Nonlinear Analysis, Theory, Methods & Applications, Vol. 13, No. 11, pp. 1339-1350, 1989.
Printed in Great Britain.

0362-546X/89 $3.00 + .00

A GLOBALLY CONVERGENT PARALLEL ALGORITHM FOR ZEROS OF POLYNOMIAL SYSTEMS

ALEXANDER P. MORGAN
General Motors Research Laboratories, Warren, MI 48090-9055, U.S.A.

and

LAYNE T. WATSON
The University of Michigan, Ann Arbor, MI 48109-2117, U.S.A.

(*Received* 15 *November* 1988; *received for publication* 16 *December* 1988)

Key words and phrases: Polynomial system, homotopy, roots of polynomials, parallel computation, hypercube, globally convergent homotopy algorithm.

1. INTRODUCTION

POLYNOMIAL systems of equations frequently arise in solid modelling, robotics, computer vision, chemistry, chemical engineering, and mechanical engineering. Locally convergent iterative methods such as quasi-Newton methods may diverge or fail to find all meaningful solutions of a polynomial system. This paper proposes a parallel homotopy algorithm for polynomial systems of equations that is guaranteed globally convergent (always converges from an arbitrary starting point) with probability one, finds all solutions to the polynomial system, and has a large amount of inherent parallelism. Some mathematical background material on homotopy algorithms and polynomial systems is included. A particular coarse-grained decomposition strategy is given in detail, and the results of some experiments on an Intel iPSC-32 hypercube are presented.

Solving nonlinear systems of equations is a central problem in numerical analysis, with enormous significance for science and engineering. A very special case, namely small polynomial systems of equations with many real solutions, occurs frequently enough in solid modelling, robotics, computer vision, chemical equilibrium computations, chemical process design, mechanical engineering, and other areas to justify special algorithms. To put polynomial systems in perspective and for the purpose of discussion here, there are three classes of nonlinear systems of equations: (1) large systems with sparse Jacobian matrices, (2) small transcendental (nonpolynomial) systems with dense Jacobian matrices, and (3) small polynomial systems with dense Jacobian matrices. Sparsity for small problems is not significant, and large systems with dense Jacobian matrices are intractable, so these two classes are not counted. Of course medium sized problems are also of practical interest, but the boundaries between small, medium, and large change with computer hardware technology and algorithmic development. Depending on algorithmic efficiency, hardware capability, and the significance of sparsity, a medium sized problem is treated like it belongs to one of the above three classes anyway, so there is no need for a "medium" class.

Large sparse nonlinear systems of equations, such as equilibrium equations in structural mechanics, have two aspects: highly nonlinear and recursive scalar computations, and large

matrix, vector operations. There is a great amount of parallelism in both aspects, but the nature of the parallelism is very different (or so it seems). Small dense transcendental systems of equations pose a major challenge, since they involve recursive, scalar intensive computation with a small amount of linear algebra. It has been argued that the communication overhead of hypercube machines makes them unsuited for such problems, but the issue is still open and algorithmic breakthroughs are yet possible. Polynomial systems are unique in that they have many solutions, of which several may be physically meaningful, and that there exist homotopy algorithms guaranteed to find all these meaningful solutions. The very special nature of polynomial systems and the power of modern homotopy algorithms are often not fully appreciated, perhaps because the theory of these modern globally convergent probability-one homotopy methods for polynomial systems is grounded in algebraic geometry and differential topology.

Algorithms for solving nonlinear systems of equations can be broadly classified as (1) locally convergent or (2) globally convergent. The former includes Newton's method, various quasi-Newton methods, and inexact Newton methods. The latter includes continuation, simplicial methods, and probability-one homotopy methods. These algorithms are qualitatively significantly different, and their performance on parallel systems may very well be the reverse of their performance on serial processors. Locally convergent methods such as the very popular least-change secant update quasi-Newton methods are very difficult to apply to polynomial systems with many solutions, because the solution they converge to cannot be controlled easily. Furthermore, although model trust region quasi-Newton methods are often touted as globally convergent, they in fact frequently fail by converging to local minima that are not zeros, unless the starting points are sufficiently close to the zeros. For polynomial systems, obtaining these good starting points is a highly nontrivial impediment.

Much work has been done on solving linear systems of equations on parallel computers, mostly on vector machines [4, 5, 7, 8, 10–12, 14–16, 18, 23, 24]. Some work has been done on nonlinear equations and Newton's method [28, 31, 36, 37], and on finding the roots of a single polynomial equation [9, 27]. Parallel algorithms for polynomial systems have not been studied, nor have parallel homotopy algorithms for nonlinear systems of equations.

