Applications of Adaptive Data Distributions *

Eric F. Van de Velde
Applied Mathematics 217-50
Caltech
Pasadena, CA 91125

Jens Lorenz
Department of Mathematics and Statistics
The University of New Mexico
Albuquerque, NM 87131

Abstract

Continuation methods compute paths of solutions of nonlinear equations that depend on a parameter. This paper examines some aspects of the multicomputer implementation of such methods. The computation is done on the Symult Series 2010 multicomputer.

One of the main issues in the development of concurrent programs is load balancing, achieved here by using appropriate data distributions. In the continuation process, a large number of linear systems have to be solved. For nearby points along the solution path, the corresponding system matrices are closely related to each other. Therefore, pivots which are good for the LU-decomposition of one matrix are likely to be acceptable for a whole segment of the solution path. This suggests to choose certain data distributions that achieve good load balancing. In addition, if these distributions are used, the resulting code is easily vectorized.

To test this technique, the invariant manifold of a system of two identical nonlinear oscillators is computed as a function of the coupling between them. This invariant manifold is determined by the solution of a system of nonlinear partial differential equations that depends on the coupling parameter. A symmetry in the problem reduces this system to one single equation, which is discretized by finite differences. The solution of this discrete nonlinear system is followed as the coupling parameter is changed.

1 Introduction

Concurrent programming is difficult and needs to be simplified. This simple statement describes a major goal of research into concurrent computing. The focus on simplification is justified, because the accumulated experience of earlier feasibility studies is overwhelmingly positive. These feasibility studies required machine-dependent program and problem reformulation. To raise the concurrent technology from the level of feasible to that of usable, much of current research focuses on simplification of the concurrent-programming task.

At the heart of most efficient concurrent programs is data locality: the data is stored in memory locations "near" the processor using the data. To achieve data locality, a data distribution must be introduced. In general, that is a task the programmer must perform, because the best data distribution is determined by global considerations not accessible to analysis by low-level system components (hardware, operating system, and compiler).

This is illustrated by the following simple example of matrix-vector multiplication. Let A be an $M \times N$ matrix, and $\mathbf{x}$ and $\mathbf{y}$ vectors of dimension N and M, respectively. The assignment:

$$\mathbf{y} := A\mathbf{x} \tag{1}$$

requires the evaluation of a matrix-vector product. If this were a self-contained program, not part of a larger program, the optimal data distribution and corresponding optimal program is easily derived. The rows of the matrix A should be distributed evenly (within divisibility constraints) over all concurrent processes. The resulting program is optimal, because it is perfectly load balanced and it requires no communication. Similarly, for the assignment:

$$\mathbf{z}^T := \mathbf{y}^T A \tag{2}$$

one should distribute the matrix columns. For a composite program that evaluates both assignments (1) and (2) neither distribution is optimal. The best distribution distributes both rows and columns; moreover, the process grid is a function of the ratio of the number of times (1) versus (2) is evaluated. Only the user can have a reasonable estimate of this last quantity; hence, only the user can determine the best distribution. (We have ignored the distribution of the vectors for ease of exposition; the conclusion remains valid if one includes them.)

Supplying the data distribution is thus a user task.

*This research is supported in part by Department of Energy Grant No. DE-AS03-76ER72012. This material is based upon work supported by the NSF under Cooperative Agreement No. CCR-8809615. The government has certain rights in this material.

Considering our goal to simplify concurrent programming, supplying the data distribution should be the *only concurrency-related user task.* Programming languages that allow postponing decisions about data-distribution are under development, see, e.g., Chen [2]. The concept, however, is independent of particular notations or languages, and it can be evaluated within existing concurrent computing systems (although some overheads are to be expected as a result). The program discussed in the remainder of this paper is a realistic illustration of such an approach to concurrent computing, where the data distribution is imposed only after the program was fully developed. In spite of the restriction that all ideas had to be implemented at the software level, instead of at the language or compiler level, excellent performance was obtained. To obtain the best performance, a dynamic data distribution is introduced, which is periodically adapted to achieve global load balance. In this respect, our approach differs significantly from conventional parallelization strategies that break up programs into small "manageable" pieces, typically program loops, and consider each as an independent entity.

