Parallel Algorithms for Linear Recurrences and for Matrix and Vector Computations

by
Antoine Marcel Atinkpahoun

A dissertation submitted to the Graduate Faculty in Computer Science in partial fulfillments for the degree of Doctor of Philosophy, The City University of New York

1995

This manuscript has been read and accepted for the Graduate Faculty in Computer Science in satisfaction of the dissertation requirement for the degree of Doctor of Philosophy.

4-18-95 — Date
Prof. Victor Y. Pan — Victor Pan
Chair of Examining Committee

4-18-95 — Date
Prof. Stanley Habib — Prof S. Habib
Executive Officer

Prof. Statis Zachos
Prof. Gordon Silverman
Prof. Thadeus Strzemecki
Supervisory Committee

THE CITY UNIVERSITY OF NEW YORK

Abstract

Parallel Algorithms for Linear Recurrences

and for Matrix and Vector Computations.

by

Antoine Marcel Atinkpahoun

Adviser: Professor Victor Y. Pan

The purposes of this study were:

1. demonstrate close correlations of banded matrix computations and computation of linear recurrence sequences, and also correlation among several known divide-and-conquer algorithms, that is, nested dissection, block cyclic reduction and prefix computation,

2. present some recent algorithms for computations with banded matrices and

3. modify the known *FFT based* algorithm for multiplying two polynomials or, equivalently, for computing the convolution of two vectors (of their coefficients), so as to decrease the requirement to the storage space used at the expense of a relatively small slowdown of the computation.

ACKNOWLEDGEMENTS

I wish to thank Professor Victor Y. Pan, my advisor, whose comments and guidance about the preliminary versions of this thesis have been so helpful. I first met with Professor Victor Y. Pan on Spring 1989 when I took his class on "Matrix and Polynomial Computations". Since then, he has been my advisor and my mentor. Matrix and polynomial computations are fundamental to the theory and practice of computation. Professor Victor Y. Pan taught me some general techniques for the design of efficient parallel algorithms for algebraic and numerical computations and gradually brought me to the frontiers of current research. He helped me recognize the links between matrix and polynomial computations as well as between numerical and algebraic approaches to computation. He also made familiar with some prepublished results of research by himself and Dr. Albert Greenberg (Bell Laboratories, AT&T), which helped me in chapter 6 on polynomial multiplication.

I wish to thank Professor Charles R. Giardina who introduced me to the world of parallel digital signal processing. His book on "Unified Signal Algebra Approach" was a useful reference that helped me understand better the concept of "building blocks" that can be used to form useful operations.

I wish to thank the members of the examining committee for having accepted to read and then judge my work.

I also want to thank all the staff of the Computer Department at the Graduate Center (CUNY), especially Professors Stanley Habib, Robert A. Orchard, Rohit Parikh, Stathis Zachos, David Arnow, Eralp Akkoyunlu, Micheal Anshel, Kenneth McAloon and Carol Tretkoff whose teaching and

advices were very helpful to me and Mr. Joseph Driscoll for his patience and constant availability in assisting and helping students.

I gratefully acknowledge my wife, my two children Ariane and Senami, my brothers and sisters, and all my friends for their encouragement and constant support of my personal goals with regard to doctoral pursuits.

Antoine Marcel Atinkpahoun

Contents

List of Figures

List of Tables

Chapter 1

Introduction (Subjects and the Main Results)

"...This earth plane is neither the beginning nor the end of our existence. It is simply a step, a schoolroom.
In God's Plan no soul is alone. No soul is ever lost."
By Pat Rodegast and Judith Stanton.

In computational practice, it is usually desired to perform computations with the largest possible input and/or to decrease the computational time as much as possible. Because many numerical problems require a large number of arithmetic operations, many researchers focus their efforts on designing algorithms particularly effective for parallel computer architecture. Our objective in this thesis is the study of parallel algorithms and their computational complexity for some major computations with vectors, sparse matrices having special structure and polynomials. We include some recent algorithms obtained jointly with Victor Y. Pan and Isdor Sobze [see PSA]. These algorithms support the current estimates for parallel complexity of some problems of computations with banded matrices. We also demonstrate close correlations of banded matrix computations and computation of linear recurrence sequences, and also correlation among several known divide-and-conquer al-

gorithms, that is, nested dissection (ND), block cyclic reduction (BCR) and prefix computation.

In chapter 2 we will survey effective parallel algorithms for several computations that can be expressed as *linear recurrences*, such as polynomial evaluation and solving banded triangular linear systems. We organize chapter 2 as follows: after some background material and definitions (presented in sections 2.1-2.3 and including a description of the *PRAM* models of parallel computations), we show various close correlations between some computations with banded triangular matrices and linear recurrences and present a logarithmic-time parallel algorithm for solving first-order linear recurrence by using a linear number of operations in section 2.4. In sections 2.4.3 and 2.4.4, we show an effective divide-and-conquer parallel algorithm for linear recurrences (closely related to the well-known prefix-sum algorithm, which we recall in Appendix A). Due to the results of section 2.4, this algorithm also applies to banded triangular systems of equations. In section 2.5 we show an application of the techniques of supereffective slow-down of parallel algorithms to improving the processor efficiency of parallel inversion of a triangular matrix and parallel solution of a general triangular linear system of equations.

In chapter 3 we first consider sparse and symmetric linear systems of equations whose sparsity structure is defined by a given family of small separators for the associated graph. Such systems can be effectively solved by

means of the nested dissection algorithm (ND), which we recall in section 3.2. This algorithm is defined as symmetric Gaussian elimination with pivoting. The only crucial step is a choice of pivoting (also called elimination ordering). In section 3.4, we show a factorization of a h.p.d. block tridiagonal matrix based on computing its $s(n)$-separator tree. We also observe that banded linear systems constitute a subclass of sparse linear systems given with a family of small separators for the associated graph. Therefore, the nested dissection algorithm (ND) can be effectively applied to banded linear systems. In section 3.6 we show that, for an appropriate choice of separator family, the resulting solution algorithm, in the case of banded matrices, turns into a well-known block cyclic reduction algorithm, which we recall in the same section.

In chapter 4 we present some alternative algorithms for computations with banded matrices. Like BCR and ND, they run in poly-logarithmic parallel time and use the optimum or nearly optimum (linear) number of arithmetic operations. Unlike the block cyclic reduction (BCR) algorithm that generally only works over the field of constants of characteristic 0, these algorithms apply over any field of constants. Comparing these algorithms with the BCR algorithm, we observe some interesting trade-off between the total number of arithmetic operations involved and the memory space required by the algorithm. For a large class of input matrices (having nonsingular lower and/or upper diagonals), we also show a faster parallel algorithm, and we also

include effective parallel algorithms that efficiently compute the determinant and the rank of a banded matrix and solve consistent but singular banded linear systems.

In chapter 5 we extend the presented algorithms to effective parallel computation of the Krylov sequences of vectors (which we represent as Krylov matrices) in the case of triangular banded matrices, as well as any sparse matrix given with its small separator family.

In chapter 6, we shift our topic from computations with matrices to ones with vectors and polynomials. Then again, we encounter and study trade-off between the arithmetic time and storage space required for the latter computations. Specially, we modify the known *FFT based* algorithms for multiplying two polynomials or, equivalently, for convolution of the two vectors of their coefficients, so as to decrease the requirement to the storage space used at the expense of only a relatively small slowdown of the computation. A known asymptotically optimum parallel implementation of the fast Fourier transform *FFT* (in logarithmic time) is recalled in Appendix B.

Chapter 2

Parallel Matrix Computations

2.1 Background and Notation

2.1.1 Matrix Notation

Let R denote the set of real numbers. We denote the vector space of all $m \times n$ real matrices by $R^{m \times n}$:

$$A \in R^{m \times n} \iff A = (a_{ij}) = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix}, \qquad a_{ij} \in R.$$

$A^T = (a_{ji})$ will denote the transpose of A over the field C of complex numbers or any subfield of C.

The vector space of $m \times n$ complex matrices is designated by $C^{m \times n}$. The scaling, addition, and multiplication of complex matrices corresponds exactly to the real case.

For an $n \times n$ matrix $A = (a_{ij})$ its transpose is the $n \times n$ matrix $A^T = (a_{ji})$. If the computations are in the complex field or in a field of characteristic 0

that with every entry a_{ij} of A contains its complex conjugate $\bar{a}_{ij}$, then the Hermitian transpose of A is denoted A^* or A^H and is defined as $A^* = A^H = (\bar{a}_{ji})$. Hereafter, we will assume the availability of A^H by default, wherever we deal with the fields of characteristic 0, and will use the notation A^H, rather than A^*. We have $A^H = A^T$ if the matrix A is filled with real numbers.

DEFINITION **2.1.1** *A matrix A is* Hermitian *if $A^H = A$. A matrix is* Hermitian nonnegative definite (h.n.d.) *if it can be represented as $W^H W$, for some matrix W. A nonsingular h.n.d. matrix is called* Hermitian positive definite (h.p.d.). *In the real case* Hermitian *matrices are usually called symmetric and the abbreviations* h.n.d. *and* h.p.d. *can be replaced by* r.s.p.d.

2.1.2 Matrix Operations

The basic manipulations with matrices include transposition $R^{m\times n} \longrightarrow R^{n\times m}$,

$$C = A^T \implies c_{ij} = a_{ji},$$

addition $R^{m\times n} \times R^{m\times n} \longrightarrow R^{m\times n}$,

$$C = A + B \implies c_{ij} = a_{ij} + b_{ij},$$

scalar-by-matrix multiplication ($R \times R^{m\times n} \longrightarrow R^{m\times n}$),

$$C = \alpha A \implies c_{ij} = \alpha a_{ij},$$

and matrix-by-matrix multiplication ($R^{m\times r} \times R^{r\times n} \longrightarrow R^{m\times n}$),

$$C = AB, \qquad c_{ij} = \sum_{k=1}^{r} a_{ik} b_{kj},$$

which include matrix-by-vector and vector-by-matrix multiplication as two particular cases, where $m = 1$ or $n = 1$, respectively. These are some major building blocks of matrix computations.

2.1.3 Vector Notation

Let R^n denote the vector space of real $n-$dimensional vectors:

$$x \in R^n \iff x = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}, \qquad x_i \in R.$$

We refer to x_i as the ith component of x.

2.1.4 Permutation Matrices, Submatrices and Block Matrices

DEFINITION **2.1.2** **(Permutation matrix).** *An $n \times n$ matrix P is called a* permutation matrix, *if there exists a permutation*

$$\pi : (1, \cdots, n) \longrightarrow (1, \cdots, n)$$

such that $P\ v = (v_{\pi(1)}, \cdots, v_{\pi(n)})^T$ and $v^T P^T = (v_{\pi(1)}, \cdots, v_{\pi(n)})$ for any n-dimensional column vector $v = (v_1, \cdots, v_n)^T$.

DEFINITION **2.1.3** **(Submatrix).** *A $k \times l$ matrix formed by the intersection of the rows $i_1, \cdots, i_k$ and the columns $j_1, \cdots, j_l$ of a matrix W, for any fixed k-tuple $(i_1, \cdots, i_k)$ and any fixed l-tuple $(j_1, \cdots, j_l)$, is a* submatrix *of W.*

If $i_{s+1} = i_s + 1$, $j_{s+1} = j_s + 1$ for all s, the submatrix is called a block of W.

A $k \times k$ submatrix W, formed by rows and columns $i_1, \cdots, i_k$, is called principal.

We may partition both sets of rows and columns of an $m \times n$ matrix A to obtain its block representation:

$$A = \begin{bmatrix} A_{11} & \cdots & A_{1q} \\ \vdots & & \vdots \\ A_{p1} & \cdots & A_{pp} \end{bmatrix} \begin{matrix} \} \, m_1 \\ \\ \} \, m_p \end{matrix} .$$
$$\underbrace{\qquad}_{n_1} \quad \underbrace{\qquad}_{n_q}$$

Here, $m_1 + \cdots + m_p = m$, $n_1 + \cdots + n_q = n$, A_{ij} designates the (i,j)−th block of A , which is a submatrix of A (see Definition 2.1.3), of dimension $m_i \times n_j$, and we say that $A = (A_{ij})$ is a $p \times q$ block matrix.

2.1.5 Matrix Inversion and Nonsingularity. Solving a Linear System of Equations

The $n \times n$ *identity matrix* I_n is defined by the column partitioning

$$I_n = [e_1, \ldots, e_n] = \begin{bmatrix} 1 & \ldots & 0 \\ & \ddots & \\ 0 & & 1 \end{bmatrix},$$

where e_k is the kth coordinate vector of dimension n:

$$e_k^{(n)} = (\underbrace{0, \ldots, 0}_{k-1}, 1, \underbrace{0, \ldots, 0}_{n-k})^T.$$

We will frequently drop the subscripts or superscripts unless this causes any confusion and write I and e_k instead of I_n and $e_k^{(n)}$.

Note: $AI = A$ for any $m \times n$ matrix A.

DEFINITION **2.1.4** *If A and X are in $R^{n\times n}$ and satisfy $AX = I$, then X is called the* inverse *of A and is denoted by A^{-1}. If A^{-1} exists, then A is said to be* nonsingular. *Otherwise A is singular.*

Matrix inversion is closely related to the solution of the following computational problem:

PROBLEM **2.1.1** *$LIN \cdot SOLVE = LS(n,\ m)$: Given a nonsingular $n \times n$ matrix A and a vector b of dimension n, compute the solution $x = A^{-1}b$ to the linear system $Ax = b$.*

Indeed, given A^{-1} and b, we may compute the solution $x = A^{-1}b$ to the linear system $Ax = b$ by multiplying A^{-1} by b. Usually, this is not an effective approach to $LIN \cdot SOLVE$, since matrix inversion is equivalent to solving n linear systems with matrix A.

We will use some simple and well-known properties of matrix inverses.

- $A^{-1}A = I$,
- $(AB)^{-1} = B^{-1}A^{-1}$,
- $(A^{-1})^T = (A^T)^{-1} \equiv A^{-T}$,
- $B^{-1} = A^{-1} - B^{-1}(B - A)A^{-1}$.

DEFINITION **2.1.5** *(***Strong nonsingularity***). A matrix is* strongly nonsingular *if all its principal submatrices are nonsingular.*

PROPOSITION **2.1.1** *(see [GL, page 140]). In the field of complex numbers (and in any of its subfields), the matrices W^HW and $(W^HW)^{-1}$ are strongly nonsingular for any nonsingular matrix W.*

REMARK **2.1.1** *Due to Proposition 2.1.1, it is theoretically sufficient to develop nonsingular linear system solvers for the special case of h.p.d. linear systems.*

2.1.6 The Determinant

If $A = (a) \in R^{1\times 1}$, that is, if A is a 1×1 matrix, then its *determinant* is given by $det(A) = a$. The determinant of an $n \times n$ matrix A is defined in terms of the determinants of the $(n-1) \times (n-1)$ submatrices of A as follows:

$$det(A) = \sum_{j=1}^{n} (-1)^{j+1} a_{1j} det(A_{1j}),$$

where A_{1j} is an $(n-1)\times(n-1)$ submatrix of A obtained by deleting the first row and the jth column of A. Useful properties of the determinant include

- $det(AB) = det(A)det(B)$, $\quad A,\ B \in R^{n\times n}$,
- $det(A^T) = det(A)$, $\quad A \in R^{n\times n}$,
- $det(cA) = c^n det(A)$, $\quad c \in R,\ A \in R^{n\times n}$,
- $det(A) \neq 0 \Leftrightarrow$ *A is nonsingular*, $\quad A \in R^{n\times n}$.

2.1.7 Vector Norms

DEFINITION **2.1.6** *A* vector norm *on* R^n *is a function* $f : R^n \longrightarrow R$ *that satisfies the following properties:*

- $f(x) \geq 0, \qquad x \in R^n, \qquad (f(x) = 0 \; iff \; x = 0),$
- $f(x+y) \leq f(x) + f(y), \qquad x,\, y \in R^n,$
- $f(\alpha x) = |\alpha| f(x), \qquad \alpha \in R,\; x \in R^n.$

We denote such a function with a double bar notation: $f(x) = \|x\|$. Subscripts on the double bar are used to distinguish between various norms.

A useful class of vector norms are the p-norms defined by

$$\|x\|_p = (|x_1|^p + \cdots + |x_n|^p)^{\frac{1}{p}}, \quad p \geq 1 \tag{2.1}$$

Of these the 1, 2, *and* ∞ norms are the ones most frequently used:

- $\|x\|_1 = |x_1| + \cdots + |x_n|,$
- $\|x\|_2 = (|x_1|^2 + \cdots + |x_n|^2)^{\frac{1}{2}} = (x^T x)^{\frac{1}{2}},$
- $\|x\|_\infty = \max_{1 \leq p \leq n} |x_i|.$

A *unit vector* with respect to the norm $\|.\|$ is a vector x that satisfies $\|x\| = 1$.

2.1.8 Matrix Norms

The analysis of matrix algorithms frequently requires to use matrix norms:

DEFINITION **2.1.7** *The definition of matrix norms extends the definition of vector norms:*

- $f(A) \geq 0, \qquad A \in R^{m \times n}, \qquad (f(A) = 0 \ \ iff \ \ A = 0),$
- $f(A + B) \leq f(A) + f(B), \qquad A,\ B \in R^{m \times n},$
- $f(\alpha A) = |\alpha| f(A), \qquad \alpha \in R,\ A \in R^{m \times n}.$

The most frequently used matrix norms in numerical linear algebra are the Frobenius norm,

$$\|A\|_F = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} |a_{ij}|}, \tag{2.2}$$

and the p-norms (called associated or subordinate to the p-norms of vectors):

$$\|A\|_p = \sup_{x \neq 0} \frac{\|Ax\|_p}{\|x\|_p}. \tag{2.3}$$

The Frobenius and p-norms (especially for $p = 1, 2, \infty$) satisfy certain inequalities that are frequently used in the analysis of matrix computations. For $A \in R^{m \times n}$ and $B \in R^{n \times q}$ we have

$$\|A\ B\|_p \leq \|A\|_p \|B\|_p, \quad for \quad all \quad p.$$

For $A \in R^{m \times n}$ we have [GL, page 57]

$$\|A\|_2 \leq \|A\|_F \leq \sqrt{n}\|A\|_2, \tag{2.4}$$

$$\max_{i,j} |a_{ij}| \leq \|A\|_2 \leq \sqrt{mn} \max_{i,j} |a_{ij}|, \tag{2.5}$$

$$\|A\|_1 = \max_{1 \leq j \leq n} \sum_{i=1}^{m} |a_{ij}|, \tag{2.6}$$

$$\|A\|_\infty = \max_{1 \leq i \leq m} \sum_{j=1}^{n} |a_{ij}|, \tag{2.7}$$

$$\frac{1}{\sqrt{n}}\|A\|_\infty \leq \|A\|_2 \leq \sqrt{m}\|A\|_\infty, \tag{2.8}$$

$$\frac{1}{\sqrt{m}}\|A\|_1 \leq \|A\|_2 \leq \sqrt{n}\|A\|_1. \tag{2.9}$$

2.2 The *PRAM* Models of Parallel Computation

2.2.1 Introduction to the Deterministic and Randomized *PRAM* Models. NC and RNC Algorithms.

A commonly accepted model for designing and analyzing algorithms consists of a central processing unit with a random-access memory attached to it. The typical instruction set for this model includes reading from and writing into the memory, and basic logical and arithmetic operations. The success of this model is due to its simplicity and its ability to capture the performance of sequential algorithms on Von Neumann-type computers. Unfortunately, parallel computation suffers from the lack of such an accepted algorithm model. Among all parallel models of computation, our choice is the *PRAM* models,

in which each nonidle processor performs one arithmetic operation (compare [KR]). Under these customary models, we focus on the arithmetic computational complexity. This is idealization of the real present day computations, since we do not include the complexity of processor communication and synchronization. However, all our high level algorithms can be easily adjusted to their implementation under more realistic models of parallel computations. (We also refer the reader to [Val], [Val1] on the pratical importance of PRAM algorithms.)

Hereafter, we will assume a variant of *Brent's scheduling principle*, which we will call the B-principle and according to which a slowdown of parallel computations by $O(s)$ times suffices in order to decrease the number of processors from p to p/s provided that s and p/s are positive integers [Bre]. We will write $O_A(T)$ and $O_A(t,p)$ to denote the sequential and parallel arithmetic cost of performing an algorithm, where T, t, and p define (all within constant factors) the number of arithmetic operations (sequential time), arithmetic parallel steps (parallel time), and processors used, respectively.

Then by the virtue of the B-principle, the bound $O_A(t,p)$ implies $O_A(st,p/s)$ as long as p/s and s are positive integers. In particular, $O_A(t,p)$ implies $O_A(tp,1)$ if we let $s=p$, and tp is called the *potential work* of an algorithm (see [KR], [PP], and Definition 2.2.3), whereas the ratio $T/(tp)$ measures its *processor efficiency*, provided that T denotes the record sequential time bound for the same computational problem.

The B-principle has a wide area of applications. For example, we may immediately sum n numbers by using $\lceil log_2\ n \rceil$ parallel addition steps and n processors, but we may easily arrive at the bound of $O_A(log\ n,\ n/log\ n)$ by slowing down the first $\lceil log_2 \lceil log_2\ n \rceil \rceil$ steps and using the B-principle.

DEFINITION **2.2.1 (randomized algorithms).** *An algorithm is* randomized *when it makes use of randomly generated data (see [Re93, page 16]). Some of the algorithms in this thesis utilize* randomization *of the* **Las Vegas type or Monte Carlo type.** *A randomized algorithm of the* **Las Vegas** *type always produces the correct output, if and when it terminates, but it may run indefinitely. On the other hand, a randomized algorithm of the Monte Carlo type always terminates in some specific amount of time, but its output may give incorrect answer to the relevant problem, with some specified upper bound on probability.*

DEFINITION **2.2.2 (NC and RNC).** *A computational problem S, with input length (size) n, is in NC^k (respectively, randomized NC^k or RNC^k), if it can be solved by a deterministic (respectively, randomized) algorithm in $O(log^k\ n)$ parallel steps that use $n^{O(1)}$ processors. The NC (respectively, randomized NC or RNC) class (named so to honor Nicholas Pippenger) is defined as the set $NC = \cup_{k \geq 0}(NC^k)$ [respectively, the set $RNC = \cup_{k \geq 0}(RNC^k)$].*

DEFINITION **2.2.3 (Potential work of a parallel algorithm,** *see [BP] or [PP]). The product $W(n) = t(n)p(n)$ is called the* potential work *of a parallel*

algorithm. It represents the number of operations potentially executable by the algorithm, provided that the p processors the computation runs in $t(n)$ steps and all $p(n)$ processors involved were kept busy during the entire computation.

DEFINITION **2.2.4** *(***efficiency and optimality**, *see [KR]). The concepts of* efficiency *and* optimality *of a parallel algorithm are defined based on the requirement that the* potential work *of the algorithm is kept as close as possible to the running time of the fastest known sequential algorithm for solving the same problem, while also keeping the algorithm in NC. Denote* $polylog(n) = \cup_{\alpha \geq 0} O(log\ n)^{\alpha}$ *and let* $T(n)$ *be the number of steps required by the fastest known sequential algorithm for some computational problem, where n denotes the length (size, dimension) of the input to the computational problem. (For instance, the input length can be represented by the number of the input parameters, such as the numbers of the coefficients of all the input polynomials or the number of the entries of all the input matrices.) A parallel algorithm for the same problem is said to be processor (or work)* efficient *if its potential work W satisfies the equation* $W(n) = O(T(n)polylog(n))$*; it is said to be* optimal *if* $W(n) = O(T(n))$.

2.2.2 Classification of the *PRAM* Models

There are various *PRAM* models based on the various assumptions about handling the simultaneous access of several processors to the same global memory. The three types of *PRAM* model are:

- The **exclusive read exclusive write (EREW)***PRAM* does not allow any simultaneous access to a single memory location.
- The **concurrent read exclusive write (CREW)***PRAM* allows simultaneous access for a read instruction only.
- The **concurrent read concurrent write (CRCW)***PRAM* allows access to a location for both read and write instructions. It has futher variations depending on how the concurrency is resolved among several processors that try to write into the same location of the memory.

It turns out that these three models (EREW, CREW, CRCW) do not differ substantially in their computational powers, although the CREW is more powerful than the EREW, and the CRCW is the most powerful [KR].

To estimate the parallel complexity of computation by means of our algorithms, we will use the least powerful EREW PRAM, so that our upper estimates apply under the other PRAM models too.

2.2.3 Summary of the Arithmetic Complexity Estimates for Some Fundamental Matrix Computations Under EREW PRAM Model.

We conclude this section by summarizing the known parallel complexity estimates for some fundamental matrix computations.

PROPOSITION **2.2.1** *[BP], [P], [CW], [P91], [KP91], [KP92].*

Let $M(n,q,r)$, $I(q)$, $D(q)$ denote the problems of $n \times q$ by $q \times r$ matrix multiplication, $q \times q$ matrix inversion, and the evaluation of the determinant of a $q \times q$ matrix, respectively; then the parallel computational complexity of solving the above problems is given by

$$(t_{M(n,q,r)},\ p_{M(n,q,r)}) = O_A(\log q,\ nqrh^{w-3}), \tag{2.10}$$

$$(t_{I(q)},\ p_{I(q)}) = O_A(\phi(\mathbf{F},\ q),\ p_{M(q,q,q)}), \tag{2.11}$$

$$(t_{D(q)},\ p_{D(q)}) = O_A(\phi(\mathbf{F},\ q),\ p_{M(q,q,q)}), \tag{2.12}$$

respectively, where $h = min\{n,q,r\}$, $2 \le w < 2.376$, and

$$\phi(\mathbf{F},\ q) \le \begin{cases} \log^2 q, & \text{if } \mathbf{F} \text{ has characteristic } 0, \\ \log^4 q, & \text{for any } \mathbf{F}, \end{cases}$$

provided that the computation is performed over the field of constants $\mathbf{F}$.

The bounds (2.11) and (2.12) have been obtained in [P91], [P92], [KP91], [KP92], by using Las Vegas randomized algorithms.

2.3 Banded Linear Systems (BLS)

DEFINITION 2.3.1 (Bandwith of a matrix).

