NRL

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

NRL

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)*

$$\begin{bmatrix} * & * & * & * \\ 0 & * & * & * \end{bmatrix} = \begin{bmatrix} c & -s \\ s & c \end{bmatrix} \begin{bmatrix} * & * & * & * \\ * & * & * & * \end{bmatrix}$$

  - 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

$$\begin{bmatrix} * & * & * & * \\ 0 & * & * & * \end{bmatrix} = \left(I - \frac{2}{v^T v} v v^T \right) \begin{bmatrix} * & * & * & * \\ * & * & * & * \end{bmatrix}$$

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

NRL

Givens Task

$$
\begin{array}{c}
\left[\begin{array}{cccccccccc}
* & * & * & * & * & * & * & * & * & * \\
0 & * & * & * & * & * & * & * & * & * \\
0 & 0 & * & * & * & * & * & * & * & * \\
0 & 0 & 0 & * & * & * & * & * & * & * \\
\hline
* & * & * & * & * & * & * & * & * & * \\
0 & * & * & * & * & * & * & * & * & * \\
0 & 0 & * & * & * & * & * & * & * & * \\
\hline
* & * & * & * & * & * & * & * & * & * \\
0 & * & * & * & * & * & * & * & * & * \\
0 & 0 & * & * & * & * & * & * & * & *
\end{array}\right]
\end{array}
\begin{array}{l}
\text{Processor 0} \\ \\ \\ \\
\text{Processor 1} \\ \\ \\
\text{Processor 2} \\ \\ \\
\end{array}
$$

- 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 use by an immediately following Givens rotation
  - Saves significant fraction of computational flops

NRL

Householder Task

Processor 0

$$\left[\begin{array}{ccc|ccccccc}
* & * & * & * & * & * & * & * & * & * \\
0 & * & * & * & * & * & * & * & * & * \\
0 & 0 & * & * & * & * & * & * & * & * \\
0 & 0 & 0 & * & * & * & * & * & * & * \\
0 & 0 & 0 & * & * & * & * & * & * & * \\
0 & 0 & 0 & * & * & * & * & * & * & * \\
0 & 0 & 0 & * & * & * & * & * & * & * \\
0 & 0 & 0 & * & * & * & * & * & * & * \\
0 & 0 & 0 & * & * & * & * & * & * & * \\
0 & 0 & 0 & * & * & * & * & * & * & *
\end{array}\right]$$

- **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 $A = A - \beta vv^T A$**
- **Householder update involves matrix-vector multiplication and an outer product update**
  - **Makes extensive use of BLAS routines**

