Fast and Efficient Parallel Algorithms for Problems in Control Theory*

Bruno Codenotti† Biswa N. Datta‡ Karabi Datta‡
Mauro Leoncini§

TR-94-025

August 1994

Abstract

Remarkable progress has been made in both theory and applications of all important areas of control. On the other hand, progress in computational aspects of control theory, especially in the area of large-scale and parallel computations, has been painfully slow. In this paper we address some central problems arising in control theory, namely the controllability and the eigenvalue assignment problems, and the solution of the Lyapunov and Sylvester observer matrix equations. For all these problems we give parallel algorithms that run in almost linear time on a Parallel Random Access Machine model. The algorithms make efficient use of the processors and are scalable, which makes them of practical worth also in the case of limited parallelism.

---

*This paper is in part based on an invited talk delivered to the 1992 American Control Conference in Chicago, Il.

†Istituto di Elaborazione dell'Informazione, Consiglio Nazionale delle Ricerche, Pisa (Italy). Partly supported by ESPRIT Basic Research Action, Project 9072 "GEPPCOM".

‡Department of Mathematical Sciences, Northern Illinois University, Decalb, Illinois. The work of B. N. Datta was supported by a NSF grant under contract number DMS -9212629.

§Dipartimento di Informatica, Università di Pisa, Pisa (Italy). Part of this work was done while the author was visiting the "International Computer Science Institute", Berkeley, CA. Supported by ESPRIT Basic Research Action, Project 9072 "GEPPCOM", and by M.U.R.S.T., 40% and 60% funds.

1 Introduction

Let $\mathbf{x}'(t) = A\mathbf{x}(t) + B\mathbf{u}(t)$ be a continuous-time linear control system, and let

$$\mathbf{x}_{i+1} = A\mathbf{x}_i + B\mathbf{u}_i \tag{1}$$

be its discrete counterpart. Here A and B are constant matrices of dimensions $n \times n$ and $n \times m$, respectively, and $\mathbf{x}(t)$ and $\mathbf{u}(t)$ are time dependent vectors. Linear control systems give rise to a variety of interesting problems, such as

- controllability and osservability,
- eigenvalue assignment (also called pole assignment),
- stability and inertia,
- matrix equations (Sylvester, Lyapunov, Riccati),

just to name a few. These, in turn, give rise to many linear algebra problems [1, 7].

In this paper we focus on the parallel complexity of a number of algorithms that solve some of the above mentioned problems. In particular, we consider the controllability and the eigenvalue assignment problems, and the solution of the Lyapunov and Sylvester observer matrix equations. Some of the algorithms we consider are already known while others are either new or unpublished.

For all the problems addressed, we give algorithms that run in almost linear time in the order of the matrix A on a Parallel Random Access Machine model with sufficiently many processors[1]. The algorithms make efficient use of the processors and are scalable, which makes them of practical worth also in the case of limited parallelism. These nice properties follow from the fact that the algorithms discussed reduce essentially to basic linear algebra computations. The latter are among the most studied parallel computations, for which there is a wide body of literature on both complexity and implementation issues (see, e.g. [20, 8]).

We are also interested in the parallel complexity of the problems investigated. In this respect we are able to prove a quadratic logarithmic upper bound on the parallel time complexity of the controllability and eigenvalue assignment problems. The algorithms that achieve this bound are mostly algebraic in nature and need numerical sophistications to be implemented in practice. Moreover, they are not efficient from the viewpoint of processor utilization. This study is therefore mainly of theoretical interest, but it is nonetheless important [17, 30].

The rest of the paper is organized as follows. In Section 2 we introduce a few basic notions that we will be using throughout the paper when assessing the cost of parallel algorithms; then we recall some simple and efficient algorithms to perform the fundamental linear algebraic operations as well as certain matrix transformations. In Section 3 we study the controllability problem for linear control systems. We show that a controllability criterion introduced by the first author is suitable to both practical (processor efficient/almost

[1] Let n denote the order of the matrix A; then "almost linear" means $O(n \log n)$ while "sufficiently many" here is $O(n^2/\log n)$.

linear time) and asymptotically very fast parallel implementations. Section 4 focusses on the eigenvalue assignment problem. As in the case of controllabilty, we give both practical and asymptotically very fast parallel algorithms. Sections 5 and 6 discuss parallel algorithms for the Lyapunov and Sylvester observer matrix equations, respectively. In both cases we provide processor efficient parallel algorithms that can be implemented in time $O(n^3/p)$ for a wide range of values of the processor bound p.

2 Preliminaries

Conforming to a huge body of literature, we adopt as the underlying computation model the *Parallel Random Access Machines (PRAM)*, a shared memory multiprocessor which has proved to be effective for the design and analysis of parallel algorithms (see, e.g., [24, 25]). More precisely, we adopt the *arithmetic* PRAM model, in which we assume that any arithmetic operation on real numbers can be performed at unit cost (see [8]).

The crucial performance parameters for a PRAM algorithm are the *parallel time* and the maximum *number of processors* operating in parallel during a computation. Clearly, parallel time is a function of both the size of the input problem and the number of processors available. On the other hand, the number of processors can be regarded either as a free parameter or as a function of the input size.

When the number p of processors is a free parameter we have *size independent* parallel algorithms (see [25]). The running time in this case is denoted by $t_p(n)$. On the other hand, in the context of *size dependent* parallel algorithms, we assume an (in principle) unbounded availability of processors and try to design algorithms that minimize time. In this latter case, the time bound of an algorithm will be denoted by $t(n)$, while $p(n)$ will stand for the minimum number of processors that support the $t(n)$ time bound.

For size independent parallel algorithms there is another important performance parameter, namely the *speedup*. Intuitively, we would like that the solution of a certain problem on a p processor machine were p time faster than on a sequential machine. The speedup measures how close we get to this. We now formally define this notion. Let Π be the problem being solved, and let A be the fastest sequential algorithm for Π[2] Finally, let P denote the parallel algorithm under investigation. The speedup of P over A is given by

$$S_p(n) = \frac{\text{Sequential time of } A}{\text{Parallel time of } P \text{ with } p \text{ processors}} = \frac{T(n)}{t_p(n)}.$$

Strictly related to the speedup is the notion of *efficiency* $E_p(n)$ of P with respect to A. This is defined as the ratio between speedup and number of processors:

$$E_p(n) = \frac{S_p(n)}{p} = \frac{T(n)}{t_p(n)p}.$$

Clearly, $E_p(n) \leq 1$. Finally, we define the *work* $w_p(n)$ of a size independent parallel algo-

[2]Sometimes we want to compare a parallel algorithm not with the fastest sequential one, but with some that is important in practical terms. A key example is given by the standard matrix multiplication algorithm with respect to the asymptotically fastest algorithm by Coppersmith and Winograd [9].

rithm as the product of time and number of processors[3]:

$$w_p(n) = t(n)p.$$

We say that a size independent parallel algorithm using p processors is *work* or *processor efficient* (with respect to a given sequential algorithm with running time $T(n)$) if $w_p(n) = O(T(n))$. In the context of size independent parallel algorithms the fundamental goal is the design of algorithms that are processor efficient for the widest possible range of the number of available processors.

The notions of speedup, efficiency, and work are meaningful also in case of size dependent parallel algorithms. However, as already pointed out, the primary concern in this case is towards minimizing the parallel time, largely disregarding the number of processors used. If the resulting speedup is low, it means that the size dependent algorithm uses too many processors to achieve its running time, i.e. it does a lot of work with respect to an efficient size independent algorithm.

The relationships among size dependent and size independent parallel algorithms can be further explained in light of the well-known Brent's scheduling principle (see, for instance, [24]). Suppose that a parallel algorithm P for a problem Π does work w on a p processors PRAM. Then Brent's scheduling principle states that there are implementations of P that do essentially the same work on any q processor PRAM, for any value of $q \in [1, p]$. Essentially, it is based on the simulation idea that any processor of a q processor machine can execute one parallel step of the algorithm being simulated in $\lceil p/q \rceil < p/q + 1$ parallel steps. If t is the running time (on the p processor machine) of the simulated algorithm, then the overall simulation time is bounded by $w/q + t$, and the work done by the simulating machine is thus bounded by $w + tq < 2w$. Therefore, the principle states that it is always possible to *scale down* the degree of parallelism without a substantial decrease in the efficiency. In particular, if a size dependent (independent) parallel algorithm P is work efficient, then there are work efficient size independent implementations of P on any q processors PRAM with q in the range $[1, p(n)]$ ($[1, p]$).

