Parallel Computing 10 (1989) 161–176
North-Holland

Message length effects for solving polynomial systems on a hypercube

Wolfgang PELZ

Department of Mathematical Sciences, University of Akron, Akron, OH 44325, U.S.A.

Layne T. WATSON *

Departments of Electrical Engineering and Computer Science, Industrial and Operations Engineering, and Mathematics, The University of Michigan, Ann Arbor, MI 48109, U.S.A.

Received September 1986
Revised January 1988

Abstract. 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. Recently a homotopy algorithm has been proposed for polynomial systems 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. For this homotopy algorithm and a given decomposition strategy, the communication overhead for several possible communication strategies is explored empirically in this paper. The experiments were conducted on an iPSC-32 hypercube.

Keywords. Polynomial systems of equations, globally convergent homotopy algorithm, parallel implementation on iPSC-32 hypercube, decomposition strategy, communication overhead.

1. Introduction

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, 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

* Current address: Department of Computer Science, Virginia Polytechnic Institute & State University, Blacksburg, VA 24061, U.S.A.

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 homotopy algorithms are often not fully appreciated, perhaps because globally convergent probability-one homotopy methods are not widely known.

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. The overall purpose of this research is to study how nonlinear systems of equations might be solved on a hypercube; this paper addresses a small part of that issue, namely probability-one homotopy methods for polynomial systems.

Much work has been done on solving linear systems of equations on parallel computers, mostly on vector machines [4,5,7,8,10–12,14–16,18,22,23]. Some work has been done on nonlinear equations and Newton's method [26,29,34,35], and on finding the roots of a single polynomial equation [9,25]. 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. 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. For a given decomposition of a globally convergent probability-one homotopy algorithm for polynomial systems of equations, the present article explores empirically the magnitude of the communication overhead for several possible communication strategies. The test problems were real engineering problems obtained from General Motors Research Laboratories.

Section 2 summarizes the mathematics behind the homotopy algorithm, and sketches a computer implementation based on ODE techniques. Section 3 briefly describes the 'hypercube' computer architecture. Section 4 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. 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 [6,19–21,30–33]) is

Proposition 2.1. *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 $D\rho$ 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 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

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

which 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 2.1 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; refer to the excellent survey [1] and recent work [32,33]. The numerical results here were obtained with preliminary versions of HOMPACK [32], a software package currently under development at Sandia National Laboratories, General Motors Research Laboratories, Virginia Polytechnic Institute and State University, and The University of Michigan. There are three primary algorithmic approaches to tracking γ:

(1) an ODE-based algorithm,

(2) a predictor-corrector algorithm whose corrector follows the flow normal to the Davidenko flow (a 'normal flow' algorithm);

(3) a version of Rheinboldt's linear predictor, quasi-Newton corrector algorithm [24] (an 'augmented Jacobian matrix' method).

Only the ODE-based algorithm will be discussed here. Alternatives (2) and (3) are described in detail in [33] and [2], respectively. Assuming that $F(x)$ is C^2 and a is such that Proposition 2.1. 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 [28,31].

Typical ODE software requires $(d\lambda/ds, dx/ds)$ explicitly, and (3), (4) only implicitly define the derivative $(d\lambda/ds, dx/ds)$. (It might be possible to use an implicit ODE technique for (3)–(4), but that seems less efficient than the method proposed here.) The derivative $(d\lambda/ds, dx/ds)$, which is a unit tangent vector to the zero curve γ, can be calculated by finding the one-dimensional kernel of the $p \times (p+1)$ Jacobian matrix

$$D\rho_a(\lambda(s), x(s)),$$

which has full rank by Proposition 2.1. It is here that a substantial amount of computation is incurred, and it is imperative that the number of derivative evaluations be kept small. Once the kernel has been calculated, the derivative $(d\lambda/ds, dx/ds)$ is uniquely determined by (4) and continuity. Complete details for solving the initial value problem (3)–(5) and obtaining $x(\bar{s})$ are in [30] and [31]. A discussion of the kernel computation follows.

