A PARALLEL PIVOTAL ALGORITHM FOR SOLVING THE LINEAR COMPLEMENTARITY PROBLEM

by

Karen M. Thompson

Computer Sciences Technical Report #707

July 1987

A Parallel Pivotal Algorithm For Solving the Linear Complementarity Problem

Karen M. Thompson

Computer Sciences Department

University of Wisconsin, Madison, Wisconsin 53706

Abstract We propose a parallel implementation of the classical Lemke's algorithm for solving the linear complementarity problem. The algorithm is designed for a loosely coupled network of computers which is characterized by relatively high communication costs. We provide an accurate prediction of speedup based on a simple operation count. The algorithm produces speedups near p, where p is the number of processors, when tested on large problems as demonstrated by computational results on the CRYSTAL token–ring multicomputer and the Sequent Balance 21000 multiprocessor.

Key Words: Parallel algorithms, Lemke's algorithm, linear complementarity problem, distributed algorithms.

Abbreviated title: Parallel pivotal solution of LCPs.

This material is based on research supported by National Science Foundation Grants DCR-8420963 and DCR-8521228 and Air Force Office of Scientific Research Grants AFOSR-86-0172 and AFOSR-86-0255.

1. INTRODUCTION

In this paper we propose a parallel distribution of Lemke's algorithm, [Lemke65], for solving LCPs. This distribution is designed to exploit a multi-computer architecture where a number of processors are loosely coupled such as CRYSTAL [DeWitt et al84].

Parallel distribution of algorithms such as Lemke's algorithm allows for solving much larger problems in shorter time. See [Schnabel84] for a discussion on parallel optimization as well as a survey of work done in parallel optimization. Bhide proposed a distributed algorithm for the simplex algorithm [Finkel et al86]. Preliminary testing was done on small LPs. The initial results showed communication overhead dominated the calculations.

Phillips and Rosen propose a parallel algorithm for solving the LCP based on solving a multiple cost row linear program [Phillips & Rosen86]. This algorithm handles an LCP when M is indefinite. However, they suggest using Lemke's algorithm when M is positive semidefinite. They present computational experience on the CRAY X-MP/48.

A major trend in computing has been the creation of large networks of computer workstations. See [Agrawal & Jagadish86] for a model for parallel processing on these networks. These networks have relatively high communication costs. Therefore it is crucial that algorithms designed to run on these networks minimize the need to communicate. The proposed parallel implementation of Lemke's algorithm is tested on the CRYSTAL multi-computer. CRYSTAL is a set of twenty

VAX – 11/750 computers with 2 megabytes of memory each connected by a 80 megabit/sec Proteon ProNet token ring. We also compare computational results with an implementation on a tightly coupled multiprocessor, the Sequent Balance 21000. This machine consists of eight processors running at 10 MHz, with a 8 kbyte cache sharing a global memory via a 32-bit bus, with 8 megabytes of physical memory.

Section 2 will discuss Lemke's Algorithm. Section 3 will discuss the implementation of a parallel Lemke's algorithm designed for multi-computers. Finally, Section 4 will present a model for predicting speedups. Computational results are compared with the expected speedups. A comparison of results with the Balance 21000 will be presented.

We briefly describe our notation now. For a vector z in the n–dimensional real space $\mathbb{R}^n$, z_+ will denote the vector in $\mathbb{R}^n$ with components $(z_+)_i = \max\{z_i, 0\}$, $i = 1, \ldots, n$. $\mathbb{R}^{m \times n}$ will denote the set of all $m \times n$ real matrices. For $A \in \mathbb{R}^{m \times n}$, A^T will denote the transpose, $A_{i.}$ will denote the ith row, $A_{.j}$ will denote the jth column and a_{ij} the element in row i and column j.

2. LEMKE'S ALGORITHM FOR SOLVING THE LCP

Pivotal methods are based on moving from one vertex to another on a polyhedral set S where

$$S := \{z | z \geq 0, Mz + q \geq 0\} \tag{2.1}$$

Mangasarian has shown that the LCP is equivalent to a constrained minimization of a concave function [Mangasarian78] which we state in the following lemma.

Lemma 3.1. *(LCP equivalence with a concave minimization problem) Let S be defined as in (2.1). Then $\bar{z}$ solves the LCP if and only if*

$$0 = \sum_{i=1}^{n} \bar{z}_i - (\bar{z}_i - (M\bar{z} + q)_i)_+ = \min_{z \in S} \sum_{i=1}^{n} z_i - (z_i - (Mz + q)_i)_+$$

Proof :

First we note that the objective function in the minimization problem is non-negative on S. Now suppose $\bar{z}$ solves the LCP. We need only show that $0 = \sum_{i=1}^{n} \bar{z}_i - (\bar{z}_i - (M\bar{z} + q)_i)_+$.

