Scalar ITPACK</th>
<th>Modified Scalar ITPACK</th>
<th>Vector ITPACK</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(Natural Ordering)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JACOBI CG</td>
<td>178</td>
<td>2.509</td>
<td>2.184</td>
<td>.262</td>
</tr>
<tr>
<td>JACOBI SI</td>
<td>362</td>
<td>5.214</td>
<td>4.480</td>
<td>.580</td>
</tr>
<tr>
<td>SOR</td>
<td>216</td>
<td>4.700</td>
<td>4.597</td>
<td>2.453</td>
</tr>
<tr>
<td>SSOR CG</td>
<td>34</td>
<td>1.970</td>
<td>1.788</td>
<td>.831</td>
</tr>
<tr>
<td>SSOR SI</td>
<td>43</td>
<td>1.791</td>
<td>1.682</td>
<td>.970</td>
</tr>
<tr>
<td></td>
<td>(Red-Black Ordering)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JACOBI CG</td>
<td>178</td>
<td>2.402</td>
<td>2.056</td>
<td>.268</td>
</tr>
<tr>
<td>JACOBI ST</td>
<td>362</td>
<td>4.987</td>
<td>4.209</td>
<td>.590</td>
</tr>
<tr>
<td>SOR</td>
<td>196</td>
<td>4.110</td>
<td>4.017</td>
<td>.523</td>
</tr>
<tr>
<td>SSOR CG</td>
<td>341</td>
<td>20.327</td>
<td>18.472</td>
<td>2.177</td>
</tr>
<tr>
<td>SSOR SI</td>
<td>196</td>
<td>7.734</td>
<td>6.690</td>
<td>.701</td>
</tr>
<tr>
<td>RS CG</td>
<td>90</td>
<td>1.445</td>
<td>1.358</td>
<td>.118</td>
</tr>
<tr>
<td>RS SI</td>
<td>182</td>
<td>2.980</td>
<td>2.779</td>
<td>.223</td>
</tr>
</tbody>
</table>
TABLE II: CHANGING PROBLEM SIZE
(Number of Iterations)

<table>
<thead>
<tr>
<th>Method</th>
<th>H = 1/16</th>
<th>1/32</th>
<th>1/64</th>
<th>1/128</th>
<th>1/256</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Natural Ordering)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JACOBI CG</td>
<td>49</td>
<td>94</td>
<td>178</td>
<td>330</td>
<td>629</td>
</tr>
<tr>
<td>JACOBI SI</td>
<td>56</td>
<td>179</td>
<td>362</td>
<td>772</td>
<td>1372</td>
</tr>
<tr>
<td>SOR</td>
<td>50</td>
<td>104</td>
<td>216</td>
<td>422</td>
<td>872</td>
</tr>
<tr>
<td>SSOR CG</td>
<td>16</td>
<td>22</td>
<td>34</td>
<td>51</td>
<td>73</td>
</tr>
<tr>
<td>SSOR SI</td>
<td>19</td>
<td>29</td>
<td>43</td>
<td>61</td>
<td>88</td>
</tr>
<tr>
<td>(Red-Black Ordering)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JACOBI CG</td>
<td>49</td>
<td>94</td>
<td>178</td>
<td>330</td>
<td>629</td>
</tr>
<tr>
<td>JACOBI SI</td>
<td>50</td>
<td>179</td>
<td>362</td>
<td>772</td>
<td>1372</td>
</tr>
<tr>
<td>SOR</td>
<td>52</td>
<td>101</td>
<td>196</td>
<td>396</td>
<td>839</td>
</tr>
<tr>
<td>SSOR CG</td>
<td>34</td>
<td>62</td>
<td>341</td>
<td>1056</td>
<td>3061</td>
</tr>
<tr>
<td>SSOR SI</td>
<td>51</td>
<td>107</td>
<td>196</td>
<td>373</td>
<td>752</td>
</tr>
<tr>
<td>RS CG</td>
<td>25</td>
<td>48</td>
<td>90</td>
<td>167</td>
<td>321</td>
</tr>
<tr>
<td>RS SI</td>
<td>42</td>
<td>88</td>
<td>182</td>
<td>375</td>
<td>704</td>
</tr>
</tbody>
</table>
### TABLE III: CHANGING PROBLEM SIZE

(Iteration Time in Seconds)