An outline of the paper follows. The mathematical aspects of continuation and its application to the computation of invariant manifolds is discussed in section 2. These aspects are covered only to the extent necessary for understanding the algorithmic aspects of the program. For a more detailed treatment, see [12]. In section 3, we discuss the implementation of the program, and in section 4 its performance.

2 Continuation and Invariant Manifolds

Consider a system of M equations:

$$G(\mathbf{u}, \lambda) = 0 \tag{3}$$

for $\mathbf{u} \in \mathbb{R}^M$, which depends on a parameter $\lambda \in \mathbb{R}$. Here,

$$G : \mathbb{R}^M \times \mathbb{R} \longrightarrow \mathbb{R}^M$$

is a smooth map. By a solution branch we mean a one parameter family

$$(\mathbf{u}(s), \lambda(s)) \in \mathbb{R}^M \times \mathbb{R}, \qquad s_a \le s \le s_b \tag{4}$$

of solutions of (3) depending smoothly on some parameter $s \in [s_a, s_b]$. Because of the importance for applications, many numerical methods have been devised and investigated to compute such branches [6,8]. Assuming that the branch (4) contains only regular points and folds, one has to solve linear systems whose matrices have the form:

$$\begin{bmatrix} A & \mathbf{b} \\ \mathbf{c}^T & \delta \end{bmatrix} \in \mathbb{R}^{(M+1)\times(M+1)} \tag{5}$$

where A is $M \times M$. The matrix A is singular at folds; the bordered system, however, is well conditioned.

We use two concurrent solution methods for such bordered systems. Our first method is a variant of Keller's bordering algorithm [7] that takes into account the possible singularity of the matrix A. The second method is a variant of Goovaerts [5]. Here, we consider only the first method, which begins by computing an LU-decomposition of A. Because the matrix A may be singular, partial pivoting is often not sufficient, and a more general pivoting strategy must be used. For simplicity, the only dynamic pivoting strategy considered here is complete pivoting, but other dynamic strategies are easily substituted. Once the LU-decomposition of A is known, the bordered system is solved using slightly modified back-solves and the solution of a 2×2 system.

Numerical and performance results are given for a — rather involved — test problem, namely the numerical calculation of the invariant manifold of a parameter dependent dynamical system:

$$\frac{d\mathbf{v}}{dt} = F(\mathbf{v}, \lambda). \tag{6}$$

Here, $\mathbf{v}(t) \in T^2 \times \mathbb{R}^2$ and F is a mapping from $T^2 \times \mathbb{R}^2 \times [0, \lambda_0]$ into $\mathbb{R}^4$. (With T^2 we denote the standard 2-torus.) The specific example that we have treated is a system of two nonlinear coupled oscillators, where

$$\mathbf{v} = [\theta_0, \theta_1, r_0, r_1]^T$$

and

$$F = \begin{bmatrix} \omega \\ \omega \\ r_0(1 - r_0^2) \\ r_1(1 - r_1^2) \end{bmatrix} + \lambda \begin{bmatrix} -\cos 2\theta_0 + \frac{r_1}{r_0}(\cos(\theta_0 + \theta_1) - \sin(\theta_0 - \theta_1)) \\ -\cos 2\theta_1 + \frac{r_0}{r_1}(\cos(\theta_0 + \theta_1) - \sin(\theta_1 - \theta_0)) \\ r_1(\sin(\theta_0 + \theta_1) + \cos(\theta_0 - \theta_1)) - r_0(1 + \sin 2\theta_0) \\ r_0(\sin(\theta_0 + \theta_1) + \cos(\theta_0 - \theta_1)) - r_1(1 + \sin 2\theta_1) \end{bmatrix}$$

(The value of ω is -0.55 in our calculations.) See Aronson, Doedel, and Othmer [1] for a motivation of this system and for the study of many interesting bifurcation phenomena. Also, see Dieci, Lorenz, and Russel [3] for a sequential calculation of some invariant manifolds.

