Parallel p-adic Method for Solving Linear Systems of Equations *

Ç. K. Koç
Electrical & Computer Engineering
Oregon State University
Corvallis, Oregon 97331

Abstract

We present a parallel algorithm for exact solution of an integer linear system of equations using the single modulus p-adic expansion technique. More specifically, we parallelize an algorithm of Dixon, and present our implementation results on a distributed-memory multiprocessor. The parallel algorithm presented here can be used together with the multiple moduli algorithms and parallel Chinese remainder algorithms for fast computation of the exact solution of a system of linear equations with integer entries.

1 Introduction

Exact solutions of a system of linear equations with integer or rational number entries can be found by using Gaussian elimination and multiple-precision arithmetic. However, this direct method breaks down even for systems of moderate size due to excessive growth of the intermediate results, even though the initial values as well as the final result are in single-precision. The congruence and the p-adic expansion techniques are preferred to the direct method, since the sizes of intermediate results are kept under control. Although some algorithmic questions remain, the basic mathematics of residue number and p-adic arithmetic are both well-known [9, 7, 15]. Efficient sequential algorithms and software for solving linear equations using the residue [13, 8, 4, 2] and the p-adic [3] techniques have been developed in the last twenty years. Recently, parallel algorithms for exact solution of linear systems using the multiple moduli congruence [12, 10, 11] and the p-adic techniques [14] have also been designed and implemented. The multiple moduli congruence and p-adic expansion algorithms are amenable for parallelism, since computations for each modulus can be separately carried out. This step is completely parallel and no communication is required among the processors. The solutions for each of the moduli are then combined using the single-radix or the mixed-radix conversion algorithms, both of which also exhibit a certain degree of parallelism. A brief comparison of direct methods utilizing the multi-precision arithmetic versus the p-adic methods is given in [14], where the execution times with respect to the upper bound on the matrix entries are also analyzed.

In this paper we concentrate on the single modulus case, and describe a parallel algorithm for obtaining the exact rational solution of a linear system of equations using the single modulus p-adic method. More specifically, we parallelize an algorithm of Dixon [3], and present our implementation results on a distributed-memory multiprocessor. The parallel algorithm described in this paper can be used together with the parallel multiple moduli algorithms given in [12, 10, 11] for fast computation of the exact rational solution of linear equations with integer entries.

*This work is supported in part by the National Science Foundation under grant ECS–9312240.

Parallel Computing, 23(13):2067-2074, December 15, 1997.

2 Dixon's Algorithm

We consider computing the exact rational solution to the linear system of equations:

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

where $\mathbf{A} \in \mathcal{I}^{n\times n}$, $\mathbf{b} \in \mathcal{I}^{n\times 1}$, and the solution $\mathbf{x} \in \mathcal{Q}^{n\times 1}$. Given the integer matrix $\mathbf{A}$ which is nonsingular for the prime modulus p, and the integer vector $\mathbf{b}$, the following algorithm by Dixon [3] computes the exact rational solution $\mathbf{x}$. Dixon's algorithm consists of three fundamental steps:

1. Inversion Step: We compute the inverse of the matrix $\mathbf{A}$ modulo p.
2. Iteration Step: The solution $\overline{\mathbf{x}}$ of $\mathbf{A}\overline{\mathbf{x}} = \mathbf{b} \bmod p^m$ is obtained by iteration.
3. Euclidean Step: The rational solution of (1) is obtained using the extended Euclid algorithm.

