Hybrid QR Factorization Algorithm for High Performance Computing Architectures

Peter G. Vouras
Radar Division, Code 5321
Naval Research Laboratory
Washington, DC 20375

Gerard G. L. Meyer
Dept. of Electrical and Computer Engineering
Johns Hopkins University
Baltimore, MD 21218

I. Abstract

This paper describes a novel QR decomposition algorithm that utilizes both Givens rotations and Householder reflections to give optimal performance on high performance computing architectures. Computing the matrix factorization $A = QR$ is a mathematical step frequently encountered in many signal processing applications, including adaptive nulling. Efficient algorithms for computing the QR factorization are vital to satisfying strict latency and throughput constraints in real-time implementations of these signal processing algorithms. By effectively combining the use of Givens rotations and Householder reflections on a parallel computing platform, it is possible to efficiently map the computational algorithm to the available computational hardware to best make use of the memory hierarchy and to optimize performance. Furthermore, by introducing adjustable software parameters that control load partitioning among the processors, cache use, and the number of computations done in serial fashion on one processor versus the calculations done concurrently on several processors, it is possible for the user to tune the algorithm to run on different machines, without rewriting any code and still maintaining optimum performance. Results given in this abstract show that properly tuning the Hybrid QR algorithm on the SGI O3000 provided better performance than the available SCALAPACK library routine PSGEQRF for computing the QR decomposition.

There exist two techniques for computing the QR decomposition of a matrix. One technique is based on the use of Givens rotations of the form

$$\begin{bmatrix} c & -s \\ s & c \end{bmatrix}.$$

Pre-multiplying rows *i-1* and *i* of a matrix *A* by a 2x2 Givens rotation matrix will zero the entry *A(i,j)*. To ensure that the application of subsequent Givens rotations do not fill-in matrix entries previously zeroed, it is necessary that *A(i,j)* be zeroed only if *A(i-1,j-1)* and *A(i,j-1)* are already zeroed. All matrix entries that satisfy this condition may be zeroed concurrently on separate processors.

The order chosen in which to zero matrix entries can have a significant impact on the temporal locality of data in cache, and consequently greatly affect performance. For example, to zero the subdiagonal entries in a 4x4 array, one may proceed in the sequence $a_{41}, a_{31}, a_{42}, a_{21}, a_{32}, a_{43}$. By introducing the adjustable parameter *c* that controls how many columns are in the computational sequence, the Hybrid QR algorithm allows the user to maximize cache line reuse. Furthermore, by aggregating the Givens rotations applied during the computational sequence before updating all of the columns in the matrix, the Hybrid QR algorithm minimizes the floating point operations wasted updating columns that will not be used by an immediately following Givens rotation. Figure 1 illustrates the effect the parameter *c* has on performance for the case of a real 100x100 array, using 4 processors on the SGI O3000, HP Superdome and Mercury G4 computers. In this case, the parameter *c* varies from 1 to 87.

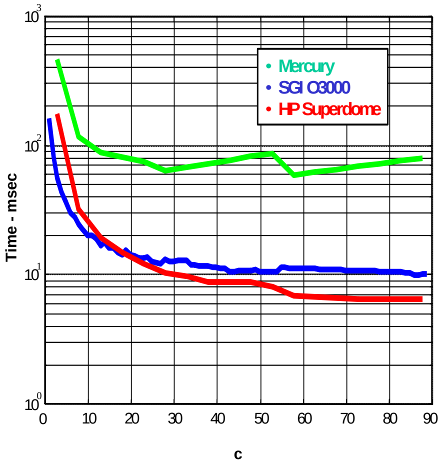

Figure 1.

In the Hybrid QR algorithm, once all the available concurrent Givens rotations in a group of *c* columns have been performed on independent processors, all the processors send their local rows of the array *A* to a root processor which then utilizes Householder reflections to zero all remaining subdiagonal entries in the group of *c* columns plus any additional entries in a group of *h* columns. Thus, the parameter *h* may be selected by the user to effectively increase or decrease the amount of serial work performed by the Hybrid QR algorithm. This parameter is a useful tool for amortizing the communication costs associated with passing data between the processors to perform the Givens rotations in parallel.

