LU FACTORIZATION AND PARALLEL EVALUATION OF CONTINUED FRACTIONS

ÖMER EĞECIOĞLU
Department of Computer Science
University of California at Santa Barbara
omer@cs.ucsb.edu

Abstract

Consider the parallelization of the LU decomposition part of the LU Algorithm for an $n \times n$ tridiagonal linear system, and the related problem of evaluation of continued fractions. Both problems have known optimal parallel algorithms. We note an interesting connection between the two: in the same matrix-chain product formulation of the recurrences, prefix products solve one problem, while suffix products solve the other. More precisely, direct application of the parallel prefix algorithm for parallel evaluation of continued fractions results in an algorithm that requires $O(\frac{n^2}{\log n})$ processors and takes $O(\log^2 n)$ parallel time, which is suboptimal. A suffix product formulation for continued fractions together with parallel prefix can be used for parallel tridiagonal LU decomposition optimally in $O(\log n)$ time using $O(\frac{n}{\log n})$ processors.

Keywords: Tridiagonal system, LU decomposition, continued fraction, prefix product, suffix product.

1 Introduction

The first n convergents of a general continued fraction (CF) can be computed optimally in logarithmic parallel time using $O(\frac{n}{\log n})$ processors. An algorithm for this computation which uses a matrix representation of CFs due to Milne-Thomson ([1], pp. 108–110), in combination with a parallel prefix algorithm for matrix products appears in [2]. Continued fractions can be used to calculate quadratic surds, solutions of second order linear recurrences, and various other (numerical and matrix) sequences whose recursions are derived from continued fraction expansions.

Tridiagonal systems of equations arise frequently in the solution of partial differential equations, interpolation, and other engineering applications. There is a large body of literature on the solution of tridiagonal systems on parallel machines, initiated with the recursive doubling algorithm of Stone [3] which makes use of the relationship between tridiagonal systems and linear recurrences to solve this problem in $O(\log n)$ parallel steps using n processors. This algorithm can be used to solve banded systems as well. See [4, 5, 6, 7]. Parallel prefix based algorithms have also been used for the solution of tridiagonal systems with limited number of processors on the hypercube multiprocessor [8].

The LU decomposition algorithm is one of the most efficient existing sequential algorithms for the solution of $\mathbf{Ax} = \mathbf{d}$ where $\mathbf{A}$ is a nonsymmetric tridiagonal matrix. A direct application of CF expansion and convergent calculations for the parallelization of the LU decomposition results in a suboptimal parallel algorithm, because with this formulation the convergents that are needed to be computed are not the first n convergents of a single CF, but rather the full evaluation of n different (but related) CFs of varying lengths. This straightforward application of CF based method results in an $O(\log^2 n)$ parallel time algorithm using $O(\frac{n^2}{\log n})$ processors. In this paper we show that with a slight twist, an alternate formulation using suffix products of certain matrix chains using a backward evaluation of CFs results in an optimal parallel algorithm for the parallel LU decomposition of nonsymmetric tridiagonal matrices.

We note that most of the analytical development used here was introduced by Kogge [9], who presented general algorithms for solving recurrence relations efficiently in parallel for a large class of recurrences.

2 LU Decomposition and Problem Statement

We consider the linear system

$$\mathbf{Ax} = \mathbf{d} \tag{1}$$

where $\mathbf{A}$ is a tridiagonal matrix of order n of the form shown in (2), $\mathbf{x} = (x_0, x_1, \ldots, x_{n-1})^T$ is the vector of unknowns, and $\mathbf{d} = (d_0, d_1, \ldots, d_{n-1})^T$ is a vector of dimension n.

$$\mathbf{A} = \begin{bmatrix} b_0 & c_0 & & & & \\ a_1 & b_1 & c_1 & & & \\ & a_2 & b_2 & c_2 & & \\ & & \ddots & \ddots & \ddots & \\ & & & a_{n-2} & b_{n-2} & c_{n-2} \\ & & & & a_{n-1} & b_{n-1} \end{bmatrix} \qquad (2)$$

