Granularity Issues for Solving Polynomial Systems via Globally Convergent Algorithms on a Hypercube

D.C.S. Allison, Amal Chakraborty, and Layne T. Watson

TR 88-5

Granularity Issues For Solving Polynomial Systems via Globally Convergent Algorithms on a Hypercube

D. C. S. Allison†, Amal Chakraborty† and Layne T. Watson†

Abstract.

Polynomial systems of equations frequently arise in many applications such as 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. There are several ways the homotopy algorithms can be decomposed to run on a hypercube. The granularity of a decomposition has a profound effect on the performance of the algorithm. The results of decompositions with two different granularities are presented. The experiments were conducted on an iPSC-16 hypercube using actual industrial problems.

1. Introduction.

Solving nonlinear systems of equations has enormous significance for science and engineering. A very special case, namely small polynomial systems of equations, occurs frequently in solid modelling, robotics, computer vision, chemical equilibrium computations, chemical process design, and mechanical engineering. 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 considered.

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 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 have not received widespread attention.

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

† Department of Computer Science, Virginia Polytechnic Institute & State University, Blacksburg, VA 24061. This work was supported in part by AFOSR grant 85-0250.

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 topic, namely granularity issues for 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 [6], [7], [9], [10], [14]–[16], [17]–[19], [20], [26], [27], [28]. Some work has been done on nonlinear equations and Newton's method [31], [33], [35], [40], [41], and on finding the roots of a single polynomial equation [12], [30]. Some work has been done in nonlinear optimization on parallel computers [5], [11], [32]. Parallel algorithms for polynomial systems have been studied in [25]. Characteristics of large granularity have been described in [13]. Granularity issues for solving polynomial systems on shared memory machines have been discussed in [2].

Section 2 summarizes the mathematics behind the homotopy algorithm, and sketches a computer implementation based on ODE techniques. 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 two parallel homotopy algorithms for polynomial systems. Computational results on an Intel iPSC-16 hypercube are 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

$$F(x) = 0, \tag{1}$$

The fundamental mathematical result behind the homotopy algorithm (see [8], [22]–[23], [36]–[39]) is

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 $D\rho$ 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 homotopy algorithm consists of following the zero curve γ of ρ_a emanating from $(0,W)$ until a zero $\bar{x}$ of $F(x)$ is reached (at $\lambda = 1$). It is nontrivial to develop a viable numerical

algorithm based on that idea, though, 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

$$\rho_W(\lambda, x) = \lambda F(x) + (1-\lambda)(x - W), \tag{2}$$

which has the same form as a standard continuation or embedding mapping. However, there are 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 $F(x) = 0$. In homotopy methods λ need not increase monotonically along γ and thus turning points present no special difficulty. The way the zero curve γ of ρ_a is followed and the full rank of $D\rho_a$ permit λ to both increase and decrease along γ and guarantee that there are never any "singular points" along γ which afflict standard continuation methods. Also, 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 (2) is a special case) can be tracked by many different techniques; refer to the excellent survey [1] and recent work [38], [39]. There are three primary algorithmic approaches to tracking γ that have been used in HOMPACK [38], a software package developed at Sandia National Laboratories, General Motors Research Laboratories, Virginia Polytechnic Institute and State University, and The University of Michigan: 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 [3], [29], (an "augmented Jacobian matrix" method).

Only the ODE-based algorithm will be discussed here. Alternatives 2) and 3) are described in detail in [38] and [3], respectively. 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

$$\rho_a(\lambda(s), x(s)) = 0$$

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

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

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 [34], [37].

Typical ODE software requires $(d\lambda/ds, dx/ds)$ explicitly, and (3), (4) only implicitly define the derivative $(d\lambda/ds, dx/ds)$. Since the dimension of the kernel of the Jacobian matrix

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

is one (this follows from the fact that $D\rho_a$ has full rank p by Proposition 1), the derivative $(d\lambda/ds, dx, ds)$ can be calculated from any nonzero vector $z \in \ker D\rho_a$. Note that the derivative $(d\lambda/ds, dx/ds)$ is a unit tangent vector to the zero curve γ. For computational efficiency it is imperative that the number of derivative evaluations be kept small. Complete details for solving the initial value problem (3–5) and obtaining $x(\bar{s})$ are in [36] and [37]. 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 [20]. 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 [4] of $D\rho_a$,

$$Q\, D\rho_a\, P^t P z = \begin{pmatrix} * & \cdots & * & * \\ & \ddots & \vdots & \vdots \\ 0 & & * & * \end{pmatrix} Pz = 0,$$

