On the Performance of Algorithms for Large-Scale Bound Constrained Problems

Jorge J. More'

CRPC-TR90038
February, 1990

Center for Research on Parallel Computation
Rice University
P.O. Box 1892
Houston, TX 77251-1892

PREPRINT MCS-P140-0290

On the Performance of Algorithms for Large-Scale Bound Constrained Problems

by

Jorge J. Moré

February 1990

Mathematics and Computer Science Division

Argonne National Laboratory

ARGONNE NATIONAL LABORATORY
9700 South Cass Avenue
Argonne, Illinois 60439

ON THE PERFORMANCE OF ALGORITHMS FOR LARGE-SCALE BOUND CONSTRAINED PROBLEMS

Jorge J. Moré

Mathematics and Computer Science Division

Preprint MCS-P140-0290

February 1990

Work supported by the Applied Mathematical Sciences subprogram of the Office of Energy Research, U.S. Department of Energy, under Contract W-31-109-Eng-38.

ABSTRACT

We discuss issues that affect the performance of algorithms for the solution of large-scale bound constrained problems on parallel computers. The discussion centers on the solution of the elastic-plastic torsion problem and the journal bearing problem. These two problems are model large-scale quadratic programming problems that arise as finite element approximations to elliptic variational inequalities. Performance issues are illustrated with the GPCG algorithm of Moré and Toraldo. This algorithm uses the gradient projection method to select an active set and the conjugate gradient method to explore the active set defined by the current iterate. We show that significant improvements in the performance of the GPCG algorithm can be obtained by using partitioning techniques in a parallel environment. We also show that these partitioning techniques lead to almost linear speedups on function-gradient evaluations and Hessian-vector products for partially separable functions.

1 Introduction

The solution of large-scale bound constrained problem has attracted considerable attention. Much of the work has centered on the combinatorial problem of deciding which constraints are active at the solution. Standard active set strategies drop or add one constraint at each iteration, and thus these strategies are inappropriate for the solution of large-scale problems unless the active set for the solution is estimated accurately. This combinatorial problem is a central issue in the performance of algorithms for large-scale linearly constrained problems. In this paper we do not dwell on this combinatorial problem. Our aim is to discuss some of the other issues that affect the performance of algorithms for large-scale problems of the form

$$\min\{q(x) : l \leq x \leq u\}, \tag{1.1}$$

where $q : \mathbb{R}^n \to \mathbb{R}$ is a strictly convex quadratic, and the vectors l and u specify the bounds on the variables. As we shall see, the issues that are raised by this problem are relevant to the general nonlinear bound constrained problem.

We are interested in algorithmic performance on parallel computers. Our interest is mainly due to recent results of Moré and Toraldo [9] that suggest that the above combinatorial problem has been solved for an important class of large-scale problems of the form (1.1). Moré and Toraldo considered problems that arise as finite element approximations to elliptic variational inequalities and showed that the gradient projection method can be combined with the conjugate gradient method to solve large-scale (number of variables between 5000 and 15000) bound constrained problems with a few (less than 15) iterations. The performance of this algorithm (GPCG) on an Alliant FX/8, however, was not adequate. We explore the reasons behind this inadequate performance and show that significant improvements in performance are possible by making use of partitioning techniques and the parallel processing capabilities of the Alliant. We also show that these partitioning techniques are applicable to a large class of problems.

We use the GPCG algorithm of Moré and Toraldo to illustrate performance issues. We also use the elastic-plastic torsion problem and the journal bearing problem as model large-scale problems of the form (1.1). The torsion and journal bearing problems are briefly described in Section 2, where we also show that finite element approximations to these problems lead to a strictly convex quadratic programming problem with bounds on the variables. The quadratic that arises from these approximations is a partially separable function. This class of functions, introduced and analyzed by Griewank and Toint [6], is of importance because most large-scale optimization problems are naturally represented by partially separable functions.

The GPCG algorithm is described in Section 3. This algorithm uses the gradient projection algorithm to choose an active set and the conjugate gradient method to explore

the active set. Several authors have proposed algorithms that combine the gradient projection and the conjugate gradient method for the solution of bound constrained quadratic programming problems. Recent papers on this topic include those of Yang and Tolle [12], Wright [11], and Bierlaire, Toint, and Tuyttens [2]. A difference between these algorithms is that the GPCG algorithm uses the gradient projection method until a sufficient decrease condition holds or the active set settles down. We feel that this use of the gradient projection method is an important feature of the search for the active set defined by the solution.

The description of the GPCG algorithm notes, in particular, where calls to the user-supplied software for function-gradient evaluations and Hessian-vector products are required. This is of importance because in many problems the computing time is dominated by these calls. This claim is supported by the performance profiles of Section 4 which show that about 90% of the computing time in the torsion problem and the journal bearing problem is consumed by calls to the user-supplied software.

Sections 5 and 6 explore the use of concurrency for function-gradient evaluations and Hessian-vector products of partially separable functions. We show that the use of parallelism in these computations usually leads to a synchronization problem but that it is possible to avoid this synchronization problem by a suitable partitioning of the element functions. Moreover, we present numerical results that demonstrate that the use of this partitioning leads to nearly linear speedups on the Alliant FX/8.