In the LU factorization, $\mathbf{A}$ is decomposed into a product of two bidiagonal matrices $\mathbf{L}$ and $\mathbf{U}$ as $\mathbf{A} = \mathbf{LU}$, where

$$\mathbf{L} = \begin{bmatrix} 1 & & & & \\ e_1 & 1 & & & \\ & \ddots & \ddots & & \\ & & e_{n-2} & 1 & \\ & & & e_{n-1} & 1 \end{bmatrix}$$

$$\mathbf{U} = \begin{bmatrix} f_0 & c_0 & & & \\ & f_1 & c_1 & & \\ & & \ddots & \ddots & \\ & & & f_{n-2} & c_{n-2} \\ & & & & f_{n-1} \end{bmatrix}.$$

The LU algorithm to solve the linear system (1) then proceeds to solve for $\mathbf{y}$ from $\mathbf{Ly} = \mathbf{d}$ and then finds $\mathbf{x}$ by solving $\mathbf{Ux} = \mathbf{y}$. More precisely, this algorithm to solve (1) consists of the following steps:

The LU Algorithm

Step 1. Compute the LU decomposition of $\mathbf{A}$ given by

$$\begin{aligned} f_0 &= b_0, \\ e_i &= a_i/f_{i-1}, \ 1 \le i \le n-1, \\ f_i &= b_i - e_i * c_{i-1}, \ 1 \le i \le n-1. \end{aligned}$$

Step 2. Solve for $\mathbf{y}$ from $\mathbf{Ly} = \mathbf{d}$ using

$$\begin{aligned} y_0 &= d_0, \\ y_i &= d_i - e_i * y_{i-1}, \ 1 \le i \le n-1. \end{aligned}$$

Step 3. Compute $\mathbf{x}$ by solving $\mathbf{Ux} = \mathbf{y}$ using

$$\begin{aligned} x_{n-1} &= y_{n-1}/f_{n-1}, \\ x_i &= (y_i - c_i * x_{i+1})/f_i, \ 0 \le i \le n-2. \end{aligned}$$

First we consider the parallelization of the LU decomposition part of the LU Algorithm to solve (1), i.e. Step 1 above. Once the diagonal entries $f_1, \ldots, f_{n-1}$ of $\mathbf{U}$ have been calculated, $e_1, e_2, \ldots, e_{n-1}$ can subsequently be computed in a single parallel step with $n-1$ processors. Thus we concentrate on the computation of the f_i's.

Let $\alpha_i = b_{n-i}$ for $1 \le i \le n$, and $\beta_i = -a_{n-i+1} * c_{n-i}$ for $2 \le i \le n$. Then the f_i satisfy the nonlinear recursion

$$\begin{aligned} f_0 &= \alpha_n \\ f_i &= \alpha_{n-i} + \beta_{n-i+1}/f_{i-1}\ , \ 1 \le i \le n-1\ . \end{aligned}$$

Thus we have continued fraction (CF) expansions for the f_i of the form

$$\begin{aligned} f_1 &= \alpha_{n-1} + \frac{\beta_n}{\alpha_n}, \\ f_2 &= \alpha_{n-2} + \cfrac{\beta_{n-1}}{\alpha_{n-1} + \frac{\beta_n}{\alpha_n}}, \\ &\ \vdots \\ f_{n-1} &= \alpha_1 + \cfrac{\beta_2}{\alpha_2 + \cfrac{\beta_3}{\ddots \cfrac{\vdots}{\alpha_{n-1} + \frac{\beta_n}{\alpha_n}}}} \end{aligned} \qquad (3)$$

The resulting computational problem for the calculation of the f_i with this formulation is the efficient evaluation of the continued fractions (3) in parallel.

