SIAM J. COMPUT.
Vol. 24, No. 4, pp. 811–821, August 1995

008

WORK-PRESERVING SPEED-UP OF PARALLEL MATRIX COMPUTATIONS*

VICTOR Y. PAN† AND FRANCO P. PREPARATA‡

Abstract. Brent's scheduling principle provides a general simulation scheme when fewer processors are available than specified by the fastest parallel algorithm. Such a scheme preserves, under slow-down, the actual number of *executed* operations, also called *work*. In this paper we take the complementary viewpoint, and rather than consider the work-preserving slow-down of some fast parallel algorithm, we investigate the problem of the achievable speed-ups of computation while preserving the work of the best-known sequential algorithm for the same problem. The proposed technique, eminently applicable to problems of matrix-computational flavor, achieves its result through the interplay of two algorithms with significantly different features. Analogous but structurally different "interplays" have been used previously to improve the algorithmic efficiency of graph computations, selection, and list ranking. We demonstrate the efficacy of our technique for the computation of path algebras in graphs and digraphs and various fundamental computations in linear algebra. Some of the fundamental new algorithms may have practical value; for instance, we substantially improve the algorithmic performance of the parallel solution of triangular and Toeplitz linear systems of equations and the computation of the transitive closure of digraphs.

Key words. parallel algorithms, processor efficiency, work-optimal algorithms, linear system of equations, path algebras, paths in graphs

AMS subject classifications. 68Q22, 68Q25, 68R10

1. Introduction.

The main objective of parallel computation, which more sharply contrasts it against sequential computation, has traditionally been the minimization of computation time t, i.e., of the number of parallel steps required to solve a given problem. Another important criterion, however, is the size of the equipment, expressed as the number p of processors used in the computation. Assume, for simplicity, that a single parameter n characterizes the problem size. When t is the chosen performance criterion, frequently the resulting algorithm running in time $t(n)$ involves a very large number $p(n)$ of processors (all assumed to be identical and capable of executing one arithmetic operation in unit time). It is reasonable to assume, however, that in general the number p of available processors will not match the requirements of the fastest algorithm for a given problem instance, i.e., $p \leq p(n)$, which reflects the situation where the number of available processors is fixed and the choice of p is dictated by economic as well as engineering reasons.

Thus a typical situation is one where we have far fewer processors for use than are necessary to achieve the minimum computation time $t(n)$; this situation is dealt with by means of a version of *Brent's scheduling principle* [Br74], [KR90], which embodies a general simulation scheme of a system with $t(n)$ processors by one with a fixed number p of processors, such that

$$1 \leq p \leq p(n).$$

Specifically, let $w(n)$ denote the total number of operations actually executed by the larger (faster) system in time $t(n)$. Then the smaller (slower) system can accomplish the same task in time

$$t \leq t(n) + w(n)/p. \tag{1}$$

*Received by the editors June 9, 1992; accepted for publication (in revised form) March 4, 1994. This work was supported in part by National Science Foundation grants CCR 90-20690 and CCR-91-96152 and PSC CUNY awards 661340 and 662478. A preliminary version of this paper appeared under a different title in *Proc. 4th Annual ACM Symp. on Parallel Algorithms and Architectures* [PP92].

†Department of Mathematics and Computer Science, Lehman College, CUNY, Bronx, New York 10468 (vpan@lcvax.bitnet).

‡Computer Science Department, Brown University, Providence, Rhode Island 02912, (franco@cs.brown.edu).

A basic assumption for this simulation scheme is that the desired allocation of the p processors to their tasks may be done simply; as observed in [KR90] and illustrated below, such processor allocation is not always a straightforward matter.

This simulation scheme can be called *work-preserving slow-down* [PP92], since, while the computation is slowed down (due to the limited resources), the total amount of *actual work* $w(n)$ is preserved. Relation (1) shows that p and t are essentially inversely proportional. It also shows that the computation time is just doubled if we balance the two terms in the right-hand side of (1), i.e., if we choose $p = w(n)/t(n)$, so that for a constant penalty in computation time we may accrue a much more substantial equipment advantage. This point is illustrated by the following classical example.

Example. Summation of n numbers, $a_1, a_2, \ldots, a_n$. (For simplicity, let $\log n$ and $\log \log n$ be positive integers.) The straightforward algorithm, allotting exactly one time unit to each execution of Step 3, uses $t(n) = \log n$ steps and $n/2$ processors:

1. **begin foreach** $i = 1, \ldots, n$ **pardo** $a_{i0} := a_i$;
2. **for** $h = 1$ **to** t **do**
3. **foreach** $i = 1, \ldots, n$ **pardo** $a_{ih} := a_{2i-1,h-1} + a_{2i,h-1}$;
4. $s := a_{it}$
5. **end**