These steps are shown in more detail below:

```
          DIXON'S ALGORITHM (A, b, p, m)
Step 1:   C = A^{-1} mod p                          ; Inversion
Step 2:   b_0 = b                                   ; Iteration
          for i = 1 to m
            x_i = C b_i mod p
            b_{i+1} = p^{-1}(b_i - A x_i)
          end
          x̄ = Σ_{i=0}^{m-1} x_i p^i
Step 3:   for j = 1 to n                            ; Euclidean
            u_{-1}(j) = p^m, u_0(j) = x̄(j)
            v_{-1}(j) = 0, v_0(j) = 1
            while u_i(j) < p^{m/2}
              q_i(j) = ⌊u_{i-1}(j)/u_i(j)⌋
              u_{i+1}(j) = u_{i-1}(j) - q_i(j)u_i(j)
              v_{i+1}(j) = v_{i-1}(j) + q_i(j)v_i(j)
            end
          end
          x(j) = ((-1)^i u_i(j)/v_i(j)), 1 ≤ j ≤ n  ; Rational solution
```

Dixon's algorithm requires $O(n^3 \log^2 n)$ arithmetic operations when the integer entries of $\mathbf{A}$ and $\mathbf{b}$ lie within the range $[-W, W]$. Denoting the Euclidean norm of the ith column of $\mathbf{A}$ as $\mu_i = \|\mathbf{A}_i\|_2$, where $\mathbf{A}_i = [a_{1i}\ a_{2i}\ \cdots\ a_{ni}]^T$ and $\mu_0 = \|\mathbf{b}\|_2$, we use the Hadamard determinant inequality to see that $|\det \mathbf{A}| \leq \prod_{i=1}^{n} \mu_i$. We can compute an absolute bound β for the numerator and denominator of the rational entries of the exact solution $\mathbf{x}$ from Cramer's rule as follows:

$$\beta = \frac{\prod_{i=0}^{n} \mu_i}{\min(\mu_i)} \leq (n^{1/2}W)^n \ . \tag{2}$$

The first step of the algorithm is to compute the modulo p inverse of $\mathbf{A}$, i.e., to compute $\mathbf{C} \in \mathcal{I}_p^{n\times n}$ such that $\mathbf{AC} = \mathbf{I} \bmod p$, where $\mathcal{I}_p^{n\times n}$ denotes an $n \times n$ matrix with entries from the residue class of integers modulo p. Using Gaussian elimination or LU decomposition, $\mathbf{C}$ can be calculated in $O(n^3)$ modular arithmetic operations. The second step which computes the p-adic approximation to $\mathbf{x}$ involves

Parallel Computing, 23(13):2067-2074, December 15, 1997.

a modular and an integer matrix-vector multiplication. The elements of $\mathbf{b}_i$ lie in the range $[-nW, nW]$, hence they are of length $\log n + \log W = O(\log n)$. Therefore matrix-vector products can be computed in $O(n^2 \log n)$ operations. This iteration is repeated m times, giving a total operation count of $O(mn^2 \log n)$. Computing $\overline{\mathbf{x}}$ guarantees

$$\mathbf{A}\overline{\mathbf{x}} = \mathbf{b} \bmod p^m \ . \tag{3}$$

This is easily proven by noting that

$$\overline{\mathbf{x}} = \sum_{i=0}^{m-1} \mathbf{x}_i p^i \ , \tag{4}$$

thus, we have

$$A\overline{\mathbf{x}} = \sum_{i=0}^{m-1} p^i \mathbf{A}\mathbf{x}_i = \sum_{i=0}^{m-1} p^i(\mathbf{b}_i - p\mathbf{b}_{i+1}) = \mathbf{b}_0 - p^m\mathbf{b}_m = \mathbf{b} - p^m\mathbf{b}_m \ . \tag{5}$$

Taking mod p^m of both sides, we obtain Equation (3). Dixon also proposes a method of computing the unknown fraction f/g, with $|f|,\ |g| \leq \beta$ such that $gs = f \bmod h$ ($s = \overline{\mathbf{x}}$ and $h = p^m$ in our case). The upper bound for f and g is found to be $\lambda h^{1/2}$, where $\lambda = (\sqrt{5}-1)/2$ is a root of $\lambda^2 + \lambda - 1 = 0$. Thus, we have $\beta \leq \lambda p^{m/2}$. Taking the logarithm of both sides, we obtain the minimum m as

$$m = \frac{2\log(\beta/\lambda)}{\log(p)} \ . \tag{6}$$

Substituting $\beta \leq (n^{1/2}W)^n$, we obtain

$$m = \frac{2}{\log(p)}\left(n\log(W) + \frac{n}{2}\log(n) - \log(\lambda)\right). \tag{7}$$