2.1 Continued Fractions

Because of the form of (3), we look at fast parallel evaluation of CF. Consider the nth convergent

$$\begin{aligned} \frac{p_n}{q_n} &= \alpha_1 + \cfrac{\beta_2}{\alpha_2 + \cfrac{\beta_3}{\ddots \cfrac{\vdots}{\alpha_{n-1} + \frac{\beta_n}{\alpha_n}}}} \\ &= \alpha_1 + \frac{\beta_2}{\alpha_2 +} \frac{\beta_3}{\alpha_3 +} \cdots \frac{\beta_n}{\alpha_n} \end{aligned} \qquad (4)$$

of a given arbitrary (i.e. not necessarily periodic) CF. By Milne-Thomson matrix formulation [1], we have

$$\begin{bmatrix} p_n & p_{n-1} \\ q_n & q_{n-1} \end{bmatrix} = \begin{bmatrix} \alpha_1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} \alpha_2 & 1 \\ \beta_2 & 0 \end{bmatrix} \cdots \begin{bmatrix} \alpha_n & 1 \\ \beta_n & 0 \end{bmatrix}$$

or

$$\begin{bmatrix} p_n \\ q_n \end{bmatrix} = \begin{bmatrix} \alpha_1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} \alpha_2 & 1 \\ \beta_2 & 0 \end{bmatrix} \cdots \begin{bmatrix} \alpha_n & 1 \\ \beta_n & 0 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

The above representation was used in [2] for the fast computation of the first n convergents of a CF. The algorithm proceeds as follows. Define

$$A_k = \begin{bmatrix} \alpha_k & 1 \\ \beta_k & 0 \end{bmatrix} \ \text{for} \quad 1 \le k \le n\ ,$$

with $\beta_1 = 1$. We can then write the above product concisely as

$$\begin{bmatrix} p_n \\ q_n \end{bmatrix} = A_1 A_2 A_3 \cdots A_n \cdot \begin{bmatrix} 1 \\ 0 \end{bmatrix}\ .$$

Therefore all convergents $\frac{p_k}{q_k}$ for $k = 1, 2, 3 \ldots, n$ of a general CF can be computed in $O(\log n)$ parallel time using $O(\frac{n}{\log n})$ processors, as we only need the prefix products

$$\begin{array}{l} A_1 \\ A_1 A_2 \\ \vdots \\ A_1 A_2 \cdots A_n \end{array} \tag{5}$$

of the 2×2 matrices A_k for $1 \le k \le n$ and a parallel prefix algorithm [10], [11], [12] can be used for the computation of the products in (5). Details of this algorithm can be found in [2].

However the f_i's in (3) are not the convergents of a single CF. In fact, direct application of this fast parallel evaluation of CF's to compute each one of the finite continued fractions for $f_1, f_2, \ldots, f_{n-1}$ in (3) independently would require

$$O(\sum_{k=2}^{n-1} \frac{k}{\log k}) = O(\int_2^n \frac{x}{\log x} dx) = O(\frac{n^2}{\log n})$$

processors an take

$$O(\sum_{k=2}^{n-1} \log k) = O(\log^2 n)$$

parallel time. Thus the prefixes in (5) compute the given CF in the "wrong direction" as far as the f_i are concerned.

2.2 A Suffix Product Formulation

We can show that the *suffix* products

$$\begin{array}{r} A_n \\ A_{n-1} A_n \\ \vdots \\ A_2 A_3 \cdots A_n \end{array} \tag{6}$$

can be used to compute all of the f_i in parallel efficiently. To see this, write each f_i in (3) in the form

$$f_i = \alpha_{n-i} + \frac{r_i}{s_i} , \ \ 1 \le i \le n-1 \ . \tag{7}$$

Consider again the finite CF

$$\alpha_1 + \frac{\beta_2}{\alpha_2 +} \frac{\beta_3}{\alpha_3 +} \cdots \frac{\beta_n}{\alpha_n} \tag{8}$$

defined in (4). Then

$$\frac{r_i}{s_i} = \frac{\beta_{n-i+1}}{\alpha_{n-i+1} +} \frac{\beta_{n-i+2}}{\alpha_{n-i+2}} \cdots \frac{\beta_n}{\alpha_n}$$

for $1 \le i \le n-1$. By (7), once the values of r_i and s_i for $1 \le i \le n-1$ are known, $f_1, f_2, \ldots, f_{n-1}$ can be computed with 2 parallel arithmetic operations using $n-1$ processors.

