Exact Solution of Linear Equations on Distributed-Memory Multiprocessors

Ç. K. Koç, A. Güvenç, and B. Bakkaloğlu
Department of Electrical & Computer Engineering
Oregon State University
Corvallis, Oregon 97331

Abstract

We present two new parallel algorithms for exact (error-free) solution of a system of linear equations on a distributed-memory multiprocessor. The exact solution is obtained using the congruence technique which consists of two steps: First, the system of linear equations is converted to systems of linear congruence equations with respect to several prime moduli, and each of these systems is solved on a separate processor. Then, these solutions are combined using the mixed-radix conversion algorithm to obtain the exact solution. The first step is completely parallel with no communication requirements among the processors. We improve our previous work and describe two efficient parallel algorithms for the second step, and present the results of our experiments on an Intel iPSC/860.

1 Introduction

In scientific computing [9] and in digital signal processing [3], often one needs to solve a system of equations with integer or rational number matrices. Roundoff errors may make it impossible to solve an ill-conditioned problem, or it may cause a real-time digital signal processing system to be unstable. The direct method, i.e., Gaussian elimination with multiple precision arithmetic, may not be suitable for such problems, since the intermediate results may grow excessively, even though the initial values as well as the final result have manageable size. The most effective methods are the congruence and the p-adic expansion techniques which obtain the exact solution without causing excessive growth of the intermediate results. The basic mathematics of congruence technique and p-adic arithmetic are both well-known [4, 2, 9]. Efficient sequential algorithms and software for solving linear equations using the residue and the p-adic techniques have been developed in the last twenty years. Parallel algorithms for exact solution of linear equations using the congruence and the p-adic techniques have also been designed and implemented [7, 5, 8]. In this paper, we present two new parallel algorithms for finding the exact solution of an integer system of linear equations using the congruence technique. We consider the solution of the system of linear equations

$$\mathbf{A}\mathbf{x} = \mathbf{b} \ , \tag{1}$$

where $\mathbf{A}$ is a $k \times k$ invertible matrix and $\mathbf{b}$ is a k vector with all integer entries. Since the solution vector $\mathbf{x}$ will in general have rational number entries, we fisrt solve the linear system

$$\mathbf{A}\mathbf{z} = d\,\mathbf{b} \ , \tag{2}$$

where $d = \det(\mathbf{A})$. In this case, the entries of the solution vector $\mathbf{z}$ will be integers, and the final solution vector $\mathbf{x}$ is obtained using floating-point arithmetic:

$$\mathbf{x} = \frac{1}{d}\,\mathbf{z} \ . \tag{3}$$

2 The Congruence Technique

In order to solve for Equation (2), we pick n prime numbers $m_1, m_2, \ldots, m_n$ such that their product M is larger than the largest entry of the matrix $\mathbf{A}$. The congruence method consists of two main steps:

1. Solve the n systems $\mathbf{A}\mathbf{y}_i = \mathbf{b} \pmod{m_i}$ using Gaussian elimination and also compute the determinant $d_i = \det(\mathbf{A}) \pmod{m_i}$ for $i = 1, 2, \ldots, n$. The solution of Equation (2) modulo m_i is obtained as $\mathbf{z}_i = d_i \mathbf{y}_i$ for $i = 1, 2, \ldots, n$.

2. Use the mixed-radix conversion algorithm to combine the solution vectors $\mathbf{z}_i$ into a single vector $\mathbf{z}$ such that $\mathbf{z} = \mathbf{z}_i \pmod{m_i}$ for $i = 1, 2, \ldots, n$. The solution of Equation (1) is then obtained as $\mathbf{x} = \frac{1}{d}\,\mathbf{z}$.

Step 1 is completely parallel since the solution of equation $\mathbf{A}\mathbf{y}_i = \mathbf{b} \pmod{m_i}$ is independent for every $i = 1, 2, \ldots, n$. Assuming the number of processors p is equal to n, we allocate processor i for modulo m_i computations: Processor i solves the system $\mathbf{A}\mathbf{y}_i = \mathbf{b} \pmod{m_i}$ and computes the determinant $d_i = \det(\mathbf{A}) \pmod{m_i}$, and then proceeds to compute $\mathbf{z}_i = d_i\,\mathbf{y}_i \pmod{m_i}$. This

computation is simultaneously performed by all processors for $i = 1, 2, \ldots n$. In general, given $p \leq n$ processors, we partition the moduli set in such a way that each processor receives at most q moduli where $q = \lceil n/p \rceil$. The above computations are then repeated q times. Since Gaussian elimination on a matrix of dimension k requires $O(k^3)$ arithmetic steps, we conclude that Step 1 of the congruence method requires $O(k^3 n/p)$ arithmetic steps with $p \leq n$ processors.

