A Parallel Algorithm for the Sylvester Observer Equation

Christian H. Bischof[*]

Mathematics and Computer Science Division
Argonne National Laboratory
Argonne, IL 60439

Biswa Nath Datta[†]

Department of Mathematical Sciences
Northern Illinois University
Dekalb, IL 60115

Avijit Purkayastha[‡]

Department of Mathematics
University of Puerto Rico, Mayagüez
Mayagüez, PR-00681-5000

Argonne Preprint ANL/MCS-P274-1191 (revised 05/94)

Abstract. We present a new algorithm for solving the Sylvester observer equation arising in the context of the Luenberger observer. The algorithm embodies two main computational phases: the solution of several independent equation systems, and a series of matrix-matrix multiplications. The algorithm is, thus, well suited for parallel and high-performance computing. By reducing the coefficient matrix A to lower Hessenberg form, one can implement the algorithm efficiently, with few floating-point operations and little workspace. The algorithm has been successfully implemented on a CRAY C-90. A comparison, both theoretical and experimental, has been made with the well-known Hessenberg-Schur algorithm which solves an arbitrary Sylvester equation. Our theoretical analysis and experimental results confirm the superiority of the proposed algorithm, both in efficiency and speed, over the Hessenberg-Schur algorithm.

Key words. Sylvester observer equation, parallel algorithm, orthogonal factorization, shared-memory parallelism, Hessenberg-Schur algorithm.

[*]The work of this author was supported by the Office of Scientific Computing, U.S. Department of Energy, under Contract W-31-109-Eng-38.

[†]The work of this author was partially supported by a grant from the NSF under contract DMS-9212629.

[‡]The work of this author was partially supported by a grant from the NSF under contract RII-8905080.

1 Introduction

The Luenberger observer problem (see [12]) for the control system

$$\begin{aligned} \dot{x} &= Ax + Bu \\ y &= Cx \end{aligned} \tag{1}$$

arises frequently in control theory. Its solution leads to a Sylvester-type matrix equation

$$AX - XH = CV. \tag{2}$$

In contrast with the usual Sylvester equation, here only A and C are given, while X, H, and V are to be chosen to satisfy certain requirements. We call (2) the *Sylvester observer equation.* The requirements for choosing H and V are as follows:

- H must be stable; that is, all the eigenvalues of H should have negative real parts.
- The spectrum of H must be disjoint from that of A (to ensure that X is the unique solution of (2)).
- V must be such that (H^t, V^t) be controllable; that is, the matrix

$$\left[\begin{array}{ccc} V^t, & H^tV^t, \ldots, & (H^t)^{n-1}V^t \end{array}\right]$$

has rank n. (3)

Since we are free to choose H as long as it satisfies the above criteria, we can choose it as a block upper-Hessenberg matrix with a suitable preassigned spectrum. It can then be shown quite easily that, for this particular structure of H, and

$$V = \left(\begin{array}{cccc} 0, & 0, \ldots, & 0, & I \end{array}\right),$$

partitioned conformally, (H^t, V^t) is controllable. This choice of V then reduces Equation (2) to the form

$$AX - XH = \left(\begin{array}{cccc} 0, & 0, & \ldots, 0, & C \end{array}\right), \tag{4}$$

where A is a given $n \times n$ matrix, C is an $n \times r$ matrix, H is a chosen $n \times n$ block upper Hessenberg matrix with a preassigned spectrum, and X is an $n \times n$ matrix to be constructed. Because we can always choose V in this fashion, we simplified the notation in (2) by using C as shorthand for $C(:, n - r + 1 : n)$. It should be clear from context whether the whole matrix or only its last r columns are meant.

If W is an invertible matrix, Equation (2) is equivalent to

$$(WAW^{-1})(WX) - (WX)H = (WC)V, \tag{5}$$

and so one can reduce the complexity of the problem by applying a suitably chosen similarity transformation to A. For example, if we were to apply the well-known Hessenberg-Schur method developed by Golub, Nash, and Van Loan [10] for the usual Sylvester equation, to the Sylvester observer equation (4), A would be reduced to Hessenberg form and H to real Schur form (RSF). The RSF of a matrix is a quasi-triangular matrix in which the diagonal entries are either 1×1 or 2×2 matrices (see [11]). Because the matrix H can be chosen for the Sylvester observer equation, one can choose it in RSF with a desired set of eigenvalues on the diagonal and then easily solve for the columns of X. This approach, though numerically effective, does not offer as good a potential for parallelism as the proposed method. A more detailed discussion of this method, from the point of view of solving the Sylvester observer equation, is given in §5.

Another possible approach is the method suggested by Van Dooren [9]. It uses an observer-Hessenberg form for the pair (A, C) in which both A and C are put in certain condensed forms. This approach also requires knowledge of the eigenvalues of the matrix A and is recursive. Like the Hessenberg-Schur method, it computes the columns of X sequentially and does not offer much scope for the exploitation of parallelism.

Yet another approach, based on Arnoldi's method, has recently been proposed by Datta and Saad [3] for the case where C is a vector. It constructs an orthonormal solution to equation (4). The method is suitable for large and sparse problems but does not offer much scope for parallelism.