In this case, $w(n) = n - 1$. We may achieve balancing by slowing down the first $\log \log n$ steps of loop 2–3, so that we use only $n/t(n) = n/\log n$ processors at these steps. At subsequent steps, these processors are fully adequate to simulate the original system with no slow-down. Although this simulation implicitly refers to a PRAM model, the interpretation of "balancing" becomes more enlightening in the network model, where the system is a binary tree network of processors with $n/(2 \log n)$ leaves and (i) the input numbers are organized as $\log n$ wavefronts of $n/\log n$ items each, (ii) each wavefront is separately tallied by the network, and (iii) the global sum is accumulated at the root. In this case the alluded-to inverse proportionality of p and t becomes explicit if one uses a variant of Brent's principle, where the work $w(n)$ is replaced by the product $p(n) \cdot t(n)$, called "cost" in [J92]. Note that $p(n) \cdot t(n)$ is the number of executable operations if all processors are kept busy during $t(n)$ time steps, and correctly represents a cost, since it expresses the expected return on investment. If a computation C can be performed in time t with sp processors (at a cost of tsp), each executing one arithmetic operation in unit time, then C can be performed in time st with p processors, for any $s \geq 1$. This statement is readily verified by noting that each time unit of the faster system can be simulated in s time units by the slower one. Note, however, that for $p \leq p(n)$, work and cost are of the same order (see, e.g., [J92]).

Brent's principle gives a straightforward general simulation scheme that preserves the work (or cost) of some parallel computation as we decrease the number p of deployed processors. However, frequently the work $w(n)$ of the fast parallel algorithm is substantially larger than the work $w_{\text{seq}}(n)$ of the best known sequential algorithm for the same problem instance, i.e.,

$$w_{\text{seq}}(n) = o(w(n)). \tag{2}$$

Therefore, for applications for which (2) holds, it is of interest to ask the symmetric question, which is the maximum number of processors that can be deployed while maintaining the same amount of work $w_{\text{seq}}(n)$? We expect, of course, that, if (2) holds, the maximum number of deployable processors will be $o(p(n))$, where $p(n)$ is the number of processors used by the fastest algorithm. It is appropriate to call such an algorithmic technique "work-preserving speed-up," although it has been previously referred to as "supereffective slow-down," [PP92]

to contrast it against the "effective" slow-down achievable with Brent's simulation (which preserves but does not reduce the work).

As we shall show, for several important computational problems, the attainable work-preserving speed-ups, although not achieving the maximum, are still very substantial. The technique has been implicitly used in [BP90] for polynomial division and in [BP93] for computing modulo x^n the square root (and similarly the mth root for any integer $m \geq 2$) of a polynomial $p(x)$.

As we shall illustrate below, the technique involves the careful interplay of two algorithms for the given problem, which have markedly distinct features. Such a general approach—the interplay of two algorithms to achieve performance improvements—is by no means new, and has appeared in various forms in the technical literature. The earliest instance was the adoption of an "inner/outer" algorithmic structure, frequently in the context of very-large-scale-integration-circuit implementation. For example, for matrix multiplication, the outer structure is systolic (slow) and the inner structure is based on the three-dimensional mesh-of-trees (fast), i.e., a systolic algorithm involving matrix blocks, which are in turn multiplied with the fastest algorithm. Such approach provided AT^2-optimal realizations over the entire spectrum of computation times [PV80].

Another, more sophisticated instance of the methodology is the "accelerating cascade technique" proposed in [CV86]. Here two parallel algorithms are cascaded: the first is (work-) optimal but slow, the second is fast but not optimal. The objective is to use the first algorithm to reduce the problem size, so that the second algorithm can complete the task using no more than allowed for optimality. By this careful interplay, optimal work and time can be achieved for list-ranking [CV86]. The same strategy was applied in [SV81] to obtain a work/time-optimal algorithm for the maximum problem on a CRCW-PRAM. Related results on parallel graph computations are reported in [UY91], [S91a], and [S91b].

Our present approach, although belonging to this general methodology, does not adopt the slow-optimal/fast-suboptimal strategy. In a way, it is more akin to the outer/inner structure approach. Specifically, it adopts as an outer structure a sequential recursive algorithm, which typically reduces a problem of size s to a problem of size $s - 1$. If such an algorithm exists, then, when several processors are available, the idea is to make them act on larger mathematical objects (rather than single entries), to which a fast algorithm is applicable (thus providing the inner structure). The objective is to carefully balance the respective works of the inner and outer structures. In contrast to "divide-and-conquer," and, with terminological analogy, this technique could be called "shrink-and-conquer."

We want to extend the outlined approach to some fundamental computations with matrices with further application to computational problems on graphs (represented as matrices). Our study shows that the work-preserving speed-up is possible for numerous parallel matrix computations that effectively extend the solution of a problem of size s to one of size ks for any positive integers s and k.

We demonstrate our techniques for only a few matrix computational problems, in particular, for the inversion and quasi-inversion of matrices and for solving systems of linear equations. These problems are fundamental and have numerous applications to computations for linear algebra (matrix inversion), to path algebras in graphs and digraphs (quasi-inversion), and to various areas of symbolic computations (structured linear systems).