In Section 7 we find that the use of the partitioning techniques of Sections 5 and 6 lead to significant improvements in the efficiency of the GPCG algorithm on the torsion and journal bearing problems. Moreover, we show that these improvements hold for problems where the number of variables range between $2,500$ and $40,000$.

Other researchers have considered the connection between partially separable functions and supercomputers, but the emphasis has been on vector machines. See, for example, Lescrenier [7] and Lescrenier and Toint [8]. We also note that there has been no discussion of the synchronization problems discussed in Sections 5 and 6.

We draw attention to the work on conjugate gradient methods for the solution of linear systems on supercomputers. See, for example, Saad's [10] survey. Although this work is related to the material presented in this paper, there are important differences. One difference is that the solution of problem (1.1) requires the approximate solution of a sequence of linear systems. In most cases the accuracy required of the approximate solution is quite low, so techniques used to obtain accurate solutions may be unnecessarily costly. Another difference is that the structure imposed by partially separable functions has not been considered. Finally, much of the work on conjugate gradient methods for linear systems has concentrated on methods for sparse matrix-vector products with full vectors, but in Section 6 we need sparse matrix-vector products with sparse vectors.

2 Test Problems

We use the elastic-plastic torsion problem and the journal bearing problem as models of large-scale problems of the form (1.1). The description below brings out the main features of these problems; a complete description can be found in Moré and Toraldo [9].

The elastic-plastic torsion problem and the journal bearing problem arise as finite element approximations to elliptic variational inequalities. These two problems can be formulated as minimization problems of the form

$$\min\{q(v) : v \in K\}$$

where $q : K \to \mathbf{R}$ is a quadratic over a closed convex set K in the Hilbert space $H_0^1(\mathcal{D})$ of all functions with compact support in $\mathcal{D}$ such that v and $\|\nabla v\|^2$ belong to $L^2(\mathcal{D})$. In the elastic-plastic *torsion* problem $\mathcal{D} = (0,1) \times (0,1)$ and

$$q(v) = \tfrac{1}{2} \int_{\mathcal{D}} \|\nabla v\|^2 d\xi_1 d\xi_2 - c \int_{\mathcal{D}} v \, d\xi_1 d\xi_2$$

for some constant c. The convex set K is defined by

$$K = \{v \in H_0^1(\mathcal{D}) : |v(x)| \leq dist(x, \partial\mathcal{D}), \; x \in \mathcal{D}\}$$

with $dist(\cdot, \partial\mathcal{D})$ the distance function to the boundary of $\mathcal{D}$. In the journal bearing problem $\mathcal{D} = (0, 2\pi) \times (0, 2b)$ for some constant $b > 0$ and

$$q(v) = \tfrac{1}{2} \int_{\mathcal{D}} (1 + \epsilon \cos \xi_1)^3 \|\nabla v\|^2 d\xi_1 d\xi_2 - \epsilon \int_{\mathcal{D}} \sin(\xi_1) v \, d\xi_1 d\xi_2$$

for some constant ϵ in $(0,1)$. The convex set K is defined by

$$K = \{v \in H_0^1(\mathcal{D}) : v \geq 0 \text{ on } \mathcal{D}\}.$$

Thus, both problems are of the form

$$q(v) = \tfrac{1}{2} \int_{\mathcal{D}} w_q \|\nabla v\|^2 d\xi_1 d\xi_2 - \int_{\mathcal{D}} w_l v \, d\xi_1 d\xi_2,$$

where $w_q : \mathcal{D} \to \mathbf{R}$ and $w_l : \mathcal{D} \to \mathbf{R}$ are functions defined on the rectangle $\mathcal{D}$. In the torsion problem $w_q \equiv 1$ and $w_l \equiv c$, while in the journal bearing problem $w_q(\xi_1, \xi_2) = (1 + \epsilon \cos \xi_1)^3$ and $w_l(\xi_1, \xi_2) = \epsilon \sin \xi_1$.

Finite element approximations to these problems are obtained by triangulating $\mathcal{D}$ and replacing the minimization of q over $H_0^1(\mathcal{D})$ by the minimization of q over the set of piecewise linear functions that satisfy the constraints specified by K. The finite element approximations thus give rise to a finite-dimensional minimization problem whose variables are the values of the piecewise linear function at the vertices of the triangulation.

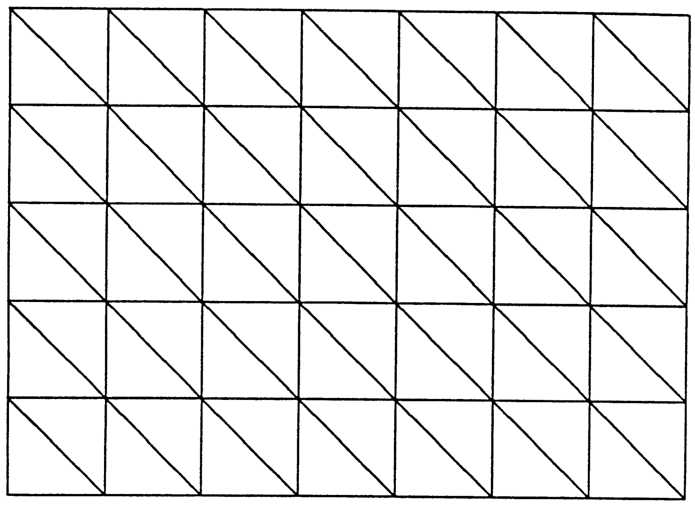

Figure 1: Triangulation of a rectangular domain

Let $\mathcal{D} = (\xi_{1,l}, \xi_{1,u}) \times (\xi_{2,l}, \xi_{2,u})$ be a rectangle in $\mathbb{R}^2$. Vertices $z_{i,j} \in \mathbb{R}^2$ for a triangulation of $\mathcal{D}$ are obtained by choosing grid spacings h_x and h_y and defining grid points