In this paper, we present a simple yet efficient method for solving (4) which is well suited for parallel and high-performance computers. The method is a block generalization of Datta's method [2] for the case when $r = 1$. In the case where C is $n \times r$, $r > 1$, our method entails solving a total of n independent system of equations to compute the first r columns of X, and then obtaining the other columns of X, r at a time, essentially through matrix-matrix multiplications. Like the Hessenberg-Schur method, our approach assumes that A is a Hessenberg matrix, and we will not concern ourselves with the reduction of A to Hessenberg form, or the backtransformation of the solution. That is, unless otherwise noted, we assume in the sequel that A in (4) is a lower Hessenberg matrix. We also note that in our approach, H will be chosen to be a block lower bidiagonal matrix.

We also point out that parallel algorithms for control problems are virtually nonexistent, with only a few algorithms being proposed in recent years (for references to these algorithms, see the recent survey papers of Datta [4] and [5]). The need for expanded research in this area has been clearly outlined in a recent panel report [13].

The outline of the paper is as follows. In the next section we present the algorithm and prove its correctness. In §3 we describe how this algorithm can be implemented efficiently. We show that by initially reducing A to lower Hessenberg form, and by employing an orthogonal reduction to solve the equation systems, we can fully exploit parallelism in the solution of the independent equation systems, while requiring little additional workspace. In §4 we report on results obtained on a CRAY C90 shared-memory multiprocessor. In §5 we show how to modify the Hessenberg-Schur method for solving the Sylvester observer equation. Results of its parallel implementation are also presented as a means of comparison with the proposed method. Lastly, we summarize our results and outline directions of further research.

2 The Algorithm

In this section we present our algorithm for solving the Sylvester observer equation and prove its correctness. To repeat, we are trying to solve

$$AX - XH = (0, C), \tag{6}$$

where A and C are given $n \times n$ and $n \times r$ matrices, respectively; X is the sought-after $n \times n$ solution matrix; and H is an $n \times n$ matrix that we can choose as long as it satisfies the requirements of (3). In our algorithm we choose

$$H = \begin{bmatrix} \Lambda_{11} & & & & \\ \Lambda_{21} & \ddots & & & \\ & \ddots & \ddots & & \\ & & \ddots & \ddots & \\ & & & \Lambda_{k,k-1} & \Lambda_{kk} \end{bmatrix}. \tag{7}$$

Let

$$(\lambda_{11}, \lambda_{12}, \ldots, \lambda_{1r}), \ldots, (\lambda_{k1}, \lambda_{k2}, \ldots, \lambda_{kr})$$

be the eigenvalues assigned to the diagonal blocks $\Lambda_{11}, \ldots, \Lambda_{kk}$ of the block bidiagonal matrix H with off-diagonal blocks $\Lambda_{i,i-1}$. All of the blocks are of size $r \times r$, where r is the number of outputs or the number of columns of C, and k is the number of blocks such that $rk = n$. The subdiagonals $\Lambda_{i,i-1}$ containing the scaling factors of the ith block X_i will be computed as a by-product of our solution algorithm such that (6) holds. We partition X conformally such that $X = (X_1, \ldots, X_k)$, where $X_i = (x_1^{(i)}, \ldots, x_r^{(i)})$, and let $C = (c_1, \ldots, c_r)$. In Figure 1 we give our algorithm to solve (6). We now prove the correctness of the algorithm.

Theorem 1 *The algorithm in Figure 1 computes the solution of the Sylvester observer equation (6).*

Proof: First, we notice from the block Hessenberg structure of H in Equation (6) that the blocks X_i of the actual solution X are obtained from the blocks $\hat{X}_i$ of the computed solution $\hat{X}$ by the relation

$$X_{i+1} = \hat{X}_{i+1}\Lambda_{i+1,i}^{-1} = AX_i - X_i\Lambda_{ii}, i = 1, \ldots, k-1,$$

where $\Lambda_{i+1,i}$ of H contain the 2-norm of the columns of the computed solution $\hat{X}_{i+1}$, that is, $\Lambda_{i+1,i} = \text{diag}(\|\hat{x}_1^{(i+1)}\|_2, \ldots, \|\hat{x}_r^{(i+1)}\|_2)$. Then the first two block columns of Equation (6) are related by the relation

$$\hat{X}_2 = X_2\Lambda_{21} = AX_1 - X_1\Lambda_{11}. \tag{8}$$

So if $x_j^{(2)}$ denotes the jth column of X_2, then

$$\hat{x}_j^{(2)} = \|\hat{x}_j^{(2)}\|_2 x_j^{(2)} = (A - \lambda_{1,j}I)x_j^{(1)} \tag{9}$$

```
**Compute** $X_1$**, the first block of** $X$**:**
    **For** $i = 1, \dots, r$ **do**
        **For** $j = 1, \dots, k$ **do**
            Solve $(A - \lambda_{ji} I) y_j = c_i$ for $y_j$
            $\alpha_j = \left( \prod_{\substack{l=1 \\ l \neq j}}^{k} (\lambda_{ji} - \lambda_{li}) \right)^{-1}$
        **Enddo**
        $x_i^{(1)} = \sum_{j=1}^{k} \alpha_j y_j$
    **Enddo**

**Compute** $X_2, \dots, X_k$**, the remaining blocks of** $X$ :
    **For** $i = 1, 2, \dots, k-1$ **do**
        $\hat{X}_{i+1} = A X_i - X_i \Lambda_{ii}$
        $\Lambda_{i+1,i} = \mathrm{diag}(\|\hat{x}_1^{(i+1)}\|_2, \dots, \|\hat{x}_r^{(i+1)}\|_2)$
        $X_{i+1} = \hat{X}_{i+1} \Lambda_{i+1,i}^{-1}$
    **End Do**
```

