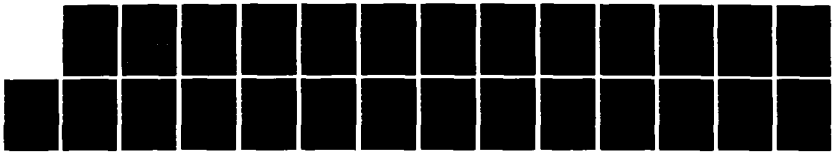
AD-A151 812 PARALLEL COMPUTING IN OPTIMIZATION(U) COLORADO UNIV AT 1/1
BOULDER DEPT OF COMPUTER SCIENCE R B SCHNABEL OCT 84
CU-CS-282-84 ARO-21453.1-MA DAAG29-84-K-0140

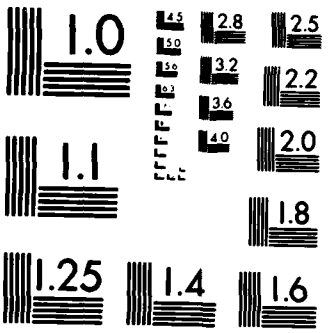
UNCLASSIFIED F/G 9/2 NL



END

FILMED

DTIC



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

ARO 21453.1-MA

(2)

AD-A151 812

PARALLEL COMPUTING IN OPTIMIZATION

Robert B. Schnabel*

Department of Computer Science
University of Colorado
Boulder, Colorado 80309

CU-CS-282-84 October, 1984

DTIC
ELECTE
MAR 20 1985
S D
B

DTIC FILE COPY

*This research supported by ARO contract DAAG 29-84-K-0140

Presented at the NATO Advanced Study Institute on Computational Mathematical Programming, Bad Windsheim, Germany F.R., July 1984

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

85 03 07 216

AD-A151812

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

MASTER COPY - FOR REPRODUCTION PURPOSES

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER ARO 21453.1-MA	2. GOVT ACCESSION NO. N/A	3. RECIPIENT'S CATALOG NUMBER N/A
4. TITLE (and Subtitle) Parallel Computing in Optimization		5. TYPE OF REPORT & PERIOD COVERED Technical Report 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Robert B. Schnabel		8. CONTRACT OR GRANT NUMBER(s) DAAG-29-84-K-0140
9. PERFORMING ORGANIZATION NAME AND ADDRESS University of Colorado Dept. of Computer Science Campus Box 430 Boulder, CO 80309		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS U. S. Army Research Office Post Office Box 12211 Research Triangle Park, NC 27709		12. REPORT DATE October 1984 13. NUMBER OF PAGES 23
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

NA

18. SUPPLEMENTARY NOTES

The view, opinions, and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy, or decision, unless so designated by other documentation.

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Parallel computing, numerical optimization, concurrent, global optimization

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

(attached)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

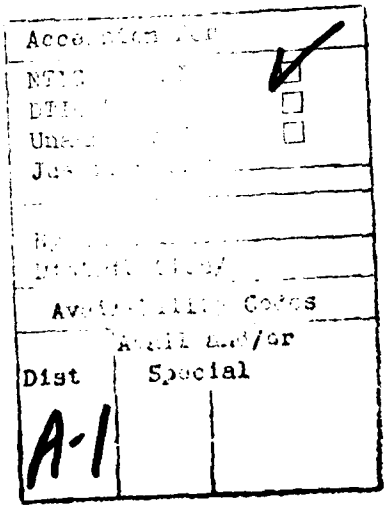
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Abstract

One of the major developments in computing in recent years has been the introduction of a variety of parallel computers, and the development of algorithms that effectively utilize their capabilities. Very little of this parallel algorithm development, however, has been in numerical optimization. Nevertheless, significant opportunities exist for the utilization of parallelism in optimization, especially on computers that support independent concurrent processes. This paper first gives a very brief survey of parallel architectures and general characteristics of parallel algorithms. Next we indicate what we see as the leading opportunities for the utilization of parallelism in optimization. Then we survey the small amount of existing research in parallel optimization; most of this has been conducted at The Hatfield Polytechnic. Finally we discuss some recently initiated research at the University of Colorado concerned with solving optimization problems by parallel algorithms suitable for implementation on a local area network of computers; we focus on a new parallel algorithm for global optimization.

Accession For
NTIS ☑
DTIC ☐
Unannounced ☐
Justification
By
Distribution/
Availability Codes
Avail and/or
Dist Special
A-1

DTIC COPY INSPECTED



1. Introduction

One of the major developments in computing in recent years has been the introduction of a variety of parallel computers, and the development of algorithms that effectively utilize their capabilities. By parallel computer, informally we mean any computer capable of performing two or more computations simultaneously or concurrently. Algorithms that are designed to use simultaneous or concurrent operations will be called "parallel algorithms". In contrast, we will call traditional computers that are designed to execute one instruction at a time "sequential computers", and algorithms designed for such computers "sequential algorithms".

The development of parallel computation is motivated by the same objectives as most progress in computing : the desire to solve an ever-increasing range of problems, and the desire to solve problems as quickly or cheaply as possible. In some cases, the main goal of parallel computation is raw speed, enabling the solution of problems that heretofore were too time-consuming to be solved in a reasonable amount of time, or allowing the solution of problems within the time constraints of real-time systems. In other cases, the main goal of parallel computation is to solve problems more cheaply than with sequential computers.

While in recent years there has been considerable development and implementation of parallel numerical algorithms, very little of this activity has been in numerical optimization. (By numerical optimization we mean the wide category of problems involving the minimization of a linear or nonlinear objective function, perhaps subject to linear or nonlinear constraints. In this paper, we only consider problems in real, as opposed to discrete, variables. For a general description of numerical optimization problems, see for example Gill, Murray, and Wright [1981] or Dennis and Schnabel [1983].) Instead, parallel numerical computation has centered mainly on the solution of partial differential equations, and associated areas such as numerical linear algebra. The reasons for this are simple and clear. First, the most obvious candidates for high speed computation are large scale problems, and these arise far more commonly in the solution of differential equations than in optimization. Secondly, the parallel computers that have been most widely available have been vector computers such as the Cray-1 and Cyber-205, and as we will discuss, these are far better suited to solving partial differential equations and linear algebra problems than optimization problems.

There are many opportunities for the use of parallelism in optimization, however, and good reasons to take advantage of these opportunities. It is well known that many optimization problems are very expensive, for one or more of the following reasons : the size of the problem (number of variables and constraints) is large; the objective function or constraints are expensive to evaluate; many iterations or function

evaluations are required to solve the problem; a large number of variations of the same problem must be solved. Indeed, often optimization problems are not solved, or are only "solved" in a very approximate manner, because the cost of solving them with existing sequential computers and algorithms is prohibitive or does not fit within real-time constraints. The availability of faster or cheaper solutions to optimization problems is likely to cause a considerable increase in the number of optimization problems that are formulated and solved.