A matrix $A = (a_{ij})$ has a lower *(respectively,* an upper*) bandwith $m_- = m_-(A)$ (respectively, $m_+ = m_+(A)$) if m_- (respectively, m_+) is the minimum nonnegative integer such that $a_{ij} = 0$ for $i \ge j+m_-$ (respectively, $j \ge i+m_+$).*

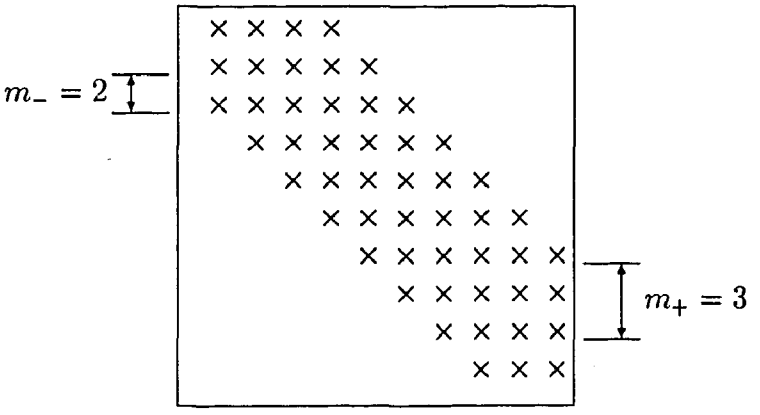

Figure 2.1: A 10×10 banded matrix with bandwith $m = 5$.

The sum $m = m_+ + m_- = m(A)$ *is called the* bandwith *of* A *(see figure 2.1). Some authors use the name* offset *to refer to the lower bandwidth.*

If A is a diagonal matrix, then $m(A) = 0$; if A is a triangular $n \times n$ matrix, then $m(A) \leq n - 1$; $m(A) \leq 2(n-1)$, for any $n \times n$ matrix A.

A $k \times h$ matrix A is called *banded* if $k + h - 2$ exceeds its bandwith $m(A)$.

DEFINITION **2.3.2 (Upper and lower edges).** *A matrix* A *has* lower (respectively, upper) *edge, if* $a_{ij} \neq 0$ *whenever* $i = j + m_-$ *(respectively,* $j = i + m_+$*).*

PROPOSITION **2.3.1** $m(W^H W) \leq 2m(W)$ *and* $m(W_k) \leq m(W)$*, for any principal submatrix* W_k *of* W*.*

FACT **2.3.1** *(***Block tridiagonal representation of a banded matrix***).* *Let A be an $n \times n$ matrix and let m be any integer such that*

$$\max(m_-(A),\ m_+(A)) \ \leq \ m < n;$$

then the matrix A has a $p \times p$ block matrix representation with $(m+1)\times(m+1)$ blocks of the following format where $p = \lceil n/m \rceil$:

$$A = \begin{pmatrix} A_{00} & A_{01} & & & 0 \\ A_{10} & A_{11} & \ddots & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & A_{p-2,p-2} & A_{p-2,p-1} \\ 0 & & & A_{p-1,p-2} & A_{p-1,p-1} \end{pmatrix}. \tag{2.13}$$

Conversely, block tridiagonal matrices with $p \times p$ blocks are also banded with bandwidth $m \leq 2p-1$.

REMARK **2.3.1** *In particular, if A is a triangular matrix, that is, $m_+(A) = 0$ and/or $m_-(A) = 0$, then (2.13) gives a block bidiagonal representation of A.*

FACT **2.3.2 (Representation-2).** *Any $n \times n$ matrix with a lower bandwith $m_- < n$ and an upper bandwidth $m_+ < n$ has the following block representaton:*

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, \tag{2.14}$$

where the blocks A_{11} and A_{22} have sizes $n_1 \times n_1$ and $n_2 \times n_2$, respectively, and where $n_1 = \lceil n/2 \rceil$, $n_2 = n - n_1$, $A_{12} = \begin{pmatrix} 0 & 0 \\ L & 0 \end{pmatrix}$, $A_{21} = \begin{pmatrix} 0 & U \\ 0 & 0 \end{pmatrix}$, U is an $m_+ \times m_+$ matrix and L is an $m_- \times m_-$ matrix.

Thereafter, we refer to (2.14) as to representation-2 of a matrix A.

FACT **2.3.3 (Representation-3)** *Any $n \times n$ matrix having upper bandwith and lower bandwidth $2m$ where $n \geq 3m$ has a block representation of the format*

$$A = \begin{pmatrix} A_{11} & A_{12} & 0 \\ A_{21} & A_{22} & A_{23} \\ 0 & A_{32} & A_{33} \end{pmatrix}, \tag{2.15}$$

where the blocks A_{11}, A_{22} and A_{33} have sizes $n_1 \times n_1$, $m \times m$ and $n_3 \times n_3$, respectively, and where $n_1 = \lceil (n-m)/2 \rceil$, $n_3 = n - m - n_1$, $A_{12} = \begin{pmatrix} 0 \\ S \end{pmatrix}$, $A_{21} = (0 \;\; E)$, $A_{23} = (W \;\; 0)$, $A_{32} = \begin{pmatrix} N \\ 0 \end{pmatrix}$, N, S, E, W are $m \times m$ matrices.

We refer to (2.15) as to representation-3 of a matrix A.

Solving banded linear systems of equations, that is, linear systems of equations with banded coefficients matrix, is among the most frequent operations in practice of scientific and engineering computations and has long been a subject of intensive research (see [GL]). In chapters 3 and 4 we will study algorithms for parallel solution of this problem.

2.4 Linear Recurrences

Many arithmetic computations can be expressed iteratively such that the value y_i generated at the end of the i^{th} iteration is a fixed *linear combination*

$$y_i = c_{i,0} + \sum_{k=1}^{m} c_{i,k} y_{i-m+k-1} \tag{2.16}$$

of the values $y_{i-1}, y_{i-2}, \ldots, y_{i-m}$ with the coefficients $c_{i,k}$ (where m, i, and k are positive integers). We refer the reader to [MT], [Kn], [GK], [J], [Sp], and [PW] for additional information on this subject.

If (2.16) holds, we say that y_1, y_2, ..., y_m defines a 1-dimensional **linear recurence of order m.** Given y_1, y_2, ..., y_m, we may evaluate y_{m+1}, y_{m+2}, ..., y_{2m} by recursively using equation (2.16). This is an effective sequential algorithm. We focus our attention in this section on the parallel implementation of the similar computation for linear recurrences.

PROBLEM **2.4.1** *($LIN \cdot RECUR = LR(m)$)* **(Forward Linear Recurence Problem)***:*

Given the coefficients $c_1, c_2, \ldots, c_m$ and the terms $y_1, y_2, \ldots, y_m$ of a linear recurrence of order m, compute the set of the next terms $y_{m+1}, y_{m+2}, \ldots, y_{2m}$.

We begin with some examples of $LIN \cdot RECUR$, the forward problem for linear recurrences.

2.4.1 Linear Recurrences of First Order and Polynomial Evaluation.

The problems of evaluating a polynomial at a point and solving banded triangular linear systems give rise to linear recurrences, as we show next.

Polynomial Evaluation. Let $p = (p_0, p_1, \ldots, p_n)$ be an array representing the coefficients of the polynomial

$$p(x) = p_0 x^n + p_1 x^{n-1} + \cdots + p_{n-1} x + p_n.$$

We wish to compute $p(x_0)$, where x_0 is a given point. **Horner's algorithm** is the classical method for computing $p(x_0)$ by using n multiplications and n additions. This method is based on rewriting the expression of $p(x_0)$ as follows:

$$p(x_0) = (\cdots((p_0x_0 + p_1)x_0 + p_2)x_0 + \cdots + p_{n-1})x_0 + p_n.$$

Hence, we can obtain $p(x_0)$ by computing the innnermost term $y_1 = p_0x_0+p_1$, then the next innermost term $y_2 = y_1x_0 + p_2$, and so on, until we get $p(x_0) = y_n = y_{n-1}x_0 + p_n$. This method can be expressed by the following linear recurrence:

$$y_0 = p_0$$

$$y_i = y_{i-1}x_0 + p_i, \quad 1 \leq i \leq n.$$

This recurrence is a **first-order** linear recurrence since y_i depends on only y_{i-1}.

2.4.2 Linear Recurrences (LR) and Banded Triangular Linear Systems (BTLS).

Let A be an $n \times n$ lower triangular matrix such that all the nonzero entries of A lie on the main diagonal or on the m diagonals below it, for some integer $m < n$. Then the matrix A is a banded lower triangular matrix with a bandwith m. The matrix A in Figure 2.4.2 is an an example of banded triangular matrix with bandwidth $m = 2$. For $m = 1$, the matrix is bidiagonal.

$$A = \begin{bmatrix} a_{11} & 0 & 0 & 0 & \cdots & \cdots & 0 \\ a_{21} & a_{22} & 0 & 0 & \cdots & \cdots & 0 \\ a_{31} & a_{32} & a_{33} & 0 & & & \vdots \\ 0 & a_{42} & a_{43} & a_{44} & \ddots & & \\ & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ & & \ddots & \ddots & \ddots & \ddots & 0 \\ & & & 0 & a_{n,n-2} & a_{n,n-1} & a_{n,n} \end{bmatrix}$$

Figure 2.2: A $n \times n$ banded triangular matrix with bandwith $m = 2$.

The solution to the linear system $Ax = b$ can be expressed by the following recurrence:

$$\begin{aligned} x_1 &= \frac{b_1}{a_{11}}, \\ x_i &= \sum_{j=i-m+1}^{i-1} -\frac{a_{ij}}{a_{ii}} x_j + \frac{b_i}{a_{ii}}, \quad 2 \le i \le n. \end{aligned} \tag{2.17}$$

We assume that all $a_{ij} = 0$ unless $0 \le i - j < m$. By setting

$$\begin{aligned} y_i &= x_i \ \ for \ \ all \ \ i, \\ c_{i,0} &= \frac{b_i}{a_{ii}}, \\ c_{i,k} &= -\frac{a_{i,\ i-k}}{a_{ii}}, \quad k = 1, \ldots, m-1; \quad i = 1, \ldots, n; \quad k < i, \\ c_{i,k} &= 0, \quad k > i, \end{aligned} \tag{2.18}$$

we reduce (2.17) to the format (2.16), where $c_{i,k} = 0$, for $k > i$. We also observe that the solution of the banded triangular system $Ax = b$ satisfies a linear recurrence (2.16) of order m of a special form where $c_{i,k} = 0$, for $k > i$.

The next results summarize the correlation between $LIN \cdot SOLVE$ for a banded triangular linear system (BTLS) and $LIN \cdot RECUR$.

FACT **2.4.1** *(a) The solution to any nonsingular banded triangular linear system of n equations with a bandwidth m satisfies the linear recurrence (2.16) where* $c_{i,k} = 0$, *for* $k > i$, $i = 1, \ldots, m$.

(b) Conversely, any m-th order linear recurrence (2.16), with the coefficients $c_{i,k} = 0$ *for* $i < k$, $i = 1, \ldots, m$, *can be expressed as a triangular linear system with a bandwidth m.*

2.4.3 Parallel Solution of Forward Problem for the First-Order Linear Recurrences.

We begin by developing a simple optimal parallel algorithm for computing all the terms y_i defined by the **first-order** linear recurrence:

$$\begin{aligned} y_1 &= b_1, \\ y_i &= a_i y_{i-1} + b_i, \quad 2 \leq i \leq n. \end{aligned} \tag{2.19}$$

At the first glance, the iterative nature of the computation defined by equation (2.19) seems to indicate a limited degree of parallelism. However, if $a_i = 1$ for all i, then the y_i expressions are the prefix sums of the elements $b_1, b_2, \cdots, b_n$, and we will now extend the well-known prefix-sum algorithm (see Appendix A) in order to compute all the terms defined by a first-order linear recurrence, thereby solving the forward problem. Such an

extension is known as the partial-cascade-sum method (see [Q87], pp. 138-139). Let $n = 2^k$ and let the index i be even, such that $2 \leq i \leq n$. Then, $y_i = a_i(a_{i-1}y_{i-2} + b_{i-1}) + b_i$, and hence $y_i = a_i a_{i-1} y_{i-2} + a_i b_{i-1} + b_i$. The resulting equations for even indices define a first-order linear recurrence of size $n/2$. More precisely, let $a'_i = a_{2i}a_{2i-1}$ and $b'_i = a_{2i}b_{2i-1} + b_{2i}$, for $1 \leq i \leq \frac{n}{2}$, and let $z_i = y_{2i}$. Then, the z_i terms satisfy the following recurrence:

$$\begin{aligned} z_1 &= b'_1, \\ z_i &= a'_i z_{i-1} + b'_i, \quad 2 \leq i \leq n/2. \end{aligned}$$

We now compute all the z_i values recursively. Once these values are obtained, we immediately have all the y_i values, for even indices i. If i is an odd index larger than 1, we can determine the y_i values by the equation

$$y_i = a_i y_{i-1} + b_i.$$

ALGORITHM **2.4.1 (forward computation of a first-order linear recurrence**, *see [Q87], [JJ]).*

- **Input**: *Two arrays* $B = (b_1, b_2, \cdots, b_n)$ *and* $A = (a_1 = 0, a_2, \cdots, a_n)$ *representing the first-order linear recurrence* $y_1 = b_1$ *and* $y_i = a_i y_{i-1} + b_i$, $(2 \leq i \leq n)$; n *is assumed to be a power of 2.*
- **Output**: *The values of all the* y_i.
- **begin**

1. **if** $n = 1$ **then** *set* $y_1 := b_1$, **exit**
2. **for** $1 \leq i \leq \frac{n}{2}$ **pardo**
   - *Set* $a'_i := a_{2i}a_{2i-1}$
   - *Set* $b'_i := a_{2i}a_{2i-1} + b_{2i}$
3. *Recursively,* **solve** *the first-order linear recurrence defined by*

$$z_1 = b'_1 \quad \textit{and} \quad z_i = a'_i z_{i-1} + b'_i, \quad \textit{where} \quad 2 \leq i \leq \frac{n}{2}.$$

4. **for** $1 \leq i \leq n$ **pardo**
   - **if** i *even* : *Set* $y_i := z_i/2$
   - **if** $i = 1$: *Set* $y_1 := b_1$
   - **if** i *odd* > 1 : *Set* $y_i := a_i z_{(i-1)/2} + b_i$

- **end**

REMARK **2.4.1** *No simultaneous memory access is required to implement Algorithm (2.4.1). Therefore, this runs on the arithmetic EREW PRAM.* ■

THEOREM **2.4.1** *Algorithm (2.4.1) correctly computes the values of all the variables* y_i, *where* $1 \leq i \leq n$. *This algorithm can be implemented to run at the arithmetic parallel cost* $O_A(\log n, \frac{n}{\log n})$.

PROOF. The correctness proof follows from the discussion preceding the statement of the algorithm. As for the complexity bounds, they can be estimated as follows. Let $T(n)$ and $W(n)$ be the running time and the total

number of operations required by algorithm (2.4.1). Step 1 requires $O(1)$ sequential time. Step 2 can be performed in $O(1)$ time, using a total of $O(n)$ operations; step 3 is a recursive call that takes $T(n/2)$ time and uses $W(n/2)$ operations. Finally, step 4 can be executed in $O(1)$ time, using a total of $O(n)$ operations. Therefore, $T(n)$ and $W(n)$ satisfy the following recurrences:

$$T(n) = T(n/2) + O(1),$$

$$W(n) = W(n/2) + O(n).$$

Hence, $T(n) = O(log\ n)$ and $W(n) = O(n)$, as stated in the theorem. ■

COROLLARY **2.4.1** *Horner's algorithm (section 2.4) for the evaluation of a polynomial of degree n at a single point can be implemented to run in $O(log\ n)$ time, using a total of $O(n/log\ n)$ operations.*

2.4.4 Higher-Dimensional Linear Recurrences

Let us generalize equation (2.19) to include first-order **higher-dimensional** linear recurrences, which are actually linear recurrences of a higher order (compare [J]).

Let $\{b_i \mid\ 1 \leq i \leq n\}$ be a set of m-dimensional vectors, and let A_i be an $m \times m$ matrix, where $2 \leq i \leq n$. Let us consider the m-dimensional linear recurrence of the first-order $LR_m(1)$ defined by

$$y_1 = b_1,$$

$$y_i = A_i y_{i-1} + b_i, \quad 2 \leq i \leq n, \tag{2.20}$$

where y_i and b_i for all i are m-dimensional vectors. The next result shows a simplification of (2.20) in a special case.

FACT **2.4.2** *Let v be an m-dimensional vector. Suppose that $b_1 = v$, $b_i = 0$ for $2 \leq i \leq n$, and $A_i = A$ in equation (2.20). Then the equation (2.20) becomes*

$$y_i = A^{i-1} v, \quad for\ 1 \leq i \leq n.$$

Some properties of the 1-dimensional linear recurrences can be extended to the case of higher dimensional linear recurrences. In particular, Fact 2.4.1 is extended:

FACT **2.4.3** *1. The solution of any block bidiagonal linear system with $m \times m$ blocks satisfies a linear recurrence of dimension m and order 1.*

2. Conversely, any linear recurrence of dimension m and order 1 satisfies a block bidiagonal linear recurrence with $m \times m$ blocks.

Clearly, every m-dimensional linear recurrence of first order can be represented as a 1-dimensional linear recurrence of order $2m$. Conversely, we have the following fact.

FACT **2.4.4** *Every 1-dimensional linear recurrence (2.17) of order m and satisfying the equations $c_{i,k} = 0$ for $k > i$, can be represented as a linear recurrence of dimension m and order 1.*

PROOF. First represent the given linear recurrence as a banded triangular system of equation (see Fact 2.4.1b). Then represent this system as a block bidiagonal linear system of equations (see Fact 2.3.1 and Remark 2.3.1). Finally apply Fact 2.4.3, part 1. ∎

We will use the representation (2.20) in order to extend the results of the previous subsection from the case of a first-order linear recurrence to any recurrence of order m. In particular, we will now extend Algorithm 2.4.1 to solve the problem $LIN \cdot RECUR$ in the case of a linear recurrence defined by (2.20). Specifically, we apply exactly the same algorithm but assume that y_i and b_i now denote the vectors of (2.20), and we replace the scalars a_i by matrices A_i.

We will now state the resulting complexity estimates in Theorem 2.4.2, where we will write

$$p_{M(m)} = p_{M(m,m,m)} = m^w, \quad 2 \leq w < 2.376 \tag{2.21}$$

(see Proposition 2.2.1).

THEOREM **2.4.2** *The problem $LIN \cdot RECUR$ for the first-order linear recurrence of dimension m defined by equation (2.20), where the matrices A_i are each of dimension $m \times m$ and the vectors y_i and b_i are each of dimension m, can be solved at the cost $O_A(\log n \log m, \frac{np_{M(m)}}{\log n})$, where $p_{M(m)} = p_{M(m,m,m)} = m^w$, $2 \leq w < 2.376$, is a processor bound for multiplying a pair $m \times m$ matrices in time $O(\log m)$.*

COROLLARY **2.4.2** *The complexity bounds of Theorem 2.4.2 also apply to* $LIN \cdot SOLVE$ *for a banded triangular linear system of* n *equations where* m *denotes the bandwidth.*

PROOF. Corollary 2.4.2 follows immediately from Fact 2.4.1 and Theorem 2.4.2. ∎

2.5 Parallel Algorithms for Solving a Triangular Linear System of Equations and for Triangular Matrix Inversion

One of the most fundamental problems in matrix computations is $LIN \cdot SOLVE$, that is, solving a linear system of equations. The classical Gaussian elimination scheme solves a linear system of equations by reducing it to a triangular form and then generates the solution by using the standard substitution algorithms. In this section we present a simple parallel algorithm for solving a triangular linear system of n equations:

$$Ax = b, \tag{2.22}$$

where x and b denote a pair of n-dimensional vectors and A denotes an arbitrary $n \times n$ unit lower triangular matrix:

$$A = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ a_{21} & 1 & 0 & \dots & 0 \\ a_{31} & a_{32} & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{n,n-1} & 1 \end{bmatrix}.$$

For a given arbitrary nonsingular lower triangular system, we make all the entries on the diagonal equal to 1 by dividing the ith equation by a_{ii}, whenever $a_{ii} \neq 1$, for $1 \leq i \leq n$.

Perhaps the simplest method for the evaluation of x is the substitution method, which successively computes $x_1 = b_1$, $x_2 = b_2 - a_{21}x_1$, $x_3 = b_2 - a_{31}x_1 - a_{32}x_2, \cdots$, $x_i = b_i - \sum_{j=1}^{i-1} a_{ij}x_j$, ... This method only requires $O(n^2)$ arithmetic operations, but does not lead to a fast parallel algorithm. We will next present an alternative method for both problems $LIN \cdot SOLVE$ and $INVERT$ (that is, the problem of computing matrix inverse), for a nonsingular input matrix. This method can be implemented at the parallel cost $O_A(log^2\ n,\ p_{M(n)})$.

Let A be a nonsingular $n \times n$ lower triangular matrix, where n is assumed to be a power of 2. A can be partitioned into blocks as follows:

$$A = \begin{bmatrix} A_1 & 0 \\ A_2 & A_3 \end{bmatrix}, \tag{2.23}$$

A_1 and A_3 are nonsingular lower triangular matrices, A_1 is an $h \times h$ matrix. Then A^{-1} is given by:

$$A^{-1} = \begin{bmatrix} A^{-1} & 0 \\ -A_3^{-1}A_2A_1^{-1} & A_3^{-1} \end{bmatrix}. \tag{2.24}$$

We can set $h = n/2$ and obtain the inverse of A by recursively computing the inverses of A_1 and A_3 and by performing two matrix multiplications to generate the term $-A_3^{-1}A_2A_1^{-1}$. This divide-and-conquer method leads to the following theorem.

THEOREM **2.5.1** *An $n \times n$ nonsingular triangular matrix can be inverted and a nosingular triangular linear system of n equations can be solved at the arithmetic cost $O_A(log^2\ n,\ p_{M(n)})$.*

PROOF. The time bound of $O(log^2\ n)$ follows from the observation that the algorithm runs by using $O(log\ n)$ iterations, each of which involves two multiplications of matrices of sizes less than or equal to $(n/2)$.

Let $W(n)$ be the number of arithmetic operations needed to invert an $n \times n$ lower triangular matrix, by means of this algorithm. $W(n)$ satisfies the following recurrence:

$$W(n) = 2W(n/2) + 2p_{M(n/2)}.$$

Clearly, $p_{M(n)} \leq M(n)/log\ n$ and $n^{2.376} > M(n) > n^2$, so that $p_{M(n)} \leq 4p_{M(n/2)}$. It follows that $2p_{M(n/2)} = O(p_{M(n)})$ processors suffice in each stage of the algorithm. As soon as A^{-1} has been computed, we immediately obtain the solution $x = A^{-1}b$ to $LIN \cdot SOLVE$. ■

By applying the B-principle one may easily improve the bound of theorem 2.5.1 (see [P] and/or [PP]) to reach the bounds

$$O_A(log^2\ n,\ \frac{p_{M(n)}}{log^2\ n}). \tag{2.25}$$

The potential work for the resulting algorithm is $O(log\ n)p_{M(n)})$, which shows that this parallel algorithm is work (processor) optimal (see Definition 2.2.4).

Next, we will show some examples of application of the technique of supereffective slow-down, which yields quite fast parallel algorithms with

potential work significantly smaller than that of the fastest parallel algorithm for the same problem. In many cases, this technique can also be viewed as a work-preserving parallel acceleration of an existing recursive sequential algorithm for the considered problem.

To achieve a supereffective slow-down for the solution of equation (2.22), we use the estimates of equation (2.25) and the fast algorithm for parallel matrix inversion.

Since A is nonsingular, x can be obtained from the equation $x = A^{-1}b$, but if we first compute A^{-1} in order to obtain x, then the resulting algorithm is not work efficient, since the simple substitution algorithm yields the bounds $O_A(n, n)$ and has a potential work $O(n^2)$.

We will, however, select a parameter h so as to define an $h \times h$ matrix A and a vector $b = [c, d]$ of dimension h in (2.22). Then we have:

$$A^{-1}b = \begin{bmatrix} A_1^{-1} & 0 \\ -A_3^{-1}A_2A_1^{-1} & A_3^{-1} \end{bmatrix} \begin{bmatrix} c \\ d \end{bmatrix}$$

$$= \begin{bmatrix} A_1^{-1}c \\ -A_3^{-1}A_2A_1^{-1}c + A_3^{-1}d \end{bmatrix}. \tag{2.26}$$

If

$$e = A_1^{-1}c,$$

and

$$f = -A_2e + d,$$

then

$$x = A^{-1}b = \begin{bmatrix} e \\ A_3^{-1}f \end{bmatrix}.$$

We can compute e at the cost $O_A(log^2\ h,\ h^3/log^2\ h)$ by substitution of h for n in equation (2.25), and subsequently compute f at the cost $O_A(log^2\ h,\ nh/log^2\ h)$. By doing so, we reduce the original problem to computing $A_3^{-1}f$, that is, to solving the triangular linear system (TLS)

$$A_3 y = f$$

of $n - h$ equations, where $y = A_3^{-1}f$.

Similary, we reduce this system to triangular systems of $n - 2h$, $n - 3h$, ... equations, and thereby solve the original system (2.22). The cost of performing the algorithm is:

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

which leads us to the bound

$$O_A(n^{1/2}\ log^2\ n,\ n^{3/2}/log^2\ n),$$

if we select $h = n^{1/2}$. ■

Chapter 3

Fast and Work Optimum Parallel Solution of a Sparse Linear System of Equations. A Specialization to the Case of Banded Linear Systems. Nested Dissection and Block Cyclic Reduction.

3.1 Introduction

Many applications to the sciences and engineering require the solution of very large sparse linear systems of equations, and in many cases this task has been made feasible on modern computers due to the application of the (generalized) nested dissection (ND) algorithms. The nested dissection (ND) algorithm applies to any sparse linear system, for whose associated graph, a recursive family of small separators is available (see [GLi] and [GT]); such an algorithm has effective parallel implementations (see [PR] and [P93]). For

banded (block tridiagonal) symmetric linear systems the desired families of small separators are readily available. In particular, banded linear systems (see Definition 2.3.1) is a special case of sparse linear systems, which can be effectively solved by the (ND) algorithms. On the other hand, among the customary techniques for solving banded linear systems (BLS) of equations, the block cyclic reduction (BCR) is considered by the users the champion for parallel implementation. The BCR algorithm applies to a block tridiagonal linear system of k block equations with $m \times m$ blocks (any linear system of km equations with lower and upper bandwidths m can be represented in this way, due to Fact 2.3.1) and solves this system in $O(log\ k)$ recursive steps, with the parallel computational cost of each step dominated by the cost of k concurrent inversions of $m \times m$ blocks. The main idea behind this technique consists of halving the size of the problem in hand, by eliminating half of the variables, until we are left with a linear system of a desired small size. The latter system is then easily solved by some standard means, and the previously eliminated variables are computed via back-substitution.