The Jacobian matrix $D\rho_a$ is $p \times (p+1)$ with (theoretical) rank p. The crucial observation is that the last p columns of $D\rho_a$, corresponding to $D_x\rho_a$, may not have rank p, and even if they do, some other p columns may be better conditioned. The objective is to avoid choosing p 'distinguished' columns, rather to treat all columns the same (not possible for sparse matrices). There are kernel finding algorithms based on Gaussian elimination and p distinguished columns [17]. Choosing and switching these p columns is tricky, and based on *ad hoc* parameters. Also, computational experience has shown that accurate tangent vectors $(d\lambda/ds, dx/ds)$ are essential, and the accuracy of Gaussian elimination may not be good enough. A conceptually elegant, as well as accurate, algorithm is to compute the QR factorization with column interchanges [3] of $D\rho_a$,

$$QD\rho_a P^{\mathrm{T}} Pz = \begin{pmatrix} * & \cdots & * & * \\ & \ddots & \vdots & \vdots \\ 0 & & * & * \end{pmatrix} Pz = 0,$$

where Q is a product of Householder reflections and P is a permutation matrix, and then obtain a vector $z \in \ker D\rho_a$ by back substitution. Setting $(Pz)_{p+1} = 1$ is a convenient choice. This scheme provides high accuracy, numerical stability, and a uniform treatment of all $p+1$ columns. Finally,

$$\left(\frac{d\lambda}{ds}, \frac{dx}{ds}\right) = \pm \frac{z}{\|z\|_2},$$

where the sign is chosen to maintain an acute angle with the previous tangent vector on γ. There is a rigorous mathematical criterion, based on a $(p+1) \times (p-1)$ determinant, for choosing the sign, but there is no reason to believe that would be more robust than the angle criterion.

Several features which are a combination of common sense and computational experience should be incorporated into the algorithm. Since most ordinary differential equation solvers only control the local error, the longer the arc length of the zero curve γ gets, the farther away the computed points may be from the true curve γ. Therefore when the arc length gets too long, 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 $\rho_a(\lambda, x)$ is followed starting from $(\bar{\lambda}, \bar{x})$. A rigorous justification for this strategy was given in [31]. If ρ_a has the special form in (1), then trivially

$$\bar{a} = (\bar{\lambda} F(\bar{x}) + (1 - \bar{\lambda})\bar{x})/(1 - \bar{\lambda}).$$

For more general homotopy maps ρ_a, this computation of $\bar{a}$ may be complicated.

Remember that tracking γ was merely a means to an end, namely a zero $\tilde{x}$ of $F(x)$. Since γ itself is of no interest (usually), one should not waste computational effort following it too closely. However, since γ is the only sure way to $\tilde{x}$, losing γ can be disastrous. The tradeoff between computational efficiency and reliability is very delicate, and a fool-proof strategy appears difficult to achieve. None of the three primary algorithms alone is superior overall, and each of the three beats the other two (sometimes by an order of magnitude) on particular problems. Since the algorithms' philosophies are significantly different, a hybrid will be hard to develop.

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 $y_1 \geqslant 1$, 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. The hypercube

The word 'hypercube' refers to an n-dimensional cube. Think of a cube in n dimensions as sitting in the positive orthant, with vertices at the points

$$(v_1, \ldots, v_n), \quad v_i \in \{0,1\}, \quad i = 1, \ldots, n.$$

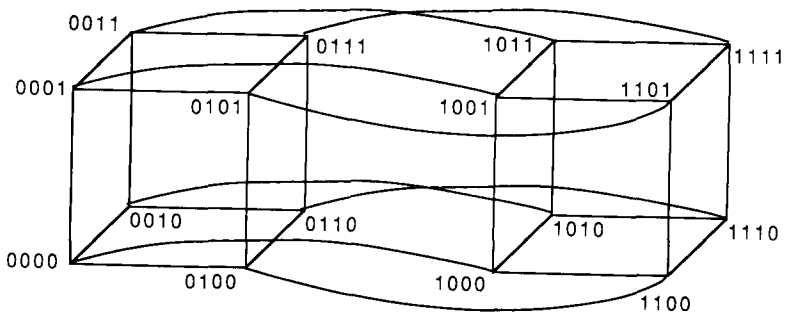

Fig. 1. 4-cube structure and node labelling.

There are thus 2^n vertices, and two vertices v and w are 'adjacent', i.e., connected by an edge, if and only if $v_i = w_i$ for all i except one. The associated graph, also sometimes referred to as an 'n-cube', has 2^n vertices (which can be labelled as above with binary n-tuples) and edges between vertices whose labels differ in exactly one coordinate (see Fig. 1).

A 'hypercube computer architecture' is a computer system with 2^n (node) processors, corresponding to the 2^n vertices (nodes), and a communication link corresponding to each edge of the n-cube. Thus each processor has a direct communication link to exactly n other processors. The distance between any two of the $P = 2^n$ processors is at most $n = \log_2 P = \log_2(2^n)$, considered an ideal compromise between total connectivity (distance = 1) and ring connectivity (distance = $P/2$). Figure 1 shows how a 4-cube is built up from two 3-cubes.

