Improving the Performance of Reduction to Hessenberg Form

GREGORIO QUINTANA-ORTÍ
Universidad Jaume I
and
ROBERT VAN DE GEIJN
The University of Texas at Austin

In this paper, a modification of the blocked algorithm for reduction to Hessenberg form is presented that improves performance by shifting more computation from less efficient matrix-vector operations to highly efficient matrix-matrix operations. Significant performance improvements are reported relative to the performance achieved by the current LAPACK implementation.

Categories and Subject Descriptors: G.1.3 [**Numerical Analysis**]: Numerical Linear Algebra; G.4 [**Mathematical Software**]: —*Efficiency*

General Terms: Algorithms;Performance

Additional Key Words and Phrases: linear algebra, eigenvalue problems, reduction to condensed form

1. INTRODUCTION

The reduction to Hessenberg form is an important and time consuming step in the computation of the Schur decomposition of a square nonsymmetric matrix. The Schur decomposition itself is important since it solves the nonsymmetric eigenvalue problem [Golub and Van Loan 1996] and is a step towards the solution of the Sylvester equation and other linear algebra equations that arise in control theory [Sima 1996; Bartels and Stewart 1972; Golub et al. 1979; Golub and Van Loan 1996]. Thus, improving the performance of this computation impacts a number of important applications.

Algorithms for the computation of the reduction to Hessenberg form date back to the early 1960s [Wilkinson 1965; Martin and Wilkinson 1968]. In the 1980s, it was observed that the key to attaining high performance on architectures with complex multi-level memories is to cast computation in terms of matrix-matrix operations, like the matrix-matrix product. These operations are supported by the

Authors' addresses: Gregorio Quintana-Ortí, Departamento de Ingeniería y Ciencia de Computadores, Universidad Jaume I, Campus Riu Sec, 12.071 – Castellón, Spain, gquintan@icc.uji.es. Robert A. van de Geijn, Department of Computer Sciences, The University of Texas at Austin, Austin, TX 78712, rvdg@cs.utexas.edu.

level-3 Basic Linear Algebra Subprograms (BLAS) [Dongarra et al. 1990]. The idea is that such operations perform $O(n^3)$ floating point operations (flops) on $O(n^2)$ data, where n equals the matrix dimension of the operands. This allows data to be brought into fast cache memory, amortizing the cost of this data movement over a large number of computations. An algorithm that casts part of the computation required for the reduction in terms of matrix-matrix operations (level 3 BLAS) was first presented in [Dongarra et al. 1989]. The currently most widely used library for linear algebra operations, the Linear Algebra Package (LAPACK) [Anderson et al. 1992], incorporates one such algorithm in the routine DGEHRD.

As we will review later in this paper, inherently a considerable part of the computation must be in terms of matrix-vector operations (level-2 BLAS [Dongarra et al. 1988]) that attain only a modest percent of the peak of an architecture due to memory bandwidth limitations. Thus, even if the rest of the computation is cast in terms of level-3 BLAS, the part that requires level-2 BLAS will remain and will limit the percentage of peak that the algorithm can achieve. The problem with the implementation that is currently part of LAPACK is that it leaves more computation in terms of level-2 BLAS than necessary. It is this shortcoming that the algorithm in this paper addresses.

The primary benefit of our new implementation is a shift of computation from level-2 to level-3 BLAS. A secondary benefit comes from the fact that the computation that is not in level-3 BLAS involves less data, improving data locality and reducing cache traffic during those operations: The current LAPACK implementation touches in every iteration (that is, n times) all columns to the right of the current column while our algorithm only touches the part of those same columns that is below the current row.

The remainder of this paper is organized as follows: Householder transformations and related transforms are reviewed in Section 2. The traditional (unblocked) algorithm for reduction to Hessenberg form is given in Section 3. Blocked algorithms are then described in Section 4. Performance results are given in Section 5 followed by concluding remarks in the final section.

2. HOUSEHOLDER TRANSFORMS

Given a nonzero vector $u \in \mathbb{R}^m$, a *Householder transformation* (or *reflector*) is defined by $H = I_m - uu^T/\tau$, where I_m denotes the (square) identity matrix of order m and $\tau = u^T u/2$ [Householder 1958]. It is an orthogonal matrix ($H^T H = HH^T = I_m$) and symmetric ($H^T = H$). This transformation has wide application in the solution of linear least-squares problems, the computation of orthonormal bases, and the solution of the algebraic eigenvalue problem.

Householder transformations will be used in this paper in an effort to annihilate elements below the first subdiagonal of a given matrix. More precisely, given vector x, partition $x = \left(\chi_1 \,\middle|\, x_2 \right)^T$, where χ_1 equals the first element of x. Define the Householder vector associated with x as the vector $u = \left(1 \,\middle|\, u_2 \right)^T$, where $u_2 = x_2/\rho_1$ with $\rho_1 = \chi_1 + \mathrm{sign}(\chi_1)\|x\|_2$. Here $\mathrm{sign}(\alpha)$ returns 1 or -1 depending on the sign of α. Let $\tau = u^T u/2$. Then $(I - uu^T/\tau)x = \{0 \backslash\!\backslash \beta\}$ where $\beta = -\mathrm{sign}(\chi_1)\|x\|_2$ and $\{0 \backslash\!\backslash \beta\}$ equals the zero vector with the first element replaced by β. Let us introduce the notation $[\{u \backslash\!\backslash \beta\}, \tau] := \mathrm{Hous}(x)$ for the operation that computes the above mentioned β, u, and τ from the given vector x. Here $\{u\backslash\!\backslash\beta\}$ indicates the vector u with the first element (which is implicitly equal to "1") overwritten by the value β.

Multiplying two or more Householder transformations results again in an orthogonal matrix, although not symmetric. This allows Householder transformations to be accumulated into a composed transformation. Traditionally, the *compact WY transform* (CWY) is used, which has the form $I - USU^T$, where the columns of U consist of the Householder vectors being accumulated: Letting the columns of U_{k-1} equal the first k Householder vectors u_j, $0 \le j < k$,

$$(I - u_0 u_0^T/\tau_0)\cdots(I - u_{k-1}u_{k-1}^T/\tau_{k-1}) = (I - U_{k-1}S_{k-1}U_{k-1}^T),$$

where $S_0 = 1/\tau_0$ and $S_k = \left(\begin{array}{c|c} S_{k-1} & -S_{k-1}U_{k-1}u_k/\tau_k \\ \hline 0 & 1/\tau_k \end{array}\right)$.