Figure 1: Algorithm for the solution of the Sylvester observer equation

or

$$x_j^{(2)} = \frac{1}{\|\hat{x}_j^{(2)}\|_2} (A - \lambda_{1,j} I) x_j^{(1)}. \tag{10}$$

The remaining blocks $\hat{X}_3, \dots, \hat{X}_k$ then satisfy :

$$\hat{X}_{i+1} = X_{i+1} \Lambda_{i+1,i} = A X_i - X_i \Lambda_{ii}, i = 2, \dots, k-1. \tag{11}$$

If we denote $\hat{x}_j^{(i)}$ to be the jth column of the ith block $\hat{X}_i$, Equation (11) together with the diagonality of Λ_{ii} implies

$$\hat{x}_j^{(i)} = \|\hat{x}_j^{(i)}\|_2 x_j^{(i)} = \frac{(A - \lambda_{i-1,j} I)}{\|\hat{x}_j^{(i-1)}\|_2} \frac{(A - \lambda_{i-2,j} I)}{\|\hat{x}_j^{(i-2)}\|_2} \dots \frac{(A - \lambda_{2j} I)}{\|\hat{x}_j^{(2)}\|_2} (A - \lambda_{1,j} I) x_j^{(1)}. \tag{12}$$

Thus, $X_2, \dots, X_k$ are completely determined by X_1.

Then, comparing the last block columns in (6), we obtain

$$A X_k - X_k \Lambda_{kk} = C, \tag{13}$$

or

$$(A - \lambda_{kj} I) x_j^{(k)} = c_j, j = 1, \dots, r. \tag{14}$$

Substituting (12) into (14), we obtain

$$\frac{(A-\lambda_{k,j}I)}{\|\hat{x}_j^{(k)}\|_2}\frac{(A-\lambda_{k-1,j}I)}{\|\hat{x}_j^{(k-1)}\|_2}\dots\frac{(A-\lambda_{2j}I)}{\|\hat{x}_j^{(2)}\|_2}(A-\lambda_{1,j}I)x_j^{(1)} = c_j, j=1,\dots,r, \tag{15}$$

or

$$(A-\lambda_{k,j}I)(A-\lambda_{k-1,j}I)\dots(A-\lambda_{1,j}I)x_j^{(1)} = \alpha_j c_j, j=1,\dots,r, \tag{16}$$

where

$$\alpha_j = \Pi_{i=2}^{k}(\|\hat{x}_j^{(i)}\|_2). \tag{17}$$

Thus, if we solve the polynomial system

$$x_j^{(1)} = p_j(A)^{-1}\alpha_j c_j, \text{ where } p_j(x) = (x-\lambda_{kj})(x-\lambda_{k-1,j})\dots(x-\lambda_{1j}), \text{ for } j=1,\dots,r, \tag{18}$$

we solve the equation (6). □

As Equation (18) of the above proof indicates, the obvious bottleneck in a practical implementation of the algorithm is the solution of the polynomial system $p_j(A)x_j^{(1)} = c_j$:

$$(A-\lambda_{kj}I)(A-\lambda_{k-1,j}I)\dots(A-\lambda_{1j}I)x_j^{(1)} = c_j, j=1,\dots,r. \tag{19}$$

The obvious way of solving this system is to successively solve the linear systems

$$(A-\lambda_{ij}I)y_{i-1} = y_i, \quad i=k,k-1,\dots,1$$

with $y_k = c_j$ and $y_1 = x_j^{(1)}$ the final solution. For a very small k, this might not pose any problem. In general, however, it is not satisfactory as it clearly is too expensive. Direct methods for factorization are excluded because A is assumed to be very large. A more effective way, proposed in Datta and Saad [3], is as follows. If we define

$$f(t) = \frac{1}{q(t)} = \frac{1}{\prod_{j=1}^{k}(t-\lambda_{ji})},$$

the desired solution x can be written as

$$x = \sum_{j=1}^{k}\frac{(A-\lambda_{ji}I)^{-1}c_i}{q'(\lambda_{ji})}.$$

In other words, all we need is to solve k independent linear systems

$$(A-\lambda_{ji}I)y_j = c_i, \quad j=1,\dots,k, \tag{20}$$

and then we obtain x as the linear combination

$$x = \sum_{j=1}^{k}\alpha_j y_j,$$

where

$$\alpha_j = \left(\prod_{\substack{l=1 \\ l \neq j}}^{k} (\lambda_{ji} - \lambda_{li}) \right)^{-1}.$$

In our experience, this approach for solving the matrix polynomial does not result in any stability problems for the solution of the Sylvester observer equation for different values of the output parameter. This experience is in line with the results in [3].

We note here that the algorithm in Figure 1 will break down if any of the eigenvalues of A and H are close – Step 1 or Equation (20) will be singular. By using an appropriate test matrix generator, however, we can guarantee that their spectra do not intersect. In general, we have observed experimentally that the spectra of H and A do not intersect for random matrices, although there is a small probability that they might.

3 Efficient Implementation on a Shared-Memory Multiprocessor

In this section we develop an efficient parallel algorithm for the Sylvester observer equation on a shared-memory multiprocessor.