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 $\rho_{\bar{a}}(\lambda, x)$ is followed starting from $(\bar{\lambda}, \bar{x})$. A rigorous justification for this strategy was given in [37]. If ρ_a has the special form in (2), then trivially

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

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 $y_1 \geq 1$, 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

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

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 may have many solutions. It is possible to define a homotopy so that all geometrically isolated solutions of (1) have at least one associated homotopy path. Generally, (1) will have solutions at infinity, which forces some of the homotopy paths to diverge to infinity as λ approaches 1. However, (1) 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 [38] 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 [23]. 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 [22], [23]. 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

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

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 [22]–[23], is:

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.*

A number of distinct homotopies have been proposed for solving polynomial systems. The homotopy map in (9) is from [23]. 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 $F(x)$:

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

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 finite 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 [24]:

Theorem. *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's 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

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

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[24]. *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 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 [22] and [38].

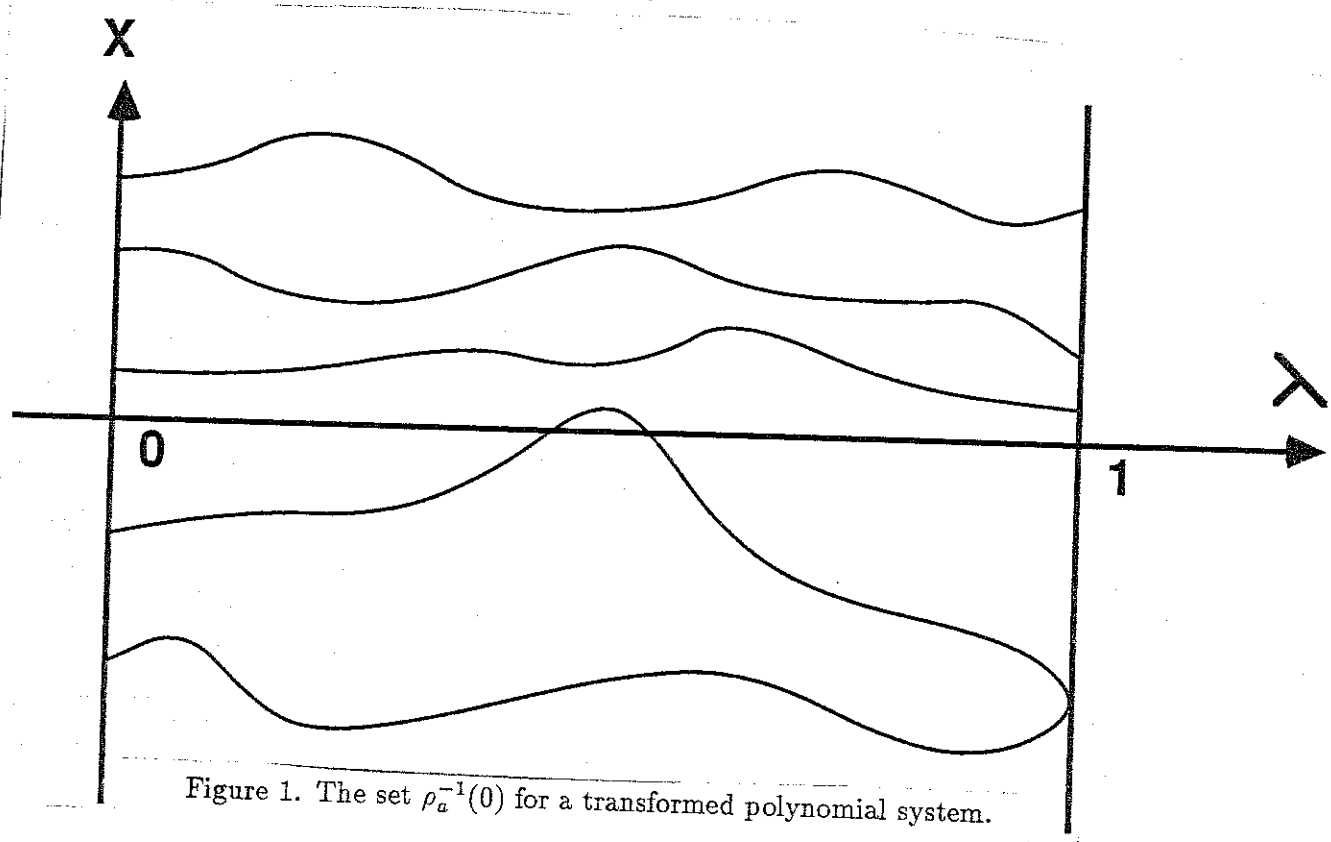

Figure 1. The set $\rho_a^{-1}(0)$ for a transformed polynomial system.

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 Figure 1. There are exactly d (the total degree of F) zero curves, which 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.

4. Parallel algorithms.

Given that d homotopy paths are to be tracked, there are two extreme approaches when executing the homotopy algorithm in parallel. In one extreme, when the granularity is the coarsest possible, each individual processor tracks as many paths as possible until all the solutions for the polynomial system of equations have been found. The host processor reads in the data and initializes parameters (this includes the starting point for each path). It then distributes paths to each node keeping as many nodes as possible busy. When a node finishes tracking one path the host prints the result of that path and assigns a new path to that node. Since there is no *a priori* knowledge about the length of the path, the assignment is made on a first come first serve basis, i.e., paths are assigned in the order they are