Typically the node label $(v_1, \ldots, v_n)$ is viewed as a binary number $v_1 v_2 \ldots v_n$, and in this view two nodes are adjacent if and only if their binary representations differ in exactly one bit. Typically node addresses are computed in programs by a gray code, a bijective function

$$g: \{0, \ldots, 2^n - 1\} \to \{0, \ldots, 2^n - 1\}$$

such that the binary representations of $g(k \pmod{2^n})$ and $g(k+1 \pmod{2^n})$ differ in exactly one bit for all k (cf. [13]).

Realizations of this abstract architecture have one additional feature: a 'host' processor with communication links to *all* the node processors. This host typically loads programs into the nodes, starts and stops processes executing in the nodes, and interchanges data with the nodes. In current hardware implementations only the host has external I/O and peripheral storage; the nodes consist of memory, a CPU, and possibly communication and floating-point hardware.

The Intel iPSC has 32, 64, or 128 nodes. Each node is an 80286/80287 with 512 kbytes of memory. The host is also an 80286/80287, but with 4 Mbyte of memory, a floppy disk drive, a hard disk, an Ethernet connection, and Xenix. The nodes have only a minimal monitor for communication and process management.

The NCUBE has up to 1024 nodes in multiples of 64, each with 128k of memory and communication and floating-point hardware. The host is an 80286, running NCUBE's operating system, a primitive version of UNIX. The node chip is NCUBE's own design, with a unique feature being communication hardware.

4. Polynomial systems

Section 2 described a homotopy algorithm for finding a single solution to a general nonlinear system of equations $F(x) = 0$. Proposition 2.1 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

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

The ith component $F_i(x)$ has n_i terms, the a_{ik} are the (real) coefficients, and the degrees d_{ijk} are nonnegative integers. The total 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 solutions. It is possible to define a homotopy so that all geometrically isolated solutions of (7) have at least one associated homotopy path. Generally, (7) will have solutions at infinity, which forces some of the homotopy paths to diverge to infinity as λ approaches 1. However, (7) can be transformed into a new system which, under reasonable hypotheses, can be proven to have no solutions at infinity and thus bounded homotopy paths. Because scaling can be critical to the success of the method, a general scaling algorithm [32] is applied to scale the coefficients and variables in (7) before anything else is done.

Since the homotopy map defined below is complex analytic, the homotopy parameter λ is monotonically increasing as a function of arc length [20]. The existence of an infinite number of solutions or an infinite number of solutions at infinity does not destabilize the method. Some paths will converge to the higher dimensional solution components, and these paths will behave the way paths converging to any singular solution behave. Practical applications usually seek a subset of the solutions, rather than all solutions [19,20]. However, the sort of generic homotopy algorithm considered here must find all solutions and cannot be limited without, in essence, changing it into a heuristic.

Define $G: C^n \to C^n$ by

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

where a_j and b_j are nonzero complex numbers and d_j is the (total) 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 $d = d_1 \cdots d_n$ be the *total degree* of the system. The fundamental homotopy result, proved and discussed at length in [19]–[21], is:

Theorem 4.1. *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.*

A number of distinct homotopies have been proposed for solving polynomial systems. The homotopy map in (9) is from [20]. As with all such homotopies, there will be paths diverging to infinity if $F(x) = 0$ has solutions at infinity. These divergent paths are (at least) a nuisance,

since they require arbitrary stopping criteria. Solutions at infinity can be avoided via the following projective transformation.

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

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

Note that, if $F'(y^0) = 0$, then $F'(\alpha y^0) = 0$ for any complex scalar α. Therefore, 'solutions' of $F'(y) = 0$ are (complex) lines through the origin in C^{n+1}. The set of all lines through the origin in C^{n+1} is called complex projective n-space, denoted CP^n, and is a smooth compact (complex) n-dimensional manifold. The solutions of $F'(y) = 0$ in CP^n are identified with the solutions and solutions at infinity of $F(x) = 0$ as follows. If $L \in CP^n$ is a solution to $F'(y) = 0$ with $y = (y_1, y_2, \ldots, y_{n+1}) \in L$ and $y_{n+1} \neq 0$, then $x = (y_1/y_{n+1}, y_2/y_{n+1}, \ldots, y_n/y_{n+1}) \in C^n$ is a solution to $F(x) = 0$. On the other hand, if $x \in C^n$ is a solution to $F(x) = 0$, then the line through $y = (x, 1)$ is a solution to $F'(y) = 0$ with $y_{n+1} = 1 \neq 0$. The most mathematically satisfying definition of *solutions to* $F(x) = 0$ *at infinity* is simply *solutions to* $F'(y) = 0$ *(in* CP^n*) generated by* y *with* $y_{n+1} = 0$.