Case I:

Suppose $\bar{z}_i = 0$. Then

$$\bar{z}_i - (\bar{z}_i - (M\bar{z} + q)_i)_+ = -(-(M\bar{z} + q)_i)_+ = 0$$

since

$$(M\bar{z} + q)_i \geq 0.$$

Case II:

Suppose $\bar{z}_i > 0$. Then

$$\bar{z}_i - (\bar{z}_i - (M\bar{z} + q)_i)_+ = \bar{z}_i - (\bar{z}_i)_+ = 0$$

since

$$(M\bar{z} + q)_i = 0.$$

Therefore

$$0 = \sum_{i=1}^{n} \bar{z}_i - (\bar{z}_i - (M\bar{z} + q)_i)_+.$$

Hence $\bar{z}$ solves the minimization problem. We now show that if $\bar{z}$ solves the minimization problem that $\bar{z}$ solves the LCP.

Suppose $\bar{z}$ solves the minimization problem. Then $\bar{z}_i - (\bar{z}_i - (M\bar{z} + q)_i)_+ = 0$ for $i = 1, 2, \ldots, n$.

Case I: Suppose $\bar{z}_i - (M\bar{z} + q)_i \geq 0 \Rightarrow (M\bar{z} + q)_i = 0 \Rightarrow \bar{z}_i \geq 0$.

Case II: Suppose $\bar{z}_i - (M\bar{z} + q)_i < 0 \Rightarrow \bar{z}_i = 0 \Rightarrow (M\bar{z} + q)_i \geq 0$.

Therefore $\bar{z}$ solves the LCP. ∎

The LCP has a solution at a vertex as a consequence of the following lemma found in [Mangasarian78].

Lemma 2.2. *If the linear complementarity problem has a solution, it has a solution at a vertex of S.*

Proof :

Because S is contained in the nonnegative orthant, it does not contain any straight lines (that go to infinity on both ends) and hence by Corollary 32.3.2 in [Rockafellar70] the concave minimization problem must have a solution at a vertex of S, which by Lemma 2.1 must solve the LCP. ∎

Lemke's algorithm [Lemke65] is based on the following equivalence obtained by enlarging the space of the problem by adding artificial variables.

$$\left\{\begin{array}{r} Mz+q\geq 0 \\ \\ z\geq 0 \\ \\ z(Mz+q)=0 \end{array}\right\} \iff \left\{\begin{array}{r} w=Mz+ez_0+q\geq 0 \\ \\ w_0=z_0 \\ \\ (z,z_0)\geq 0 \\ \\ (z,z_0)\begin{pmatrix} w \\ w_0 \end{pmatrix}=0 \end{array}\right\}$$

This representation has the advantage that we can easily identify a feasible point. Before we specify the algorithm we need to state the following definition.

Definition 2.1. *The points* $(z,z_0),(w,w_0)$ *are said to be* **almost complementary** *if* $z_i w_i = 0$ *except for at most one* i.

We are now ready to specify the algorithm due to Lemke.

Algorithm 2.1. (Lemke's Algorithm)

Step I Add an artificial nonbasic variable z_0 and an artificial basic variable $w_0 = z_0$. Increase the value of z_0 until the most negative w_i becomes zero. This corresponds to pivoting on the row with most negative q_i and column z_0. The problem is now feasible and almost complementary.

Step II Exchange a nonbasic variable and some basic variable while maintaining feasibility and almost complementarity. Therefore the nonbasic variable that enters the basis is the variable complementary to the variable which just became nonbasic. Continue this step until the process terminates at a solution or an unbounded ray.

The LCP is feasible if S is nonempty. Lemke's algorithm will converge to a solution if the LCP is feasible and M is copositive plus or a P- matrix.

3. IMPLEMENTATION OF PARALLEL LEMKE

The time consuming portion of Lemke's algorithm is the pivoting step. The pivoting step is $O(n^2)$ therefore when n is large each iteration will be very time consuming. The idea then is to distribute the pivoting among r processors in the hopes of obtaining a solution r times as fast.

Suppose we're trying to solve the LCP and define

$$A := [M \quad e \quad q]$$

where e is a vector of ones. Then A is the pivot matrix and $A_{.(n+2)}$ is the right hand side where n is the dimension of the LCP. Suppose that r is the pivot row and s is the pivot column, then for $i \neq r, j \neq s$ we have

$$a_{ij} \leftarrow a_{ij} - \frac{a_{is}a_{rj}}{a_{rs}}$$

The key to parallelizing Lemke's algorithm is to note that when updating the pivot matrix we need only elements in the pivot row and the pivot column. Therefore there are two natural ways to divide the pivot matrix, in horizontal blocks or in vertical blocks. Then each horizontal or vertical block will be updated simultaneously in separate processors.