NRL

Dependency Graph - Path 1

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

- **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**

NRL

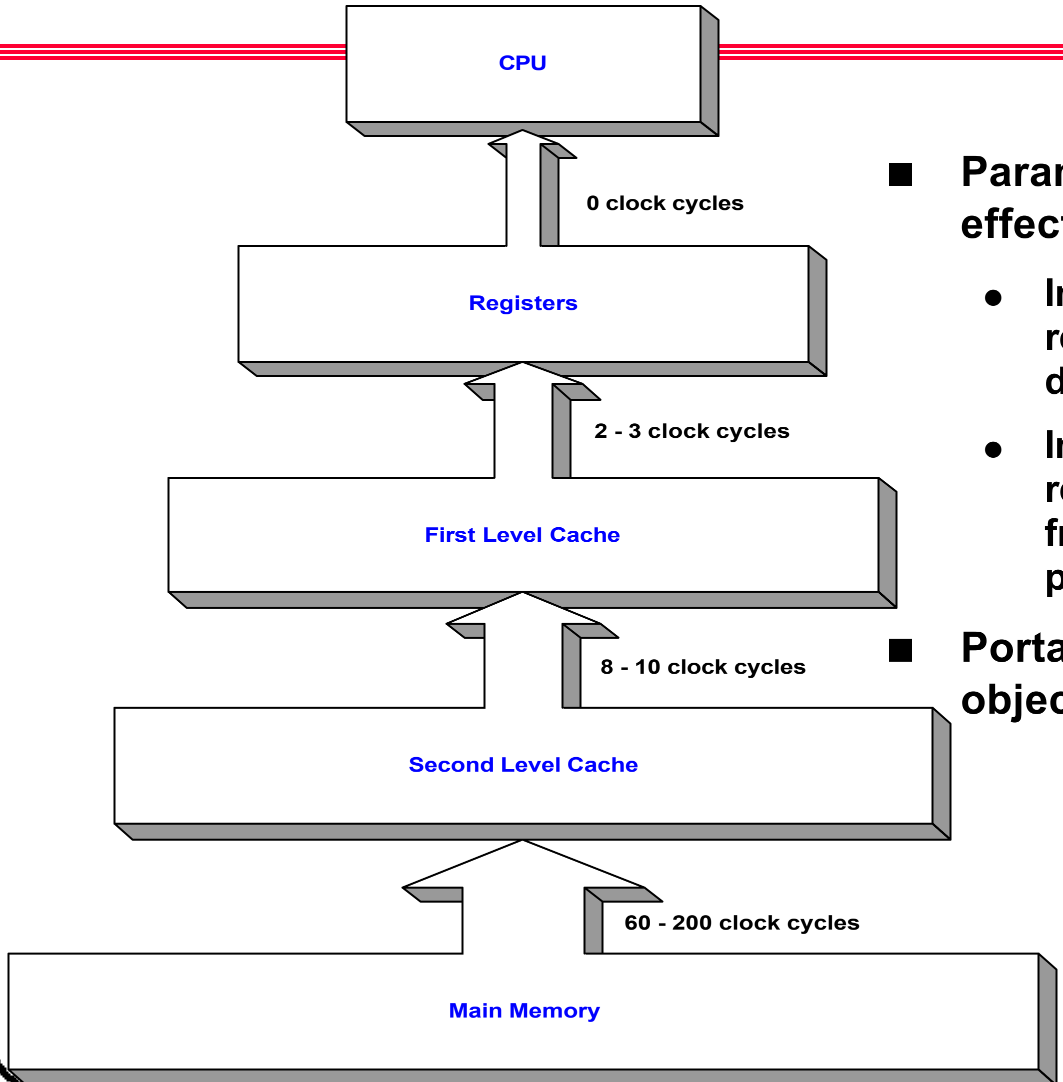
Dependency Graph - Path 2

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

- **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**

NRL

Parameterized Algorithms : Memory Hierarchy



Memory Hierarchy of SGI O2000

- 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

NRL

Givens Parameter

$$\left[\begin{array}{ccc|ccccccc}
* & * & * & * & * & * & * & * & * & * \\
0 & * & * & * & * & * & * & * & * & * \\
0 & 0 & * & * & * & * & * & * & * & * \\
0 & 0 & 0 & * & * & * & * & * & * & * \\
* & * & * & * & * & * & * & * & * & * \\
0 & * & * & * & * & * & * & * & * & * \\
0 & 0 & * & * & * & * & * & * & * & * \\
* & * & * & * & * & * & * & * & * & * \\
0 & * & * & * & * & * & * & * & * & * \\
0 & 0 & * & * & * & * & * & * & * & *
\end{array}\right]$$

c

- **Parameter *c* controls the number of columns in Givens task**
- **Determines how many matrix entries can be zeroed before rows are flushed from cache**