<table>
<thead>
<tr>
<th>Method</th>
<th>( H = \frac{1}{16} )</th>
<th>( \frac{1}{32} )</th>
<th>( \frac{1}{64} )</th>
<th>( \frac{1}{128} )</th>
<th>( \frac{1}{256} )</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>(Natural Ordering)</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JACOBI CG</td>
<td>.010</td>
<td>.040</td>
<td>.251</td>
<td>1.800</td>
<td>14.115</td>
</tr>
<tr>
<td>JACOBI SI</td>
<td>.014</td>
<td>.091</td>
<td>.560</td>
<td>4.196</td>
<td>28.741</td>
</tr>
<tr>
<td>SOR</td>
<td>.035</td>
<td>.292</td>
<td>2.446</td>
<td>19.828</td>
<td>164.940</td>
</tr>
<tr>
<td>SSOR CG</td>
<td>.027</td>
<td>.133</td>
<td>.828</td>
<td>4.953</td>
<td>28.157</td>
</tr>
<tr>
<td>SSOR SI</td>
<td>.029</td>
<td>.163</td>
<td>.967</td>
<td>5.583</td>
<td>32.249</td>
</tr>
<tr>
<td><strong>(Red-Black Ordering)</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JACOBI CG</td>
<td>.010</td>
<td>.041</td>
<td>.257</td>
<td>1.847</td>
<td>14.511</td>
</tr>
<tr>
<td>JACOBI SI</td>
<td>.013</td>
<td>.091</td>
<td>.571</td>
<td>4.277</td>
<td>29.394</td>
</tr>
<tr>
<td>SOR</td>
<td>.011</td>
<td>.066</td>
<td>.475</td>
<td>4.028</td>
<td>34.939</td>
</tr>
<tr>
<td>SSOR CG</td>
<td>.018</td>
<td>.075</td>
<td>2.105</td>
<td>25.779</td>
<td>302.712</td>
</tr>
<tr>
<td>SSOR SI</td>
<td>.021</td>
<td>.113</td>
<td>.663</td>
<td>4.452</td>
<td>36.053</td>
</tr>
<tr>
<td>RS CG</td>
<td>.006</td>
<td>.019</td>
<td>.109</td>
<td>.757</td>
<td>5.981</td>
</tr>
<tr>
<td>RS SI</td>
<td>.008</td>
<td>.033</td>
<td>.207</td>
<td>1.557</td>
<td>11.881</td>
</tr>
</tbody>
</table>
COMMENTS ON TABLE I

- Two versions of Scalar ITPACK were compared with the CYBER 205 version of Vector ITPACK
- Mesh size $H = 1/64$ used for all runs
- Scalar ITPACK: unrolled DO-loops used in basic vector operations for increased optimization on scalar computers
- Modified Scalar ITPACK: standard tight DO-loops used
- Vector Fortran compiler recognizes tight loops as vectorizable but not unrolled loops
- A slight increase in speed from Scalar to Modified Scalar version
- Vector ITPACK uses tight loops, Fortran vector syntax, and a column-oriented sparse storage scheme
- This data structure allows the matrix-vector product operation to vectorize to a great extent

* Considerable improvement in performance from scalar to vector version of ITPACK *
COMMENTS ON TABLE II & III

- These tables are results of using Vector ITPACK on the same problem with varying mesh sizes.

- The number of iterations increase as the problem size increase.

- Comparisons based on number of iterations misleading as to the best method!

- On scalar computers, SOR with natural ordering is widely used while JACOBI is not but on vector computers ...

- Most efficient method on the CYBER 205:
  
  JACOBI CG method when natural ordering is used
  
  RS CG when red-black ordering is used
SCALAR ITPACK vs. VECTOR ITPACK

- Total time for each method is not significantly greater than the iteration time in the vector version (this was not the case in the scalar version)

- Only $N$ additional workspace locations required for the vector version over the scalar version

- Faster scaling and permuting of the system with the column-oriented sparse storage scheme

- Improved performance of the SSOR methods with the red-black ordering in the vector version in spite of the greater number of iterations
A PRE-CONDITIONED CONJUGATE GRADIENT PACKAGE

Thomas C. Oppe, a graduate student at UT Austin, is working on a package which allows flexibility in the choice of basic methods and acceleration schemes.

The package has been designed to make the addition of further preconditionings and acceleration schemes easy.

Particular attention has been paid to the choice of matrix storage schemes with a view to maximizing vectorizability.

Features of Package:

- Conjugate Gradient Acceleration

- Pre-conditioning matrix Q (Jacobi, Symmetric Successive Overrelaxation, Reduced System, Incomplete Cholesky, Modified Incomplete Cholesky, Neumann Polynomial, Parameterized Polynomials, Other preconditionings planned such as Incomplete Block Cyclic Reduction)

- Realistic Stopping Tests

- Automatic estimation of iteration parameters with adaptive procedures

- Two possible data structures allowed
DATA STRUCTURES

Data structures which allow vectorization to varying degree:

EXAMPLE:

\[
A = \begin{bmatrix}
4 & -1 & -2 & 0 \\
-1 & 4 & 0 & -2 \\
-2 & 0 & 4 & -1 \\
0 & -2 & -1 & 4 \\
\end{bmatrix}
\]

ELLPACK Data Structure:

\[
\text{COEF} = \begin{bmatrix}
4 & -1 & -2 \\
4 & -2 & -1 \\
4 & 1 & -2 \\
4 & -2 & -1 \\
\end{bmatrix}
\]

\[
\text{JCOEF} = \begin{bmatrix}
1 & 2 & 3 \\
2 & 4 & 1 \\
3 & 4 & 1 \\
4 & 2 & 3 \\
\end{bmatrix}
\]

- matrix-vector product vectorizes with the use of gathering routines

- operations such as forward (back) substitutions using lower (upper) triangular matrices do not vectorize

DIAGONAL Data Structure:

\[
\text{COEF} = \begin{bmatrix}
4 & -1 & -2 \\
4 & 0 & -2 \\
4 & -1 & * \\
4 & * & * \\
\end{bmatrix}
\]

\[
\text{JCOEF} = (0, 1, 2)
\]

- the matrix-vector product operation vectorizes without the use of gathering routines

- operations such as forward (back) substitution and factorizations vectorize to some extent
REFERENCES


David R. Kincaid, Tom Oppe, and David M. Young, "Adapting ITPACK Routines for Use on Vector Computers," Report CNA-177, Center for Numerical Analysis, University of Texas at Austin, TX, August 1982. (In the Proceedings of the 1982 Symposium on CYBER 205 Applications, Institute for Computational Studies at Colorado State University, Fort Collins, CO.)