A little-known alternative [Joffrain et al. ; Puglisi 1992; Walker 1988], which we call the UT transform, has the form $I - UT^{-1}U^T$ where

$$(I - u_0 u_0^T/\tau_0)\cdots(I - u_{k-1}u_{k-1}^T/\tau_{k-1}) = (I - U_{k-1}T_{k-1}^{-1}U_{k-1}^T),$$

with $T_0 = \tau_0$ and $T_k = \left(\begin{array}{c|c} T_{k-1} & U_{k-1}u_k \\ \hline 0 & \tau_k \end{array}\right)$. Note that T can be computed from U as $T = U^TU$ (upper triangular part only) followed by the dividing of the diagonal elements of the resulting T by two. Upper triangular matrices T and S are related by $S = T^{-1}$.

Both the CWY and the UT transform allow the application of a series of Householder transformations to be cast in terms of high-performance matrix-matrix operations:

$$(I - USU^T)A = \underbrace{A - U[\,\underbrace{S[\,\underbrace{U^TA}_{\text{GEMM}}\,]}_{\text{TRMM}}\,]}_{\text{GEMM}} \quad\text{and}\quad (I - UT^{-1}U^T)A = \underbrace{A - U[\,\underbrace{T^{-1}[\,\underbrace{U^TA}_{\text{GEMM}}\,]}_{\text{TRSM}}\,]}_{\text{GEMM}}.$$

3. NEW NOTATION FOR THE TRADITIONAL REDUCTION ALGORITHM

In this section, we discuss the basic idea behind the algorithm for computing the Hessenberg reduction of a square matrix A: $A = QBQ^T$, where Q is unitary and B is upperHessenberg (B has zeroes below the first subdiagonal). We will see that Q will be computed as a sequence of Householder transformations, that B can overwrite A, and that the Householder vectors can overwrite the parts of A below the first subdiagonal.

Let us denote the original contents of matrix A as $\hat{A}$ and assume that A is to be overwritten by the upperHessenberg matrix. Partition

$$A \rightarrow \left(\begin{array}{c|c} \alpha_{11} & a_{12}^T \\ \hline a_{21} & A_{22} \end{array}\right) \quad\text{and}\quad \hat{A} \rightarrow \left(\begin{array}{c|c} \hat{\alpha}_{11} & \hat{a}_{12}^T \\ \hline \hat{a}_{21} & \hat{A}_{22} \end{array}\right).$$

Let $[\{u\backslash\!\backslash\beta\}_{21}, \tau_1] = \text{Hous}(a_{21})$ and $H_0 = H(u_{21}, \tau_1) \equiv I - u_{21}u_{21}^T/\tau_1$ so that

$\{0\backslash\!\backslash\beta\}_{21} = H_0 a_{21}$. Then the first step of the reduction updates

$$A := \left(\begin{array}{c|c} \alpha_{11} & a_{12}^T \\ \hline a_{21} & A_{22} \end{array}\right) := \left(\begin{array}{c|c} 1 & 0 \\ \hline 0 & H_0 \end{array}\right) \left(\begin{array}{c|c} \alpha_{11} & a_{12}^T \\ \hline a_{21} & A_{22} \end{array}\right) \left(\begin{array}{c|c} 1 & 0 \\ \hline 0 & H_0 \end{array}\right) = \left(\begin{array}{c|c} \alpha_{11} & a_{12}^T H_0 \\ \hline \{0\backslash\!\backslash\beta\}_{21} & H_0 A_{22} H_0 \end{array}\right).$$

To complete an upperHessenberg reduction, the procedure continues by recursively computing Householder transformations from the updated A_{22}, and applying them to that matrix as well as the part of matrix A that appears above A_{22}.

The above procedure can be described more completely as follows. After k steps, partition A and $\hat{A}$ as

$$A \rightarrow \left(\begin{array}{c|c} B_{TL} & A_{TR} \\ \hline \{0\backslash\!\backslash\beta\}_{BL} & A_{BR} \end{array}\right) \quad \text{and} \quad \hat{A} \rightarrow \left(\begin{array}{c|c} \hat{A}_{TL} & \hat{A}_{TR} \\ \hline \hat{A}_{BL} & \hat{A}_{BR} \end{array}\right),$$

where B_{TL} and $\hat{A}_{TL}$ are $k \times k$, B_{TL} is upperHessenberg, and $\{0\backslash\!\backslash\beta\}_{BL}$ is the matrix of all zeroes except with β_{BL} in the top-right corner. Notice that by now A has been computed from $\hat{A}$ by computing $\{\check{H}_0, \ldots, \check{H}_{k-1}\}$ so that

$$A = \check{H}_{k-1} \cdots \check{H}_0 \hat{A} \check{H}_0 \cdots \check{H}_{k-1}, \quad \text{where } \check{H}_j = \left(\begin{array}{c|c} I_{j+1} & 0 \\ \hline 0 & H_j \end{array}\right).$$

Let us examine how A must be updated as part of the kth step. Repartition

$$\left(\begin{array}{c|c} B_{TL} & A_{TR} \\ \hline \{0\backslash\!\backslash\beta\}_{BL} & A_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} B_{00} & a_{01} & A_{02} \\ \hline \{0\backslash\!\backslash\beta\}_{10}^T & \alpha_{11} & a_{12}^T \\ \hline 0 & a_{21} & A_{22} \end{array}\right),$$

where $\{0\backslash\!\backslash\beta\}_{10}^T$ equals the row vector of all zeroes except for the last element, which equals β_{10}. H_k is now computed as $H_k = H(u_{21}, \tau_1)$ where $[\{u\backslash\!\backslash\beta\}_{21}, \tau_1] = \text{Hous}(a_{21})$ and A is updated with

$$\left(\begin{array}{c|c|c} I_k & 0 & 0 \\ \hline 0 & 1 & 0 \\ \hline 0 & 0 & H_k \end{array}\right) \left(\begin{array}{c|c|c} B_{00} & a_{01} & A_{02} \\ \hline \{0\backslash\!\backslash\beta\}_{10}^T & \alpha_{11} & a_{12}^T \\ \hline 0 & a_{21} & A_{22} \end{array}\right) \left(\begin{array}{c|c|c} I_k & 0 & 0 \\ \hline 0 & 1 & 0 \\ \hline 0 & 0 & H_k \end{array}\right) =$$

$$\left(\begin{array}{c|c|c} B_{00} & a_{01} & A_{02} H_k \\ \hline \{0\backslash\!\backslash\beta\}_{10}^T & \alpha_{11} & a_{12}^T H_k \\ \hline 0 & \{0\backslash\!\backslash\beta\}_{21} & H_k A_{22} H_k \end{array}\right).$$

The thick lines that indicate progress through the matrix can then be moved to include the next diagonal element:

$$\left(\begin{array}{c|c} B_{TL} & A_{TR} \\ \hline \{0\backslash\!\backslash\beta\}_{BL} & A_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c|c} B_{00} & a_{01} & A_{02} H_k \\ \hline \{0\backslash\!\backslash\beta\}_{10}^T & \alpha_{11} & a_{12}^T H_k \\ \hline 0 & \{0\backslash\!\backslash\beta\}_{21} & H_k A_{22} H_k \end{array}\right).$$