We believe that some of our algorithms have practical value. In particular, for computations in numerical linear algebra, such as solving triangular linear systems of equations (see §5, which can be read independently of §§3 and 4 and from the second half of §2), these algorithms run faster than the known customary algorithms, even when the number of processors is reasonably bounded. Furthermore, our algorithms intensively use block matrix computations, which can be effectively implemented on loosely coupled multiprocessors.

We organize our paper as follows. After some definitions and preliminaries in §§2 and 3, we show how to apply a work-preserving speed-up to quasi-inversion of matrices over the semirings and to their inversion over the fields. In §5, we treat the inversion of triangular matrices and the solution of triangular linear systems of equations. In §6 we consider the case of Toeplitz-like input matrices, having further extension to polynomial computations. Appendix A contains some auxiliary material on basic properties of Toeplitz-like matrices.

2. Definitions and preliminary results. To have a machine-independent high-level presentation, we will assume the customary PRAM models of parallel computing [KR90], [V90]. Under such models, each processor in each step performs at most one arithmetic operation. We shall adopt the "work-time framework" for evaluating algorithmic performance (see [J92]) and use the notation $O_A(t, w)$ for the pair $O(t)$ and $O(w)$, which are, respectively, simultaneous asymptotic bounds on the numbers of arithmetic steps (arithmetic parallel time) and of operations (work). It is well known that if we choose $p \leq w/t$ processors to execute the algorithm, then the number of attainable arithmetic steps is $O(w/p)$. Our algorithms do not actually depend upon choosing the PRAM model; in fact each of them consists of a simple sequence of blocks, each routinely implementable on fixed interconnections. We shall say that a parallel algorithm, with work w, is efficient if $w = \Theta(w_{\text{seq}})$, where, as above, w_{seq} is the work of the fastest known sequential algorithm for the same problem instance.

We will use some known facts about computations with matrices over the fields $\boldsymbol{F}$ and semirings $\boldsymbol{R}$ (also called dioids and path algebras). Over any field or semiring, we may compute an $m \times n$ by $n \times p$ matrix product within the following cost bounds:

$$O_A(\log n, M(m, n, p)), \tag{3}$$

where

$$M(m, n, p) \leq mnp, M(n) = M(n, n, n) \leq n^3. \tag{4}$$

Remark 1. Over the fields (and rings), we may theoretically improve this bound at least to

$$M(m, n, p) \leq mnp/(\min\{m, n, p\})^{3-\omega},$$
$$M(n) = M(n, n, n) \leq n^{\omega},$$

with $\omega < 2.376$.

The algorithms supporting the bound $\omega < 2.376$ [CW90], [P87], [BP94] are not practical, unlike some algorithms supporting the bounds $\omega < 2.81$ and even $\omega < 2.78$, which have, or promise to have, some limited practical value [GL89], [LPS92]. In the remainder of this paper, however, we shall assume that both inequalities (4) hold as equalities.

For the randomized inversion of an $n \times n$ matrix over a field F, we have the following estimates [P91], [P92], [KP91], [KP92], [BP94]:

$$O_A(\gamma(n, q) \log^2 n, M(n) \log n), \tag{5}$$

where $M(n)$ is defined by (3), (4), q is the characteristic of F, and

$$\gamma(n, q) = \begin{cases} \lceil \log n / \log q \rceil & \text{if } q > 0, \\ 1 & \text{if } q = 0. \end{cases} \tag{6}$$

For the same computational problem, we have the following deterministic estimates:

$$O_A(\log^2 n, n^{\beta} M(n)), \tag{7}$$

where $\beta \leq 1$ over any field F [Be84], [Ch85]; however, if F has characteristic $q = 0$ or $q > n$, then $\beta < 1/2$ [Cs76], [PS78], [GP89].

Over the semirings with additive operator $\oplus$, instead of matrix inverses, we compute the quasi-inverses (A^* denoting the quasi-inverse of A), as follows:

$$A^* = \lim_{h\to\infty} A^{(h)}, \quad A^{(0)} = I, \quad A^{(h)} = A^{(h-1)} \oplus A^h = I \oplus A \oplus \ldots \oplus A^h, \quad h = 1, 2, \ldots,$$

where I denotes the identity matrix in the semiring [Ca79], [PR89], [P93]. In many applications, we deal with semirings where

$$A^* = A^{(n-1)} = \prod_{j=0}^{\lceil \log(n-1)\rceil - 1} (I + A^{2^j}), \tag{8}$$

for any $n \times n$ matrix A. Then the bounds (3), (4) for $M(n) = n^3$ are extended to the evaluation of A^*; these bounds take the form

$$O_A(\log^2 n, n^3 \log n), \tag{9}$$

thus yielding a parallel algorithm that is fast but not work-optimal, since in this case $w_{\text{seq}} = O(n^3)$ (see, e.g., [AHU74]).

3. A factorization of a matrix and its (quasi-) inverse. Hereafter, I and 0 denote the identity and the null matrices of appropriate sizes, respectively. A^T and A^H denote the transpose and the Hermitian transpose of a matrix A, respectively (see [GL89] for these standard definitions).