NRL

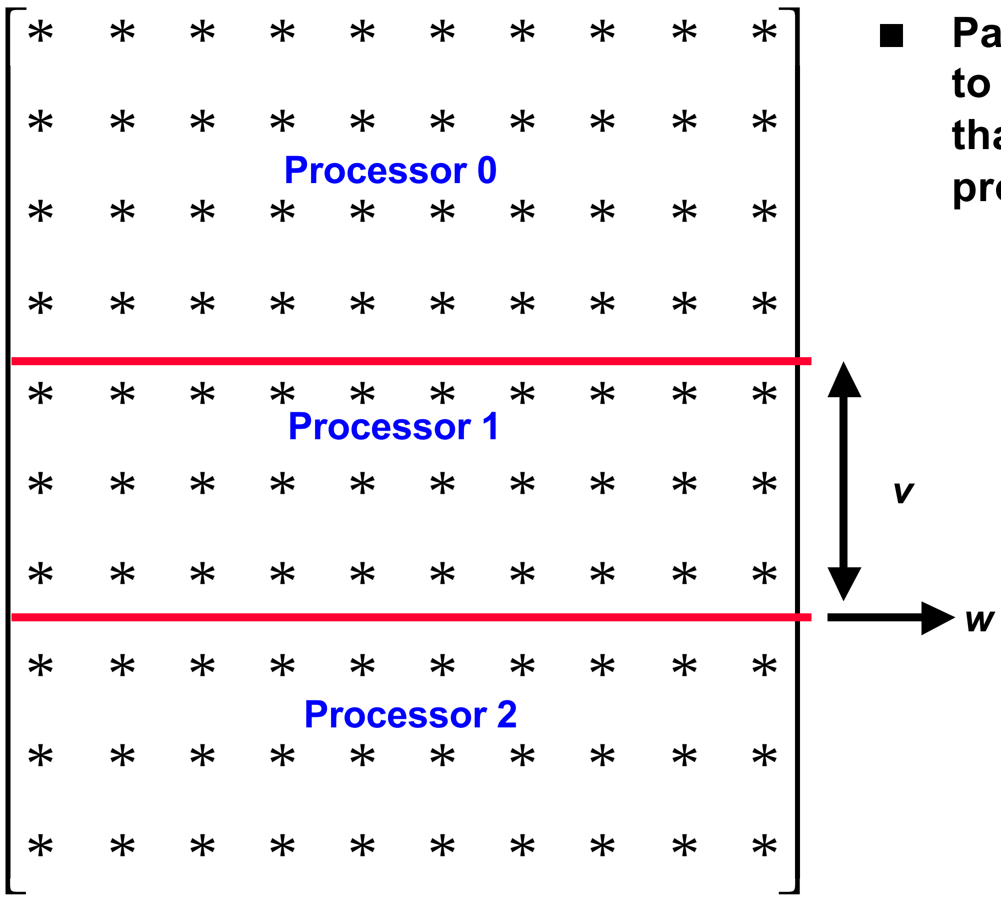
Householder Parameter

$$\begin{array}{c}\left[\begin{array}{ccc|cc|ccccc}
* & * & * & * & * & * & * & * & * & * \\
0 & * & * & * & * & * & * & * & * & * \\
0 & 0 & * & * & * & * & * & * & * & * \\
0 & 0 & 0 & * & * & * & * & * & * & * \\
0 & 0 & 0 & 0 & * & * & * & * & * & * \\
0 & 0 & 0 & 0 & 0 & * & * & * & * & * \\
0 & 0 & 0 & 0 & 0 & * & * & * & * & * \\
0 & 0 & 0 & 0 & 0 & * & * & * & * & * \\
0 & 0 & 0 & 0 & 0 & * & * & * & * & * \\
0 & 0 & 0 & 0 & 0 & * & * & * & * & *
\end{array}\right] \\
\begin{array}{ccc}
\xleftrightarrow{\quad c \quad} & \xleftrightarrow{\; h \;} & \qquad\qquad\qquad\qquad\qquad
\end{array}\end{array}$$

- 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

NRL

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.