Since W is presumed to be a constant, we have $m = O(n \log n)$. Hence the operation count for the iteration becomes $O(n^3 \log^2 n)$. The final step of the algorithm converts the p-adic approximation $\overline{\mathbf{x}}$ to the rational exact solution $\mathbf{x}$. We can estimate the number of operations required to convert the p-adic number to its rational form. We take the logarithm of the inequality

$$h^{1/2} \geq \mathbf{v}_i(j) \geq \prod_{k=0}^{i-1} q_k(j) \geq 2^{i/2} \ , \tag{8}$$

and conclude that both $\sum_{k=0}^{i-1} \log q_k(j)$ and i are $O(\log h)$, i.e., $O(m \log p)$. Since the integers $\mathbf{u}_i$ and $\mathbf{v}_i$ are bounded by h and they are of length $O(m \log p)$, the computation of $\mathbf{u}_i/\mathbf{v}_i$ takes at most $O(m^2 \log^2 p)$ operations.

3 Parallelization of Dixon's Algorithm

In this section we assume that we have a distributed-memory multiprocessor with P identical processors. The single modulus parallel p-adic algorithm consists of three steps:

1. Distribute $\mathbf{A}$ to all the processors, and compute $\mathbf{C} = \mathbf{A}^{-1} \bmod p$ in parallel.
2. Distribute $\mathbf{C}$ to all the processors for the iteration, and perform all matrix-vector multiplications in parallel. Obtain the p-adic solution $\overline{\mathbf{x}}$ in parallel.

Parallel Computing, 23(13):2067-2074, December 15, 1997.

3. Distribute the solution vector $\overline{\mathbf{x}}$ among processors to compute the rational expression from the p-adic approximation using the extended Euclidean algorithm.

As we will show later, although the calculation of $\mathbf{C} = \mathbf{A}^{-1} \bmod p$ is done only once, solving it in parallel greatly reduces the total computation time. In order to reduce the communication penalty and to work on a consistent data arrangement $\mathbf{A}$ is copied to all the processors. Each processor works on the corresponding columns of $\mathbf{A}$ during parallel LU decomposition. After $\mathbf{C}$ is computed it is distributed to all the processors in a similar manner to reduce the communication penalty during the iteration stage.

We use row-pivoting for the elimination phase where each processor computes its portion of $\mathbf{C}$. After Gaussian elimination is completed each processor updates $\mathbf{C}$ by a multi-node broadcast. Gaussian elimination in a finite field requires pivot exchange only for the case where the pivoting element is zero modulo p. Hence no additional serial search is done for the element with maximum modulus, which is one of the major shortcomings of column-wrapped row-pivot Gaussian elimination [5].

With $P < n$ processors, each processor performs Gaussian elimination and back-substitution on a $n/P \times n$ rectangular array, which requires $O(n^3/P)$ arithmetic steps [6]. At each pivoting step, the pivot element is checked for singularity, and exchanged with the first nonzero element in the same column. During the elimination phase, the vector of multipliers l and u of length $n-1$ is broadcast to the rest of the processors. If there is a pivot exchange, the new pivot index is also included in the vector of multipliers. Thus, we find the communication penalty for the elimination step as $n(n-1)T_S = O(n^2 T_S)$, where T_S is the required time to send an integer from one processor to another in the architecture. For the hypercube connection, we have $1 \leq T_S \leq \log P$, i.e., $T_S = 1$ if the data is sent to a neighboring processor, and $T_S = \log P$ if the data is sent to the most distant processor [1]. This gives the maximum communication penalty as $O(n^2 \log P)$. After receiving the vector of multipliers l, u, and the pivot index, the rest of the processors update their columns in parallel.