2.1 Basic linear algebra algorithms

In this section we recall some known results about the complexity of certain basic problems that we will be widely using throughout the rest of the paper. In all this section we assume that A and B are $n \times n$ matrices, and that $\mathbf{x}$ and $\mathbf{y}$ are vectors with n elements.

Lemma 1 *The function $f(x_1, \ldots, x_n) = x_1 @ x_2 @ \ldots @ x_n$, where $@$ is any associative binary operation, can be computed in about $2 \log n$ parallel steps provided that $n/\log n$ processors are available.*

Proof Assume, for simplicity, that $k = n/\log n$ is an integer. The algorithm is the following.

[3] More refined notions of the work done by a parallel algorithms are possible. Our measure is rather crude because we assume that the processors that are allocated to an algorithm will be disposed only at the end of its execution. In certain sophisticated parallel computing environments, however, a processor that is no longer needed (or that it's not needed for a long time) can be allocated to a different algorithm running concurrently.

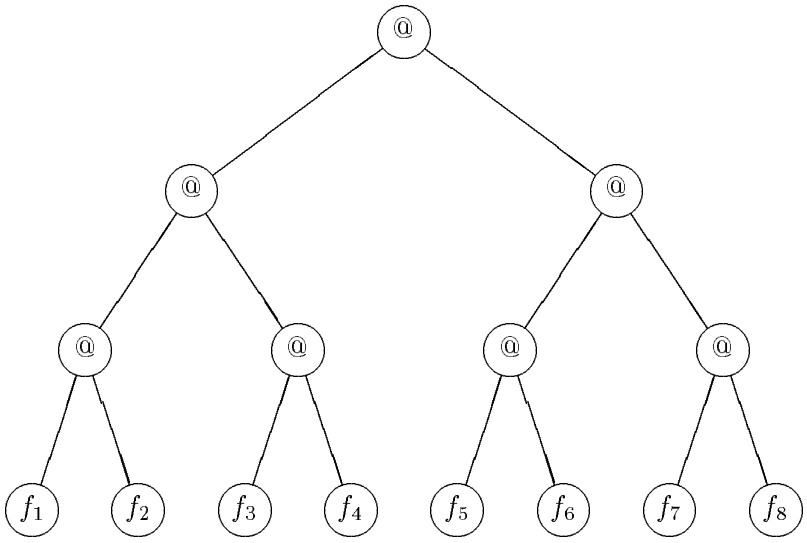

Figure 1: Fanin computation on 8 inputs.

- For $i = 1, \ldots, k$, processor i computes $f_i = x_{(i-1)+1} @ \ldots @ x_{(i-1)+\log n}$. This requires $\log n - 1$ steps and, clearly, $n/\log n$ processors.
- Using the trivial fanin algorithm (see Figure 1, where $k = 8$), the first $k/2$ processors compute $f = f_1 @ f_2 @ \ldots @ f_k$. This requires $\left\lceil \log\left(\frac{n}{\log n}\right)\right\rceil \approx \log n$ parallel steps (which is the height of the tree describing the fanin computation, as shown in Figure 1).

The algorithm of Lemma 1 is almost optimal from the viewpoint of parallel time. Also, it is processor efficient because it does only twice as much work with respect to the best sequential algorithm. Note here that directly applying the fanin algorithm to the n input values would require the availability of $n/2$ processors. In this case the overall work would be $O(n \log n)$, resulting in an efficiency $O(1/\log n)$ that tends to 0 as n grows. Similar remarks can be done regarding the algorithm in the following lemma.

Lemma 2 *The product $\mathbf{x}^T\mathbf{y}$ can be computed in about $3 \log n$ steps, provided that $n/\log n$ processors are available.*

Proof Assume, for simplicity, that $k = n/\log n$ is an integer. Let $\mathbf{x}$ and $\mathbf{y}$ be partitioned as follows: $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_k)$, $\mathbf{y} = (\mathbf{y}_1, \ldots, \mathbf{y}_k)$, where $\mathbf{x}_i$ and $\mathbf{y}_i$ are vectors of $\log n$ elements each, $i = 1, \ldots, k$. The algorithm is the following.

- For $i = 1, \ldots, k$ processor i computes $\mathbf{z}_i = \mathbf{x}_i^T \mathbf{y}_i$. Clearly this requires $2 \log n - 1$ steps.
- Compute $\mathbf{x}^T\mathbf{y} = \mathbf{z}_1 + \ldots + \mathbf{z}_k$. This can be done in time $\log k < \log n$ using $k/2$ processors using the standard fanin algorithm.

Theorem 3 *The product* $A\mathbf{x}$ *can be computed in about* $3\log n$ *parallel steps, provided that* $n^2/\log n$ *are available.*

Theorem 4 *The product* AB *can be computed in about* $3\log n$ *parallel steps, provided that* $n^3/\log n$ *processors are available.*

As an application of the principle of down scalability, consider the algorithm for matrix multiplication. It follows from the principle and from Theorem 4 that p processors PRAM implementations running in parallel time $O(n^3/p)$ are available for any value of p in the range $[1, n^3/\log n]$.

The algorithms in Theorem 3 and 4 are both optimal from the viewpoint of parallel time. They are also work efficient with respect to the classical sequential algorithms for matrix-vector multiplication and matrix multiplication. The matrix multiplication algorithm, however, is not work efficient with respect to the fast sequential algorithms for matrix multiplication [29, 9]. In the rest of this paper we will use $M(n)$ to denote the minimum number of processors that support $O(\log n)$ matrix multiplication. This is related to the best sequential running time for matrix multiplication, which is currently n^α, with $2 < \alpha \leq 2.38$ (see [9]).

From Theorem 4 and the remarks in the previous paragraph we easily obtain the following result.

Corollary 5 *The first* n *powers of* A*, i.e.* $A, A^2, \ldots, A^n$*, can be computed in parallel time* $O(\log^2 n)$ *using* $O(nM(n))$ *processors.*

Theorem 6 *Solving the system of linear equations* $A\mathbf{x} = \mathbf{b}$*, computing* $\det(A)$*, and computing rank*(A) *can all be done in parallel time* $O(\log^2 n)$ *provided that* $O(nM(n))$ *processors are available.*

For a proof of Theorem 6 see, e.g., [6]. Since $n^3/(nM(n)) \to 0$, it follows that the presently available polylogarithmic time parallel linear system solvers are not work efficient. The price that has to be paid to obtain such a large parallelism is therefore a substantial increase in the processor demand.

2.2 Similarity transformations

A number of matrix problems that arise in control theory can be greatly simplified, from the computational viewpoint, in case the matrix (or matrices) involved have the Hessemberg structure. This turns out not to be a loss of generality in many cases. That is, it is often possible to transform the input matrix into the Hessemberg form without substantially increasing the demand of computational resources, and in such a way that the solution of the original problem can be easily recovered from the solution of the "Hessemberg problem". In this section we study two important matrix transformations, namely from general to Hessemberg and from Hessemberg to Companion.

We say that a matrix $H = (h_{ij})$ is lower Hessemberg if $h_{ij} = 0$ for $j > i + 1$. A *companion* matrix is a special kind of Hessemberg's. Let $a(x) = a_0 + a_1 x + \ldots + a_n x^n$ be a

polynomial, and let

$$C_a = \begin{pmatrix} 0 & 1 & 0 & \dots & 0 \\ \vdots & \ddots & 1 & \ddots & \vdots \\ \vdots & & \ddots & \ddots & 0 \\ 0 & \dots & \dots & 0 & 1 \\ -\frac{a_0}{a_n} & -\frac{a_1}{a_n} & \dots & -\frac{a_{n-2}}{a_n} & -\frac{a_{n-1}}{a_n} \end{pmatrix}.$$

The matrix C_a is the companion of the polynomial a. Also, a matrix with the structure of C_a is said to be a companion matrix.

Given a matrix A, its companion C_A is the companion of its characteristic polynomial $det(A - \lambda I) = (-1)^n \lambda^n + a_{n-1}\lambda^{n-1} + \ldots + a_1\lambda + a_0$. It is easy to see that $\det(A - \lambda I) = \det(C_A - \lambda I)$, and thus that A and C_A have the same eigenvalues. Moreover, if A and C_A are similar, then A is said a *nonderogatory* matrix.

The transformations sought are actually *similarity* transformations, i.e. ones of the type $A \mapsto X^{-1}AX$. Given an arbitrary matrix A, we first look for a nonsingular matrix X such that $X^{-1}AX$ is Hessemberg. Then, given an unreduced Hessemberg matrix H, we will determine a nonsingular matrix Y such that $Y^{-1}HY$ is companion.