If we divide the pivot matrix into horizontal blocks each processor has the portion of the pivot column it needs to update all the elements in its block. Therefore each processor needs only the pivot row to complete the update. Likewise if we divide the pivot matrix into column blocks we need only the pivot column to complete the update.

Because we are considering an implementation on a loosely coupled network with relatively heavy communication costs we would like to avoid unnecessary communication. In Lemke's algorithm we must determine the pivot column and row. The pivot column is always the column corresponding to the complement of the variable which last became nonbasic. Therefore locating the pivot column is a simple matter of keeping track of the location of the variables. This is easily done independently in each processor. However, we find that the horizontal partition results in extra communications to complete the ratio test necessary to determine the appropriate pivot row.

The pivot row is selected by employing a ratio test as follows

$$r = \underset{1 \leq i \leq n}{\operatorname{argmin}} \{-\frac{a_{i,n+2}}{a_{is}} \mid a_{is} < 0\}.$$

If the A is partitioned in horizontal blocks each processor would find the minimum ratio of the portion of the pivot column it owned. Then a communication to a master processor would be necessary whereby this processor would determine the minimum of all ratios and notify the appropriate processor with another communication. The processor which owned the pivot row would then proceed to share it with all

other processors. Therefore this results in $2(p-1)+1$ communications, the first $p-1$ communications are needed to send the master the minimum ratio, the next communication to notify the proper processor, and finally the last $p-1$ to send the pivot row.

If the pivot matrix was divided in vertical blocks then each processor either waits for or sends the pivot column. Then if each processor holds a copy of $A_{.(n+2)}$ each processor can independently determine the pivot row and proceed with the update. Therefore the vertical division saves p communications.

We suggest the following scheme. Each processor j holds a column block of the matrix A and a copy of $A_{.n+2}$ in its memory. Then processor j proceeds through the following steps.

Step I. *Determine the pivot column s*

If s is in memory

send pivot column

else

receive pivot column

Step II. *Ratio Test*

Step III. *Pivot*

Step IV. *If complementarity is satisfied then*

stop

else

go to step I.

4. COMPUTATIONAL EXPERIENCE

Table 4.1 displays the time in seconds to solve three randomly generated problems of dimension 50, 200, and 425 on the CRYSTAL multi–computer. The problems were tested using one to eight processors.

Table 4.1

CRYSTAL TIME IN SECONDS

processors	*Problem Dimension*		
p	50	200	425
1	1.8163	90.8340	629.7770
2	1.0159	46.5438	316.573
3	.7808	31.6945	212.6680
4	.6771	24.6202	161.5050
5	.6261	20.4335	129.9620
6	.5972	17.6500	110.2570
7	.6011	15.7600	96.0057
8	.6164	14.2546	86.1556

To assess the speedup from using p processors we define the following measurement of speedup.

$$S_p := \frac{\text{Computing time on one processor}}{\text{Computing time using p processors.}}$$

An upper bound on S_p would be p. This would occur if there was no overhead due to the parallelism of the algorithm. Therefore a goal in producing an efficient parallel algorithm would be to try to approach a speedup of p for p processors.

Expected speedup based on an elementary operation count for the proposed algorithm will be defined as

$$S_p^e := \frac{\text{Expected time for one iteration on one processor}}{\text{Expected time for one iteration on } p \text{ processors}}$$

Note that we have both serial and parallel components of the algorithm. Finding the pivot column and finding the pivot row are repeated independently in each processor and therefore can be thought of as serial. This step is $O(n)$. Pivoting is done in parallel and takes $\frac{2n^2}{p}$ steps. We also must take into account communication costs. At each iteration one processor sends a pivot column in $k(n)$ buffers to the $p-1$ remaining processors. Here $k(n)$ is the number of buffers needed to send a vector of n real numbers and is given by $k(n) := \lceil \frac{n}{s} \rceil$. Therefore $k(n)$ is the smallest integer greater than or equal to $\frac{n}{s}$ where s is the number of elements that can be transmitted by one buffer. Therefore define

$$S_p^e := \frac{2n^2 + \beta n}{2n^2/p + \beta n + \gamma k(n)(p-1)} = \frac{2n + \beta}{2n/p + \beta + \gamma k(n)(p-1)/n}$$

where β is an unknown parameter and γ is a ratio of time for communication divided by the time for one operation. Using this definition we can maximize speedup for fixed n by using $p = \sqrt{\frac{2}{\gamma k(n)}} \times n$ processors. A least squares fit over observed data

gives us $\beta = 4.4014$ and $\gamma = 143.5467$. Therefore we would expect speedup to be maximized when p is 5.9, 16.7 and 28.9 for dimensions 50, 200,and 425 respectively. Clearly, p must be an integer number. Figure 4.1 shows the percentage of the maximum speedup we obtain for a fixed dimension given the number of processors p. We see that the curve is steep for $p < \bar{p}$ where $\bar{p} = \sqrt{\frac{2}{\gamma k(n)}} \times n$. However, for $p > \bar{p}$ the curve decreases slowly. Therefore when there is a question of rounding $\bar{p}$ one should round up rather than down. Table 4.2 compares expected speedup with the speedups obtained on test problems for problem sizes 50, 200, and 425. Figures 4.2, 4.3 and 4.4 display table 4.2 graphically. Finally figure 4.5 graphs the computed speedups for the different dimensions so we can see that as n grows large the speedup approaches p for $p << n$.