$$z_{i,j} = (\xi_{1,l} + ih_x, \xi_{2,l} + jh_y), \qquad 0 \le i \le n_x + 1, \quad 0 \le j \le n_y + 1$$

such that $z_{n_x+1,n_y+1} = (\xi_{1,u}, \xi_{2,u})$. The triangulation consists of triangular elements T_L with vertices at $z_{i,j}, z_{i+1,j}, z_{i,j+1}$, and triangular elements T_U with vertices at $z_{i,j}, z_{i-1,j}, z_{i,j-1}$. These triangular elements are shown in Figure 1.

A finite element approximation to the torsion and journal bearing problems is obtained by minimizing q over the space of piecewise linear functions v with values $v_{i,j}$ at $z_{i,j}$. The values $v_{i,j}$ are obtained by solving a quadratic programming problem of the form

$$\min\{q(v) : v \in \Omega\} \tag{2.1}$$

where q is the quadratic

$$q(v) = \frac{1}{2} \sum q_{i,j}(v) - h_x h_y \sum w_l(z_{i,j}) v_{i,j}, \tag{2.2}$$

the quadratic $q_{i,j}$ is defined by

$$q_{i,j}(v) = \mu_{i,j} \left\{ \left(\frac{v_{i+1,j} - v_{i,j}}{h_x} \right)^2 + \left(\frac{v_{i,j+1} - v_{i,j}}{h_y} \right)^2 \right\} +$$

$$\lambda_{i,j} \left\{ \left(\frac{v_{i-1,j} - v_{i,j}}{h_x} \right)^2 + \left(\frac{v_{i,j-1} - v_{i,j}}{h_y} \right)^2 \right\},$$

and the constants $\mu_{i,j}$ and $\lambda_{i,j}$ are defined by

$$\mu_{i,j} = \frac{h_x h_y}{6} \{w_q(z_{i,j}) + w_q(z_{i+1,j}) + w_q(z_{i,j+1})\} \tag{2.3}$$

and

$$\lambda_{i,j} = \frac{h_x h_y}{6} \{w_q(z_{i,j}) + w_q(z_{i-1,j}) + w_q(z_{i,j-1})\}. \tag{2.4}$$

For the torsion problem the feasible set Ω is

$$\Omega = \{v \in \mathbb{R}^{n_x n_y} : |v_{i,j}| \leq d_{i,j}\} \tag{2.5}$$

where $d_{i,j}$ is the value of $dist(\cdot, \partial\mathcal{D})$ at $z_{i,j}$, while

$$\Omega = \{v \in \mathbb{R}^{n_x n_y} : v_{i,j} \geq 0\} \tag{2.6}$$

for the journal bearing problem. For both problems $q : \mathbb{R}^n \to \mathbb{R}$ is a strictly convex quadratic, and the feasible set Ω is of the form

$$\Omega = \{x \in \mathbb{R}^n : l \leq x \leq u\} \tag{2.7}$$

for some vectors l and u in $\mathbb{R}^n$, where $n = n_x n_y$. As we shall see in our discussion of performance issues, the representation (2.2) of the quadratic q is of importance. The essential feature of this representation is that q is the sum of functions of a few variables, and thus q is a partially separable function.

3 Algorithms

We now provide a concise description of the GPCG algorithm of Moré and Toraldo [9]. Since we are interested in performance issues, the description below notes the need for function-gradient evaluations and Hessian-vector products.

The GPCG algorithm uses the conjugate gradient method to explore the active set

$$\mathcal{A}(x) = \{i : x_i \in \{l_i, u_i\}\} \tag{3.1}$$

defined by the current iterate. Once this exploration is completed, the gradient projection method is used to choose a new active set.

Given the current iterate x_k, algorithm GPCG explores the active set defined by the current iterate by computing an approximate minimizer of the subproblem

$$\min\{q(x_k + d) : d_i = 0, \ i \in \mathcal{A}(x_k)\}. \tag{3.2}$$

Given an approximate minimizer d_k of subproblem (3.2), algorithm GPCG uses a projected search to choose a search parameter α_k such that $q(x_{k+1}) < q(x_k)$ where

$$x_{k+1} = P(x_k + \alpha_k d_k) \tag{3.3}$$

and P is the projection into the feasible region Ω. The projected search requires a function-gradient evaluation for each trial value of α_k. Also note that for the bound constrained Ω defined by (2.7), the computation of the projection P only requires order n operations because

$$P(x) = mid(l, u, x)$$

where $mid(l, u, x)$ is the vector whose i-th component is the median of the set $\{l_i, u_i, x_i\}$.