The main purpose of this paper is to discuss the prospects for the application of parallel computation to numerical optimization problems, and to survey the small amount of research that has been conducted in this area. While a self-contained treatment of this subject also would require a thorough discussion of parallel computer architectures, it is not within the scope of this paper to discuss this subject in detail. Therefore in Section 2 we provide a thumbnail sketch of the main types of parallel architectures, stressing those characteristics relevant to our discussion of parallel optimization. Section 3 provides a brief overview of important general characteristics of parallel algorithms and their relation to the types of architectures discussed in Section 2. Together, Sections 2 and 3 provide a minimal background in parallel computation; two excellent and thorough references on this subject are the books by Hockney and Jesshope [1981] and Hwang and Briggs [1984].

Sections 4-7 are concerned with parallel optimization. In Section 4 we summarize what we see as the main opportunities for the utilization of parallelism in optimization. Section 5 surveys the small amount of research that has been conducted to date in parallel optimization; much of this work has been conducted at The Hatfield Polytechnic. We and our colleagues at the University of Colorado have recently begun a large project in distributed computation that includes a significant component in parallel optimization; we discuss this work in Section 6. Section 7 contains some concluding remarks on the future of parallel optimization.

2. Brief overview of parallel computer architectures

The development of parallel computer architectures and parallel algorithms are concurrent activities, each motivated partially by the capabilities and needs of the other. This paper mainly discusses parallel algorithms suitable for several broad classes of parallel computer architectures. In this section we provide a minimal description of these classes of parallel computers. Parallel computer architectures

are discussed in many research papers and several books; especially recommended are the books by Hockney and Jesshope [1981] (for pipeline and processor array (SIMD) computers), Hwang and Briggs [1984] (pipeline, processor array, and multiprocessor (MIMD) computers), and Lampson, Paul, and Siegert [1981] (distributed systems).

The four types of parallel architectures we will refer to are pipeline computers (specifically its subclass, vector computers), processor arrays (SIMD computers), multiprocessors (MIMD computers), and local area networks of computers. The terms SIMD and MIMD are from the well-known taxonomy by Flynn [1966] and are explained below; Flynn's taxonomy is not adequate, however, to differentiate the four classes mentioned above.

Pipeline / vector computers

Pipeline computers are machines that overlap computations by subdividing and interleaving them in assembly line fashion. That is, roughly speaking, computer instructions are subdivided into m portions, and the pipeline machine executes part 1 of instruction k, part 2 of instruction k+1, ..., and part m of instruction of instruction k+m-1 simultaneously. Vector computers are a subclass of pipeline machines that use this technique to rapidly compute arithmetic operations on vectors. For example, by dividing the addition instruction into several (typically 6-10) subparts, a vector machine can perform the vector instruction

$$z(i) = x(i) + y(i), \qquad i = 1, \cdots, n$$

rapidly by interleaving the successive additions. Vector computers are intended to perform the above operation, as well as the componentwise subtraction and multiplication of vectors, and the "saxpy" operation

$$z(i) = a * x(i) + y(i), \qquad i = 1, \cdots, n$$

rapidly as long as n is moderately large. Thus they are best suited to large scale problems where these vector computations are dominant, such as solving large systems of linear equations and many algorithms for solving systems of partial differential equations. The primary vector computers in use today include the Cray-1 and Cyber-205. The designs of these machines differ in important ways that effect, for example, the length (n) at which vector processing becomes efficient, but these considerations are beyond the scope of this paper.

Processor arrays (SIMD computers)

Processor arrays are computers capable of executing a given instruction simultaneously on multiple data. Hence in Flynn's taxonomy they are referred to as "single

instruction multiple data" (SIMD) computers. A processor array thus can execute a given program segment simultaneously (in "lockstep") on multiple data sets so long as the program segment contains no data dependent branches. Processor arrays typically consist of hundreds or thousands of processing elements, each with a small amount of local memory, connected to a single control processor from which instructions are issued. Usually an interconnection network permits data to be exchanged between various processing elements; sometimes data is shared between neighboring processors. Examples of processor arrays are the Illiac-IV, the Burroughs Scientific Processor, the Goodyear Massively Parallel Processor, and the ICL Distributed Array Processor (DAP) of Queen Mary College, London.

Multiprocessors (MIMD computers)

Multiprocessors are computers capable of executing multiple instructions on multiple data independently and concurrently. Hence in Flynn's taxonomy they are referred to as "multiple instruction multiple data" (MIMD) computers. Multiprocessors thus are versatile parallel computers that can be used to concurrently perform similar or dissimilar tasks. Typically, multiprocessors consist of a small number (2-50) of processors, usually with their own local memory, which share access to common memory modules and are controlled by a single operating system. Several multiprocessors have been developed in experimental environments, including the C.mmp and Cm* of Carnegie Mellon University, the NEPTUNE of Loughborough University, the EMPRESS of ETH Zurich, and the Erlangen General Purpose Array. Commercially available multiprocessors include the Denelcor Heterogeneous Element Processor (HEP) and the Cray X-MP. (The HEP achieves MIMD performance through the use of multiple pipelined functional units.)

Local area networks of computers

Local area networks of computers are multiple computers, within fairly close physical proximity, connected by a high speed communication network. While such networks usually are not primarily intended for parallel computation, they may be used for parallel computation by distributing various portions of a concurrent algorithm among various computers on the network. (Considerable operating systems support is required to make this convenient.) Thus local area networks of computers can be used to achieve concurrent independent processing similar to that available on a multiprocessor, but communication between processors is substantially slower than between multiple processors on a single computer. The use of local area networks for concurrent computation is currently being explored in various university and industrial research projects; projects considering the use of such networks for concurrent

numerical computation include the Crystal project of the University of Wisconsin, and the ENCOMP project of the University of Colorado that is discussed in Section 6.

The above list of parallel computer architecture categories is sufficient for the purposes of this paper, but it is by no means exhaustive. Another important category of parallel machines for numerical computation are "multi-microprocessors", computers consisting of substantial numbers of simple processing elements. These may be configured as SIMD or MIMD computers and thus could also be considered as further examples of processor arrays or multiprocessors. Examples include the NASA Langley finite element machine, the University of Texas reconfigurable array processor, and the University of Maryland ZMOB. Additional parallel architectures include "data-flow machines" and "systolic arrays"; machines of this type are just beginning to appear in practice and have hardly been used for numerical computation.