We would expect to get better speedups on a shared memory machine like the Balance 21000. The implementation on Balance 21000 is similar except we save on communication costs because the pivot column is in shared memory. Table 4.3 compares computed speedups on Balance 21000 with computed speedups on CRYSTAL for problem sizes 50,200,425. We see that the CRYSTAL speedups drop off faster as we move to more processors than the Balance 21000 speedups. This drop is noticed almost immediately for the smaller dimensioned problem, however CRYSTAL is competitive with Balance 21000 at dimension 425 through seven processors.

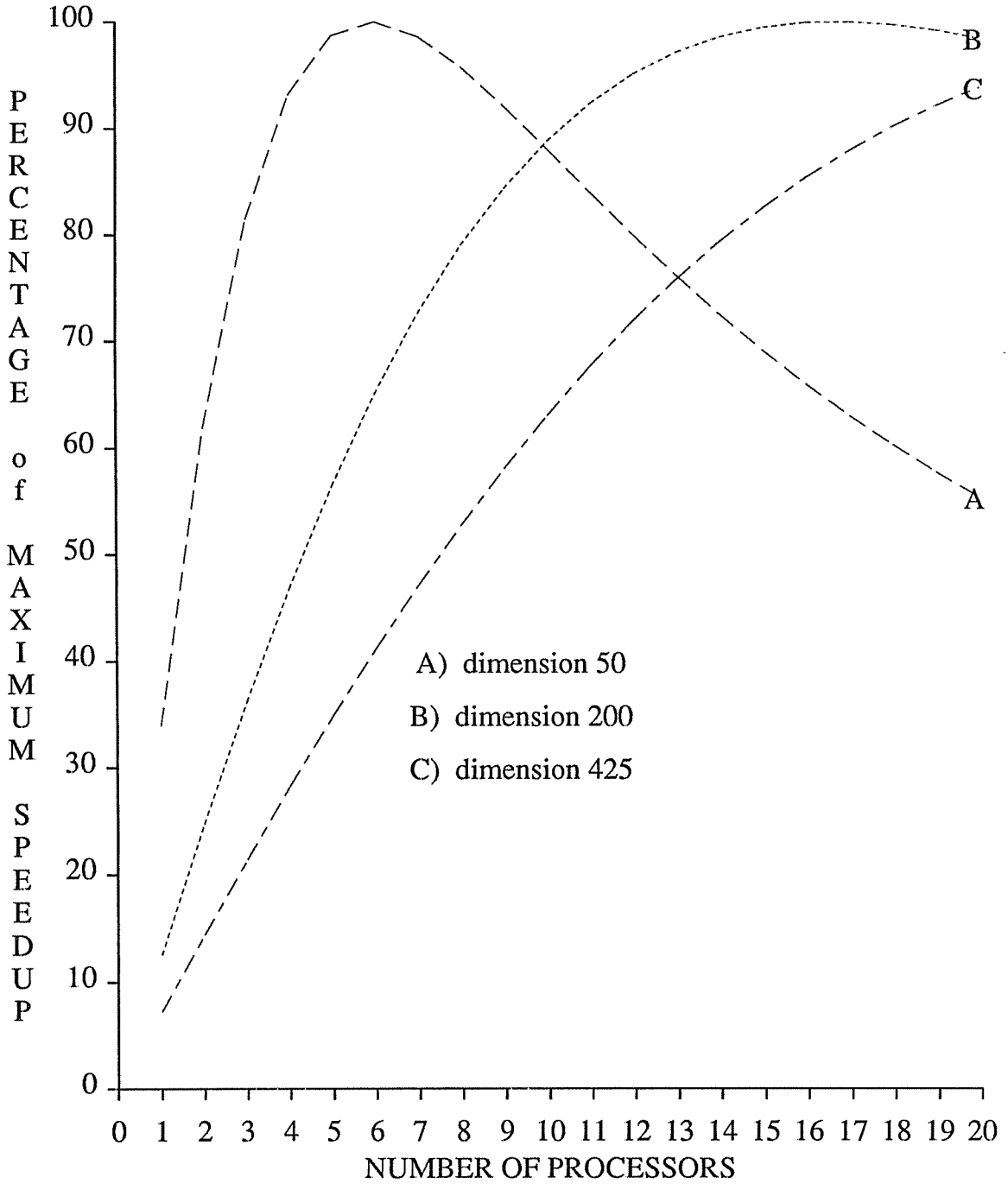

Figure 4.1 Percentage of Maximum Speedup

Table 4.2

EXPECTED VERSUS COMPUTED SPEEDUPS

	50		200		425	
p	S_p	S_p^e	S_p	S_p^e	S_p	S_p^e
2	1.7878	1.8227	1.9515	1.9646	1.9893	1.9850
3	2.3262	2.4009	2.8659	2.8760	2.9613	2.9486
4	2.6824	2.7458	3.6894	3.7198	3.8994	3.8845
5	2.9009	2.9087	4.4453	4.4857	4.8458	4.7875
6	3.0413	2.9467	5.1464	5.1677	5.7119	5.6529
7	3.0216	2.9065	5.7635	5.7634	6.5597	6.4767

Table 4.3

CRYSTAL VERSUS BALANCE SPEEDUPS

	50		200		425	
p	CRYS	Balan	CRYS	Balan	CRYS	Balan
2	1.7878	1.8108	1.9646	1.8891	1.9959	1.9893
3	2.3262	2.3928	2.8760	2.8309	2.9613	2.9762
4	2.6824	3.0454	3.7198	3.6932	3.8994	3.9860
5	2.9009	3.3500	4.4857	4.4933	4.8945	4.9521
6	3.0413	3.7222	5.1677	5.4505	5.7119	5.8777
7	3.0216	3.9411	5.7634	6.3699	6.5597	6.7854

