Hybrid QR Factorization Algorithm for High Performance Computing Architectures

Peter Vouras
Naval Research Laboratory Radar Division

Professor G.G.L. Meyer
Johns Hopkins University Parallel Computing and Imaging Laboratory
Outline

- Background
- Problem Statement
- Givens Task
- Householder Task
- Paths Through Dependency Graph
- Parameterized Algorithms
- Parameters Used
- Results
- Conclusion
Background

- In many least squares problems, QR decomposition is employed
  - Factor matrix $A$ into unitary matrix $Q$ and upper triangular matrix $R$ such that $A = QR$

- Two primary algorithms available to compute QR decomposition
  - Givens rotations
    - Pre-multiplying rows $i-1$ and $i$ of a matrix $A$ by a 2x2 Givens rotation matrix will zero the entry $A(i,j)$
    
    $$
    \begin{bmatrix}
    * & * & * & * \\
    0 & * & * & *
    \end{bmatrix}
    =
    \begin{bmatrix}
    c & -s \\
    s & c
    \end{bmatrix}
    \begin{bmatrix}
    * & * & * & * \\
    0 & 0 & 0 & *
    \end{bmatrix}
    $$

  - Householder reflections
    - When a column of $A$ is multiplied by an appropriate Householder reflection, it is possible to zero all the subdiagonal entries in that column
    
    $$
    \begin{bmatrix}
    * & * & * & * \\
    0 & * & * & *
    \end{bmatrix}
    =
    \left( I - \frac{2}{v^T v} vv^T \right)
    \begin{bmatrix}
    * & * & * & * \\
    0 & 0 & 0 & *
    \end{bmatrix}
    $$
Problem Statement

- Want to minimize the latency incurred when computing the QR decomposition of a matrix $A$ and maintain performance across different platforms.

- Algorithm consists of parallel Givens task and serial Householder task.

- Parallel Givens task
  - Allocate blocks of rows to different processors. Each processor uses Givens rotations to zero all available entries within block such that
    - $A(i, j) = 0$ only if $A(i-1, j-1) = 0$ and $A(i, j-1) = 0$.

- Serial Householder task
  - Once Givens task terminates, all distributed rows are sent to root processor which utilizes Householder reflections to zero remaining entries.
Each processor uses Givens rotations to zero entries up to the topmost row in the assigned group.

Once task is complete, rows are returned to the root processor.

Givens rotations are accumulated in a separate matrix before updating all of the columns in the array.

- Avoids updating columns that will not be used by an immediately following Givens rotation.
- Saves a significant fraction of computational flops.
Householder Task

- Root processor utilizes Householder reflections to zero remaining entries in Givens columns.
- By computing a-priori where zeroes will be after each Givens task is complete, root processor can perform a sparse matrix multiply when performing a Householder update for additional speed-up.
  - Householder update is $A = A - \beta vv^T$.
- Householder update involves matrix-vector multiplication and an outer product update.
  - Makes extensive use of BLAS routines.

```
  * * * *
Processor 0
  0 * * *
  0 0 * *
  0 0 0 *
  0 0 0 *
  0 0 0 *
  0 0 0 *
  0 0 0 *
  0 0 0 *
  0 0 0 *
```

```plaintext
<table>
<thead>
<tr>
<th>Processor 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 * * *</td>
</tr>
<tr>
<td>0 0 * *</td>
</tr>
<tr>
<td>0 0 0 *</td>
</tr>
<tr>
<td>0 0 0 *</td>
</tr>
<tr>
<td>0 0 0 *</td>
</tr>
<tr>
<td>0 0 0 *</td>
</tr>
<tr>
<td>0 0 0 *</td>
</tr>
<tr>
<td>0 0 0 *</td>
</tr>
<tr>
<td>0 0 0 *</td>
</tr>
<tr>
<td>0 0 0 *</td>
</tr>
</tbody>
</table>
```
**Dependency Graph - Path 1**