In this chapter we will recall both the nested dissection and the block cyclic reduction algorithms. At the end of this study, we will observe that, in the case of banded matrices and for an appropriate choice of small separators, both techniques coincide with each other (see remark 3.6.2), as well as with the partial-cascade-sum/prefix-sum algorithm.

Stating our results, we will keep assuming the arithmetic *EREW PRAM*

model of parallel computation and the B-principle, and will use the definitions of chapter 2, section 2.2.1 and the known complexity estimates for parallel matrix computation recalled in chapter 2, section 2.2.3.

3.2 Graph Models of Sparse Elimination

3.2.1 Introduction and Definitions

Most frequently, in matrix computations, and particularly in applications to solving linear systems of equations, matrices are either very large but special (sparse or dense but structured) or have smaller sizes. Hereafter, we will refer to general matrices when we allow to include the class of dense unstructured matrices.

PROBLEM **3.2.1** *(LU* **factorization of a matrix** *A)*

Given an $n \times n$ nonsingular matrix A, find a nonsingular lower triangular matrix L and a nonsingular upper triangular U such that $A = LU$.

FACT **3.2.1** *[GL]. There exists a unique LDL^H factorization of any h.p.d. or r.s.p.d. matrix, where L is a unit lower triangular matrix, D is a diagonal matrix with positive diagonal entries, and L^H is the Hermitian transpose of L; $L^H = L^T$, a transpose of L, in the real case.*

Solution of Problem 3.2.1 reduces $LIN \cdot SOLVE$ (for $Ax = b$) to solving two triangular linear systems $Ly = b$ and $Ux = y$. The customary subroutines for Gaussian elimination for $LIN \cdot SOLVE$ consist of computing the

LU factorization followed by solving such a pair of triangular linear systems by means of substitution.

In this section we will represent the *LU* factorization of a sparse matrix *A* by using the associated ordered graph. We will assume that *A* is h.p.d. or r.s.p.d. (see definition 2.1.1 and remark 2.1.1); in this case, its *LU* factorization is unique, and moreover, the numerical stability problems do not contaminate the known effective solution algorithms [GL], which thereby are useful in practice.

DEFINITION **3.2.1** **A binary tree** *is a finite set of elements which is either empty or contains a single element called the* **root** *of the tree and whose remaining elements are partitioned into two disjoint subsets, each of which is itself a binary tree. These two subsets are called* **left** *and* **right subtrees** *of the original tree. Each element of a binary tree is called a* **node**.

DEFINITION **3.2.2** *The* **height** *of a binary tree is the maximum level of its leafs (this is also known sometimes as the* **depth** *of the tree).*

DEFINITION **3.2.3** *A* **balanced binary tree** *(sometimes called an* **AVL tree***) is a binary tree in which the heights of the two subtrees of every node never differ by more than 1. The* **balance** *of a node in a binary tree is defined as the height of its left subtree minus the height of its right subtree.*

DEFINITION **3.2.4** $|S|$ *denotes the cardinality of a set* S.

DEFINITION **3.2.5** *A* **graph** *is a pair* $G = (V, E)$ *where* V *is a finite set of* $|V|$ *elements called* **vertices** *and* $E \subseteq \{\{v,\ w\} | v,\ w \in V\ \ v \neq w\}$ *is a set of* $|E|$ *vertex pairs called* **edges.** *An edge* $\{v,\ w\}$ *will also be denoted simply as* vw.

We will only need to consider the case of undirected graphs (compare remark 3.2.1).

DEFINITION **3.2.6** *For* $v \in V$ *the set* $adj(v) = \{w \in V \mid vw \in E\}$ *is the set of vertices* **adjacent** *to* v.

DEFINITION **3.2.7** *If* $W \subseteq V$ *then the* **induced subgraph** $G[W]$ *of* G *is the subgraph* $G[W] = (W, E[W])$ *where* $E[W] = xy \in E \mid x,\ y \in W$. *For* $W \subseteq V$ *let* $C(W) = \{\{u, v\} \mid u,\ v \in W,\ \ u \neq v\}$; *then* W *is called a* **clique** *in* G *if and only if* $C(W) \subseteq E$.

DEFINITION **3.2.8** *A* **(u, v)-path** *of length* k *is a sequence of vertives* $p = (u = u_1,\ u_2, \cdots,\ u_{k+1} = v)$ *such that* $u_{i+1} \in adj(u_i)$ *for* $1 \leq i \leq k$ *and* $u_i \neq u_j$ *for* $i \neq j$ *except, possibly, for* $u = v$, *and then* p *is a* **cycle.** *Hence all paths (except cycles) contain distinct vertices. A graph* G *is connected if there exists a* $(u,\ v)$*-path for each distinct pair* $u,\ v \in V$.

DEFINITION **3.2.9** *An* **ordering** *of the vertices of* V $(|V| = n)$ *for a graph* $G = (V,\ E)$ *is a one-to-one correspondence* $\alpha :\ \ \{1,\ 2, \cdots,\ n\} \longleftrightarrow V$. *[Occationally, we will also denote an ordering by writing* $V = \{v_i\}_{i=1}^{n}$.] $G_\alpha =$

(V, E, α) is an **ordered graph.** *For $u, v \in V$ we write $u <_\alpha v$ if $\alpha^{-1}(u) < \alpha^{-1}(v)$.*

DEFINITION **3.2.10** *A graph $G = (V, E)$ is said to be* planar *if it can be drawn on a plane in such a way that no edges cross one another, except, of course, at common vertices.*

DEFINITION **3.2.11** **($s(n)$-separator family).** *A graph $G = (V, E)$ is said to have an* $s(n)$-separator family *(with respect to two constants, $\alpha < 1$ and n_0) if either $|V| \leq n_0$ or, by deleting some separator set S of vertices such that $|S| \leq s(|V|)$, we may partition G into two disconnected subgraphs with the vertex sets V_1 and V_2, such that $|V_i| \leq \alpha|V|$, $i = 1, 2$, and if, furthermore, each of the two subgraphs of G defined by the vertex sets $S \bigcup V_i$, $i = 1, 2$, also has an $s(n)$-separator family (with respect to the same constants $\alpha < 1$ and n_0). The resulting recursive decomposition of G is known as the* **$s(n)$-separator tree,** *so that each of the subgraph of G in the decomposition defines its children in the tree.*

DEFINITION **3.2.12** *A graph is called $s(n)$-separatable if it has an $s(n)$-separator family and if its $s(n)$-separator tree is available.*

3.2.2 Graph Representation of a Matrix

Let M be a symmetric $n \times n$ matrix.

DEFINITION **3.2.13** *The* ordered graph *$G(M)$ of M is defined as $G(M) = (V, E)$ where $V = \{v_i\}_{i=1}^n$ and $v_i v_j \in E$ if and only if $m_{ij} \neq 0$ and $i \neq j$.*

$$M = \begin{bmatrix} \times & \times & 0 & 0 & 0 \\ \times & \times & \times & 0 & 0 \\ 0 & \times & \times & \times & 0 \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & \times & \times \end{bmatrix}$$

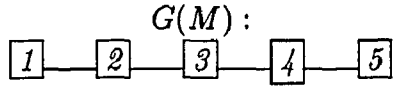

Figure 3.1: A Tridiagonal Matrix and the Corresponding Path Graph

REMARK **3.2.1** *We assume that $m_{ij} \neq 0$ if and only if $m_{ji} \neq 0$, that is, we assume undirected edges (unordered pairs $\{v_i, v_j\}$) and hence undirected graphs (also simply called graphs) when we define $G(M)$.*

Below are some examples of symmetric matrices M and their corresponding graph $G(M)$.

EXAMPLE **3.2.1** *In figure 3.1 we show a tridiagonal matrix M, of order 5, and its corresponding path graph $G(M)$. Each $\times$ denotes a nonzero entry m_{ij} of M.*

REMARK **3.2.2** *When M is an $n \times n$ real symmetric tridiagonal matrix, $G(M)$ is the* **path graph** *on n vertices; i.e.,*

$$G(M) = (\{v_i\}_{i=1}^{n}, \quad \{v_i v_{i+1} \mid 1 \leq i \leq n-1\}.)$$

Tridiagonal matrices M will be denoted as $M = [b_i, a_i, b_{i+1}]$ with diagonal entries denoted a_i, $1 \leq i \leq n$, and off-diagonal entries by b_i, $2 \leq i \leq n$.

EXAMPLE **3.2.2** *Let M be an $n^2 \times n^2$ block tridiagonal matrix denoted by $[D_i, T_i, D_{i+1}]$, where the $n \times n$ matrices D_i, $2 \leq i \leq n$, and T_i, $1 \leq i \leq n$, are*

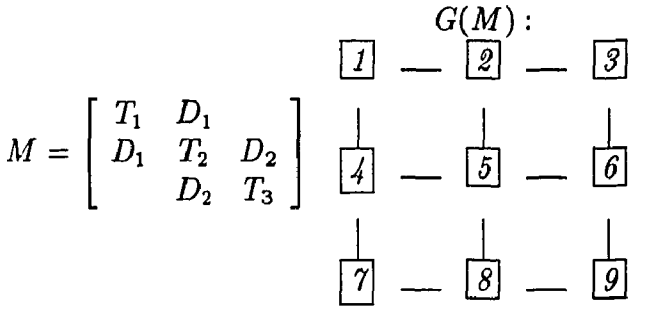

Figure 3.2: A Block Tridiagonal Matrix and the Corresponding Grid Graph

respectively diagonal matrices and tridiagonal matrices. A block tridiagonal matrix of order 9 *is shown in figure 3.2. The* D_i *and* T_i *are* 3×3 *matrices.*

The graph of the $n^2 \times n^2$ block tridiagonal matrix $M = [D_i\ T_i\ D_{i+1}]$ will be called the **grid graph** $G_n = (V,\ E)$ with $|V| = n^2$. We can regard G_n embedded in the plane as follows:

$$\begin{aligned} V &= \{(i,j) \mid 1 \le i,\ \ j \le n\}, \\ E &= \{(i,j),\ (i,j+1) \mid 1 \le i \le n \text{ and } 1 \le j \le n-1\} \\ &\cup\ \{(i,j),\ (i+j,j) \mid 1 \le i \le n-1 \text{ and } 1 \le j \le n\}. \end{aligned}$$

3.2.3 Vertex Elimination on a Graph

Let M be a sparse h.p.d. or r.s.p.d. $n \times n$ matrix and $G(M) = (V, E, \alpha)$ be the ordered graph of M. Then for any permutation matrix P, the matrix

PMP^T is also h.p.d. or r.s.p.d. and, therefore, has a unique LDL^T decomposition (which depends on P). Gaussian elimination applied to the matrix PMP^T is equivalent to graph elimination with a certain pivoting policy (of row and column interchange) apllied to the original matrix M. Equivalent representation is in terms of an ordering of the associated ordered graph $G(M) = (V, E)$. A major problem of sparse Gaussian elimination is to choose a permutation P or, equivalently, an ordering for the graph $G(M)$ so as to make the elimination process efficient.

Next, we will specify efficiency of this process in terms of the concept of fill-in of the graph $G(M)$. Let M be written as

$$M = \begin{bmatrix} a & c^T \\ c & B \end{bmatrix}, \tag{3.1}$$

where a is a scalar, c is $(n-1) \times 1$ and B *is* $(n-1) \times (n-1)$. Recall that the first step in the factorization of M is

$$M = \begin{bmatrix} 1 & 0 \\ c/a & I \end{bmatrix} \begin{bmatrix} a & 0 \\ 0 & B^{(2)} \end{bmatrix} \begin{bmatrix} 1 & c^T/a \\ 0 & I \end{bmatrix}, \tag{3.2}$$

where $B^{(2)} = B - cc^T/a$. Notice that $B^{(2)}$ has potentially more zeros than B due to the term cc^T/a (here and hereafter we ignore the possibility of random cancellation of the entries of matrices). Any new nonzero elements created during the factorization process is called **fill-in.** To complete the LDL^T decomposition, one proceeds recursively, letting $B^{(2)} = L_2 D_2 L_2^T$. Using equation (3.2) we will write

$$M = \begin{bmatrix} 1 & 0 \\ c/a & L_2 \end{bmatrix} \begin{bmatrix} a & 0 \\ 0 & D_2 \end{bmatrix} \begin{bmatrix} 1 & c^T/a \\ 0 & L_2^T \end{bmatrix} = LDL^T. \tag{3.3}$$

Let us now model this process of vertex elimination on a graph $G = (V, E)$. For any $v \in V$, the **deficiency,** $D(v)$, is the set of all distincts pairs of $adj(v)$ that are not themselves adjacent, that is,

$$D(v) = \{\{xy\} \mid x, y \in adj(v),\ x \notin adj(v),\ x \neq y\}. \tag{3.4}$$

The graph G_v is called the **v-elimination graph** of G if it is obtained from G by

1. deleting v and its incident vertices and

2. adding edges so that all vertices in $adj(v)$ are paiwise adjacent, that is,

$$G_v = (V - v,\ E[V - v] \cup D(v)). \tag{3.5}$$

Let $G = (V,\ E,\ \alpha)$ be an ordered graph with $V = \{v_i\}_{i=1}^{n}$ and consider the sequence of recursively defined v_i—elimination graphs, $G_0 \equiv G$, $G_i = (G_{i-1}))_{v_i}$ for $1 \leq i \leq n-1$. If $G_i = (V_i,\ E_i)$, $1 \leq i \leq n-1$, the **fill-in** $F(G_\alpha)$ is defined by

$$F(G_\alpha) = \cup_{i=1}^{n-1} \tau_i, \tag{3.6}$$

where $\tau_i = D(v_i) \in G_{i-1}$, and the **elimination graph** G_α^* is defined as

$$G_\alpha^* = (V,\ E \cup F(G_\alpha)).$$

Given M as in equation (3.1) with $G(M) = (v, E, \alpha)$ we can see immmediately that v_1— elimination graph, G_{v_1}, is the ordered graph of the submatrix

$B^{(2)}$ of equation (3.2) (defined after the first step of elimination), and the deficiency $\tau_1 = D(v_1)$ in G_0 corresponds to the numerical fill-in. Similarly, the elimination graph G_i would correspond to the matrix $B^{(i+1)}$ as we continue to factorize $B^{(2)}$ in the numerical process of symmetric elimination. Finally, the graph G^*_α corresponds to the total LDL^T decomposition in the sense that $G^*_\alpha = G(C)$, where the strictly lower and upper triangles of C are L and L^T, respectively, and the diagonal of C is D.

Thus the graphs G_i, $0 \leq i \leq n-1$, and G^*_α capture the combinatorial aspects of symmetric Gaussian elimination. Also, since we have assumed that A is h.p.d. or r.s.p.d., its LDL^T decomposition is unique [GL], and therefore, G^*_α is independent of the numerical process used to compute it.

DEFINITION **3.2.14** *If the graph $G = (V, E)$ has an ordering α such that $F(G_\alpha) = \phi$, we call α a* **perfect elimination ordering** *and G a* **perfect elimination graph.**

PROPOSITION **3.2.1** *Let $G(M) = (V, E, \alpha)$ with $V = \{v_i\}_{i=1}^n$ and $M = LDL^T$ as in equation (3.3). The following are equivalent:*

1. $m_{ij} = 0 \longrightarrow l_{ij} = 0$ *for* $i > j$;

2. α *is a perfect elimination ordering for* G;

3. $adj(v_i)$ *is a clique in* G_{i-1} *for* $1 \leq i \leq n-1$.

For some graphs there exists no perfect elimination ordering. Also, the problem of computing an optimum elimination ordering is NP-hard, that

is, unlikely to have any acceptable solution already for inputs of moderate sizes [GJ]. Therefore, we may rely either on heuristics or on some effective (although generally not optimum) elimination ordering under some assumptions on the class of input matrices. The famous algorithm of nested dissection (ND) is one of the most effective and customary policies of ordering the elimination, in the important case, covering many pratical classes of sparse linear systems ([GLi], [GT], [LT], [PR85], [PR93], [P93]), where the ordered graph $G(M)$ is given with its family of $s(n)$-separators, for $s(n) = o(n)$. In particular, this case includes the class of planar graphs, grid graphs, various finite element graphs associated with the discretized *PDEs* and graphs $G(M)$ associated with block tridiagonal h.p.d. or r.s.p.d. matrices.

In this case the separator family and appropriate elimination ordering define block vertex elimination and, therefore, Guassian elimination for the associated matrix PMP^T.

3.3 A 2-Separator Tree for an H.p.d. Block Tridiagonal Matrix

Let the *graph* associated with an $n \times n$ h.p.d. or r.s.p.d. matrix $M = (m_{ij})$ be an undirected graph denoted $G(M) = (V(M),\ E(M))$ with the vertex set $V(M) = \{1, \ldots, n\}$ and the edge set $E(M) = \{(i,j) \mid m_{ij} \neq 0\}$. Hereafter,

M denotes a $q \times q$ block tridiagonal matrix of the format:

$$\begin{pmatrix} D_1 & F_1 & & & & & \\ E_2 & D_2 & F_2 & & & & \\ & \ddots & \ddots & \ddots & & & \\ & & & \ddots & \ddots & \ddots & \\ & & & & E_{q-1} & D_{q-1} & F_{q-1} \\ & & & & & E_q & D_q \end{pmatrix}, \tag{3.7}$$

where D_i, E_j and F_h are $m \times m$ matrices. We will assume that $det(D_i) \neq 0$ for $i = 1, \ldots, q$ and M is h.p.d. or r.s.p.d., so that $E_{j+1} = F_j$ for $j = 1, \ldots, q-1$. For simplicity (with no loss of generality), we will assume that

$$q = n/m = 2(2^r - 1), \tag{3.8}$$

for some fixed integer $r \geq 1$.

Let $G = G(M) = (V,\ E) = (V(M),\ E(M))$ denote the graph associated with the matrix M. According to the definition of the graph of a matrix, $|V| = n$, and $|E| \leq (3q-2)m$ (the maximum number of nonzero scalar entries of M). However, we may also interpret $G(M)$ from a block matrix point of view, by setting $V = \{1\ldots, q\}$, and $E = \{(i,j) \mid m_{ij} \neq 0\}$. Hereafter, in this chapter, we will assume the block matrix representation of $G(M)$, unless specified otherwise. We seek an $s(n)$-separator tree $\mathcal{T}(G(M))$ of $G(M)$ (note that $s(n)$-separatability of $G(M)$, with the latter interpretation of $G(M)$, immediately induces $[ms(n)]$-separatability of $G(M)$ from the former point of view).

Let $1 < k < q-1$ and consider the partition of V defined by

$$V_1 = \{1, \ldots, k-1\}, \quad S = \{k, k+1\}, \quad V_2 = \{k+2, \ldots, q\};$$

observe that the subgraphs G_1 and G_2 of $G(M)$, associated with the vertex sets V_1 and V_2, respectively, are disconnected. This observation immediately induces a balanced 2-separator tree $\mathcal{T}(G(M))$ of the graph $G(M)$, which can be obtained by performing the following computations with input $V(M) = \{1, \ldots, q\}$:

ALGORITHM **3.3.1** *(***balanced 2-separator tree.***)*

input: *a set of consecutive integers $U = \{a, a+1, \ldots, b\} \subseteq V$;*

output: *a 2-separator tree $\mathcal{T}(G_U)$ of the subgraph G_U of $G(M)$ defined by the vertex set U;*

computations:

1. *if $|U| \leq 2$, then let $\mathcal{T}(G_U)$ be the binary tree consisting of a single (root and leaf) node $N(G_U) = U$.*

2. *otherwise (if $U > 2$), let $w = \lfloor (a+b)/2 \rfloor$, $U_1 = \{a, \ldots, w-1\}$, $S = \{w, w+1\}$, $U_2 = \{w+2, \ldots, b\}$ (the set U_2 may possibly be empty); compute $\mathcal{T}(G_{U_1})$ and $\mathcal{T}(G_{U_2})$ by recursively applying this algorithm (balanced 2-separator tree) to the sets U_1 and U_2 (in the case where U_2 is empty, simply set $\mathcal{T}(G_{U_2}) = \emptyset$); form the tree $\mathcal{T}(G_U)$ with root=$N(G_U) = S$, the left subtree $\mathcal{T}(G_{U_1})$, and the right subtree $\mathcal{T}(G_{U_2})$;*

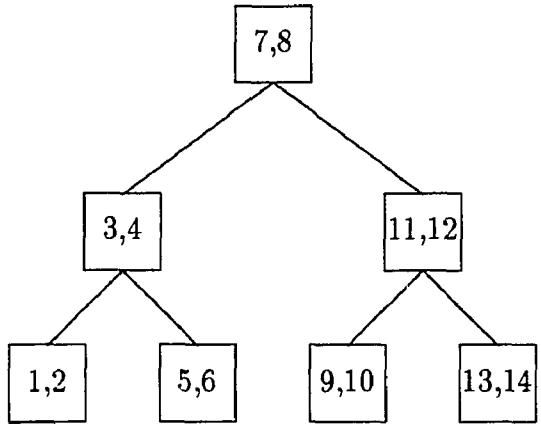

Figure 3.3: A 2-separator tree form a 14×14 block tridiagonal matrix.

---

end.

The tree $\mathcal{T}(G(M))$, output by algorithm 3.3.1 (balanced 2-separator tree) is commonly known as a *heap*, and is often used as a sorting tool in the literature on data structures (see [AT, p.481]). An illustration (for $V = \{1, \ldots, 14\}$) is shown on Figure 3.3. (Note that the leaf nodes of the output tree consists of all the pairs of integers $\{1 + 4j,\ 2 + 4j\}$ for $j = 0, \ldots, (q-2)/4 = 2^{r-2} - 1$, provided that $q = 2(2^r - 1)$; moreover, r is the depth of the binary tree $\mathcal{T}(G(M))$.

3.4 Factorization of an H.p.d. Block Tridiagonal Matrix, Based on Its $s(n)$-Separator Tree

Recall that any $n \times n$ r.s.p.d. or h.p.d. matrix M has a unique factorization (see section 3.2.3) of the format

$$M = LDL^T, \tag{3.9}$$

where L is a unit lower triangular matrix, $D = diag(\ d_1^2,\ \ldots,\ d_n^2)$ is a real diagonal matrix with nonzero positive entries. This factorization is commonly known as *Cholesky factorization* of M (see [GL] or [DB]). The notion of an $s(n)$-separator tree of a graph associated with an h.p.d. or r.s.p.d. matrix was originally introduced by Lipton, Rose and Tarjan, in order to reduce the number of operations required by the computation of the *Cholesky factorization*:

$$PMP^T = LDL^T, \tag{3.10}$$

where P denotes a pemutation matrix, defining the ordering of variable elimination. Recall that the *Cholesky factorization* is well defined for any h.p.d. or r.s.p.d. matrix (see [GL] or [DB]), and, since the matrix PMP^T is h.p.d. or r.s.p.d. for any h.p.d. or r.s.p.d. matrix M, the factorization (3.10) is well defined. Once a factorization of the format (3.10) has been computed, the asymptotic computational complexity of solving the linear system $Mx = b$ is bounded from above by the asymptotic complexity of solving a triangular linear system.

When the matrix M is a dense matrix, the computation of the factorization (3.10) generally requires $O(n^3)$ field operations. However, when the matrix M is sparse, a "good" choice for P can considerably improve the overall number of required operations, by reducing the *fill-in* (that is, the number of nonzero entries in L and L^T that appear in the positions where M contains zeros), provided that the graph associated with the matrix M is $s(n)$-separatable. Indeed, when an $s(n)$-separator tree of $G(M)$ is available, we may apply the ordering policy of [LRT79]. This ordering starts with the variables corresponding to the vertices in the leaf nodes. It continues, bottom up, with the variables corresponding to the vertices in the nodes of the preceding level, and so on, and ends with the variables corresponding to the vertices in the root node. For example, according to the policy of [LRT79], an ordering of vertices defined by the $s(n)$-separator tree of Figure 3.3 could be (1, 2, 5, 6, 9, 10, 13, 14, 3, 4, 11, 12, 7, 8). (Note that this ordering is not unique).

Hereafter, given an $s(n)$-separator tree $\mathcal{T}(G)$ of a graph G, we write $P = P_{\mathcal{T}(G)}$, to denote the permutation matrix corresponding to the ordering of [LRT79], where all the vertices at each level of $\mathcal{T}(G(M)$ are sorted in the ascending order (as is the case for the ordering defined above). Our algorithm does not utilize the Cholesky factorization of [LRT79]. Instead, given an input matrix A, we will rely on a simplified version of a recursive block matrix $s(n)$-factorization by Pan and Reif (see [PR85], [PR93], [P93]). The

latter factorization is valid if the graph $G(A)$ of the input matrix A is $s(n)$-separatable (with the assumption that an $s(n)$-separator tree of the graph is readily available). By specializing this decomposition to the particular case of an h.p.d. or r.s.p.d. block tridiagonal matrix A, we obtain the sequence of matrices $A_0, A_1, \ldots, A_r$, hereafter referred to as *recursive $s(n)$-factorization of A relative to P*:

$$A_0 = PAP^T; \tag{3.11}$$

$$A_h = \begin{pmatrix} X_h & Y_h^T \\ Y_h & Z_h \end{pmatrix}, \quad A_{h+1} = Z_h - Y_h X_h^{-1} Y_h^T, \quad h = 0, 1, \ldots, r-1, \tag{3.12}$$

where X_h is an $n_h \times n_h$ *h.p.d.* or *r.s.p.d.* block *diagonal* matrix with $2m \times 2m$ blocks, $n_h = 2^{r-h}$. Note that each block of X_h is formed by intersection of the two block rows (k and $k+1$) and two block columns (k and $k+1$), where $\{k, k+1\}$ is a node at level $r-h$ of the $s(n)$-separator tree $\mathcal{T}(G(A))$, which defines the vertex ordering represented by P. The above factorization is induced by the following matrix identities:

$$A_h = \begin{pmatrix} I & 0 \\ Y_h X_h^{-1} & I \end{pmatrix} \begin{pmatrix} X_h & 0 \\ 0 & A_{h+1} \end{pmatrix} \begin{pmatrix} I & X_h^{-1} \\ 0 & I \end{pmatrix}, \tag{3.13}$$

$$A_h^{-1} = \begin{pmatrix} I & -X_h - 1 Y_h^T \\ 0 & I \end{pmatrix} \begin{pmatrix} X_h^{-1} & 0 \\ 0 & A_{h+1}^{-1} \end{pmatrix} \begin{pmatrix} I & 0 \\ -Y_h X_h^{-1} & I \end{pmatrix}. \tag{3.14}$$

We refer the reader to [PR85], [PR93], and [P93] on the many further aspects and details of this algorithm, whose simple specialization to the block triadiagonal case supports the *deterministic* computational complexity cost

bound

$$O_A\left((log\ (\frac{n}{m})t_{I(m)},\ \frac{\frac{n}{m}}{log\ \frac{n}{m}}p_{I(m)}\right), \tag{3.15}$$

provided that the input matrix A has size $n \times n$, has a bandwidth m and is nonsingular. Moreover, this algorithm can be extended to accelerate the computation of the solution of several linear systems having the same banded coefficient matrix; at a preprocessing stage called *preprocessing-nd*, we compute an $s(n)$-separator tree associated with the graph of the input matrix A, at the overall computational cost bound (3.15); after that, we solve any linear system having coefficient matrix A, at optimum *deterministic* cost bounded by

$$O_A\left((log\ n)(log\ m),\ \frac{nm}{(log\ n)(log\ m)}\right), \tag{3.16}$$

which implies the bound

$$O_A\left((log\ n)(log\ m),\ \frac{snm}{(log\ n)(log\ m)}\right) \tag{3.17}$$

on solving s linear equations having the same coefficient matrix A.