The approximate minimizer d_k of subproblem (3.2) is obtained by first noting that if $i_1, \dots, i_{m_k}$ are the indices of the *free* variables, that is, those variables with indices outside of $\mathcal{A}(x_k)$, then subproblem (3.2) is equivalent to the unconstrained subproblem

$$\min\{q_k(w) : w \in \mathbb{R}^{m_k}\}, \tag{3.4}$$

where $q_k : \mathbb{R}^{m_k} \to \mathbb{R}$ is defined by $q_k(w) \equiv q(x_k + Z_k w)$, and Z_k is the matrix in $\mathbb{R}^{n \times m_k}$ whose j-th column is the i_j-th column of the identity matrix in $\mathbb{R}^{n \times n}$. Given the starting point $w_0 = 0$ in $\mathbb{R}^{m_k}$, algorithm GPCG uses the conjugate gradient algorithm until it generates w_j such that

$$q_k(w_{j-1}) - q_k(w_j) \le \eta_1 \max\{q_k(w_{l-1}) - q_k(w_l) : 1 \le l < j\} \tag{3.5}$$

for some fixed constant $\eta_1 > 0$. The approximate solution of subproblem (3.2) is then $d_k = Z_k w_{j_k}$, where j_k is the first index j that satisfies (3.5).

Each iteration of the conjugate gradient method requires a Hessian-vector product. It is important to note that the vector involved in this calculation has zero components whenever the index of that component is in the active set. Moreover, only the free components of the Hessian-vector product are needed.

If the iterate x_{k+1} generated by the conjugate gradient method appears to have identified the active set defined by the solution, then algorithm GPCG explores this active set further. The decision to continue the conjugate gradient method is based on the observation that if $\mathcal{A}(x) = \mathcal{A}(x^*)$, then the *binding set*

$$\mathcal{B}(x) = \{i : x_i = l_i \text{ and } \partial_i q(x) \ge 0, \text{ or } x_i = u_i \text{ and } \partial_i q(x) \le 0\}$$

agrees with the active set $\mathcal{A}(x)$. Thus, if the conjugate gradient method produces an iterate x_{k+1} such that $\mathcal{B}(x_{k+1}) = \mathcal{A}(x_{k+1})$, then algorithm GPCG continues to use the conjugate gradient method to explore this active set.

Once the conjugate gradient algorithm has explored an active set, algorithm GPCG uses the gradient projection method

$$y_{k+1} = P(y_k - \alpha_k \nabla f(y_k)) \tag{3.6}$$

with $y_0 = x_k$ to select a new active set. If for some fixed constant $\eta_2 > 0$ either of the two tests

$$\mathcal{A}(y_j) = \mathcal{A}(y_{j-1}), \tag{3.7}$$

$$q(y_{j-1}) - q(y_j) \le \eta_2 \max\{q(y_{l-1}) - q(y_l) : 1 \le l < j\} \tag{3.8}$$

is satisfied, then algorithm GPCG sets $x_{k+1} = y_{j_k}$, where j_k is the first index j that satisfies (3.7) or (3.8).

We note that other algorithms that combine the gradient projection and the conjugate gradient algorithm do not use tests (3.7) and (3.8). For example, Bierlaire, Toint, and Tuyttens [2] use only one iteration of the gradient projection algorithm, while Wright [11] uses the gradient projection algorithm until (3.7) holds. In our numerical results the gradient projection only requires a few iterations to satisfy (3.7) or (3.8), and in most cases (3.7) is satisfied first.

The same algorithm is used to choose the search parameter α_k in (3.3) and in (3.6). Thus, each iteration of the gradient projection method requires a function-gradient evaluation for each trial value of α_k. In addition, the initial trial value of α_k in (3.6) requires a Hessian-vector product; this is not required for (3.3) because in this case the initial trial value is $\alpha_k = 1$. Also note that the vector in the Hessian-vector product needed to obtain the initial trial value of α_k in (3.6) is $\nabla f(y_k)$, which has zero components whenever the index of that component belongs to $\mathcal{B}(y_k)$.

The convergence properties of algorithm GPCG are summarized in the following result of Moré and Toraldo [9].

Theorem 3.1 *Let $q : \mathbf{R}^n \to \mathbf{R}$ be a strictly convex quadratic. If $\{x_k\}$ is the sequence generated by algorithm GPCG for problem (1.1), then $\{x_k\}$ converges to the solution x^* of problem (1.1). If the solution x^* of problem (1.1) satisfies*

$$\partial_i q(x^*) \ne 0, \qquad i \in \mathcal{A}(x^*),$$

then algorithm GPCG terminates at the solution x^* in a finite number of steps.

This result is of interest because finite termination does not depend on finding the global minimum of subproblem (3.2). In most other algorithms for problem (1.1), finite termination is guaranteed by adding one constraint at a time until the global minimum of subproblem (3.2) is found for some active set. In the GPCG algorithm, finite termination is obtained by using the gradient projection method to identify the active set of the solution and using the conjugate gradient method to obtain the solution of subproblem (3.2) for this active set.