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

Theorem 4.2. *There are no more than* d *isolated solutions to* $F'(y) = 0$ *in* CP^n. *If* $F'(y) = 0$ *has only a finite number of solutions in* CP^n, *it has exactly* d *solutions, counting multiplicities.*

Recall that a solution is *isolated* if there is a neighborhood containing that solution and no other solution. The multiplicity of an isolated solution is defined to be the number of solutions that appear in the isolating neighborhood under an arbitrarily small random perturbation of the system coefficients. If the solution is nonsingular (i.e., the system Jacobian matrix is nonsingular at the solution), then it has multiplicity one. Otherwise it has multiplicity greater than one.

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

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

So $F''(y) = 0$ is a system of $n+1$ equations in $n+1$ unknowns, referred to as *the projective transformation of* $F(x) = 0$. Since $u(y)$ is linear, it is easy in practice to replace $F''(y) = 0$ by an equivalent system of n equations in n unknowns. The significance of $F''(y)$ is given by

Theorem 4.3 ([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 of $F''(y) = 0$. 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 projective transformation functions essentially as a scaling transformation. Its effect is to shorten arc lengths and bring solutions closer to the unit sphere. The coefficient and variable scaling is different, in that it directly addresses extreme values in the system coefficients. The two scaling schemes work well together; see [21] and [32].

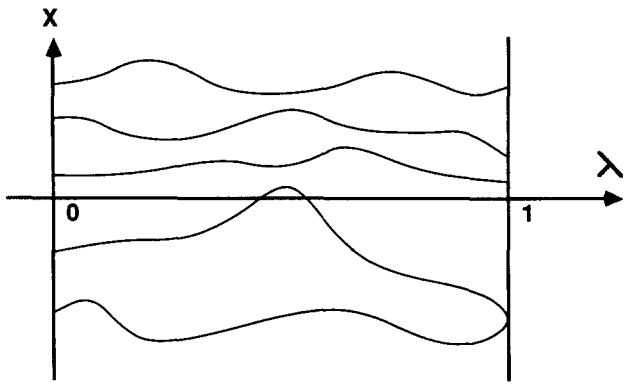

Fig. 2. The set $\rho_a^{-1}(0)$ for a transformed polynomial system.

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. 2. 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.

5. Computational results

Polynomial systems of equations arise frequently in such diverse areas as computational geometry, robotics, chemical engineering, mechanical engineering, and computer vision. A small problem has total degree $d < 100$ and a large problem has $d > 1000$. An example of a chemical equilibrium problem (403 in Table 1) is

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

where

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

The exact solutions (to four significant figures) are

$$\begin{aligned} (x_1, x_2) &= (0.09089, -0.09115), \\ &\quad (2342, -0.7883), \\ &\quad (0.01615 + 1.685\text{i}, 0.0002680 + 0.004428\text{i}), \\ &\quad (0.01615 - 1.685\text{i}, 0.0002680 - 0.004428\text{i}). \end{aligned}$$

Given that d homotopy paths are to be tracked, one method of solution 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 actual path tracking is done with subroutines FIXPNF (on the nodes) and POLSYS (on the host) of HOMPACK [32], ideally modified only by the insertion of SEND's and RECEIVE's. The information passing between the host and the nodes using SEND's and RECEIVE's is a subset of the information that would ordinarily be passed using subroutine CALL's. FIXPNF and POLSYS are complicated production codes, and a detailed discussion of exactly which data must be sent and received, and which data can be inferred or need not be

transmitted, would be a lengthy digression. We therefore precisely describe the parallel algorithm in terms of message passing, without getting mired in the details of what information is being transferred (such details are best left to the code itself, where they are thoroughly documented). Omitting the tracking and initialization details of FIXPNF and POLSYS, 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.

When using subroutines to solve a polynomial system on a serial machine, a large amount of information must be passed back and forth between the calling and the called routines. However, using the message passing mechanism available on the hypercube, it is possible to limit the quantity of data by transmitting it only in the required direction. For example, in the parallel algorithm described above, the initializations need only be passed from the host to the