We finish this section by noting that vector u_k can be stored in the elements of A from which it was computed since it by design has a first element that equals "1", which therefore needs not be stored. The scalars τ_k are typically stored in a vector. Thus, after k steps A contains

$$\left(\begin{array}{c|c} \{U\backslash\!\backslash B\}_{TL} & A_{TR} \\ \hline \{U\backslash\!\backslash\beta\}_{BL} & A_{BR} \end{array}\right),$$

Algorithm: $[A,t] := \text{HesRedUnb}(A)$

Partition $A \rightarrow \left(\begin{array}{c|c} \{U\backslash\!\!\backslash B\}_{TL} & A_{TR} \\ \hline \{U\backslash\!\!\backslash \beta\}_{BL} & A_{BR} \end{array}\right)$, $t \rightarrow \left(\begin{array}{c} t_T \\ \hline t_B \end{array}\right)$

  where $\{U\backslash\!\!\backslash B\}_{TL}$ is 0×0 and t_T has 0 elements

while $m(A_{TL}) < m(A)$ **do**

  Repartition

$$\left(\begin{array}{c|c} \{U\backslash\!\!\backslash B\}_{TL} & A_{TR} \\ \hline \{U\backslash\!\!\backslash \beta\}_{BL} & A_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} \{U\backslash\!\!\backslash B\}_{00} & a_{01} & A_{02} \\ \hline \{u\backslash\!\!\backslash \beta\}_{10}^T & \alpha_{11} & a_{12}^T \\ \hline U_{20} & a_{21} & A_{22} \end{array}\right) \text{ and } \left(\begin{array}{c} t_T \\ \hline t_B \end{array}\right) \rightarrow \left(\begin{array}{c} t_0 \\ \hline \tau_1 \\ \hline t_2 \end{array}\right)$$

  where α_{11} and τ_1 are scalars

$[\{u\backslash\!\!\backslash \beta\}_{21}, \tau_1] := \text{Hous}(a_{21})$ ($\{u\backslash\!\!\backslash \beta\}_{21}$ overwrites a_{21})

$$\left.\begin{array}{l} A_{02} := A_{02} H(u_{21}, \tau_1) \\ a_{12}^T := a_{12}^T H(u_{21}, \tau_1) \\ A_{22} := H(u_{21}, \tau_1) A_{22} H(u_{21}, \tau_1) \end{array}\right\} \text{ via the steps in (1)–(4).}$$

  Continue with

$$\left(\begin{array}{c|c} \{U\backslash\!\!\backslash B\}_{TL} & A_{TR} \\ \hline \{U\backslash\!\!\backslash \beta\}_{BL} & A_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c|c} \{U\backslash\!\!\backslash B\}_{00} & b_{01} & A_{02} \\ \hline \{u\backslash\!\!\backslash \beta\}_{10}^T & \beta_{11} & a_{12}^T \\ \hline U_{20} & \{u\backslash\!\!\backslash \beta\}_{21} & A_{22} \end{array}\right) \text{ and } \left(\begin{array}{c} t_T \\ \hline t_B \end{array}\right) \leftarrow \left(\begin{array}{c} t_0 \\ \hline \tau_1 \\ \hline t_2 \end{array}\right)$$

endwhile

Fig. 1. Unblocked reduction to upperHessenberg form.

which is meant to indicate that the upperHessenberg matrix B_{TL} is stored in the upperHessenberg part of $\{U\backslash\!\!\backslash B\}_{TL}$, the element β_{BL} is stored in the top-right element of $\{U\backslash\!\!\backslash \beta\}_{BL}$, and the kth column[1] of A stores u_k below the first subdiagonal of that column. The complete algorithm is now given in Fig. 1.

In Fig. 1, it is important to realize that $H(u_{21}, \tau_1)$ is never explicitly formed, and the following formulae:

$$\begin{array}{l} A_{02} := A_{02} H(u_{21}, \tau_1) = A_{01}(I - u_{21} u_{21}^T / \tau_1) \\ a_{12}^T := a_{12}^T H(u_{21}, \tau_1) = a_{12}^T (I - u_{21} u_{21}^T / \tau_1) \\ A_{22} := H(u_{21}, \tau_1) A_{22} H(u_{21}, \tau_1) = (I - u_{21} u_{21}^T / \tau_1) A_{22} (I - u_{21} u_{21}^T / \tau_1) \end{array}$$

can be implemented as

$$\left(\begin{array}{c} v_{01} \\ \hline \nu_{11} \\ \hline v_{21} \end{array}\right) := \left(\begin{array}{c} A_{02} \\ \hline a_{12}^T \\ \hline A_{22} \end{array}\right) u_{21} \tag{1}$$

$$\left(\begin{array}{c} A_{02} \\ \hline a_{12}^T \\ \hline A_{22} \end{array}\right) := \left(\begin{array}{c} A_{02} \\ \hline a_{12}^T \\ \hline A_{22} \end{array}\right) - \left(\begin{array}{c} v_{01} \\ \hline \nu_{11} \\ \hline v_{21} \end{array}\right) u_{21}^T / \tau_1 \tag{2}$$

$$w_{21}^T := u_{21}^T A_{22} \tag{3}$$

$$A_{22} := A_{22} - u_{21} w_{21}^T / \tau_1 \tag{4}$$

For the step where $\{U\backslash\!\!\backslash B\}_{TL}$ is $k \times k$, the cost of each computation in (1) and (2) is roughly $2n(n-k-1)$ flops while each cost in (3) and (4) is roughly $2(n-k-1)^2$

[1]The 0th column of A is the left-most column here, since we start indexing at 0.

flops. The total cost for reducing $A \in \mathbb{R}^{n\times n}$ is thus approximately

$$\sum_{k=0}^{n-1} \left(4n(n-k-1) + 4(n-k-1)^2\right) \text{ flops} \approx \frac{10}{3} n^3 \text{flops}.$$

4. BLOCKED ALGORITHMS

In Section 2 we noted that by accumulating multiple Householder transformations into a single transform higher performance can be achieved when these transformations are to be applied to a matrix. The complication is that A must be updated with part of the computations in (1)–(4) before the next Householder transform can be computed. We first show how to progress to the point where a number of Householder transformations have been accumulated, after which we show how to then cast the remainder of the computation mostly in terms of matrix-matrix products.

4.1 Building up a block

We will need a temporary matrix $V \in \mathbb{R}^{n\times b}$ in which to store the vectors v that appeared in (1) and a matrix $T \in \mathbb{R}^{b\times b}$ that appears in the UT transform. In this discussion we will also treat $U \in \mathbb{R}^{n\times b}$, which stored the Householder vectors, as a separate matrix although in practice it overwrites part of A.

