JOURNAL OF OPTIMIZATION THEORY AND APPLICATIONS: Vol. 66, No. 1, JULY 1990

Vectorization of Conjugate-Gradient Methods for Large-Scale Minimization in Meteorology[1]

I. M. NAVON,[2] P. K. H. PHUA,[3] AND M. RAMAMURTHY[4]

Communicated by M. Avriel

Abstract. During the last few years, conjugate-gradient methods have been found to be the best available tool for large-scale minimization of nonlinear functions occurring in geophysical applications. While vectorization techniques have been applied to linear conjugate-gradient methods designed to solve symmetric linear systems of algebraic equations, arising mainly from discretization of elliptic partial differential equations, due to their suitability for vector or parallel processing, no such effort was undertaken for the nonlinear conjugate-gradient method for large-scale unconstrained minimization.

Computational results are presented here using a robust memoryless quasi-Newton-like conjugate-gradient algorithm by Shanno and Phua applied to a set of large-scale meteorological problems. These results point to the vectorization of the conjugate-gradient code inducing a significant speed-up in the function and gradient evaluation for the nonlinear conjugate-gradient method, resulting in a sizable reduction in the CPU time for minimizing nonlinear functions of 10^4 to 10^5 variables. This is particularly true for many real-life problems where the gradient and function evaluation take the bulk of the computational effort.

It is concluded that vector computers are advantageous for large-scale numerical optimization problems where local minima of nonlinear functions are to be found using the nonlinear conjugate-gradient method.

[1] This research was supported by the Florida State University Supercomputer Computations Research Institute, which is partially funded by the US Department of Energy through Contract No. DE-FC05-85ER250000.

[2] Associate Professor, Department of Mathematics and Faculty Associate, Supercomputer Computations Research Institute, Florida State University, Tallahassee, Florida.

[3] Associate Professor, Department of Information Systems and Computer Science, National University of Singapore, Kent Ridge, Singapore.

[4] Assistant Professor, Department of Atmospheric Sciences, University of Illinois, Urbana-Champaign, Illinois.

Key Words. Conjugate-gradient methods, large-scale minimization, vectorization, direct minimization, meteorological problems.

1. Introduction

Among the major developments in recent years in the field of computing, one should count the introduction of a variety of vector and parallel computers and the development of adequate algorithms designed to efficiently utilize their capabilities. Recently, and only in a relatively small measure, there has been a start toward algorithm development of numerical optimization problems, most of the research being directed toward parallel algorithms.

By large-scale numerical optimization, we mean the minimization of functions where the number of variables is large, typically for meteorological problems of the order of 10^4 to 10^5 variables. As we are interested in large-scale optimization using nonlinear conjugate-gradient methods, which require only the storage of a few vectors, the main purpose of the present paper is to discuss and analyze the vectorization of a typical robust, modern nonlinear conjugate-gradient code and to point out the computational advantages, including the total speed-up in terms of CPU time. When using a vectorized conjugate-gradient code for the unconstrained minimization of a nonlinear function of the large-scale type, the objective function and its gradient become quite expensive to evaluate, suggesting an important role and significant gains using a vector computer.

For the nonlinear conjugate-gradient method, which constitutes the topic of the present research paper, a thorough review of the available literature points to the fact that the totality of the research activity carried by a small number of researchers was directed toward efforts in parallelizing the method; to our best knowledge, no effort was directed toward vectorizing the method.

Parallelization of the nonlinear conjugate-gradient method can be introduced by approximating the successive gradients by finite differences of the function values calculated in parallel, and one can accelerate the linear searches by simultaneous function evaluations at preselected grid-points along the search direction.

Several authors (Refs. 1-4) designed parallel versions of Powell's nongradient method (Ref. 5), generating conjugate search directions by minimization over geometrically parallel manifolds. This results in simultaneous line searches, but computational experience up to date is too limited (see Ref. 6).