Table 1
Comparison of runs for full and reduced arrays

Problem number	Total degree	Number of bytes	Number of messages	Execution time (s)	ω	Number of bytes	Number of messages	Execution time (s)	ω
102	256	7536	1536	645	0.76	1080	1536	400	1.23
103	625	7536	3750	1616	0.65	1080	3750	867	1.21
402	4	5232	24	54	0.94	888	24	54	0.94
403	4	5232	24	19	0.88	888	24	19	0.88
405	64	5232	384	335	0.33	888	384	329	0.34
601	60	12144	360	257	1.11	1400	360	247	1.16
602	60	13152	360	2795	0.31	1472	360	2788	0.31
603	12	4704	72	243	0.38	848	72	244	0.38
803	256	105648	2560	11527	–	7256	1536	11436	–
1702	16	12432	96	163	0.60	1416	96	151	0.64
1703	16	12432	96	162	0.60	1416	96	151	0.65
1704	16	14064	96	108	0.89	1528	96	101	0.95
1705	81	14064	486	378	1.15	1528	486	349	1.24
5001	576	61104	4608	11786	–	4440	3456	11610	–

nodes; passing this information in the reverse direction is unnecessary. Similarly, the solution data must be passed only from the nodes to the host.

By reducing the amount of information that is to be transmitted to an absolute minimum (and still be able to solve the problem!), the number of messages and/or the lengths of these messages can be reduced. Thus the time spent on waiting for permission to transmit as well as the time actually spent on transmission can be reduced. In addition the host may not have to wait as long for a solution to be received from a node before continuing processing. The penalty for reducing the information transmitted is that the very complicated data structures of POLSYS must be modified, and the new minimal data structures must be assembled and disassembled for each transmission. Since POLSYS is a sophisticated production code, it is very important to determine whether substantial changes to its data structures are justified; i.e., how deleterious is transmitting some unnecessary data?

Table 1 contains the results of a study designed to examine these effects on an Intel iPSC-32. The problems are all real engineering problems in solid modelling, chemistry, and robotics that have arisen at General Motors and elsewhere. The problem number refers to an internal numbering scheme used at General Motors Research Laboratories; complete problem data is available on request. Total degree refers to the number of paths d to be followed. The next four columns show the number of bytes transmitted (per path), the total number of messages, the execution time, and the efficiency ω for each problem when the full data arrays (unmodified data structures of FIXPNF and POLSYS) are transmitted. The last four columns give the same information for the case in which only the essential information (minimal data structures) is transmitted. The serial times and speedups are not shown, but can be easily calculated from the given data. (*Speedup* is the serial elapsed time using only the host divided by the parallel elapsed time using the hypercube. *Efficiency* is speedup divided by the number of processors actually used.)

The efficiencies ω range from 0.31 to 1.24; 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.

The change in efficiencies between full array and reduced array runs for Problems 102 and 103 also deserves some discussion. The number of paths (total degree) divided into the parallel execution time gives a rough measure of how much time is spent on the 'average' path for each of the problems. Problems 102 and 103 have the lowest ratios of any of the problems indicating that they are less computationally intensive than the others. Another way of saying the same thing is to say that the proportion of time spent by the host on message passing, rather than waiting for the nodes to respond, is higher for these two problems than for the others. Thus any reduction in time spent passing information has a larger effect on overall time. This relatively low time per path for Problems 102 and 103 is consistent with the fact that their paths are innocuous and they have fewer singular solutions at infinity compared to the Problems 602, 803, and 5001, which have the largest times per path. Comparing the two efficiency columns, a reasonable conclusion is that using the minimal data structures is worthwhile only for 'easy' problems, with well-behaved paths and few singular solutions.

The problems given in Table 1 require different times for the computations needed to obtain the solutions irrespective of any message passing time, so direct comparisons among problems are meaningless. However, a comparison of execution times within a particular problem is meaningful since the difference in times is independent of the calculation times and is a function of the difference in the number of messages, the difference in the number of bytes transmitted, and the total degree (the number of paths) in the problem. A model was constructed using multiple regression, resulting in the equation

$$y = -21.0 + 2.28x_1 - 0.11x_2 + 1.89x_3 \tag{12}$$

where y = difference in execution times, x_1 = difference in number of messages, x_2 = difference in total number of kilobytes transmitted, and x_3 = total degree. The value of the multiple correlation coefficient for this model is $R^2 = 0.98$, indicating that 98% of the variation in the data is explainable by the above model. Note that this interpretation only gives an indication of the appropriateness of the model; it does not speak to the question of the true physical model which causes the observed effects. Furthermore, since the independent variables are serially correlated, the regression equation (12) should not be used to predict changes in execution times for changes in one of the variables while keeping the other variables fixed. Figure 3 is the scatter diagram with the variable y on the vertical axis and the variable x_3 on the horizontal axis.

The fit is not perfect for a number of reasons related to the hardware and software of the system in addition to the possibility that the wrong model is being fit. The messages are transmitted 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%. Thus the execution time is a random variable, and statistical analysis is appropriate.