<p>| | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>9</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>8</td>
<td>17</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>7</td>
<td>16</td>
<td>24</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>6</td>
<td>15</td>
<td>23</td>
<td>30</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>5</td>
<td>14</td>
<td>22</td>
<td>29</td>
<td>35</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>4</td>
<td>13</td>
<td>21</td>
<td>28</td>
<td>34</td>
<td>39</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>3</td>
<td>12</td>
<td>20</td>
<td>27</td>
<td>33</td>
<td>38</td>
<td>42</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>2</td>
<td>11</td>
<td>19</td>
<td>26</td>
<td>32</td>
<td>37</td>
<td>41</td>
<td>44</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
<td>18</td>
<td>25</td>
<td>31</td>
<td>36</td>
<td>40</td>
<td>43</td>
<td>45</td>
<td>*</td>
<td>*</td>
</tr>
</tbody>
</table>

- Algorithm must zero matrix entries in such an order that previously zeroed entries are not filled-in.
- Implies that $A(i, j)$ can be zeroed only if $A(i-1, j-1)$ and $A(i, j-1)$ are already zero.
- More than one sequence exists to zero entries such that above constraint is satisfied.
- Choice of path through dependency graph greatly affects performance.
By traversing dependency graph in zig-zag fashion, cache line reuse is maximized

- Data from row already in cache is used to zero several matrix entries before row is expunged from cache
Parameterized Algorithms make effective use of memory hierarchy

- Improve spatial locality of memory references by grouping together data used at the same time
- Improve temporal locality of memory references by using data retrieved from cache as many times as possible before cache is flushed

Portable performance is primary objective
Parameter $c$ controls the number of columns in Givens task.

Determines how many matrix entries can be zeroed before rows are flushed from cache.
Parameter $h$ controls the number of columns zeroed by Householder reflections at the root processor.

- If $h$ is large, the root processor performs more serial work, avoiding the communication costs associated with the Givens task.

- However, the other processors sit idle longer, decreasing the efficiency of the algorithm.
Work Partition Parameters

Parameters $v$ and $w$ allow operator to assign rows to processors such that the work load is balanced and processor idle time is minimized.
Results
Server Computer (1)

- 48 550-MHz PA-RISC 8600 CPUs
- 1.5 MB on-chip cache per CPU
- 1 GB RAM / Processor

HP Superdome
SPAWAR in San Diego, CA
Server Computer (2)

- 512 R12000 processors running at 400 MHz
- 8 MB on-chip cache
- Up to 2 GB RAM / Processor

SGI O3000
NRL in Washington, D.C.
Embedded Computer

Mercury
JHU in Baltimore, MD

- 8 Motorola 7400 processors with AltiVec units
- 400 MHz clock
- 64 MB RAM per processor
Effect of $c$

100 x 100 array
4 processors
$p = 12, h = 0$
Effect of $h$

100 x 100 array
4 processors
$c = 63$, $p = 12$
Effect of $w$

100 x 100 array
4 processors
$h = 15$, $p = 10$, $c = 60$, $v = 15$
Performance vs Matrix Size

- Mercury
- SGI O3000
- HP Superdome
Scalability

Time - msec

Number of processors

- Mercury
- SGI O3000
- HP Superdome

500 x 500 array
For matrix sizes on the order of 100 by 100, the Hybrid QR algorithm outperforms the SCALAPACK library routine PSGEQRF by 16%. Data distributed in block cyclic fashion before executing PSGEQRF.
Hybrid QR algorithm using combination of Givens rotations and Householder reflections is efficient way to compute QR decomposition for small arrays on the order of 100 x 100.

Algorithm implemented on SGI O3000 and HP Superdome servers as well as Mercury G4 embedded computer.

Mercury implementation lacked optimized BLAS routines and as a consequence performance was significantly slower.

Algorithm has applications to signal processing problems such as adaptive nulling where strict latency targets must be satisfied.