We will represent an $n \times n$ matrix A as a 2×2 block matrix and recall the following factorizations of A and A^{-1} [AHU74]:

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}, \qquad A = \begin{bmatrix} I & 0 \\ A_{21}A_{11}^{-1} & I \end{bmatrix} \begin{bmatrix} A_{11} & 0 \\ 0 & S \end{bmatrix} \begin{bmatrix} I & A_{11}^{-1}A_{12} \\ 0 & I \end{bmatrix}, \tag{10}$$

$$A^{-1} = \begin{bmatrix} I & -A_{11}^{-1}A_{12} \\ 0 & I \end{bmatrix} \begin{bmatrix} A_{11}^{-1} & 0 \\ 0 & S^{-1} \end{bmatrix} \begin{bmatrix} I & 0 \\ -A_{21}A_{11}^{-1} & I \end{bmatrix}, \tag{11}$$

$$S = A_{22} - A_{21}A_{11}^{-1}A_{12}. \tag{12}$$

Here, A_{11} is an $m \times m$ matrix, $m \leq n$, and we assume nonsingularity of A, A_{11}, and S. (S is known as the Schur complement of A_{11} in A). For a nonsingular matrix A, we may ensure nonsingularity of A_{11} and S, with a high probability, by means of a simple preconditioning of A, for instance, by the transition to the matrix UAL^T, where U and L^T are unit lower triangular Toeplitz matrices with ones on their diagonals and with other entries of their first columns chosen at random (compare [KP91], [BP94]). Over the real or complex fields (and over their subfields) we may deterministically ensure the nonsingularity of A_{11} and S by shifting from the original matrix A to the positive definite product A^HA.

No extension of (10) to semirings is known, but (11) and (12) are extended as follows ([AHU74, p. 205], [PR89], [P93]):

$$A^* = \begin{bmatrix} I & A_{11}^*A_{12} \\ 0 & I \end{bmatrix} \begin{bmatrix} A_{11}^* & 0 \\ 0 & S^* \end{bmatrix} \begin{bmatrix} I & 0 \\ A_{21}A_{11}^* & I \end{bmatrix}, \tag{13}$$

$$S = A_{22} + A_{21}A_{11}^*A_{12}. \tag{14}$$

4. Work-preserving speed-up for matrix inversion and quasi-inversion. The fastest parallel algorithms that support the bounds (5), (7), and (9) are not efficient since Gaussian elimination over any field and its extensions to semirings [Ca79], [PR89] support the bounds $O_A(n, n^3)$. Next, we will demonstrate that matrix inversion and quasi-inversion lend themselves to work-preserving speed-ups.

Let us consider a semiring such that property (8) holds for $n \times n$ matrices for any n (this implies that (9) is applicable). Let A be an $n \times n$ matrix. As in the preceding section, we partition A as a 2×2 block matrix, where the upper-left block A_{11} is an $h \times h$ matrix (h to be determined). The technique based on (13) and (14) reduces the computation of A^* to the computation of A_{11}^*, S^*, and six multiplications of pairs of rectangular matrices, whose dimensions never exceed n and one of which has at least one dimension equal to h. Therefore, these multiplications have global performance $O_A(\log n, n^2 h)$ (see (3) and (4)). If we explicitly compute A_{11}^* and all matrix products we reduce the original problem to the (recursive) computation of S^*. By (8), A_{11}^* is computed at the cost $O_A(\log^2 h, h^3 \log h)$. Thus, the nonrecursive part of the computation satisfies the cost bound

$$O_A(\max(\log^2 h, \log n), \max(h^3 \log h, n^2 h)). \tag{15}$$

It follows that the performance of the global computation is obtained by multiplying by n/h each of the terms of (15). Next, we impose the condition $h^3 \log h \leq n^2 h$, which yields $\max(h^3 \log h, n^2 h) = n^2 h$. If, specifically, we select $h = n/(\log n)^{1/2}$ (which satisfies the preceding condition), then we have $(n/h)\max(\log^2 h, \log n) = (n/(n/(\log n)^{1/2}))\log^2 n = \log^{5/2} n$ and $(n/h)\max(h^3 \log h, n^2 h) = (n/h)n^2 h = n^3$, so that we arrive at the solution A^* of the original problem with the following (work-optimal) performance:

$$O_A((\log n)^{5/2}, n^3). \tag{16}$$

The parallel algorithm that supports (15) is efficient, according to our definition, and still quite fast, for it fails to attain just by a factor $O((\log n)^{1/2})$ the time of the fastest known algorithm for this problem (see (9)).

Over the fields of characteristic q, a similar approach based on (5), (10)–(12) [where we choose $h = n/(\log n)^{1/2}$ (see (5)) as the dimension of block A_{11} and where we precondition A to avoid the singularities] yields the bounds

$$O_A((\log n)^{5/2}\gamma(n, q), n^3)$$

on the randomized complexity of the inversion of an $n \times n$ matrix.