NRL

Embedded Computer

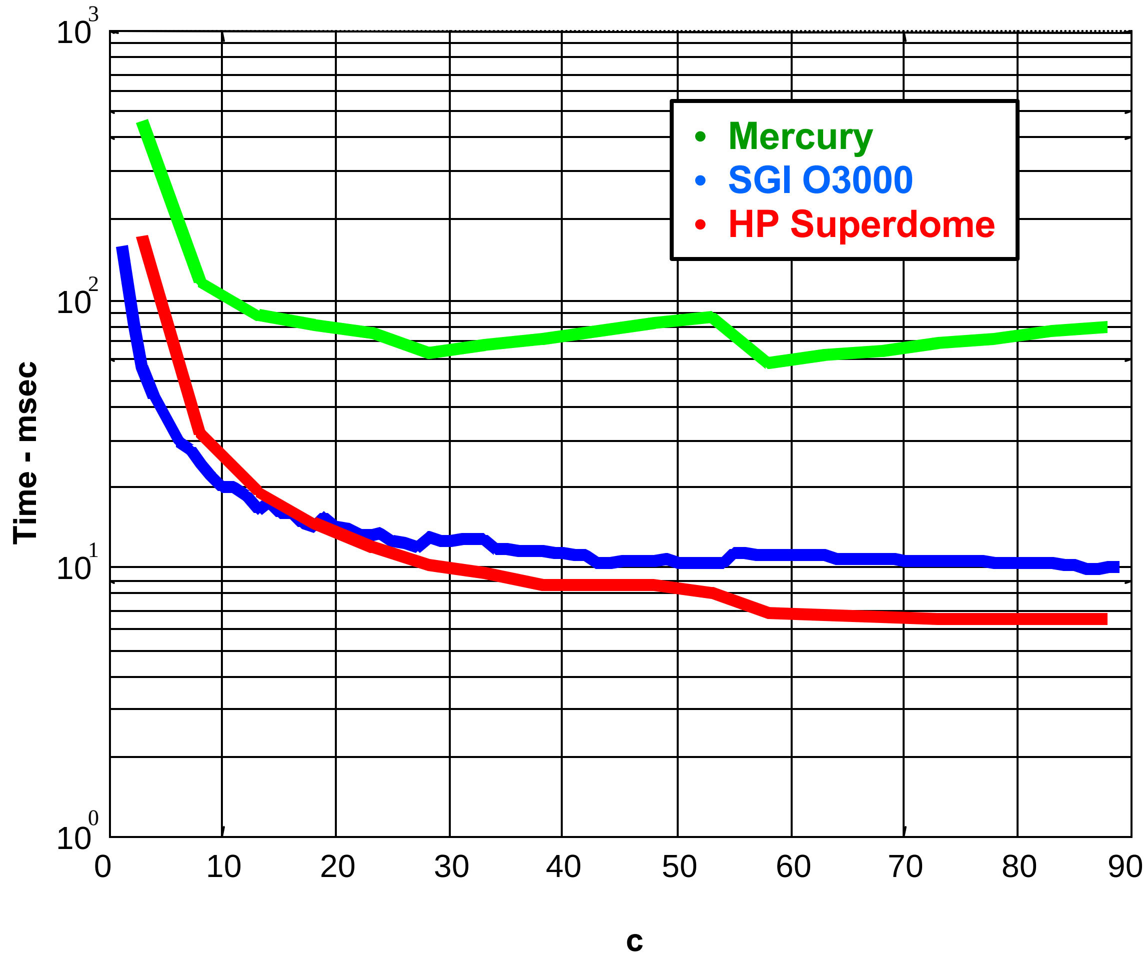
Mercury

JHU in Baltimore, MD

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

NRL