3. General characteristics of parallel algorithms

Before discussing parallel optimization algorithms, it is helpful to mention some general characteristics of parallel algorithms, and their relation to the classes of parallel computers discussed in the previous section. In this section we briefly discuss five attributes that are emerging as important characteristics of parallel algorithms in general and parallel optimization algorithms in particular. These attributes are : the level of parallelism; communication requirements; uniformity of operations; generation of processes; and control of processes. We also discuss how the suitability of various parallel architectures to a particular parallel algorithm depends on these characteristics. The attributes we consider constitute a primitive taxonomy of parallel algorithms, something now just starting to emerge. Indeed, this section draws upon the ideas discussed at the Taxonomy of Parallel Algorithms Workshop held in Santa Fe in 1983, especially the paper by Siegel [1983]. An interesting discussion of characteristics of parallel algorithms for multiprocessors is given in Kung [1976].

To aid our considerations, it is helpful to consider a simple optimization task that is very conducive to parallelism, the calculation of a finite difference gradient. That is, given a function $f(x)$ of a vector x of n variables, approximate ∇f at a specific point z by g, where

$$g_i = (f(x+h_i e_i) - f(x)) / h_i , \quad i=1, \cdots ,n .$$

Here e_i is the ith unit vector and h_i is an appropriate stepsize. The most obvious way to parallelize this calculation is to have various processors concurrently compute various components of g. Another alternative is to parallelize each computation of $f(x)$ in some manner. Now we will relate these possibilities to the attributes of parallel algorithms given above.

The level of parallelism refers to the size and complexity of the portions of the algorithm that are executed concurrently (or alternatively, the position of the concurrent segments in a top down description of the algorithm). In a high-level parallel algorithm, major portions of the algorithm are separated and executed concurrently. In a low-level parallel algorithm, individual instructions or small groups of instructions are executed concurrently. For example, the first parallel finite difference gradient algorithm mentioned above, distributing the calculation of various components of g to various processors, is a high-level subdivision of this algorithm. Such a subdivision is clearly well suited to a multiprocessor. It would be suited to a processor array only if the calculation of $f(x)$ contained no branches dependent upon x. A vector computer would not be appropriate for this calculation. In general, high-level parallel algorithms are best suited to multiprocessors, and may also be suited to local area networks of computers if relatively little inter-processor communication is required. Returning to the finite difference example, suppose the calculation of $f(x)$ consists mainly of large scale matrix-vector operations. An algorithm that calculates the components of g sequentially but vectorizes these portions of each calculation of $f(x)$ makes a low-level parallelization of the algorithm. As this example suggests, vector computers are best suited to low-level parallelism.

Communication requirements refer to the amount of data that must be shared between concurrent processes, as well as the dependence of the execution of some processes upon the receipt of information from other processes. In the finite difference example, processes computing various components of g would not have to communicate with each other at all, but presumably would pass their value of g_i back to a master process. In contrast, a Gaussian elimination algorithm where each processor holds one row of the matrix will require considerable communication between processors. In general, multiprocessors are usually well suited to parallel algorithms with high communication requirements, while networks of computers are only suited to algorithms with low communication requirements. Processor arrays often are well suited to fairly high communication requirements, especially between nearby processing elements.

Uniformity of operations refers to whether identical or different operations are performed in parallel. Clearly, vector computers require the highest uniformity, multiple applications of the same arithmetic operation to vectors of data, while processor arrays require the next highest uniformity, execution of the same sequence of instructions to multiple data in lockstep. Only multiprocessors and networks of computers can perform different instructions concurrently.

By generation of processes we refer to the determination of when and where processes are generated. Parallel algorithms may generate processes statically, meaning that the schedule for generating processes is known before execution of the program begins, or they may generate processes dynamically, meaning that the number and distribution of processes is determined during execution and usually varies as the program progresses. Most of the algorithms discussed in Section 5 generate processes statically, while the algorithm discussed in Section 6 generate processes dynamically.

Control of processes refers to the dependence of the execution of processes upon the progress of other processes. Parallel algorithms that require the start of some processes to wait for the completion of other processes are called synchronous algorithms, algorithms without this requirement are called asynchronous algorithms. Clearly, algorithms that run on processor arrays must be highly synchronous. Multiprocessors and local area networks are used for both synchronous and asynchronous algorithms, and examples of both types of optimization algorithms are given in this paper. Asynchronous algorithms usually are more difficult to design than synchronous algorithms, but often offer the greatest chance of near-optimal usage of independent multiple processors.

Finally, we need to briefly discuss how one measures the performance of algorithms on parallel computers. For vector computers, performance usually is measured in the average number of floating point operations performed per second. For the remaining parallel architectures, the issue is much less clear cut. Traditional sequential measures such as arithmetic complexity, number of iterations, and number of function evaluations, often are not appropriate for assessing parallel algorithms. Instead, one often wants some measure of how fully the parallel algorithm utilizes the computational power of the parallel computer, and how much advantage is gained over sequential algorithms. This measure will be influenced not only by the number of arithmetic operations required, but also by delays caused by synchronization, communication, and contention for shared data. The term most often used to characterize this measure of performance is "speedup"; however, no uniform definition of speedup exists. Speedup often is defined as the time required to solve a problem on a parallel machine using a given p-processor algorithm divided by the time to solve the same

problem on the same machine using the best 1-processor algorithm; in this case, the speedup should be between 1 and p. Sometimes, however, speedup is defined as the time to solve a problem by a given parallel algorithm on a given parallel machine divided by the time to solve the same problem by a sequential algorithm on a sequential machine; in this case, wider variation is possible. The results mentioned in Section 5 use both definitions.

4. Opportunities for parallelism in optimization

The primary objective of parallel computation is to solve expensive problems more quickly or more cheaply. Since many optimization problems are expensive to solve, it is worthwhile to consider whether parallel algorithms can aid in their solution. In this section, we discuss four leading reasons why optimization problems are expensive, and for each, summarize the opportunities for parallelism that are suggested and the types of parallel architectures most likely to be suitable. The four causes of expense we consider are : the size of the problem is large; the objective function or constraints are expensive to evaluate; many iterations or function evaluations are required to solve the problem; many variations of the problem must be solved. Another view of the opportunities for parallelism in optimization is presented by Dixon, Patel, and Ducksbury [1983].