When we operate over the fields of characteristic 0, we deterministically ensure nonsingularity in all such recursive computations by using $A^H A$ in lieu of A (since A^{-1} is easily obtained through the identity $A^{-1} = (A^H A)^{-1} A^H$) (compare [BP94]). Then we may also apply (7) (in this case with $\beta = 1/2$), and (10)–(12). If we choose $h = n^{4/5}$, the same techniques enable us to compute the inverse A^{-1} of an $n \times n$ nonsingular matrix A with the performance

$$O_A(n^{1/5} \log^2 n, n^3). \tag{17}$$

Note that (7), and consequently (17), are deterministic bounds. Again, we obtain an efficient algorithm, although in this case the speed-up falls substantially short of the maximum (the attainable time is $n^{1/5} \log^2 n$, rather than $\log^2 n$).

Remark 2. The latter estimate relies on (7) with $\beta = 1/2$. For $\beta < 1/2$, we may decrease the time bound, preserving the work n^3.

As an exercise, the reader may work out the extension of (15) and (16) based on the estimates of Remark 1. Then the asymptotic estimate for the work will decrease but will not reach the bound h^{ω} since (according to equations (10)–(12)) this involves rectangular matrix multiplication, which generally is not asymptotically as fast as square matrix multiplication (refer to Remark 1).

5. Triangular matrix inversion and solution of triangular linear systems. Let A be an $n \times n$ nonsingular triangular matrix. Then we may improve the estimates of the previous section for the inversion of A as follows [BM75, p. 146]:

$$O_A(\log^2 n, n^3). \tag{18}$$

Indeed, represent A and A^{-1} as 2×2 block matrices

$$A = \begin{bmatrix} A_{11} & 0 \\ A_{21} & A_{22} \end{bmatrix}, \qquad A^{-1} = \begin{bmatrix} A_{11}^{-1} & 0 \\ -A_{22}^{-1}A_{12}A_{11}^{-1} & A_{22}^{-1} \end{bmatrix}. \tag{19}$$

Assume for convenience that $n = 2^k$, for an integer k, and that A_{11} is an $(n/2) \times (n/2)$ block and note that (19) reduces the inversion of A to the concurrent inversions of the half-size matrices A_{11} and A_{22} and to two subsequent multiplications of $(n/2) \times (n/2)$ matrices. Recursively apply the same observations to invert A_{11} and A_{22}. Using (3), (4), and Brent's principle, we obtain a fast and efficient $O_A(\log^2 n, n^3)$ algorithm.

We may now use this algorithm to achieve a work-preserving speed-up for the solution of the triangular linear systems

$$A\mathbf{x} = \mathbf{b}, \tag{20}$$

since $\mathbf{x} = A^{-1}\mathbf{b}$. Note that the computation of A^{-1} is not required to obtain $\mathbf{x}$; if we first compute A^{-1} to subsequently obtain $\mathbf{x}$, the resulting algorithm is not efficient, since the simple (forward or back) substitution algorithm yields the bounds $O_A(n, n^2)$. We next explore whether a work-preserving speed-up is achievable for this problem.

We assume now that A_{11} is an $h \times h$ matrix (again, h is a parameter to be selected) and let $\mathbf{b}^T = [\mathbf{b}_1^T, \mathbf{b}_2^T]$, where $\mathbf{b}_1$ is a vector of dimension h. From (19) we deduce that

$$\mathbf{x} = A^{-1}\mathbf{b} = \begin{bmatrix} \mathbf{e} \\ A_{22}^{-1}\mathbf{f} \end{bmatrix}, \quad \mathbf{e} = A_{11}^{-1}\mathbf{b}_1, \quad \mathbf{f} = -A_{12}\mathbf{e} + \mathbf{b}_2. \tag{21}$$

These relations suggest to compute $\mathbf{e}$ at the cost $O_A(\log^2 h, h^3)$ (see (18)) and subsequently to compute $\mathbf{f}$ at the cost $O_A(\log h, nh)$. By so doing, we have reduced the original problem to the problem of computing $A_{22}^{-1}\mathbf{f}$, that is, to solving the triangular linear system $A_{22}\mathbf{y} = \mathbf{f}$ of $n-h$ equations. Iterating this approach, we could successively reduce this system to triangular systems of $n-2h, n-3h, \ldots$ equations, and thereby solve the original system (20). However, to avoid the inefficiency deriving from the imbalance between the computation times of $\mathbf{e}$ and $\mathbf{f}$ given above ($O(\log^2 h)$ vs. $O(\log h)$), we propose the following more efficient strategy. The computation is carried out as a sequence of $n/(h \log h)$ major iterations, each consisting of $\log h$ minor iterations, essentially as described above, the only difference being that the inversions of the $\log h$ diagonal blocks are carried out, concurrently, at the beginning of the major iterations, at the cost $O_A(\log^2 h, h^3 \log h)$. It follows that the overall cost is

$$O_A\left(\frac{n}{h \log h} \log^2 h, \frac{n}{h \log h} \max\left(h^3 \log h, nh \log h\right)\right),$$

which leads to selecting $h = n^{1/2}$ and to the estimate

$$O_A(n^{1/2}\log n, n^2). \tag{22}$$

Note that the algorithm is efficient (the work equals n^2), but we do not know how to avoid inefficiency if t is polylogarithmic in n or even if $t = n^{1/4}$ (for example).