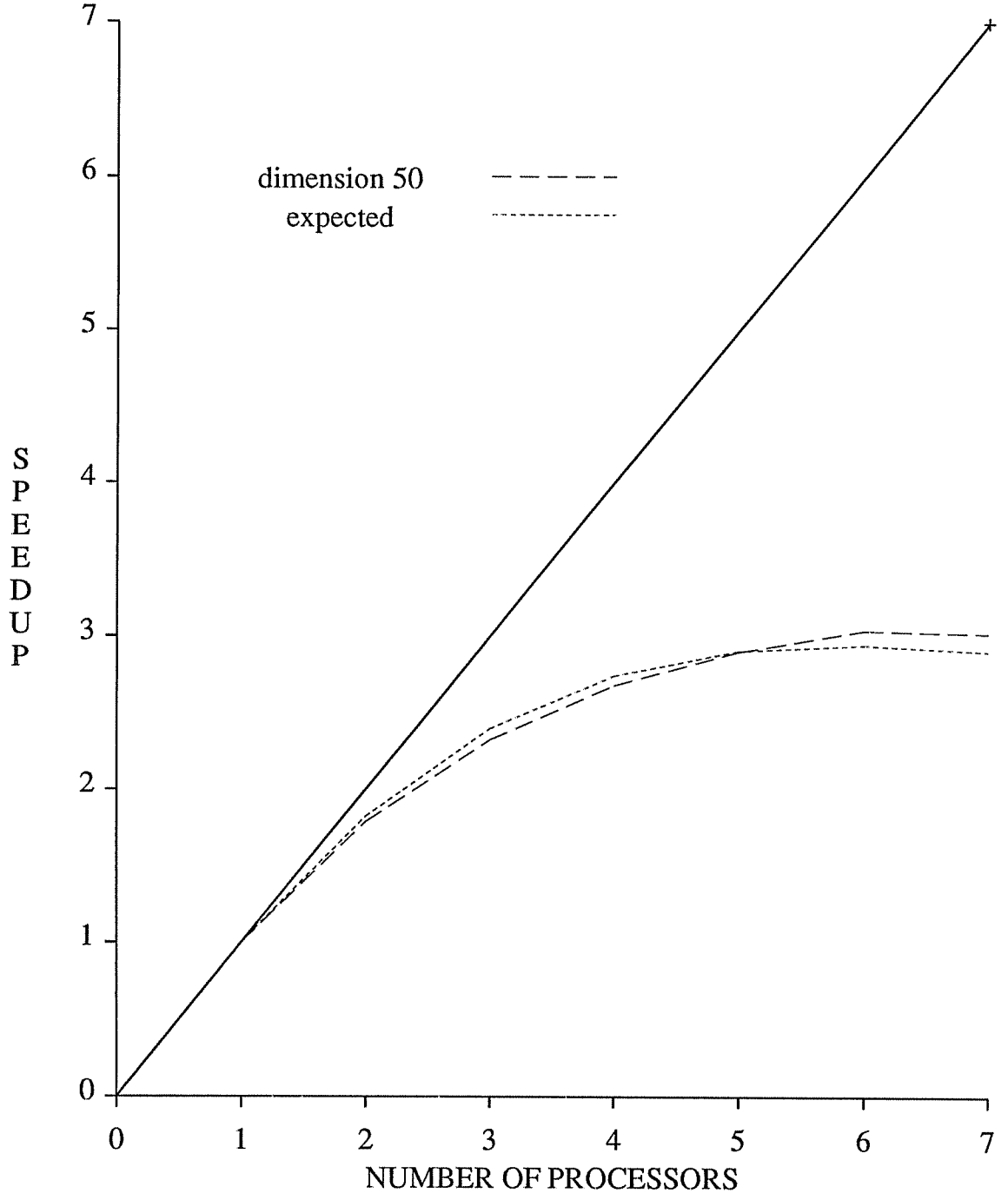

Figure 4.2 CRYSTAL Speedups: Expected versus Computed

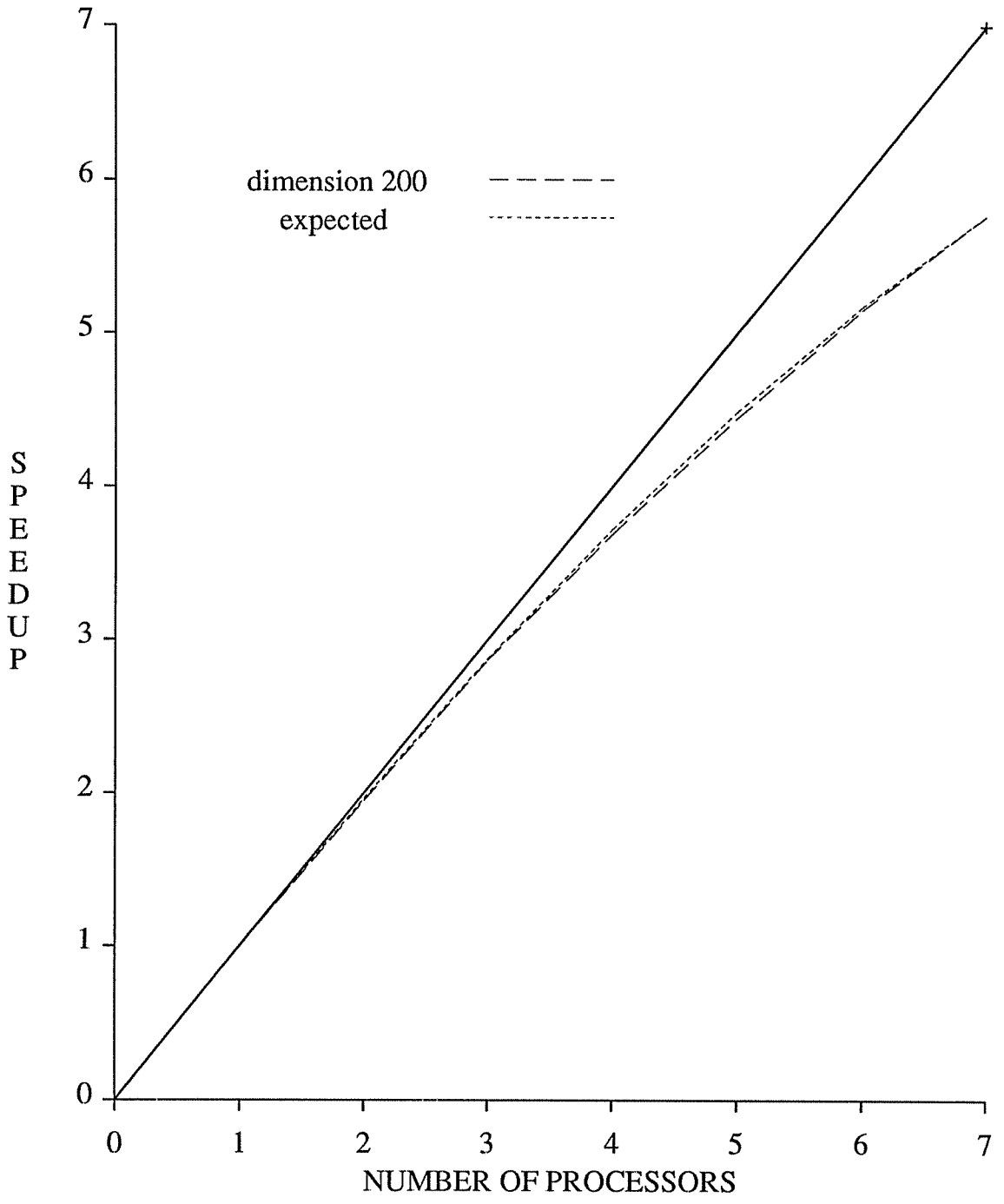

Figure 4.3 CRYSTAL Speedups: Expected versus Computed

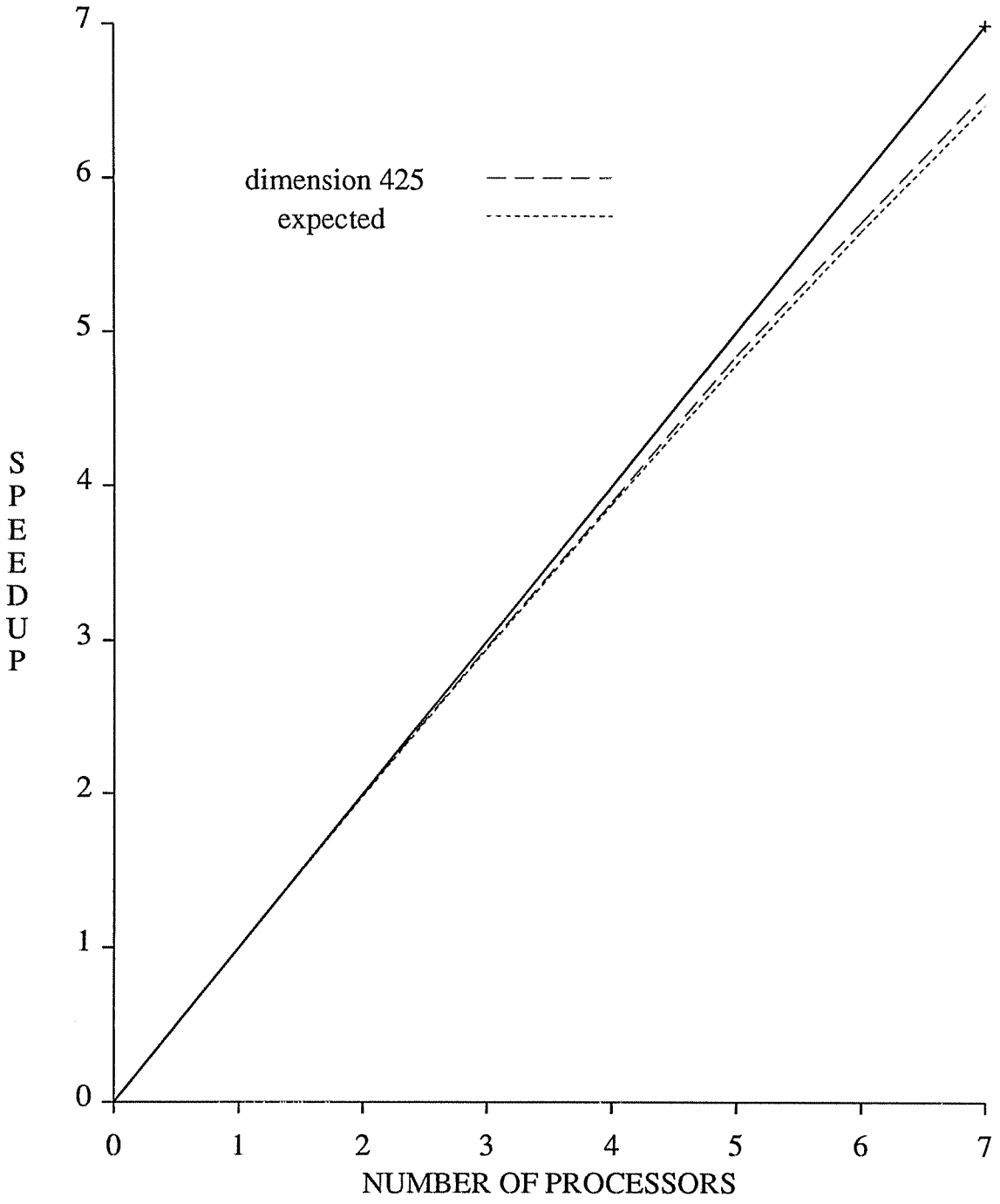

Figure 4.4 CRYSTAL Speedups: Expected versus Computed

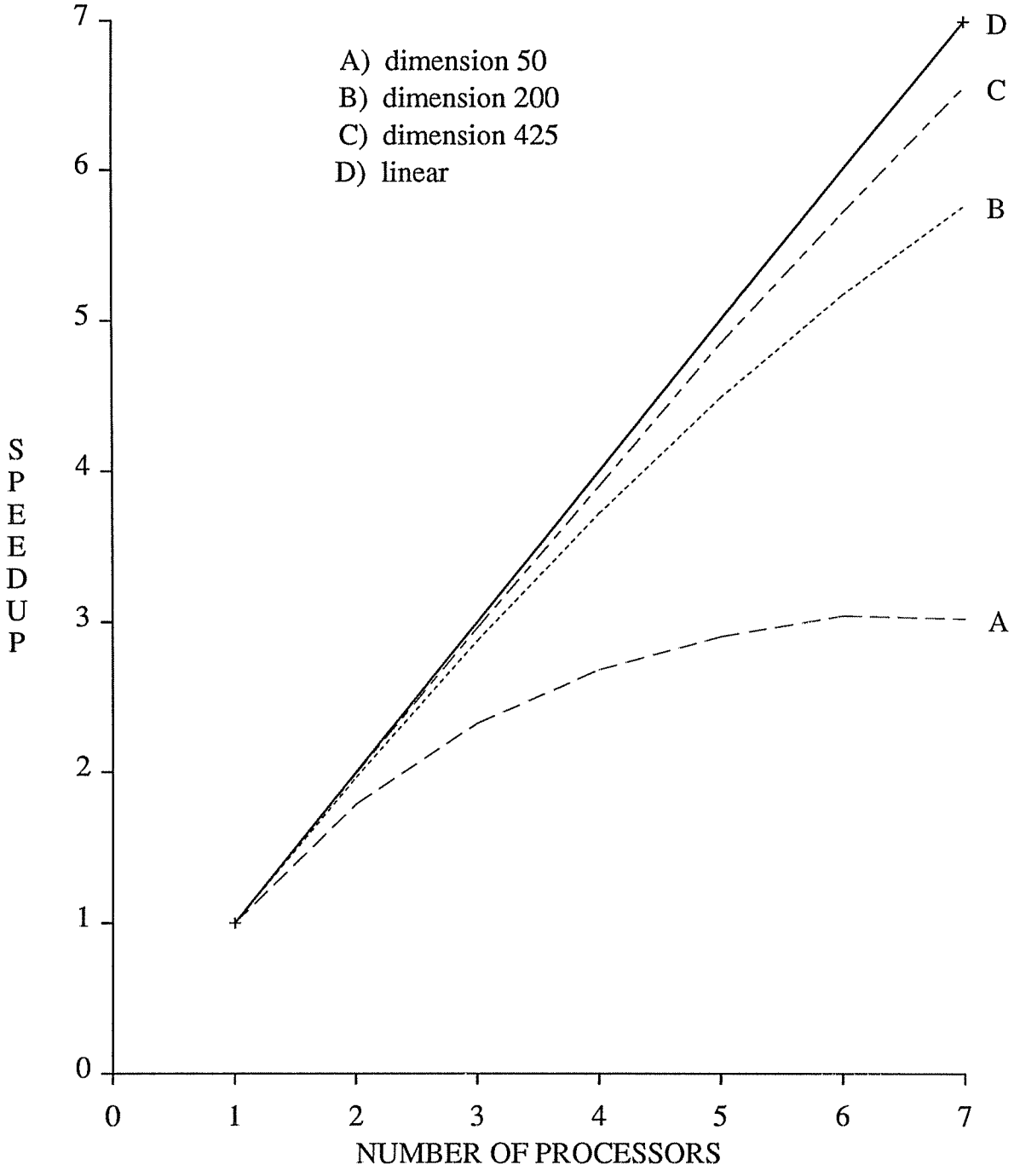

Figure 4.5 CRYSTAL Speedups

References