The matrix X defining the first transformation can be easily determined using Householder's method. In this case X is orthogonal, i.e. is such that $XX^T = I$. As is well-known, Householder's method constructs the matrix X as the product of $n-2$ Householder elementary matrices, $X = P_1 P_2 \ldots P_{n-2}$. Actually, X is not explicitly determined; instead, as soon as any P_i is constructed the transformation $A_i \mapsto A_{i+1} = P_i^T A_i P_i$ is computed (with $A_1 = A$). The construction of each elementary matrix P_i as well as the transformation $P_i^T A_i P_i$ can be computed in $O(\log n)$ parallel time, provided that $O(n^2/\log n)$ processors are available. This follows from the particular structure of the matrix P_i. In fact

$$P_i = I - \frac{2}{\mathbf{v}^T\mathbf{v}}\mathbf{v}\mathbf{v}^T,$$

for a certain vector $\mathbf{v}$, and thus a multiplication by P_i can be easily reduced essentially to the computation of a matrix-vector and an external vector products (for the details see, e.g., [19]). Performing $n-2$ such stages requires therefore $O(n \log n)$ time on an $O(n^2/\log n)$ processor PRAM.

We now address the question of whether an unreduced Hessemberg matrix can be transformed by similarity to its companion form. That is, we seek a nonsingular matrix X such that XHX^{-1} is in companion form. This turns out to be quite simple. Let $\mathbf{x}_i^T$ denote the i^{th} row of the matrix X, $i = 1, \ldots, n$, and assume that

$$XH = CX = \begin{pmatrix} 0 & 1 & 0 & \dots & 0 \\ \vdots & \ddots & 1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & \dots & 0 & 1 \\ c_1 & c_2 & \dots & c_{n-1} & c_n \end{pmatrix} X = \begin{pmatrix} \mathbf{x}_2^T \\ \vdots \\ \mathbf{x}_n^T \\ \mathbf{c}^T X \end{pmatrix},$$

where $\mathbf{c}^T = (c_1, c_2, \ldots, c_n)$. It is therefore easy to see that H and C must satisfy the following equations:

$$\begin{aligned} \mathbf{x}_1^T H &= \mathbf{x}_2^T \\ \mathbf{x}_2^T H &= \mathbf{x}_3^T \\ &\vdots \\ \mathbf{x}_{n-1}^T H &= \mathbf{x}_n^T \\ \mathbf{x}_n^T H &= c_1 \mathbf{x}_1^T + c_2 \mathbf{x}_2^T + \ldots + c_n \mathbf{x}_n^T. \end{aligned} \tag{2}$$

Eliminating $\mathbf{x}_2^T$ through $\mathbf{x}_n^T$, we obtain

$$\mathbf{x}_1^T H^n = c_1 \mathbf{x}_1^T + c_2 \mathbf{x}_1^T H + \ldots + c_n \mathbf{x}_1^T H^{n-1},$$

or, equivalently,

$$\mathbf{x}_1^T \left(H^n - c_1 I - c_2 H - \ldots - c_n H^{n-1} \right) = \mathbf{x}_1^T \Phi(H) = \mathbf{0}. \tag{3}$$

Clearly, forgetting the sign, $\Phi(\lambda)$ is the characteristic polynomial of C, and also of H which is similar to C. It follows by the Cayley-Hamilton theorem that $\Psi(H) = O$, and thus that equation (3) is satified by any arbitrary vector $\mathbf{x}_1^T$. Once $\mathbf{x}_1^T$ has been fixed, the equations (2) allow us to compute all the other rows of the matrix X. Also, it is easy to see that, since H is unreduced, if we choose $\mathbf{x} = \mathbf{e}_1$ then the resulting matrix X is non singular.

Equations (2) are well suitable for fast parallel implementations. Using the $O(\log n)$ time $O(n^2/\log n)$ processors parallel algorithm of Lemma 3 for computing a matrix-vector product, we can compute the matrix X in time $O(n \log n)$ using $O(n^2/\log n)$ processors. By Theorem 4 and Brent's scheduling principle, it follows that the computation of X^{-1} and the products XH and $(XH)X^{-1}$ can still be performed within the same resource bounds. Altogether, the computation of C requires $O(n \log n)$ arithmetic steps on an $O(n^2/\log n)$ processor PRAM. This algorithm is work efficient with respect to the practical $O(n^3)$ sequential methods, but it takes more than linear time.

An upper bound on the parallel time required to compute the matrix C from H is $O(\log^2 n)$. This can be easily seen by observing that $\mathbf{x}_2^T = \mathbf{x}_1^T H, \mathbf{x}_3^T = \mathbf{x}_1^T H^2, \ldots, \mathbf{x}_n^T = \mathbf{x}_1^T H^{n-1}$. Therefore, the cost of computing X is essentially the cost of computing the first n powers of the matrix H. By Corollary 5, this can be done in $O(\log^2 n)$. Finally, the computation of C requires two additional matrix products that can be performed in $O(\log n)$ parallel steps. The number of processors for this very fast parallel algorithm is huge (it depends on the underlying algorithm chosen for performing matrix multiplication).

It is worth mentioning that the above transformation (from Hessemberg form to Companion) can be very ill-conditioned, in the sense that the transformation matrix X might have large condition number. This happens when the elements $h_{i,i+1}$, $i = 1, \ldots, n-1$, of the matrix H are small. For this reason, it is rarely computed in practice. There are applications, however, where the transformation matrix X is well conditioned. In thse cases, dealing with the companion matrix leads to both very fast and accurate algorithms.

3 Controllability

The system $\mathbf{x}'(t) = A\mathbf{x}(t) + B\mathbf{u}(t)$ is said to be controllable if a unique control vector $\mathbf{u}(t)$ can be found such that, in some specified time, $\mathbf{u}(t)$ will derive the system from the initial state $\mathbf{x}(0)$ to a desired state $\mathbf{x}(n)$. Since the system (1) is uniquely identified by the pair of matrices (A, B), we can refer to the controllability problem for (1) as to the problem of *controlling the pair* (A, B).

Many equivalent criteria for determining the controllability of (A, B) are known. If m is close to n or the matrix A is not in Hessemberg form, all these methods are characterized by an operation count as high as n^4, and are therefore impractical. For instance, Kalman's criterion [23] requires that

$$\text{rank}\left(B|AB|\ldots|A^{n-1}B\right) = n.$$

We therefore assume that A is an $n \times n$ lower Hessember matrix while B is any $n \times m$ matrix, with $m << n$. The assumption that A be Hessemberg is not a great loss of generality. In fact, as shown in the previous section, bringing a generic $n \times n$ matrix M into Hessemberg form requires $O(n \log n)$ parallel time using $O(n^2/\log n)$ processors. Also, it is known that a pair (M, N) is controllable if and only if (PMP^{-1}, PN) is, where P is any nonsingular matrix (and, in particular, the matrix that brings M to the Hessemberg form). When the conditions above are satisfied (A Hessemberg and $m << n$) the sequential cost of the practical algorithms for solving the controllability problem is $O(n^3)$.

Our parallel algorithm for testing the controllability of the pair (A, B) is based on the criterion introduced in [10]. In the multi input case, i.e. when $m > 1$, this requires that the matrix A be further brought to the companion form. In the cases when this transformation is numerically stable, the resulting method will be both accurate and fast. As pointed out in Section 2.2, the reduction to companion form of an $n \times n$ matrix can be done $O(n \log n)$ parallel time using $O(n^2/\log n)$ processors. A system (1) in which the matrix A is companion is said in *dual phase variable canonical form.*

All the known controllability criteria, such as the one mentioned due to Kalman, reduce to testing the nonsingularity (or to computing the rank) of certain matrices. In our case, given a companion matrix C,

$$C = \begin{pmatrix} 0 & 1 & 0 & \ldots & 0 \\ \vdots & \ddots & 1 & \ddots & \vdots \\ \vdots & & \ddots & \ddots & 0 \\ 0 & \ldots & \ldots & 0 & 1 \\ c_1 & c_2 & \ldots & c_{n-1} & c_n \end{pmatrix} \tag{4}$$

and a vector $\mathbf{b}$, consider the matrix $X = X_C(\mathbf{b})$ whose rows $\mathbf{x}_i^T$ are defined in the following way:

$$\mathbf{x}_i^T = \begin{cases} \mathbf{b}^T & \text{if } i = n \\ \mathbf{x}_{i+1}^T C - c_{i+1}\mathbf{x}_n^T & \text{if } 1 \le i < n \end{cases} \tag{5}$$

Theorem 7 *[10] Let (C, B) be a system in dual phase variable canonical form. Let $\mathbf{b}_i$ denote the i-th column of B, $i = 1, \ldots, m$. Then (C, B) is controllable if and only if, for all $i \in \{1, \ldots, m\}$, $X_C(\mathbf{b}_i)$ is nonsingular.*

We now prove that the above criterion is well-suited for a number of computational approaches, that are relevant from either a theoretical or a practical viewpoint.