A hypercube computer consists of 2^n processors (nodes), each with memory, floating-point hardware, and (possibly) communication hardware (see, e.g. [25] or [29]). The nodes are independent and asynchronous, and connected to each other like the corners of an n-dimensional cube. At first glance it appears that the time needed to solve a problem on one processor is reduced by a factor of 2^n for a hypercube with that number of processors. Of course this reduction is only theoretical since it assumes that the computations are equally divided among the nodes and it ignores the time associated with communication among nodes. Typically the overhead for this communication is considered to be small relative to the time used for computational purposes. The present article proposes a particular coarse-grained decomposition of a globally convergent probability-one homotopy algorithm for polynomial systems of equations. The parallel algorithm is tested on some real engineering problems in chemical engineering, solid modelling, and robotics obtained from General Motors Research Laboratories.

Only polynomial systems and homotopy algorithms are considered here. Large sparse nonlinear systems, small transcendental systems, and quasi-Newton algorithms will be considered in future work. Section 2 summarizes the mathematics behind the homotopy algorithm, and sketches a computer implementation of the serial algorithm. Section 3 discusses the special

case of polynomial systems in some detail, giving the theoretical justification for the claim that the homotopy algorithm is guaranteed to be globally convergent and to find all solutions. Section 4 describes the parallel homotopy algorithm for polynomial systems. Computational results on an Intel iPSC-32 hypercube are presented and discussed in Section 5.

2. HOMOTOPY ALGORITHM

Let E^p denote p-dimensional real Euclidean space, and let $F: E^p \to E^p$ be a C^2 (twice continuously differentiable) function. The general problem is to solve the nonlinear system of equations

$$F(x) = 0.$$

The fundamental mathematical result behind the homotopy algorithm (see [6, 19–22, 32–35]) is the following.

PROPOSITION 1. Let $F: E^p \to E^p$ be a C^2 map and $\rho: E^m \times [0, 1) \times E^p \to E^p$ a C^2 map such that:

(1) the Jacobian matrix D_ρ has full rank on $\rho^{-1}(0)$ and for fixed $a \in E^m$;
(2) $\rho(a, 0, x) = 0$ has a unique solution $W \in E^p$;
(3) $\rho(a, 1, x) = F(x)$;
(4) the set of zeros of $\rho_a(\lambda, x) = \rho(a, \lambda, x)$ is bounded.

Then for almost all $a \in E^m$ there is a zero curve γ of

$$\rho_a(\lambda, x) = \rho(a, \lambda, x),$$

along which the Jacobian matrix $D\rho_a(\lambda, x)$ has full rank, emanating from $(0, W)$ and reaching a zero $\bar{x}$ of F at $\lambda = 1$. Furthermore, γ has a finite arc length if $DF(\bar{x})$ is nonsingular.

The general idea of the algorithm is apparent from the proposition: just follow the zero curve γ of ρ_a emanating from $(0, W)$ until a zero $\bar{x}$ of $F(x)$ is reached (at $\lambda = 1$). Of course it is nontrivial to develop a viable numerical algorithm based on that idea, but at least conceptually, the algorithm for solving the nonlinear system of equations $F(x) = 0$ is clear and simple. A typical form for the homotopy map is

$$\rho_W(\lambda, x) = \lambda F(x) + (1 - \lambda)(x - W), \tag{1}$$

although more complicated homotopy maps tailored for special situations are commonly used in practice. The homotopy map ρ_W in (1) has the same form as a standard continuation or embedding mapping. However, there are two crucial differences. In standard continuation, the embedding parameter λ increases monotonically from 0 to 1 as the trivial problem $x - W = 0$ is continuously deformed to the problem $F(x) = 0$. The present homotopy method permits λ to both increase and decrease along γ with no adverse effect; that is, turning points present no special difficulty. The second important difference is that there are never any "singular points" which afflict standard continuation methods. The way in which the zero curve γ of ρ_a is followed and the full rank of $D\rho_a$ along γ guarantee this. Observe that proposition 1 guarantees that γ cannot just "stop" at an interior point of $[0, 1) \times E^p$.

The zero curve γ of the homotopy map $\rho_a(\lambda, x)$ (of which $\rho_W(\lambda, x)$ in (1) is a special case) can be tracked by many different techniques. Older curve tracking techniques were based on Gaussian elimination [17], which is less robust than QR factorization [3] based techniques. Some

modern curve tracking algorithms were designed with special applications in mind (e.g. [26]), and do not adapt well for the polynomial system situation. The most comprehensive curve tracking package currently available is HOMPACK [34], a joint development effort between Virginia Polytechnic Institute and State University, Sandia National Laboratories, General Motors Research Laboratories, and The University of Michigan. Complete algorithmic details and the theoretical justifications for several different curve tracking algorithms are given in [34] and [35]. The serial curve tracking algorithm described here is based on subroutine FIXPDF in [34].