By large problems we mean problems where the number of variables or constraints is large, say greater than 100. The solution of such problems is likely to require large scale linear algebra calculations at each iteration; in addition, the evaluation of the objective function, constraints, and their derivatives may be expensive due to the number of variable and constraints. Each of these areas presents opportunities for parallelism. The most obvious use for parallelism for large problems is to speed the linear algebra calculations. Thus vector computers are excellent candidates. Note, however, that the vector algorithms required rarely are particular to optimization, and have mainly been developed. Since it is also possible to construct efficient parallel linear algebra algorithms for processor arrays and multiprocessors, these computers are also candidates for large problems. If the evaluation of the functions or derivatives also involves large scale linear algebra calculations, vector computers or processor arrays again are suggested, whereas a multiprocessor could be used to evaluate a number of constraints concurrently. Yet another possibility is to evaluate or approximate derivative components concurrently; multiprocessors or processor arrays are

most likely to be suitable. In summary, large scale optimization problems present opportunities for virtually all parallel architectures, but many of the parallel computations they seem to suggest are not particular to optimization.

Problems where the objective function or constraints are expensive to evaluate suggest two types of opportunities for parallelism. The first is to perform each individual function evaluation by a parallel algorithm. For example, if the objective function is itself a system of partial differential equations, a parallel partial differential equation algorithm and a computer appropriate for this algorithm may be suggested. Again, the development of this routine usually is outside the realm of optimization. The second possibility is to perform multiple evaluations of the objective function concurrently. The most obvious use of concurrent function evaluations in optimization algorithms is in concurrent finite difference derivative algorithms such as the one mentioned in the previous section. The development of other effective optimization algorithms that utilize concurrent function evaluations is one of the fundamental challenges in the development of parallel optimization algorithms; some possibilities are mentioned in Sections 5 and 6. Concurrent function evaluation algorithms are best suited to multiprocessors or networks of computers, although processor arrays also can be used if the function has no data dependent branches.

Optimization problems often require many iterations (and function evaluations) when the starting guess is far from a minimizer and the objective function or constraints are highly nonlinear. Another important class of problems that usually require many function evaluations are "global optimization" problems, where multiple local minima exist and the lowest of these is required. In general, these problems present two obvious opportunities for parallelism : concurrent function evaluations, or concurrent iterations. The use of concurrent iterations in the search for a single ("local") minimizer is an interesting possibility that has hardly been explored. The use of concurrent function evaluations or iterations in solving the global optimization problems appears to be one of the most fruitful areas for the application of parallelism to optimization, and is discussed in both Sections 5 and 6. Again, multiprocessors or networks of computers are appropriate parallel architectures for these types of concurrent algorithms, while processor arrays might sometimes be appropriate.

Finally, optimization applications are often expensive because many versions of the same problem must be solved. A common example is a parametric study where the objective function depends not only on the variables x but also on some parameters y, and the minimizer with respect to x is required for various values of y. Here the most obvious use of parallelism is to run multiple problems concurrently; this should be very effective and requires no new algorithm development. In some cases, each run depends upon information from previous runs; in this case, it may be necessary to

speed individual runs instead. Techniques for this case are likely to be application dependent, and we are not aware of any existing applications of parallelism to this area.

There is one important case when the use of parallelism may be important even though the optimization problem being solved is not particularly expensive. This is the case of "real-time" problems that must be solved within an allotted time limit. Clearly, the best ways to speed the solution of real-time problems may be application dependent. Since the predominant cost of most optimization algorithms is either linear algebra, expense per function evaluation, or number of function evaluations and iterations, however, the approaches indicated above for expensive problems should often be appropriate for real-time problems as well.

5. Survey of parallel optimization research

In recent years a considerable amount of research activity has been devoted to developing and testing parallel numerical algorithms. Very little of this activity, however, has been in numerical optimization. The reasons for this become clear if one considers the opportunities for parallel optimization mentioned in the previous section together with the availability of parallel computers so far. From the previous section, it appears that most of the opportunities for parallelism in optimization involve independent concurrent processes, and are therefore best implemented on multiprocessors or local area networks of computers, and sometimes may be implemented on processor arrays. There does not appear to be much opportunity for the exploitation of vector processing that is particular to optimization. The availability of parallel computers up to this time has been the opposite of these needs, that is, good access to vector computers but very little availability of processor arrays, multiprocessors, or networks of computers. This is an important reason why parallel optimization has been relatively slow to develop.

There has been a limited amount of work in parallel optimization, however, as well as work in parallel linear algebra that is especially relevant to optimization. In this section we survey this research. First we give references to some relevant parallel linear algebra research, especially conjugate gradient methods for solving systems of linear equations. The predominant research in parallel optimization that we are aware of has been conducted by a group at The Hatfield Polytechnic. We summarize their research in parallel algorithms for four applications : a modified Newton's method for

unconstrained optimization, a nonlinear conjugate gradient method, global optimization, and nonlinear least squares. We conclude the section with references to a few other research projects in parallel optimization.

Much of the research to date in parallel numerical computation has concerned the solution of systems of linear equations. Algorithms have been developed for full, sparse, tridiagonal, and banded matrices. Most work has concentrated on algorithms for vector computers but algorithms for processor arrays and multiprocessors have also been developed. Many algorithms are adaptations of the standard sequential algorithms such as Gaussian elimination, QR factorization, and Cholesky decomposition, some are more specially developed for parallel computation. Many of the implementations have achieved efficient speed or speedup. Some representative references include : Heller [1978] and Sameh [1977] (surveys of algorithms), Duff [1983], Fong and Jordan [1977], Jordan [1979], and Rodrigue [1982] (all mainly concerned with vector computation), Hockney and Jesshope [1981] (vector computers and processor arrays), and Dongarra and Hiromoto [1983], Kapur and Browne [1981], and Lord, Kowalik, and Kumar [1980] (multiprocessors).

Of special interest to optimization are parallel algorithms for solving linear systems by conjugate gradient methods. Linear conjugate gradient algorithms consist mainly of matrix-vector multiplications and inner products, so they lend themselves nicely to vectorization or concurrent computation. Much of the interesting research has concerned the effective use of preconditioners in these algorithms. Some references are Adams [1983], Adams and Ortega [1982], Kowalik and Kumar [1982], and van der Vorst [1982]. These techniques are of special interest to optimization because conjugate gradient algorithms for nonlinear optimization (see e.g. Gill, Murray, and Wright [1981]) are closely related to the linear algorithm. One parallel nonlinear conjugate gradient algorithm is described below.

A final parallel linear equations algorithm with applications to optimization is the parallel relaxation algorithm implemented by Baudet [1978] on the C.mmp multiprocessor. Baudet achieved high speedup by implementing a "chaotic relaxation" algorithm with asynchronous concurrent processes. An application of Baudet's approach to networks of computers and nonlinear optimization is mentioned at the end of the next section.