generated during the initialization process. However, this results in poor performance on those occasions when a few extremely long paths are tracked last. In this case most of the processors will be sitting idle while a few processors will be tracking the long paths. If some knowledge about the length of the paths were available, the paths could be assigned in the decreasing order of their length. This would result in much better load balancing among the nodes. Omitting the tracking and initialization details of the algorithm, the coarse-grained 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. If this is the case the information must be passed in multiple messages.

Note 2: The computation carried out by each node is a loop of the form:

RECEIVE initializations $\rightarrow$ track homotopy zero curve $\rightarrow$ SEND solution.

In the other extreme, where the granularity is the finest possible, the primary task of tracking the solutions is delegated to the host processor and only during the evaluation of the polynomial system and its Jacobian matrix is the work distributed among the nodes. It has been observed that in the serial version of the algorithm about 60% of the execution time is spent in evaluating these values. Thus in the finest granularity version about 60% of the serial algorithm is parallelized. However, one possible advantage of this approach is a better load balancing among the nodes. A high level description of the fine-grained algorithm is:

FOR THE HOST:

(1) SEND initializations and other parameters to all nodes.

(2) Start tracking all the paths.

(3) Continue tracking all the paths. If all paths are completed, then STOP. When a function needs to be evaluated at some point, SEND the location of the point and the index of the next row to the first available node.

(4) If all the rows have been assigned, go to (9).

(5) If all the nodes are busy, go to (7).

(6) Assign next row to next available node. Go to (4).

(7) Wait for a node to send the calculated values.

(8) RECEIVE the desired values from one node and go to (6).

(9) Wait for all the nodes to send their results back to the host.

(10) RECEIVE the desired values from all the nodes and then go to (3).

FOR EACH NODE:

(1) RECEIVE initializations and other parameters.

(2) Wait for host to send a point location and row index.

(3) RECEIVE the location of the point and the row index.

(4) Evaluate the functions and derivatives.

(5) SEND the results to the host and go to (2).

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

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

where

$$\begin{array}{llll} a_{11} = -.00098 & a_{14} = -235 & a_{21} = -.01 & a_{24} = .00987 \\ a_{12} = 978000 & a_{15} = 88900 & a_{22} = -.984 & a_{25} = -.124 \\ a_{13} = -9.8 & a_{16} = -1.0 & a_{23} = -29.7 & a_{26} = -.25 \end{array}$$

The exact solutions (to four significant figures) are

$$\begin{aligned} (x_1, x_2) = &(.09089, -.09115), \\ &(2342, -.7883), \\ &(.01615 + 1.685i, .0002680 + .004428i), \\ &(.01615 - 1.685i, .0002680 - .004428i). \end{aligned}$$

Table 1. Execution time (secs).