Assuming that $F(x)$ is C^2 and a is such that proposition 1 holds, the zero curve γ is C^1 and can be parametrized by arc length s. Thus $\lambda = \lambda(s), x = x(s)$ along γ, and

$$\rho_a(\lambda(s), x(s)) = 0 \tag{2}$$

identically in s. Therefore

$$\frac{d}{ds}\rho_a(\lambda(s), x(s)) = D\rho_a(\lambda(s), x(s))\begin{pmatrix} \frac{d\lambda}{ds} \\ \frac{dx}{ds} \end{pmatrix} = 0, \tag{3}$$

$$\left\| \left(\frac{d\lambda}{ds}, \frac{dx}{ds} \right) \right\|_2 = 1. \tag{4}$$

With the initial conditions

$$\lambda(0) = 0, \qquad x(0) = W, \tag{5}$$

the zero curve γ is the trajectory of the initial value problem (3–5). When $\lambda(\bar{s}) = 1$, the corresponding $x(\bar{s})$ is a zero of $F(x)$. Thus all the sophisticated ODE techniques currently available can be brought to bear on the problem of tracking γ [30, 33].

If the computed points $y = (\lambda, x)$ wander too far from the true curve γ, the last computed point $(\bar{\lambda}, \bar{x})$ is used to calculate a new parameter vector $\bar{a}$ such that

$$\rho_{\bar{a}}(\bar{\lambda}, \bar{x}) = 0 \tag{6}$$

exactly, and the zero curve of $\rho_{\bar{a}}(\lambda, x)$ is followed starting from $(\bar{\lambda}, \bar{x})$. A rigorous justification for this strategy was given in [33].

In summary, the algorithm is:

1. Set $s := 0$, $y := (0, W)$, *ypold*: = *yp*: = $(1, 0, \ldots, 0)$, *restart*: = false, *error*: = initial error tolerance for the ODE solver.
2. If $y_1 < 0$ then go to 23.
3. If $s >$ some constant then
   4. $s := 0$.
   5. Compute a new vector a satisfying (6). If

$$\|\text{new } a - \text{old } a\| > 1 + \text{constant} * \|\text{old } a\|,$$

   then go to 23.
6. *ode error*: = *error*.
7. If $\|yp - ypold\|_\infty >$ (last arc length step) * constant, then *ode error*: = *tolerance* $\ll$ *error*.
8. *ypold*: = *yp*.

9. Take a step along the trajectory of (3-5) with the ODE solver. $yp = y'(s)$ is computed for the ODE solver by 10-12:
   10. Find a vector z in the kernel of $D\rho_a(y)$ using Householder reflections.
   11. If $z^t ypold < 0$, then $z := -z$.
   12. $yp := z/\|z\|$.
13. If the ODE solver returns an error code, then go to 23.
14. If $y_1 < 0.99$, then go to 2.
15. If *restart* = true, then go to 20.
16. *restart*: = true.
17. *error*: = final accuracy desired.
18. If $y_1 \geq 1$, then set (s, y) back to the previous point (where $y_1 < 1$).
19. Go to 4.
20. If $y_1 < 1$ then go to 2.
21. Obtain the zero (at $y_1 = 1$) by interpolating mesh points used by the ODE solver.
22. Normal return.
23. Error return.

3. POLYNOMIAL SYSTEMS

Section 2 described a homotopy algorithm for finding a single solution to a general nonlinear system of equations $F(x) = 0$. Proposition 1 provided the theoretical guarantee of convergence. The rich structure and multiple solutions of polynomial systems dictate that the general theory in Section 2 must be sharpened. This section develops a globally convergent (with probability one) homotopy algorithm that finds all solutions to a polynomial system, and provides the theoretical justification for that algorithm.

Suppose that the components of the nonlinear function $F(x)$ have the form

$$F_i(x) = \sum_{k=1}^{n_i} a_{ik} \prod_{j=1}^{n} x_j^{d_{ijk}}, \qquad i = 1, \ldots, n. \tag{7}$$

The ith component $F_i(x)$ has n_i terms, the a_{ik} are the coefficients, and the degrees d_{ijk} are nonnegative integers. The degree of F_i is

$$d_i = \max_k \sum_{j=1}^{n} d_{ijk}.$$

For technical reasons it is necessary to consider $F(x)$ as a map $F: C^n \to C^n$, where C^n is n-dimensional complex Euclidean space. A system of n polynomial equations in n unknowns, $F(x) = 0$, may have many physically meaningful solutions, all of which are desired. Examples include the six-degree-of-freedom robot manipulator equations, surface intersections for solid modelling, the structure from motion equations in computer vision, and distillation tower equations in chemical engineering. In a technical mathematical sense, all solutions (real, complex, and at infinity) of a polynomial system are created equal. Thus to be assured of finding all the real solutions one must find all solutions; it turns out that this is indeed possible.