Lemma 8 *Let C be the companion matrix (4), and let* $\mathbf{b}$ *be an n vector. Computing the matrix $X_C(\mathbf{b})$ and testing its singularity takes $O(n)$ parallel time on an $O(n^2)$ processor PRAM.*

Proof Given the particular structure of the matrix C, the i-th row of $X_C(\mathbf{b})$, $i < n$, can be computed in 3 parallel steps using $2n-1$ processors. Therefore $X_C(\mathbf{b})$ can be computed in $3(n-1)$ parallel steps using $2n-1$ processors. To test whether $X_C(\mathbf{b})$ is singular, parallel algorithms exist which requires $O(n)$ time on $O(n^2)$ processors (see, e.g., [28]).

It easily follows from Theorem 7 and Lemma 8 that the controllability of a system in dual phase variable canonical form can be tested in parallel time $O(n)$ using $O(mn^2)$ processors. If we also consider the transformation from Hessemberg to companion, then we obtain slightly superlinear time (i.e. $O(n \log n)$), but we can invoke the down scalability principle to keep the algorithm efficient, dropping the processor demand to $O(mn^2/\log n)$.

We now show that the criterion of Theorem 7 is also suitable for very fast (i.e. poly-logarithmic time) parallel implementations. However, before presenting our next algorithm, we must recall a few facts concerning companion matrices and Bezoutians. Let $a(x) = a_0 + a_1 x + \ldots + a_n x^n$ and $b(x) = b_0 + b_1 x + \ldots + b_n x^n$ be polynomials. It can be easily verified that the expression

$$\frac{a(x)b(y) - b(x)a(y)}{x - y}$$

is a polynomial in x and y, i.e.

$$\frac{a(x)b(y) - b(x)a(y)}{x - y} = \sum_{j,k=0}^{n-1} \beta_{jk} x^j y^k.$$

The matrix $B = (\beta_{jk})$ is called the *Bezoutian* of $a(x)$ and $b(x)$. The Bezoutian associated with $a(x)$ and $b(x)$ is often denoted by $Bez(a,b)$. The following lemma can be easily proved.

Lemma 9 *Let C_a be the companion matrix associated with the polynomial $a(x)$. Then, $C_a^k = Bez(a,1)^{-1} Bez(a,y^k)$.*

The important fact, from the computational point of view, is that both $Bez(a,1)$ and $Bez(a,y^k)$ are Hankel matrices. More precisely,

$$Bez(a,1) = \begin{pmatrix} a_1 & a_2 & \ldots & a_n \\ a_2 & & a_n & \\ \vdots & \iddots & & \\ a_n & & & \end{pmatrix}$$

while $Bez(a,y^k)$ is block diagonal with only two blocks, i.e.

$$Bez(a,y^k) = \begin{pmatrix} B_1 & O \\ O & B_2 \end{pmatrix},$$

where

$$B_1 = \begin{pmatrix} & & & -a_0 \\ & & \iddots & -a_1 \\ & \iddots & \iddots & \vdots \\ -a_0 & -a_1 & \dots & -a_{k-1} \end{pmatrix}, \qquad B_2 = \begin{pmatrix} a_{k+1} & a_{k+2} & \dots & a_n \\ a_{k+2} & & \iddots & \\ \vdots & \iddots & & \\ a_n & & & \end{pmatrix}.$$

Theorem 10 *Let $C = C_a$ be the companion of the polynomial $a(x)$, and let $\mathbf{b}$ be an n vector. For $i = 0, \dots, n-1$ define $c_{i+1} = -a_i/a_n$. Then computing the matrix $X = X_C(\mathbf{b})$, as defined in (5), takes time $O(\log^2 n)$ on an $O(nM(n))$ processors PRAM.*

Proof From the recurrence

$$\mathbf{x}_i^T = \begin{cases} \mathbf{b}^T & \text{if } i = n, \\ \mathbf{x}_{i+1}^T C_a - c_{i+1}\mathbf{x}_n^T & \text{if } 1 \le i < n, \end{cases}$$

it is possible to obtain the following explicit definition of the matrix X:

$$X = \begin{pmatrix} \mathbf{b}^T \left[C_a^{n-1} - c_n C^{n-2} - \dots - c_3 C_a - c_2 I\right] \\ \mathbf{b}^T \left[C_a^{n-2} - c_n C_a^{n-3} - \dots - c_3 I\right] \\ \vdots \\ \mathbf{b}^T \left[C_a^2 - c_n C_a - c_{n-1} I\right] \\ \mathbf{b}^T \left[C_a - c_n I\right] \\ \mathbf{b}^T \end{pmatrix}$$

In other words, using Lemma 9, we have

$$\mathbf{x}_k^T = -\sum_{j=1}^{n-k+1} c_{k+j} \mathbf{b}^T C_a^{j-1} = -\sum_{j=1}^{n-k+1} c_{k+j} \mathbf{b}^T Bez(a,1)^{-1} Bez(a, y^{j-1}), \tag{6}$$

where we set $c_{n+1} = -1$. The following algorithm, that computes (6), achieves the bounds stated.

Algorithm C

1. Solve the linear system $\mathbf{w}^T Bez(a,1) = \mathbf{b}^T$ in time $O(\log n)$ using $O(n)$ processors. These bounds can be obtained thanks to the Hankel structure of $Bez(a,1)$ and using a parallel implementation of the FFT (see [Bi84]).

2. Compute the products $\mathbf{w}_j^T = c_j \mathbf{w}^T$, $j = 2, \dots, n+1$, in one step using n^2 processors.

3. Compute the vector-matrix products $\mathbf{w}_j^T Bez(a, y^k)$, $j = 2, \dots, n+1$, $k = 0, \dots, j-2$. This can be done in time $O(\log n)$ using $O(n^3/\log n)$ processors.

4. Compute $\mathbf{x}_k^T$, $k = 1, \dots, n-1$, in time $O(\log n)$ using $O(n^3/\log n)$ processors.

5. Compute $\det(X)$ in $O(\log^2 n)$ parallel time algorithms using $O(nM(n))$ processors.

Note that steps 1-4 of Algorithm C in Theorem 10 can be carried out in time $O(\log n)$ using $O(n^3/\log n)$ processors. Thus the algorithm is not work efficient only because of step 5. Clearly, any improvement in the cost (either parallel time or processor demand) of computing in parallel the determinant will reflect in an improvement on our algorithm for controllability. We must also point out that the controllability problem when the matrix A is not already in Hessemberg form is not known to be solvable in $O(\log^2 n)$ parallel time. This is becasuse we do not know how to bring a generic matrix into this special form in time $O(\log^2 n)$.

We conclude this section by considering the controllability problem in case of single input, i.e. when B is a vector ($m = 1$). In this case, the reduction to the companion form is not required. To test the controllability of the pair (A, B), where A is lower Hessemberg, it is sufficient to test the singularity of the matrix X, with rows $\mathbf{x}_i$, defined in the following way (see [10]):

$$\mathbf{x}_i^T = \begin{cases} B^T & \text{if } i = n, \\ \frac{1}{a_{i,i+1}} \left(\mathbf{x}_{i+1}^T A - a_{i+1,i+1} \mathbf{x}_{i+1}^T - \ldots - a_{n,i+1} \mathbf{x}_n^T \right) & \text{if } 1 \leq i < n, \end{cases}$$

From the above recurrence, and proceeding as in the multi input case with companion matrices, it is easy to design either almost linear time efficient parallel algorithms, or $O(\log^2 n)$ time, but inefficient, algorithms.

4 Eigenvalue Assignment

Given a pair of matrices (A, B), the *Eigenvalue Assignment Problem* (often referred to as the *pole placement problem* in control theory) is the problem of finding a feedback matrix F such that $A + BF$ has a desired spectrum Ω. To point out the importance of EAP, suppose that the pair (A, B) is controllable but unstable. Then it is natural to investigate whether it can be stabilized by means of a feedback matrix F. Thus EAP is very important in the process of designing a control system.