Remark 3. We may obtain (theoretical) asymptotic improvements of the bounds (22) by applying the estimates of Remark 1. Then we just need to set $h = n^{1/(\omega-1)}$ and arrive at the bounds $O_A(n^\alpha \log^2 n, n^2)$, $\alpha = 1 - 1(\omega - 1)$, so that $\omega < 2.378$ implies that $\alpha < 0.28$. Clearly, a polylogarithmic time bound is achieved only for $\alpha = 0$ implied by the value $\omega = 2$.

6. Solving structured linear systems and further applications. In this section we will refer to computations over the complex field of constants, so that fast Fourier transform (FFT) on m points can be performed at the cost $O_A(\log m, m \log m)$. We will show a work-preserving speed-up for solving a linear system (20) in the special case where A denotes an $n \times n$ Toeplitz or Toeplitz-like matrix represented by its displacement generator of length $O(1)$.

We refer to Appendix A for the definition and basic properties of Toeplitz-like matrices and their displacement generators. By using these representations and their properties, one may devise two algorithms that compute a short displacement generator of the inverse of a nonsingular $n \times n$ Toeplitz-like matrix A at the cost bounded by $O_A(n, n\log^2 n)$, [BA80], [M80], and $O_A(\log^2 n, n^2 \log n)$ [P92], respectively.

Then, after this preprocessing stage and independent of the vector $\mathbf{b}$, the solution $\mathbf{x} = A^{-1}\mathbf{b}$ to the linear system $A\mathbf{x} = \mathbf{b}$ can be immediately computed at the cost $O_A(\log n, n \log n)$. Adopting the latter of the two cited algorithms, one should proceed as in §4, noting, however, that the matrix $A^H A$, as well as the matrices A_{11}, A_{11}^{-1}, A_{12}, A_{21}, A_{22}, and S of (10)–(12), are now Toeplitz-like matrices, to be represented in terms of their displacement generators of lengths $O(1)$. Much less computational work is involved when operating with such matrices than when dealing with general matrices as in §4.

In this way, one will finally obtain an algorithm that computes $A^{-1}\mathbf{b}$ within the bounds

$$O_A(n^{1-a}\log^2 n, n^{1+a}\log n), \tag{23}$$

for any a, $1/2 \le a \le 1$. For instance, (23) turns into $O_A(n^{1/2}\log^2 n, n^{3/2}\log n)$ for $a = 1/2$. To achieve this result, for a given a, one should choose as $n^a \times n^a$ the size of the leading block A_{11} in (10)–(12), thereby obtaining a Schur complement S of A_{11} in A of size $(n - n^a) \times (n - n^a)$ at the cost $O_A(\log^2 n, n^{2a}\log n)$, and, similarly, for each of the subsequent n^{1-a} iterations. Note, however, that one never explicitly computes the n^2 entries of A^{-1} (of which $(n+1)n/2$ may be distinct even when A is a Toeplitz matrix), but only recursively expresses the displacement generators of A^{-1} via those of A_{11}^{-1} and S^{-1}; furthermore, S^{-1} is a trailing principal submatrix of A^{-1}, and similarly are the Schur complements in S (and in its trailing blocks), arising in the outlined process of the recursive factorization of A^{-1}.

These bounds can be immediately extended to the solution of linear systems with $n \times n$ Hankel-like, Hilbert-like, and Vandermonde-like matrices (see [P90]), to the evaluation of greatest common divisor and least common multiple of two polynomials of degrees $O(n)$ (see [P92]), and to several other fundamental computations with matrices and polynomials (see [BP94]).

Appendix A. Some basic properties of Toeplitz and Toeplitz-like matrices. In this appendix we recall some basic properties of Toeplitz and Toeplitz-like matrices (compare [KKM79], [CKL87], [BP94], [P90], [P92], [P92a]).

The $n \times n$ matrix $Z = [z_{i,j}]$, which is 0 everywhere except for all 1's on the first subdiagonal, is called the $n \times n$ lower shift (displacement) matrix. (Note that $Z^n = 0$.) Z

generates the algebra of $n \times n$ lower triangular Toeplitz matrices, $\{\sum_{i=0}^{n-1} a_i Z^i = L(\mathbf{a})\}$, and similarly, Z^T generates the algebra of $n \times n$ upper triangular Toeplitz matrices,

$$\left\{ L^T(\mathbf{b}) = \sum_{i=0}^{n-1} b_i (Z^T)^i \right\},$$

where $\mathbf{a} = (a_0, \ldots, a_{n-1})^T$ and $\mathbf{b} = (b_0, \ldots, b_{n-1})^T$. An $n \times n$ Toeplitz matrix $T = [t_{i,j}]$ is a matrix representable as the sum $L(\mathbf{a}) + L^T(\mathbf{b})$ for any fixed pair of vectors $\mathbf{a}$ and $\mathbf{b}$. (Note that $t_{i,j} = t_{i+k,j+k}$ for any triple i, j, k, where $i, j, i+k, j+k$ are within the range $1, 2, \ldots, n$.) More generally, a pair $G = [\mathbf{g}_1, \ldots, \mathbf{g}_d]$, $H = [\mathbf{h}_1, \ldots, \mathbf{h}_d]$ of $n \times d$ matrices generates the matrices