For the computation of r_i and s_i, we claim that

$$\begin{aligned} \begin{bmatrix} s_i \\ r_i \end{bmatrix} &= \begin{bmatrix} \alpha_{n-i+1} & 1 \\ \beta_{n-i+1} & 0 \end{bmatrix} \begin{bmatrix} \alpha_{n-i+2} & 1 \\ \beta_{n-i+2} & 0 \end{bmatrix} \cdots \\ & \qquad \cdots \begin{bmatrix} \alpha_n & 1 \\ \beta_n & 0 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 0 \end{bmatrix} \\ &= A_{n-i+1} A_{n-i+2} \ldots A_n \cdot \begin{bmatrix} 1 \\ 0 \end{bmatrix} \end{aligned} \tag{9}$$

for $1 \le i \le n-1$. The assertion in (9) can easily be proved by induction on i by using the fact that

$$\frac{\beta_{i-1}}{\alpha_{i-1} + \frac{r_i}{s_i}} = \frac{\beta_{i-1} * s_i}{r_i + \alpha_{i-1} * s_i} \ .$$

The following example illustrates this calculation.

Example
Suppose that the finite CF (8) in question is

$$1 + \frac{2}{3 + \frac{4}{5 + \frac{6}{7}}} \ .$$

Then $\alpha_1 = 1, \alpha_2 = 3, \alpha_3 = 5, \alpha_4 = 7$ and $\beta_2 = 2, \beta_3 = 4, \beta_4 = 6$. From the expressions in (3) we compute that

$$\begin{aligned} f_1 &= 5 + \frac{6}{f_0} = 5 + \frac{6}{7} = \frac{41}{7} \\ f_2 &= 3 + \frac{4}{f_1} = 5 + \frac{28}{41} = \frac{151}{41} \\ f_3 &= 1 + \frac{2}{f_2} = 1 + \frac{82}{151} = \frac{233}{151} \ . \end{aligned} \tag{10}$$

We have

$$A_2 = \begin{bmatrix} 3 & 1 \\ 2 & 0 \end{bmatrix}, \ A_3 = \begin{bmatrix} 5 & 1 \\ 4 & 0 \end{bmatrix}, \ A_4 = \begin{bmatrix} 7 & 1 \\ 6 & 0 \end{bmatrix} .$$

Thus

$$\begin{aligned} A_4 &= \begin{bmatrix} 7 & 1 \\ 6 & 0 \end{bmatrix} \\ A_3 A_4 &= \begin{bmatrix} 41 & 5 \\ 28 & 4 \end{bmatrix} \\ A_2 A_3 A_4 &= \begin{bmatrix} 151 & 19 \\ 82 & 10 \end{bmatrix} \end{aligned} \tag{11}$$

From the first columns of the matrices in (12) we obtain

$$\begin{bmatrix} s_1 \\ r_1 \end{bmatrix} = \begin{bmatrix} 7 \\ 6 \end{bmatrix}, \begin{bmatrix} s_2 \\ r_2 \end{bmatrix} = \begin{bmatrix} 41 \\ 28 \end{bmatrix}, \begin{bmatrix} s_3 \\ r_3 \end{bmatrix} = \begin{bmatrix} 151 \\ 82 \end{bmatrix},$$

which is in agreement with the fractions $6/7$, $28/41$, and $82/151$ that appear in (11).

3 Parallel CF-LU Algorithm

We summarize the steps involved in the parallel computation of the LU decomposition of $\mathbf{A}$ as given in (2) by means of CF computations:

Parallel CF–LU Decomposition Algorithm

Input: A tridiagonal matrix $\mathbf{A}$ of order n as in (2).

Output: The matrices $\mathbf{L}$ and $\mathbf{U}$ of the LU decomposition of $\mathbf{A}$.