3.5 Banded *LIN·SOLVE*: Statement of the Problem

Our first focus in this section is the computational problem $LIN \cdot SOLVE$ (see Problem 2.1.1), in which A is a nonsingular $n \times n$ banded matrix, with bandwidth m. Let us assume that all the diagonal entries are nonzeros and scale the equations so as to turn all these entries into -1. In this case the linear system can be interpreted as a generalized linear recurrence:

$$x_i = \sum_{j=i-m_-}^{i-1} a_{ij}x_j + \sum_{j=i+1}^{i+m_+} a_{ij}x_j + b_i, \quad i = 1, \ldots, n,$$

where each terms x_i is a linear combination of its m_- preceding terms x_j, $j = i - m_-, \ldots, i-1$, and its m_+ subsequent terms x_j, $j = i+1, \ldots, i+m_+$.

The nonsingularity assumption of $LIN \cdot SOLVE$ can be verified by solving any of the two following problems (also of independent interest):

$DET = D(n, m)$: Given an $n \times n$ matrix A, with bandwidth m, compute its determinant, $\det A$;

$|DET|^2 = |D(n, m)|^2$: For an $n \times n$ matrix A, defined in a field of characteristic 0 and having the bandwidth m, compute $|\det A|^2$.

Frequently, one has to solve several linear systems $Ax = b(i)$ for the same nonsingular banded matrix A and for several vectors $b(i)$. The problem can be solved in two stages:

1. $PREPROCESS = P(n, m)$: given a nonsingular $n \times n$ matrix A with bandwidth m, compute a set Γ of parameters, implicitly defining the inverse matrix A^{-1} and independent of the vector $b(i)$.

2. *BACK · SOLVE* $= B(n, m)$: given a nonsingular $n \times n$ matrix A with bandwidth m, a vector $b(i)$, and the set Γ of parameters output by *PREPROCESS*, $P(n, m)$, compute the vector $A^{-1}b(i)$.

The current record asymptotic estimates on the sequential arithmetic time complexity of both $LS(n,m)$ and $D(n,m)$ are $\Omega(nm)$ and $O(nM(m)/m)$, where $M(m) = o(m^{2.376})$. Here $\Omega(nm)$ is the information lower bound, which follows since the solution depends on $\Omega(nm)$ input values, and $O(nM(m)/m)$ is an upper bound supported by the block Gaussian elimination algorithm, where the blocks are of the size $O(m) \times O(m)$. One needs to apply some techniques of symmetrization or randomization for avoiding the inversion of some singular blocks in the block elimination (see section 4.6 and the end of this section). Symmetrization enables us to solve the problem of singular blocks over for computations over a field of constants of characteristic 0, randomization for computations over an arbitrary field.

In the next section we will recall the block cyclic reduction (BCR) algorithm for block triadiagonal linear system (thus covering the solution of banded *LIN · SOLVE*, due to Proposition 2.2.1) and we will show its close correlation to the nested dissection (ND) algorithm and to the partial-cascade-sum algorithm. In this section we will compare the arithmetic computational cost of this algorithm with the preceeding records of [E] obtained

for banded linear systems without using the correlation with nested dissection ND and related techniques. To the advantage of the approach of [E] over the ND and BCR algorithms, it solves the banded linear systems over any field of constants (using randomization if the field has a positive characteristic). The algorithm of [E] supports the following bounds on the randomized parallel complexity of $LIN \cdot SOLVE$ and DET over any field of constants:

$$\begin{aligned}
\left(t_{D(n,\,m)},\, p_{D(n,\,m)}\right) &= O_A\left(T^*(n),\, P^*(n,\,m)\right),\\
\left(t_{LS(n,\,m)},\, p_{LS(n,\,m)}\right) &= O_A\left(T^*(n),\, P^*(n,\,m)\right),\\
T^*(n) &= (\log^3 n)\psi(n)t_{I(n)},\\
P^*(n,\,m) &= \left(\frac{n}{m}\right)p_{I(m)}\log^{O(1)} n,\\
\psi(n) &= \begin{cases} 1, & \text{if } |\mathbf{F}| \geq n;\\ \left(1+\log\log_{|\mathbf{F}|} n\right)\log_{|\mathbf{F}|} n, & \text{otherwise.}\end{cases}
\end{aligned}$$

On the other hand, by using the fact that for any banded matrix A having bandwidth m, a family of $O(m)$-separators for a graph associated with A is readily available (see definition 3.2.11), we arrive at the bound $O_A((\log n)t_{I(m)}, p_{I(m)}n/\log n)$ for $LIN \cdot SOLVE$ over the fields of characteristic 0, by applying the parallel nested dissection algorithm of [GLi], [GT] (also compare [PR85, 93], [P93]). Yet another alternative derivation of the

same asymptotic complexity estimate for $LIN \cdot SOLVE$ over the field of characteristic zero relies on the well-known block cyclic reduction reduction algorithm, which we recall in our next section. Shifting from A and DET to $A^H A$ and $|DET|^2$ enables us to relax the assumption about strong non-singularity of A, for computations over any field of characteristic zero in which the Hermitian transpose A^H of a matrix A is readily available. This follows from Propositions 2.1.1, 2.3.1 and the equations $A^{-1} = (A^H A)^{-1} A^H$, $\det(A^H A) = |\det A|^2$. To relax the strong nonsingularity assumption in the case of *any* field of constants, we apply an alternative argument based on randomization (see section 4.6).

3.6 The Block Cyclic Reduction Algorithm.

Consider a $k \times k$ block tridiagonal matrix with $m \times m$ blocks,

$$A = \begin{pmatrix} A_0 & B_0 & 0 & \dots & & 0 \\ C_1 & A_1 & B_1 & \dots & & 0 \\ 0 & C_2 & A_2 & B_2 & & \vdots \\ \vdots & \vdots & & \ddots & \ddots & \\ 0 & 0 & \dots & & A_{k-2} & B_{k-2} \\ 0 & 0 & \dots & & C_{k-1} & A_{k-1} \end{pmatrix}. \tag{3.18}$$

We next recall the classical block cyclic reduction algorithm ([H], [BGN], [GL]), for a linear system $Ax = b$, defined over a field of characteristic 0.

We assume nonsingularity of the diagonal blocks of the matrix A and of the smaller coefficient matrices (such as $A^{(1)}$ below) of the auxiliary linear systems that arise in the recursive applications of the block cyclic reduction. This restriction can be relaxed by means of the symmetrization techniques (see the end of section 3.5 and remark 3.6.1).

We now present the block cyclic reduction (BCR) algorithm, which applies to a block tridiagonal linear system of k block equations with $m \times m$ blocks.

ALGORITHM **3.6.1** (k, m)-block cyclic reduction*:*

- **Input:** *A nonsingular $k \times k$ block tridiagonal matrix A with nonsingular $m \times m$ subdiagonal blocks, defined as in (3.18), and a vector $b = (b_0^T, \ldots, b_{k-1}^T)^T$, such that $Ax = b$.*

- **Output:** *Vector $x = (x_0^T, \ldots, x_{k-1}^T) = A^{-1}b$ from the linear system $Ax = b$.*

Initiation:

Set $C_0 = B_{k-1} = 0, \quad x_{-1} = x_k = 0.$

Computation:

- **Stage 1.** *Rewrite the linear system $Ax = b$ in the block form:*

$$C_i x_{i-1} + A_i x_i + B_i x_{i+1} = b_i, \quad i = 0, 1, \ldots, k-1, \tag{3.19}$$

where x_i, b_i are m-dimensional vectors.

- **Stage 2.** *Compute the inverses of the matrices A_{2j} for all j, the $k_1 \times k_1$ block matrix*

$$A^{(1)} = \begin{pmatrix} A_0^{(1)} & B_0^{(1)} & 0 & \dots & & 0 \\ C_1^{(1)} & A_1^{(1)} & B_1^{(1)} & \dots & & 0 \\ 0 & C_2^{(1)} & A_2^{(1)} & B_2^{(1)} & & \vdots \\ \vdots & \vdots & \ddots & \ddots & \ddots & \\ 0 & 0 & \dots & & A_{k_1-2}^{(1)} & B_{k_1-2}^{(1)} \\ 0 & 0 & \dots & & C_{k_1-1}^{(1)} & A_{k_1-1}^{(1)} \end{pmatrix}, \tag{3.20}$$

and the k_1-dimensional vector

$$b^{(1)} = \left(b_j^{(1)}, \quad j = 0, \dots, k_1 - 1 \right), \tag{3.21}$$

where

$$\begin{aligned} A_j^{(1)} &= A_{2j+1} - C_{2j+1} A_{2j}^{-1} B_{2j} - \\ &\qquad B_{2j+1} A_{2j+2}^{-1} C_{2j+2}, && (3.22) \\ B_j^{(1)} &= -B_{2j+1} A_{2j+2}^{-1} B_{2j+2}, && (3.23) \\ C_j^{(1)} &= -C_{2j+1} A_{2j}^{-1} C_{2j}, && (3.24) \\ b_j^{(1)} &= b_{2j+1} - C_{2j+1} A_{2j}^{-1} b_{2j} - \\ &\qquad B_{2j+1} A_{2j+2}^{-1} b_{2j+2}, && (3.25) \end{aligned}$$

$$j = 0, 1, \dots, k_1 - 1; \quad k_1 = \lfloor k/2 \rfloor.$$

- **Stage 3.** *Solve the linear system*

$$A^{(1)} x^{(1)} = b^{(1)}, \tag{3.26}$$

that is, find the subvector $x^{(1)} = \left(x^{(1)}_{2j+1}, \quad j = 0, 1, \ldots\right)$ *of the solution vector* x *for the original system* $Ax = b$.

- **Stage 4.** *Substitute the vector* $x^{(1)}$ *into the linear system (3.19) and compute the remaining components of the vector* x.

end

To solve the linear system (3.26) at stage 3 one should recursively apply Algorithm 3.6.1 to this system until arriving at a system with the matrix of a size at most $m \times m$. Then one may solve this system as a general linear system of m equations and extend the solution to the evaluation of the vector x.

The computation of the matrices $A^{(1)}$ and A_{2j}^{-1}, for all j, at stage 2, will be called *preprocessing-bcr*; it can be performed at a cost bounded by $O_A(t_{l(m)}, kp_{l(m)})$. The vector $b^{(1)}$ can be computed (after preprocessing-bcr), at a cost bounded by

$$O_A(\log m, m^2 k/\log m). \tag{3.27}$$

The computation of the remaining components of the vector x, after the substitution of the vector $x^{(1)}$ into the linear system (3.19), at stage 4, can be done at a cost bounded by (3.27).

Summarizing and using the B-principle and Proposition 4.2.2, we obtain

THEOREM **3.6.1** *The block tridiagonal linear system* $Ax = b$ *of (3.19) can be solved by means of (the block cyclic reduction) Algorithm 3.6.1 provided*

that all the auxiliary diagonal blocks that appear in the recursive process of the application of this algorithm are nonsingular. The computational cost of this solution is bounded by:

a) $O_A\left((\log k)t_{I(m)},\ kp_{I(m)}/\log k\right)$, *at all stages independent of the vector b (and called preprocessing), and*

b) $O_A\left((\log k)(\log m),\ \frac{km^2}{(\log k)(\log m)}\right)$, *at all other stages (called backsolving).*

The same estimates, for $k = n/m$, *hold for the solution of the linear system* $Ax = b$ *with a nonsingular* $n \times n$ *matrix* A *having bandwidth* $m(A) \leq m$. *The assumption about nonsingularity of the diagonal blocks can be relaxed, over any field of characteristic 0, by means of the transition from the linear system* $Ax = b$ *to* $A^H Ax = A^H b$, *and from* $|\det A|$ *to* $\det(A^H A) = |\det A|^2$.

As we have already mentioned in the introduction, the block cyclic reduction amounts to block Gaussian elimination, for some appropriate block elimination ordering. Therefore, the block cyclic reduction defines a block factorization of A into a product of block triangular matrices. Consequently, *det* A equals the product of the determinants of the diagonal blocks of these triangular matrices and of the determinants of the permutation matrices involved; each of the latter determinants equals 1 or -1 and is readily available. From these observations we obtain that the bound of part a) of Theorem 3.6.1 also applies to the evaluation of det A.

REMARK **3.6.1** The matrix $A^{(1)}$ is h.p.d. if A is h.p.d. (see [GL, p.140]).

REMARK **3.6.2** A simple inspection reveals that the block cyclic redution algorithm (which is closely related to the partial-cascade-sum algorithm) is identical with the nested dissection [GLi], [GT] applied to $LIN \cdot SOLVE$ for banded matrices. To observe the equivalences of the two methods, recall that both nested dissection and block cyclic reduction amount to Gaussian elimination with specific elimination ordering. Therefore, we only need to compare this ordering for the two methods. In the case of a tridiagonal linear system, the associated vertex graph for the nested dissection is given by the line with n consecutive vertices. The nested dissection immediately leads to the eliminations of all the even numbered vertices-variables, which is exactly the ordering also defined by the cyclic reduction. In the block tridiagonal case, the elimination process is the same, except that a block of vertices-variables replaces each single vertex.

The time bounds singular of part a) of Theorem 3.6.1 matches ones of (4.1) (see next chapter) and the processor bound of part a) of Theorem 3.6.1 improves one of (4.1) by a logarithmic factor. Note, however, that currently we only know how to ensure performing the block cyclic reduction over fields of characteristic 0, whereas (4.1) is proved over any field. Moreover, the preprocessing stage of the block cyclic reduction outputs information which is as large as the input information, thus doubling the entire storage space. The storage space can be actually decreased back to the original size (but at the expense of increasing the processor bound by a logarithmic factor) if

one overwrites the original matrix A by the two input matrices $A^{(1)}$ and $A^{(2)}$ obtained by excluding the odd-indexed block equations of Theorem 3.6.1 and then recursively applies the same process to *both* matrices $A^{(1)}$ and $A^{(2)}$.

3.7 Summary.

Comparison of the two latter algorithms, that is, the nested dissection algorithm and the block cyclic reduction algorithm, reveals that they are very close to each other in the case of banded linear systems and can be made identical by means of an appropriate choice of a separator family for the nested dissection and of block sizes for the block cyclic reduction. The block cyclic reduction is also equivalent to the partial-cascade-sum algorithm.

Chapter 4

Alternative Methods for Solving Banded Linear Systems.

4.1 Introduction.

The ND/BCR algorithm effectively solves banded linear systems over any field of constants of characteristic 0, but it does not generally work over a field of a positive characteristic. In most of the applications, the solution of banded block tridiagonal linear systems is sought over fields of characteristic zero, but the extension to any field is needed to complete theoretical study of banded linear systems. More important, alternative algorithms which would be equally or almost equally fast and work efficient are desirable because the ND/BCR solution of some block tridiagonal linear systems is prone to numerical stability problems. Finally, a practical deficiency of the ND/BCR solution of block tridiagonal linear systems is some extra storage space needed for the information that is required in order to recover the odd indexed variables by back substitution at the final stage of the computation. Some recipes for

repairing the latter deficiency are given in [BMP], but in this chapter we propose alternative approaches to ND/BCR that address all the listed problems. Besides, we accelerate the parallel solution (preserving its work optimality) for a large class of banded linear systems (that is, for ones whose coefficient matrix has a lower or upper edge, compare Definition 2.3.2) and solve the problem in the singular case. We also extend our parallel algorithms to computing the determinant of a banded matrix. Like the ND/BCR algorithm, all our algorithms are in NC or RNC and support the current record bounds on the parallel time complexity of above computations; simultaneously the bounds on their *potential work* (the product of time and processor bounds) match (within constant or logarithmic factors) the record sequential time bounds for the same computations; moreover, these algorithms are in NC^1 or RNC^1 if the bandwidth is a constant.

We develop two main approaches. In the first one (presented in sections 4.3– 4.6), we reach the Las-Vegas randomized bounds

$$O_A\left(\left(\log\frac{n}{m}\right)t_{I(m)},\ \left(\frac{n}{m}\right)p_{I(m)}\right) \tag{4.1}$$

for $LIN \cdot SOLVE$ and

$$O_A\left(\left(\log\frac{n}{m}\right)(t_{I(m)}+t_{D(m)}),\ \frac{\left(\frac{n}{m}\right)(p_{D(m)}+p_{I(m)})}{\log\frac{n}{m}}\right) \tag{4.2}$$

for DET (in both cases over any field of constants). (4.1) and (4.2) imply that our parallel algorithms for $LIN \cdot SOLVE$ (respectively, DET) are in RNC

(and even in RNC^1 if m is a constant) and are *work efficient* (respectively, *work optimum*) according to the Definition 2.2.4.

Moreover, as in Theorem 3.6.1, for the ND/BCR algorithm for block tridiagonal linear systems, if we solve more than an order of m linear systems $Ax(i) = b(i)$ with the same $n \times n$ matrix A having bandwidth m, we obtain a further improvement: We solve $PREPROCESS$ at the same randomized Las-Vegas cost (4.1), and we give a solution of $BACK \cdot SOLVE$ at deterministic cost

$$O_A\left((\log n)(\log m), \frac{mn}{(\log n)(\log m)}\right). \tag{4.3}$$

To obtain these improvements, we applied several techniques. In particular, we introduced special preprocessing based on a 2×2 block factorization of the banded input matrix A and on utilizing some auxiliary matrices, such as $I_F(n, m)$, $I_L(n, m)$ defined in section 4.2, which helped us to reveal and to exploit the sparsity structure of the input matrix A. In addition, we achieved the required nonsingularity of the auxiliary block matrices at a low computational cost, due to using symmetrization and randomization. Our techniques can also be combined with the 3×3 factorization of [E] to obtain the same asymptotic estimates for the arithmetic complexity of $LIN \cdot SOLVE$ and DET in the banded case (see [PSA]), although the latter approach leads to a little more complicated code, and over arbitrary fields it requires to involve larger random matrices. Under some mild additional assumptions, we obtain further results:

a) We deterministically obtain (4.1) for $LIN \cdot SOLVE$ and (4.2) for $|DET|^2$ over the fields of characteristic zero in which the Hermitian transpose $A^* = A^H$ of a matrix A is readily available.

b) The problem $LIN \cdot SOLVE$ for a block bidiagonal linear system (which includes the banded triangular case) can be solved, by means of a variant of the block cyclic reduction algorithm, at deterministic computational cost bounded (over any field of constants) by

$$O_A\left(t_{I(m)} + (log\ \frac{n}{m})(log\ m),\ \frac{(\frac{n}{m})p_{I(m)}\ t_{I(m)}}{t_{I(m)} + (log\ \frac{n}{m})(log\ m)}\right), \tag{4.4}$$

at the preprocessing stage, and by

$$O_A\left(\left(\log \frac{n}{m}\right)(\log m),\ \frac{mn}{\left(\log \frac{n}{m}\right)(\log m)}\right), \tag{4.5}$$

at the subsequent backsolving stage.

c) We extend the latter deterministic solution of the block bidiagonal linear systems to the deterministic solution, over any field of constants, for the problems $LIN \cdot SOLVE^*$, $PREPROCESS^*$ and $BACK \cdot SOLVE^*$, which denote $LIN \cdot SOLVE$, $PREPROCESS$ and $BACK \cdot SOLVE$, respectively, in the special case where the banded input matrix A has a lower and/or an upper edge (compare Definition 2.3.2). The latter requirement holds for a large class of banded matrices; in particular, it typically holds for the matrices encountered in applications to $PDE's$ and $ODE's$ (see [A] and [LP]). Our extension leads to

deterministic solution algorithms that perform at a computational cost bounded by

$$O_A\left(\left(\log\frac{n}{m}\right)(\log m)+t_{I(m)},\ \left(\frac{n}{m}\right)\frac{p_{I(m)}t_{I(m)}}{\left(\log\frac{n}{m}\right)(\log m)+t_{I(m)}}\right), \quad (4.6)$$

for $PREPROCESS^*$ and $LIN\cdot SOLVE^*$, and (4.5), for $BACK\cdot SOLVE^*$. [Note an improvement against (4.1).]

In section 4.9, we show an extension of (4.5) and (4.6) to the complexity estimates for the approximate solution of banded linear systems over the fields of characteristic 0 and for the exact solution over the rationals.

We organize the chapter in the following order: after some definitions and other preliminaries presented in section 4.2, we preprocess a strongly nonsingular input matrix A in section 4.3. We use the results of this preprocessing for the solution of $BACK\cdot SOLVE$ in section 4.4 and DET in section 4.5. In section 4.6, we relax the strong nonsingularity condition imposed on the input matrix A in sections 4.3–4.5. In section 4.7, we treat the block bidiagonal case, and we solve $PREPROCESS^*$ and $BACK\cdot SOLVE^*$ in section 4.8. In section 4.9, we show a reduction of $LIN\cdot SOLVE$ to $LIN\cdot SOLVE^*$. Sections 4.3-4.9 follow [PSA]. In section 4.10 we present efficient parallel algorithms of [E95] for singular banded matrix computations over arbitrary fields.

4.2 Definitions and Auxiliary Results

Hereafter, 0 denotes the null matrices, I_q the $q \times q$ identity matrix, $I_F(k,\ q)$ $(k \geq q)$ the $k \times k$ matrix $\begin{pmatrix} I_q & 0 \\ 0 & 0 \end{pmatrix}$, $I_L(k,\ q)$ $(k \geq q)$ the $k \times k$ matrix $\begin{pmatrix} 0 & 0 \\ 0 & I_q \end{pmatrix}$, $\operatorname{diag}(U, V)$ the matrix $\begin{pmatrix} U & 0 \\ 0 & V \end{pmatrix}$.

PROPOSITION **4.2.1** *Let $q, r, s, r_1, s_1, r_2, s_2$ be seven positive integers such that $r = r_1 + r_2$, $s = s_1 + s_2$, $q \leq r$, $q \leq s$; B be a $q \times q$ matrix, $W = \begin{pmatrix} W_{11} & W_{12} \\ W_{21} & W_{22} \end{pmatrix}$ be an $r \times r$ matrix, where W_{11} is an $r_1 \times r_1$ matrix; then*

$$I_F(r, r-1)W = \begin{pmatrix} W_{11} & W_{12} \\ 0 & 0 \end{pmatrix}, \quad I_L(r, r-2)W = \begin{pmatrix} 0 & 0 \\ W_{21} & W_{22} \end{pmatrix},$$

$$WI_F(s, s-1) = \begin{pmatrix} W_{11} & 0 \\ W_{21} & 0 \end{pmatrix}, \quad WI_L(s, s-1) = \begin{pmatrix} 0 & W_{12} \\ 0 & W_{22} \end{pmatrix},$$

$$I_F(r, r-1)WI_F(s, s-1) = \begin{pmatrix} W_{11} & 0 \\ 0 & 0 \end{pmatrix}, \quad I_F(r, r-1)WI_L(s, s-2) = \begin{pmatrix} 0 & W_{12} \\ 0 & 0 \end{pmatrix},$$

$$I_L(r, r-2)WI_F(s, s-1) = \begin{pmatrix} 0 & 0 \\ W_{21} & 0 \end{pmatrix}, \quad I_L(r, r-2)WI_L(s, s-2) = \begin{pmatrix} 0 & 0 \\ 0 & W_{22} \end{pmatrix}.$$

Moreover, If the matrices $(B \quad 0)$ and $(0 \quad B)$ [respectively, $\begin{pmatrix} B \\ 0 \end{pmatrix}$ and $\begin{pmatrix} 0 \\ B \end{pmatrix}$] have size $q \times r$ [respectively, $s \times q$], then

$$(B \quad 0) = (B \quad 0)I_F(r,\ q), \qquad (0 \quad B) = (0 \quad B)I_L(r,\ q),$$

,

$$\begin{pmatrix} B \\ 0 \end{pmatrix} = I_F(s,\ q)\begin{pmatrix} B \\ 0 \end{pmatrix}, \qquad \begin{pmatrix} 0 \\ B \end{pmatrix} = I_L(s,\ q)\begin{pmatrix} 0 \\ B \end{pmatrix}.$$

PROPOSITION **4.2.2** *A $km \times km$ matrix A, with $m_-(A) < m$, $m_+(A) < m$, can be represented as a $k \times k$ block tridiagonal matrix with $m \times m$ blocks. For any $k \times k$ block tridiagonal matrix A, with $m \times m$ blocks, we have $m_-(A) \leq 2m-2$ and $m_+(A) \leq 2m-2$.*

4.3 Preprocessing for the Solution of a Banded Linear System (using a 2×2 block decomposition of the input matrix).

Hereafter, A denotes an $n \times n$ banded matrix having bandwidth m. We will call the blocks in the 2×2 block representation

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \tag{4.7}$$

balanced if the matrices A_{11} and A_{22} have sizes $n_1 \times n_1$ and $n_2 \times n_2$, respectively, with $n_1 = \lceil n/2 \rceil$, $n_2 = n - n_1$. Until section 4.6, we assume that A is strongly nonsingular, so that A and A^{-1} have the factorizations

$$A = \begin{pmatrix} I_{n_1} & 0 \\ A_{21}A_{11}^{-1} & I_{n_2} \end{pmatrix} \begin{pmatrix} A_{11} & 0 \\ 0 & S \end{pmatrix} \begin{pmatrix} I_{n_1} & A_{11}^{-1}A_{12} \\ 0 & I_{n_2} \end{pmatrix}, \tag{4.8}$$

$$A^{-1} = \begin{pmatrix} I_{n_1} & -A_{11}^{-1}A_{12} \\ 0 & I_{n_2} \end{pmatrix} \begin{pmatrix} A_{11}^{-1} & 0 \\ 0 & S^{-1} \end{pmatrix} \begin{pmatrix} I_{n_1} & 0 \\ -A_{21}A_{11}^{-1} & I_{n_2} \end{pmatrix}, \tag{4.9}$$

$$S = A_{22}Y, \tag{4.10}$$

$$Y = I_{n_2} - A_{22}^{-1}A_{21}A_{11}^{-1}A_{12}, \tag{4.11}$$

where S is called the Schur complement of A_{11} in A, and S^{-1} is a block of A^{-1}.