In the uncoupled case, $\lambda = 0$, the system has the attracting invariant 2-torus

$$\mathcal{M}(\lambda = 0) = \{(\boldsymbol{\theta}, 1, 1) : \boldsymbol{\theta} \in T^2\} \subset T^2 \times \mathbb{R}^2.$$

It follows from general theory (see Fenichel [4] and Sacker [9]) that the torus persists for a sufficiently small coupling constant λ and that it can be parameterized in the form:

$$\mathcal{M}(\lambda) = \{(\boldsymbol{\theta}, R(\boldsymbol{\theta}, \lambda) : \boldsymbol{\theta} \in T^2\},$$

where $\theta \longrightarrow R(\theta, \lambda)$ is a function from $T^2 \longrightarrow \mathbb{R}^2$. This vector function $R(\cdot, \lambda)$ is the solution of a first order system of partial differential equations, which depends on λ. These partial differential equations are discretized, and a symmetry is utilized to obtain a finite dimensional system of the form (3).

From general theory one expects that the tori $\mathcal{M}(\lambda)$ loose more and more derivatives as λ increases. The torus "breaks" in a certain λ-region and disappears. The calculations in [3] show breaking at about $\lambda = 0.2527$. In [12], we compute a solution branch of the discretized system on a 25×25 grid, and we obtain several fold points of this discrete system between $\lambda = 0.2430$ and $\lambda = 0.2448$.

3 Implementation

The concurrent efficiency of the bordering algorithm is determined almost exclusively by the efficiency of the LU-decomposition. The latter, in turn, depends crucially on an interplay between the pivot locations and the distribution of the matrix entries over the concurrent processes. In particular, if the pivots are known in advance, the data distribution can be chosen accordingly, and near ideal load balance can be achieved. In this case, the algorithm is also easily vectorized because all active data remain in contiguous blocks.

Hence, efficiency can be obtained with preset pivots, but numerical stability will, in general, require a different pivoting strategy. In our approach these two requirements are hardly in conflict, because many highly correlated matrices are factored in the course of the continuation procedure. The reasonable belief that the pivot locations can be kept constant along a whole piece of the branch is indeed confirmed by our experience. Therefore, both numerical stability and load balance can be achieved by using a dynamic pivoting strategy occasionally (when the growth factor has exceeded some limit), followed by an adaptation of the data distribution to the new pivot locations.

This data distribution strategy differs from most others in two essential aspects. First, it takes into consideration the global behavior of the program, i.e., the fact that the matrices result from a continuation procedure. Second, adapting the data distribution to the computation itself is an integral part of the strategy. In section 4, we shall see that the combination of these two ingredients leads to high efficiency. Here, we consider the implementation aspects of this strategy.

Because the data distribution is adaptive and depends on the global nature of the continuation program, component routines like the LU-decomposition and the backsolve should be written so that they are correct independently of the data distribution. For such routines, the data distribution is part of the input data supplied in the argument list when calling the routine. We use the LU-decomposition described in [11] and its companion backsolve algorithm. To achieve independence of the data distribution, the LU-decomposition must do all pivoting implicitly (otherwise the data distribution would depend on the pivots!). In fact, all routines called by our program must have the property that they are correct independently of the data distribution. If we consider these routines as the components of a library, the necessity for this property follows from the observation that the writer of the library routines cannot know the global properties of the program in which this routine will be used. Hence, the data distribution cannot be fixed at the time of writing the library. In fact, our LU-decomposition, matrix-vector operations, and other related linear algebra routines are packaged in a data-distribution-independent library. Our continuation program uses this library and imposes a data distribution on it at run-time.

To provide maximum flexibility, our LU-decomposition allows pivoting of both rows and columns. Besides allowing classical pivoting strategies (row, column, diagonal, and complete), this flexibility also leads to two intrinsically concurrent pivoting techniques with increased numerical stability and load balance. For details on those techniques, we refer to [11]. For the discussion of our continuation program we introduce just one dynamic strategy, complete pivoting, and one static strategy, preset pivoting. Complete pivoting is, in general, overkill since numerical stability can be obtained with less expensive pivoting strategies. However, complete pivoting ensures that the pivot locations are highly unpredictable and, hence, illustrates best the adaptivity of our program. Moreover, complete pivoting is used only occasionally, i.e., when the growth factor exceeds a set tolerance. For most LU-decompositions, we use preset pivots, determined by the last LU-decomposition with dynamic pivoting. Hence, the cost of dynamic pivoting is amortized over many LU-decompositions.