The overall performance of the algorithm hinges critically on efficiently exploiting the apparent parallelism in the computation of X_1, where we have to solve n equation systems $(A - \lambda_{ji} I) y_j = c_i$. Omitting indices for simplicity, we can either use Gaussian elimination or an orthogonal decomposition:

Gaussian Elimination: We decompose $(A - \lambda I)P = LR$, where L is lower triangular, R upper triangular, and P a permutation, then solve the triangular systems $Lw = c$ and $Rz = w$, and lastly undo the permutation by computing $y = Pz$.

Orthogonal Decomposition: Here we have two choices:

QR Factorization: Decompose $(A - \lambda I) = QR$, and solve $Ry = Q^T c$.

LQ Factorization: Decompose $(A - \lambda I) = LQ$, solve $Lz = c$, and compute $y = Q^T z$.

Again, L and R are lower and upper triangular, respectively, and Q is orthogonal. Note that, in contrast to the Gaussian elimination approach, no pivoting is required.

Because we are not interested in the factorization of $(A - \lambda_{ji} I)$ per se, but only in the solution of the equation systems $(A - \lambda_{ji} I) y_j = c_i$, we can perform a forward solve involving a lower triangular matrix L at the same time that we compute L during the factorization. By the same token, we can apply the orthogonal matrix Q to c on the fly if we use the QR factorization.

The drawback is that in both Gaussian elimination and the QR factorization we have to store the upper triangular matrix R, because we cannot start a backsolve involving R before its last element has been computed. Because A is assumed to be dense, both Gaussian elimination and the QR factorization produce a dense upper triangular factor R, which requires a storage of $O(n^2/2)$ words per equation system.

On the other hand, if Q does require little storage, an LQ factorization is well suited. This is the case when A is lower Hessenberg, because then Q can be computed by a sequence of n Givens rotations, requiring $O(n)$ storage and only $O(4n^2)$ flops overall. Assuming, as we have done so far, that A is of lower Hessenberg form, the Sylvester observer equation algorithm in Figure 1 then requires

1. the computation of X_1 by solving a series of systems of equations with lower Hessenberg coefficient matrices, through an LQ factorization; and
2. the computation of $X_2, \ldots, X_k$ through a recurrence relation involving matrix-matrix products of a lower Hessenberg and a dense matrix.

As will be seen, the choice of lower Hessenberg form for A allows us to fully exploit parallelism while keeping extra working storage to a minimum.

3.1 Computing the First Block of the Solution

To compute the first block X_1 of X, we have to solve n independent systems of linear equations with a lower Hessenberg coefficient matrix. Each of the r columns c_i of C is the right-hand side for k equation systems.

We have observed experimentally that the conditioning of the problem or the accuracy of our results is not altered appreciably by varying the value of k relative to n, and we assume the ratio $k = n/r$ to be a small constant. This is motivated by the need for obtaining an efficient loop parallelism strategy, as discussed below.

The LQ solver for solving $(A - \lambda I)y = c$ is shown in Figure 2. Here we assume that the vector y holds c on entry and that it contains the solution x on exit. The vector e_i is the ith canonical unit vector. Let $P = [1 \rightarrow n, 2 \rightarrow 1, \cdots, n \rightarrow n-1]$ be a left cyclical shift. Then $(A - \lambda I)P = [L, t]$, where L is lower triangular and t a column vector.

The use of an LQ factorization for a lower Hessenberg coefficient matrix allows us to compute the orthogonal factor Q, reducing $[L, t]$ to a lower triangular form as a sequence of n Givens rotations:

$$Q = G_1 \cdots G_n,$$

where G_i involves columns i and n. By solving the triangular system $Lz = c$ on the fly, we need to store only two columns of L at any given time. Thus, if we solve p such systems in parallel, we require storage for

$$\underset{\text{(for the lower Hessenberg } A)}{n(n+1)/2} \qquad + \qquad \underset{\text{(work space for equation solvers)}}{4pn}$$

words. The other alternatives considered before (Gaussian elimination, QR factorization) require $O(pn^2)$) storage instead. Our LQ solver requires roughly $4n^2$ flops to solve an $n \times n$ equation system: $3n^2$ flops for the computation of L and Q, and n^2 flops for the forward solve $Lz = c$. The original solution y is then obtained from $y \leftarrow PQ^T y$. The algorithm is shown in Figure 2.

```
Workspace: $w_1, w_2, s$ and $c$, all $n$-vectors.
            $w_1$ and $w_2$ hold the current columns of $L$
            $s$ and $c$ store the $(\sin(\phi_i), \cos(\phi_i))$ pairs.
            We also assume that $A \leftarrow AP$, $P$ being the left cyclic shift.
(1) Initialization: $w_1 = A(:,n) - \lambda e_1, w_2 = A(:,1) - \lambda e_2$
(2) Compute $L$ and forward solve $Ly = c$ on the fly:
    for $i = 1$ to $n-1$
        Generate $(c(i), s(i))$ such that $\begin{pmatrix} c(i) & s(i) \\ -s(i) & c(i) \end{pmatrix} \begin{pmatrix} w_1(i) \\ w_2(i) \end{pmatrix} = \begin{pmatrix} 0 \\ \bullet \end{pmatrix}$
        $(w_1(i:n), w_2(i:n)) \leftarrow (w_1(i:n), w_2(i:n)) \begin{pmatrix} c(i) & s(i) \\ -s(i) & c(i) \end{pmatrix}$
        $y(i) \leftarrow y(i)/w_2(i)$
        $y(i+1:n) \leftarrow y(i+1:n) - y(i) w_2(i+1:n)$
        if $i < n-1$ then
            $w_2(i+1:n) \leftarrow A(i+1:n, i+1) - \lambda e_2$
        endif
    endfor
    $y(n) \leftarrow y(n)/w_1(n)$
(3) Apply $Q^T$ to $y$:
    for $i = n-1$ downto $1$
    $\begin{pmatrix} y(i) \\ y(n) \end{pmatrix} \leftarrow \begin{pmatrix} c(i) & s(i) \\ -s(i) & c(i) \end{pmatrix}^T \begin{pmatrix} y(i) \\ y(n) \end{pmatrix}$
(4) Undo Permutation $P$:
    $y \leftarrow Py$
    endfor
```

Figure 2: LQ solver for $(A - \lambda I)y = c$. On entry, the vector y contains the right-hand side c; on exit, it contains the solution of the equation.