A complete and rigorous exposition of the mathematical properties of polynomial systems and how homotopy theory applies to them can be found in [19-21]. Only the essential facts are

repeated here. Define $G: C^n \to C^n$ by

$$G_j(x) = b_j x_j^{d_j} - a_j, \qquad j = 1, \ldots, n, \tag{8}$$

where a_j and b_j are nonzero complex numbers and d_j is the degree of $F_j(x)$, for $j = 1, \ldots, n$. Define the homotopy map

$$\rho_c(\lambda, x) = (1 - \lambda)G(x) + \lambda F(x), \tag{9}$$

where $c = (a, b)$, $a = (a_1, \ldots, a_n) \in C^n$ and $b = (b_1, \ldots, b_n) \in C^n$. Let $d = d_1 \ldots d_n$ be the total degree of the system. The fundamental homotopy result, proved in [20] and discussed at length in [21], is the following theorem.

THEOREM. For almost all choices of a and b in C^n, $\rho_c^{-1}(0)$ consists of d smooth paths emanating from $\{0\} \times C^n$, which either diverge to infinity as λ approaches 1 or converge to solutions to $F(x) = 0$ as λ approaches 1. Each geometrically isolated solution of $F(x) = 0$ has a path converging to it.

Define $F'(y)$ to be the homogenization of $F(x)$:

$$F_j'(y) = y_{n+1}^{d_j} F_j(y_1/y_{n+1}, \ldots, y_n/y_{n+1}), \qquad j = 1, \ldots, n. \tag{10}$$

A basic result on the structure of the solution set of a polynomial system is the following classical theorem of Bezout [21].

THEOREM. There are no more than d isolated solutions to $F'(y) = 0$ in complex projective n-space CP^n. If $F'(y) = 0$ has only a finite number of solutions in CP^n, it has exactly d solutions, counting multiplicities.

Define a linear function

$$u(y_1, \ldots, y_{n+1}) = \xi_1 y_1 + \xi_2 y_2 + \cdots + \xi_{n+1} y_{n+1}$$

where $\xi_1, \ldots, \xi_{n+1}$ are nonzero complex numbers, and define $F'': C^{n+1} \to C^{n+1}$ by

$$\begin{aligned} F_j''(y) &= F_j'(y), \qquad j = 1, \ldots, n, \\ F_{n+1}''(y) &= u(y) - 1. \end{aligned} \tag{11}$$

The significance of $F''(y)$ is given by the following.

THEOREM [19]. If $F'(y) = 0$ has only a finite number of solutions in CP^n, then $F''(y) = 0$ has exactly d solutions (counting multiplicities) in C^{n+1} and no solutions at infinity, for almost all $\xi \in C^{n+1}$.

Under the hypothesis of the theorem, all the solutions of $F'(y) = 0$ can be obtained as lines through the solutions to $F''(y) = 0$. Thus all the solutions to $F(x) = 0$ can be obtained easily from the solutions to $F''(y) = 0$, which lie on bounded homotopy paths (since $F''(y) = 0$ has no solutions at infinity).

The import of the above theory is that the nature of the zero curves of the projective transformation $F''(y)$ of $F(x)$ is as shown in Fig. 1. There are exactly d (the total degree of F) zero curves, they are monotone in λ, and have finite arc length. The homotopy algorithm is to track these d curves, which contain all isolated (transformed) zeros of F.

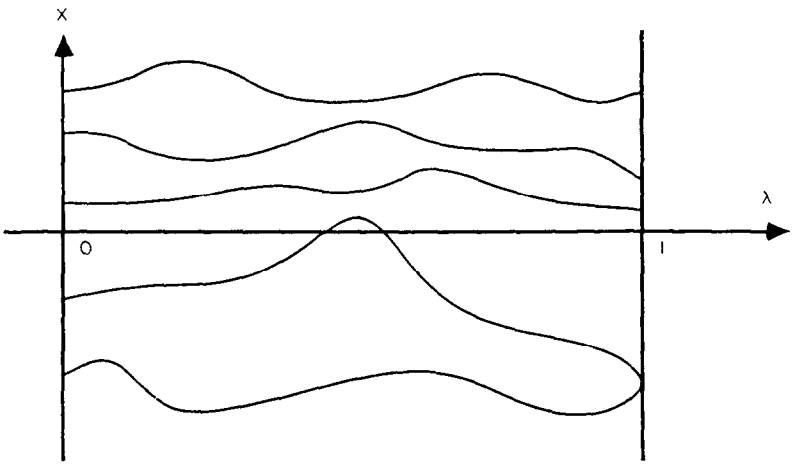

Fig. 1. The set $\rho_a^{-1}(0)$ for a transformed polynomial system.