At the end of Step 1, we will have a k vector $\mathbf{z}_i$ and an integer d_i in processor i for all $i = 1, 2, \ldots, n$. We now need to apply the mixed-radix conversion algorithm to compute a k vector $\mathbf{z}$ and an integer d. Let $\mathbf{u}_i = [\mathbf{z}_i \ , \ d_i]^T$ be a $(k+1)$ vector. We use the mixed-radix conversion algorithm to compute the $(k+1)$ vector $\mathbf{u}$ such that $\mathbf{u} = \mathbf{u}_i \pmod{m_i}$. The mixed-radix conversion algorithm returns the vector $\mathbf{u}$, from which we extract the value of the determinant d and the elements of the vector $\mathbf{z}$.

```
The Sequential MRC Algorithm
for j = 1 to n − 1 do
    for i = j + 1 to n do
        u_i := (u_i − u_j)c_ij   (mod m_j)
```

Here c_{ij} are the multiplicative inverses of m_i modulo m_j for $1 \leq i < j \leq n$, computed using the extended Euclidean algorithm [4]. After the above process, the elements of the vectors $\mathbf{u}_i$ are the mixed-radix coefficients of the elements of the final vector $\mathbf{u}$. Thus, the final vector $\mathbf{u}$ is obtained by computing

$$\begin{aligned} \mathbf{u} \ = \ & \mathbf{u}_1 + m_1\mathbf{u}_2 + m_1m_2\mathbf{u}_3 + m_1m_2m_3\mathbf{u}_4 + \cdots \\ & + m_1m_2\cdots m_{n-1}\mathbf{u}_n \ . \end{aligned} \tag{4}$$

Since the mixed-radix conversion of n integers requires $O(n^2)$ arithmetic operations [4], we calculate the number of arithmetic operations required by the sequential MRC algorithm as $O(kn^2)$.

The data dependences among the computations required to obtain the final vector $\mathbf{u}$ lend themselves to systolic implementation. The parallel algorithms given in [7, 5] are derived from systolic schedules. Although these algorithms were implemented on an Intel iPSC1 hypercube, they require only linear or ring connectivity among the processors, and therefore, do not fully exploit the hypercube connectivity. In the following, we describe two new parallel algorithms which are particularly suitable for implementation on distributed-memory architectures.

3 Parallel Algorithms

We will describe two parallel algorithms which are termed as the *Single-Node Broadcast* (SNB) algorithm and the *Multi-Node Broadcast* (MNB) algorithm. Step 1 of the congruence method is parallelized exactly the same way for both of these algorithms. These algorithms differ only in the way the computational and communication requirements of Step 2 of the congruence method are handled.

3.1 Single-Node Broadcast

At the end of Step 1, we have the vector $\mathbf{u}_i$ in processor i for all $i = 1, 2, \ldots, n$. The SNB algorithm is a direct parallel implementation of the sequential MRC algorithm, and goes through $n-1$ steps for $j = 1, 2, \ldots, n-1$ where at step j processor j broadcasts its vector $\mathbf{u}_j$ to all processors whose index is larger than j. These processors update their $\mathbf{u}$ vectors with the vector $\mathbf{u}_j$ received. When the update process is completed, the last processor contains a copy of each of the $\mathbf{u}_i$ vectors for $i = 1, 2, \ldots, n$. This processor then uses Equation (4) and Equation (3) in floating-point arithmetic to compute the vector $\mathbf{x}$. The pseudocode below describes these operations:

```
                    The SNB Algorithm
                    for j = 1 to n − 1 do
                      begin
        PROC j:       send u_j to PROC j + 1, ..., n
        PROC i:       u_i := (u_i − u_j)c_ij   (mod m_i)
(i = j + 1, .., n)    end
        PROC n:     compute x using Eqs. (4) and (3)
```

This way we parallelize the i-loop of the sequential MRC algorithm. The SNB algorithm requires $O(kn)$ arithmetic steps with n processors.

In order to calculate the communication penalty, we notice that at each step a single-node broadcast of a vector of dimension $k+1$ is performed. It is well-known that a single-node broadcast operation requires $\log_2 p$ routing steps on a hypercube architecture with p processors [1]. The algorithm is based on the spanning tree of the hypercube graph, rooted at the node which broadcats its data. At each step a single-node broadcast of a vector of dimension $k+1$ is performed, however, the number of processors, to which the data is sent, decreases by one. Thus, we obtain the communication penalty of the SNB algorithm as

$$(k+1) \sum_{i=n-1}^{1} \lceil \log_2 i \rceil = O(kn \log n) \ .$$

3.2 Multi-Node Broadcast

The SNB algorithm interleaves the computational and communication steps of the mixed-radix conversion algorithm. The MNB algorithm first performs all necessary communications; it then proceeds to compute the elements of the $\mathbf{u}$ vector using the sequential MRC algorithm with the locally available data. In the beginning of the multi-node broadcast we have $\mathbf{u}_i$ in processor i for $i = 1, 2, \ldots, n$. At the end, all processors have all of the vectors $\mathbf{u}_i$. If $k = n = p$, then processor i picks the ith

and the last element (corresponding to the determinant) of the vectors $\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_n$, and then performs two *sequential* scalar mixed-radix conversion operations in order to obtain the mixed-radix coefficients of the element $\mathbf{z}[i]$ and the determinant d, respectively. It then computes $\mathbf{x}[i]$ using Equations (4) and (3) in floating-point arithmetic.