Turning to nonlinear optimization, most of the work on parallel algorithms that we are aware of has been conducted by a group at The Hatfield Polytechnic led by L. Dixon. Most of their research has been conducted on the ICL DAP, a processor array with 4096 bit serial processing elements that can compute in lockstep. Each processing element has 16K bits of local storage. The processing elements are connected in a 64 x 64 grid with each processor having access to the four neighboring elements and their storage.

The DAP is connected to an ICL 2980 computer. While we mentioned in Section 4 that processor arrays are not as well suited to parallel numerical optimization as are multiprocessors, they are reasonably well suited to *experimental* studies in parallel optimization. This is because any algorithm requiring parallel function evaluations can be tested on a processor array as long as the test functions contain no data dependent branches, a condition often satisfied by test (as opposed to real-world) objective functions. Processor arrays may not be used, however, to test algorithms that involve concurrent execution of different program segments. For this reason, most of the algorithms tested by the Hatfield group involve concurrent function evaluation or parallel linear algebra computations. A small amount of their work has been conducted on the Loughborough NEPTUNE computer, a multiprocessor with four processors.

The parallel optimization research at Hatfield has concentrated on four types of algorithms, modified Newton methods for unconstrained optimization, nonlinear conjugate gradient methods, global optimization methods, and methods for nonlinear least squares. It is reported in a number of survey papers, including Dixon and Patel [1982], Dixon, Patel, and Ducksbury [1983], and Patel [1982a], as well as many papers mentioned below.

The parallel unconstrained optimization methods discussed in Dixon [1981] and Patel [1982b] are adaptations to an processor array of a finite difference Newton's method - line search algorithm. Parallelism is achieved in three ways : by computing the function evaluations involved in calculating the finite difference gradient and Hessian in parallel; by using a parallel algorithm to solve the system of linear equations required to calculate the Newton step; and by conducting 1, 2, and 4 dimensional line searches which evaluate the function at up to 4096 points simultaneously. (The four possible search directions are the Newton direction, the steepest descent direction, and the directions to the previous past two iterates.) In all of the test results reported by the Hatfield group, the times required by the parallel algorithms on the DAP are compared to the times required by a sequential algorithm on a faster sequential machine (a DEC 1091). Since the relative speeds of the DAP and the sequential machine that are given as a benchmark are highly variable, it is difficult to calibrate the gains that are achieved. It is clear, however, that on test problems such as those reported with $n = 64$, the parallel algorithm is substantially faster than the sequential Newton algorithm or a sequential variable metric method. This is largely because the finite difference calculations and the solution of linear equations dominate the computing cost, and are done considerably faster by the parallel machine than by the sequential machine. The intriguing idea of concurrent multiple dimension, multiple step length line searches is not shown to produce any great advantage; the computational results show no clear cut advantage in going from one to two or four dimensions, while

the data reported do not permit an assessment of the effect of multiple step lengths.

Experience in running parallel linear and nonlinear conjugate gradient algorithms on the DAP is reported in Dixon, Ducksbury, and Singh [1982]. The algorithms are designed specifically for finite element problems, with each processor handling one finite element. For the linear problem, the matrix-vector products and inner products that constitute the bulk of the algorithm are computed in parallel by having each processor concurrently compute the contribution of its finite element to the desired result. The nearest neighbor connections of the DAP are very helpful here. For the nonlinear problem, the gradient is also computed in parallel by combining the contribution of each finite element. Test results for both algorithms are reported for linear problems only; for problems with enough nodes to utilize most of the processors on the DAP, the parallel algorithm clearly is much faster than a sequential algorithm since up to 4096 computations are performed concurrently during much of the parallel algorithm. Neither preconditioning, nor the interesting question of how to handle problems with more nodes than the number of processors, are addressed.

Parallel global optimization algorithms designed for the DAP and the NEPTUNE multiprocessor are considered in Dixon and Patel [1981] and Ducksbury [1982]. As we stated in Section 4, the global optimization problem is perhaps the most obvious candidate for parallel optimization, because the solution of these problems usually requires many function evaluations, and because the problem does not appear to require as inherently sequential algorithms as the calculation of a single local minimizer.

Dixon, Patel, and Ducksbury investigate how a primitive global optimization algorithm, the algorithm of Price [1981] that is largely intended for mini-computers, can be implemented on parallel computers. Price's algorithm consists of the random sampling of $f(x)$ at a number of points, followed by the repeated replacement of a high point by a new lower point generated by a reflection operation through some of the current points. No derivative information or local minimization techniques are used. The version developed by the Hatfield group for the DAP samples $3n$ groups of 4096 points concurrently, resulting in $3n$ points at each processor; then it repeatedly generates new points at each processor concurrently. The total number of function evaluations required by the parallel algorithm on test problems is usually 10 to 100 times that required by the sequential algorithm, but since 4096 test function evaluations are performed concurrently on the DAP, the parallel algorithm requires less time than the sequential algorithm on the faster sequential machine, by factors of 3 to 68. The multiprocessor algorithm is a different adaptation of Price's algorithm where the ith processor, $i = 0, \cdots, 3$, repeatly attempts to replace the ith worst remaining point by reflection. The algorithm is asynchronous with the processors accessing a global store of the sampled points. Speedups in the range 2 to 3.8 are reported when testing

the four-processor algorithm against a similar one-processor algorithm on the NEPTUNE.

While the results achieved by the Hatfield group show considerable speedup of Price's algorithm on processor arrays and multiprocessors, it must be acknowledged that Price's algorithm is not a competitive global optimization algorithm for standard sequential machines. For example, in tests by Rinnooy Kan and Timmer [1983], a more sophisticated version of Price's algorithm still usually requires an order of magnitude more function evaluations than a state-of-the-art stochastic method. Parallel global optimization that use more sophisticated optimization techniques still need to be developed. A brief discussion of a more sophisticated parallel global optimization algorithm is included in Dixon and Patel [1982]; another alternative is presented in Section 6.

The final type of parallel optimization research reported by the Hatfield group is a brief study by McKoewn [1979] of two-processor nonlinear least squares algorithms run on an earlier, two-processor version of the NEPTUNE multiprocessor. Two techniques are considered for achieving parallelism. The first is the obvious idea of evaluating different portions of the sum of squares function concurrently. Clearly for a data fitting problem where each residual function requires a similar amount of time to evaluate, this should lead to good speedup on a multiprocessor, and McKeown's computational results support this contention. The second parallel algorithm tested is an interesting asynchronous algorithm where one processor computes the search directions while the other does line searches using the latest available information. Test results showed that the total number of function evaluations required may degrade very little (the desired result) or significantly over the comparable sequential algorithm, depending on seemingly unrelated factors, such as amount of input/output, that effect the timing. Speedup information is not reported.