The Hatfield Polytechnic Group has investigated the conjugate-gradient methods of Ref. 7, which generate conjugate-search directions without exact linear searches (Refs. 7-12). Other work on parallel optimization is reported in Ref. 13. In Refs. 14 and 15, the authors used pseudo-conjugate directions for the solution of the nonlinear unconstrained optimization problem on a parallel computer using the nongradient method of Ref. 5. Other efforts in this direction involved Refs. 16-18, etc.

No report appears to be available concerning the speeding up of the nonlinear conjugate-gradient method for large-scale optimization on vector computers. This issue is of crucial importance when we solve problems with expensive function/gradient evaluations, which appears to be the case for large-scale meteorological applications.

It is important to develop very efficient unconstrained minimization algorithms, not only because the problem occurs in many instances on its own, but even more so because an unconstrained minimization problem must be solved in the inner loop of the solution of important constrained nonlinear problems. As mentioned in Ref. 12, vector computers may be advantageous in the case of large-scale unconstrained minimization. These large-scale minimization problems occur in applications in meteorology, computational chemistry, and structural optimization, to cite but a few of the application fields.

The plan of our paper is as follows. In Section 2, we will describe the relevant large-scale meteorological problems where the constrained nonlinear optimization (e.g., the augmented Lagrangian formulation) was applied. A large-scale unconstrained optimization problem must be invariably solved in the inner loop of the solution of the augmented Lagrangian constrained nonlinear minimization. The robust memoryless quasi-Newton-like conjugate-gradient solver due to Ref. 20, its structure, and its computational complexity will be described in Section 3.

Numerical tests (Ref. 21) show that limited-memory quasi-Newton-like conjugate-gradient methods with inexact line searches require substantially fewer function evaluations than the simple conjugate-gradient method where little additional storage is required. Numerical results concerning the vectorization of the function/gradient evaluation part, which is problem dependent but which in real life is always the most computationally intensive part of it, will be presented in Section 4.

Results concerning the performance of the conjugate-gradient code under scalar, automatic vectorization, and refined manual vectorization will be numerically and graphically presented and discussed in Section 5. The resulting speed-up of the conjugate-gradient method and the relative improvements in performance will be tabulated and summarized. Finally, the impact of the number of variables in the nonlinear function to be minimized on the speed-up performance of the vectorized nonlinear conjugate-gradient code for a particular vector supercomputer (e.g., the CYBER 205) will be discussed.

Section 6 will include a summary and concluding remarks with implications for the vectorization of different nonlinear conjugate-gradient methods applied to large-scale and very large-scale unconstrained minimization.

2. Large-Scale Meteorological Problems

Here, we will introduce the two large-scale meteorological problems where nonlinear constrained minimization was applied. The inner loop of the constrained minimization (i.e., the augmented Lagrangian method) involved large-scale unconstrained minimization solved by the quasi-Newton memoryless conjugate-gradient method of Ref. 20, hereby referred to as CONMIN.

2.1. Conservation of Integral Invariants of the Shallow-Water Equations (GUSTAF Problem). An augmented Lagrangian constrained minimization method is applied to enforce the conservation of the three integral invariants of the shallow-water equations model on a limited-area domain (see Refs. 22-24). The three integral invariants are the total mass, the total energy, and the potential enstrophy.

The augmented Lagrangian method approximates the nonlinear equality constrained minimization problem by solving a series of unconstrained minimization problems (Ref. 25). In our case, we define the functional f by

$$f=\sum_{j=1}^{N_x}\sum_{k=1}^{N_y}[\tilde{\alpha}(u_{j_k}^n-\tilde{u}_{j_k}^n)^2+\tilde{\alpha}(v_{j_k}^n-\tilde{v}_{j_k}^n)^2+\tilde{\beta}(h_{j_k}^n-\tilde{h}_{j_k}^n)^2]; \tag{1}$$

here,

$$N_x\Delta x=L, \qquad N_y\Delta y=D, \qquad \Delta x=\Delta y=h;$$

h is the grid size; n designates the time level $t_n=n\Delta t$; Δt is the time step; L and D are the respective dimensions of the rectangular domain over which the shallow-water equations are being solved (see Ref. 23).