There are various sequential methods available for solving the pole placement problem. Among the best known are:

- the implicit QR methods [26, 27];
- the matrix equation approach [3];
- the solution via the real Schur form [31].

Some of these approaches do not seem to be suitable for parallel processing. For instance, in the implicit QR-type methods the eigenvalues are assigned one at the time.

In this section we present a processor efficient parallel algorithm for the single input case, i.e. when $B = \mathbf{b}$ is a vector. The algorithm is based on one presented in [13] and runs in almost linear time. Moreover, differently from other algorithms that appear in the literature, it does not require that the matrix A be transformed into Hessemberg form. This allows us to prove (by means of a different "implementation" of the same algorithm) that the parallel time complexity of the problem is much smaller than linear.

Let A be $n \times n$ and let $\mathbf{b}$ be an n vector. Also, let $\Omega = \{\lambda_1, \ldots, \lambda_n\}$ be the desired spectrum. We consider the following algorithm.

Algorithm P.

1. Construct the matrix $L = (l_1|\ldots|l_n)$ defined by

$$\begin{aligned} \mathbf{l}_n &= \mathbf{b}, \\ \mathbf{l}_i &= (A - \lambda_{i+1}I)\mathbf{l}_{i+1}, \quad i = n-1, \ldots, 1. \end{aligned}$$

2. Compute $\mathbf{r} = (A - \lambda_1 I)l_1$.

3. Solve $L^T\mathbf{f} = \mathbf{e}_1$, where $\mathbf{e}_1 = (1, 0, \ldots, 0)^T$.

4. Compute $C = \mathbf{r}\mathbf{f}^T$ and return $A - C$.

Theorem 11 *Assume that $(A, \mathbf{b})$ is controllable. Then the matrix $A - C$ of Algorithm P has the desired spectrum Ω.*

Proof By writing down the matrix L explicitly

$$L = ((A - \lambda_2 I)\ldots(A - \lambda_n I)\mathbf{b}|\ldots|(A - \lambda_n I)\mathbf{b}|\mathbf{b}),$$

we can easily verify the following equality:

$$AL - LB = R,$$

where

$$B = \begin{pmatrix} \lambda_1 & 1 & 0 & \ldots & 0 \\ 0 & \lambda_1 & 1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ \vdots & \ldots & \ddots & \ddots & 1 \\ 0 & \ldots & \ldots & 0 & \lambda_n \end{pmatrix}$$

and R is a matrix with all zeros except for the first column, $\mathbf{r}$, that equals $(A - \lambda_1 I)\mathbf{l}_1$. Now, if $(A, \mathbf{b})$ is controllable, then by the Kalman's characterization of controllability recalled at the beginning of Section 3 it follows that the matrix $(A^{n-1}\mathbf{b}|\ldots|A\mathbf{b}|\mathbf{b})$ is nonsingular. But clearly this is equivalent to saying that L is nonsingular, since its i^{th} column can be expressed as $P_{i-1}(A)\mathbf{b}$, where $P_{i-1}(x)$ is a polynomial of degree exactly $i - 1$, $i = 1, \ldots, n$. Thus there is a unique solution f to the system $L^T\mathbf{f} = \mathbf{e}_1$. Putting all this together we obtain:

$$A - C = A - \mathbf{r}\mathbf{f}^T = \mathbf{r}\mathbf{e}_1^T L^{-1} = A - RL^{-1} = LBL^{-1}.$$

That is, $A - C$ is similar to B and hence has the same spectrum as B.

A processor efficient implementation of Algorithm P can be obtained by using processor efficient implementations of the single steps. This easily leads to an $O(n \log n)$ time bound using $O(n^2/\log n)$ processors. However, using Algorithm P we can also prove that the complexity of the single input pole placement problem is $O(\log^2 n)$.

Lemma 12 *Let A be any $n \times n$ matrix and let $\mathbf{b}$ be an n vector. Also, as in Theorem 6, let $M(n)$ denote the minimum number of processors that support $O(\log n)$ matrix multiplication. Then the set of vectors $A^n\mathbf{b}, A^{n-1}\mathbf{b}, \ldots, A\mathbf{b}, \mathbf{b}$ can be computed in parallel time $O(\log^2 n)$ using $M(n)$ processors.*

Proof Let $\bar{n} = \lceil \log n \rceil$. We first compute the matrix powers

$$A^2, A^4, \ldots, A^{2^{\bar{n}}}.$$

Clearly, this can be done in $O(\log^2 n)$ parallel time using $M(n)$ processors. Now we recursively define a set of matrices B_i, $i = 0 \ldots, \bar{n}$, in the following way.

$$\begin{aligned} B_0 &= (\mathbf{b}|\mathbf{0}|\ldots|\mathbf{0}), \\ B_i &= \left(B_1^{(i-1)}|\ldots|B_{2^{i-1}}^{(i-1)}|A^{2^{i-1}}B_1^{(i-1)}|\ldots|A^{2^{i-1}}B_{2^{i-1}}^{(i-1)}|\mathbf{0}|\ldots|\mathbf{0}\right), \end{aligned}$$

where $B_k^{(j)}$ denotes the kth column of the matrix B_j. In other words, the first 2^{i-1} columns of B_i coincide with the correspondent columns of B_{i-1}, the next 2^{i-1} columns are the first 2^{i-1} columns of $A^{2^{i-1}}B_{i-1}$, and the remaining columns are $\mathbf{0}$. Clearly, the first n columns of $B_{\bar{n}}$ are the vectors that we want to compute. Moreover, the cost of computing B_i is essentially the cost of performing a matrix multiplication. Therefore the parallel time required to compute $B_{\bar{n}}$ is again $O(\log^2 n)$ provided that $M(n)$ processors are available.

Theorem 13 *Algorithm P solves the single input Eigenvalue Assignment Problem in parallel time $O(\log^2 n)$ using $O(nM(n))$ processors.*

Proof Consider the following $n \times (n+1)$ matrix

$$B = (\mathbf{r}|\mathbf{l}_1|\ldots|\mathbf{l}_n),$$

where $\mathbf{r}$ and the $\mathbf{l}_i$ are defined in Algorithm P. Let $B^{(i)}$ denote the ith column of B. We already know that

$$B^{(i)} = \prod_{j=i}^{n} (A - \lambda_j I)\,\mathbf{b}, \quad i = 1, \ldots, n+1,$$

where we have set $\prod_{j=n+1}^{n}(A - \lambda_j I) = I$. We now show how to compute the matrix B in a very fast and efficient way. Consider, for instance, the computation of the first column $B^{(1)} = \mathbf{r}$. It is well known that the coefficients of the polynomial

$$p_1(\lambda) = \prod_{j=1}^{n}(\lambda - \lambda_j) = \lambda^n + \alpha_1^{(1)}\lambda^{n-1} + \ldots + \alpha_{n-1}\lambda + \alpha_n$$

can be computed in parallel time $O(\log n)$ using $O(n)$ processors. These bounds can be achieved using the Fast Fourier Transform algorithm. Once the coefficients α_i and the vectors $A^n\mathbf{b}, \ldots, A\mathbf{b}, \mathbf{b}$ are known, the vector $B^{(1)}$ can be determined in the following way

$$B^{(1)} = A^n\mathbf{b} + \alpha_1 A^{n-1}\mathbf{b} + \ldots + \alpha_{n-1}A\mathbf{b} + \alpha_n\mathbf{b}.$$

The computation of $B^{(1)}$ using the above formula can be done in parallel time $O(\log n)$ using $O(n^2/\log n)$ processors (by applying the result of Lemma 1 to the computation of the single components of $B^{(1)}$). Since the computation of the columns of B can be performed in parallel, it follows that B can be determined in parallel time $O(\log n)$ using $O(n^3/\log n)$ processors. Taking the computation of the vectors $A^i\mathbf{b}$ into account (see Lemma 12) brings the time bound to $O(\log^2 n)$ but leaves the processor bound unchanged, since clearly $M(n) = O(n^3/\log n)$. The last step of Algorithm P can be also be performed within the above bounds. Therefore, the overall cost of the Algorithm is dominated by Step 3, which asks for the solution of a linear system. By Lemma 6, this requires $O(\log^2 n)$ using $O(nM(n))$ processors.

Observe that the $O(\log^2 n)$ implementation of Algorithm P is not processor efficient. As in the case of Algorithm C in Theorem 10, this is due to a determinant (or equivalent) computation. Therefore, any improvement in the processor bound required for the fast (i.e. $O(\log^2 n)$ time) computation of the determinant will result in the same improvement for Algorithm P.