4 Performance

The calculations were performed on a Symult Series 2010 multicomputer with up to 64 nodes. We investigate the dependence of the execution time on the data distribution for one LU-decomposition. Here, we used 64 nodes and an 8×8 process grid. As expected, the adapted data distribution turned out to be superior. We consider also, for each fixed strategy, the dependence of the execution time on the number of nodes. We used 2, 4, 8, 16, 32, and 64 nodes, and obtained excellent speedup for each strategy. For absolute performance, we made a comparison of the sequential version of our code with a fully optimized C-version of the LINPACK benchmark [10]. Due

Pivoting	Distrib.	Time(s)	Spdp.	Eff.(%)
Complete	Linear	75.3	41.4	64.7
Complete	Random	63.7	49.9	78.0
Complete	Scatter	62.8	46.2	72.2
Complete	Adapted	51.3	54.2	84.7
Preset	Linear	48.9	36.9	57.7
Preset	Scatter	40.3	42.6	66.6
Preset	Adapted	33.8	48.9	76.4
F. Preset	Adapted	29.7	50.0	78.2

Table 1: LU-Decomposition times for a 25 × 25 grid problem on 64 node Symult Series 2010. Number of megaFLOPS is based on $M^3/3$ floating point operations, where $M = 25^2$ is the number of unknowns.

to memory restrictions, this comparison was done with a random 300 × 300 matrix. A sequential version of our fast preset pivoting algorithm ran about 5% slower than LINPACK. (These 5% result from the fact that we have not implemented a number of low level optimizations used by LINPACK.)

We consider the example of section 2 with $h = 2\pi/25$, i.e., the number of unknowns at every step is $M = 625$. In Table 1, we present timings for one (typical) LU-decomposition using complete pivoting and preset pivoting in combination with different data distributions for the factored matrix. The linear and scatter distributions are static distributions. The linear distribution allocates blocks of contiguous rows and columns to processes. The scatter distribution uses a wrap mapping. The adapted distribution uses the pivot locations of the previous LU-decomposition to distribute the current matrix such that ideal load balance is achieved, if the pivot locations of the current matrix coincide with those of the previous matrix. In the version "Fast Preset" of preset pivoting, certain administrative overhead is eliminated using the information that the pivots are preset and that a particular distribution is used. All calculations were done on a 64 node machine using 64 processes, one process running on each node. The process grid was partitioned into $P = 8$ process rows and $Q = 8$ process columns.

To test the concurrent performance of our code, we determine the execution time as a function of the number of nodes. The same example as in Study 1 is computed successively using 2, 4, 8, 16, 32, and 64 nodes, and always choosing the number of processes equal to the number of nodes, one process running on each node. The numbers P and Q of process rows and columns were chosen equal within divisibility constraints. When the logarithm of the execution time is plotted as a function of the logarithm of the number of processes, ideal speedup is characterized by a straight line with slope -1 if appropriate scales are used. Figure 1 shows that, for each strategy,

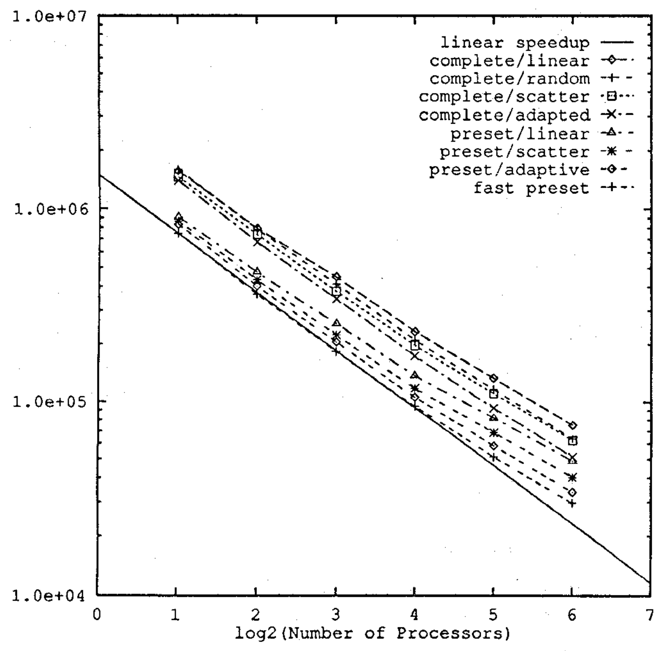

Figure 1: LU-Decomposition times for a 25×25 grid problem as a function of number of nodes on a Symult Series 2010.