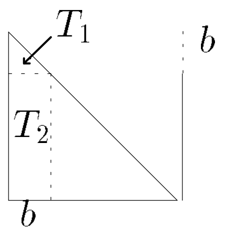

Figure 3: Partially completed LQ factorization

The partial drawback of this algorithm is that it employs vector-vector operations, which in general do not perform well on high-performance processors because they require many memory accesses per floating-point operation. (For a discussion of this issue, see, for example, [1,7,8].) If we allow more workspace per processor, we can partially overcome this drawback and arrive at a variant that computes the forward solve $Lz = c$ with matrix-vector operations. In particular, if we allow for b columns of workspace for L, we do not have to update the right-hand side y until b columns of L have been updated, while eliminating the first b entries of the last column of A, as shown in Figure 3. After obtaining a strictly lower triangular block, as discussed above, we can now compute the first b entries of y with a triangular solve (BLAS2 routine STRSV),

$$y(1:b) \leftarrow T_1^{-1}\, y(1:b),$$

and update the remaining entries of y with a matrix-vector multiplication (BLAS2 routine SGEMV),

$$y(b+1:n) \leftarrow y(b+1:n) - T_2\, y(1:b).$$

Therefore, instead of computing y one column at a time, the computation is carried out b columns at a time. Thus, in the overall algorithm, roughly 25% of the work is now done by using matrix-vector instead of vector-vector kernels.

3.2 Computing the Remaining Blocks of X

The computational performance of the third step depends on the performance of the matrix-matrix product AX_i. To achieve optimal performance, we should compute this matrix-matrix product in parallel, employing matrix-matrix multiplication as much as possible, while exploiting the lower Hessenberg structure of A. To this end, we partition A in block rows A_j of width b (not necessarily the same b that is used for the equation solve) and compute AX_i in a block rowwise fashion. That is, we independently compute the first b rows of AX_i by forming A_1X_i, the second b rows by forming A_2X_i, and so on. In general, A_j will be an $b \times (j * b + 1)$ matrix, with a trailing zero upper triangle. A sample partition of A into four block rows is shown in Figure 4. Since in general $b \ll n$, the computations involving zeros will account for only a small portion of the overall computations performed. In particular, for $b = 1$ we employ a matrix-vector multiplication kernel, and no operations with zeros are performed.

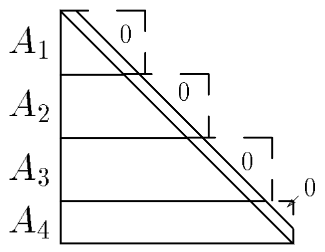

Figure 4: Lower Hessenberg matrix partitioned into four block rows

4 Numerical Results

We tested our parallel Sylvester observer equation algorithm on a CRAY C90/16256, a sixteen-processor vector machine with 256 Mwords of shared memory. The CRAY C90 has two sets of vector units per processor, each producing two results per clock-cycle, resulting in a peak performance of 16 Gflops.

As our test problem we generated matrices of dimensions

$$n = 512, 1024, 1536, 1920.$$

To generate lower Hessenberg matrices with the desired spectrum, we used the LAPACK test matrix generator SLATME [6] to generate a dense nonsymmetric matrix with the desired spectrum, used the LAPACK routine SGEHRD to reduce this matrix to upper Hessenberg form, and then transposed the resulting matrix. In all cases, k, the number of blocks is 4, so $r = n/4$. We checked the accuracy of our results by computing $\hat{C} \equiv A\hat{X}_k - \hat{X}_k\Lambda_{kk}$ which, acccording to (13) should equal C. In all cases, $\hat{C}$ and C agreed to 12 digits. This test is not only cheaper than the usual residual check, but the last block contains the accumulation of all the recurrences and thus is a good indicator of the accuracy of the algorithm.

The BLAS on the CRAY C90 were assembler implementations provided by Cray Research, which exploit multiple CPUs in a fashion that is transparent to the user (unless they are called within a parallel loop, as is the case when we compute the first block of X). Our code obtained the performance and parallel efficiency shown in Figures 5 and 6, respectively.

The plots labeled "First Block of X", and "Remaining Blocks of X" correspond to the two main steps of the Sylvester observer equation algorithm. Figure 7 shows execution rate and efficiency of the two steps combined. In these figures, the solid, dashed, dotted, and dash-dotted lines correspond to runs with 1, 4, 8, and 16 processors, respectively. Efficiency is defined as $\frac{T_1}{T_p \times p} \times 100$, where T_p is the wall clock time for executing any algorithm on p processors. For all of the segments (i.e., systems solutions and matrix products) we get the best results, in general, for blocksize $b = 1$.