Table 1: Elastic-plastic torsion problem

n	nfree	iterations	nf/iter	nh/iter	time
10000	7016	14	4.5	23.3	197

Table 2: Journal bearing problem

n	nfree	iterations	nf/iter	nh/iter	time
10000	6768	10	4.6	33.3	339

4 Performance Profiles

The aim of this section is to analyze the performance of the GPCG algorithm on the torsion and journal bearing problems. The numerical results presented in this section were obtained by using double precision (16 decimal places) on the Alliant FX/8 at the Advanced Computing Research Facility of Argonne National Laboratory.

We have already noted that the elastic-plastic torsion problem and the journal bearing problem are of the form (2.1) where q is the quadratic (2.2). The quadratics $q_{i,j}$ which define q are expressed in terms of functions w_q and w_l. In the torsion problem $w_q \equiv 1$ and $w_l \equiv c$. For the test problem we set $c = 5$. In the journal bearing problem $w_q(\xi_1, \xi_2) = (1+\epsilon \cos \xi_1)^3$ and $w_l(\xi_1, \xi_2) = \epsilon \sin \xi_1$. For the test problem we set $\epsilon = 0.1$.

The starting point for the GPCG algorithm in the torsion problem is the vector $x_0 = u$, where u is the vector of upper bounds; in the journal bearing problem $x_0 = l$, where $l \equiv 0$ is the vector of lower bounds.

In the triangulation of the rectangle $\mathcal{D}$ we set $n_x = n_y$. Recall that for the torsion problem $\mathcal{D} = (0,1) \times (0,1)$, while for the journal bearing problem $\mathcal{D} = (0, 2\pi) \times (0, 2b)$. In our numerical results $b = 10$.

In Tables 1 and 2 we present the numerical results obtained with the GPCG algorithm for a problem with $n = 10,000$ variables. We have used the same parameter settings (for example, $\eta_1 = 0.1$ and $\eta_2 = 0.25$) as in the results of Moré and Toraldo [9]. In these tables *nfree* is the number of free variables at the solution. This is a measure of the effort required to solve the problem, because the GPCG algorithm will need to solve (approximately) a linear system of equations with *nfree* variables. The number of iterations is of importance because it represents the number of active sets that were explored by the conjugate gradient method. We also provide the number *nf* of function-gradient evaluations, and the number *nh* of Hessian-vector products. Finally, we list the time needed to satisfy the convergence

test

$$\|\nabla_\Omega q(x)\| \le \tau \|\nabla q(x_0)\|$$

with $\tau = 10^{-5}$. In this test the projected gradient $\nabla_\Omega q$ is defined by

$$[\nabla_\Omega q(x)]_i = \begin{cases} \partial_i q(x) & \text{if } x_i \in (l_i, u_i) \\ \min\{\partial_i q(x), 0\} & \text{if } x_i = l_i \\ \max\{\partial_i q(x), 0\} & \text{if } x_i = u_i \end{cases} .$$

For more information on the numerical results presented in Tables 1 and 2, see Moré and Toraldo [9].

The timing information that we present in this paper depends on the compiler options. In all cases we have used full optimization, automatic vectorization, and automatic concurrency. Moreover, we have allowed automatic associativity. The Alliant manual [1] contains more information on these options. In general, the use of automatic vectorization and automatic associativity leads to smaller speedups, but faster computing times.

A performance profile was obtained by executing the GPCG algorithm on one of the processors of the Alliant. These results show that most of the time is spent in function-gradient evaluations and Hessian-vector products. For the torsion problem 18.5% of the time is spent in function-gradient evaluations, and 70.7% of the time is spent in Hessian-vector products. Similar results are obtained in the journal bearing problem. For this problem 17.1% of the time is spent in function-gradient evaluations and 72.6% of the time in Hessian-vector products. Thus, on either problem, at least 89% of the time is spent in calls to the user-supplied software. This implies that any significant performance improvements for the GPCG algorithm must come about by reducing the computing time required to evaluate the user-supplied subroutines.

5 Concurrent Evaluation of Function and Gradient

We have noted that on our test problems the execution time of algorithm GPCG is dominated by the time required for the user-supplied function-gradient evaluations and Hessian-vector products. In this section we examine the impact of parallel processing on the execution times of the user-supplied subroutines. The discussion below is restricted to the quadratic defined by (2.2), but many of our remarks extend to the class of partially separable functions, that is, to any function that is the sum of functions of a few variables.

We first consider the parallel evaluation of the quadratic q defined by (2.2). The expressions $q_{i,j}(v)$ and $w_l(z_{i,j})$ in the definition of q can be evaluated in parallel. In general, these values would need to be stored in an array, and the results added at a later time. The accumulation of these values to obtain the final function value $q(v)$ can be done in parallel by dividing the values $q_{i,j}(v)$ and $w_l(z_{i,j})$ into groups and assigning the accumulation of a

group to a processor. Since we use automatic associativity in our numerical results, the Alliant compiler parallelizes the computation of q without the need for additional storage.