Agrawal, R. and Jagadish, H. V. (1986) Parallel Computation on Loosely–Coupled Workstations, Computer Technology Research Laboratory Technical Report, AT&T Bell Laboratories.

DeWitt, D. J., Finkel, R. and Solomon, M. (1984) The CRYSTAL Multicomputer: Design and Implementation Experience, Computer Sciences Technical Report #553, University of Wisconsin–Madison.

Finkel, R., Barzideh, B., Bhide, C. W., Lam, M. O., Nelson, D., Polisetty, R., Rajaraman, S., Steinberg, I., Venkatesh, G. A. (1986) Experience with Crystal, Charlotte and Lynx Second Report, Computer Sciences Technical Report #649, University of Wisconsin–Madison.

Lemke, C. E. (1965) Bimatrix Equilibrium Points and Mathematical Programming, Management Science, 11, 681–689.

Mangasarian, O. L. (1978) Characterization of Linear Complementarity Problems as Linear Programs, Mathematical Programming Study 7, North Holland, Amsterdam, The Netherlands, 74–87.

Phillips, A. T. and Rosen, J. B. (1986) Multitasking Mathematical Programming Algorithms, Computer Science Department Technical Report #86-10, University of Minnesota.

Rockafellar, R. T. (1970) "Convex Analysis", Princeton, University Press, Princeton, New Jersey.

Schnabel, R. B. (1984) Parallel Computing in Optimization, Department of Computer Science Technical Report #CU-CS-282-84, University of Colorado.