Partition all matrices involved as

$$A = \left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right), \quad \hat{A} = \left(\begin{array}{c|c} \hat{A}_{TL} & \hat{A}_{TR} \\ \hline \hat{A}_{BL} & \hat{A}_{BR} \end{array}\right), \quad U = \left(\begin{array}{c|c} U_{TL} & 0 \\ \hline U_{BL} & U_{BR} \end{array}\right),$$

$$V = \left(\begin{array}{c|c} V_{TL} & V_{TR} \\ \hline V_{BL} & V_{BR} \end{array}\right), \quad \text{and} \quad T = \left(\begin{array}{c|c} T_{TL} & T_{TR} \\ \hline 0 & T_{BR} \end{array}\right),$$

where $X_{TL} \in \mathbb{R}^{k\times k}$, $k < b$, for $X \in \{A, \hat{A}, U, V, T\}$. Consider

$$\begin{aligned}
&(I - u_{k-1}u_{k-1}^T/\tau_{k-1}) \cdots (I - u_0 u_0^T/\tau_0)\, \hat{A}\, (I - u_0 u_0^T/\tau_0) \cdots (I - u_{k-1}u_{k-1}^T/\tau_{k-1}) \\
&= \left(I - \left(\frac{U_{TL}}{U_{BL}}\right) T_{TL}^{-T} \left(\frac{U_{TL}}{U_{BL}}\right)^T\right) \left(\begin{array}{c|c} \hat{A}_{TL} & \hat{A}_{TR} \\ \hline \hat{A}_{BL} & \hat{A}_{BR} \end{array}\right) \left(I - \left(\frac{U_{TL}}{U_{BL}}\right) T_{TL}^{-1} \left(\frac{U_{TL}}{U_{BL}}\right)^T\right) \\
&= \left(I - \left(\frac{U_{TL}}{U_{BL}}\right) T_{TL}^{-T} \left(\frac{U_{TL}}{U_{BL}}\right)^T\right) \left(\left(\begin{array}{c|c} \hat{A}_{TL} & \hat{A}_{TR} \\ \hline \hat{A}_{BL} & \hat{A}_{BR} \end{array}\right) - \left(\frac{V_{TL}}{V_{BL}}\right) T_{TL}^{-1} \left(\frac{U_{TL}}{U_{BL}}\right)^T\right),
\end{aligned}$$

where $\left(\frac{V_{TL}}{V_{BL}}\right) = \left(\begin{array}{c|c} \hat{A}_{TL} & \hat{A}_{TR} \\ \hline \hat{A}_{BL} & \hat{A}_{BR} \end{array}\right) \left(\frac{U_{TL}}{U_{BL}}\right)$. The idea is that not all of this has overwritten A. Only the first k columns have been updated with that part of the desired Hessenberg matrix:

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \text{ currently contains } \left(\begin{array}{c|c} B_{TL} & \hat{A}_{TR} \\ \hline \{0\backslash\beta\}_{BL} & \hat{A}_{BR} \end{array}\right),$$

where

$$\left(\frac{B_{TL}}{\{0\backslash\beta\}_{BL}}\right) = \left(I - \left(\frac{U_{TL}}{U_{BL}}\right) T_{TL}^{-T} \left(\frac{U_{TL}}{U_{BL}}\right)^T\right) \left(\left(\frac{\hat{A}_{TL}}{\hat{A}_{BL}}\right) - \left(\frac{V_{TL}}{V_{BL}}\right) T_{TL}^{-1} U_{TL}^T\right).$$

The question now becomes how to update the next column of A so that the next Householder transform can be computed (the next column of U), from which then the next columns of V and T can be computed. Repartition

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array}\right), \quad \left(\begin{array}{c|c} \hat{A}_{TL} & \hat{A}_{TR} \\ \hline \hat{A}_{BL} & \hat{A}_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} \hat{A}_{00} & \hat{a}_{01} & \hat{A}_{02} \\ \hline \hat{a}_{10}^T & \hat{\alpha}_{11} & \hat{a}_{12}^T \\ \hline \hat{A}_{20} & \hat{a}_{21} & \hat{A}_{22} \end{array}\right),$$

$$\left(\begin{array}{c|c} V_{TL} & V_{TR} \\ \hline V_{BL} & V_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} V_{00} & v_{01} & V_{02} \\ \hline v_{10}^T & \nu_{11} & v_{12}^T \\ \hline V_{20} & v_{21} & V_{22} \end{array}\right), \quad \left(\begin{array}{c|c} U_{TL} & 0 \\ \hline U_{BL} & U_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} U_{00} & 0 & 0 \\ \hline u_{10}^T & 0 & 0 \\ \hline U_{20} & u_{21} & U_{22} \end{array}\right),$$

and

$$\left(\begin{array}{c|c} T_{TL} & T_{TR} \\ \hline 0_{BL} & T_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} T_{00} & t_{01} & T_{02} \\ \hline 0 & \tau_{11} & t_{12}^T \\ \hline 0 & 0 & T_{22} \end{array}\right).$$

The next column of A must be updated by

$$\left(\begin{array}{c} b_{01} \\ \hline \beta_{11} \\ \hline b_{21} \end{array}\right) := \left(I - \left(\begin{array}{c} U_{00} \\ \hline u_{10}^T \\ \hline U_{20} \end{array}\right) T_{00}^{-T} \left(\begin{array}{c} U_{00} \\ \hline u_{10}^T \\ \hline U_{20} \end{array}\right)^T \right) \left(\left(\begin{array}{c} \hat{a}_{01} \\ \hline \hat{\alpha}_{11} \\ \hline \hat{a}_{21} \end{array}\right) - \left(\begin{array}{c} V_{00} \\ \hline v_{10}^T \\ \hline V_{20} \end{array}\right) T_{00}^{-1} u_{10} \right)$$

before the next Householder vector can be computed from the so updated b_{21}. After this, the next columns of V and T can be computed by the formulae

$$\left(\begin{array}{c} v_{01} \\ \hline \nu_{11} \\ \hline v_{21} \end{array}\right) := \left(\begin{array}{c|c|c} \hat{A}_{00} & \hat{a}_{01} & \hat{A}_{02} \\ \hline \hat{a}_{10}^T & \hat{\alpha}_{11} & \hat{a}_{12}^T \\ \hline \hat{A}_{20} & \hat{a}_{21} & \hat{A}_{22} \end{array}\right) \left(\begin{array}{c} 0 \\ \hline 0 \\ \hline u_{21} \end{array}\right) = \left(\begin{array}{c} \hat{A}_{02} u_{21} \\ \hline \hat{a}_{12}^T u_{21} \\ \hline \hat{A}_{22} u_{21} \end{array}\right) \tag{5}$$

and

$$\left(\begin{array}{c} t_{01} \\ \hline \tau_{11} \\ \hline 0 \end{array}\right) := \left(\begin{array}{c} U_{20}^T u_{21} \\ \hline \tau_{11} \\ \hline 0 \end{array}\right),$$

where τ_{11} is the value returned by the routine that computes the Householder reflection. We note that V here is accumulated since at some future point this next column of V will be needed in order to update the next column of A.