Problem number	total degree	Elxsi 6400 (serial)	Elxsi 6400 (coarse)	Elxsi 6400 (fine)	Alliant $FX/8$ (serial)	Alliant $FX/8$ (coarse)	Alliant $FX/8$ (fine)
102(4)	256	508	63	442	362	52	215
103(4)	625	1081	127	936	769	108	457
402(2)	4	10	7	11	6	2	4
403(2)	4	3	3	5	2	1	1
405(2)	64	124	26	107	96	16	55
601(2)	60	392	105	289	245	35	126
602(2)	60	769	135	558	793	148	406
603(2)	12	63	20	64	47	13	32
803(8)	256	6991	759	3045	4459	711	1642
1702(4)	16	50	15	37	36	9	16
1703(4)	16	50	15	37	36	9	16
1704(4)	16	50	11	34	35	7	15
1705(4)	81	426	53	267	308	46	134
5001(8)	576	17449	1829	9051	11579	1765	4736

Table 1 (continued). Execution time (secs).

Problem number	total degree	Encore Multimax	iPSC-16 (serial)	iPSC-16 (coarse)	iPSC-16 (fine)	iPSC-32 (coarse)
102(4)	256	1022	6480	541	11277	645
103(4)	625	2157	13753	1032	23839	1616
402(2)	4	21	108	35	198	54
403(2)	4	5	35	13	66	19
405(2)	64	287	1615	301	11277	335
601(2)	60	836	4045	383	4382	257
602(2)	60	2317	11782	1400	12283	2795
603(2)	12	133	869	195	1559	243
803(8)	256	10428	—	13750	—	11527
1702(4)	16	91	605	180	862	163
1703(4)	16	92	605	180	862	162
1704(4)	16	91	593	151	811	108
1705(4)	81	800	5302	463	5762	378
5001(8)	576	28969	—	14061	—	11786

Table 2. Efficiency: [(serial time)/(parallel time)] / (number of processors used).

Problem number	total degree	Elxsi 6400 (coarse)	Elxsi 6400 (fine)	Alliant FX/8 (coarse)	Alliant FX/8 (fine)	iPSC–16 (coarse)	iPSC–16 (fine)
102(4)	256	.81	.29	.87	.42	.75	.14
103(4)	625	.85	.29	.89	.42	.83	.14
402(2)	4	.36	.40	.72	.83	.77	.27
403(2)	4	.25	.28	.65	.80	.67	.27
405(2)	64	.48	.58	.75	.87	.33	.32
601(2)	60	.37	.68	.88	.97	.66	.31
602(2)	60	.57	.69	.67	.98	.53	.24
603(2)	12	.32	.49	.45	.74	.37	.19
803(8)	256	.92	.29	.78	.34	—	—
1702(4)	16	.33	.34	.48	.54	.21	.18
1703(4)	16	.33	.34	.47	.54	.21	.18
1704(4)	16	.45	.37	.67	.57	.25	.18
1705(4)	81	.80	.40	.84	.58	.73	.23
5001(8)	576	.95	.24	.82	.31	—	—

Tables 1 and 2 contain the results of a study designed to examine the granularity effects on an Intel iPSC hypercube and some other machines. The iPSC-32 was an 80286 based machine, while the iPSC-16 was a newer 80386 based system with special message routing hardware not available in the older system. Although this paper is mainly concerned with the results for the hypercube, the others are included for the sake of completeness. 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. The number in parentheses is the number of equations n. The total degree refers to the number of paths d to be followed.

It can be seen from Tables 1 and 2 that for the Intel iPSC the coarse grained parallel algorithm always outperforms the fine grained algorithm. It is also evident from Table 1 that for the Intel iPSC the performance of the fine grained parallel algorithm is worse than that of the serial version. This is due to the fact that the communication overhead in the fine grained version is greater than the amount of computation done in parallel. The quantum of computation done in each stage in evaluating the polynomial system and its Jacobian matrix is small. From Table 2 it can be seen that the efficiency for the coarse grained algorithm is sometimes rather low. However, increasing the number of processors will almost always (whenever the number of processors is less than the number of paths) speedup the computation substantially. This happens because a relatively small polynomial system can have a reasonably large number of paths. All the paths can be tracked in parallel if a sufficient number of processors are available. This explains why the execution times of both the iPSC-32 and iPSC-16 are almost the same for problems 102, 103, 405, 803, 1705, and 5001 even though each node of the iPSC-32 is about half as fast as each node of the iPSC-16. In general, shared memory machines (Elxsi, Alliant) have many fewer processors than distributed memory machines. Thus for large problems the hypercube has a speedup advantage over a shared memory machine. As the total degree of the polynomial system increases, the efficiency of shared memory machines goes down significantly for the fine grained algorithm, possibly because of more memory contention. This is apparent from the results of problems 803 and 5001 which have efficiencies of .34 and .31 on the Alliant and .29 and .21 on the Elxsi. The serial version and the fine grained version on the hypercube take too long for these two problems and thus were not run.