In summary, it should be noted that most of the parallel optimization algorithms developed by the Hatfield group are closely related to existing sequential optimization algorithms. While these are important, in fact groundbreaking, contributions, it is our opinion that future developers of parallel optimization algorithms will also need to consider new algorithmic approaches that are not necessarily best or even suitable for sequential computation. This principle has already been demonstrated many times in the development of parallel algorithms; see for example the discussion of recurrences and tridiagonal linear systems in Hockney and Jesshope [1981]. We believe that the design of fundamentally parallel optimization algorithms offers many new and exciting possibilities; hopefully the material in the next section provides an indication of approaches geared specifically to parallel computation.

A few other projects related to parallel optimization have been reported. Straeter [1973], Straeter and Markos [1975], Housos and Wing [1980], and van Laarhoven [1984] have proposed parallel versions of a variety of unconstrained optimization algorithms, including variable metric and conjugate direction methods. Mohan [1982] has developed a parallel traveling salesman algorithm and tested it on the Cm*. Feijoo and Meyer [1984] currently are testing parallel nonlinear network optimization algorithms on the local area network of computers developed under the Crystal project at the University of Wisconsin.

6. Parallel optimization at the University of Colorado

We and several colleagues at the University of Colorado have recently begun research into the development of parallel optimization algorithms suitable for implementation on a local area network of computers. This research is part of a larger project at the Computer Science Department of the University of Colorado , the ENCOMP project, investigating the use of a local area network of computers for parallel computation. In this section we first give a very brief description of the ENCOMP project. Then we discuss some of the approaches to parallel optimization we are investigating, focusing on an approach to global optimization that is suitable to multiprocessors as well as networks of computers.

The aim of the ENCOMP project is to develop and test parallel numerical algorithms designed specifically for a local area network of computers. This computing environment is of interest for three main reasons. First, it is becoming increasingly common in practice. Second, as will be indicated below, the areas of numerical computation we are interested in, such as optimization and VLSI design, lend themselves naturally to solution by concurrent algorithms that require little inter-process communication. Thus they are well suited to parallel solution on a local area network of computers, where the communication speed between processors is relatively slow compared to the computing speed of each processor. Third, there now exist commercially available operating systems that support inter-process communication, thereby enabling one to readily develop and test concurrent programs on networks of computers. In particular, we are using a network of Sun workstations and VAX computers with each node running the Berkeley Unix 4.2 operating system, and the nodes connected on an Ethernet, and we have been able to develop and run concurrent programs almost immediately upon installation of this network. Naturally, additional support software is

still desirable to make the development, debugging, testing, and evaluation of concurrent programs more convenient.

The main class of optimization problems we are considering share the characteristic that they can solved by partitioning the original problem, often dynamically, into a number of subproblems, and then solving each subproblem, with limited communication required between the subproblems. In many cases this leads to a quite different algorithm than has been considered for sequential computation. The foremost optimization problem we propose to solve in this manner is the global optimization problem,

$$\underset{x \in D}{\text{minimize}} \; f : R^n \to R \; .$$

Other problems that can be approached in this way include determining the feasibility of a system of constraints,

$$\text{given} \quad D \in R^n \quad \text{and} \quad c_i : R^n \to R, \quad i = 1, \cdots, m$$

$$\text{determine whether there exists } x \in D \text{ for which } c_i(x) \leq 0, \quad i = 1, \cdots, m \; ,$$

the nonlinear mini-max problem,

$$\underset{x \in D}{\text{minimize}} \; \alpha$$

$$\text{subject to} \quad c_i(x) \leq \alpha, \quad i = 1, \cdots, m$$

and the global solution to a system of nonlinear equations.

The global optimization algorithm we are developing is most closely related to the stochastic optimization method of Boender et al [1982], to our knowledge the most efficient known sequential stochastic optimization method. Our algorithm basically consists of adaptively partitioning the variable space into subregions likely to contain one local minimizer each, and then simultaneously finding these local minimizers using separate processors, recurring this procedure as necessary. Additional efficiency is attained by the early termination of subregions where the function is high. Below we give a high level, oversimplified description of the algorithm that is executed by each process invoked by the concurrent algorithm. Variable and procedure names are italicized and inter-process communication is in boldface.

Global (*Function*, *Subregion*, *Lowest-anywhere-so-far* (* shared variable *))

1. evaluate *Function* at several points in *Subregion* ;
   throw away very high points and take a steepest descent step from the rest
   (* for initial (entire) region, this step can be executed in parallel,

and *lowest-anywhere-so-far* is initialized to the lowest function value found *)

2. cluster points with the aim of forming one cluster for all points in the region of attraction of each local minimizer ;
   partition *Subregion* into smaller subregions containing these clusters
3. IF (number of subregions identified at step 3) > 1
   THEN (attempt to) **create a process for each new subregion**
   (* subsequent instantiations of *Global* begin at step 4 *)
4. Run a local minimization algorithm in *Subregion* for at most a predetermined number of steps
5. IF (lowest function value found in *Subregion*) < *Lowest-anywhere-so-far*)
   THEN **broadcast this new value of** *Lowest-anywhere-so-far* **to all other processes**
6. IF (local minimizer found at step 4) THEN
   perform steps 1-2
   IF (number of clusters found) > 1
   THEN go to step 3
   ELSE terminate this process
   (* global minimizer in *Subregion* has been found *)
7. IF (lowest function value found in *Subregion*) = *lowest-anywhere-so-far*
   THEN go to step 4
   ELSE
   (* try to determine whether this subregion can be discarded because *Function* has high values throughout *)
   estimate a lower bound on *Function* in *Subregion*
   (* may involve further sampling *)
   IF (lower bound) < *lowest-anywhere-so-far*
   THEN go to step 4
   ELSE terminate this process

The parallelism in this algorithm is at the highest level; each process is an embellished local optimization algorithm. The communication requirements between processes, namely initiating and terminating a small number of processes and broadcasting and receiving values of the single shared variable *lowest-anywhere-so-far*, are very small in comparison to the computational requirements of each process, since each evaluation of *Function* typically takes many milliseconds or seconds. The "shared variable" actually can be maintained by each process and updated asynchronously. Thus the entire algorithm can be implemented as an asynchronous concurrent algorithm on a loosely or tightly coupled multiprocessor or on a local area network of computers.