An algorithm that embodies the above insights is given in Fig. 2.

4.2 A new blocked algorithm

Now we show how a blocked algorithm can be achieved by repeatedly performing the steps in Section 4.1.

Once again, assume the computation has proceeded to where

$$A \rightarrow \left(\begin{array}{c|c} \{U \backslash\backslash B\}_{TL} & A_{TR} \\ \hline \{U \backslash\backslash \beta\}_{BL} & A_{BR} \end{array}\right),$$

where $B_{TL} \in \mathbb{R}^{k \times k}$ and A_{TR} and A_{BR} have been updated according to the Householder transformations computed so far: $A = \check{H}_{k-1} \cdots \check{H}_0 \hat{A} \check{H}_0 \cdots \check{H}_{k-1}$. Reparti-

Algorithm: $[A, V, T] := \textsc{HesRedBuildBlk}(A, V, T)$

Partition $A \rightarrow \left(\begin{array}{c|c} \{U\backslash\!\backslash B\}_{TL} & A_{TR} \\ \hline \{U\backslash\!\backslash \beta\}_{BL} & A_{BR} \end{array}\right)$, $V \rightarrow \left(\begin{array}{c|c} V_{TL} & V_{TR} \\ \hline V_{BL} & V_{BR} \end{array}\right)$, $T \rightarrow \left(\begin{array}{c|c} T_{TL} & T_{TR} \\ \hline 0 & T_{BR} \end{array}\right)$

where $\{U\backslash\!\backslash B\}_{TL}$ is 0×0, V_{TL} is 0×0, T_{TL} is 0×0

while $n(V_{TL}) < n(V)$ **do**

Repartition

$$\left(\begin{array}{c|c} \{U\backslash\!\backslash B\}_{TL} & A_{TR} \\ \hline \{U\backslash\!\backslash \beta\}_{BL} & A_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} \{U\backslash\!\backslash B\}_{00} & a_{01} & A_{02} \\ \hline \{u\backslash\!\backslash \beta\}_{10}^T & \alpha_{11} & a_{12}^T \\ \hline U_{20} & a_{21} & A_{22} \end{array}\right),$$

$$\left(\begin{array}{c|c} V_{TL} & V_{TR} \\ \hline V_{BL} & V_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} V_{00} & v_{01} & V_{02} \\ \hline v_{10}^T & \nu_{11} & v_{12}^T \\ \hline V_{20} & v_{21} & V_{22} \end{array}\right), \left(\begin{array}{c|c} T_{TL} & T_{TR} \\ \hline 0 & T_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} T_{00} & t_{01} & T_{02} \\ \hline 0 & \tau_{11} & t_{12}^T \\ \hline 0 & 0 & T_{22} \end{array}\right)$$

where α_{11} is 1×1 , ν_{11} is 1×1 , τ_{11} is 1×1

$$\left(\begin{array}{c} b_{01} \\ \hline \beta_{11} \\ \hline b_{21} \end{array}\right) := \left(I - \left(\begin{array}{c} U_{00} \\ \hline u_{10}^T \\ \hline U_{20} \end{array}\right) T_{00}^{-T} \left(\begin{array}{c} U_{00} \\ \hline u_{10}^T \\ \hline U_{20} \end{array}\right)^T \right) \left(\left(\begin{array}{c} a_{01} \\ \hline \alpha_{11} \\ \hline a_{21} \end{array}\right) - \left(\begin{array}{c} V_{00} \\ \hline v_{10}^T \\ \hline V_{20} \end{array}\right) T_{00}^{-1} u_{10} \right)$$

via the steps

$$\left\{\begin{array}{l} y_{10} := T_{00}^{-1} u_{10} \\ \left(\begin{array}{c} a_{01} \\ \hline \alpha_{11} \\ \hline a_{21} \end{array}\right) := \left(\begin{array}{c} a_{01} \\ \hline \alpha_{11} \\ \hline a_{21} \end{array}\right) - \left(\begin{array}{c} V_{00} \\ \hline v_{10}^T \\ \hline V_{20} \end{array}\right) y_{10} \\ \left(\begin{array}{c} b_{01} \\ \hline \beta_{11} \\ \hline b_{21} \end{array}\right) := \left(I - \left(\begin{array}{c} U_{00} \\ \hline u_{10}^T \\ \hline U_{20} \end{array}\right) T_{00}^{-T} \left(\begin{array}{c} U_{00} \\ \hline u_{10}^T \\ \hline U_{20} \end{array}\right)^T \right) \left(\begin{array}{c} a_{01} \\ \hline \alpha_{11} \\ \hline a_{21} \end{array}\right) \end{array}\right.$$

$[\{u\backslash\!\backslash \beta\}_{21}, \tau_{11}] := \mathrm{Hous}(a_{21})$ $\qquad$ ($\{u\backslash\!\backslash \beta\}_{21}$ overwrites a_{21})

$$\left(\begin{array}{c} v_{01} \\ \hline \nu_{11} \\ \hline v_{21} \end{array}\right) := \left(\begin{array}{c} \hat{A}_{02} u_{21} \\ \hline \hat{a}_{12}^T u_{21} \\ \hline \hat{A}_{22} u_{21} \end{array}\right); \qquad \left(\begin{array}{c} t_{01} \\ \hline \tau_{11} \\ \hline 0 \end{array}\right) := \left(\begin{array}{c} U_{20}^T u_{21} \\ \hline \tau_{11} \\ \hline 0 \end{array}\right)$$

Continue with

$$\left(\begin{array}{c|c} \{U\backslash\!\backslash B\}_{TL} & A_{TR} \\ \hline \{U\backslash\!\backslash \beta\}_{BL} & A_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c|c} \{U\backslash\!\backslash B\}_{00} & b_{01} & A_{02} \\ \hline \{u\backslash\!\backslash \beta\}_{10}^T & \beta_{11} & a_{12}^T \\ \hline U_{20} & \{u\backslash\!\backslash \beta\}_{21} & A_{22} \end{array}\right),$$

$$\left(\begin{array}{c|c} V_{TL} & V_{TR} \\ \hline V_{BL} & V_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c|c} V_{00} & v_{01} & V_{02} \\ \hline v_{10}^T & \nu_{11} & v_{12}^T \\ \hline V_{20} & v_{21} & V_{22} \end{array}\right), \left(\begin{array}{c|c} T_{TL} & T_{TR} \\ \hline 0 & T_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c|c} T_{00} & t_{01} & T_{02} \\ \hline 0 & \tau_{11} & t_{12}^T \\ \hline 0 & 0 & T_{22} \end{array}\right)$$

endwhile

Fig. 2. Algorithm for building up blocks for the blocked algorithm in Fig. 3.