As stated previously, the percentage of serial execution time that is spent in the evaluation of the polynomial system and its Jacobian matrix ranges from 50%-80%. The percentage depends on the complexity of the polynomial system. As the complexity increases the fraction that can be parallelized increases. This also increases the granule of parallelization and thus the ratio of communication overhead to computation carried out in parallel also decreases. This suggests that for certain classes of polynomial systems (complex function evaluation and large Jacobian matrix), the fine grained version can perform substantially better than the serial version. In this case a mixed strategy can be employed. The coarse-grained algorithm can be used until there are no paths remaining to be tracked. Then the fine-grained algorithm can be used to finish the tracking of the uncompleted paths.

Future work will consider the parallelization of some of the linear algebra subroutines used in the HOMPACK package, and "medium grained" versions of the homotopy algorithm.

6. References.

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

[2] D. C. S. Allison, S. Harimoto, and L. T. Watson, *The Granularity of Parallel Homotopy Algorithms for Polynomial Systems of Equation*, Manuscript.

[3] S. C. Billups, *An augmented Jacobian matrix algorithm for tracking homotopy zero curves*, M.S. Thesis, Dept. of Computer Sci., VPI&SU, Blacksburg, VA, Sept., 1985.

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

[5] R. H. Byrd, R. B. Schnabel, and G. A. Shultz, *Using parallel function evaluation to improve Hessian approximation for unconstrained optimization*, Technical Report CS-CU-361-87, Dept. of Computer Science, University of Colorado, Boulder, Colorado 80309, 1987.

[6] A. C. Chen and C. L. Wu, *Optimum solution to dense linear systems of equations*, Proc. 1984 Internat. Conf. on Parallel Processing, August 21–24, 1984, pp. 417–425.

[7] M. Y. Chern and T. Murata, *Fast algorithm for concurrent LU decomposition and matrix inversion*, Proc. Internat. Conf. on Parallel Processing, Computer Society Press, Los Alamitos, CA, 1983, pp. 79–86.

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

[9] E. Cloète and G. R. Joubert, *Direct methods for solving systems of linear equations on a parallel processor*, Proc. 8th South African Symp. on Numerical Mathematics, Durban, South Africa, July 19–21, 1982.

[10] M. Cosnard, Y. Robert, and D. Trystran, *Comparison of parallel diagonalization methods for solving dense linear systems*, Sessions of the French Acad. of Sci. on Math., Nov., 1985, p. 781ff.

[11] J. E. Dennis Jr., and R. B. Schnabel, *A view of unconstrained optimization*, Tech. Rep. CU-CS-376-87, Dept. of Computer Science, University of Colorado, Boulder, Colorado 80309, 1987.

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

[13] R. A. Finkel, *Large-grain Parallelism – Three case studies*, in The characteristics of parallel algorithms, L. H. Jamieson, D. B. Gannon, and R. J. Douglass, eds., (1987), pp. 21–63.

[14] D. D. Gajski, A. H. Sameh, and J. A. Wisniewski, *Iterative algorithms for tridiagonal matrices on a WSI-multiprocessor*, Proc. Internat. Conf. Parallel Processing, Bellaire, MI, Aug. 24–27, pp. 82–89, 1982.

[15] W. Gentzsch and G. Schafer, *Solution of large linear systems on vector computers*, Parallel Computing 83, North Holland, Amsterdam, 1984, pp. 159–166.

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

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

[18] J. S. Kowalik, *Parallel computation of linear recurrences and tridiagonal equations*, Proc. IEEE 1982 Internat. Conf. on Cybernetics and Society, 1982, pp. 580–584.

[19] J. S. Kowalik and S. P. Kumar, *An efficient parallel block conjugate gradient method for linear equations*, Proc. Internat. Conf. Parallel Processing, Bellaire, MI, Aug. 24–27, pp. 47–52, 1982.