Note that nonsingularity of Y follows from nonsingularity of the matrices A, A_{11} and A_{22}.

PROPOSITION **4.3.1** *If the first m and the last m columns of the matrices A_{11}^{-1} and A_{22}^{-1} are known, then the matrix Y can be computed at a cost bounded by*

$$O_A\left(t_{M(n/2,m,m)}, p_{M(n/2,m,m)}\right), \tag{4.12}$$

and the matrix Y^{-1} can be computed at a cost bounded by

$$O_A\left(t_{I(m)} + t_{M(n/2,m,m)}, \frac{t_{I(m)}p_{I(m)} + p_{M(n/2,m,m)}t_{M(n/2,m,m)}}{t_{I(m)} + t_{M(n/2,m,m)}}\right). \tag{4.13}$$

PROOF. Observe that the matrix A_{12} has the form $\begin{pmatrix} 0 & 0 \\ L & 0 \end{pmatrix}$ and A_{21} has the form $\begin{pmatrix} 0 & U \\ 0 & 0 \end{pmatrix}$, where L and U are $m \times m$ matrices, and apply Proposition 4.2.1 to deduce that

$$A_{12} = I_L(n_1,m)A_{12} = A_{12}I_F(n_2,m) = I_L(n_1,m)A_{12}I_F(n_2,m), \tag{4.14}$$

$$A_{21} = I_F(n_2,m)A_{21} = A_{21}I_L(n_1,m) = I_F(n_2,m)A_{21}I_L(n_1,m). \tag{4.15}$$

Then combine (4.14), (4.15) and (4.11) to deduce that

$$Y = I_{n_2} - A_{22}^{-1}I_F(n_2,m)A_{21}I_L(n_1,m)A_{11}^{-1}I_L(n_1,m)A_{12}I_F(n_2,m). \tag{4.16}$$

The latter equation implies the bound (4.12) on the cost of computing Y. It also implies that Y is a block triangular matrix of the following format

where Y_{11} is an $m \times m$ matrix:

$$Y = \begin{pmatrix} Y_{11} & 0 \\ Y_{21} & I_{n_2-m} \end{pmatrix}. \tag{4.17}$$

Since Y is nonsingular, it follows that the block Y_{11} is nonsingular, and from (4.17), we obtain that

$$Y^{-1} = \begin{pmatrix} Y_{11}^{-1} & 0 \\ -Y_{21}Y_{11}^{-1} & I_{n_2-m} \end{pmatrix}. \tag{4.18}$$

This immediately implies the bound (4.13) on the cost of the computation of Y^{-1}. (Note that only the first m columns of Y and of Y^{-1} need be computed.)

■

DEFINITION **4.3.1** Let A be a strongly nonsingular $n \times n$ matrix and recursively define

$$A_{i_1 i_1 \ldots i_k i_k} = \begin{cases} A, & k = 0, \\ (A_{i_1 i_1 \ldots i_{k-1} i_{k-1}})_{i_k i_k} & k > 0, \end{cases} \tag{4.19}$$

where $i_k = 1, 2$, $C_{i_k i_k}$ denotes the two diagonal blocks [that is, the blocks (1,1) and (2,2)] in the balanced 2×2 representation of the matrix C. Then recursively define the set of matrices

$$\begin{aligned} \Gamma_2(A) &= \begin{cases} A^{-1} & \text{if } n < 2m; \\ \gamma_2(A) \cup \Gamma_2(A_{11}) \cup \Gamma_2(A_{22}) & \text{otherwise;} \end{cases} \\ \gamma_2(A) &= \{A^{-1}I_F(n, m), A^{-1}I_L(n, m), Y^{-1}\}. \end{aligned} \tag{4.20}$$

Define *preprocessing-2* of matrix A [based on 2×2 block decompositions (4.8)-(4.11)] as the computation of the set $\Gamma_2(A)$. (In section 4.4, we will prove that $\Gamma_2(A)$ is a solution to the *PREPROCESS* problem for the matrix A.)

THEOREM **4.3.1** *The complexity of preprocessing-2 is bounded by (4.1).*

PROOF. Expand (4.9) and then apply (4.14), (4.15) and Proposition 4.2.1 to deduce that

$$A^{-1}I_F(n,\,m) \;=\; \begin{pmatrix} A_{11}^{-1}I_F(n_1,m) + A_{11}^{-1}A_{12}Y^{-1}A_{22}^{-1}A_{21}A_{11}^{-1}I_F(n_1,m) & 0 \\ -Y^{-1}A_{22}^{-1}A_{21}A_{11}^{-1}I_F(n_1,m) & 0 \end{pmatrix}.$$

$$\begin{aligned} A^{-1}I_L(n,\,m) \;&=\; \begin{pmatrix} 0 & -A_{11}^{-1}A_{12}Y^{-1}A_{22}^{-1}I_L(n_2,m) \\ 0 & Y^{-1}A_{22}^{-1}I_L(n_2,m) \end{pmatrix} \\ &=\; \begin{pmatrix} 0 & -A_{11}^{-1}I_L(n_1,m)A_{12}Y^{-1}A_{22}^{-1}I_L(n_2,m) \\ 0 & Y^{-1}A_{22}^{-1}I_L(n_2,m) \end{pmatrix}. \end{aligned} \tag{4.21}$$

Let $t_1(n,\,m)$ denote the number of parallel steps and let $p_1(n,\,m)$ denote the number of processors needed for the computation of $\Gamma_2(A)$. Combining the recursive definition (4.20) and the recursive formulae (4.21) and (4.21) with Proposition 4.3.1 leads to the following recursive estimates, where c_1 and c_2 are two constants:

$$t_1(n,\,m) \;\le\; \begin{cases} t_{I(m)}, & n \le 2m \\ t_1(n/2,m) + c_1 t_{I(m)} + c_2 t_{M(n/2,\,m,\,m)}, & \text{otherwise,} \end{cases}$$

$$p_1(n,\,m) \;\le\; \begin{cases} p_{I(m)}, & n \le 2m \\ \max(2p_1(n/2,m),\; p_{I(m)},\; p_{M(n/2,\,m,\,m)}), & \text{otherwise.} \end{cases}$$

Recursive application of the latter relations and the B-principle yield the desired complexity bound (4.1). ∎

REMARK **4.3.1** At each parallel step, the computation of each triple in the set $\Gamma_2(A)$ only involves the results obtained at the previous recursive step. Therefore, we may overwrite these previous results by the current ones, to keep the storage space complexity for the entire preprocessing computation bounded by $O(mn)$. Such a possibility of overwriting the input at each stage, in contrast to the ND/BCR algorithm for banded linear systems, actually enables us to perform our algorithm by using half of the space used in the ND/BCR algorithm, which in practice may more than compensate the user for a very minor slowdown of the algorithm of this section versus the ND/BCR algorithm.

4.4 Solving a Preprocessed Banded Linear System.

THEOREM **4.4.1** *If the set $\Gamma_2(A)$ output by preprocessing-2 is available, then the complexity of the subsequent computation of $A^{-1}b$, for a given vector b, is bounded by (4.3).*

PROOF. Let $b = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$, where b_1 and b_2 are two vectors of dimensions n_1 and n_2, respectively. Expand (4.9), then multiply both sides by b, and apply (4.14) and (4.15) to deduce that

$$A^{-1}b = \begin{pmatrix} A_{11}^{-1}b_1 + A_{11}^{-1}A_{12}Y^{-1}A_{22}^{-1}A_{21}A_{11}^{-1}b_1 - A_{11}^{-1}A_{12}Y^{-1}A_{22}^{-1}b_2 \\ -Y^{-1}A_{22}^{-1}A_{21}A_{11}^{-1}b_1 + Y^{-1}A_{22}^{-1}b_2 \end{pmatrix} \quad (4.22)$$

Due to preprocessing, the matrices Y^{-1}, $A_{11}^{-1}I_L(n_1,m)$ and $A_{22}^{-1}I_F(n_2,m)$ are available. Therefore, given the vectors $A_{11}^{-1}b_1$ and $A_{22}^{-1}b_2$, the computation of

$A^{-1}b$ only requires a constant number of multiplications of matrices of sizes at most $m \times q$ by vectors of dimensions at most q, where $q \leq \lceil n/2 \rceil$.

Let $t_2 = t_2(n, m)$ denote the number of parallel steps required for the computation of $A^{-1}b$, assuming that preprocessing-2 has been performed, and let $p_2 = p_2(n, m)$ denote the corresponding number of required processors. The above argument leads us to the following estimates for the pair (t_2, p_2), where c is a constant:

$$t_2(n, m) \leq \begin{cases} t_{M(m,m,1)}, & n \leq 2m \\ t_2(n/2, m) + ct_{M(n/2,m,1)}, & \text{otherwise,} \end{cases}$$

$$p_2(n, m) \leq \begin{cases} p_{M(m,m,1)}, & n \leq 2m \\ \max\left(2p_2(n/2, m),\ p_{M(n/2,m,1)}\right), & \text{otherwise.} \end{cases}$$

Recursive application of the latter relations and the B-principle yields the desired complexity bound (4.3). ∎

REMARK **4.4.1** The problem $BACK \cdot SOLVE$, for s linear systems, can be defined by replacing the vectors x and b by $n \times s$ matrices. Due to Remark 4.3.1, the storage space complexity of $LIN \cdot SOLVE$ is thus bounded by $S = S_p + O(nm + ns)$, where S_p bounds the space required for the storage of all the values output at the preprocessing stage, and actually S_p is the dominanting term of S.

4.5 Solving *DET*.

THEOREM **4.5.1** *The complexity of DET is bounded by (4.2).*

PROOF. Deduce from (4.8) and (4.10) that

$$\det A = \det A_{11} \det A_{22} \det Y. \tag{4.23}$$

Moreover, each of the matrices A_{11} and A_{22} has bandwidth at most m. Therefore, solving DET is reduced to two problems of half-size each, at the cost of computing the determinant of the $m \times m$ matrix Y; moreover, due to (4.17), $\det Y = \det Y_{11}$, and

$$I_F(n_2, m) Y I_F(n_2, m) = \operatorname{diag}(Y_{11}, 0). \tag{4.24}$$

We may now define the set

$$\begin{aligned} \Gamma_D(A) &= \begin{cases} \{\det A\} & \text{if } n < 2m; \\ \gamma_D(A) \bigcup \Gamma_D(A_{11}) \bigcup \Gamma_D(A_{22}) & \text{otherwise;} \end{cases} \\ \gamma_D(A) &= \{\det A, Y^{-1}, I_U(n, m) A^{-1} I_V(n, m))\}. \end{aligned} \tag{4.25}$$

where the pair (U, V) takes on all the choices of the pairs (F, F), (F, L), (L, F) and (L, L).

Let $t_3(n, m)$ denote the number of parallel steps and $p_3(n, m)$ the number of processors needed for solving DET. By combining the equations (4.16), (4.21), (4.21) and (4.24), we arrive at the following complexity estimates for the pair $(t_3(n, m), p_3(n, m))$ where $n \geq 2m$:

$$\begin{aligned} t_3(n, m) &\leq t_3(n/2, m) + t_{D(m)} + t_{I(m)} + t_{M(m,m,m)}, \\ p_3(n, m) &\leq \max\left(2p_3(n/2, m), p_{D(m)}, p_{I(m)}, p_{M(m,m,m)}\right). \end{aligned}$$

Due to (1.1)-(1.3), we may ignore the terms $t_{M(m,m,m)}$ and $p_{M(m,m,m)}$. Recursive application of the latter estimates and the B-principle yields the complexity bound (4.2). ∎

4.6 How to Relax the Strong Nonsingularity Assumption.

Our algorithm of section 4.5 (for DET) computes, as a by-product, the determinants of all the matrices $A_{i_1 i_1 \ldots i_k i_k}$, $Y_{i_1 i_1 \ldots i_k i_k}$ [see definition 4.3.1 and the equation (4.19)], unless at least one of these determinants is zero. In the latter case, we just output $\det A = 0$ [this equation follows, due to (4.23) and to its recursive extension]. Otherwise, if $\det A \neq 0$, all the matrices $A_{i_1 i_1 \ldots i_k i_k}$, $Y_{i_1 i_1 \ldots i_k i_k}$ are nonsingular, and our algorithms of sections 4.3 and 4.4 solve $PREPROCESS$ and $BACK \cdot SOLVE$.

Again, we only need to ensure nonsingularity of the matrices A, Y, A_{11}, Y_{11}, A_{22}, Y_{22}, ... Furthermore, since the nonsingularity of $Y_{i_1 i_1 \ldots i_k i_k}$ follows from the nonsingularity of $A_{i_1 i_1 \ldots i_k i_k}$, $A_{i_1 i_1 \ldots i_k i_k 11}$ and $A_{i_1 i_1 \ldots i_k i_k 22}$, it is sufficient to ensure nonsingularity of $A_{i_1 i_1 \ldots i_k i_k 11}$ and $A_{i_1 i_1 \ldots i_k i_k 22}$, assuming that $A_{i_1 i_1 \ldots i_k i_k}$ is nonsingular.

We will next apply randomization to yield nonsingularity of the two diagonal blocks A_{11} and A_{22}, in the balanced 2×2 representation of A (this argument can be extended to all other diagonal blocks as well). More precisely, we will reach our goal (of insuring nonsingularity of the two diagonal blocks) by

shifting from the matrix A to PA, where $P = \text{diag}\left(I_{n_1-m_+(A)}, R, I_{n_2-m_-(A)}\right)$, and R is a random $m \times m$ matrix. This is a valid transition because $\det P$ is readily available, $\det A = \det(PA)/\det P$, and $A^{-1} = (PA)^{-1}P$. Moreover, we immediately verify by inspection that the equations (4.8)–(4.11), (4.14), (4.15) (4.21), (4.21) and, consequently, the proofs of Proposition 4.3.1 and Theorem 4.3.1 are extended in the transition from the matrix A to the matrix PA, as long as the matrix PA is nonsingular, together with $(PA)_{11}$, $(PA)_{22}$, the two principal submatrices of PA, that are the two diagonal blocks of PA in its balanced 2×2 block representation.

We now recursively apply this process of regularization via randomization: first to the two diagonal blocks $(PA)_{11}$ and $(PA)_{22}$, thus obtaining the matrices $B^{(i,j)} = P^{(i,j)}(PA)_{jj}$, $i, j = 1, 2$; then to the two diagonal blocks of $B_{11}^{(i,j)}$ and $B_{22}^{(i,j)}$ of each matrix $B^{(i,j)}$, and so on, until we arrive at the blocks of size less than $3m \times 3m$. Theorem 4.6.1 shows that in every recursive step it suffices to shift from the $k \times k$ input matrix B^* to the matrix P^*B^* where P^* has the format

$$\text{diag}\,(I_i, R^*, I_{k-i-m}), \tag{4.26}$$

provided that $k \geq 3m$, $i = \lceil k/2 \rceil$, and R^* is an $m \times m$ random matrix. Due to the middle position of the blocks R^* and to the bound $k \geq 3m$, the entire regularization process amounts to the transition from the input matrix A to the matrix $\tilde{P}A$, $\tilde{P} = \text{diag}\,(D_1, D_2, \ldots, D_h)$, where each D_i is either the identity matrix or an $m \times m$ random matrix, and all the diagonal blocks in

the balanced recursive factorization of $\tilde{P}A$ are nonsingular. It follows that $m(\tilde{P}A) \leq 3m$, and we may apply our solutions and our complexity estimates to $LIN \cdot SOLVE$ and DET with the input matrix $\tilde{P}A$. Clearly, the solutions for $\tilde{P}A$ are immediately extended to the ones for A. It remains to substantiate the claim that using the matrices of the format (4.26) as multipliers suffices to ensure the desired regularization of the diagonal blocks. This will follow from the next result.

THEOREM **4.6.1** *Let $A = W + \text{diag}\,(U, 0, V)$ where U and V are $m \times m$ matrices and W is a nonsingular banded matrix of size $n \times n$, with bandwidth $m = m(W)$. Let $n_1 = \lfloor n/2 \rfloor$ and $n_2 = n - n_1$. Then there exists a permutation matrix R of size $m \times m$, such that the matrix $B = PA = \text{diag}\left(I_{n_1 - m_+(W)}, R, I_{n_2 - m_-(W)}\right) A$ has the balanced 2×2 block representation of the format (4.7), where the matrix B_{11} is nonsingular.*

PROOF. Our proof follows [S] and was inspired by an argument of [E] applied in the existence proof for a permutation matrix with similar properties, but for a different, 3×3, block representation of the input matrix. Let A_L denote the $n \times n_1$ matrix $\begin{pmatrix} A_{11} \\ A_{21} \end{pmatrix}$. Without loss of generality, we will assume that $m_+(A) = m_-(A)$ and set $m_+(A) = m_-(A) = k$. Since A is nonsingular, A_L has full (column) rank n_1. The top $n_1 - k$ rows of A_L are linearly independent, since the top $n_1 - k$ rows of A are linearly independent, and since these rows have no nonzero entries outside A_L. The top $n_1 + k$ rows of A_L have full rank, n_1, as well, because A_L has this rank and has no

nonzero entries below these rows. It follows that there exists a set of n_1 rows of A_L that includes the first $n_1 - k$ top rows, as well as k of the next $2k = m$ rows, that form a nonsingular $n_1 \times n_1$ matrix. Consequently, there exists a permutation matrix R of size $2k \times 2k$ such that the leading principal $n_1 \times n_1$ submatrix of the matrix $B = \text{diag}\ (I_{n_1-k}, R, I_{n_2-k})\, A$ is nonsingular. ■

Similarly, nonsingularity of B_{22} (for a certain assignment of the entries of R) follows. Therefore, the matrices B_{11} and B_{22} are nonsingular for a generic choice of the matrix R, that is, for the matrix R filled with indeterminates. By the standard argument of [Sc], [Z], the nonsingularity of B_{11} and of B_{22} then follows (with a high probability) for a random assignment of the values (from a fixed large set) to the entries of R. Similar results apply to subsequent stages of the recursive regularization of the diagonal blocks. The overall number of random parameters used in order to achieve the entire recursive factorization is $O\left(m^2(1 + 2 + 2^2 + \ldots + 2^{\log(n/m)})\right) = O(nm)$.

4.7 Block Bidiagonal Linear Systems.

In this section, A denotes a nonsingular $k \times k$ block matrix of the following format, where $k \geq 2$, A_i $(0 \leq i \leq k-1)$ and B_j $(1 \leq j \leq k-1)$ are $m \times m$

blocks:

$$A = \begin{pmatrix} A_0 & B_1 & 0 & \dots & & 0 \\ 0 & A_1 & B_2 & \dots & & 0 \\ 0 & 0 & A_2 & \dots & & \vdots \\ \vdots & \vdots & & \ddots & \ddots & \\ 0 & 0 & \dots & & A_{k-2} & B_{k-1} \\ 0 & 0 & \dots & & 0 & A_{k-1} \end{pmatrix}. \tag{4.27}$$

Assume that the matrix A of (4.27) is nonsingular and observe that $\det A = \prod_{i=0}^{k-1} \det A_i$, so that all the diagonal blocks A_i of A are nonsingular too. With no loss of generality, let $k = 2^v$ for an integer v.

The problem $LIN \cdot SOLVE$ for $Ax = b$, where the matrix A has format (4.27), has actually been solved in section 2.4.4 (see Algorithm 2.4.1 and Remarks 3.6.2 and 4.7.1), but we will now show a solution based on the BCR algorithm (by following [PSA]).

Let us hereafter denote

$$D_{k,m} = \operatorname{diag}(A_0, A_1, \dots, A_{k-1}), \quad V_i = A_{i-1}^{-1} B_i, \quad i = 1, \dots, k-1, \tag{4.28}$$

$$V = D_{k,m}^{-1} A = \begin{pmatrix} I & V_1 & 0 & \dots & & 0 \\ 0 & I & V_2 & \dots & & 0 \\ 0 & 0 & I & V_3 & & \vdots \\ \vdots & \vdots & & \ddots & \ddots & \\ 0 & 0 & \dots & & I & V_{k-1} \\ 0 & 0 & \dots & & 0 & I \end{pmatrix} \tag{4.29}$$

and define *preprocessing-bd* of a matrix A of (4.27) as the computation of the matrices V of (4.29) and $V_i = V_i^{(0)}$ of (4.28), for $i = 1, \dots, k-1$, comple-

mented by the computation of the matrices

$$V_{i+1}^{(h+1)} = V_{2i+1}^{(h)} V_{2i+2}^{(h)}, \quad i = 0, \ldots, 2^{v-h} - 1, \quad V_{2^{v-h}}^{(h)}, \quad h = 0, 1, \ldots, v.$$

PROPOSITION **4.7.1** *Preprocessing-bd can be performed at deterministic computational cost bounded by*

$$O_A\left(t_{I(m)} + (log\ k)(log\ m),\ kp_{I(m)} \frac{t_{I(m)}}{t_{I(m)} + (log\ k)(log\ m)}\right).$$

Next, we will estimate the cost of solving a linear system $Ax = b$ with the matrix A of (4.27), provided that the preprocessing-bd of A has been performed.

PROPOSITION **4.7.2** *Let A be a nonsingular matrix of the format (4.27). Suppose that the preprocessing-bd of A has been performed. Then the deterministic computational complexity of solving the linear system $Ax = b$ is bounded by*

$$O_A\left((\log k)(\log m),\ \frac{k\,m^2}{(\log k)(\log m)}\right). \tag{4.30}$$

PROOF. Our proof is based on a variant of the block cyclic reduction algorithm. Since we have preprocessing-bd done, we now shift to the solution of the equivalent linear system

$$Vx = c, \tag{4.31}$$

where $c = D_{k,m}^{-1} b$, and $D_{k,m}$ and V are the matrices of (4.28), (4.29). [Clearly the computational cost of multiplication of the matrix $D_{k,m}^{-1}$ by the vector b is

within the bound (4.30).] We will keep assuming (with no loss of generality) that $k = 2^r$. Let S_r denote the linear system (4.31), represented by the following 2^r equations where each of the vectors $x^{(h)}$ and $c^{(h)}$ has dimension m:

$$x^{(h)} + V_{h+1}x^{(h+1)} = c^{(h)},\ 0 \le h \le 2^r - 1. \tag{4.32}$$

If we multiply the equation (4.32), for $h = 2i + 1$, by the block $(-V_{2i+1})$ $(0 \le i \le 2^{r-1})$ and add the resulting equation to the equation (4.32), for $h = 2i$, then we will arrive at the following linear system, to be denoted S_{r-1} and consisting of 2^{r-1} equations:

$$x^{(2i)} - V_{2i+1}V_{2i+2}x^{(2i+2)} = c^{(2i)} - V_{2i+1}c^{(2i+1)},\ 0 \le i \le \frac{k}{2} - 1,\ V_k = 0. \tag{4.33}$$

As soon as we solve this linear system, we may immediately obtain the values of the vectors $x^{(1)}$, $x^{(3)}$, $x^{(5)}$, ..., $x^{(k-1)}$ satisfying the vector equation (4.32). Indeed, we just need to substitute the known values of the vectors $x^{(2i)}$ into (4.32) and then concurrently perform, at first, $k/2$ multiplications of $m \times m$ matrices V_{2i+1} by vectors $x^{(2i+1)}$ and then $k/2$ subtractions of the resulting vectors from $b^{(2i+1)}$, for $i = 0, \ldots, (k/2) - 1$. These steps are performed at the cost $O(\log m, km^2/\log m)$. Due to this reduction of the input size, we have the following recurrence relations for the deterministic parallel complexity $(t_k,\ p_k)$ of solving the system (4.32):

$$t_k \le t_{k/2} + O(\log m),$$

$$p_k \le p_{k/2} + O(km^2/\log m).$$

Recursive application of the latter bounds and the B-principle yield the bound (4.30). ∎

Now repeat the proof of Proposition 4.7.2, with the vector b replaced by an $n \times m$ matrix, then apply Proposition 4.7.1 and obtain

PROPOSITION **4.7.3** *For a nonsingular block bidiagonal matrix A of the format (4.27), the first m rows of its inverse A^{-1} can be computed at deterministic cost bounded by*

$$O_A\left(t^*,\ \frac{kp_{I(m)}}{t^*}\right), \quad t^* = t_{I(m)} + (\log k)\,(\log m). \tag{4.34}$$

PROPOSITION **4.7.4** *The estimates of Propositions 4.7.1–4.7.3, for $k = n/m$, apply to the solution of a nonsingular linear system $Ax = b$ with an $n \times n$ triangular matrix A having its bandwidth equal to $m(A) = m$.*

REMARK **4.7.1** *Simple inspection shows that the above variant of BCR algorithm applied to block bidiagonal linear system of equations is computationally equivalent, that is, produces the same set of computed values as Algorithm 2.4.1 for solving $LR_m(1)$, a m-dimensional linear recurrence of the first order. Also taking into account Remark 3.6.2, we observe close correlation among several well-known computational techniques, that is, BCR, ND, Algorithm 2.4.1 and prefix computation (all being examples of the divide-and-conquer method).*

4.8 Solution of $LIN \cdot SOLVE^*$ via the Reduction of a Banded Triangular Linear System to the Block Bidiagonal Form.

Assume that A, the $n \times n$ input matrix to the problem $LIN \cdot SOLVE^*$, has a lower edge. Denote that $q = m_-(A) \leq m = m(A)$, define the $(n+q) \times (n+q)$ matrix $B = \begin{pmatrix} V & A \\ 0 & W \end{pmatrix}$, where $V = \begin{pmatrix} I_q \\ 0 \end{pmatrix}$, $W = (0 \quad I_q)$, and consider the auxiliary linear system

$$B \begin{pmatrix} 0 \\ x \end{pmatrix} = \begin{pmatrix} b \\ z \end{pmatrix}, \tag{4.35}$$

where the vector z will be chosen later on, so as to make the system (4.35) equivalent to the original linear system $Ax = b$. We observe that the matrix B is nonsingular (since the matrix A has a lower edge) and has bandwidth $m(B) \leq m = m(A)$. Denote that $B^{-1} = \begin{pmatrix} G & H \\ K & L \end{pmatrix}$ where H is a $q \times q$ block. From the nonsingularity of A and the factorization

$$B = \begin{pmatrix} V & A \\ 0 & W \end{pmatrix} = \begin{pmatrix} I & 0 \\ WA^{-1} & I \end{pmatrix} \begin{pmatrix} A & 0 \\ 0 & -WA^{-1}V \end{pmatrix} \begin{pmatrix} A^{-1}V & I \\ I & 0 \end{pmatrix}, \tag{4.36}$$

we deduce that $H = -(WA^{-1}V)^{-1}$, which implies nonsingularity of H.