5 Lyapunov Matrix Equation

Let A and C be $n \times n$ matrices. The matrix equation

$$AX + XA^T = -C, \tag{7}$$

is known as the *Lyapunov* matrix equation and has wide applications in linear and nonlinear control theory. It is known that a unique solution X exists to (7) if A and $-A^T$ do not have eigenvalues in common. Moreover, if C is positive definite, then there exists a symmetric solution if A does not have any purely immaginary eigenvalues. In several important applications, such as model reduction and balanced realization, the matrix C has indeed the form $C = BB^T$ and therefore is positive semidefinite.

A variety of methods exist for the solution of the Lyapunov matrix equation. One that is most effective from a numerical point of view is the method of Bartels and Stewart [2]. The method is based on the reduction of the matrix A to the real Schur form (instead of Hessemberg form), with the subsequent solution of the reduced problem. Unfortunately, the method appears not to be suitable for parallel implementations. There are at least two reasons for this. First, the QR iterations, that are most commonly used to reduce a matrix into real Schur form, are highly sequential in nature. Second, the method requires the solution of a series of linear systems which are defined recursively and that must be solved sequentially (unless there is the availability of an exponential number of processors). A different algorithm, based on the reduction of the matrix A to diagonal form, has been recently developed by Hodel and Poola for the sparse Lyapunov equation problem [21]. This is much better from the viewpoint of parallelism, but has numerical limitations due to the diagonal reduction process (see [19]).

In this section, based on the works [14, 11], we consider two different algorithms for the solution of the equation (7) in case C is a rank-one symmetric positive semidefinite matrix. Both algorithms are characterized by running time $O(n \log n)$, provided that sufficiently many processors are available. Both algorithms are processor efficient, and are thus very

appealing when a high accuracy is not a big concern (as is the case in many engineering applications, where a reasonable accuracy but a fast solution is more desirable).

When C is a rank-one symmetric positive semidefinite matrix, (7) can be rewritten as

$$AX + XA^T = -\mathbf{b}\mathbf{b}^T,$$

where $\mathbf{b}$ is a vector. Moreover, it is known that if the pair (A, b) is controllable, then (7) is equivalent to the following matrix equation

$$HY + YH^T = -\alpha^2 \mathbf{e}_n \mathbf{e}_n^T, \tag{8}$$

where $H = PAP^T$ is an unreduced Hessemberg matrix and $P\mathbf{b} = \alpha \mathbf{e}_n$. Once (8) has been solved, the solution X to (7) can be recovered as $X = P^T Y P$. Clearly, there is no loss of generality in assuming $\alpha = 1$ in (8), so that the system becomes

$$HY + YH^T = -\begin{pmatrix} 0 & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \dots & \vdots \\ 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & \dots & 1 \end{pmatrix} \tag{9}$$

Equating the columns on both sides of (9) gives

$$\begin{aligned} H\mathbf{y}_1 + h_{11}\mathbf{y}_1 + h_{12}\mathbf{y}_2 &= \mathbf{0} \\ H\mathbf{y}_2 + h_{21}\mathbf{y}_1 + h_{22}\mathbf{y}_2 + h_{23}\mathbf{y}_3 &= \mathbf{0} \\ &\vdots \\ H\mathbf{y}_{n-1} + h_{n-1,1}\mathbf{y}_1 + h_{n-1,2}\mathbf{y}_2 + \ldots + h_{n-1,n}\mathbf{y}_n &= \mathbf{0} \\ H\mathbf{y}_n + h_{n1}\mathbf{y}_1 + h_{n2}\mathbf{y}_2 + \ldots + h_{nn}\mathbf{y}_n &= \mathbf{e}_n. \end{aligned} \tag{10}$$

Now, eliminating successively $\mathbf{y}_2$ through $\mathbf{y}_n$ in (10), we obtain

$$P\mathbf{y}_1 = \prod_{i=1}^{n-1} h_{i,i+1}\mathbf{e}_n, \tag{11}$$

where P is a matrix polynomial in H. Actually, it can be proved that the matrix P in (11) coincides with $\psi(-H)$, where $\psi(x)$ is the characteristic polynomial of H. Now, the computation of $\psi(-H)$ can be performed very efficiently using the results of the following two lemmas.

Lemma 14 *[14] Given a matrix A, there always exists a matrix U, with rows $\mathbf{u}_i$, $i = 1, \ldots, n$, such that $UA + AU = \mathbf{e}_n \mathbf{r}^T$ (i.e. $UA + AU$'s first $n-1$ rows are zero). Moreover, if $\psi(x)$ is the characteristic polynomial of A, then*

$$\mathbf{r}^T = (-1)^n \mathbf{u}_1^T \psi(-H).$$

Lemma 15 *[12, 22] Let H be an unreduced Hessemberg matrix, and let $g(x)$ and $h(x)$ be two polynomials of same degree such that $\mathbf{e}_1^T g(H) = \mathbf{e}_1^T h(H)$. Then $g(H) = h(H)$. In other words, a matrix polynomial in an unreduced Hessemberg matrix is uniquely determined by its first row.*

A matrix satisfying Lemma 14 can be constructed according to the following scheme. Let $\mathbf{u}_i^T$ denote the i^{th} row of U, $i = 1, \ldots, n$, and let $\mathbf{u}_1$ be given; then

$$\mathbf{u}_{i+1}^T = -\frac{1}{h_{i,i+1}} \left(\sum_{j=1}^{i-1} h_{ij} \mathbf{u}_j + \mathbf{u}_i^T (H + h_{ii} I) \right) \quad i = 1, \ldots, n-1.$$

To compute the first row of $\psi(-H)$ we therefore simply choose $\mathbf{u}_1 = (-1)^n \mathbf{e}_1$ in the construction above. The other rows of $\psi(-H)$, call them $\mathbf{p}_i$, can be determined using the following recurrence:

$$\mathbf{p}_{i+1}^T = \frac{1}{h_{i,i+1}} \left(\mathbf{p}_i^T (H - h_{ii} I) - \sum_{j=1}^{i-1} h_{ij} \mathbf{p}_j \right), \quad i = 1, \ldots, n-1. \tag{12}$$

Putting all this together, we obtain the following algorithm.

Algorithm L.

1. Compute L, with rows $\mathbf{l}_i^T$, $i = 1, \ldots, n$, such that $\mathbf{l}_1 = \mathbf{e}_1$ and
$$LH + HL = \mathbf{e}_n \mathbf{p}^T.$$
2. Compute a matrix P, with rows $\mathbf{p}_i^T$, $i = 1, \ldots, n$, such that $\mathbf{p}_1 = \mathbf{p}$ and, for $i = 1, \ldots, n-1$, $\mathbf{p}_{i+1}$ is given by (12), i.e. $P = \psi(-H)$.
3. Solve the linear system $P\mathbf{y}_1 = \prod_{i=1}^{n-1} h_{i,i+1} \mathbf{e}_n$.
4. Compute the other columns of the matrix Y using the recurrences (10).

Let us now analyze in detail the cost of Algorithm L. Step 1 can be performed in parallel time $O(n \log n)$ provided that $O(n^2/\log n)$ processors are available. In fact, the computation of any row of the matrix L requires one matrix-vector multiplication and the addition of up to $n-1$ vectors. These can be easily performed in parallel time $O(\log n)$ using $O(n^2/\log n)$ processors. The same bounds apply (exactly for the same reasons) also to step 2. Step 3 requires the solution of a linear system, which can be done in linear time, provided that $O(n^2)$ processors are available [28]. Finally, the computation of the columns 2 through n of the matrix Y (step 4) can be done according to (10) in time $O(n \log n)$ on $O(n^2/\log n)$ processors. Altogether, by suitably scaling down the processor bound in step 3, the parallel time bound of Algorithm L is $O(n \log n)$, provided that $O(n^2/\log n)$ processors are available. The algorithm is processor efficient with respect to the classical, widely used sequential implementation of the algorithm by Bartels and Stewart, which is characterized by an $O(n^3)$ operation count.

We now describe a second algorithm for the solution of the equation

$$HX + XH^T = C$$

where H is lower Hessenberg and $C = \mathbf{c}\mathbf{c}^T$ is rank one positive semidefinite. Like the algorithm discussed above, this second algorithm still constructs the matrix polynomial $\psi(-H)$. However, the construction of the solution matrix X does not use the recurrences (10). We begin with a lemma that tells us how to solve the simple equation $SA = A^T S$.