A Householder reflection is a matrix of the form

$$I - \frac{2}{v^T v} v v^T .$$

The vector *v* is called a Householder vector. When a column of the matrix *A* is multiplied by an appropriately chosen Householder reflection, it is possible to zero all the subdiagonal entries in that column. Therefore, Householder reflections are also frequently used to compute QR factorizations. Since a Householder update involves a matrix-vector multiplication and an outer product update, the Hybrid QR algorithm makes heavy use of optimized BLAS routines to perform the Householder computations. Once the Householder step is complete, the Hybrid QR algorithm proceeds by performing another set of Givens rotations in parallel on the next block of *c* columns. Figure 2 shows the effect on performance of varying the parameter *h* from 0 to 50, given that *c*=51, for the case of a 100x100 matrix on an SGI O3000, HP Superdome, and Mercury G4 using 4 processors.

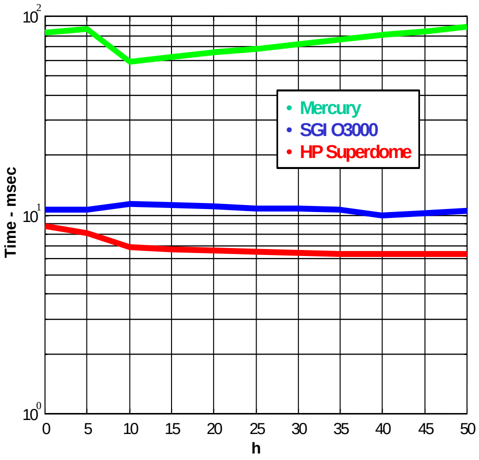

Figure 2.

To make the most efficient use of the available processors, the user must partition the computational load between the processors as evenly as possible so that one processor is not idle for long periods while the other processors are working. In the Hybrid QR algorithm, two parameters *v* and *w* allow the user to select how many rows of the matrix *A* are assigned to each processor to perform Givens rotations. Figure 3 shows the impact on performance of varying the parameter *w* from 35 to 52 while *v*=15. In this case, *c*=50 and *h*=15.

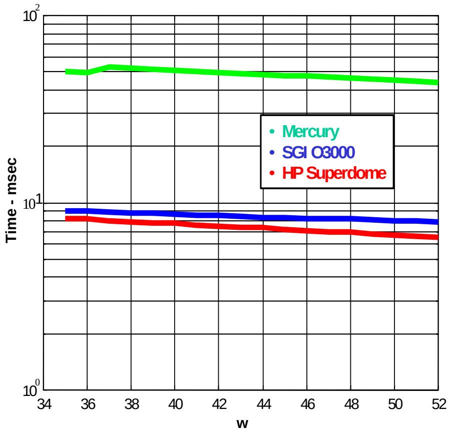

Figure 3.

By optimizing performance over the parameter set *c*, *h*, *w*, and *v*, it is possible to obtain the combination that yields the minimum execution time. Table 1 shows the execution time achieved using the Hybrid QR algorithm to factor a 100x100 real array on the SGI O3000 compared to the execution time measured using the SCALAPACK library routine PSGEQRF. The Hybrid QR algorithm demonstrated a 16% improvement in execution time over PSGEQRF.

Table 1.

Hybrid QR (4 processors)	PSGEQRF (4 processors)
7.9 msec attained with *c*=50, *h*=15, *w*=51, *v*=15	9.4 msec

II. References

1. Boleng, Jeff and Misra, Manavendra, "Load Balanced Parallel QR Decomposition", Technical Report, Department of Math and Computer Sciences, Colorado School of Mines.
2. Carrig, James J. and Meyer, Gerard G. L., "A Fast Givens QR Algorithm for Efficient Cache Utilization", Technical Report, Department of Electrical and Computer Engineering, Johns Hopkins University.
3. Golub, Gene H. and Van Loan, Charles F., Matrix Computations, Third Edition, The Johns Hopkins University Press, 1996.