The parallel global optimization algorithm contains several features which represent significant departures from existing sequential algorithms, and will require substantial research. While sequential algorithms such as Boender et al [1982] include clustering, the clustering at step 2 is different in that it tries to locate convex regions, and not just proximity as do most existing clustering algorithms. The partitioning required at the end of step 2 may be accomplished by formulating the partitioning problem as a linear program or by various heuristic approaches such as the perceptron algorithm; these approaches must be compared and new ones may be

investigated. Of course the partitioning problem may be infeasible, in which case the clusters must be modified. Perhaps the most interesting aspect of the global optimization algorithm is the lower bound required at step 7. This step is related in its objective to the termination step of existing sequential global optimization algorithms, but better procedures should be possible because the objective function should often be convex inside the subregion. One possible way to calculate a lower bound is to calculate a convex function that interpolates or underestimates all the points sampled in the subregion so far, and take the minimum of this underestimating function as the lower bound.

We have not yet implemented the above parallel global optimization algorithm, so we can only speculate on its efficiency. It would appear that on problems where a sequential algorithm must identify even a handful of local minimizers before declaring that it has found the global minimizer, the parallel algorithm has a good chance of doing this work concurrently and hence achieving a speedup close to the number of processors. In fact, even greater speedup over existing sequential algorithms may be possible from the early termination of subregions, if this allows the parallel algorithm to skip finding local minimizers that the sequential algorithm would find. Only experimentation will show if these hopes are realized. Experimentation is also required to determine whether the communication requirements are, in fact, insignificant in relation to the total amount of computation.

The feasibility of nonlinear constraints and nonlinear mini-max problems also lend themselves nicely to concurrent solution techniques based on partitioning. First, both problems are themselves global optimization problems, so it should be useful to partition the feasible region into subregions as discussed above. In addition, the constraints may be partitioned into subsets. Combining these two types of partitioning leads to interesting algorithmic possibilities. Consider for example the feasibility of constraints problem with two constraints. Suppose we divide the region D into two subregions $D1$ and $D2$, and then determine whether each constraint is feasible in each subregion. If, for example, c_1 is infeasible in $D1$ and c_2 is infeasible in $D2$, then the entire problem is infeasible. If both constraints are feasible only in $D1$, then the algorithm can restrict its attention to this subregion. These types of approaches again seem to offer the possibility of effective use of multiple processors with small interprocessor communication requirements, but experimentation is required to determine whether this goal is achieved.

Our experience in implementing and testing concurrent optimization algorithms so far is very limited, because the Berkeley Unix 4.2 operating system which we require to run concurrent algorithms on a network of computers has only been available for a few months. So far we have shown the feasibility of using our local area network for concurrent computing by implementing a finite difference gradient algorithm of the type discussed in Section 3, and a version of the chaotic relaxation algorithm of Baudet [1978] for solving linear systems. We plan next to develop and test a chaotic relaxation method for solving systems of nonlinear equations. This is another natural parallel

optimization method, but it may require a higher rate of inter-process communication than the algorithms discussed above, and hence be better suited to multiprocessors than to networks of computers.

7. Concluding remarks

The development of parallel algorithms for nonlinear optimization clearly is in its infancy. Hopefully, the previous three sections have indicated that parallel optimization is a fruitful area for future research, especially as multiprocessors and local area networks of computers become available. We have already summarized, in Section 4, the opportunities we see for parallelism in optimization, grouped according to the main factors that make optimization problems expensive. We conclude by focusing on a few of the main types of optimization problems that seem most conducive to parallelization. Our discussion is related to that in Dixon, Patel, and Ducksbury [1983] and in many other papers by the Hatfield group.

We have already stated that the global optimization problem may be the most obvious candidate for significant gains from parallelism in optimization. The stochastic methods discussed in Sections 5 and 6 are just two of a myriad of possibilities for parallel global optimization. Many sequential approaches to solving global optimization problems exist, including various types of stochastic algorithms (see e.g. Rinnooy Kan and Timmer [1983] for further references), deterministic methods such as the tunneling algorithm of Levy et al [1981], and a large class of methods for constrained global optimization problems with concave objective functions (see e.g. Rosen [1983]). All of these approaches seem to suggest excellent possibilities for parallel algorithms suitable to multiprocessors, networks of computers, and in some cases, processor arrays.

While numerical optimization problems are not as inherently large scale as, say, the solution of many differential equations problems, there do exist a number of important optimization problems that are inherently large. These include semi-infinite programming problems, optimal control problems, and network optimization problems. Many of the approaches to solving these problems, including conjugate gradient methods, relaxation methods, branch and bound methods, and the partially separable methods introduced recently by Griewank and Toint [1982], appear well suited to parallelization. Vector computers may be advantageous in some of these cases.

For local (as opposed to global) optimization problems where function evaluation is expensive, but the optimization problem itself is not too difficult to solve, the best

approach may often be to parallelize the function evaluation itself. Barring this, the most obvious utilization of parallelism is the concurrent calculation of components of finite difference derivatives. But there remain many other interesting possibilities, mostly unexplored, for the utilization of concurrent function evaluations.

Real-time optimization problems occur in a variety of applications including on-line process control. A far wider range of possibilities may be admissible for solving real-time problems than for standard off-line problems, since sub-optimal use of computing resources may be far more tolerable if it leads to satisfaction of a time bound. The solution of real-time optimization problems by parallel algorithms is likely to be application dependent, however.

The above list is by no means exhaustive. As in any new research area, the best problems may not yet have been identified, and some of the best solutions almost certainly have not yet been developed. In fact, the above list falls somewhat into the trap of suggesting parallel versions of sequential algorithms. We close by reiterating the need to also look at fundamentally new algorithms specifically designed for parallel computers.

8. References

L. Adams [1983], "An M-step preconditioned conjugate gradient method for parallel computation", *Proceedings of the 1983 International Conference on Parallel Processing*, pp. 36-43.

L. Adams and J. Ortega [1982], "A multi-color SOR method for parallel computation", *Proceedings of the 1982 International Conference on Parallel Processing*, pp. 53-56.

G. Baudet [1978], "Asynchronous iterative methods for multiprocessors", *Journal of the Association for Computing Machinery* 25, pp. 226-244.

C. G. E. Boender, A. H. G. Rinnooy Kan, L. Stougie, and G. T. Timmer [1982], "A stochastic method for global optimization", *Mathematical Programming* 22, pp. 125-140.

J. E. Dennis Jr. and R. B. Schnabel [1983], *Numerical Methods for Nonlinear Equations and Unconstrained Optimization*, Prentice-Hall, Englewood Cliffs, New Jersey.

L. C. W. Dixon [1981], "The place of parallel computation in numerical optimization I, the local problem", Technical Report No. 118, Numerical Optimisation Centre, The Hatfield Polytechnic.

L. C. W. Dixon and K. D. Patel [1981], "The place of parallel computation in numerical optimization II, the multiextremal global optimisation problem", Technical Report No. 119, Numerical Optimisation Centre, The Hatfield Polytechnic.