4. PARALLEL HOMOTOPY ALGORITHM

Polynomial systems arising from diverse areas such as chemistry, computational geometry, solid modelling, robotics, chemical engineering, mechanical engineering, and computer vision come in various sizes. A small problem might have total degree $d < 100$ and a large problem, say, $d > 1000$. Given that d homotopy paths (containing all solutions, as explained in the previous section) are to be tracked, there are two extreme approaches using a hypercube.

The first extreme, with the coarsest granularity possible, is to assign one path to each node processor, with the host controlling the assignment of paths to the nodes, keeping as many nodes busy as possible, and post-processing the answers computed by the nodes.

The second extreme, with the finest granularity, is to track all d paths on the host processor, distributing the curve tracking logic, numerical linear algebra, polynomial system evaluation, Jacobian matrix evaluation, and possibly other tasks amongst the nodes. Because of the high communication overhead (whether hardware of software) on a hypercube, this approach requires an immense amount of sophistication and analysis to prevent the communication costs from overwhelming the computational costs. Also the algorithm at this granularity would have to be a major modification of the serial algorithm. A possible advantage is that the load could be balanced better, resulting in an overall speedup over a coarser grained algorithm.

These issues are not simple, and much more research is needed on these and intermediate approaches. Neither extreme can be declared inferior *a priori*. The first approach, having a node track an entire path, is described below in detail. The subroutine POLSYS of HOMPACK [34] solves a polynomial system, using the curve tracking subroutine FIXPNF to track the d zero curves. The parallel hypercube code is a modification of POLSYS for the host program and of FIXPNF for the (identical) node programs. Note that the information passing between

the host and the nodes using SEND's and RECEIVE's is exactly the information that would ordinarily be passed using subroutine CALL's. We can therefore concentrate on the message passing scheme without getting mired in the details of what information (the argument list of FIXPNF [34]) is being transferred.

The parallel algorithm is:

FOR THE HOST:

(1) Initialize the data space and calculate a starting point for each path.
(2) SEND initializations and a starting point to a node.
(3) If the message in (2) is incomplete, go to (2).
(4) If another path needs to be assigned and a node is available, go to (2).
(5) Now wait for a message from a node.
(6) RECEIVE a "ready to transmit solution" message from a node (call it the "current" node).
(7) SEND an acknowledgement ("ready to receive" message) to the current node.
(8) If a "ready to transmit" message is received from another node, put the node identification into a queue until the current node completes transmitting a solution.
(9) RECEIVE a "solution" message from the current node.
(10) If the "solution" message is incomplete, go to (8).
(11) Process the solution sent by the current node and print it.
(12) If another path needs to be assigned, SEND initializations and a starting point to the current node.
(13) If the message in (12) is incomplete, go to (12).
(14) If any nodes are in the queue (see (8)), remove the first node from the queue, call it the current node and go to (7).
(15) If awaiting messages from any other nodes, go to (5).
(16) All paths have been assigned and all nodes have reported back, so STOP.

FOR EACH NODE:

(1) RECEIVE initializations and a starting point from the host.
(2) If the message in (1) is incomplete, go to (1).
(3) Track the path associated with the starting point.
(4) SEND a "ready to transmit solution" message to the host.
(5) RECEIVE a "ready to receive" message from the host.
(6) SEND the "solution" message to the host.
(7) If the message in (6) is incomplete, go to (6).
(8) Go to (1).

Note 1: The initialization and solution messages may be longer than permitted by the message buffer. The information must therefore be passed in multiple messages.

Note 2: Each node program is in an infinite.

RECEIVE initializations → track homotopy zero curve → SEND solution

loop.