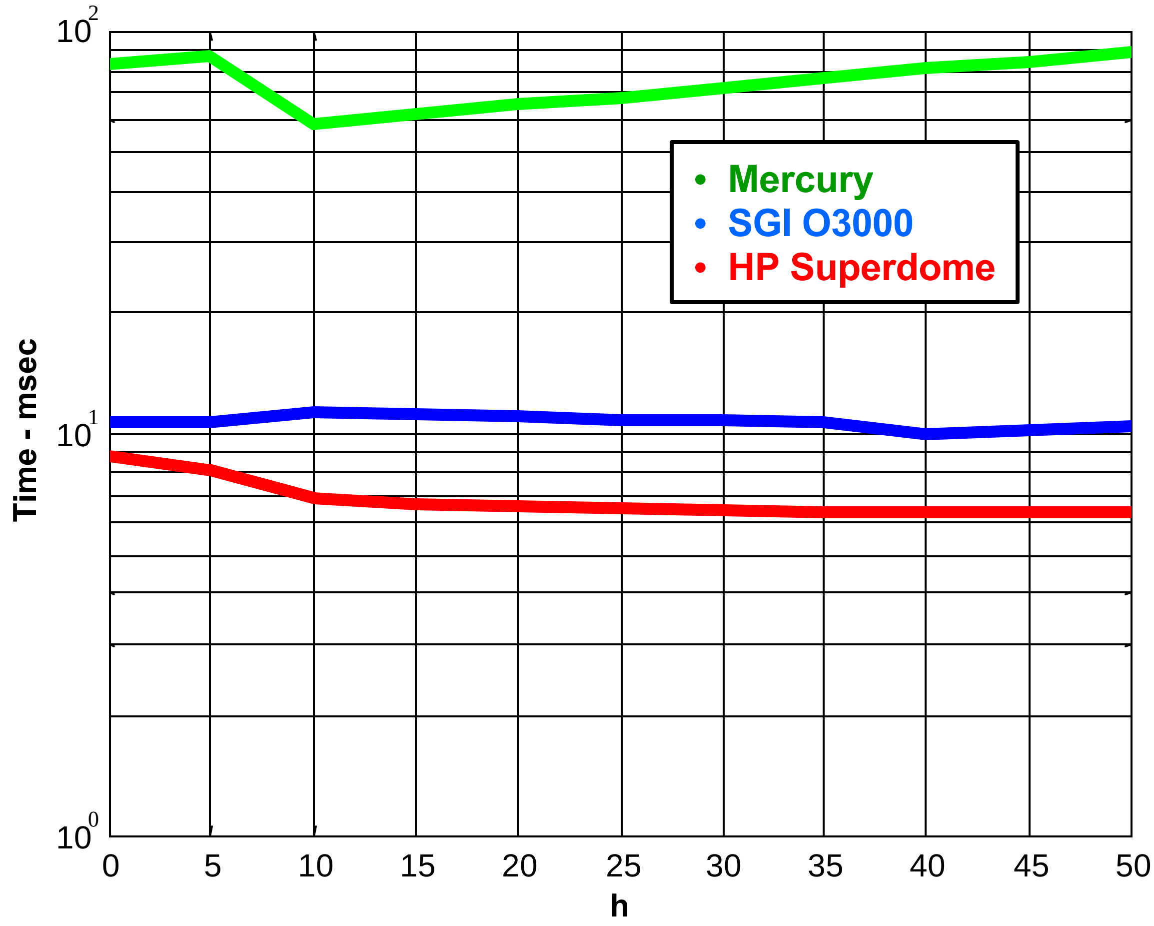
Effect of c



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

NRL

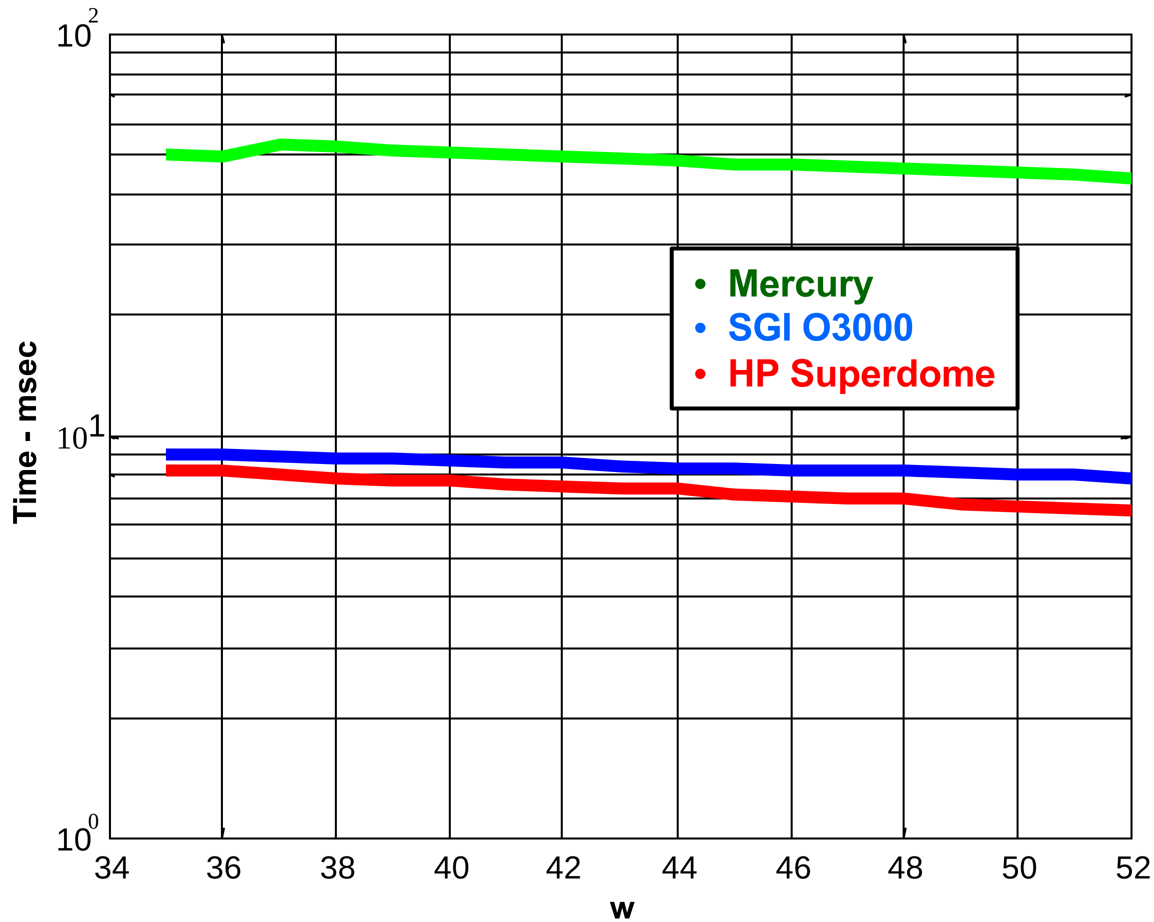
Effect of h