We now show that the parallel evaluation of the gradient usually leads to a synchronization problem. Note that the gradient ∇q is related to the gradients $\nabla q_{i,j}$ by the expression

$$\nabla q(v) = \frac{1}{2} \sum \nabla q_{i,j}(v) - h_x h_y \left(\sum w_l(z_{i,j}) \right) e$$

where $e \in \mathbb{R}^n$ is a vector with all components set to unity. Since each $q_{i,j}$ depends on 5 components of v, the gradient $\nabla q_{i,j}$ has at most 5 nonzero components. A synchronization problem arises if we try to use the storage for the gradient $\nabla q(v)$ to store these values, because different gradients $\nabla q_{i,j}$ have nonzero components in the same position. This synchronization problem can be avoided by allocating additional storage for the nonzero elements of each gradient $\nabla q_{i,j}$.

We can avoid the synchronization problem and the need for additional storage by evaluating the gradients $\nabla q_{i,j}$ in a carefully chosen order. First note that it is natural to write the function $q_{i,j}$ in the form

$$q_{i,j}(v) = q_{i,j}^L(v) + q_{i,j}^U(v)$$

where

$$q_{i,j}^L(v) = \mu_{i,j} \left\{ \left(\frac{v_{i+1,j} - v_{i,j}}{h_x} \right)^2 + \left(\frac{v_{i,j+1} - v_{i,j}}{h_y} \right)^2 \right\}$$

and

$$q_{i,j}^U(v) = \lambda_{i,j} \left\{ \left(\frac{v_{i-1,j} - v_{i,j}}{h_x} \right)^2 + \left(\frac{v_{i,j-1} - v_{i,j}}{h_y} \right)^2 \right\}.$$

This is a natural decomposition because the function $q_{i,j}^L$ is associated with the triangular element T_L with vertices at $z_{i,j}, z_{i+1,j}, z_{i,j+1}$, and $q_{i,j}^U$ is the function associated with the triangular element T_U with vertices at $z_{i,j}, z_{i-1,j}, z_{i,j-1}$.

The gradients of the functions $q_{i,j}^L$ can be evaluated in parallel by splitting the functions $q_{i,j}^L$ into groups so that functions in a group do not have variables in common. Splitting the functions in this manner guarantees that the gradients of the functions in a group do not have nonzero components in the same position, and thus avoids the synchronization problem and the need for additional storage.

The *partition* problem of splitting the functions $q_{i,j}^L$ into groups so that functions in a group do not have variables in common can be attacked by partitioning the triangles T_L into groups so that triangles in a group do not intersect. We claim that if we identify each triangular element T_L with vertices at $z_{i,j}, z_{i+1,j}, z_{i,j+1}$ with the element (i,j) of the set

$$S = \{(i,j) : 0 \le i \le n_x,\ 0 \le j \le n_y\},$$

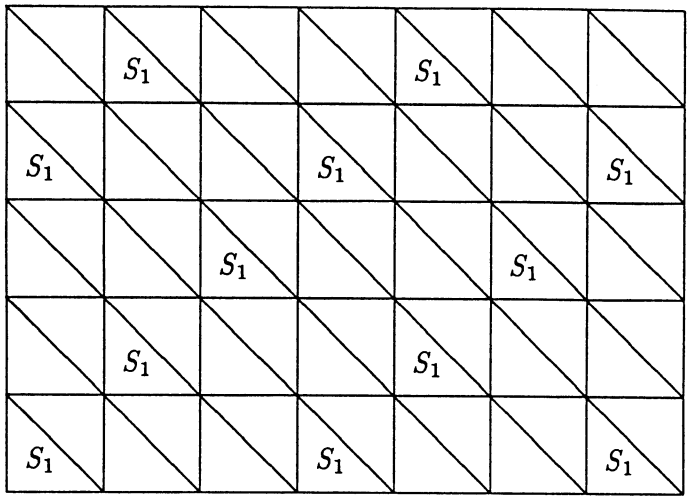

Figure 2: Triangular elements T_L in S_1

then the three sets

$$S_1 = \{(i,j) \in S : k = i - mod(j,3) \geq 0,\ mod(k,3) = 0\}$$
$$S_2 = \{(i,j) \in S : k = i - mod(j+1,3) \geq 0,\ mod(k,3) = 0\}$$
$$S_3 = \{(i,j) \in S : k = i - mod(j+2,3) \geq 0,\ mod(k,3) = 0\}$$

define a suitable partition. The proof that this partition has the desired properties is obvious once it is viewed in terms of the triangular elements T_L. For example, the triangular elements T_L associated with S_1 are marked in Figure 2.

At the programming level the partitioning can be done by splitting the code that evaluates the gradients of the functions $q^L_{i,j}$ into three loops. For example, the loop

```
do j = 0, ny
   do i = mod(j,3), nx, 3
```

can be used to evaluate the gradients of the functions in the set S_1 under the assumption that the body of this loop evaluates the gradient of the function $q^L_{i,j}$.

We have shown how to partition the functions $q^L_{i,j}$ so that functions in a group do not have variables in common. Similar techniques apply to the functions $q^U_{i,j}$.

Tables 3 and 4 present the time (in seconds) needed to evaluate the function and gradient for $n = 10,000$. The sequential code uses the unmodified function-gradient evaluation, while the parallel code partitions the function into three groups as described above. Since the time for one evaluation is small, the timings are the averages over 100 evaluations.