tion

$$\left(\begin{array}{c|c} \{U\backslash\!\backslash B\}_{TL} & A_{TR} \\ \hline \{U\backslash\!\backslash \beta\}_{BL} & A_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} \{U\backslash\!\backslash B\}_{00} & A_{01} & A_{02} \\ \hline \{U\backslash\!\backslash \beta\}_{10} & A_{11} & A_{12} \\ \hline U_{20} & A_{21} & A_{22} \end{array}\right),$$

where $A_{11} \in \mathbb{R}^{b \times b}$. The idea now is to call the algorithm in Fig. 2 to compute

$$\left(\begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array}\right) := \left(\begin{array}{c|c} \{U\backslash\!\backslash B\}_{11} & A_{12} \\ \hline \{U\backslash\!\backslash \beta\}_{21} & A_{22} \end{array}\right), \quad \left(\begin{array}{c} V_1 \\ \hline V_2 \end{array}\right), \quad \text{and} \quad T_1,$$

where A_{12} and A_{22} are not updated yet. Upon return, the following computations

Algorithm: $[A, V, T] := \text{HesRedBlk}(A, V, T)$

Partition $A \rightarrow \left(\begin{array}{c|c} \{U\backslash\!\backslash B\}_{TL} & A_{TR} \\ \hline \{U\backslash\!\backslash \beta\}_{BL} & A_{BR} \end{array}\right)$, $V \rightarrow \left(\begin{array}{c} V_T \\ \hline V_B \end{array}\right)$, $T \rightarrow \left(\begin{array}{c} T_T \\ \hline T_B \end{array}\right)$

where $\{U\backslash\!\backslash B\}_{TL}$ is 0×0, V_T has 0 rows, T_T has 0 rows

while $m(A_{TL}) < m(A)$ **do**

Determine block size b

Repartition

$$\left(\begin{array}{c|c} \{U\backslash\!\backslash B\}_{TL} & A_{TR} \\ \hline \{U\backslash\!\backslash \beta\}_{BL} & A_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} \{U\backslash\!\backslash B\}_{00} & A_{01} & A_{02} \\ \hline \{U\backslash\!\backslash \beta\}_{10} & A_{11} & A_{12} \\ \hline U_{20} & A_{21} & A_{22} \end{array}\right),$$

$$\left(\begin{array}{c} V_T \\ \hline V_B \end{array}\right) \rightarrow \left(\begin{array}{c} V_0 \\ \hline V_1 \\ \hline V_2 \end{array}\right), \left(\begin{array}{c} T_T \\ \hline T_B \end{array}\right) \rightarrow \left(\begin{array}{c} T_0 \\ \hline T_1 \\ \hline T_2 \end{array}\right)$$

where A_{11} is $b \times b$, V_1 has b rows, T_1 has b rows

$$\left[\left(\begin{array}{c|c} \{U\backslash\!\backslash B\}_{11} & A_{12} \\ \hline \{U\backslash\!\backslash \beta\}_{21} & A_{22} \end{array}\right), \left(\begin{array}{c} V_1 \\ \hline V_2 \end{array}\right), T_1\right] := \text{HesRedBuildBlk}\left(\left(\begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array}\right), \left(\begin{array}{c} V_1 \\ \hline V_2 \end{array}\right), T_1\right)$$

$$V_0 := \left(\begin{array}{c|c} A_{01} & A_{02} \end{array}\right) \left(\begin{array}{c} U_{11} \\ \hline U_{12} \end{array}\right)$$

$$\left(\begin{array}{c|c} A_{01} & A_{02} \end{array}\right) := \left(\begin{array}{c|c} B_{01} & A_{02} \end{array}\right) = \left(\begin{array}{c|c} A_{01} & A_{02} \end{array}\right) - V_0 T_1^{-1} \left(\begin{array}{c} U_{11} \\ \hline U_{21} \end{array}\right)^T$$

$$\left(\begin{array}{c} A_{12} \\ \hline A_{22} \end{array}\right) := \left(I - \left(\begin{array}{c} U_{11} \\ \hline U_{21} \end{array}\right) T_1^{-T} \left(\begin{array}{c} U_{11} \\ \hline U_{21} \end{array}\right)^T\right) \left(\left(\begin{array}{c} A_{12} \\ \hline A_{22} \end{array}\right) - \left(\begin{array}{c} V_1 \\ \hline V_2 \end{array}\right) T_1^{-1} U_{21}^T\right).$$

Continue with

$$\left(\begin{array}{c|c} \{U\backslash\!\backslash B\}_{TL} & A_{TR} \\ \hline \{U\backslash\!\backslash \beta\}_{BL} & A_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c|c} \{U\backslash\!\backslash B\}_{00} & B_{01} & A_{02} \\ \hline \{U\backslash\!\backslash \beta\}_{10} & \{U\backslash\!\backslash B\}_{11} & A_{12} \\ \hline U_{20} & \{U\backslash\!\backslash \beta\}_{21} & A_{22} \end{array}\right),$$

$$\left(\begin{array}{c} V_T \\ \hline V_B \end{array}\right) \leftarrow \left(\begin{array}{c} V_0 \\ \hline V_1 \\ \hline V_2 \end{array}\right), \left(\begin{array}{c} T_T \\ \hline T_B \end{array}\right) \leftarrow \left(\begin{array}{c} T_0 \\ \hline T_1 \\ \hline T_2 \end{array}\right)$$

endwhile

Fig. 3. Blocked algorithm for computing the reduction to upperHessenberg form.

still need to be performed on the non-blank submatrices:

$$\left(\begin{array}{c|c|c} & A_{01} & A_{02} \\ \hline & & A_{12} \\ \hline & & A_{22} \end{array}\right).$$

These parts of the matrix must then be updated by

$$\begin{aligned} \left(\begin{array}{c|c} A_{01} & A_{02} \end{array}\right) &:= \left(\begin{array}{c|c} A_{01} & A_{02} \end{array}\right) \left(I - \left(\begin{array}{c} U_{11} \\ \hline U_{21} \end{array}\right) T_1^{-1} \left(\begin{array}{c} U_{11} \\ \hline U_{21} \end{array}\right)^T\right) \\ &= \left(\begin{array}{c|c} A_{01} & A_{02} \end{array}\right) - V_0 T_1^{-1} \left(\begin{array}{c} U_{11} \\ \hline U_{21} \end{array}\right)^T \end{aligned}$$

and

$$\left(\begin{array}{c} A_{12} \\ \hline A_{22} \end{array}\right) := \left(I - \left(\begin{array}{c} U_{11} \\ \hline U_{21} \end{array}\right) T_1^{-T} \left(\begin{array}{c} U_{11} \\ \hline U_{21} \end{array}\right)^T\right) \left(\left(\begin{array}{c} A_{12} \\ \hline A_{22} \end{array}\right) - \left(\begin{array}{c} V_1 \\ \hline V_2 \end{array}\right) T_1^{-1} U_{21}^T\right).$$

A complete blocked algorithm based on these insights given in Fig. 3.