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

NRL

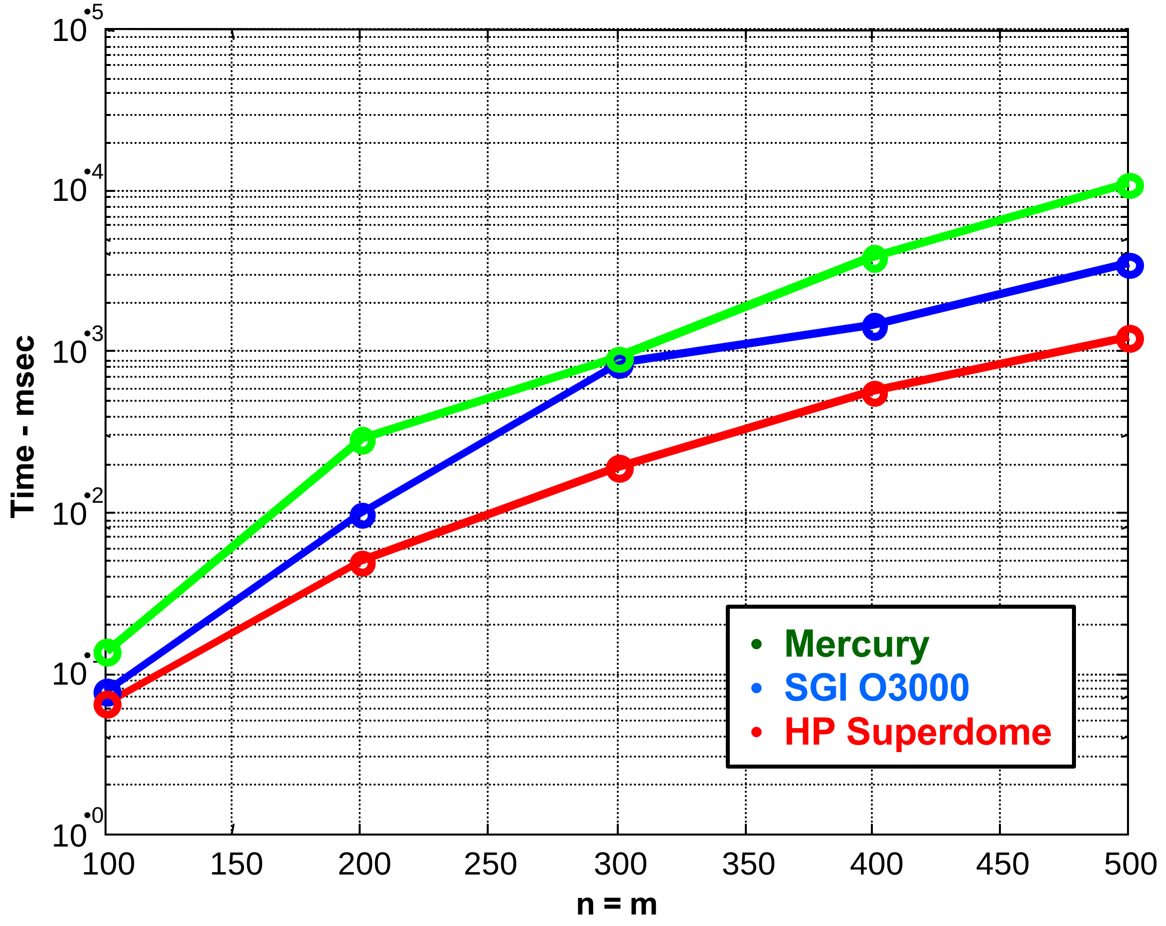
Effect of w



100 x 100 array
4 processors
$h = 15$, $p = 10$,