Another study using only the problem set 402 addressed the issue of the effect on execution times as the number of messages increased while keeping both calculation times and the number of transmitted bytes constant. The Intel Hypercube allows a maximum of 16 kbytes to be transmitted in any single message. By artificially restricting the channel bandwidth to values smaller than 16k, the resulting execution times can be related to the number of messages since a decrease in bandwidth per message forces an increase in the number of messages. Figure 4 shows that execution time (vertical axis) is a function of reciprocal bandwidth (horizontal axis).

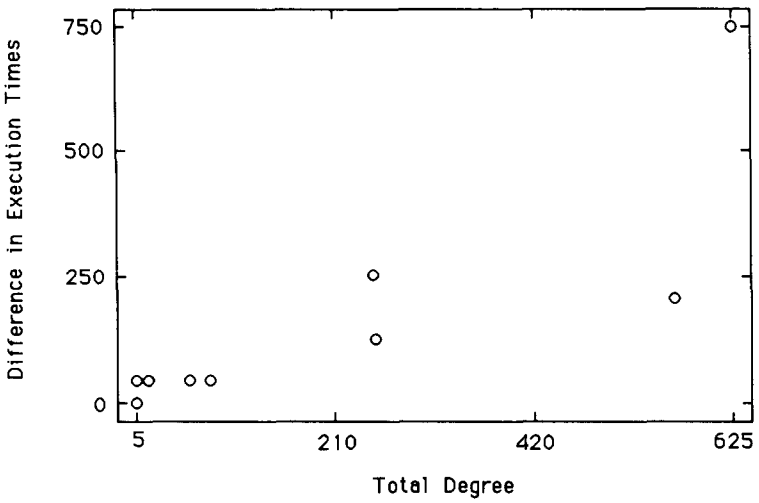

Fig. 3. Difference in execution times versus total degree.

Figure 5 shows that execution time (vertical axis) increases nonlinearly as the number of messages (horizontal axis) increases.

The scatter diagrams show that the execution time is inversely proportional to bandwidth and a function of the number of messages raised to the 3/2 power. A number of models were examined with the most reasonable model being

$$y = 66.12 + 0.00222(x_1)^{3/2} \tag{13}$$

where y = execution time and x_1 = number of messages. The resultant R^2 is 0.99+, indicating an excellent fit to the data. Since the number of messages and the bandwidth are highly correlated, no additional benefit (i.e., gain in information) is obtained by including the latter in the model. The equation (13) is valid within the range of the data, but should not be used to make predictions outside the region of the data.

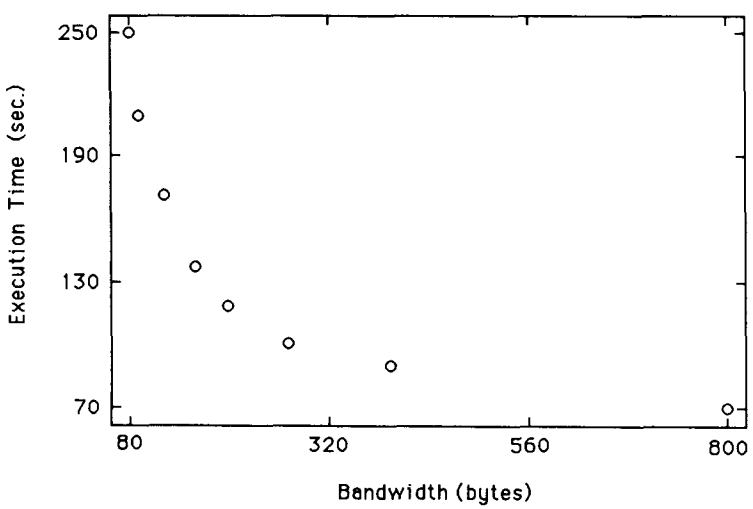

Fig. 4. Execution time versus bandwidth.