	The MNB Algorithm
	`for` $j = 1$ `to` n `do`
PROC j:	`send` $\mathbf{u}_j$ `to` PROC $1, \ldots, n$ `except` j
PROC i:	`compute` $\mathbf{z}[i]$ using sequential MRC
($i = 1, .., n$)	`compute` d using sequential MRC
	`compute` $\mathbf{x}[i]$ using Eqs. (4) and (3)

If $k \geq n = p$, then processor i performs $\lceil k/n \rceil$ mixed-radix conversions and obtains as many elements of the vector $\mathbf{z}$. Thus, the number of mixed-radix conversions to be performed by processor i is equal to $\lceil k/n \rceil + 1$, which gives the total number of arithmetic operations as $O(n^2 k/n) = O(kn)$. Therefore, the MNB algorithm requires $O(kn)$ arithmetic steps with n processors.

Furthermore, the MNB algorithm needs to perform n broadcast operations in order to distribute the data to all processors. A naive method of accomplishing this task is to perform n sequentially-arranged single-node broadcast operations. A better strategy is to perform simultaneous broadcast operations in order to achieve maximum concurrency. Details of the multi-node broadcast operation on a hypercube computer can be found in [1, 6]. The algorithm has exactly $d = \log_2 p$ steps and the amount of the data exchanged is doubled at each step. The total number of routing operations required to broadcast all vectors $\mathbf{u}_i$ is found as

$$(k+1) \sum_{k=1}^{d} 2^k = O(kn) \ .$$

4 Implementation and Results

We have implemented the above algorithms on an Intel iPSC/860 multiprocessor with 8 processors. In our experiments, we solve integer linear systems of equations with dimensions $k = 8, 16, 32, 64$, and 128. The number of prime moduli takes the values $n = 8, 16, 32, 64$, and 128. The prime numbers are selected such that the basic arithmetic operations $\{+, -, \times\}$ can be performed in single-precision integer arithmetic. Since the largest single-precision signed integer is equal to $2^{31} - 1$, we choose $m_i < 2^{15}$ in order to perform a modular multiplication operation. When the number of primes is equal to 128, the entries of the matrix $\mathbf{A}$ and the vector $\mathbf{b}$ can be as large as $2^{15 \times 128} = 2^{1920} \approx 10^{577}$.

The detailed timing values for the sequential and parallel algorithms are given in the full version of the paper [6]. As an example, a linear system of dimension 128 with integer entries as large as 10^{577} is solved in approximately 194.6 seconds by the MNB algorithm, while it is solved in 1,553.2 seconds by the sequential algorithm. This represents a speedup of 7.9815, or an efficiency of 99.76 %. In Table 1, we list the efficiency values as a function of n by fixing the system size at $k = 128$, and as a function of k by fixing n at 128.

Table 1. Efficiency of the parallel algorithms.

	$k = 128$			$n = 128$	
n	SNB	MNB	k	SNB	MNB
8	0.9454	0.9787	8	0.0473	0.4638
16	0.9459	0.9892	16	0.1134	0.7062
32	0.9475	0.9944	32	0.3298	0.8975
64	0.9471	0.9968	64	0.7231	0.9816
128	0.9371	0.9976	128	0.9371	0.9976

References

[1] D. P. Bertsekas and J. N. Tsitsiklis. *Parallel and Distributed Computation, Numerical Methods.* Englewood Cliffs, NJ: Prentice-Hall, 1989.

[2] R. T. Gregory and E. V. Krishnamurthy. *Methods and Applications of Error-Free Computation.* New York, NY: Springer-Verlag, 1984.

[3] B. Harms and S. Keller-McNulty. Error-free solution to a Toeplitz system of equations. *IEEE Transactions on Signal Processing*, 39(5):1212–1215, May 1991.

[4] D. E. Knuth. *The Art of Computer Programming: Seminumerical Algorithms*, volume 2. Reading, MA: Addison-Wesley, Second edition, 1981.

[5] Ç. K. Koç. A parallel algorithm for exact solution of linear equations via congruence technique. *Computers and Mathematics with Applications*, 23(12):13–24, 1992.

[6] Ç. K. Koç, A. Güvenç, and B. Bakkaloğlu. Exact solution of linear equations on distributed-memory multiprocessors. *Parallel Algorithms and Applications*, 1994. To appear.

[7] Ç. K. Koç and R. M. Piedra. A parallel algorithm for exact solution of linear equations. In *Proceedings of International Conference on Parallel Processing*, volume III, pages 1–8, St. Charles, Illinois, August 12–16 1991. Boca Raton, FL: CRC Press.

[8] G. Villard. Exact parallel solution of linear systems. In J. Della Dora and J. Fitch, editors, *Computer Algebra and Parallelism*, pages 197–205. New York, NY: Academic Press, 1989.

[9] D. M. Young and R. T. Gregory. *A Survey of Numerical Mathematics*, volume 2. New York, NY: Dover Publications, 1988.

14TH WORLD CONGRESS
at Georgia Tech

on Computational and Applied Mathematics

July 11-15, 1994

School of Mathematics
Georgia Institute of Technology
Atlanta, Georgia
USA

Proceedings in 3 Volumes
Volume 3