the execution-time plot is almost parallel to the line characterizing ideal speedup. Table 1 can be used to identify the individual timing plots.

The problem was too big to run on a one-node machine. Precise speedups could thus not be calculated. In Table 1, we give speedups and efficiencies with respect to two-node timings, i.e., the real speedup is estimated by:

$$S_{PQ} \approx 2 * T_2/T_{PQ},$$

and the real efficiency is estimated by:

$$\epsilon_{PQ} \approx 2 * T_2/(PQT_{PQ}).$$

Here, T_2 is the two-node timing and T_{PQ} is the timing with $P \times Q$ nodes. Speed-up and efficiency are good measures for the overhead due to communication and load imbalance.

When varying the data distribution and keeping the pivoting strategy fixed, it is clear that the adapted data distribution is the most efficient. This is easily explained by the increased load balance of the adapted data distribution. This observation holds for both complete pivoting and preset pivoting.

When comparing efficiencies for the same distribution but for different pivoting strategies (i.e., in Table 1 compare lines 1 and 5, 3 and 6, 4 and 7), it is seen that complete pivoting is more efficient. This is because the pivot-search cost leads to a higher ratio of computation to communication time for complete pivoting than for preset pivoting.

Another interesting observation, which follows from Table 1, is that complete pivoting with the random distribution (line 2) is more efficient than complete pivoting with the scatter distribution (line 3). The execution time, however, is lower for the scatter distribution. The random distribution is better than the scatter distribution for load balancing, and hence, has higher efficiency. The random distribution leads to very irregular memory access patterns, however, and that causes the absolute execution time to be larger.

References

[1] D. G. Aronson, E. J. Doedel, and H. G. Othmer. An analytical and numerical study of the bifurcations in a system of linearly-coupled oscillators. *Physica*, 25D:20–104, 1987.

[2] M. Chen, Y.-I. Choo, and J. Li. Compiling parallel programs by optimizing performance. *The Journal of Supercomputing*, 2:171–207, 1988.

[3] L. Dieci, J. Lorenz, and R. D. Russel. Numerical calculation of invariant tori. 1989. To appear in SIAM Journal on Scientific and Statistical Computing.

[4] N. Fenichel. Persistence and smoothness of invariant manifolds for flows. *Indiana University Mathematics Journal*, 21:193–226, 1971.

[5] W. Goovaerts. *Stable Solvers and Block Elimination for Bordered Singular Systems.* Report, Rijksuniversiteit Gent, Ghent, Belgium, 1989.

[6] H.B. Keller. *Numerical Methods in Bifurcation Problems.* Tata Institute of Fundamental Research, Bombay, 1987.

[7] H.B. Keller. Practical procedures in path following near limit points. In R. Glowinski and J.L. Lions, editors, *Computing Methods in Applied Sciences and Engineering*, North-Holland, 1982.

[8] W.C. Rheinboldt. *Numerical Analysis of Parametrized Nonlinear Equations.* Wiley, New York, NY, 1986.

[9] R. Sacker. A perturbation theorem for invariant manifolds and Hölder continuity. *Journal Mathematical Mechanics*, 18:705–762, 1969.

[10] B. Toy. Private Communication.

[11] E. F. Van de Velde. *Experiments with Multicomputer LU-Decomposition.* Report CRPC-89-1, Center for Research in Parallel Computing, 1989. To appear in Concurrency: Practice and Experience.

[12] E. F. Van de Velde and J. L. Lorenz. *Adaptive Data Distributions for Concurrent Continuation.* Report CRPC-89-4, Center for Research in Parallel Computing, 1989.