5. COMPUTATIONAL RESULTS

Some computational results are shown in Table 1 for real industrial engineering problems that have arisen at General Motors and elsewhere in chemical equilibrium computations, solid modelling, and robotics. The problem number refers to an internal numbering scheme used at General Motors Research Laboratories; problem data is available on request. An example of a solid modelling problem (403 in Table 1) is

$$F_j(x) = a_{j1}x_1^2 + a_{j2}x_2^2 + a_{j3}x_1x_2 + a_{j4}x_1 + a_{j5}x_2 + a_{j6} = 0, \qquad \text{for } j = 1, 2,$$

where

$$a_{11} = -0.00098 \quad a_{14} = -235 \quad a_{21} = -0.01 \quad a_{24} = 0.00987$$
$$a_{12} = 978000 \quad a_{15} = 88900 \quad a_{22} = -0.984 \quad a_{25} = -0.124$$
$$a_{13} = -9.8 \quad a_{16} = -1.0 \quad a_{23} = -29.7 \quad a_{26} = -0.25.$$

The exact solutions (to four significant figures) are

$$(x_1, x_2) = (0.09089, -0.09115),$$
$$(2342, -0.7883),$$
$$(0.01615 + 1.685i, 0.0002680 + 0.004428i),$$
$$(0.01615 - 1.685i, 0.0002680 - 0.004428i).$$

Table 1. Execution time (s)

Problem number	Total degree	80286/80287	iPSC-32	ω	VAX 11/750	VAX 11/780	IBM 3090/200	SUN-2 /50	SUN-3 /160
102	256	16257	645	0.76	2438	1248	77	10545	8041
103	625	34692	1616	0.65	5260	2656	163	22634	17126
402	4	255	54	0.94	41	8	2	158	111
403	4	84	19	0.88	14	6	1	54	38
405	64	3669	335	0.33	703	334	14	2958	2429
601	60	9450	257	1.11	1707	796	78	7417	5903
602	60	28783	2795	0.31	4332	2054	124	21897	16831
603	12	1200	243	0.38	325	152	9	1339	1060
803	256	-	11527	-	29779	16221	667	130311	77113
1702	16	1655	163	0.60	216	112	7	986	658
1703	16	1657	162	0.60	216	112	7	984	658
1704	16	1628	108	0.89	216	112	6	1005	667
1705	81	14336	378	1.15	1884	999	55	8907	6313
5001	576	-	11786	-	49736	27815	1997	-	237685

The serial execution time shown in Table 1 is the total real elapsed time, including all I/O and the program load. The execution time for the iPSC is real elapsed time including all I/O and the host program load, but not the node program load. All the runs (except for the IBM 3090/200) were made on dedicated systems. Real elapsed time was measured by the UNIX time command on the UNIX systems, and by job statistics on the IBM and VAXen.

Several interesting conclusions can be drawn from Table 1. For medium sized polynomial systems (with total degree in the hundreds), the iPSC-32 parallel machine compares favorably with larger serial machines, and is also more cost effective for such problems. There is considerable variation within this problem set; this is evident from comparing the ratios of the execution times for just the serial machines. There is also sometimes considerable variation among the zero curves for a problem (e.g., problems 405, 602, and 603), resulting in a severe load imbalance. What happens is that several paths are especially hard to track, and the entire system waits (with most nodes idle) while these (relatively slow) nodes finish off their paths. Even so, a 31% efficiency rating (problem 602) is still respectable, and the iPSC hypercube is still cost effective even for problem 602.

The efficiencies ω range from 0.31 to 1.15; an efficiency $\omega > 1$ is theoretically impossible. This is a good illustration of the difficulty of using statistics such as speedup and efficiency to measure performance. The host is memory poor compared to the aggregate memory of 32 nodes, and for larger polynomial systems memory does make a difference. The point is that it is unfair to compare a memory starved serial time (on the host or on one node) to a memory abundant parallel time. Another contributing factor to this $\omega > 1$ paradox is the significant difference between the node and host operating systems. Note also that the total degree d (which is a direct measure of the potential parallelism) is not a particularly predictive statistic, because problems 601 and 602 both have $d = 60$ but have efficiencies $\omega = 1.11$ and $\omega = 0.31$, respectively.

It is worthwhile to comment here on some realities of parallel computation. The parallel computation scheme of assigning a path to each node seems completely transparent and straightforward. Why this is not so is illustrative of the difficulty of the whole field of parallel computation. The following is a list of tasks that should be trivial, with comments as to why they were not.

1. *Assigning data to the nodes*

There are exactly d paths and starting points, completely determined by the parameters in the homotopy map ρ_c. Once ρ_c is chosen, these starting points are fixed, and may not be changed in any way. In production use of HOMPACK, not every solution may be desired, so only some of the paths may need to be tracked. Thus the parallel computer code has to keep track of all the paths, which of those are being tracked and have been assigned to a node, and which are to be tracked but have not yet been assigned to a node. This is further complicated by the fact that the number of paths does not equal (in general) the number of nodes. Also the nodes are finishing paths at random (thus becoming available to track another path), and so must be reused in random order. Now all this bookkeeping can, of course, be done, but the point is that these considerations do not even arise for the serial program.