Before the iteration step, $\mathbf{b}_0$ is broadcast to all the processors to compute the modular matrix-vector product $\mathbf{x}_i = \mathbf{C}\mathbf{b}_i \bmod p$. At this step each processor computes inner-products between the corresponding rows of $\mathbf{C}$ and $\mathbf{b}_i$ sequentially. This way each processor computes n/P elements of the column vector $\mathbf{x}_i$, and broadcasts it to other processors to update their vectors. For the second part of the iteration, each processor computes inner-products between the corresponding rows of $\mathbf{A}$ and updated $\mathbf{x}_i$, and subtracts from n/P elements of $\mathbf{b}_i$ to compute $\mathbf{b}_{i+1}$. In a similar fashion, each processor broadcasts n/P elements of $\mathbf{b}_{i+1}$ to update other processors' vectors, which brings a communication overhead of $mnT_B/P = O(n^2 \log n T_B/P)$, where T_B is the time required to broadcast an element from one processor to all the other processors. It is well-known that, in a hypercube architecture, a broadcast operation can be achieved in $T_B = \log P$ units of time [1]. This gives the communication overhead of this step as $O(n^2 \log n \log P/P)$. Computation of vector sequences $\mathbf{b}_{i+1}$ and $\mathbf{x}_i$ involves parallel matrix-vector multiplication which has an algebraic complexity of $O(n^3 \log^2 n/P)$.

After $\mathbf{x}_m$ and $\mathbf{b}_{m+1}$ is computed, the iteration terminates, and each processor computes n/P elements of $\overline{\mathbf{x}}$. Once $\overline{\mathbf{x}}$ is computed, the Euclidean algorithm works without any communication among the processors. Each processor converts n/P elements of $\overline{\mathbf{x}}$ to the rational form using $(n/P)(m^2 \log^2 p) = O(n^3 \log^2 n/P)$ operations. Thus, the parallel exact solution of a set of linear equations has an approximate time complexity of $O(n^3 \log^2 n/P)$.

4 Implementation Results

We have implemented the parallel p-adic algorithm on an 8-processor partition of a Meiko CS-2 multiprocessor, in which each node is a Sparc processor equipped with 256 MBytes of memory. The algorithm is implemented using the PVM software.

Parallel Computing, 23(13):2067-2074, December 15, 1997.

In our experiments, we solve integer linear systems of equations with various dimensions from 128 up to 1280. The prime p is selected so that the basic arithmetic operations involving modulo p can be performed in single-precision integer arithmetic, and computed $\overline{\mathbf{x}}$ satisfies $\mathbf{A}\overline{\mathbf{x}} = \mathbf{b} \bmod p^m$. Since p is in the order of W, its higher powers are not representable in finite precision. We used multi-precision exponentiation and addition to compute $\overline{\mathbf{x}}$. Two factors effect the efficiency of our algorithm: 1) the number of iteration steps m, which depends both on the system size and the absolute maximum of the system matrix entries. 2) the complexity of converting to the rational form from p-adic approximation, which depends on the magnitude of the numerator in the exact solution.

In Table 1, we tabulate the timings of the sequential and the parallel algorithm for $P = 2, 4, 8$ processors for certain values of n in the range $128 \leq n \leq 1280$. Furthermore, Table 2 shows the percentages of times spent during the steps of the Dixon algorithm. Table 2 clearly indicates that, even though the inversion of the matrix $\mathbf{A}$ is performed only once, it uses up to 96 % of the total time of algorithm. Thus, in order to obtain any considerable speedup, the parallel efficiency of the inversion step must be increased. Also we analyze the individual performance of the algorithm steps. Parallelizing the vector iteration gives an efficiency of 0.90 for 4 processors, whereas the parallel modular inversion has an efficiency of 0.60 for the same number of processors. The parallel efficiency of the iteration (Step 2) and the Euclidean (Step 3) steps are considerably higher, typically above 90 %. The overall parallel efficiency of the algorithm is shown in Table 3. This table indicates that the parallel algorithm has almost constant efficiency (linear speedup) for $n > 384$. However, as P increases, the efficiency decreases due to the high communication overhead of the inversion step. Any increase in the efficiency of the Gaussian elimination algorithm would have a direct positive effect on the efficiency of the algorithm presented.