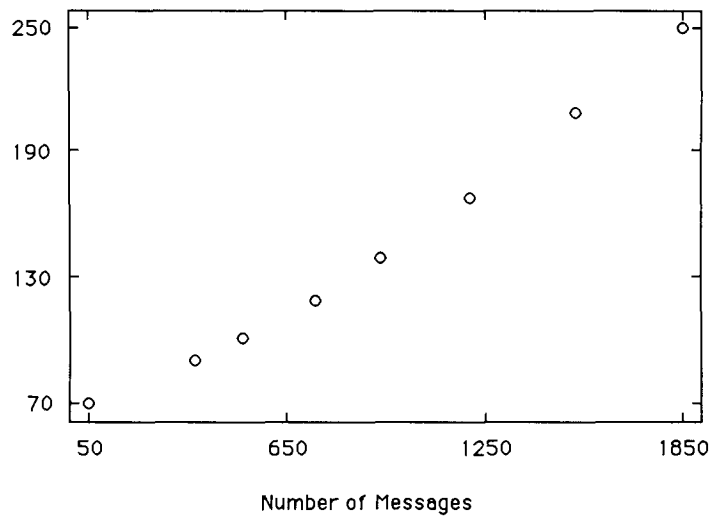

Fig. 5. Execution time versus number of messages.

6. Conclusions

The models in equations (12) and (13) give strong indications as to the form of the underlying physical system. Considering equation (12) and the data in Table 1 first, we may conclude that in general it is unnecessary to minimize the amount of transmitted data. Only for Problems 102 and 103 were large decreases in execution times evident. Although the model (12) showed a good fit overall, predictions for individual cases may not be of much value. Two examples highlight the variability of individual predictions. Problem 103 has an actual difference in execution time of 749 seconds, whereas the model predicts 725 seconds, an error of 24 seconds (3%). However, Problem 601 with an actual difference of 10 seconds has a predicted value of 23 seconds, an error of 13 seconds (130%). Because of the nature of least squares regression fitting, it is entirely possible that the model may predict increases in execution times for cases in which the acutal difference is near zero.

The model given in equation (13) quantifies the relationship seen in Fig. 5. The variation between actual and predicted times is small enough over the range of the data to allow meaningful predictions for cases in which the number of meassages lies between 24 and 2000.

Both statistical studies indicate that in general it is unnecessary to minimize the amount of information being transmitted. Only in those situations where the nodes spend relatively little time on calculations (i.e., the host spends relatively little time waiting for results compared to message passing) is the benefit of time savings on message passing worth the additional effort to redesign the data structures.

In summary: The total degree d, a direct measure of potential parallelism, is not a useful predictive statistic. There are efficiencies $\omega > 1$, illustrating the difficulty of using simple statistics such as speedup and efficiency to measure performance. The regression models (12) and (13) are useful for indicating overall trends. Modifying the mathematical subroutines POLSYS and FIXPNF to get parallel versions for the hypercube that use minimal data structures is worthwhile only for 'easy' problems of moderate to large total degree with well-behaved paths and few singular solutions.

Acknowledgment

The authors are indebted to Dr. Alexander P. Morgan and the Mathematics Department of the General Motors Research Laboratories for providing access to an Intel iPSC-32 hypercube, without which this work would have been impossible. Thanks are also due to the referees whose comments resulted in a substantial improvement in the exposition of this work. Both authors were supported in part by AFOSR Grant 85-0250.

References

[1] E. Allgower and K. Georg, Simplicial and continuation methods for approximating fixed points, *SIAM Rev.* **22** (1980) 28–85.

[2] S.C. Billups, An augmented Jacobian matrix algorithm for tracking homotopy zero curves, M.S. Thesis, Department of Computer Science, VPI&SU, Blacksburg, VA, 1985.

[3] P. Businger and G.H. Golub, Linear least squares solutions by Householder transformations, *Numer. Math.* **7** (1965) 269–276.

[4] A.C. Chen and C.L. Wu, Optimum solution to dense linear systems of equations, *Proc. 1984 International Conference on Parallel Processing* (1984) 417–425.

[5] M.Y. Chern and T. Murata, Fast algorithm for concurrent LU decomposition and matrix inversion, *Proc. International Conference on Parallel Processing* (Computer Society Press, Los Alamitos, CA, 1983) 79–86.

[6] S.N. Chow, J. Mallet-Paret and J.A. Yorke, Finding zeros of maps: Homotopy methods that are constructive with probability one, *Math. Comp.* **32** (1978) 887–899.

[7] E. Cloete and G.R. Joubert, Direct methods for solving systems of linear equations on a parallel processor, *Proc. 8th South African Symposium on Numerical Mathematics*, Durban, South Africa (1982).

[8] M. Cosnard, Y. Robert and D. Trystran, Comparison of parallel diagonalization methods for solving dense linear systems, *Sessions of the French Academy of Science on Mathematics* (1985) 781ff.

[9] G.H. Ellis and L.T. Watson, A parallel algorithm for simple roots of polynomials, *Comput. Math. Appl.* **10** (1984) 107–121.