2. *Transmitting data to the nodes*

In the serial version of HOMPACK, all work arrays (below the level of the main program) have variable dimensions, and there are absolutely no static arrays limiting the size of the problem. Current hardware implementations of the hypercube architecture have nodes with relatively small memories (128K or 512K), and the nodes do not support virtual memory. Furthermore, what executes on the nodes is a "main program", not a "subroutine". Thus memory is tight, and arrays cannot be dynamically allocated. Since the actual storage needed depends on several factors—dimension, total degree, number of terms, etc.—it is difficult to

optimally fit a problem into the available memory. Another annoyance is that only short messages may be sent (on the NCUBE the node program itself must be transmitted by the host program), so the data defining the problem must be broken up and transmitted in pieces. All this communication sending, receiving, queueing, blocking, and unblocking is currently (and unfortunately) the responsibility of the application programs.

3. *Transmitting answers to the host*

To appreciate the situation here, consider the following three facts: (1) the complete answer (point $(1, \bar{x})$ at end of path and concomitant path statistics) cannot be sent all in one message; (2) the nodes finish tracking their assigned paths asynchronously; (3) for hardware and software reasons, the host cannot just "listen" to one mode. No matter how the host resolves this situation, it is going to be nontrivial.

One possibility is to maintain a huge data structure, and as the answer pieces arrive, insert them in the correct slot in the structure. This requires an identification stamp with each message, and some blocking and unblocking overhead at the ends. A second possibility is to establish a "handshake" protocol, whereby each node simply informs the host that its path is complete, and does not transmit its solution until the host requests it. Even in this case an answer reception may be punctuated by completion notices from other nodes, which must be queued. This second approach (outlined by the pseudo code in Section 4) was the one used for the times reported in Table 1. There are many other reasonable approaches to this issue; more research is needed.

4. *Replicating performance measurements*

Message transmission is a combination of hardware and software, and may involve several (intermediate) nodes between the sender and recipient. All this happens asynchronously in real time, and the temporal order of events may depend on such things as buffer status, free space list size, timer interrupts, and even random (corrected) hardware errors. The state of the node operating systems and disk file fragmentation on the host can affect durations and the temporal order of events, sometimes by as much as 10%. Obtaining performance data is difficult, and for complex, realistic codes, replicating performance results may be impossible.

These problems were also attempted on the NCUBE, but as of this writing the NCUBE f77 compiler would not correctly compile HOMPACK, which is written in ANSI standard FORTRAN 77.

REFERENCES

1. Allgower E. & Georg K., Simplicial and continuation methods for approximating fixed points, *SIAM Rev.* **22**, 28–85 (1980).
2. Billups S. C., An augmented Jacobian matrix algorithm for tracking homotopy zero curves, M.S. Thesis, Dept. of Computer Sci., VPI & SU, Blacksburg, VA (1985).
3. Businger P. & Golub G. H., Linear least squares solutions by Householder transformations, *Num. Math.* **7**, 269–276 (1965).
4. Chen A. C. & Wu C. L., Optimum solution to dense linear systems of equations, *Proc.* 1984 *Int. Conf. Parallel Process*, 417–425 (1984).
5. Chern M. Y. & Murata T., Fast algorithm for concurrent LU decomposition and matrix inversion, *Proc. Int. Conf. Parallel Process*, Computer Society Press, Los Alamitos, CA, 79–86 (1983).
6. Chow S. N., Mallet-Paret J. & Yorke J. A., Finding zeros of maps: Homotopy methods that are constructive with probability one, *Math. Comput.* **32**, 887–899 (1978).