References

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

[2] S. Cabay and T. P. L. Lam. Congruence techniques for the exact solution of integer systems of linear equations. *ACM Transactions on Mathematical Software*, 3(4):386–397, December 1977.

[3] J. D. Dixon. Exact solution of linear equations using p-adic expansions. *Numerische Mathematik*, 40(1):137–141, 1982.

[4] A. S. Fraenkel and D. Loewenthal. Exact solutions of linear equations with rational coefficients. *Journal of Research of the National Bureau of Standards*, 75B(1–2):67–75, January–June 1971.

[5] G. A. Geist and C. H. Romine. LU factorization algorithms on distributed-memory multiprocessor architectures. *SIAM Journal on Scientific and Statistical Computing*, 9(4):639–649, July 1988.

[6] G. H. Golub and C. F. van Loan. *Matrix Computations.* Baltimore, MD: The Johns Hopkins University Press, 2nd edition, 1989.

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

[8] J. A. Howell and R. T. Gregory. An algorithm for solving linear algebraic equations using residue arithmetic I–II. *BIT*, 9(3,4):200–224 and 324–337, 1969.

Parallel Computing, 23(13):2067-2074, December 15, 1997.

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

[10] 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.

[11] Ç. K. Koç, A. Güvenç, and B. Bakkaloğlu. Exact solution of linear equations on distributed-memory multiprocessors. *Parallel Algorithms and Applications*, 3:135–143, 1994.

[12] 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.

[13] M. Newman. Solving equations exactly. *Journal of Research of the National Bureau of Standards*, 71B(4):171–179, October–December 1967.

[14] 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.

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

Parallel Computing, 23(13):2067-2074, December 15, 1997.

Table 1: Timings for the sequential and parallel p-adic solution (in milliseconds).

n	$T(P) \to T(1)$	$T(2)$	$T(4)$	$T(8)$
128	23340	22442	12156	6630
256	189600	148125	84642	47400
384	608900	441231	249549	140949
512	1484800	1016986	580000	603200
640	2895360	1904842	1080358	1109109
768	5501184	3526400	1993182	1109109
896	9352800	5919493	3293239	1826718
1024	15711381	9819613	5455340	3021419
1152	26552234	16595146	9093230	5106198
1280	37173128	23233205	12730523	7148678

Table 2: Percentage distribution of total time among algorithm steps.

P	Step	$n \to$ 128	256	384	512
1	1	99.30	99.60	99.86	99.87
	2	0.43	0.25	0.09	0.08
	3	0.27	0.15	0.05	0.05
2	1	99.45	99.67	99.89	99.89
	2	0.30	0.17	0.07	0.07
	3	0.25	0.05	0.04	0.04
4	1	99.48	99.88	99.90	99.91
	2	0.27	0.07	0.06	0.05
	3	0.23	0.04	0.03	0.04
8	1	99.52	99.90	99.92	99.92
	2	0.25	0.06	0.05	0.05
	3	0.23	0.04	0.04	0.04

Table 3: Efficiency as a function of matrix size for $P = 2,\ 4$ and 8..

n	E_2	E_4	E_8
128	0.52	0.48	0.44
256	0.64	0.56	0.50
384	0.69	0.61	0.54
512	0.73	0.64	0.57
640	0.76	0.67	0.60
768	0.78	0.69	0.62
896	0.79	0.71	0.64
1024	0.80	0.72	0.65
1152	0.80	0.73	0.65
1280	0.80	0.73	0.65

Parallel Computing, 23(13):2067-2074, December 15, 1997.