$(\tilde{u}_{j_k}^n, \tilde{v}_{j_k}^n, \tilde{h}_{j_k}^n)$ are the predicted variables at the nth time step using a finite-difference algorithm (i.e., the nonlinear ADI method of Ref. 26 for solving the nonlinear shallow-water equations system); $(u_{j_k}^n, v_{j_k}^n, h_{j_k}^n)$ are the field values adjusted by the nonlinear constrained optimization method using the augmented-Lagrangian technique to enforce conservation of the three integral invariants of the shallow-water equations; $\tilde{\alpha}$ and $\tilde{\beta}$ are weights determined by the principle (Ref. 27) that the relative weights are selected so as to make the fractional adjustment of variables proportional to the fractional magnitude of the truncation errors in the predicted variables.

We used

$$\tilde{\alpha} = 1, \qquad \tilde{\beta} = g/H, \tag{2}$$

where H is the mean depth of the shallow fluid. The augmented Lagrangian function L is defined by

$$L(x, \lambda, r) = f(x) + \lambda^T e(x) + (1/2r)|e(x)|^2, \tag{3}$$

and the minimization of (3) replaces the problem

$$\min f(x),$$

$$\text{s.t.} \quad e(x) = 0, \qquad e = (e_1, \ldots, e_m), \qquad m \leq n, \tag{4}$$

where $e(x)$ are the equality constraints. Here,

$$x^n = (\tilde{u}^n_{11}, \ldots, \tilde{u}^n_{N_xN_y}, \tilde{v}^n_{11}, \ldots, \tilde{v}^n_{N_xN_y}, \tilde{h}^n_{11}, \ldots, \tilde{h}^n_{N_xN_x})^T; \tag{5}$$

in our particular case, the equality constraint vector has three components given by

$$e(x) = \begin{bmatrix} E^n - E^0 \\ Z^n - Z^0 \\ H^n - H^0 \end{bmatrix}, \tag{6}$$

where

$$E^n = \tfrac{1}{2} \sum_{j=1}^{N_x} \sum_{k=1}^{N_y} [\tilde{h}^n_{jk}(\tilde{u}^n_{jk})^2 + (\tilde{v}^n_{jk})^2 + g(\tilde{h}^n_{jk})^2]\Delta x \Delta y, \tag{7a}$$

$$Z^n = \tfrac{1}{2} \sum_{j=1}^{N_x} \sum_{k=1}^{N_y} [(\partial \tilde{v}^n_{jk}/\partial x - \partial \tilde{u}^n_{jk}/\partial y + f_j)/\tilde{h}^n_{jk}]^2 \Delta x \Delta y, \tag{7b}$$

$$H^n = \sum_{j=1}^{N_x} \sum_{k=1}^{N_y} \tilde{h}^n_{jk} \, \Delta x \Delta y. \tag{7c}$$

Here, E^n, Z^n, H^n are the discrete values of the integral invariants of the total energy, the potential enstrophy (i.e., the discrete sum of the square of the absolute vorticity), and the mass at time $t_n = n\Delta t$; E^0, Z^0, H^0 are the values of the same integral invariants at the initial time $t = 0$; λ is an m-component Lagrange multiplier vector,

$$\lambda = (\lambda_1, \ldots, \lambda_m)^T; \tag{8}$$

r is a penalty parameter; g is the acceleration of gravity; and $1 \leq j \leq N_x$, $1 \leq k \leq N_y$, such that $N_x \Delta x = L$, $N_y \Delta y = L$.

In our application, we follow the augmented Lagrangian algorithm (Refs. 28 and 25) for minimizing the augmented Lagrangian

$$L_{r_k}(x, \lambda_k) = f(x) + \lambda_k^T e(x) + (1/2r_k)|e(x)|^2 \tag{9}$$

and updating the Lagrange multipliers and penalty parameters. Here, k is an index of the iterations sequence. For the inexact unconstrained minimization of the augmented Lagrangian function $L_{r_k}(x_k, \lambda_k)$, we use a conjugate-gradient method which has the virtue of requiring only a few vectors for memory storage; this suits us, since we are dealing with a large-scale minimization problem. The conjugate-gradient method will be described in ample detail in the next section.