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

[21] S. Lakshmivarahan and S. K. Dhall, *Parallel algorithms for solving certain classes of linear recurrences*, Foundations of Software Technology and Theoretical Computer Science, Lecture Notes in Computer Science, Vol. 206, Springer-Verlag, Berlin, 1985, pp. 457–477.

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

[23] ——, *A homotopy for solving polynomial systems*, Appl. Math. Comput., 18 (1986), pp. 87–92.

[24] ——, *Solving polynomial systems using continuation for engineering and scientific problems*, Prentice-Hall, Englewood Cliffs, NJ, 1987.

[25] A. P. MORGAN AND L. T. WATSON, *A globally convergent parallel algorithm for zeros of polynomial systems*, Tech. Rep. TR-86-25, Dept. of Computer Science, VPI&SU, Blacksburg, VA 24060, 1986.

[26] D. PARKINSON, *The solution of N linear equations using P processors*, Parallel Computing 83, North Holland, Amsterdam, 1984, pp. 81–87.

[27] W. PELZ AND L. T. WATSON, *Message length effects for solving polynomial systems on a hypercube*, Tech. Rep. TR-86-26, Dept. of Computer Science, VPI&SU, VA 24060, 1986.

[28] D. A. REED AND M. L. PATRICK, *A model of asynchronous iterative algorithms for solving large sparse linear systems*, Proc. 1984 Internat. Conf. on Parallel Processing, August 21–24, 1984, pp. 402–410.

[29] W. C. RHEINBOLDT AND J. V. BURKARDT, *Algorithm 596: A program for a locally parameterized continuation process*, ACM Trans. Math. Software, 9 (1983), pp. 236-241.

[30] T. A. RICE AND L. J. SIEGEL, *A parallel algorithm for finding the roots of a polynomial*, Proc. Internat. Conf. Parallel Processing, Bellaire, MI, Aug. 24–27, pp. 57–61, 1982.

[31] R. B. SCHNABEL, AND P. D. FRANK, *Solving systems of nonlinear equations by tensor methods*, Tech. Rep. CU-CS-334-86, Dept. of Computer Science, Univ. of Colorado, Boulder, Colorado 80309, June, 1986.

[32] R. B. SCHNABEL, *Concurrent function evaluations in local and global optimization*, Tech. Rep. CS-CU-345-86, Dept. of Computer Science, Univ. of Colorado, Boulder, Colorado 80309, 1986.

[33] H. SCHWANDT, *Newton-like interval methods for large nonlinear systems of equations on vector computers*, Computer Phys. Comm., 37 (1985), pp. 223–232.

[34] L. F. SHAMPINE AND M. K. GORDON, *Computer Solution of Ordinary Differential Equations: The Initial Value Problem*, W. H. Freeman, San Francisco, 1975.

[35] H. J. SIPS, *A parallel processor for nonlinear recurrence systems*, Proc. 1st Internat. Conf. on Supercomputing Systems, IEEE Computer Society Press, Los Alamitos, CA, 1984, pp. 660–671.

[36] L. T. WATSON AND D. FENNER, *Chow-Yorke algorithm for fixed points or zeros of C^2 maps*, ACM Trans. Math. Software, 6 (1980), pp. 252–260.

[37] L. T. WATSON, *A globally convergent algorithm for computing fixed points of C^2 maps*, Appl. Math. Comput., 5 (1979), pp. 297–311.

[38] L. T. WATSON, S. C. BILLUPS, AND A. P. MORGAN, *HOMPACK: A suite of codes for globally convergent homotopy algorithms*, Tech. Rep. 85–34, Dept. of Industrial and Operations Eng., Univ. of Michigan, Ann Arbor, MI, 1985.

[39] L. T. WATSON, *Numerical linear algebra aspects of globally convergent homotopy methods*, Tech. Rep. TR–85–14, Dept. of Computer Sci., VPI&SU, Blacksburg, VA, 1985.

[40] R. WHITE, *Parallel Algorithms for Nonlinear Problems*, SIAM J. Algebraic Discrete Methods, 7 (1986), pp. 137–149.

[41] R. WHITE, *A Nonlinear Parallel Algorithm with Application to the Stefan Problem*, SIAM J. Numer. Anal., 23 (1986), pp. 639–652.