[10] D.D. Gajski, A.H. Sameh and J.A. Wisniewski, Iterative algorithms for tridiagonal matrices on a WSI-multiprocessor, *Proc. International Conference on Parallel Processing*, Bellaire, MI (1982) 82–89.

[11] W. Gentzsch and G. Schafer, Solution of large linear systems on vector computers, in: M. Feilmeier et al., eds., *Parallel Computing 83* (North-Holland, Amsterdam, 1984) 159–166.

[12] D. Heller, A survey of parallel algorithms in numerical linear algebra, *SIAM Rev.* **20** (1978) 740–777.

[13] Intel Corporation, iPSC Users' Manual, 1985.

[14] Y. Kaneda and M. Kohata, Highly parallel computing of linear equations on the matrix-broadcast-memory connected array processor system, *Proc. 10th IMACS World Congress*, Vols. 1–5 (1982) 320–322.

[15] J.S. Kowalik, Parallel computation of linear recurrences and tridiagonal equations, *Proc. IEEE 1982 International Conference on Cybernetics and Society* (1982) 580–584.

[16] J.S. Kowalik and S.P. Kumar, An efficient parallel block conjugate gradient method for linear equations, *Proc. International Conference on Parallel Processing*, Bellaire, MI (1982) 47–52.

[17] M. Kubicek, Dependence of solutions of nonlinear systems on a parameter, *ACM Trans. Math. Software* **2** (1976) 98–107.

[18] S. Lakshmivarahan and S.K. Dhall, Parallel algorithms for solving certain classes of linear recurrences, in: *Foundations of Software Technology and Theoretical Computer Science*, Lecture Notes in Computer Science **206** (Springer, Berlin, 1985) 457–477.

[19] A.P. Morgan, A transformation to avoid solutions at infinity for polynomial systems, *Appl. Math. Comput.* **18** (1986) 77–86.

[20] A.P. Morgan, A homotopy for solving polynomial systems, *Appl. Math. Comput.* **18** (1986) 87–92.

[21] A.P. Morgan, *Solving Polynomial Systems using Continuation for Engineering and Scientific Problems* (Prentice-Hall, Englewood Cliffs, NJ, 1987).

[22] D. Parkinson, The solution of N linear equations using P processors, in: M. Feilmeier et al., eds., *Parallel Computing 83* (North-Holland, Amsterdam, 1984) 81–87.

[23] D.A. Reed and M.L. Patrick, A model of asynchronous iterative algorithms for solving large sparse linear systems, *Proc. 1984 International Conference on Parallel Processing* (1984) 402–410.

[24] W.C. Rheinboldt and J.V. Burkardt, Algorithm 596: A program for a locally parameterized continuation process, *ACM Trans. Math. Software* **9** (1983) 236–241.

[25] T.A. Rice and L.J. Siegel, A parallel algorithm for finding the roots of a polynomial, *Proc. International Conference on Parallel Processing*, Bellaire, MI (1982) 57–61.
[26] H. Schwandt, Newton-like interval methods for large nonlinear systems of equations on vector computers, *Comput. Phys. Comm.* **37** (1985) 223–232.
[27] C.L. Seitz, The cosmic cube, *Comm. ACM* **28** (1985) 22–33.
[28] L.F. Shampine and M.K. Gordon, *Computer Solution of Ordinary Differential Equations: The Initial Value Problem* (Freeman, San Francisco, 1975).
[29] H.J. Sips, A parallel processor for nonlinear recurrence systems, *Proc. 1st International Conference on Supercomputing Systems* (IEEE Computer Society Press, Los Alamitos, CA, 1984) 660–671.
[30] L.T. Watson and D. Fenner, Chow–Yorke algorithm for fixed points or zeros of C^2 maps, *ACM Trans. Math. Software* **6** (1980) 252–260.
[31] L.T. Watson, A globally convergent algorithm for computing fixed points of C^2 maps, *Appl. Math. Comput.* **5** (1979) 297–311.
[32] L.T. Watson, S.C. Billups and A.P. Morgan, HOMPACK: A suite of codes for globally convergent homotopy algorithms, Technical Report 85-34, Department of Industrial and Operations Engineering, University of Michigan, Ann Arbor, MI, 1985.
[33] L.T. Watson, Numerical linear algebra aspects of globally convergent homotopy methods, Technical Report TR-85-14, Department of Computer Science, VPI&SU, Blacksburg, VA, 1985.
[34] R. White, Parallel algorithms for nonlinear problems, *SIAM J. Algebraic Discrete Methods* **7** (1986) 137–149.
[35] R. White, A nonlinear parallel algorithm with application to the Stefan problem, *SIAM J. Numer. Anal.* **23** (1986) 639–652.