LEMMA **4.8.1** *Set*

$$z = -H^{-1}Gb. \tag{4.37}$$

Then the two linear systems $Ax = b$ and (4.35) are equivalent to each other.

PROOF. The nonsingular system $Ax = b$ is the subsystem of (4.35) formed by the first n equations of (4.35). Substitution of $x = A^{-1}b$ into (4.35)

defines the unique vector z satisfying (4.35). On the other hand, the vector z must satisfy the first q linear equations of the system $B^{-1}\begin{pmatrix} b \\ z \end{pmatrix} = \begin{pmatrix} G & H \\ K & L \end{pmatrix}\begin{pmatrix} b \\ z \end{pmatrix} = \begin{pmatrix} 0 \\ x \end{pmatrix}$, that is, this vector must satisfy the q equations $Gb + Hz = 0$. This defines a unique vector z of (4.37). ∎

We have reduced the original linear system $Ax = b$ to the banded triangular linear system (4.35), by means of performing the following stages:

1. $(PREPROCESS \cdot REDUCE)$: first compute the first q rows of the matrix B^{-1}, that is, compute the matrix $I_F(n+q, q)B^{-1} = (G \quad H)$ (compare Proposition 4.2.1); then compute the matrix H^{-1}.

2. $(BACK \cdot REDUCE)$: compute the vector $-H^{-1}Gb$.

We call the computation at stage 1 preprocessing because it does not involve the vector b. The computational cost of performing stages 1 and 2 is bounded by $O_A(t^*, \frac{n}{q} P_{I(q)}/t^*)$, $t^* = t_{I(q)} + \left(\log \frac{n}{q}\right)(\log q)$, for stage 1 (due to Proposition 4.7.2), and by $O_A(\log n, nq/\log n)$, for stage 2. The next theorem summarizes the computational costs of both the reduction of the original system to the triangular linear system and the subsequent solution of the latter system.

THEOREM **4.8.1** *A nonsingular $n \times n$ matrix A, having bandwidth $m = m(A)$ and having at least one edge, can be preprocessed at a computational cost bounded by (4.6) (by applying Proposition 4.7.2 and the reduction algorithm*

of this section). After that, for any fixed vector b of dimension n, the linear system $Ax = b$ can be solved at a computational cost bounded by (4.5).

REMARK **4.8.1** The estimates of Theorem 4.8.1 can be immediately extended to the case where the entries of A are block (matrices), in particular, to the case of a nonsingular block tridiagonal input matrix A whose all subdiagonal and/or all superdiagonal blocks are nonsingular (compare Proposition 4.2.2).

4.9 Extensions from $LIN \cdot SOLVE^*$ to $LIN \cdot SOLVE$.

In this section, rounding to the nearest integer will be allowed as one of the unit cost operations. For the extension from $LIN \cdot SOLVE^*$ to $LIN \cdot SOLVE$ we may apply various modifications of the techniques of the variable artificial diagonal of [P85], [P87]. Consider a $k \times k$ block tridiagonal matrix with $m \times m$ blocks defined as in (3.18). The class of such matrices includes, in particular, all the $(km) \times (km)$ matrices A with $m_-(A) \leq m$, $m_+(A) \leq m$. With no loss of generality, assume that

$$\det A_i \neq 0, \quad \text{for all } i, \tag{4.38}$$

and transform the matrix A into a block matrix polynomial $A(\lambda) = A + \lambda Z^{2m}$ in the scalar parameter λ where Z denotes the $(km) \times (km)$ matrix filled with zeros, except for its first subdiagonal, filled with ones, so that $A(\lambda)$ is obtained from A by means of inserting the matrix λI_m below each block

C_i. Moreover, $A(\lambda)$ is a matrix polynomial of degree 1 with a lower edge, and $(A(\lambda))^{-1} \bmod \lambda = A^{-1}$. If the computation is over a field of characteristic 0, we may set $\lambda = \epsilon$, for a sufficiently small positive ϵ, and compute an approximate solution, $A^{-1}(\epsilon)b$, to a linear system $Ax = b$. If the computation is over the rationals, then we may shift to the integer input by scaling, choose a sufficiently small ϵ and then apply the techniques of [UP], to recover the exact solution $A^{-1}b$ from its approximation $A^{-1}(\epsilon)b$. Furthermore, adj $A = A^{-1}\det A$ is an integer matrix, so we may easily recover det A and adj A via rounding-off (their approximations by) det $A(\epsilon)$ and adj $A(\epsilon)$ to the nearest integers. An alternative way is to fix a prime p and to compute the vectors $x(p) = A^{-1}(p)b$ and $A^{-1}(p)b \bmod p = x(p) \bmod p$. Then again, we may compute det $A(p) \bmod p$ and adj $A(p)b \bmod p$ and recover det A and (adj $A)b$ if p is suffiently large. Such a computation may fail if det $A \bmod p = 0$, but this may occur only with a small probability if p is sufficiently large and det $A \neq 0$ (see [P], [BP]). Thus, over the rationals, we extend our arithmetic complexity estimates, (4.6) for *PREPROCESS* and (4.5) for $BACK \cdot SOLVE$, to any $n \times n$ matrix with bandwidth m.

REMARK **4.9.1** We may decrease the precision of the latter computation, $\lceil \log p \rceil$, if we first replace p by several primes, $p_1, \ldots, p_k$ such that $p_1 \ldots p_k > p$, then perform the above computation modulo p_i for $i = 1, \ldots, k$, and recover det $A \bmod p$ and adj $A \bmod p$, by means of the Chinese remainder theorem. Alternatively, we may apply p-adic lifting of [MC].

REMARK **4.9.2** Suppose that only s entries equal to 0 prevent a matrix A from having lower or upper edges. Then we may replace these entries by λ, solve the resulting system $A(\lambda)x = b$, and output $x = x(\lambda) \bmod \lambda = A^{-1}b$. If s is small, then the computation of $x(\lambda)$ only involves polynomials in λ of small degrees (as well as the ratios of such polynomials). In particular, the overall complexity of the solution is still bounded according to (4.5) and (4.6), if $s = O(1)$, that is, if s is bounded by a fixed constant independent of m and n.

4.10 Singular Banded Matrix Computations.

Gaussian Elimination can be modified slightly and applied in order to produce a solution to a consistent singular linear system of equations, implying an upper bound of $O(nm^2)$ arithmetic operations if the input matrix has order n and bandwidth m.

In this section, we recall the efficient parallel algorithms of [E95] for singular banded matrix computations over arbitrary fields. (In our presentation in this section, we will follow [E95].) A unit cost operation in these algorithms is an addition, a subtraction, a multiplication or a division of a pair of elements of the ground field, a zero test of an element, or a uniform and random selection of an element from a finite set in the ground field. The parallel computational complexity of solving a nonsingular system of equations $Ax = b$ or computing the determinant of A, where A is a (dense) nonsingular

$n \times n$ matrix with entries in a field F is given by

$$\left(t_{I(n)},\ p_{I(n)}\right) = \left(t_{D(n)},\ p_{D(n)}\right) = O_A\left(\phi(F,\ n),\ p_{M(n)}\right),$$

where

$$\phi(F,\ n) \leq \begin{cases} log^2\ n, & \text{if } F \text{ has characteristic 0,} \\ log^4\ n, & \text{for any } F, \end{cases}$$

provided that the computation is performed over the field of constants F and $p_{M(n)} = max(n^w log\ n,\ n^2 log\ n\ log\ n)$ processors (where $w < 2.376$). The running time bound for randomized algorithms that compute a single solution of the system $Ax = b$ and a basis for the right nullspace of A using $p_{M(n)}$ processors if A is a singular $n \times n$ matrix over a field F will be denoted $t_{Sing}(n)$. According to [BP], appendix C of chapter 4,

$$t_{Sing}(n) = O(t_{I(n)}); \tag{4.39}$$

furthermore, one may compute a maximal nonsingular minor of an $n \times n$ matrix A over F, at the randomized parallel cost bounded by

$$0_A(t_{Sing}(n) log\ n,\ p_{M(n)}).$$

DEFINITION **4.10.1** *Let $A \in F^{q \times s}$ and $B \in F^{r \times s}$. A matrix $N \in F^{s \times t}$ is an $(A,\ B)$-separator if $t \leq s$ and if simultaneously we have*

1. $AN = 0$,

2. *For all $x \in F^{s \times 1}$, if $Ax = 0$, then there exists $y \in F^{t \times 1}$ such that $Bx = BNy$,*

3. *For all* $z \in F^{s \times u}$, *if* $zBN = 0$, *then there exists* $w \in F^{l \times q}$ *such that* $zB = wA$.

REMARK **4.10.1** *The* $(A,\ B)$*-separator defined in this section is distinct from the separators of chapter 3.*

DEFINITION **4.10.2** *A matrix* $T \in F^{s \times u}$ *is an* A*-compressor if* $u < s$ *and* $rank(A) = rank(AT)$.

DEFINITION **4.10.3** *Let* $A \in F^{n \times n}$ *and* $B \in F^{m \times n}$. *If* A *is a banded matrix with a bandwidth* m, *then an* A-decomposition *is an* $(A_{Mid},\ B)$-decomposition, *where* B *includes the top* m *rows of* A *and* $A_{Mid} \in F^{(n-m) \times n}$ *includes the rest of the rows of* A *(see (4.41)).*

If $m \geq \frac{n}{c}$ *for some constant* $c > 1$, *so that matrix* A *is dense, we define* $(A_{Mid},\ B)$*-decomposition to be a pair* $(N,\ NB)$ *where* $N \in F^{n \times n}$ *is an* $(A_{Mid},\ B)$*-separator (see Definition 4.10.1), and identify* N *as the* $(A_{Mid},\ B)$*-separator induced by this decomposition. Otherwise an* $(A_{Mid},\ B)$*-decomposition contains the following:*

(i) an $(A_i^{(1)},\ B_i^{(1)})$*-decomposition for* $1 \leq i \leq h$, *where* $(A_i^{(1)}$ *and* $B_i^{(1)})$ *are the* i^{th} *blocks of* $(A_i^{(1)}$ *and* $B_i^{(1)})$ *respectively, as this will be shown in (4.41);*

(ii) a $(B^{(1)}N^{(1)})$*-compressor* T *(see Definition 4.10.2) with the properties to be described in Lemma 4.10.3, where* $N^{(1)}$ *is the block diagonal*

$(A^{(1)}, B^{(1)})$-separator whose diagonal blocks are the separators induced by the decompositions in (i) above (see Lemma 4.10.1(b));

(iii) *an $(A^{(2)}N^{(1)}T, B^{(2)}N^{(1)}T)$-decomposition;*

(iv) *$BN^{(1)}TN^{(2)}$, where $N^{(2)}$ is the $(A^{(2)}N^{(1)}T, BN^{(1)}T)$-separator induced by (iii);*

(v) *$N^{(1)}T$.*

Let us write

$$\begin{aligned} p_{M(n,\,m)} &= \frac{n}{m}\, p_{M(m\log(n/m))} \\ &= max(nm^{w-1}log(n/m),\ nmlog(n/m)\ log\ log(n/m)). \end{aligned}$$

THEOREM **4.10.1** *Let $A \in F^{n\times n}$ be a banded matrix with bandwidth m, where F includes $\Omega(n)$ distinct elements.*

(a) *An A-decomposition can be computed by using a Monte Carlo randomized algorithm at the cost bound*

$$O_A(t_{n,m},\ p_{M(n,\,m)}),$$

where

$$t_{n,m} = log(\frac{n}{m})\ t_{Sing}(m\ log(\frac{n}{m}))\ +\ (log\ \frac{n}{m})(log\ log\ \frac{n}{m}).$$

(b) Given an A-decomposition and a vector $b \in F^{n\times 1}$, *a vector* $x \in F^{n\times 1}$ *can be found (if one exists), or "no solution" reported, by a randomized Las Vegas algorithm using the time and the number of processors given in (a).*

(c) Given an A-decomposition, the rank of A can be computed by a randomized Las Vegas algorithm at the cost

$$O_A((t_{Sing}(m\ log(n/m)) + log\ n),\ p_{M(n,\ m)}(log\ log(n/m))).$$

(d) Given an A- and an A^T*-decomposition, a maximal nonsingular minor of A can be computed at the cost*

$$O_A((t_{Minor}(m\ log(n/m))+log\ n\ log\ log(n/m),\ p_{M(n,\ m)}(log\ log(n/m))).$$

4.10.1 A Recursive Decomposition for a Banded Matrix

LEMMA **4.10.1** *Let* $A \in F^{q\times s}$ *and* $B \in F^{r\times s}$ *be as above.*

(a) If $N \in F^{s\times t}$ *is a matrix whose columns span the right nullspace of A, then N is an* (A, B)*-separator.*

(b) Suppose that A is a block matrix diagonal with blocks $A_i \in F^{q_i\times s_i}$ *for* $1 \le i \le h$, *and suppose that B has blocks* $B_i \in F^{r\times s}$ *for* $1 \le i \le h$, *so that* $q = \sum_{i=1}^{h} q_i$, $s = \sum_{i=1}^{h} s_i$, *and*

$$A = \begin{bmatrix} A_1 & & & 0 \\ & A_2 & & \\ & & \ddots & \\ 0 & & & A_h \end{bmatrix}, \quad B = [B_1\ \ B_2\ \ \cdots\ \ B_h]. \qquad (4.40)$$

Let $N_i \in F^{s_i \times t_i}$ *be an* $(A,\ B)$*-separator for* $1 \leq i \leq h$*, let* $t = \sum_{i=1}^{h} t_i$ *and set*

$$N = \begin{bmatrix} N_1 & & & 0 \\ & N_2 & & \\ & & \ddots & \\ 0 & & & N_h \end{bmatrix} \in F^{s \times t}.$$

Then N *is an* $(A,\ B)$*-separator.*

(c) Suppose that $A^{(1)} \in F^{q_1 \times s}$ *and* $A^{(2)} \in F^{q_2 \times s}$ *partition the rows of* A*, so that* $q = q_1 + q_2$ *and there exists a permutation matrix* $P \in F^{q \times q}$ *such that*

$$A = P \begin{bmatrix} A^{(1)} \\ A^{(2)} \end{bmatrix}.$$

Set $B^{(1)} \in F^{(q_2+r) \times s}$ *to be a matrix whose rows are those of* $A^{(2)}$ *and* B*. In this case, if* $N^{(1)} \in F^{s \times t_1}$ *is an* $(A^{(1)},\ B^{(1)})$*-separator, and* $N^{(2)} \in F^{t_1 \times t}$ *is an* $(A^{(2)}N^{(1)},\ BN^{(1)})$*-separator, then* $N = N^{(1)}N^{(2)} \in F^{s \times t}$ *is an* $(A,\ B)$*-separator.*

(d) If $T \in F^{s \times t}$ *is a* C*-compressor for* $C = \begin{bmatrix} A \\ B \end{bmatrix}$*, then* T *is also a* B*-compressor, and if* $N_T \in F^{t \times u}$ *is an* $(AT,\ BT)$*-separator, then* $N = TN_T \in F^{s \times u}$ *is an* $(A,\ B)$*-separator.*

LEMMA 4.10.2 *If* $T \in F^{s \times t}$ *is a* B*-compressor and* $b \in F^{s \times 1}$*, then the equation* $Bx = b$ *has a solution* $x \in F^{s \times 1}$ *if and only if the equation* $(BT)y = b$ *has a solution* $y \in F^{t \times 1}$*. Furthermore, if* y *is a solution to the latter equation, then* Ty *is a solution to the former.*

We will consider $(A_{Mid},\ B)$-separator in the case $A_{Mid} \in F^{l \times n}$, $B \in F^{m \times n}$ such that $l+m = n$ and $A \in F^{n \times n}$ is a banded matrix with bandwidth m, where

$$B = \begin{bmatrix} B_{Top} \\ B_{Bottom} \end{bmatrix} \in F^{m \times n} \quad A = \begin{bmatrix} B_{Top} \\ A_{Mid} \\ B_{Bottom} \end{bmatrix} \in F^{n \times n}, \tag{4.41}$$

so that the rows of B enclose those of A_{Mid} in A.

- If $m \geq \frac{n}{c}$ for some constant $c > 1$, matrix A will be dense, and by lemma 4.10.1(a), $(A_{Mid},\ B)$-separator can be obtained by computing a basis for the right nullspace of A_{Mid}, in time $t_{Sing}(n) \in O(t_{Sing}(m))$ using $O(p_{M(n,\ m)}) \in O(\frac{n}{m} p_{M(n,\ m)})$ arithmetic processors.

- If $m < \frac{n}{m}$, we set $k = \lceil \sqrt{\frac{n}{m}} \rceil$ and partition the rows of A_{Mid} into two subsets, S_1 and S_2. We include the top $(k-1)m$ rows in S_1, the next m rows of S_2, and iterate, so that S_1 includes sets of $(k-1)m$ contiguous rows in A_{Mid} that are bordered by sets of m contiguous rows in S_2, possibly ending with an "incomplete" set of a smaller number of rows in S_1 or S_2.

- Let l_j be the number of rows in S_j and let $A^{(j)} \in F^{l_j \times n}$ include those rows of A belonging to S_j (in order), for $j \in \{1, 2\}$. Then $l = l_1 + l_2$, and there exists a permutation $P_A \in F^{l \times l}$ such that

$$A_{Mid} = P_A \begin{bmatrix} A^{(1)} \\ A^{(2)} \end{bmatrix}.$$

$A^{(1)}$ is a block diagonal matrix, with h blocks $A_i \in F^{l_i \times n_i}$ for $1 \le i \le h$, where $h \le k$, $\sum_{i=1}^{h} l_i = l$, $\sum_{i=1}^{h} n_i = n$, $l_i \le (k-1)m$ for $1 \le i \le h$ (and $l_i = (k-1)m$ if $i < h$), and $n_i \le km$ for $1 \le i \le h$ (and $n_i = km$ if $i < h$). Let $B^{(1)} \in F^{(l_2+m)\times n}$ include the rows of B and $A^{(2)}$ (with the rows of $A^{(2)}$ below the rows of B_{Top} and above those of B_{Bottom}, for B_{Top} and B_{Bottom} defined as in (4.41)).

LEMMA **4.10.3** *Let $B^{(1)}$ and $N^{(1)}$ be as given above. Then there exists a block matrix $T \in F^{n\times(l_2+m)}$ with tall, thin blocks consisting of at most km rows and m columns, and with nonzero entries only in blocks on and immediately above the diagonal, such that $B^{(1)}N^{(1)}T \in F^{(l_2+m)\times(l_2+m)}$ is a banded matrix with bandwidth at most $2m$, and such that $rank(B^{(1)}N^{(1)}) = rank(B^{(1)}N^{(1)}T)$, so that T is a $(B^{(1)}N^{(1)})$-compressor.*

Then the desired matrix T of Lemma 4.10.3 can be found with a high probability by choosing random elements of the nonzero blocks of T and under the uniform probability distribution independently of each other from a finite subset of F of size $O(n)$. Theorem 4.10.1(a) is an immediate consequence of the following lemma.

LEMMA **4.10.4** *Let A_{Mid} and B be as above. Then an (A_{Mid}, B)-decomposition can be computed by a randomized Monte Carlo algorithm by using the time and number of processor given in Theorem 4.10.1(a).*

For the sketch of proof of lemma 4.10.4 we refer the reader to [E95].

4.10.2 Solving a Singular Banded Linear System

Let $A \in F^{n\times n}$ be a banded matrix with a bandwidth m, and suppose that an A-decomposition has been computed. Let $b \in F^{n\times 1}$.

1. If $m \geq \frac{n}{m}$, then a vector $x \in F^{n\times 1}$ can be found such that $Ax = b$ (if such vector exists) at the cost stated in Theorem 4.10.1(b) by treating A as dense.

2. Otherwise let $A^{(1)}$ and $A^{(2)}$ be as described in section 4.10.1. By definition of A-decomposition (see Definition 4.10.3), this decomposition includes an $A^{(1)}$-decomposition and a $(B^{(1)}N^{(1)}T)$-decomposition, where $N^{(1)}$ is the $(A^{(1)},\ B^{(1)})$-separator and T is the $(B^{(1)}N^{(1)})$-compressor defined above.

3. Since $A^{(1)}$ and $B^{(1)}$ partition the rows of A, the above equation is equivalent to the equations $A^{(1)}x = b_A$ and $B^{(1)}x = b_B$, where b_A and b_B are "restrictions" of b to the rows included in $A^{(1)}$ and $B^{(1)}$. Suppose the system $Ax = b$ has some (unknown) solution x; then the system $A^{(1)}x = b_A$ must have a solution as well. Let $x^* \in F^{n\times 1}$ such that $A^{(1)}x^* = b_A$; then $A^{(1)}(x - x^*) = 0$, so (by definition of $(A,\ B)$-separator) there must exist some vector y such that

$$B^{(1)}N^{(1)}y = B^{(1)}(x - x^*) = b_B - B^{(1)}x^*.$$

Since T is a $(B^{(1)}N^{(1)})$-compressor, there must exist some vector z such that

$$(B^{(1)}N^{(1)}T)z = b_B - B^{(1)}x^*.$$

Inspection shows that if $\tilde{x} = x^* + (N^{(1)})z$, then $A\tilde{x} = b$, so that $\tilde{x}$ is the solution to the original equation.

REMARK **4.10.2** *Since $B^{(1)}N^{(1)}T$ and the diagonal blocks of $A^{(1)}$ are banded matrices of order $\sqrt{nm}$, they have bandwidths at most $2m$, and their A-decompositions are available. Futhermore, the (sparse) matrix $N^{(1)}T$ is included as part of the given A-decomposition. It follows that the system $Ax = b$ can be solved by recursively solving two systems of equations whose coefficients matrices have order $\sqrt{nm}$ and bandwidth at most $2m$, multiplying a sparse matrix by a vector, and adding two vectors together. An analysis of this method leads to Theorem 4.10.1(b).*

REMARK **4.10.3** *The rank of A is the sum of the ranks of the diagonal blocks $A_1^{(1)}$, $A_2^{(1)}$, ..., $A_h^{(1)}$ of $A^{(1)}$ and $B^{(1)}N^{(1)}T$. Thus the rank of A can be computed at the cost claimed in Theorem 4.10.1(c) by computing the ranks of these smaller matrices in parallel and then adding them together.*

REMARK **4.10.4** *To compute a maximal linearly independent set of rows of A it suffices to compute such a set of the rows of $B^{(1)}N^{(1)}T$ and to merge the corresponding rows in $B^{(1)}$ with a maximal linearly independent set of rows of $A^{(1)}$. (This is a consequence of the final condition in the definition of an*

(A, B)-separator, as well as the definition of an A-compressor). Thus the problem can be solved by recursively solving $(1+\sqrt{n/m})$ smaller instances of the problem in parallel and then renumbering or reordering the resulting rows (depending on the output representation used) at the cost of additional time $O(\log n)$. Analysis of this recursive algorithm yields the time and processor bounds given in Theorem 4.10.1(d).

4.11 Summary

By using various recent techniques, we presented several algorithms for computations with banded matrices. In particular, for DET and in the case of nonsingular case of $LIN \cdot SOLVE$, we reached processor optimality of our NC and RNC solutions. In the case of a constant bandwidth m, we reached the RNC^1 time bound $O(\log n)$. Moreover, our algorithms are deterministic over the fields of characteristic 0, and improved solution of several linear systems with the same banded matrix can be obtained by using preprocessing algorithms.

Chapter 5

Parallel Computation of Krylov Sequences for Special Input Matrices

5.1 Introduction

DEFINITION **5.1.1** *The matrix*

$$K(A, v, m) = [v, Av, A^2v, \cdots, A^{m-1}v] \tag{5.1}$$

is called the Krylov matrix defined by an $n \times n$ matrix A, an n-dimensional vector v, and a nonnegative integer m.

This matrix represents a linear recurrence of order 1 and dimension n. Its sequential computation is straightforward:

$$v_0 = v, \quad v_{i+1} = Av_i = A^{i+1}v, \quad i = 0, 1, \ldots, m-1.$$

In this chapter we follow [P92] and [P94] in order to reduce parallel computation of a Krylov matrix to solving a parametrized linear system of equations.

Such a method is effective in the cases of banded matrices, sparse and structured matrices, and triangular matrices. For a general matrix A, the best parallel algorithm for computing $K(A,v,m)$ follows [BM], [KG].

ALGORITHM **5.1.1** *(***Krylov sequences computation***)*

Input: *A pair of natural m and n, an $n \times n$ matrix A, and an n-dimensional vector v. (We assume $m = 2^h - 1$ for simplicity.)*

Output: *the Krylov matrix of (5.1).*

Computation:

At first, for $i = 1,2,\cdots,h-1$, recursively compute the matrices A^{2^i}. Then perform $h-1$ multiplications of $n \times n$ matrices A^{2^i}, $i = 1,\ldots,h-1$, as follows:

$$\begin{aligned} A^2(v,Av) &= (A^2v, A^3v), \\ A^4(v,Av,A^2v,A^3v) &= (A^4v, A^5v, A^6v, A^7v), \\ \cdots\cdots\cdots\cdots &= \cdots\cdots\cdots\cdots \end{aligned} \tag{5.2}$$

end ∎

Due to this algorithm, we arrive at the following

PROPOSITION **5.1.1** *[KG], [BM]. Given an $n \times n$ matrix A, an n-dimensional vector v, and a nonnegative integer m, one may compute the Krylov matrix $K(A,v,m)$ at the cost $O_A((\log n)(\log m), p_{M(n)}/\log m)$.*

5.2 Definitions and Notation

DEFINITION **5.2.1** *Let A be an $n \times n$ matrix whose entries come from an arbitrary field $\mathcal{F}$, and let λ be an indeterminate. The* characteristic polynomial *of A is defined to be $\phi(\lambda) = det(\lambda I - A) = \sum_{i=0}^{n} c_i \lambda^i$, which is the n^{th} degree polynomial whose roots are called the* eigenvalues *of A.*

In particular, this definition implies that $\phi(0) = c_0 = det(-A) = (-1)^n c_0$.