$$A = \sum_{i=1}^{d} L(\mathbf{g}_i) L^T(\mathbf{h}_i), \tag{A.1}$$

$$B = \sum_{i=1}^{d} L^T(\mathbf{g}_i) L(\mathbf{h}_i). \tag{A.2}$$

The pair (G, H) is called either the (ZZ^T)-displacement generator of length d for A or the $(Z^T Z)$-displacement generator of length d for B. (Note the simple transition from (A.1), (A.2) to the displacement generators for A^H, B^H). For a given A, the smallest possible length d^* of its displacement generators is called the displacement rank of A.

THEOREM A.1 [KKM79]. *Equation* (A.1) *holds if and only if* $A - ZAZ^T = GH^T$; (A.2) *holds if and only if* $B - Z^T BZ = GH^T$.

By Theorem A.1, matrices can be represented by their displacement generators, which reduces the storage requirements if the generators are short (i.e., of small length). Moreover, recalling that multiplication of a Toeplitz matrix by a vector reduces to polynomial multiplication (convolution of vectors) and can be performed at the cost $O_A(\log n, n \log n)$, one can multiply either A or B, given by (A.1) and (A.2), by a vector at the cost $O_A(\log n, dn \log n)$ (since each such multiplication is resolved into $2d$ multiplications of a Toeplitz matrix by a vector). If $d = O(1)$, then A and B are called "Toeplitz-like."

The sum, difference, and product of two Toeplitz-like matrices U and V are Toeplitz-like matrices, whose displacement generators are explicitly expressed via the homologous generators of U and V. Moreover, similar properties hold for all the leading and trailing principal submatrices of U and V. Finally, if $UV = I$, then the displacement rank of U equals the displacement rank of V [KKM79], and for a Toeplitz-like matrix A represented with its displacement generator (G, H) according to (A.1), one may effectively compute its shortest ZZ^T- and $Z^T Z$-displacement generators, and similarly for the matrix B represented according to (A.2) [BA80], [P92], [P93a].

Acknowledgment. The authors wish to thank David Eppstein for his interest in the subject of this paper and for his useful comments.

REFERENCES

[AHU74] V. AHO, J. E. HOPCROFT, and J. D. ULLMAN, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, 1974.

[BA80] R. R. BITMEAD AND B. D. O. ANDERSON, *Asymptotically fast solution of Toeplitz and related systems of linear equations*, Linear Algebra Appl., 41 (1980), pp. 111–130.

[Be84] S. BERKOWITZ, *On computing the determinant in small parallel time using a small number of processors*, Inform. Process. Lett., 18 (1984), pp. 147–150.

[Br74] R. P. BRENT, *The parallel evaluation of general arithmetic expressions*, J. Assoc. Comput. Mach., 21 (1974), pp. 201–206.

[BM75] A. BORODIN AND I. MUNRO, *The Computational Complexity of Algebraic and Numeric Problems*, American Elsevier, New York, 1975.

[BP90] D. BINI AND V. Y. PAN, *Parallel polynomial computations by recursive processes*, in Proc. ACM SIGSAM Intern. Symp. on Symb. and Alg. Comp. (ISSAC-90), ACM Press, NY, 1990, p. 294.

[BP93] ———, *Improved parallel polynomial division*, SIAM J. Comput., 22 (1993), pp. 617–626.

[BP94] ———, *Polynomial and Matrix Computations* I, Birkhauser, Boston, 1994.

[Ca79] B. A. CARRÉ, *Graphs and Networks*, The Claredon Press, Oxford, 1979.

[Ch85] A. L. CHISTOV, *Fast parallel calculation of the rank of matrices over a field of arbitrary characteristics*, in Proc. FCT 85, Springer Lecture Notes in Computer Science, 199, Springer-Verlag, Boston, 1985, pp. 63–69.

[Cs76] L. CSANKY, *Fast parallel matrix inversion algorithms*, SIAM J. Comput., 5 (1976), pp. 618–623.

[CKL87] J. CHUN, T. KAILATH, AND H. LEV-ARI, *Fast parallel algorithm for QR-factorization of structured matrices*, SIAM J. Sci. Statist. Comput., 8 (1987), pp. 899–913.

[CV86] R. COLE AND U. VISHKIN, *Deterministic coin tossing with applications to optimal parallel list ranking*, Inform. and Control, 70 (1986), pp. 32–53.

[CW90] D. COPPERSMITH AND S. WINOGRAD, *Matrix multiplication via arithmetic progression*, J. Symbolic Comput., 9 (1990), pp. 251–280.