L. C. W. Dixon and K. D. Patel [1982], "The place of parallel computation in numerical optimization IV, parallel algorithms for nonlinear optimisation", Technical Report No. 125, Numerical Optimisation Centre, The Hatfield Polytechnic.

L. C. W. Dixon, K. D. Patel, and P. G. Ducksbury [1983], "Experience running optimisation algorithms on parallel processing systems", Technical Report No. 138, Numerical Optimisation Centre, The Hatfield Polytechnic.

L. C. W. Dixon, P. G. Ducksbury, and P. Singh [1982], "A parallel version of the conjugate gradient algorithm for finite element problems", Technical Report No. 132, Numerical Optimisation Centre, The Hatfield Polytechnic.

J. J. Dongarra and R. E. Hiromoto [1983], "A collection of parallel linear equations routines for the Denelcor HEP", Technical Report ANL/MCS-TM-15, Mathematics and Computer Science Divsion, Argonne National Laboratory.

P. G. Ducksbury [1982], "The implementation of a parallel version of Price's (CRS) algorithm on an ICL DAP", Technical Report No. 127, Numerical Optimisation Centre, The Hatfield Polytechnic.

I. S. Duff [1983], "The solution of sparse linear systems on the Cray-1", Technical Report CSS 125 (revised), Computer Science and Systems Divsion, AERE Harwell.

B. Feijoo and R. R. Meyer [1984], "Piecewise-linear approximation methods for non-separable convex optimization", Technical Report No. 521, Computer Sciences Department, University of Wisconsin - Madison (to appear).

M. J. Flynn [1966], "Very high-speed computing systems", *Proceedings of the IEEE* 54, pp. 1901-1909.

K. W. Fong and T. L. Jordan [1977], "Some linear algebraic algorithms and their performance on the Cray-1", Report LA-6774, Los Alamos National Laboratory.

P. E. Gill, W. Murray, and M. H. Wright [1981], *Practical Optimization*, Academic Press, London.

A. O Griewank and Ph. L. Toint [1982], "On the unconstrained optimization of partially separable functions", in *Nonlinear Optimization 1981*, M. J. D. Powell ed., Academic Press, London, pp. 301-312.

D. Heller [1978], "A survey of parallel algorithms in numerical linear algebra", *SIAM Review* 20, pp. 740-777. pp. 409-436.

R. W. Hockney and C. R. Jesshope [1981], *Parallel Computers*, Adam-Hilger Ltd., Bristol, England.

E. C. Housos and O. Wing [1980], "Parallel nonlinear minimization by conjugate directions", *Proceedings of the 1980 International Conference on Parallel Processing*, pp. 157-158.

K. Hwang and F. A. Briggs [1984], *Computer Architecture and Parallel Processing*, McGraw-Hill, New York.

T. L. Jordan [1979], "A performance evaluation of linear algebra software in parallel architectures", in *Performance Evaluation of Numerical Software*, L. D. Fosdick, ed., North-Holland, Amsterdam, pp. 59-76.

R. Kapur and J. Browne [1981], "Block tridiagonal system solution on reconfigurable array computers", *Proceedings of the 1981 International Conference on Parallel Processing*, pp. 92-99.

J. Kowalik and S. P. Kumar [1982], "An efficient parallel block conjugate gradient method for linear equations", *Proceedings of the 1982 International Conference on Parallel Processing*, pp. 47-52.

H. T. Kung [976], "Synchronized and asynchronous parallel algorithms for multiprocessors", in *Algorithms and Complexity : Recent Results and New Directions*, J. E. Traub, ed., Addison-Wesley, pp. 153-200.

B. W. Lampson, M. Paul, and H. J. Siegert [1981], eds., *Distributed Systems - Architecture and Implementation*, Springer-Verlag, Berlin.

A. V. Levy, A. Montalvo, S. Gomez, and A. Calderon [1981], "Topics in global optimization", in *Proceedings of the Third IIMAS Workshop*, Cocoyoc, Mexico, January 1981, J. P. Hennart, ed.

R. Lord, J. Kowalik, and S. Kumar [1980], "Solving linear algebraic equations on a MIMD computer", *Proceedings of the 1980 International Conference on Parallel Processing*, pp. 205-210.

J. J. McKeown [1979], "Experiments in implementing a nonlinear least-squares algorithm on a dual-processor computer", Technical Report No. 102, Numerical Optimisation Centre, The Hatfield Polytechnic.

J. Mohan [1982], "A study in parallel computation -- the traveling salesman problem", Technical Report CMU-CS-82-136, Department of Computer Science, Carnegie-Mellon University.

K. D. Patel [1982b], "Implementation of a parallel (SIMD) modified Newton method on the ICL DAP", Technical Report No. 131, Numerical Optimisation Centre, The Hatfield Polytechnic.

K. D. Patel [1982a], "Parallel Computation and Numerical Optimisation", Technical Report No. 129, Numerical Optimisation Centre, The Hatfield Polytechnic.

W. L. Price [1981], "A new version of the controlled random search procedure for global optimisation", Technical Report, Engineering Department, University of Leicester, England.

A. H. G. Rinnooy Kan and G. T. Timmer [1983], "Stochastic methods for global optimization", Report 8317/0, Econometric Institute, Erasmus University, Rotterdam.

G. Rodrigue [1982], ed., *Parallel Computations*, Academic Press, New York.

J. B. Rosen [1983], "Global minimization of a linearly constrained concave function by partition of feasible domain", *Mathematics of Operations Research* 8, pp.

A. Sameh [1977], "Numerical parallel algorithms - a survey", in *High Speed Computer and Algorithm Organization*, D. Kuck, D. Lawrie, and A. Sameh, eds., Academic Press, pp. 207-228.

L. J. Siegel [1983], "Characteristics of Parallel Algorithms", presented at the Taxonomy of Parallel Algorithms Workshop, Santa Fe, New Mexico, December 1983.

T. A. Straeter [1973], "A parallel variable metric optimization algorithm," NASA Technical Note D-7329, Langley Reseach Center, Hampton, Virginia.

T. A. Straeter and A. T. Markos [1975], "A parallel Jacobson-Oksman optimization algorithm," NASA Technical Note D-8020, Langley Reseach Center, Hampton, Virginia.

H. van der Vorst [1982], "A vectorizable variant of some ICCG methods", *SIAM Journal on Scientific and Statistical Computing* 3, pp. 350-356.

P. van Laarhoven [1984], "Parallel algorithms for unconstrained optimization," *Mathematical Programming*, to appear.

END

FILMED

5-85

DTIC