Figure 5 shows that the AX_i recursion of the parallel Sylvester observer equation algorithm performs very well. This result is not surprising, because it relies on the highly optimized assembler implementations of the BLAS.

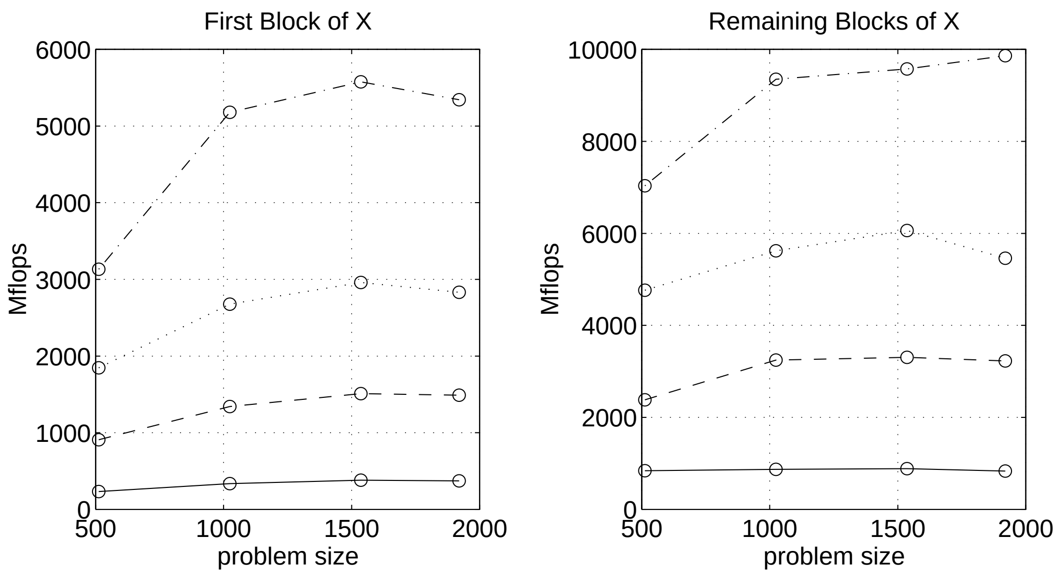

Figure 5: Performance of the main kernels of the parallel Sylvester observer equation algorithm on a CRAY C90 (– 1 proc., - - 4 procs., ... 8 procs., -. 16 procs.)

On the other hand, the computation of the first block of X runs much more slowly, at slightly more than half the speed of the other transformations. This is because our on-the-fly LQ solver relies only on BLAS 1 operations for all of its work and the number of systems being solved. The fact that we blocked the computation of y did not result in any improvement on this machine — at most, 8% principally on smaller problems. As a matter of fact, the performance advantage of the blocked version diminished for larger problems because of an increased number of copies in and out of buffers. We also noted that, due to the high internal bandwidth of the C90, the unblocked matrix-matrix multiply does much better than the blocked version, usually performing around 25% faster. However, as the computation of X_1, and the computations of $X_2, \ldots, X_k$ account respectively for roughly 80% and 20% of the floating-point operations to be performed, the performance of the overall program very much reflects the performance of the LQ solver. There is little we can do about this situation, since any other solver would require $O(pn^2)$ workspace, which is clearly undesirable.

On the other hand, by exploiting the parallelism inherent in the computation of X, our algorithm scales very well with the number of processors, as the plots in Figures 6 and 7 demonstrate. Not surprisingly, the parallel solution of the equation systems scales the best: because there are many parallel jobs, all with the same computational requirements, the parallel loop is almost perfectly load-balanced. This step is responsible for about half of all floating-point operations, resulting in the high overall parallel efficiency of our code.

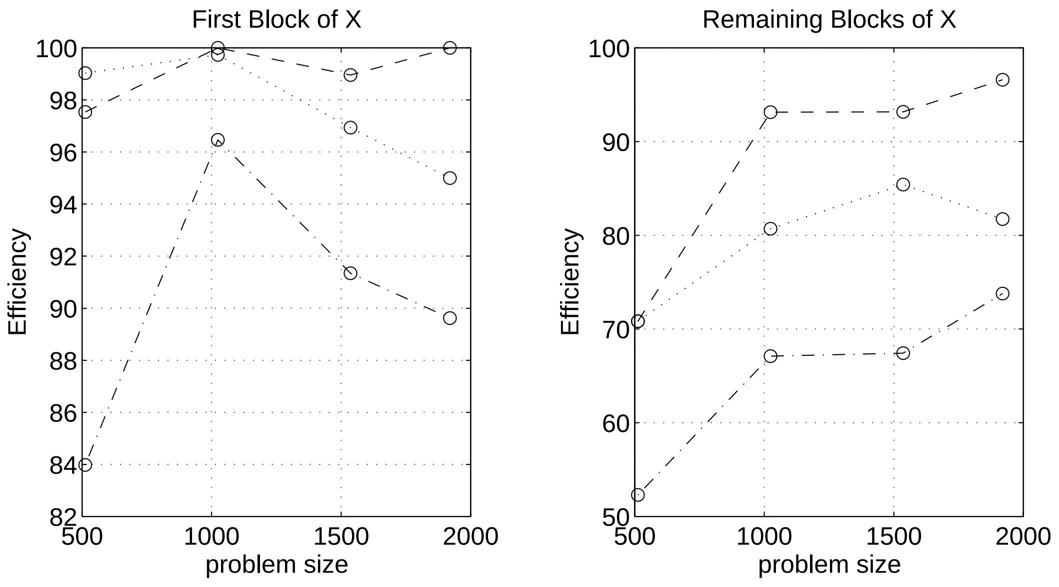

Figure 6: Efficiency of the main kernels of the parallel Sylvester observer equation algorithm on a CRAY C90 (- - 4 procs., ... 8 procs., -. 16 procs.)