Lemma 16 *[15] Let A be given, and let $\mathbf{q}$ be an arbitrary vector. Let S be a matrix, with rows $\mathbf{s}_1, \ldots, \mathbf{s}_n$, defined according to the following scheme:*

$$\begin{aligned} \mathbf{s}_n &= \mathbf{q} \\ \mathbf{s}_i^T &= \frac{1}{a_{i,i+1}} \left(\mathbf{s}_{i+1}^T A - \sum_{j=i+1}^{n} a_{j,i+1} \mathbf{s}_j \right). \end{aligned}$$

Then $SA = A^T S$.

Lemma 14 and Lemma 16 leave us the freedom of choosing an arbitrary vector in the contruction of both U and S, respectively. This is useful in the contruction of the solution X to the Lyapunov matrix equation.

Theorem 17 *Let $\mathbf{d}$ be the solution to the system $\mathbf{d}^T \psi(-A) = \mathbf{c}^T$. Then a solution to the Lyapunov matrix equation $AX + XA^T = \mathbf{c}\mathbf{c}^T$ is given by $X = U^T S$, where U ans S are constructed as in Lemma 14 and Lemma 16, with $\mathbf{d}^T$ being the first row of U.*

Proof

$$\begin{aligned} XA + A^T X &= U^T S A + A^T U^T S && \text{/* Since} \\ & && \text{/* } X = U^T S \\ &= U^T A^T S + A^T U^T S && \text{/* Choosing } S \text{ as} \\ & && \text{/* in Lemma 16} \\ &= (AU + UA)^T S && \text{/* Algebra} \\ &= R^T S && \text{/* Choosing } U \text{ as} \\ & && \text{/* in Lemma 14} \\ &= \mathbf{r}_n \mathbf{r}_n^T && \text{/* Fixing now} \\ & && \text{/* } \mathbf{s}_n = \mathbf{r}_n \\ &= \left((-1)^n \mathbf{d}^T \psi(-A) \right)^T \left((-1)^n \mathbf{d}^T \psi(-A) \right) && \text{/* By Lemma 14} \\ &= \left((-1)^n \mathbf{c}^T \right)^T \left((-1)^n \mathbf{c}^T \right) && \text{/* By hypothesis} \\ &= \mathbf{c}\mathbf{c}^T && \text{/*} \end{aligned}$$

In view of Theorem 17 it is easy to see that the following algorithm does indeed compute a solution X to the matrix equation $XH + H^T X = \mathbf{c}\mathbf{c}^T$, where H is a lower Hessemberg matrix.

Algorithm L2.

1. Compute L, with rows $\mathbf{l}_i^T$, $i = 1, \ldots, n$, such that $\mathbf{l}_1 = \mathbf{e}_1$ and

$$LH + HL = \mathbf{e}_n \mathbf{p}^T.$$

2. Compute a matrix P, with rows $\mathbf{p}_i^T$, $i = 1, \ldots, n$, such that $\mathbf{p}_1 = \mathbf{p}$ and $\mathbf{p}_i$ given by (12), i.e. $P = \psi(-H)$.

3. Solve the linear system $\mathbf{d}^T P = \mathbf{c}^T$.

4. Compute a matrix U, with rows $\mathbf{u}_i^T$, $i = 1, \ldots, n$, such that $\mathbf{u}_1 = \mathbf{d}$ and

$$UH + HU = \mathbf{e}_n \mathbf{r}^T.$$

5. Compute a matrix S, with rows $\mathbf{s}_i^T$, $i = 1, \ldots, n$, such that $\mathbf{s}_n = \mathbf{r}$ and

$$SH = H^T S.$$

6. Compute $X = U^T S$.

We already know that steps 1-4 can be performed in time $O(n \log n)$ using $O(n^2/\log n)$ processors. The same bounds apply also to the computation of the matrix S in step 5. Step 6, which is a matrix multiplication, can be performed in time $O(\log n)$ on $O(n^3/\log n)$ processors. Altogether, by suitably scaling down the processors used by step 6, the parallel time bound of Algorithm L2 is $O(n \log n)$, provided that $O(n^2/\log n)$ processors are available.

Algorithms L and L2 are characterized by the same asymptotic performance. However, for a fixed number p of processors, the running time of Algorithm L is slightly better. More precisely, if we let $I_p(n)$ stand for the parallel time required to solve an $n \times n$ linear system using p processors, then a more detailed analysis leads to a time bound of approximately $15n^3/p + I_p(n)$ for Algorithm L, and to a time bound $23n^3/p + I_p(n)$ for Algorithm L2. Such a small difference could still justify using Algorithm L2, in case this would guarantee a better numerical accuracy (cf. the numerical experiments reported in [11, 14]).

6 Sylvester-Observer Matrix Equation

In the general Sylvester matrix equation

$$AX - XB = C \tag{13}$$

the matrices A, B, and C are given, and we are asked to find a matrix X which satisfies (13). There are important problems, however, such as the Luenberger Observer problem, in which the matrix B is essentially arbitrary, provided that it satisfies certain conditions. In this section we consider the following restricted form of the Sylvester equation:

$$HX - XL = C, \tag{14}$$

where H is $n \times n$ Hessemberg, $C = (O|F)$, F is $n \times r$, and L is any matrix satisfying the two following conditions:

1. L is stable (i.e. all its eigenvalues have negative real part);

2. L and H have no eigenvalue in common.

Equation (14) results from the above mentioned Luenberger Observer problem in the multi-input (r inputs) case, and we call it the *Sylvester -Observer* equation.

A well-known strategy for solving the Sylvester equation is the Hessenberg-Schur approach due to Golub, Nash, and Van Loan [18]. For the Sylvester-Observer equation a widely used approach is the Observer-Hessenberg method of Van Dooren [16]. Both methods appear not to be easily parallelizable. On the other hand, the method discussed here is suitable for large scale parallel machines. The algorithm has been first presented in [5]; here we prove that optimal (theoretical) speedup is possible for a wide range of values of the number of processors available.

Consider again the equation (14). Clearly, a matrix L satisfying the conditions 1 and 2 stated above could be chosen in the class of diagonal matrices. For stability purposes, it is more convenient to pick L from a slightly larger class. Let

$$L = \begin{pmatrix} \Lambda_{11} & & & \\ \Lambda_{21} & \ddots & & \\ & \ddots & \ddots & \\ & & \Lambda_{k,k-1} & \Lambda_{kk} \end{pmatrix}$$

where each block of L is $r \times r$, with $n = kr$, and let $\{\lambda_1^{(i)}, \ldots, \lambda_r^{(i)}\}$ be the spectrum assigned to Λ_{ii}, $i = 1, \ldots, k$. Clearly, the $\lambda_j^{(i)}$ must be chosen in a way that L satisfies the conditions 1 and 2 above. Let X be partitioned conformally, i.e. $X = (X_1, \ldots, X_k)$, where each X_j is $n \times r$, and let $F = (\mathbf{f}_1, \ldots, \mathbf{f}_r)$.

The following algorithm is adapted from [5] in order to exploit maximal parallelism.

Algorithm S.

1. Solve the n linear systems

$$\left(H - \lambda_j^{(i)} I\right) \mathbf{y}_j^{(i)} = \mathbf{f}_i, \quad 1 \leq i \leq r, 1 \leq j \leq k.$$

2. Compute the n values

$$\alpha_j^{(i)} = \prod_{\substack{l=1 \\ l \neq j}}^{k} \left(\lambda_l^{(i)} - \lambda_j^{(i)}\right), \quad 1 \leq i \leq r, 1 \leq j \leq k.$$

3. Compute the columns $\mathbf{x}_i^{(1)}$ of X_1 as follows

$$\mathbf{x}_i^{(1)} = \sum_{j=1}^{k} \alpha_j^{(i)} \mathbf{y}_j^{(i)}, \quad 1 \leq i \leq r.$$

4. For $i = 1, \ldots, k-1$ do (computation of the remaining blocks)

   (a) $\hat{X}_{i+1} = H X_i - X_i \Lambda_{ii}$

   (b) $\Lambda_{i+1,i} = \text{diag}\left(\|\hat{\mathbf{x}}_1^{(i+1)}\|_2, \ldots, \|\hat{\mathbf{x}}_r^{(i+1)}\|_2\right)$

(c) $X_{i+1} = \hat{X}_{i+1}\Lambda_{i+1,i}^{-1}$

Theorem 18 *[5] Algorithm S above computes the solution X to the Sylvester-Observer equation (14).*

Algorithm S is well-suited to fast parallel implementations.