4.3 Outline of LAPACK-style algorithm

The implementation that is currently part of LAPACK updates A slightly differently. Consider the repartitioning

$$\left(\begin{array}{c|c} \{U\backslash\!\backslash B\}_{TL} & A_{TR} \\ \hline \{U\backslash\!\backslash \beta\}_{BL} & A_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} \{U\backslash\!\backslash B\}_{00} & A_{01} & A_{02} \\ \hline \{U\backslash\!\backslash \beta\}_{10} & A_{11} & A_{12} \\ \hline U_{20} & A_{21} & A_{22} \end{array}\right)$$

where $A_{11} \in \mathbb{R}^{b\times b}$. In the LAPACK implementation, the routine equivalent to HessRedBuildBlk, dlahrd, returns having computed V_0, V_1, and V_2, and having updated A_{01} as well as A_{11} and A_{21} with the final results for those submatrices. Upon return, the updates still need to be performed on the non-blank submatrices:

$$\left(\begin{array}{c|c|c} & & A_{02} \\ \hline & & A_{12} \\ \hline & & A_{22} \end{array}\right).$$

These parts of the matrix must then be updated by

$$A_{02} := A_{02} - V_0 T_1^{-1} U_{21}^T$$

and

$$\left(\frac{A_{12}}{A_{22}}\right) := \left(I - \left(\frac{U_{11}}{U_{21}}\right) T_1^{-T} \left(\frac{U_{11}}{U_{21}}\right)^T\right) \left(\left(\frac{A_{12}}{A_{22}}\right) - \left(\frac{V_1}{V_2}\right) T_1^{-1} U_{21}^T\right).$$

An inefficiency lies with the fact that the update of A_{01} as well as the computation of V_0 are cast in terms of level-2 BLAS rather than level-3 BLAS.

An additional performance hit comes from the fact that the LAPACK implementation touches in every iteration (that is, n times) the whole rest of the matrix (A_{TR}, and A_{BR}). By contrast, in every iteration the algorithm in Section 4.2 only touches A_{BR}. This improves data locality and reduces cache traffic.

We note that the LAPACK-style algorithm described above and its current implementation as part of LAPACK is different than the original LAPACK implementation described in [Dongarra et al. 1989]. The original algorithm proposed in that paper cast even more computation in terms of matrix-vector product.

4.4 Cost analysis

The problem in all algorithms for reducing a matrix to upperHessenberg form is that some of the computation is in level-2 BLAS operations, which attain only a fraction of the peak performance of current architectures. In particular, it is the matrix-vector product in (5) that contributes $O(n^3)$ flops aggregate over all iterations. It is the constant before n^3 that sets the algorithms in Sections 4.2 and 4.3 apart.

For the new algorithm proposed in Section 4.2 the matrix in (5) is roughly $(n-k)\times(n-k)$ during the iteration involving the kth column of A. The total number of flops in this operation, over all iterations, is approximately $2\sum_{k=0}^{n-k}(n-k)^2 \approx \frac{2}{3}n^3$. By contrast, in the LAPACK-like algorithm the matrix in that computation spans all rows and is thus roughly $n \times (n-k)$. Combined over all iterations, the number of flops computed with matrix-vector products for the LAPACK-style algorithm is given by approximately $2\sum_{k=0}^{n-k} n(n-k) \approx n^3$. Recalling that the total cost of a reduction is about $\frac{10}{3}n^3$ flops, the new algorithm performs about 20% of its computation in level-2 BLAS and about 80% in level-3 BLAS. By contrast, the LAPACK-style algorithm spends about 30% in level-2 BLAS and about 70% in level-3 BLAS.

Even though most computation is in high-performing level-3 BLAS, the time spent in these matrix-vector products is often the dominant term since they are executed at a much lower rate. As a result, the reduction of the amount of computation being performed in the matrix-vector products is significant as we will see in performance reported in the next section.

5. EXPERIMENTS

We now demonstrate that by shifting the computation from level-2 BLAS to level-3 BLAS operations, a noticeable performance improvement can be observed.

Three different implementations were tested. The first two were from the LAPACK library: DGEHD2 and DGEHRD which implement an unblocked algorithm and the LAPACK-style blocked algorithm. The third one, FLA_HRD, implements the new blocked algorithm using the Formal Linear Algebra Methods Environment (FLAME) Application Programming Interface (API) for the C programming language. The FLAME APIs allow code to closely resemble the algorithms as they are given in Figs. 1–3. We refer the interested reader to other papers on FLAME [Bientinesi et al. 2005; Bientinesi et al. 2005].

The implementations of the LAPACK-style and the new algorithm do not accumulate T. Rather, they accumulate its inverse, S. Moreover, in our implementation this matrix S is combined with U so that the product US^T is accumulated in a matrix W: $I - UT^{-T}U^T$ becomes $I - WU^T$ and $VT^{-1}U^T$ becomes VW^T. We note that in the previous sections we presented the algorithms using the UT transform since it is a more general way of stating the algorithm before these kinds of details are incorporated.

In Fig. 4 performance is reported for the Xeon (2.4GHz), Pentium4 (1.8GHz), and Itanium2 (900MHz) processors. The Pentium4 has two levels of cache, with a 512 Kbyte L2 cache. The Xeon and Itanium2 each have three levels of cache, with 1 Mbyte and 1.5 Mbyte L3 caches, respectively. On all platforms BLAS libraries implemented by Kazushige Goto were used [Goto 2004]. Since in this paper we are primarily concerned with demonstrating the benefits of the new algorithm rather than a complete study of the effect of blocking on different architectures, the block size was fixed at 32 for both blocked algorithms. From additional experiments it was obvious that for smaller problems smaller blocksizes should be employed. On all three platforms the new algorithm was noticeably faster than the one implemented in LAPACK. As can be expected, the difference was the least for the Itanium2 processor, which has a very fast and very large (1.5 MBytes) L3 cache.