THEOREM **5.2.1** *(***Cayley-Hamilton***) Let A be a matrix whose entries come from a fixed arbitrary field, and let $\phi(\lambda) = \sum_{i=0}^{n} c_i \lambda^i$ be the characteristic polynomial of A. Then $\phi(A) = \sum_{i=0}^{n} c_i A^i = 0$.*

LEMMA **5.2.1** *Given the coefficients of the characteristic polynomial $\phi(\lambda)$ of an $n \times n$ matrix A, the linear system of equations defined by $Ax = b$, where x and b are $n-$dimensional vectors, can be solved in $O_A(log^3 n, p_{M(n)})$, for $p_{M(n)}$ of (2.21).*

PROOF. :

Let the characteristic polynomial be given by:

$$\phi(\lambda) = \sum_{i=0}^{n} c_i \lambda^i.$$

By Theorem 5.2.1,

$$\sum_{i=0}^{n} c_0 A^i = -c_0 I.$$

This equation implies that, for an arbitrary vector x, we have

$$\sum_{i=0}^{n} c_0 A^i x = -c_0 x.$$

In particular, if x is such that $Ax = b$, we get

$$A^i x = A^{i-1}(Ax) = A^{i-1}b,$$

and thus

$$\sum_{i=0}^{n} c_i A^{i-1} b = c_0 x.$$

The coefficients c_i are given, and $\sum_{i=0}^{n} c_i A^{i-1} b$ is just a linear combination of the columns of the Krylov matrix. If $c_0 = 0$, we declare the system to be singular; otherwise, we set $x = -\frac{1}{c_0} \sum_{i=0}^{n} c_i A^{i-1} b$. ■

Implicit in the proof of Lemma 5.2.1 is a method for computing A^{-1} (if we are given a nonsingular input matrix A) and the polynomial $\phi(\lambda)$. In fact, the Caley-Hamilton theorem implies that

$$(\sum_{i=0}^{n} c_i A^{i-1})A = -c_0 I.$$

The matrix A is invertible if and only if $c_0 \neq 0$ (recall that $det(A) = (-1)^n c_0$). And, in this case, A^{-1} is given by

$$A^{-1} = -\frac{1}{c_0} \sum_{i=0}^{n} c_i A^{i-1}.$$

To summarize, given a nonsingular matrix A and the coefficients of its characteristic polynomial, we have showed an algorithm that inverts A and

solves a linear system of equations with the coefficients matrix A. In addition, we recall that $det(A) = (-1)^n c_0$, where c_0 is the constant coefficient of the charecteristic polynomial.

5.2.1 Computation of the Characteristic Polynomial.

DEFINITION **5.2.2** *The* **trace** *of a matrix A is the sum of the entries on the diagonal of A, that is, $tr(A) = \sum_{i=0}^{n} a_{ii}$. Let $s_k = tr(A^k)$, for $1 \leq k \leq n-1$. The coefficients c_i of the characteristic polynomial of A and the terms s_k are related by the following linear system of equations:*

$$\begin{bmatrix} 1 & 0 & 0 & \dots & \dots & & 0 \\ s_1 & 2 & 0 & \dots & \dots & & \vdots \\ s_2 & s_1 & 3 & \dots & \dots & & \vdots \\ s_3 & s_2 & s_1 & 4 & \dots & & \\ \vdots & & \ddots & \ddots & \ddots & & \\ & & & & \ddots & \ddots & \vdots \\ s_{n-1} & \dots & & s_3 & s_2 & s_1 & n \end{bmatrix} \begin{bmatrix} c_{n-1} \\ c_{n-2} \\ c_{n-3} \\ \vdots \\ \vdots \\ c_1 \\ c_0 \end{bmatrix} = - \begin{bmatrix} s_1 \\ s_2 \\ s_3 \\ \vdots \\ \vdots \\ s_{n-1} \\ s_n \end{bmatrix}. \qquad (5.3)$$

It follows from Equation (5.3) that the problem of computing the coefficients of $\phi(\lambda)$ can be reduced to that of solving a special triangular linear system of equations defined by $\{s_k | 1 \leq k \leq n-1\}$, where the terms s_k are the traces of the matrices $A^k s$. The traces can be computed trivially from the matrix powers $A^2, A^3, \dots, A^{n-1}$, which can be computed fast in parallel.

ALGORITHM **5.2.1** *(***Algorithm for computing the coefficients** c_k, *for* $1 \leq k \leq n-1$*) [Cs]*

- *Step 1: Compute* $A^2, A^3, \ldots, A^{n-1}$.
- *Step 2: Compute* $s_k = tr(A^k)$, *where* $1 \leq k \leq n$.
- *Step 3: Solve the triangular system of equations (5.3) relating the coefficients of the characteristic polynomial to the terms* s_k.

THEOREM **5.2.2** *[BP] Given an* $n \times n$ *matrix A, the coefficients of the characteristic polynomial of A can be determined at the cost* $O_A(log^2\ n,\ nM(n))$.

5.2.2 Newton's Iteration For Matrix Inversion

Let A be an arbitrary $n \times n$ matrix and let λ be an indeterminate. Since

$$(I - \lambda A)(I + \lambda A + \lambda^2 A^2 + \cdots + \lambda^k A^k) = I - \lambda^{k+1} A^{k+1}, \qquad (5.4)$$

for any positive integer k; we may write

$$(I - \lambda A)(I + \lambda A + \lambda^2 A^2 + \cdots + \lambda^k A^k) = I \ mod\ \lambda^{k+1} A^{k+1}, \qquad (5.5)$$

that is,

$$(I - \lambda A)^{-1}\ mod\ \lambda^{k+1} = (I + \lambda A + \lambda^2 A^2 + \cdots + \lambda^k A^k). \qquad (5.6)$$

ALGORITHM **5.2.2** *(***Newton's Iteration***)*

Input: *An arbitrary* $n \times n$ *matrix A, where* $n + 1$ *is a power of 2.*

Output: *The matrix* $(I - \lambda A)^{-1}\ mod\ \lambda^{n+1}$, *where* λ *is an indeterminate.*

begin

1. *Set* $X_0 = (I - \lambda A)$
2. **for** $i = 1$ *to log* $(n+1)$ **do**

   Set $X_i = (2I - X_{i-1}B)X_{i-1}$

end

LEMMA **5.2.2** *The matrix X_i generated during the ith stage of Newton's iteration (Agorithm 5.2.2) satisfies the matrix equation $X_i = (I - \lambda A)^{-1}$ mod λ^{2^i}.*

Let us define the matrix $B(\lambda)^{-1} = (I - \lambda A)^{-1}$, and then compute the entries of the column $A^i v$ of $K(A, v, m)$ as the coefficients of Newman's expansion,

$$B(\lambda)^{-1}v = (I - \lambda A)^{-1}v = \sum_{i=0}^{\infty} (\lambda A)^i v, \tag{5.7}$$

reduced modulo λ^m.

Due to Equation (5.7), the computation of the Krylov matrix A reduces to the solution of the linear system $B(\lambda)X(\lambda) = v$, whose solution,

$$X(\lambda) = B(\lambda)^{-1}v = v \ mod \ \lambda^m, \tag{5.8}$$

is a vector polynomial with coefficients being the columns of $K(A, v, m)$.

5.2.3 Computation of a Krylov Matrix for a Sparse and Structured Input

Let us assume that A is an $k \times k$ *block tridiagonal matrix* with $m \times m$ blocks, which includes the case of any *banded matrix* A with the lower and upper

bandwidths $k_- \leq k$ and $k_+ \leq k$, respectively. Let us apply the block cyclic reduction algorithm (BCR) described in Chapter 3, Section 3.6 to solve the system (5.8). The cost of application of the block cyclic reduction algorithm (BCR) to the scalar input matrix is bounded by

$$O_A(t_{I(k)}\ log\ s, \frac{p_{I(k)}s}{log\ s}), \tag{5.9}$$

[apply theorem 3.6.1 for $k = s$ and $m = k$]. Since the cost of every arithmetic operation increases from $O(1,1)$ to $O_A(log\ m, m)$, because we are dealing with an input matrix whose entries are polynomials reduced modulo λ^m, we reach the bound

$$O_A((log\ k)t_{I(m)}\ log\ s, \frac{m\ p_{I(k)}s}{log\ s})$$

on the overall cost of the solution.

For the case of *sparse matrices* A whose $s(n)$-separator families are available, for $s(n) = o(n)$, (see Definitions 3.2.11), the application of the parallel generalized nested dissection algorithm of [PR] for solving a linear system with a sparse input matrix gives us the cost bound

$$O_A(log^3\ n, \frac{s(n)^2}{log^2\ n}).$$

This result is immediately extended to the cost bound

$$O_A((log\ m)\ log^3\ n, \frac{(s(n)^2)m}{log^2\ n})$$

for computing the vector $B(\lambda)^{-1}v$ of (5.8).

For the case of triangular matrices A, we may apply the algorithm of [P94], [PP], which supports the cost bound $O_A(\sqrt{n}\ log\ n, \frac{n^{3/2}}{log\ n})$ for solving a triangular linear system with a scalar coefficient matrix. For solving the linear system (5.8), the cost bound turns into $O_A((log\ m)\ \sqrt{n}\ log\ n, \frac{n^{3/2}m}{log\ n})$. We note an acceleration if we compare this cost bound to the bound $O(m\ log\ n, \frac{n^2}{log\ n})$ on the cost of the straightforward evaluation of $K(A, v, m)$ via $m-1$ successive multiplications of A by vectors with $\sqrt{n}\ log\ n = O(m)$.

Chapter 6

Time and Space Complexities of Polynomial Multiplication (Vector Convolution)

6.1 Introduction

The purpose of this chapter is to describe an algorithm for the computation of the lower half or the upper half of a convolution vector, that is, its leading or its trailing components. The algorithm reduces the memory space used by slowing down the computation. (These problems are motivated by practical applications to signal and data transmission.) The algorithm depends upon the computation of the positive wrapped and the negative wrapped convolution.

Suppose that we need to compute the lower part of the convolution vector. Then we first compute the sum of the positive wrapped and the negative wrapped convolution, but the last part of the computation of each wrapped convolution is the inverse Fourier transform. To save computer time and

space we do not compute the inverse Fourier transform twice.

The Fast Fourier Transform (*FFT*) has numerious applications in sciences and engineering. We consider one of its specific applications, which consists of computing the convolution of two vectors. In this chapter we will first recall the known framework for our study, [AHU], [BP], and then show an *FFT* based convolution algorithm, which performs using *smaller* space than the other known FFT based convolution algorithms.

6.2 The Discrete Fourier Transform and Its Inverse

Let $(R, +, *, 0, 1)$ be a commutative ring and α an element of R.

DEFINITION **6.2.1** *$\alpha = \alpha_n$ is said to be a principal n^{th} root of unity if and only if:*

$$\alpha^s \neq 1, \quad for \quad 0 < s < n \tag{6.1}$$

$$\alpha^n = 1, \tag{6.2}$$

and

$$\sum_{j=0}^{n-1} \alpha^{jp} = 0 \quad for \quad 1 \leq p < n. \tag{6.3}$$

The elements $\alpha^0, \alpha^1, \cdots, \alpha^{n-1}$ are the n^{th} roots unity.

Let $u = [u_0, u_1, \cdots, u_{n-1}]^T$ be a column vector of dimension n, with elements from R. Without loss of generality, let us assume that the integer n has multiplicative inverse in $R^+ = \{R - \{0\}\}$, that R has a principal n^{th} root of

unity α, and that F is an $n \times n$ matrix such that $F[i,j] = \alpha^{ij}$, for $0 \leq i,\ j < n$. The vector $F(u)$ whose i^{th} component c_i is $\sum_{k=0}^{n-1} u_k \alpha^{ik}$, $0 \leq i < n$, is called the *Discrete Fourier Transform (DFT)* of u. The matrix F is *nonsingular*, its inverse F^{-1} has its $(i,j)^{th}$ element equal to $(1/n)\alpha^{-ij}$.

The vector $F^{-1}(u) = F^{-1}u$ with its i^{th} component

$$\frac{1}{n}\sum_{k=0}^{n-1} u_k \alpha^{-ik}, \qquad 0 \leq i < n,$$

is called the *inverse discrete Fourier transform (IDFT)* of u. It is easy to verify that the inverse transform of the transform of u is u itself:

$$F^{-1}F(u) = u.$$

6.3 Discrete Fourier Transforms and Polynomial Multipoint Evaluation and Interpolation at Roots of Unity

Let

$$p(x) = \sum_{i=0}^{n-1} u_i x^i$$

be an $(n-1)^{st}$-degree polynomial. This polynomial may be uniquely represented in various ways, in particular, by a list of its coefficients $u_0, u_1, \cdots, u_{n-1}$ or by a list of its values at distinct points $x_0, x_1, \cdots, x_{n-1}$. The process of finding the coefficient representation of a polynomial given by its values at $x_0, x_1, \cdots, x_{n-1}$ is called *interpolation.*

Converting the coefficient representation of a polynomial $\sum_{i=0}^{n-1} u_i x^i$ to its value representation at the points $\alpha^0, \alpha^1, \cdots, \alpha^{n-1}$ is equivalent to computing the Fourier transform of a vector $[u_0, u_1, \cdots, u_{n-1}]$. Likewise, the inverse Fourier transform is equivalent to interpolating a polynomial given its values at the n^{th} roots of unity.

6.4 Convolution Problem

- **Input:** A natural n and two column vectors u and v such that:

$$u = [u_0, u_1, \cdots, u_{n-1}]^T,$$

$$v = [v_0, v_1, \cdots, v_{n-1}]^T.$$

- **Output:** The convolution of u and v, denoted $w = u \odot v$, $w = [w_0, w_1, \cdots, w_{2n-1}]^T$, with

$$w_i = \sum_{j=0}^{n-1} u_j v_{i-j} = \sum_{j=0}^{i} u_j v_{i-j}, \tag{6.4}$$

$$u_k = v_k = 0 \quad if \quad k < 0, \quad or \quad k \geq n.$$

Thus $w_0 = u_0 v_0$, $w_1 = u_0 v_1 + u_1 v_0$, $w_2 = u_0 v_2 + u_1 v_1 + u_2 v_0$, and so on. Note that $w_{2n-1} = 0$ and is included here for convenience only.

We observe that the product of two $(n-1)^{st}$-degree polynomials

$$u(x) = \sum_{i=0}^{n-1} u_i x^i \quad and \quad v(x) = \sum_{j=0}^{n-1} v_j x^j$$

is a $(2n-2)^{nd}$-degree polynomial whose coefficients are exactly the components of the convolution of the coefficient vectors

$$[u_0, u_1, \cdots, u_{n-1}]^T \quad and \quad [v_0, v_1, \cdots, v_{n-1}]^T$$

of the original polynomials:

$$u(x)v(x) = \sum_{i=0}^{2n-2} [\sum_{j=0}^{i} u_j v_{i-j}] x^i.$$

If $u(x)$ and $v(x)$ are represented by their values at the n^{th} roots of unity, then we may simply multiply pairs of values at the corresponding roots of unity in order to obtain the value representation of the product of $[u(x)v(x)]$. We may write symbolically that:

$$u \odot v = F^{-1}(F(u) \bullet F(v)),$$

which suggests that the convolution of two vectors u and v is the inverse Fourier transform of the componentwise product of the transform of the two vectors. This is the convolution theorem, which we will prove in the next section (compare [AHU]).

6.5 Convolution Theorem

THEOREM **6.5.1**

$$Let \quad u = [u_0, u_1, \cdots, u_{n-1}, 0, \cdots, 0]^T,$$

and

$$u = [v_0, v_1, \cdots, v_{n-1}, 0, \cdots, 0]^T$$

be two column vectors of length $2n$. *Let*

$$F(u) = [u'_0, u'_1, \cdots, u'_{2n-1}]^T,$$

$$\textit{and} \quad F(v) = [v'_0, v'_1, \cdots, v'_{2n-1}]^T$$

be their Fourier transforms. Then

$$u \odot v = F^{-1}(F(u) \bullet F(v)).$$

PROOF. Since $u_i = v_i = 0 \quad for \quad n \leq i < 2n$, we note that

$$u'_l = \sum_{j=0}^{n-1} u_j \alpha^{lj} \quad and \quad v'_l = \sum v_k \alpha^{lk}, \quad for \quad 0 \leq l < 2n.$$

Thus

$$u'_l v'_l = \sum_{j=0}^{n-1} \sum_{k=0}^{n-1} u_j v_k \alpha^{l(j+k)}. \tag{6.5}$$

Let $F(w) = [w'_0, w'_1, \cdots, w'_{2n-1}]^T$. We may write:

$$w'_l = \sum_{p=0}^{2n-1} \sum_{j=0}^{2n-1} u_j v_{p-j} \alpha^{lp}. \tag{6.6}$$

Interchanging the order of summation in equation (6.5) and substituting k for $p - j$ yields:

$$w'_l = \sum_{j=0}^{2n-1} \sum_{k=-j}^{2n-1-j} u_j v_k \alpha^{l(j+k)}. \tag{6.7}$$

Since $v_k = 0$ for $k < 0$, we may raise the lower limit on the summation to $k = 0$. Likewise, since $u_j = 0 \quad j \geq n$, we may lower the upper limit on the outer summation to $n - 1$. The upper limit on the inner summation is at

least n, no matter what the value of j is. Thus we make these changes, (6.7) becomes identical to (6.5), hence $w'_l = u'_l v'_l$. We have now shown that:

$$F(u \odot v) = F(u) \bullet F(v), \tag{6.8}$$

from which it follows that:

$$u \odot v = F^{-1}(F(u) \bullet F(v)). \tag{6.9}$$

■

6.6 Positive and Negative Wrapped Convolution

DEFINITION **6.6.1** *Let* $u = [u_0, u_1, \cdots, u_{n-1}]^T$ *and* $v = [v_0, v_1, \cdots, v_{n-1}]^T$ *be two vectors of length* n. *The positive wrapped convolution of* u *and* v *is the vector:*

$$w^+ = [w_0^+, w_1^+, \cdots, w_{n-1}^+]^T,$$

where

$$w_i^+ = \sum_{j=0}^{i} u_j v_{i-j} + \sum_{j=i+1}^{n-1} u_j v_{n+i-j}. \tag{6.10}$$

The negative wrapped convolution of u *and* v *(also called* **cyclic convolution***) is the vector:*

$$w^- = [w_0^-, w_1^-, \cdots, w_{n-1}^-]^T,$$

where

$$w_i^- = \sum_{j=0}^{i} u_j v_{i-j} - \sum_{j=i+1}^{n-1} u_j v_{n+i-j}. \tag{6.11}$$

6.6.1 Problem 1.1 (Positive and Negative Wrapped Convolution)

Given the value $u_0, v_0, u_1, v_1, \cdots, u_n, v_n$, let us compute the coefficients w_i^+ and w_i^- of the polynomials

$$w^+(x) = \sum_{i=0}^{n} w_i^+ x^i, \qquad w_i^-(x) = \sum_{i=0}^{n} w_i^- x^i,$$

where

$$w_i^+ = w_i + \hat{w}_i, \qquad w_i^- = w_i - \hat{w}_i,$$

$$w_i = \sum_{j=0}^{i} u_j v_{i-j}, \quad \hat{w}_i = \sum_{j=i+1}^{n} u_j v_{n+i-j}, \quad i = 0, 1, \cdots, n.$$

Here we assume that $\hat{w}_n = 0$. In fact,

$$w^+(x) = u(x)v(x) \mod (x^{n+1} - 1),$$

$$w^-(x) = u(x)v(x) \mod (x^{n-1} + 1),$$

with

$$u(x) = \sum_{j=0}^{n} u_j x^j \quad \textit{and} \quad v(x) = \sum_{j=0}^{n} v_j x^j. \quad \blacksquare$$

6.6.2 Solution of Problem 1.1

To compute w_i^+, we apply *DFT* twice to determine the values of the two polynomials $u(x) = \sum_{i=0}^{n} u_i x^i$ and $v(x) = \sum_{j=0}^{n} v_j x^j$ at points $x_k = \alpha^k$ with $k = 0, 1, \cdots, n-1$, where $\alpha = \alpha_n$ is a primitive n^{th} root of unity, that is,

$$\alpha_n^n = 1, \quad \alpha_n^s \neq 1, \quad \textit{for} \quad 1 < s < n.$$

We may then compute the product $w(x) = u(x)v(x)$ at the same points α^k, which gives us $w(\alpha^k)$. By observing that $w^+(\alpha^k) = (\alpha^k)$ with $k = 0, 1, \cdots, n$, we may interpolate

$$w(x) = \sum_{i=0}^{n} w_i x_i + x^{n+1} \sum_{i=0}^{n-1} \hat{w}_i x^i$$

to the polynomial $w^+(x)$ by means of inverse *DFT*. All the three *DFTs* are at $n + 1$ points, which saves about 50% arithmetic operations *(ops)* versus the algorithms that compute the convolution vector $w = (w_0, \ldots, w_{n-1})^T$.

To compute the negative wrapped convolution w_i^-, we first determine the values

$$\sum_{i=0}^{n} u_i \psi^i \alpha^{ij} \quad \text{and} \quad \sum_{i=0}^{n} v_i \psi^i \alpha^{ij}$$

by applying *DFT*, and then compute their pairwise products, where ψ is a primitive $2(n+1)^{th}$ root of unity. We use these products to determine the values of the polynomial

$$\sum_{i=0}^{n} (w_i^-) \psi^i x^i$$

at $x = \alpha^k$, for $k = 0, 1, \cdots, n$. The last step consists in recovering the coefficients $(w_i^-)\psi^i$ by means of the inverse *DFT*. ∎

6.6.3 Problem 1.2

- **Input:** A natural n and two vectors, $u = [u_0, u_1, \cdots, u_{n-1}]^T$ and $v = [v_0, v_1, \cdots, v_{n-1}]^T$.

- **Output:** $u \odot v = w = [w_0, w_1, \cdots, w_{2n-2}]^T$, $w_k = \sum u_i v_{k-i}$ for $k = 0, 1, \cdots, 2n-2$.

6.6.4 Solution 1 of Problem 1.2.

Recall that w is the coefficient vector of the polynomial

$$w(x) \quad = \quad \sum_{i=0}^{2n-2} w_i x^i \quad = \quad u(x)v(x),$$

where

$$u(x) = \sum_{i=0}^{n-1} u_i x^i, \qquad v(x) = \sum_{i=0}^{n-1} v_i x^i.$$

Denote: $F_s(u)$ is the *DFT* of a vector u on s points α_s^i, $i = 0, 1, \cdots, s-1$, that is, (on the set of the s^{th} roots of unity). $F_s^{-1}(v)$ is the inverse *DFT* of a vector v. $u \odot v$ is obtained by multiplying the i^{th} component of u and v. Then

$$w \quad = \quad F_s^{-1}(F_s(u) \bullet F_s(v)) \text{ for any } \quad s > 2n.$$

To save the computer memory space, we may use positive and negative wrapped convolution, which gives us an alternative solution of Problem 1.2.

6.6.5 Solution 2 of Problem 1.2.

Compute the coefficient vectors w_+ and w_- of the polynomials

$$w(x) \ mod \ (x^n - 1) = w_+(x) \tag{6.12}$$

and

$$w(x) \ mod \ (x^n + 1) = w_-(x). \tag{6.13}$$

Then

$$w_+(x) + w_-(x) = 2w(x) \ mod \ x^n. \tag{6.14}$$

We may write

$$w_+ = F_n^{-1}(F_n(u) \bullet F_n(v)), \quad S_{2n}w_- = F_n^{-1}(F_n(S_{2n}u) \bullet F_n(S_{2n}v)), \tag{6.15}$$

where

$$S_t[c_0, c_1, \cdots, c_{n-1}]^T = (c_0, c_1\alpha_t, \cdots, c_{n-1}\alpha_t^{n-1}),$$

and α_t is a primitive t^{th} root of unity.

Hereafter, F stands for F_n and S stands for S_{2n}.

6.7 Storage Efficient Computation of $w(x)$ mod x^n.

ALGORITHM **6.7.1** *(***Storage computation***)*

Input: *A natural n and two column vectors u and v, each of length n, such that:*

$$\begin{aligned} u &= [u_0, u_1, \cdots, u_{n-1}]^T, \\ v &= [v_0, v_1, \cdots, v_{n-1}]^T. \end{aligned}$$

Output: *Compute:*

$$w_+ = F^{-1}(F(u) \bullet F(v)), \tag{6.16}$$

$$w_- = S^{-1}(F^{-1}(F(Su) \bullet F(Sv))), \tag{6.17}$$

and

$$w = \frac{w_+ + w_-}{2}. \tag{6.18}$$

6.7.1 Space Computation for Algorithm 6.7.1

Since the input vectors u and v are each of length n, the computation of the vectors w_+, w_-, and w as defined by equations (6.16), (6.17), and (6.18), respectively, will require a storage space of $4n$ words. The purpose of the next algorithm is to decrease the storage space of $4n$ words to $3n$ by overwriting the input vectors u or v by the output.

6.7.2 The Storage Space Efficient Computation

1. Use $2n$ words storage space to store the components of u and v.

2. Use the same $2n$ words storage space defined in step 1 to store the components of $F(u)$, and $F(v)$. Overwrite u by $F(u)$, and v by $F(v)$.

3. Use n words storage space to compute the coefficients of $F(u) \bullet F(v)$. (At this point we have $3n$ words in the working array.)

4. Use the same storage space as in step 3 to compute the components of vector w_+, that is, $F^{-1}(F(u) \bullet F(v))$.

5. Recover the coefficients of vectors u and v by computing $F^{-1}(F(u))$ and $F^{-1}(F(v))$, respectively. Use the same storage space as in step 2.