Step I. Let $f_0 := b_0$ and $\alpha_i := b_{n-i}$ for $1 \leq i \leq n$. For $2 \leq i \leq n$ compute

$$\beta_i = -a_{n-i+1} * c_{n-i} \ .$$

Step II. For $2 \leq i \leq n$ let

$$A_i := \left[\begin{array}{cc} \alpha_i & 1 \\ \beta_i & 0 \end{array} \right]$$

and compute the suffix products $A_i A_{i+1} \cdots A_n$.

Step III. For $1 \leq i \leq n-1$ compute

$$f_i := \alpha_{n-i} + \frac{r_i}{s_i}$$

where $[s_{n-i+1}, r_{n-i+1}]^T$ is the first column of $A_i A_{i+1} \cdots A_n$ computed in Step II.

Step IV. For $1 \leq i \leq n-1$ compute

$$e_i := \frac{a_i}{f_{i-1}} \ .$$

Theorem 1 *Parallel CF–LU decomposition algorithm computes the LU factorization of an $n \times n$ tridiagonal matrix in $O(\log n)$ parallel arithmetic operations using $O(\frac{n}{\log n})$ processors.*

Proof Each one of the Steps I, III, and IV can be computed with either $n-1$ processors in $O(1)$ time, or with $O(\frac{n}{\log n})$ processors in $O(\log n)$ time. All of the suffixes of the matrix products in Step III can be computed by an appropriately modified parallel prefix algorithm in $O(\log n)$ time using $O(\frac{n}{\log n})$ processors. Therefore total time required by the parallel CF–LU algorithm is $O(\log n)$ using $O(\frac{n}{\log n})$ processors. □

4 Conclusions

We considered the parallelization of the LU decomposition part of the LU Algorithm for an $n \times n$ tridiagonal linear system. Optimal parallel algorithms for this problem through matrix recurrences are already known. However, LU decomposition is related to parallel evaluation of continued fractions in an interesting way. In the matrix-chain product formulation of the recurrences, *prefix* products solve one problem, while *suffix* products solve the other. Both products can in turn can be computed fast by known parallel prefix methods. The suffix product formulation for continued fractions together with parallel prefix gives an optimal LU decomposition algorithm which runs in $O(\log n)$ time using $O(\frac{n}{\log n})$ processors.

References

[1] L. M. Milne-Thomson, *The Calculus of Finite Differences* (2nd Unaltered Edition, Chelsea Publishing Company, New York, 1981).

[2] Ö. Eğecioğlu, Ç. K. Koç, J. Rifà Coma, Fast Computation of Continued Fractions, *Computers Math. Applic.*, 21(2–3), 1991, 167–169.

[3] H. S. Stone, An efficient parallel algorithm for the solution of tridiagonal system of equations, *J. Assoc. Comput. Mach.*, 20(1), 1973, 27–38.

[4] J. Ortega and R. Voigt, Partial differential equations on vector and parallel computers, *SIAM Rev.*, 1985, 149–240.

[5] H. S. Stone, Parallel tridiagonal equation solvers, *ACM Trans. Math. Software*, 1(4), 1975, 289–307.

[6] S. L. Johnsson, Solving tridiagonal systems on ensemble architectures, *SIAM J. Sci. Statist. Comput.*, 8 (3), 1987, 354–392.

[7] D. Heller, A survey of parallel algorithms in numerical linear algebra, *SIAM Rev.*, 1978, 740–777.

[8] Ö. Eğecioğlu, Ç. K. Koç, A. Laub, A recursive doubling algorithm for solution of tridiagonal systems on hypercube multiprocessors, *Journal of Computational and Applied Mathematics*, 27, 1989, 95–108.

[9] P. M. Kogge, Parallel Solution of Recurrence Problems, *IBM J. Res. Develop.*, 18(2), 1974, 138–148.

[10] C. P. Kruskal, L. Rudolph, and M. Snir, The Power of Parallel Prefix, *IEEE Transactions on Computers*, C–34(10), 1985, 965–968.

[11] R. Ladner and M. Fischer, Parallel Prefix Computation, *Journal of ACM*, 27(4), 1980, 831–838.

[12] M. Snir, Depth-Size Trade-offs for Parallel Prefix Computation, *Journal of Algorithms*, 7(2), 1986, 185–201.