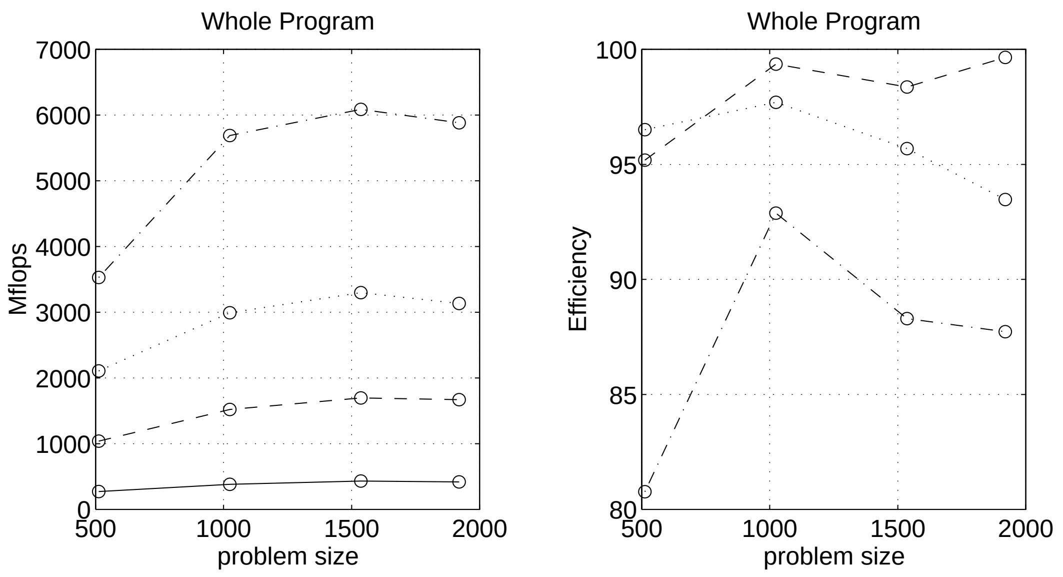

Figure 7: Performance (left) and efficiency (right) of parallel Sylvester observer equation algorithm on a CRAY C90 (– 1 proc., - - 4 procs., ... 8 procs., -. 16 procs.)

5 Comparison of the Proposed Algorithm with Hessenberg Schur Method

In this section we make a performance comparison of our proposed parallel algorithm with that of the Hessenberg-Schur method [10] adapted for the solution of the Sylvester observer equation $AX - XH = (0, C)$. The Hessenberg-Schur method solves the usual Sylvester equation $AX - XH = C$, where the matrices A, H, and C are given and X needs to be found. We first show how the Hessenberg-Schur method can be adapted for the solution of the Sylvester observer equation (2).

In the Sylvester observer equation, the matrix H can be chosen arbitrarily as long as it has a preassigned spectrum and its spectrum is different from that of A. Again, we choose H block bidiagonal with $r \times r$ blocks as in (7). Then the Hessenberg-Schur method adapted to the Sylvester observer equation can be described as given by the algorithm in Figure 8.

Like assumed in our algorithm, A is also first reduced to Hessenberg form. The difference between the two algorithms lies in the computation of the individual blocks of the solution matrix X. In our new algorithm, the first block X_1 is computed by solving n systems of linear equations in parallel, and the remaining blocks are then computed recursively using higher-level BLAS operations, while in the Hessenberg-Schur algorithm, all the blocks of the solution matrix X are computed by solving n linear systems, r at a time. In the Hessenberg Schur approach, it makes sense to parallelize the inner loop, because, as before, r is much larger than k. So we spawn r parallel jobs k times, incurring a greater overhead than in the proposed algorithm, where we spawn n parallel jobs once.

```
**Compute $X_k$, the last block of $X$:**
    **For** $i = 1, \ldots, r$ **do**
        Solve $(A - \lambda_{ki} I)\hat{x}_j^{(k)} = c_i$
    **Enddo**
    $\Lambda_{k,k-1} = \mathrm{diag}(\|\hat{x}_1^{(k)}\|_2, \ldots, \|\hat{x}_r^{(k)}\|_2)$
    $X_k = \hat{X}_k \Lambda_{k,k-1}^{-1}$

**Solve for $X_{k-1}, \ldots, X_1$, the remaining blocks of $X$ :**
    **For** $j = k-1, \ldots, 1$ **do**
        **For** $i = 1, \ldots, r$ **do**
            Solve $(A - \lambda_{ji} I)\hat{x}_i^{(j)} = x_i^{(j+1)}$
        **Enddo**
        $\Lambda_{j,j-1} = \mathrm{diag}(\|\hat{x}_1^{(j)}\|_2, \ldots, \|\hat{x}_r^{(j)}\|_2)$
        $X_j = \hat{X}_j \Lambda_{j,j-1}^{-1}$
    **Enddo**
```

Figure 8: Hessenberg-Schur method for solution of the Sylvester observer equation

Also, there are almost no opportunities for using higher-level BLAS operations in the Hessenberg-Schur algorithm, except in the first step, common to both algorithms, for reducing the matrix to Hessenberg form.