Cost analysis of Algorithm S.

1. To complete this step we can choose either the polylog fast algorithm or the linear time algorithm for linear system solution. In the latter case we can exploit the Hessemberg structure of H to further bound the processor demand of the linear system solver presented in [28]. In fact, when the input matrix is Hessemberg, any Givens rotation can be performed in costant time, provided that $O(n)$ processors are available. It follows that the linear time solution requires only $O(n)$ processors.

2. Clearly, it follows by Lemma 1 that $O(\log k)$ parallel time is sufficient, provided that $O(rk^2/\log k) = O(nk/\log k)$ processors are available.

3. Each $\mathbf{x}_i^{(1)}$ can be computed in $O(\log k)$ parallel time if $O(nk/\log k)$ processors are available. Altogher, $O(nkr/\log k) = O(n^2/\log k)$ processors are required to complete the step in time $O(\log k)$.

4. The cost of this step is dominated by the execution ($k-1$ times), of substep (a). This is essentially the cost of performing (in sequence) $k-1$ multiplications of a matrix $n \times n$ by a matrix $n \times r$. If $O(n^2 r/\log n)$ processors are available, such cost is clearly $O(\log n)$. The overall cost of step 4 is thus $O(k \log n)$.

Theorem 19 *Algorithm S runs in parallel time $O(\max\{\log^2 n, k \log n\})$, provided that $n^{O(1)}$ processors are available. Moreover, if $k = \Omega(n/\log n)$, a processor efficient implementation can be devised.*

Proof The first statement follows directly from the above cost analysis. To achieve processor efficiency we need to choose the linear time solution in step 1. This requires that the number of processors must not exceed $\Theta(n^2)$. In turn this implies, that $n^2 r/\log n = O(n^2)$ ($n^2 r/\log n$ is the processor demand of step 4), or, equivalently, that $r = O(\log n)$. But this means $k = \Omega(n/\log n)$.

Theorem 19 implies that a processor efficient parallel implementation is possible provided that the number r of inputs is small compared to n.

7 Conclusions and further work

The need for a significant progress in the computational aspects of control theory urges for a tight cooperation between many scientific communities: control theorists, numerical analysts, software engineers, and experts in large-scale and parallel computations. In fact, differently from what has happened in other areas of science and engineering, control scientists and engineers have not been able to take much advantage of the hardware and software

developments of the last years. In this paper we have shown that a number of important problems arising in control theory can be solved using standard linear algebraic computations. The latter are among the most widely studied computations from the viewpoint of parallelism, and a large body of literature on complexity and implementation issues is already available. As an immediate consequence, for some of the central problems in control theory we obtain practical (i.e. processor efficient and scalable) parallel algorithms that are essentially constituted of sequences of calls to linear algebra routines. For some problems we are also able to prove very small upper bounds on the parallel time required. In the latter case, however, the algorithms achieving these bounds are not efficient.

The numerical behavior of our algorithms is still to be fully understood. Preliminary investigations show that the practical algorithms presented in this paper do provide an accuracy that is sufficient in many engineering applications. However, a deeper analysis is still required. Also, we have presented our results using the PRAM model. Though a useful device in the design and analysis of algorithms, the PRAM is not believed to be the "right" model for massive parallelism. What we want to do is to study our algorithms with respect to distributed memory models, such as the mesh and various hypercubic topologies. Given the nature of our algorithms, we argue that efficient distributed memory implementations will exist, as long as efficient implementations exist for the underlying basic linear algebraic computations.

References

[1] S. Barnett, *Introduction to Mathematical Control Theory*, Clarendon, Oxford, 1975.

[2] R. H. Bartels and G. W. Stewart, Solution of the Matrix Equation $AX + XB = C$, *Comm. Ass. Comput. Mach.* 15:820–826, 1972.

[3] S. P. Bhattacharyya and E DeSouza, Pole Assignment via Sylvester's Equation, *Systems Control Letters* 1:261–263, 1982.

[4] D. Bini, Parallel Solution of Certain Toeplitz Linear Systems, *SIAM J. Comput.* 13:268–276, 1984.

[5] C. Bischof, B. N. Datta, and A. Purkayastha, A Parallel Algorithm for the Multi-input Sylvester-Osbserver Equation, *Argonne Technical Report* MCS-P274-1191, revision 1, 1994.

[6] D. Bini and V. Pan, *Polynomial and Matrix Computations: Fundamental Algorithms*, Birkhäuser, Boston, 1994.

[7] C. T. Chen, *Linear System Theory and Design*, Holt, Rinehart and Wilson, 1984.

[8] B. Codenotti and M. Leoncini, *Parallel Complexity of Linear System Solution*, World-Scientific Pu. Co., Singapore, 1991.

[9] D. Coppersmith and S. Winograd, Matrix Multiplication via Arithmetic Progression, 19th *Annual ACM Symposium on Theory of Computing*, 1987, 1–6.

[10] B. N. Datta, A New Criterion of Controllability, *IEEE Trans. Aut. Control* AC-29:444–446, 1984.

[11] B. N. Datta, Parallel Algorithms for the Lyapunov Matrix Equation, *manuscript.*

[12] B. N. Datta and K. Datta, An Algorithm for Computing Powers of a Hessemberg Matrix and its Applications, *Linear Algebra Appl.* 14:273–284, 1976.

[13] B. N. Datta and K. Datta, On Parallel Arithmetic Complexities of Some Linear Algebra Problems, Manuscript, Northern Illinois University, Dekalb, Illinois, 1984.

[14] B. N. Datta and K. Datta, A Fast Solution Method for Positive Semidefinite Lyapunov Matrix Equation, *Proc. 2nd Latin American Conference on Applied Mathematics*, Rio de Janeiro, 83–104.

[15] B. N. Datta and K. Datta, The Matrix Equation $XA = A^T X$ and an Associated Algorithm for Solving the Inertia and Stability Problems, *Linear Algebra Appl.* 97:103–119, 1987.

[16] P. Van Dooren, Reduced Order Observers: a New Algorithm and Proofs, *Systems and Control Letters* 4:243–251, 1984.

[17] J. Von zur Gathen, Parallel Linear Algebra. In: J. Reif (ed.), *Synthesis of Parallel Algorithm*, Morgan and Kaufmann Publishers, San Mateo, CA, 1993, 573–617.

[18] G. H. Golub, S. Nash, and C. Van Loan, A Hessenberg-Schur Method for the Problem $AX + XB = C$, *IEEE Trans. Aut. Control* 24:209–213, 1979.

[19] G. H. Golub and C. F. Van Loan, *Matrix Computations*, The Johns Hopkins University Press, Baltimore, 1989.

[20] D. Heller, A Survey of Parallelism in Numerical Linear Algebra, *SIAM Rev.* 20:740–777, 1978.

[21] A. S. Hodel and K. Poola, Parallel Solution of Large Lyapunov Equations, *SIAM J. Matrix Analysis Appl.* 13:1189–1203 1992.

[22] C. P. Huang, Computing Powers of Arbitrary Hessenberg Matrices, *Linear Algebra Appl.* 21:123–134, 1978.

[23] R. E. Kalman, On the General Theory of Control Systems, *Proc. 1st IFAC Congress*, vol. 1, 1960, 481–491.

[24] R. M. Karp and V. Ramachandran, Parallel Algorithms for Shared Memory Machines. In: J. van Leeuwen (ed.), *Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity*, The MIT Press/Elsevier, 1990, 869–941.

[25] C. P. Kruskal, L. Rudolph, and M. Snir A Complexity Theory of Efficient Parallel Algorithms *Theoret. Comput. Sci.* 71:95–132, 1990.

[26] G. Minimis and C. C. Paige, An Algorithm for Pole Assignment of Time Invariant Linear Systems, *International Journal on Control* 35:130–138, 1981.

[27] R. V. Patel and P. Misra, Numerical Algorithms for Eigenvalue Assignment by State Feedback, *Proc. of the IEEE* 17:1755–1764, 1984.

[28] A. H. Sameh and D. J. Kuck On Stable Parallel Linear System Solvers, *J. Assoc. Comput. Mach.* 25:81–91, 1978.

[29] V. Strassen Gaussian Elimination is not Optimal *Numer. Math.* 13:354–356, 1969.

[30] V. Strassen Algebraic Complexity Theory. In: J. van Leeuwen (ed.), *Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity*, The MIT Press/Elsevier, 1990, 633–672.

[31] A. Varga, A Schur Method for Pole Assignment, *IEEE Trans. Aut. Control* 26:517–519, 1981.