For this application, we used two grids. The first grid was a coarse grid with a space increment of

$$\Delta x = \Delta y = 400 \text{ km}, \qquad \Delta t = 3600 \text{ sec},$$

where Δt is the time step. This resulted in a 12×15 grid in the x and y directions respectively for a rectangular domain of $L = 4400$ km and $D = 6000$ km. The augmented Lagrangian function was a function of x with $12 \times 15 \times 3 = 540$ variables; i.e., the unconstrained minimization was carried out on a nonlinear function of 540 variables. A second grid, using a refined mesh space increment of

$$\Delta x = \Delta y = 40 \text{ km}, \qquad \Delta t = 360 \text{ sec},$$

was also tested. This results in $150 \times 111 \times 3 \approx 50{,}000$ variables in the nonlinear unconstrained minimization.

2.2. Constrained Adjustment to Suppress Lamb Waves (AUGLAG Problem). In meteorological applications, one is often interested in suppressing external gravity waves by modifying the observed wind field in such a way that the vertical motions vanish at the lowest level of a three-dimensional atmospheric model. An alternative way is to regard this adjustment as a variational adjustment of the horizontal wind fields in a pressure coordinate system (x, y, p), so that the pressure tendency dp/dt is zero everywhere; here, p_s is the surface pressure.

The continuity equation in pressure coordinates is given by

$$\partial u/\partial x + \partial v/\partial y + \partial w/\partial p = 0. \tag{10}$$

Integrating this equation from the top to the bottom of the atmosphere and assuming the vertical velocity $w = 0$ at both endpoints, we obtain (see Ref. 29)

$$\int_0^{p_s} (\partial u/\partial x + \partial v/\partial y)\, dp = 0. \tag{11}$$

Using this equation as a constraint will ensure that

$$dp_s/dt = 0. \tag{12}$$

In other words, using the continuity equation as a strong constraint will enable us to suppress Lamb waves.

The Lamb waves are high-speed acoustic-gravity waves which appear as solutions to the primitive equations in numerical weather prediction along with slow, physically relevant meteorological waves. As such, we are interested in suppressing the Lamb waves, which can be viewed as noise in a meteorological model and which, moreover, impose very stringent computational stability conditions on the allowable time step Δt.

The functional for which the stationary value is to be found for this problem is

$$\begin{aligned} f = &\int_x \int_y \int_p [(u-\tilde{u})^2 + (v-\tilde{v})]^2 \, dx \, dy \, dp \\ &+ \int_x \int_y \left[\lambda \int_0^{P_s} (\partial u/\partial x + \partial v/\partial y) \, dp \right] dx \, dy; \end{aligned} \tag{13}$$

here, $\tilde{u}$ and $\tilde{v}$ are the analyzed horizontal wind components, u and v are the observed horizontal wind components; and λ is the Lagrange multiplier.

In a discrete augmented Lagrangian formulation, we obtain

$$\begin{aligned} L = &\sum_i \sum_j \sum_k [(u_{ijk} - \tilde{u}_{ijk})^2 + (v_{ijk} - \tilde{v}_{ijk})^2] \Delta x \Delta y \Delta p \\ &+ \sum_i \sum_j \lambda_{ij} \left[\sum_k \left(\frac{u_{i+1,j,k} - u_{i-1,j,k}}{2\Delta x} + \frac{v_{i,j+1,k} - v_{i,j-1,k}}{2\Delta y} \right) \Delta p \right] \Delta x \Delta y \\ &+ (1/2) \sum_i \sum_j C_{ij} \left[\sum_k \left(\frac{u_{i+1,j,k} - u_{i-1,j,k}}{2\Delta x} + \frac{v_{i,j+1,k} - v_{i,j-1,k}}{2\Delta y} \right) \Delta p \right]^2 \Delta x \Delta y, \end{aligned} \tag{14}$$

where C_{ij} are the penalty terms and λ_{ij} are the Lagrange multipliers.