The main conclusion that can be drawn from the results in Tables 3 and 4 is that the partitioning technique that we have described leads to almost linear speedups on the number

Table 3: Torsion problem: Timings for function-gradient evaluation

Type of code	1 processor	8 processors	speedup
sequential	0.560	0.397	1.41
parallel	0.555	0.0778	7.23

Table 4: Journal bearing problem: Timings for function-gradient evaluation

Type of code	1 processor	8 processors	speedup
sequential	1.52	0.934	1.62
parallel	1.21	0.165	7.33

of processors. This is an important observation because this partitioning technique can be generalized to any partially separable function.

The partition problem for general partially separable functions is a difficult combinatorial problem that arises in a number of areas. For example, the general partition problem arises in the context of estimating sparse Jacobian matrices. For details and references, see Coleman and Moré [4], where the partition problem is shown to be equivalent to a graph coloring problem. Recent work in this area includes the software for the partition problem described by Coleman, Garbow, and Moré [3], and the work of Goldfarb and Toint [5] on the partition problem for matrices that arise from finite element approximations.

6 Concurrent Evaluation of Hessian-Vector Products

In this section we consider the parallel evaluation of Hessian-vector products for a partially separable function. We focus, in particular, on the case where the nonzero components of the vector are in a specified list and when it is necessary to compute only the components of the Hessian-vector product in this list.

The discussion below extends to general partially separable functions, but we consider only the quadratic q defined by (2.2). In this case, the product of the Hessian with vector d in $\mathbb{R}^n$ is given by

$$\nabla^2 q(v)d = \frac{1}{2}\sum \nabla^2 q_{i,j}(v)d.$$

From this expression it is clear that the Hessian-vector product can be obtained by evaluating each product $\nabla^2 q_{i,j}(v)d$ and accumulating the result. This technique, however, runs into the same synchronization problem as in the function-gradient evaluation. Indeed, note

Table 5: Torsion problem: Timings for Hessian-vector evaluation

Type of code	1 processor	8 processors	speedup
sequential	0.480	0.473	1.01
parallel	0.423	0.0584	7.24

Table 6: Journal bearing problem: Timings for Hessian-vector evaluation

Type of code	1 processor	8 processors	speedup
sequential	0.983	0.939	1.04
parallel	0.627	0.0863	7.26

that the vector $\nabla^2 q_{i,j}(v)d$ has the same sparsity pattern as the gradient $\nabla q_{i,j}(v)$. Given this observation, it is clear that the synchronization problem can be avoided by partitioning the functions as in Section 5.

Tables 5 and 6 contain the time (in seconds) needed to evaluate the Hessian-vector product of the quadratic q for $n = 10,000$. The sequential code uses the unmodified Hessian-vector product, while the parallel code partitions the functions $q^L_{i,j}$ and $q^U_{i,j}$ into three groups as in Section 5. We also note that the times in these tables are the averages over 100 evaluations.

The vector d used in Tables 5 and 6 had all components nonzero. Since in algorithm GPCG the vector d is generally sparse, these timings and speedups are not representative.

If we wish to take advantage of the zero components in the vector d, we can do this by constructing a boolean array *mask* so that the i-th component of d is zero whenever *mask*(i) is false. In the GPCG algorithm, the information needed to construct the array *mask* is a natural by-product of the determination of the active and binding sets.

Given the array *mask*, the product $\nabla^2 q_{i,j}(v)d$ is computed only if *mask*(l) is true for any variable v_l in $q_{i,j}$. Of course, if *mask*(l) is false for all variables v_l in $q_{i,j}$, then the product $\nabla^2 q_{i,j}(v)d$ is zero.

The above technique can lead to some wasted effort because we usually do not need to compute all the components of $\nabla^2 q_{i,j}(v)d$. Consider, for example, the case where some $q_{i,j}$ depends on variables in the set $\{1, 3, 5\}$. If the vector d has a zero component in the first component, then usually the first component of the product $\nabla^2 q_{i,j}(v)d$ is nonzero if the third or fifth component of d is nonzero. However, in applications the first component is usually not needed. This is precisely the situation in the GPCG algorithm of Section 3.

If we need to only compute those components of the Hessian-vector product specified by the boolean array *mask*, we can satisfy this requirement by making a test on *mask(l)* before the computation of the l-th component of $\nabla^2 q_{i,j}(v)d$. The amount of effort saved by this technique depends on the function $q_{i,j}$. However, this test eliminates the computation of unnecessary components.

The sequential and the parallel code in Tables 5 and 6 used the array *mask* to define the nonzero components of the array d and the components of the Hessian-vector product $\nabla^2 q(v)d$ that must be computed. We have already noted that the vector d used in these tables had all components nonzero; for sparse vectors these results tend to improve. For example, the speedup for a vector with 7000 nonzero components is 7.4 for the torsion problem and 7.5 for the journal bearing problem. Since both problems have roughly 7000 free variables at the solution, these speedups are representative of the Hessian-vector product in these problems.

7 Performance Evaluation

In the previous two sections we have considered the performance improvements in the function-gradient evaluations and Hessian-vector products for partially separable functions. We now consider the improvements in computing times for the solution of the model problems.

Tables 7 and 8 present timings (in seconds) for the solution of the torsion and journal bearing problem with the parallel version of the GPCG algorithm. The purpose of these tables is to examine the performance of the GPCG algorithm as a function of the number of variables.