$c = 60$, $v = 15$

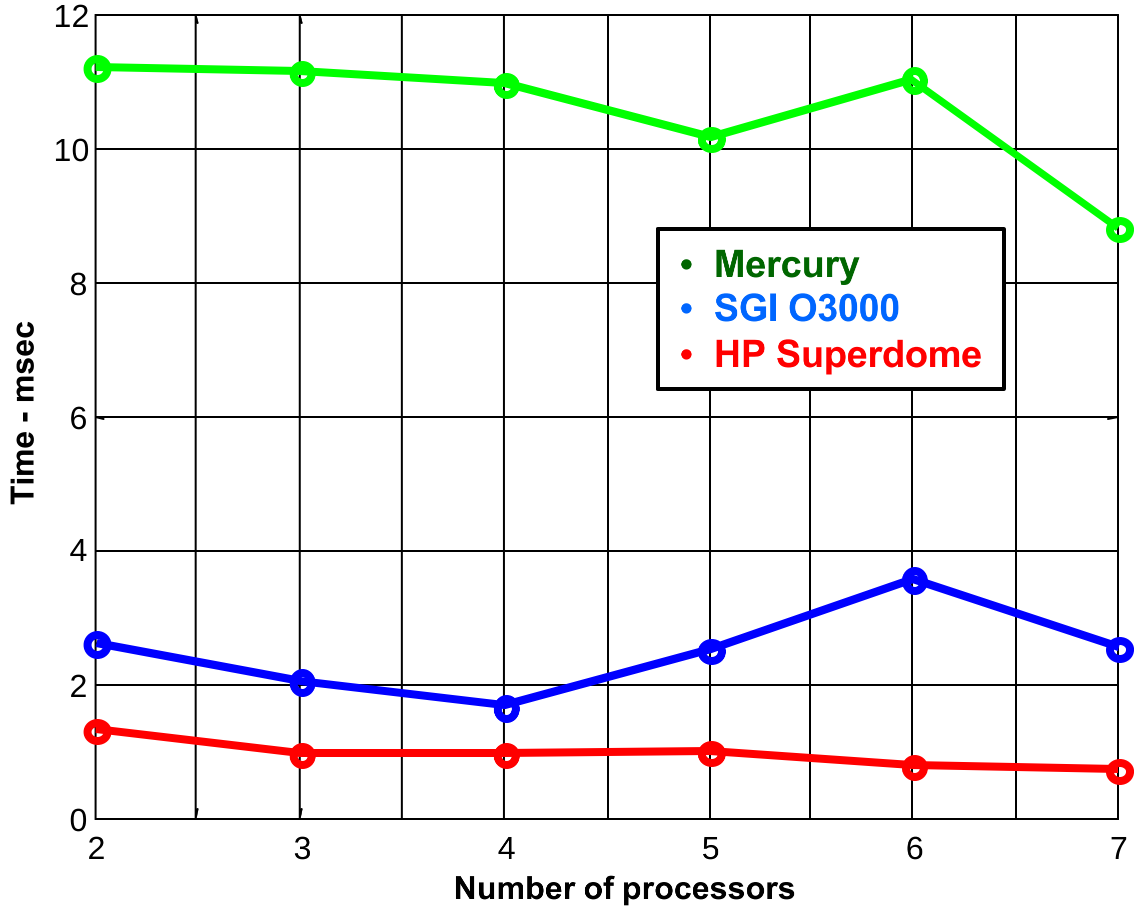
NRL

Performance vs Matrix Size



4 processors

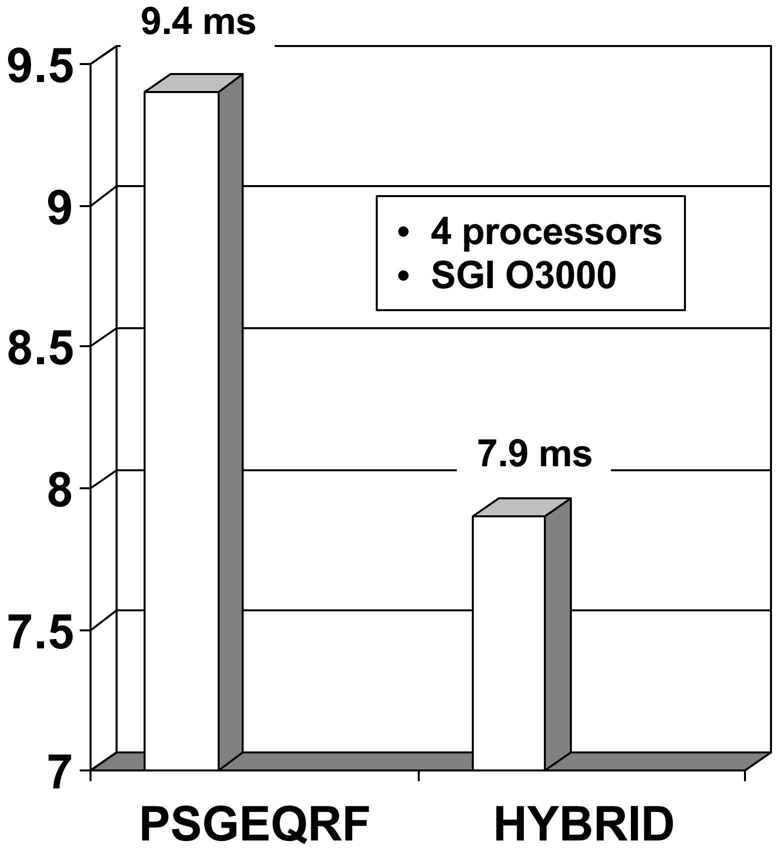
Scalability



500 x 500 array

NRL

Comparison to SCALAPACK



- 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

NRL

Conclusion

- 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