Our model domain is rectangular in the horizontal sense; in the vertical sense, we have 10 discrete levels, resulting in this application in a function of $46 \times 46 \times 10 \times 2$ components $\approx 42{,}320$ variables. A coarser mesh case, where the mesh spacing was increased by a factor of 2 in the horizontal sense, resulted in a function of $23 \times 23 \times 10 \times 2$ components $\approx 10{,}000$ variables. The gradient of the discrete augmented Lagrangian function L with respect to the vector x, where x is given by

$$x = (u_{111}, \ldots, u_{N_x N_y N_p}, v_{111}, \ldots, v_{N_x N_y N_p})^T, \tag{15}$$

for the three-dimensional limited-area domain in x, y, p ($N_x \Delta x = L$, $N_y \Delta y = D$, $N_p \Delta p = H$) is given by

$$\left.\frac{\partial L}{\partial u}\right|_{ijk} = 2(u_{ijk} - \tilde{u}_{ijk})\Delta x \Delta y \Delta p + \left(\frac{\lambda_{i-1} - \lambda_{i+1}}{2\Delta x}\right)\Delta x \Delta y \Delta p + \sum_k \left(\frac{u_{i+1,j,k} - u_{i-1,j,k}}{2\Delta x} + \frac{v_{i,j+1,k} - v_{i,j-1,k}}{2\Delta y}\right) \times \Delta p \left(\frac{C_{i-1,j} - C_{i+1,j}}{2\Delta x}\right)\Delta x \Delta y, \tag{16}$$

$$\left.\frac{\partial L}{\partial v}\right|_{ijk} = 2(v_{ijk} - \tilde{v}_{ijk})\Delta x \Delta y \Delta p + \left(\frac{\lambda_{i,j-1} - \lambda_{i,j+1}}{2\Delta y}\right)\Delta x \Delta y \Delta p + \sum_k \left(\frac{u_{i+1,j,k} - u_{i-1,j,k}}{2\Delta x} + \frac{v_{i,j+1,k} - v_{i,j-1,k}}{2\Delta y}\right) \times \Delta p \left(\frac{C_{i,j-1} - C_{i,j+1}}{2\Delta y}\right)\Delta x \Delta y. \tag{17}$$

The same inexact minimization of the augmented Lagrangian of Ref. 25 is applied using the same rules for updating the multipliers and penalties. The same conjugate-gradient unconstrained minimization method (CONMIN, Ref. 20) is used to minimize the augmented Lagrangian discrete functional.

3. Conjugate-Gradient Method

In our applications, we have decided to use the memoryless quasi-Newton conjugate-gradient method due to Ref. 30 and proposed in Ref. 20. This was found to be robust and performing for meteorological applications (see Ref. 19), when compared with other conjugate-gradient methods such as Fletcher–Reeves, Polak–Ribiere, and the method of Ref. 33 (IMSL Mathematical Software Library), and when compared with the E04DGF software (Ref. 34) and the methods of Refs. 35–36. The last two methods are also memoryless or limited memory quasi-Newton-like conjugate-gradient methods.

The CONMIN routine proposed in Ref. 20 finds the local minimizer of a nonlinear function $f(x)$ of n variables, where

$$x = (x_1, \ldots, x_n), \qquad n \geq 1, \tag{18}$$

can be any real numbers. This subroutine incorporates two nonlinear optimization methods (i.e., a memoryless quasi-Newton-like conjugate-gradient algorithm and a BFGS quasi-Newton algorithm), with the choice of the method being left to the user.

The conjugate-gradient algorithm option in CONMIN (Ref. 37) is the restarted memoryless variable-metric algorithm documented in Refs. 30-31. This method requires approximately $7n$ single/double precision words of working space to be provided by the user.