The version of the GPCG algorithm used to obtain these results uses the partitioning approach of Sections 5 and 6 to speedup function-gradient evaluations and Hessian-vector products. We have also used compiler directives to improve the performance of the indirect addressing required by algorithm GPCG. For example, a loop of the form

```
do k = 1, nfree
   i = list(k)
   z(i) = min(max(l(i),x(i)+alpha*d(k)),u(i))
```

is used to compute the iterate in (3.3). This loop needs a compiler directive to certify that the use of indirect addressing does not lead to synchronization problems in the computation of the array `z`. In this loop the integer array `list` contains the indices that are free during the conjugate gradient iteration, and the array `d` specifies the approximate minimizer of subproblem (3.2). The array `d` has a packed representation because efficiency considerations have dictated the use of packed arrays in the implementation of the conjugate gradient algorithm.

Table 7: Timings for the torsion problem: Parallel algorithm

Dimension	1 processor	8 processors	speedup
2500	23.5	3.84	6.11
5625	72.0	11.7	6.15
10000	189	29.6	6.38
22500	559	90.8	6.15
40000	1172	191	6.13

Table 8: Timings for the journal bearing problem: Parallel algorithm

Dimension	1 processor	8 processors	speedup
2500	34.5	5.34	6.46
5625	121	18.0	6.72
10000	261	38.3	6.81
22500	806	128	6.29
40000	1733	287	6.03

The speedups obtained in Tables 7 and 8 compare favorably with those obtained for typical basic operations. For example, the speedup for the computation of the gradient projection iterate in (3.3) with the above loop is 5.92 for $nfree = 10,000$. In general, speedups above 6 on an Alliant FX/8 are considered quite good. A speedup of 6 on the Alliant means that we are operating with 75% efficiency.

8 Conclusions

The speedups obtained in Tables 7 and 8 show that significant improvements in the performance of the GPCG algorithm are obtained by making use of partitioning techniques and the parallel processing capabilities of the Alliant. We have concentrated on the GPCG algorithm because the numerical results of Moré and Toraldo [9] show that this algorithm is an efficient solver of large-scale problems of the form (1.1).

We have also shown in Sections 5 and 6 that the partitioning techniques lead to almost linear speedups on function-gradient evaluations and Hessian-vector products. These results are of interest because the partitioning techniques are applicable to computations with

partially separable functions on any reasonable parallel computer.

We end with a cautionary note. In the presentation of our numerical results we have concentrated on the speedup obtained by various computations, but we must not forget that the motivation for this work is to reduce the total computing time for these problems. This is one of the reasons for using full optimization, automatic vectorization, and automatic concurrency in our numerical results. As we have noted, the use of automatic vectorization and automatic associativity leads to smaller speedups, but faster computing times.

Acknowledgments

I would like to thank David Levine for his advice on the Alliant FX/8, and Gail Pieper for her comments on the manuscript.

References

[1] ALLIANT COMPUTER SYSTEMS CORPORATION, *FX/Fortran Programmer's Handbook*, 1988.

[2] M. BIERLAIRE, P. L. TOINT, AND D. TUYTTENS, *On iterative methods for linear least squares problems with bound constraints*, Report 89-5, Namur University, Namur, Belgium, 1989.

[3] T. F. COLEMAN, B. S. GARBOW, AND J. J. MORÉ, *Software for estimating sparse Jacobian matrices*, ACM Trans. Math. Software, 10 (1984), pp. 329–345.

[4] T. F. COLEMAN AND J. J. MORÉ, *Estimation of sparse Jacobian matrices and graph coloring problems*, SIAM J. Numer. Anal., 20 (1983), pp. 187–209.

[5] D. GOLDFARB AND P. L. TOINT, *Optimal estimation of Jacobian and Hessian matrices that arise in finite difference calculations*, Math. Comp., 43 (1984), pp. 69–88.

[6] A. GRIEWANK AND P. L. TOINT, *On the unconstrained optimization of partially separable functions*, in Nonlinear Optimization 1981, M. J. D. Powell, ed., Academic Press, 1982.

[7] M. LESCRENIER, *Partially separable optimization and parallel computing*, Ann. Oper. Res., 14 (1988), pp. 213–224.

[8] M. LESCRENIER AND P. L. TOINT, *Large scale nonlinear optimization on the FPS164 and CRAY X-MP vector processors*, International Journal of Supercomputer Applications, 2 (1988), pp. 66–81.

[9] J. J. MORÉ AND G. TORALDO, *On the solution of large quadratic programming problems with bound constraints*, Report MCS-P77-0589, Argonne National Laboratory, Argonne, Illinois, 1989.

[10] Y. SAAD, *Krylov subspace methods on supercomputers*, SIAM J. Sci. Statist. Comput., 10 (1989), pp. 1200–1232.

[11] S. J. WRIGHT, *Implementing proximal point methods for linear programming*, Report MCS-P45-0189, Argonne National Laboratory, Argonne, Illinois, 1989.

[12] E. K. YANG AND J. W. TOLLE, *A class of methods for solving large convex quadratic programs subject to box constraints*, preprint, University of North Carolina, Department of Operations Research, Chapel Hill, North Carolina, 1988.