6. Having recovered the coefficients of u and v compute Su and Sv using the working array of step 5.

7. Use n words storage space to compute at first

(a) $F(Su) \bullet F(Sv)$, then

(b) $F^{-1}(F(Su) \bullet F(Sv))$, and finally

(c) $S^{-1}(F^{-1}(F(Su) \bullet F(Sv)))$. (At this point we have $4n$ words in the working array.)

8. We may reuse the storage space already used in steps 4 or 7 to obtain the components of vector w as defined by equation (6.18). At this stage of the computation the storage space of $4n$ is reduced to $3n$ words.

u	w_-	v	w_+
$F(u)$		$F(v)$	
			$F(u) \bullet F(v)$
			$w_+ = F^{-1}(F(u) \bullet F(v))$
$u = F^{-1}(F(u))$		$v = F^{-1}(F(v))$	
Su		Sv	
	$F(Su) \bullet F(Sv)$		
	$F^{-1}(F(Su) \bullet F(Sv))$		
$Su = F^{-1}(F(Su))$	$w_- = S^{-1}(F^{-1}(F(Su) \bullet F(Sv)))$	$Sv = F^{-1}(F(Sv))$	
			$w = (w_+ + w_-)/2$

Table 6.1: Economization of space in fast convolution computation.

As shown in Table 6.7.2, the storage space of $4n$ can be decreased to $3n$ if we overwrite the input vectors u and v by the output, during the computation process. We will achieve a further reduction of memory space if we assume that $n = rh$, where r and h are integers, with $r = O(1)$ (say 2, 3, 4, 5, 8, 10,

etc...). We denote:

$$u(x) = \sum_{i=0}^{r-1} u_i(x)y^i,$$
$$v(x) = \sum_{i=0}^{r-1} v_i(x)y^i, \quad y = x^s, \text{ for } s = 1, 2, \cdots, r.$$

We rewrite $w(x)$ as:

$$w(x) \ mod \ x^n = u(x)v(x) \ mod \ x^n = \sum_{k=0}^{r-1}\sum_{i=0}^{k} u_i(x)v_{k-i}y^k \ mod \ x^n,$$

and successively compute

$$w_k(x) = \sum_{i=0}^{K} u_i(x)v_{k-i}(x) \ mod \ x^{n-ks} \quad for \ \ k = r-1, \ r-2, \cdots, 1, 0.$$

Having computed $w_k(x)$, we release the space allocated for $u_{k-1}(x)$ and $v_k(x)$, and then reuse it as a working array. When $w_k(x)$ and $w_{k-1}(x)$ have been computed, we obtain the components $w_{kh}, \cdots, w_{kh+h-1}$ *of* w. We then print the newly computed components and reuse the storage space for more computations. This way, going from $k = r-1$ down to 0, we release more and more space.

Let us now specify the computation of $w_k(x)$ for $k = r-1$, where r is an integer. We will recall that $[w(x) \ mod \ x^n]$ is obtained as $[\ (w_+(x)+w_-(x))/2]$, so we will first compute $[w_k(x)]_+$, then $[w_k(x)]_-$, and finally $[w_k(x)]$.

6.7.3 Computation of $[(w_k(x))_+ + (w_{k-1}(x))_+]$ for $k = r-1$.

This computation gives h leading coefficients of $[(w(x))_+ \ mod \ x^n]$.

ALGORITHM **6.7.2** *(***Positive Wrapped Convolution***)*

Input: *Two integers* r *and* s, *the components*

$$u_0(x),\ u_1(x),\ \cdots,\ u_{r-2}(x),\ u_{r-1}(x) \text{ of vector } u \text{ and}$$

$$v_0(x),\ v_1(x),\ \cdots,\ v_{r-2}(x),\ v_{r-1}(x),\ \textit{of vector } v.$$

Output: *The components* $(w_+)_k$ *of the vector* w_+ *of equation (6.16).*

Computation:

1. *First compute the* DFT *of each component of* $u(x)$ *and* $v(x)$, *that is, output*

$$F(u_0),\ F(u_1),\ \cdots,\ F(u_{r-2}),\ F(u_{r-1})$$

and

$$F(v_0),\ F(v_1),\ \cdots,\ F(v_{r-2}),\ F(v_{r-1}).$$

2. *Compute each pairwise product* $F(u)_{r-s} \bullet F(v)_{s-1}$ *and then apply the inverse DFT to the vector of all products, that is, compute*

$$F^{-1}(F(u)_{r-s} \bullet F(v)_{s-1}) \quad \textit{for} \quad s = 1, 2, \cdots, r.$$

3. *Compute* $(w_+)_k = \sum_{s=1}^{r}(F^{-1}(F(u)_{r-s} \bullet F(v)_{s-1}))$ ■

6.7.4 Computation of $[(w_k(x))_- + (w_{k-1}(x))_-]$ for $k = r-1$.

This computation gives h leading coefficients of $[(w(x))_-\ mod\ x^n]$.

ALGORITHM **6.7.3** *(***Negative Wrapped Convolution***)*

Input: *Two integers,* r *and* s*, and two vectors,* u *and* v*, whose components are*

$$u_0(x), \quad u_1(x), \quad \cdots, \quad u_{r-2}(x), \quad u_{r-1}(x),$$

and

$$v_0(x), \quad v_1(x), \quad \cdots, \quad v_{r-2}(x), \quad v_{r-1}(x).$$

Output: *The component* $(w_-)_k$ *of the vector* w_- *of equation (6.17).*

Computation:

1. *Compute*

$$F(Su)_0, \quad F(Su)_1, \quad \cdots, \quad F(Su)_{r-2}, \quad F(Su)_{r-1}$$

and

$$F(Sv)_0, \quad F(Sv)_1, \quad \cdots, \quad F(Sv)_{r-2}, \quad F(Sv)_{r-1}.$$

2. *Compute each pairwise product*

$$F(Su)_{r-s} \bullet F(Sv)_{s-1},$$

and then apply the inverse DFT to the vector of all products, that is, compute

$$S^{-1}(F^{-1}(F(Su)_{r-s} \bullet F(Sv)_{s-1})) \quad with \quad s = 1, 2, \cdots, r.$$

3. *Add all the components obtained in the previous step, that is, compute*

$$(w_k)_- \; = \; \sum_{s=1}^{r} S^{-1}(F^{-1}(F(Su)_{r-s} \bullet F(v)_{s-1})) \quad \blacksquare$$

Thus, only $2h$ words of storage space are needed at the stage of computing $(w_+)_{r-1}$ (h first components of w_+) in the working array (in addition to the $2n$ words for the input) assuming that when the need arises, we recover u_s from $F(u_s)$ and $F(Su_s)$ and v_s from $F(v_s)$ and $F(Sv_s)$ for any S. At the stage of computing $[(w_+)_{r-1} + (w_-)_{r-1}]$, we need only h words in the working array. For $(w_+)_s$ in $[(w_+)_s + (w_-)_s]$, we need less than $2(r-s-1)h$ words for $(w_+)_{r-1}$ and $[(w_+)_{r-1} + (w_-)_{r-1}]$, respectively.

REMARK **6.7.1** *If we do not want to recover the values of* $u_s(x)$ *and* $v_s(x)$ *from the vectors* $F(u)_s$, $F(v)_s$, $F(Su)_s$, *and/or* $F(Sv)_s$ *and if we want to store them (that is,* $u_s(x)$ *and* $v_s(x)$*), then we obviously need more storage space. But with more storage space, a higher use of parallelism is possible.*

Appendix A. **An Optimal Prefix-Sums Algorithm.**

Let us consider a sequence of n elements $\{x_1, x_2, \cdots, x_n\}$ drawn from a set S with a binary **associative** operation, denoted by $*$. The **prefix sums** of this sequence are the n partial sums (or products) defined by

$$s_i = x_1 * x_2 * \ldots * x_i, \quad 1 \leq i \leq n.$$

A trivial sequential algorithm computes s_i from s_{i-1} with a single operation by using the identity

$$s_i = s_{i-1} * x_i, \quad for\ 2 \leq i \leq n,$$

and hence takes $O(n)$ time. This algorithm is inherently sequential.

We can derive a fast parallel algorithm to compute the prefix sums by using a balanced binary tree. The following algorithm describes the steps needed to obtain the data stored in the nodes at height 1 and the steps needed to obtain the prefix sums after the recursive call on the nodes at height 1 terminates.

Algorithm A.1 (Prefix Sums)

- **Input:** *An array of* $n = 2^k$ *elements* $(x_1, x_2, \ldots, x_n)$, *where* k *is a nonnegative integer.*
- **Output:** *The prefix sums* s_i, *for* $1 \leq i \leq n$.

Computation

begin

1. **if** $n = 1$ **then** $\{set\ s_1 := x_1$; **exit**$\}$

2. **for** $1 \leq i \leq n/2$ **pardo**

Set $y_i := x_{2i-1} * x_{2i}$

3. Recursively, compute the prefix sums of $\{y_1, y_2, \ldots, y_{n/2}\}$ *and then store them in* $z_1, z_2, \ldots, z_{n/2}$.

4. **for** $1 \leq i \leq n$ **pardo**

$$\begin{aligned} \{i\ even &: set\ s_i := z_{i/2} \\ i = 1 &: set\ s_1 := x_1 \\ iodd > 1 &: set\ s_i := z_{(i-1)} * x_i\} \end{aligned}$$

end

This algorithm runs on the EREM PRAM model (see chapter 2).

Proposition A.1 (Parallel Prefix Computation, *[LF]) The prefix-sums Algorithm (Algorithm A.1) computes the sums of* n *elements in*

$$O\left(log\ n,\ \frac{n}{log\ n}\right) \tag{A.1}$$

PROOF. We assume (without loss of generality) that $n = 2^k$, where k is a positive integer. Step 1 of the algorithm A.1 takes only $O(1)$ sequential time. Step 2 and 4 can be executed in $O(1)$ parallel steps using $O(n)$ operations. Let $T(n)$ and $W(n)$ be the running time and the work required by the

algorithm, respectively. $T(n)$ and $W(n)$ satisfy the following recurrences:

$$\begin{aligned} T(n) &= T(\frac{n}{2}) + a \\ W(n) &= W(\frac{n}{2}) + bn, \end{aligned}$$

where a and b are constants. ■

Appendix B. **Parallel Polynomial Multiplication and Convolution .**

We begin this appendix by describing a *Fast Fourier Transform (FFT)* algorithm, which can be implemented in $O_A(log\ n,\ n\ log\ n)$, on the *EREW PRAM*. We then apply this *FFT* algorithm to polynomial multiplication and convolution.

Algorithm B.1 (Fast Fourier Transform)

- **Input:** *An n-dimensional vector x whose entries are complex numbers, and* $w = e^{i\frac{2\pi}{n}}$, *where n is assume to be a power of 2.*
- **Output:** *The vector y that is the* DFT *of x.*

begin

1. **if** $n = 2$ **then** { *Set* $y_1 := x_1 + x_2, \quad y_2 := x_1 - x_2$, **exit** }

2. **for** $0 \leq l \leq \frac{n}{2} - 1$ **pardo**

$$Set \quad u_l := x_l + x_{\frac{n}{2}+l}$$

$$Set \quad v_l := w^l(x_l - x_{\frac{n}{2}+l}$$

3. Recursively, compute the DFT *of the two vectors*

$$[u_0, u_1, \cdots, u_{\frac{n}{2}-1}]$$

and

$$[v_0, v_1, \cdots, v_{\frac{n}{2}-1}],$$

and store the results in vectors

$$z^{(1)} = [z_0^{(1)}, z_1^{(1)}, \cdots, z_{\frac{n}{2}-1}^{(1)}]$$

and

$$z^{(2)} = [z_0^{(2)}, z_1^{(2)}, \cdots, z_{\frac{n}{2}-1}^{(2)}],$$

respectively.

4. **for** $0 \leq j \leq n-1$ **pardo**

$$j \text{ even}: \ Set\ y_j := z_{\frac{j}{2}}^{(1)}$$

$$j \text{ odd}: \ \ Set\ y_j := z_{\frac{j-1}{2}}^{(2)}$$

end

Theorem B.1 *The FFT algorithm computes the DFT of an n-dimensional vector at the cost* $O_A(log\ n,\ n\ log\ n)$.

PROOF. Let $T(n)$ *and* $W(n)$ be the running time and the total number of operations, respectively. Then

$$T(n) = T(n/2) + O(1)$$

$$W(n) = 2W(n/2) + O(n),$$

and hence $T(n) = O(log\ n)$ and $W(n) = O(n\ log\ n)$. ∎

B.2 Polynomial Multiplication Using The FFT Algorithm

Let

$$p(x) = \sum_{k=0}^{n-1} a_k x^k,\ and\ q(x) = \sum_{k=0}^{n-1} b_K x^k$$

be two polynomials of degree $n-1$.

The product $r(x) = p(x)q(x) = \sum_{k=0}^{2n-2} c_k x^k$ is such that $c_k = \sum_{j=0}^{k} a_j b_{k-j}$, where a_k and b_{j-k} are assumed to be 0 if their indices fall outside the range $[0, 1, \ldots, n-1]$.

Theorem B.2 *The coefficients of the product of two polynomials of degree* $n-1$ *can be computed at the cost* $O_A(\log n,\ n)$.

PROOF. Multiplication of two polynomials is exactly the same problem as convolution of their coefficient vectors, and we recall that parallel algorithm for convolution runs within the same asymptotic cost bounds that we stated for the parallel arithmetic complexity of the FFT algorithm. ∎

B.3 Convolution Using the FFT Algorithm

If

$$a = [a_0, a_1, \cdots, a_{n-1}]^T$$

and

$$b = [b_0, b_1, \cdots, b_{n-1}]^T,$$

then

$$a \odot b = [c_0, c_1, \cdots, c_{2n-1}]^T,$$

such that $c_k = \sum_{j=0}^{k} a_j b_{k-j}$, where a_j *and* $b_j = 0$ *for* $j > n-1$.

Corollary B.1 *The convolution of two vectors of dimension* n *can be computed at the cost* $O_A(\log n,\ n \log n)$. ∎

Bibliography

[Am] W. F. Ames, Numerical Methods for Partial Differential Equations, *Academic Press*, NY, 1977.

[AHU] V. Aho, E. Hopcroft, D. Ulmam, The Design And Analysis Of Computer Algorithms, *Addison-Wesley*, Reading, Mass., 1974.

[AR] D. Armon, J. Reif, An optimal Space Efficient Parallell Nested Dissection Algorithm, *4 Ann. ACM Symposium on Parallel Algorithms and Architectures*, San Diego, CA, pp. 344–352, 1992.

[AT] M. Augenstein, A. Tenenbaum, Data Structure and PL/1 Programming, *Prentice Hall*, NJ, 1979.

[BM] A. Borodin, I. Munro, The Computational Complexity of Algebraic and Numeric Problems, *American Elsevier*, New York, 1975.

[BG73] G. Birkhoff, J. George, Elimination by Nested dissection, in Complexity of Sequential and Parallel Numerical Algorithms, *Traub JB, ed. Academic Press*, New York, 1973.

[Br74] R. P. Brent, The Parallel Evaluation of General Arithmetic Expressions, *J. ACM*, 21, 2, pp. 201–206, 1974.

[BGN] B. Buzbee, G. Golub and C.Nielson, On Direct Methods for Solving Poisson's Equations, *SIAM J. Num. Anal.*, 7, 4, pp. 627–655, 1970.

[BD] B. Buzbee, F. Door, The Direct Solution of the Harmonic Equation on Rectangular Regions and the Poisson's Equation on Irregular Regions, *SIAM J. Num. Anal.*, 11, 4, pp. 753–763, 1974.

[BMP] D. Bini, B. Meini, V. Y. Pan, to appear.

[BP] D. Bini, V. Y. Pan, Polynomial and Matrix Computations: v. 1, Fundamental Algorithms, *Birkhauser, Boston, Mass.*, 1994.

[BP93] D. Bini, V. Y. Pan, Improved Parallel Computation with Toeplitz-like and Hankel-like Matrices, *Linear Algebra and Applications*, 188, 189, pp. 3–29, 1993.

[Ba] S. Barnet, Matrices Methods and Applications, Oxford Applied Mathematics and Computing Series, *Oxford University Press*, 1992.

[Co81] S. A. Cook, Towards a Complexity Theory of Synchroneous Parallel Computation, *Enseign. Math.*, 27, pp. 99–124, 1981.

[CR] J. Cheriyan, J. H. Reif, Parallel and output sensitive algorithms for combinatorial and linear algebra problems, in *Proceedings*, 5th ACM *Symposium* on Parallel Algorithms and Architectures, *ACM Press*, New York, pp. 50–56, 1993.

[Cs] L. Csanky, Fast Parallel Matrix Inversion Algorithms, *SIAM J. Computing*, 5, pp. 618-623, 1976.

[CW] D. Coppersmith, S. Winograd, Matrix Multiplication via Arithmetic Progression, *J. of Symbolic Comp.*, 9, 3, pp. 251–280, 1990.

[DB] G. Dahlquist, Å. Björck, Numerical Methods, *Prentice-Hall*, Englewood Cliff, NJ, 1974.

[E] W. Eberly, On Efficient Band Matrix Arithmetic, *Proc. 33 Ann. IEEE Computer Society Press*, IEEE Symp. FOCS, pp. 457–463, 1992.

[E95] W. Eberly, Fast parallel band matrix arithmetic, *Proc 6th. Ann. ACM-SIAM Symp. on Discrete Algorithms*, San Francisco, CA, *ACM Press*, New York, and *SIAM Publication*, Philadelphia, pp. 132–138, 1995.

[EG88] D. Eppstein, Z. Gallil, Parallel Algorithmic Techniques for Combinatorial Computing, *Ann. Rev. Comput. Sci.*, 3, pp. 233–283, 1988.

[Ge73] J. A. George, Nested Dissection of a Regular Finite Mesh, *SIAM J. Numer. Anal.*, 10, pp. 345–367, 1973.

[GJ] M. R. Garey, D. S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, *W. H. Freeman*, San Francisco, CA, 1979.

[GK] D. H. Greene, D. E. Knuth, Mathematics for the Analysis of Algorithms, *Progress in Computer Science*, Vol. 1, Birkhäuser, Boston* Basel* Stuttgart, 1981.

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

[GLi] A. George, J. W.-H Liu, Computer Solution of Large Positive Definite Systems, *Prentice-Hall*, Englewood Cliffs, NJ, 1981.

[GT] J. R. Gilbert, R. E. Tarjan, The Analysis of Nested Dissection Algorithms, *Numer. Math.*, 50, pp. 377–404, 1987.

[Ho] R. W. Hockney, A fast Direct Solution of Poisson Equation using Fourier Analysis, *J. ACM*, 12, pp. 95–113, 1965.

[He] D. Heller, Some Aspects of the Cyclic Reduction Algorithm for Block Tridiagonal Linear Systems, *SIAM J. Num. Anal.*, 13, 4, pp. 484–496, 1976.

[Ho] R. W. Hockney, A Fast Direct Solution of Poisson Equation using Fourier Analysis, *J. ACM*, 12, pp. 95–113 , 1965.

[JJ] J. JáJá, Introduction to Parallel Algorithms, *Addison-Westley Publishing Company*, Reading, MA, 1992.

[Kn] D. Knuth, The Art of Computer Programming: Seminumerical Algorithms, second edition. *Addison-Wesley*, Reading, MA, 1981.

[Ko] P.M. Kogge, Parallel Solution of Recurrence Problems. *IBM Journal of Research and Development*, 18(2), pp. 138–148, 1974.

[KP91] E. Kaltofen, V.Pan, Processor Efficient Parallel Solution of Linear Systems over an Abstract Field, *in Proc. 3 Ann. ACM Symposium on Parallel Algorithms and Architectures, ACM Press*, New York, pp. 180–191, 1991.

[KP92] E. Kaltofen, V. Pan, Processor Efficient Parallel Solution of Linear System II. The Positive Characteristic and Singular Cases, *Proc. 33 Ann. IEEE Symp. FOCS, IEEE Computer Society Press*, pp. 714–723, 1992.

[KR] R. Karp, V. Ramachandran, A Survey of Parallel Algorithms for Shared Memory Machines, *Handbook for Theoretical Computer Science* (J. van Leeuwen Editor), North Holland, Amsterdam, pp. 869–941, 1990.

[KS] P.M. Kogge, H.S. Stone, A Parallel Algorithm for the Efficient Solution of a Class of Recurrence Equations. *IEEE Translations on Computers*, 22(8), pp. 786–792, 1973.

[LF] R.E. Ladner, M.J. Fisher, Parallel Prefix Computation, *ACM Press*, 27, 4, pp. 831–838, 1980.

[LRT79] R.J. Lipton, D. Rose, R.E. Tarjan, Generalized Nested Dissection, *SIAM J. Numer. Anal.*, 16, 2, pp. 346–358, 1979.

[LP] L. Lapidus, G.F. Pinder, Numerical Solutions of Partial Differential Equations in Science and Engineering, *Willey*, NY, 1982.

[MP] I. Munro, M. Paterson, Optimal Algorithms for Parallel Polynomial Evaluation. *Journal of Computer and System Sciences*, 7(2), pp. 189–198, 1973.

[MT] L. M. MilneThomson, The Calculus of Finite Differences, *MacMillan*, New York, 1933.

[P85] V. Y. Pan, Fast and Efficient Parallel Algorithms for the Exact Inversion of Integer Matrices, *Tech. Report 85-2. Department of Computer Science, State University of New York, Albany, NY*, 1985.

[P87] V. Y. Pan, Complexity of Parallel Matrix Computations, *Theoret. Comput. Sci.*, 54, pp. 65–85, 1987.

[P91] V. Y. Pan, 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, 77, Springer, Berlin, pp. 27–56, 1991.

[P92,a] V. Y. Pan, Complexity of Computations with Matrices and Polynomials, *SIAM Review*, 34, 9, pp. 225-262, 1992.

[P92] V. Y. Pan, Parametrization of Newton Iteration for Computation with Structured Matrices and Applications, *Computers and Mathematics (with Applications)*, 24, 3, pp. 61–75, 1992.

[P93] V. Y. Pan, Parallel Solution of Sparse Linear and Path Systems, in *Synthesis of Parallel Algorithms* (J.H. Reif Editor), 621-678, Morgan Kaufmann publisher, San Mateo, California, 1993.

[P94] V. Y. Pan, Improved Parallel Solution of a Triangular Linear System. *Computers and Math. (with Applications)*, 11, 27, pp. 41–43, 1994.

[P94,a] V. Y. Pan, Parallel Computation of a Krylov Matrix for a Sparse and Structured Input, *Math and Computer Modelling*, to appear.

[PP] V. Y. Pan, F. Preparata, Supereffective Slow-Down of Parallel Computations, *Proc. 4 Ann. ACM Symposium on Parallel Algorithms and Architectures, ACM Press*, New York, pp. 402–409, 1992.

[PR85] V. Y. Pan, J. H. Reif, Efficient Parallel Solution of Linear Systems, *Proc. 17 Ann. ACM Symp. on Theory of Computing*, pp. 402–409, ACM Press, New York, 1985.

[PR93] V. Y. Pan, J. H. Reif, Fast and Efficient Solution of Sparse Linear Systems, *SIAM J. on Computing*, 22, 6, pp. 1227-1250, 1993.

[PSA] V. Y. Pan, I. Sobze, A. Atinkpahoun, Improved Band Matrix Algorithms, Tech. Report, *Intern. Computer Science Institute*, Berkeley, CA, 1993.

[PSA,a] V. Y. Pan, I. Sobze, A. Atinkpahoun, Optimum parallel computations with Banded Matrices, *in Proceedings, 5th ACM-SIAM Symposium on Discrete Algorithms, ACM Press*, New York, and *SIAM Publications*, Philadelphia, pp. 649–658, 1994.

[PSA,b] V. Y. Pan, I. Sobze, A. Atinkpahoun, On Parallel Computation with Band Matrices, *Information and Computation, to appear.*

[Q87] M. J. Quinn, Designing effective algorithms for parallel computers, *McGraw-Hill, New York*, 1987.

[Re93] J. H. Reif, Synthesis of Parallel Algorithms, *edited by J. H. Reif Editor, Morgan Kaufmann Publishers, San Mateo, CA*, 1993.

[S] I. Sobze. Parallel Algorithms for Banded Linear Systems of Equations. Ph.D. Thesis, *CUNY Graduate Center*, New York, 1994.

[Sc] J.T. Schwartz, Fast Probabilistic Algorithms for Verification of Polynomial Identities, *J. ACM*, 27, 4, pp. 701–717, 1980.

[Sn] M. Snir, On Parallel Searching, *SIAM J. Comput.*, 14, pp. 688–708, 1985.

[Sp] M. Spiegel, Calculus of Finite Differences and Difference Equations, *Schaum's Outline Series, McGraw-Hill*, 1971

[St69] V. Strassen, Gaussian Elimination is Not Optimal, *Numer. Math.*, 13, pp. 354–356, 1969.

[Sw] P. Swarztrauber, A Direct Method for the Direct Solution of Separable Elliptic Equations, *SIAM J. Num. Anal.*, 11, 6, pp. 1137–1149, 1974.

[Swe74] R. Sweet, A Generalized Cyclic Reduction Algorithm, *SIAM J. Num. Anal.*, 11, 3, pp. 507–520, 1974.

[Swe77] R. Sweet, A Cyclic Reduction Algorithm for Solving Block Tridiagonal Systems of Arbitrary Dimension, *SIAM J. Num. Anal.*, 14, 3, pp. 706–720, 1977.

[UP] S. Ursic, C. Patarra, Exact Solution of Systems of Linear Equations with Iterative Methods, *SIAM J. Alg. Discrete Methods*, 4, pp. 111-115, 1983.

[Val] L. G. Valiant, Parallelism in Comparison Problems, *SIAM J. Computing*, 4, 3, pp. 348-355, 1975.

[Val1] L. G. Valiant, On Non-Linear Lower Bounds in Computational Complexity, *Peoc. 7th Ann. ACM Symp. on Theory of Computing, ACM Press*, New York, 1975.

[Wi70] S. Winograd, On the Number of Multiplications necessary to Compute certain Functions, *Comm. Pure and Applied Math.*, 23, pp. 165–179, 1970.

[Zi] R.E. Zippel, Probabilistic Algorithms for Sparse Polynomials, *Proc. EUROSAM 79, Lecture Notes in Computer Science, 72*, Springer, Berlin, pp. 216–226, 1979.