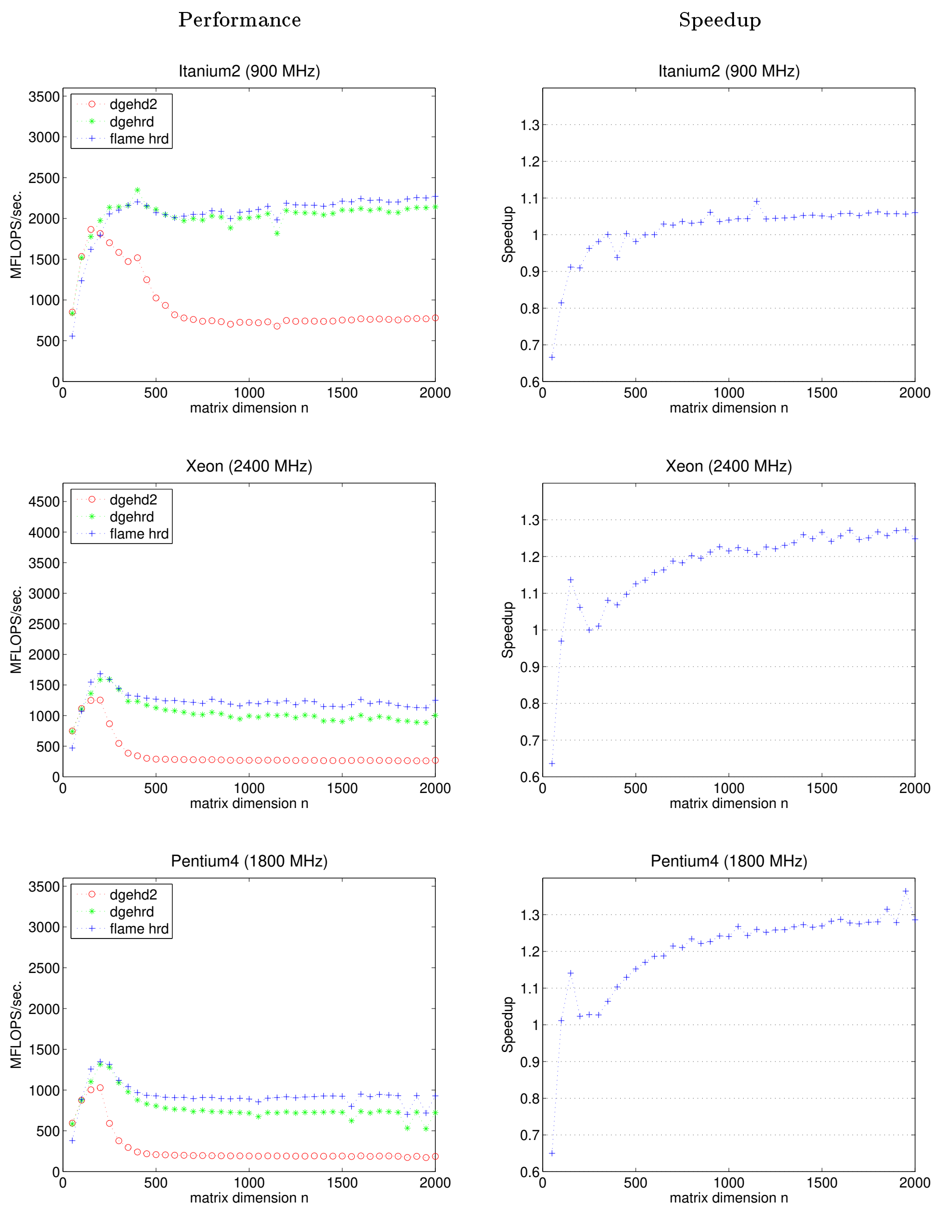

Fig. 4. Performance on various architectures as a function of matrix size, n, using a block size of $nb = 32$. When the matrix-vector product operation is relatively fast (on the Itanium2), the overall performance is better and the speedup that is observed is less, since less time is spent in matrix-vector product. When the matrix-vector product is slow (on the Xeon and Pentium4) the overall performance is worse and the speedup gained by shifting computation from the matrix-vector product to the matrix-matrix product is more noticeable.

6. CONCLUSION

In this paper we have presented a new blocked algorithm for the reduction of a matrix to upperHessenberg form. While the new algorithm performs roughly the same number of computations as the algorithm that is currently included in LAPACK, it shifts more computation to high-performing matrix-matrix computations (level-3 BLAS). As a result, the overall performance of the computation is improved. The predicted improvement in performance was observed in practice on processors that are currently in common use.

Acknowledgments

This research was partially sponsored by NSF grants ACI-0305163 and CCF-0342369 and an equipment donation from Hewlett-Packard.

REFERENCES

Anderson, E., Bai, Z., Demmel, J., Dongarra, J. E., DuCroz, J., Greenbaum, A., Hammarling, S., McKenney, A. E., Ostrouchov, S., and Sorensen, D. 1992. *LAPACK Users' Guide*. SIAM, Philadelphia.

Bartels, R. and Stewart, G. 1972. Solution of the matrix equation $AX + XB = C$: Algorithm 432. *Commun. ACM 15*, 820–826.

Bientinesi, P., Gunnels, J. A., Myers, M. E., Quintana-Ortí, E. S., and van de Geijn, R. A. 2005. The science of deriving dense linear algebra algorithms. *ACM Trans. Math. Soft.*. to appear.

Bientinesi, P., Quintana-Orí, E. S., and van de Geijn, R. A. 2005. Representing linear algebra algorithms in code: The FLAME APIs. *ACM Trans. Math. Soft.*. to appear.

Dongarra, J. J., Du Croz, J., Hammarling, S., and Duff, I. 1990. A set of level 3 basic linear algebra subprograms. *ACM Trans. Math. Soft. 16,* 1 (March), 1–17.

Dongarra, J. J., Du Croz, J., Hammarling, S., and Hanson, R. J. 1988. An extended set of FORTRAN basic linear algebra subprograms. *ACM Trans. Math. Soft. 14,* 1 (March), 1–17.

Dongarra, J. J., Hammarling, S. J., and Sorensen, D. C. 1989. Block reduction of matrices to condensed forms for eigenvalue computations. *Journal of Computational and Applied Mathematics 27*.

Golub, G. H., Nash, S., and Van Loan, C. F. 1979. A Hessenberg–Schur method for the problem $AX + XB = C$. *AC-24*, 909–913.

Golub, G. H. and Van Loan, C. F. 1996. *Matrix Computations*, 3nd ed. The Johns Hopkins University Press, Baltimore.

Goto, K. 2004. `http://www.cs.utexas.edu/users/kgoto`.

Householder, A. S. 1958. Unitary triangularization of a nonsymmetric matrix. *Journal of the ACM 5*, 339–42.

Joffrain, T., Low, T. M., Quintana-Ortí, E. S., van de Geijn, R., and Zee, F. V. On accumulating householder transformations. *TOMS*. submitted.

Martin, R. S. and Wilkinson, J. H. 1968. Similarity reduction of a general matrix to Hessenberg form. *Numer. Math. 12*, 349–68.

Puglisi, C. 1992. Modification of the householder method based on the compact wy representation. *SIAM J. Sci. Stat. Comput. 13*, 723–726.

Sima, V. 1996. *Algorithms for Linear-Quadratic Optimization*. Pure and Applied Mathematics, vol. 200. Marcel Dekker, Inc., New York, NY.

Walker, H. F. 1988. Implementation of the GMRES method using Householder transformations. *SIAM J. Sci. Stat. Comput. 9,* 1, 152–163.

Wilkinson, J. H. 1965. *The Algebraic Eigenvalue Problem*. Oxford University Press, Oxford, England.