7. CLOETE E. & JOUBERT G. R., Direct methods for solving systems of linear equations on a parallel processor, *Proc. 8th S. Afr. Symp. Num. Math.*, Durban, South Africa (1982).
8. COSNARD M., ROBERT Y. & TRYSTRAN D., Comparison of parallel diagonalization methods for solving dense linear systems, *Sess. French Acad. Sci. Math.* **781**, (1985).
9. ELLIS G. H. & WATSON L. T., A parallel algorithm for simple roots of polynomials, *Comput. Math. Applic.* **10**, 107–121 (1984).
10. GAJESKI D. D., SAMEH A. H. & WISNIEWSKI J. A., Iterative algorithms for tridiagonal matrices on a WSI-multiprocessor, *Proc. Int. Conf. Parallel Process.*, Bellaire, MI, 82–89 (1982).
11. GENTZSCH W. & SCHAFTER G., Solution of large linear systems on vector computers, *Parallel Comput.* **83**, 159–166 (1984).
12. HELLER, D., A survey of parallel algorithms in numerical linear algebra, *SIAM Rev.* **20**, 740–777 (1978).
13. INTEL CORPORATION, iPSC Users' Manual (1985).
14. KANEDA Y. & KOHATA M., Highly parallel computing of linear equations on the matrix-broadcast-memory connected array processor system, *10th IMACS Wd Cong.* **1-5**, 320–322 (1982).
15. KOWALIK J. S., Parallel computation of linear recurrences and tridiagonal equations, *Proc. IEEE* 1982 *Int. Conf. Cybernetics Soc.* 580–584 (1982).
16. KOWALIK J. S. & KUMAR S. P., An efficient parallel block conjugate gradient method for linear equations, *Proc. Intrn. Conf. Parallel Proces.*, Bellaire, MI, 47–52 (1982).
17. KUBICEK M., Dependence of solutions of nonlinear systems on a parameter, *ACM Trans. Math. Software* **2**, 98–107 (1976).
18. LAKSHMIVARAHAN S. & DHALL S. K., Parallel algorithms for solving certain classes of linear recurrences, Foundations of software technology and theoretical computer Science, *Lecture Notes in Computer Science* **206**, Springer, Berlin, 457–477 (1985).
19. MORGAN A. P., A transformation to avoid solutions at infinity for polynomial systems, *Appl. Math. Comput.* **18**, 77–86 (1986).
20. MORGAN A. P., A homotopy for solving polynomial systems, *Appl. Math. Comput.* **18**, 87–92 (1986).
21. MORGAN A. P., *Solving Polynomial Systems Using Continuation for Engineering and Scientific Problems*. Prentice-Hall, Englewood Cliffs, NJ (1987).
22. MORGAN A. P. & WATSON L. T., A globally convergent parallel algorithm for zeros of polynomial systems, Tech. Report TR-86-25, Dept. of Computer Sci., VPI & SU, Blacksburg, VA (1986).
23. PARKINSON D., The solutions of N linear equations using P processors, *Parallel Comput.* **83**, 81–87 (1984).
24. REED D. A. & PATRICK M. L., A model of asynchronous iterative algorithms for solving large sparse linear systems, *Proc.* 1984 *Int. Conf. Parallel Process.*, 402–410 (1984).
25. REED D. A. & GRUNWALD D. C., The performance of multicomputer interconnection networks, *IEEE Comput.* **20**, 63–73 (1987).
26. RHEINBOLDT W. C. & BURKARDT J. V., Algorithm 596: a program for a locally parameterized continuation process, *ACM Trans. Math. Software* **9**, 236–241 (1983).
27. RICE T. A. & SIEGEL L. J., A parallel algorithm for finding the roots of a polynomial, *Proc. Int. Conf. Parallel Process.*, Bellaire, MI, 57–61 (1982).
28. SCHWANDT H., Newton-like interval methods for large nonlinear systems of equations on vector computers, *Comput. Phys. Communs* **37**, 223–232 (1985).
29. SEITZ C. L., The cosmic cube, *Communs ACM* **28**, 22–33 (1985).
30. SHAMPINE L. F. & GORDON M. K., *Computer Solution of Ordinary Differential Equations: The Initial Value Problem*. Freeman, San Francisco (1975).
31. SIPS H. J., A parallel processor for nonlinear recurrence systems, *Proc. 1st Int. Conf. Supercomput. Syst.*, IEEE Computer Society Press, Los Alamitos, CA, 660–671 (1984).
32. WATSON L. T. & FENNER D., Chow-Yorke algorithm for fixed points or zeros of C^2 maps, *ACM Trans. Math. Software* **6**, 252–260 (1980).
33. WATSON L. T., A globally convergent algorithm for computing fixed points of C^2 maps, *Appl. Math. Comput.* **5**, 297–311 (1979).
34. WATSON L. T., BILLUPS S. C. & MORGAN A. P., HOMPACK: a suite of codes for globally convergent homotopy algorithms, Tech. Report **85-34**, Dept. of Industrial and Operations Eng., Univ. of Michigan, Ann Arbor, MI (1985); also *ACM Trans. Math. Software* **13**, 281–310 (1987).
35. WATSON L. T., Numerical linear algebra aspects of globally convergent homotopy methods, Tech. Report TR-85-14, Dept. of Computer Sci., VPI&SU, Blacksburg, VA (1985).
36. WHITE R., Parallel algorithms for nonlinear problems, *SIAM J. Algebraic Discrete Meth.* **7**, 137–149 (1986).
37. WHITE R., A nonlinear parallel algorithm with application to the Stefan problem, *SIAM J. Num. Analysis* **23**, 639–652 (1986).