[GL89] G. H. GOLUB AND C. F. VAN LOAN, *Matrix Computations*, Johns Hopkins University Press, Baltimore, 1989.

[GP89] Z. GALIL AND V. Y. PAN, *Parallel evaluation of the determinant and of the inverse of a matrix*, Inform. Proc. Lett., 30 (1989), pp. 41–45.

[J92] J. JÀJÀ, *An Introduction to Parallel Algorithms*, Addison-Wesley, Reading, MA, 1992.

[KKM79] T. KAILATH, S.-Y. KUNG, AND M. MORF, *Displacement rank of matrices and linear equations*, J. Math. Anal. Appl., 68 (1979), pp. 395–407.

[KP91] E. KALTOFEN AND V. Y. PAN, *Processor efficient parallel solution of linear systems over an abstract field*, in Proc. 3rd Annual ACM Symp. on Parallel Algorithms and Architecture, Hilton Head, SC, ACM Press, New York, 1991, pp. 180–191.

[KP92] ———, *Parallel and processor efficient solution of linear systems* II: *The general case*, in Proc. 33rd IEEE Symposium on Foundations of Computer Science, Pittsburgh, PA, IEEE Computer Society Press, Los Alamitos, CA, 1992, pp. 714–723.

[KR90] R. KARP AND V. RAMACHANDRAN, *A survey of parallel algorithms for shared memory machines*, in Handbook of Theoretical Computer Science, North-Holland, Amsterdam, 1990, pp. 869–941.

[LPS92] J. LADERMAN, V. Y. PAN, AND X.-H. SHA, *On practical acceleration of matrix multiplication*, Linear Algebra Appl., 162–164 (1992), pp. 557–588.

[M80] M. MORF, *Doubling algorithms for Toeplitz and related equations*, in Proc. IEEE Intern. Conference on ASSP, IEEE Computer Society Press, 1980, pp. 954–959.

[P87] V. Y. PAN, *Complexity of parallel matrix computations*, Theoret. Comput. Sci., 54 (1987), pp. 65–85.

[P90] ———, *On computations with dense structured matrices*, Math. of Comput., 55 (1990), pp. 179–190.

[P91] ———, *Complexity of Algorithms for Linear Systems of Equations*, in Computer Algorithms for Solving Linear Algebraic Equations (The State of the Art), edited by E. Spedicato, NATO ASI Series, Series F: Computer and Systems Sciences, Springer-Verlag, Berlin, 77, 1991, pp. 27–56.

[P92] ———, *Parameterization of Newton's iteration for computations with structured matrices and applications*, Comput. Math. Appl., 24 (1992), pp. 61–75.

[P92a] ———, *Complexity of computations with matrices and polynomials*, SIAM Rev., 34 (1992), pp. 225–262.

[P93] ———, *Parallel solution of sparse linear and path systems*, in Synthesis of Parallel Algorithms, J. Reif, ed., Morgan Kaufmann, San Mateo, CA, 1993, pp. 621–678.

[P93a] ———, *Decreasing the displacement rank of a matrix*, SIAM J. Matrix Anal. Appl., 14 (1993), pp. 118–121.

[PP92] V. Y. PAN AND F. P. PREPARATA, *Supereffective slow-down of parallel computations*, in Proc. 4th Annual ACM Symp. on Parallel Algorithms and Architectures, San Diego, CA, ACM Press, New York, 1992, pp. 402–409.

[PR89] V. Y. PAN AND J. REIF, *Fast and efficient solution of path algebra problems*, J. Comput. System Sci., 38 (1989), pp. 494–510.

[PS78] F. P. PREPARATA AND D. V. SARWATE, *An improved parallel processor bound in fast matrix inversion*, Inform. Proc. Lett. 7 (1978), pp. 148–149.

[PV80] F. P. PREPARATA AND J. VUILLEMIN, *Area-time optimal VLSI networks for multiplying matrices*, Inform. Process. Lett., 11 (1980), pp. 77–80.

[S91a] T. SPENCER, *Time-work tradeoffs for parallel graph algorithms*, in Proc. 2nd ACM-SIAM Symp. on Discrete Algorithms, San Francisco, CA, 1991, ACM Press, New York, and Society for Industrial and Applied Mathematics, Philadelphia, PA, pp. 425–432.

[S91b] ———, *More time-work tradeoffs for parallel graph algorithms*, in Proc. 3rd Annual ACM Symp. on Parallel Algorithms and Architectures, Hilton Head, SC, 1991, ACM Press, New York, pp. 81–93.

[SV81] Y. SHILOACH AND U. VISHKIN, *Finding the maximum, merging, and sorting in a parallel computation model*, J. Algorithms, 2 (1981), pp. 88–102.

[V90] L VALIANT, *General purpose parallel architectures*, in Handbook of Theoretical Computer Science, J. van Leeuwen, ed., North Holland, Amsterdam, (1990), pp. 943–971.

[UY91] J. ULLMAN AND M. YANNAKAKIS, *High-probability parallel transitive closure algorithms*, SIAM J. Comput., 20 (1991), pp. 100–125.