This is borne out by experimental results on the C90. A comparison of Figure 7 with Figure 9 shows that the Hessenberg-Schur algorithm does not perform as well as the new proposed algorithm. For example, for $n = 1536$ and $p = 16$, we obtain around 6 GFlops with the new algorithm and around 5 Gflops with the Hessenberg-Schur approach–an improvement of around 20%. Due to memory limitations (the Hessenberg-Schur method seems to require more space to execute), we could not run the case $n = 1920$.

6 Conclusions

In this paper we presented a new parallel algorithm for solving the Sylvester observer equation. The algorithm is simple and relies on standard linear algebra building blocks. The main computational steps are a reduction to Hessenberg form, the solution of a series of independent equation systems, and a recurrence relation based on matrix-matrix multiplies. These attributes, together with the parallelism in the algorithm, are key requirements for an efficient implementation on a shared-memory multiprocessor. By reducing the coefficient matrix to lower Hessenberg form, we can implement our algorithm with little additional workspace, thereby ensuring that we can solve big problems and that our algorithm scales well with the number of processors. Experimental results on a CRAY C-90 show that the algorithm is indeed well suited for a shared-memory multiprocessor.

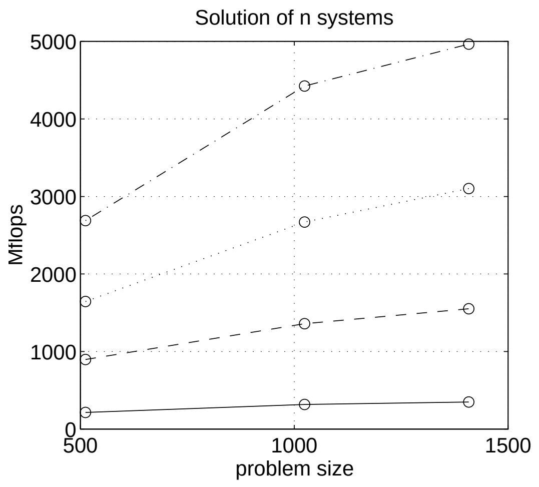

Figure 9: Performance of Hessenberg-Schur algorithm on a CRAY C90; (– 1 proc., - - 4 procs., ... 8 procs., -. 16 procs.)

Also, a comparison is made with the well known Hessenberg-Schur algorithm.

At the moment we are working on a version of this algorithm that is suitable for a distributed-memory multiprocessor. As in our current implementation, the key issue will be an efficient implementation of the parallel equation solves and the limitation of workspace. Research into sparse implementations is also in progress.

Acknowledgments

We are indebted to Bruce Kelly and Moe Jette of the National Energy Research Supercomputer Center at Lawrence Livermore National Laboratory for their dedicated help with the benchmarking of our code. We also thank Klaus Geers of the University of Karlsruhe and Phuong Vu of Cray Research for performing some preliminary benchmark runs for us. We are also indebted to Phuong Vu for suggesting some programming enhancements to us. Lastly, we thank an anonymous referee for suggesting the left cyclic shift reordering in the LQ solver.

References

[1] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. DuCroz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen. *LAPACK User's Guide.* SIAM, Philadelphia, 1992.

[2] Biswa N. Datta. Parallel and large-scale matrix computations in control: Some ideas. *Linear Algebra and Its Applications*, 121:243–264, 1989.

[3] Biswa N. Datta and Youcef Saad. Arnoldi Methods for Large Sylvester-Like Observer Matrix Equations, and an Associated Algorithm for Partial Spectrum Assignment. *Linear Algebra and Its Applications*, 154–156:225–244, 1991.

[4] Biswa N. Datta. Parallel Algorithms in Control Theory. *Proceedings of IEEE Conference on Decision and Control*, pp. 1700-1704, 1991.

[5] Biswa N. Datta. High Performance in Linear Control. *Proceedings of SIAM Conference on Parallel Processing*, pp. 274-281, 1993.

[6] James Demmel and Alan McKenney. LAPACK working note 9: A test matrix generation suite. Preprint MCS-P69-0389, Mathematics and Computer Science Division, Argonne National Laboratory, August 1989.

[7] J. J. Dongarra, F. G. Gustavson, and A. Karp. Implementing linear algebra algorithms for dense matrices on a vector pipeline machine. *SIAM Review*, 26:91–112, 1984.

[8] Jack Dongarra and Sven Hammarling. Evolution of Numerical Software for Dense Linear Algebra. in *Evolution of Numerical Software for Dense Linear Algebra*, M. G. Cox and S. Hammarline, Eds., pages 297–327. Oxford University Press, Oxford, UK, 1989.

[9] Paul Van Dooren. Reduced order observers: A new algorithm and proof. *Systems and Control Letters*, 4:243–251, 1984.

[10] Gene Golub, Stephen Nash, and Charles Van Loan. A Hessenberg Schur method for the problem $AX + XB = C$. *IEEE Transactions in Automatic Control*, 24(6):209–213, 1979.

[11] Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, Baltimore, 1983.

[12] D. Luenberger. Observing the state of a linear system. *IEEE Transaction on Military Electronics*, MIL-8:74-80, 1964.

[13] Report of the panel on *"Future Directions in Control Theory: A Mathematical Perspective"*. SIAM, Philadelphia, 1988.