The full quasi-Newton option in CONMIN is the BFGS algorithm with initial scaling, documented in Ref. 32. This method requires approximately $n^2/2+11n/2$ [i.e., $O(n^2)$] double-precision words of working storage.

For solving large-scale nonlinear optimization problems, memory considerations generally mandate using the conjugate-gradient algorithm; i.e., we used only the memoryless quasi-Newton-like conjugate-gradient option of CONMIN, requiring an $O(n)$ working storage (7 vectors of length n).

The CONMIN subroutine was modified so as to maximize the vectorization of its code on the CYBER 205 vector supercomputer. As will be shown in the next section, the performance of this conjugate-gradient code can be improved significantly by careful implementation on supercomputers when solving large-scale nonlinear optimization problems.

4.1. Description of the Shanno–Phua Conjugate Gradient Method (Ref. 20).

Step 1. Initialization. Choose x_0, ϵ, $H_0=I$; set $k=0$; compute

$$f_k=f(x_k), \tag{19a}$$

$$g_k=g(x_k), \tag{19b}$$

$$s_k=-g_k, \tag{19c}$$

$$s_k^T g_k=-g_k^T g_k. \tag{19d}$$

Step 2. Linear Search Procedure. In this step, we perform the inexact linear search procedure, proposed in Ref. 31, with some modifications. As shown in Refs. 30 and 31, inexact linear searches are preferable to exact searches, particularly for the memoryless quasi-Newton method with Beale restarts. The basic linear search uses Davidon's cubic interpolation to find a steplength α_k which satisfies the following two conditions:

$$f(x_k+\alpha_k s_k)<f(x_k)+0.0001\alpha_k s_k^T g_k, \tag{20}$$

$$|s_k^T g(x_k+\alpha_k s_k) s_k^T g_k|<0.9. \tag{21}$$

Step 3. Test for Convergence. Set

$$x_{k+1}=x_k+\alpha_k s_k, \tag{22a}$$

$$f_{k+1}=f(x_{k+1}), \tag{22b}$$

$$g_{k+1}=g(x_{k+1}), \tag{22c}$$

$$p_k=x_{k+1}-x_k, \tag{22d}$$

$$y_k=g_{k+1}-g_k. \tag{22e}$$

If

$$\|g_{k+1}\| \leq \epsilon \max(1, \|x_{k+1}\|),$$

then stop. Else, proceed to Step 4. In our case, the gradient is obtained by numerical differentiation, as is evident from the fact that we use a finite-difference discretization.

Step 4. Beale Restart According to Powell's Criterion. If the criterion suggested in Ref. 33 holds, then perform the restart procedure of Ref. 37, described in this step. Otherwise, proceed to Step 5. The restart criteria of Ref. 33 are the following:

(a) the iteration k is a multiple of n; (23a)

(b) $|g_{k+1}^T g_k| \geq \|g_{k+1}\|^2$. (23b)

If either one of the above two conditions holds, then compute the new search direction s_{k+1} by

$$s_{k+1} = \gamma g_{k+1} - \left[\frac{(1+\gamma y_k^T y_k}{p_k^T y_k}\frac{p_k^T g_{k+1}}{p_k^T y_k} - \frac{\gamma y_k^T g_{k+1}}{p_k^T y_k}\right] p_k + \frac{\gamma p_k^T g_{k+1}}{p_k^T y_k} y_k, \tag{24}$$

where

$$\gamma = p_k^T y_k / y_k^T y_k. \tag{25}$$

Set $p_t = s_k$, $y_t = y_k$, and go to Step 2.

Step 5. New Search Direction by the Two-Step Memoryless BFGS Formula. This is a nonrestart step in which we compute the new search direction by using the two-step memoryless BFGS scheme as suggested in Ref. 30. That is, we compute s_{k+1} by

$$s_{k+1} = -\hat{H}_k g_{k+1} + \frac{p_k^T g_{k+1}}{p_k^T y_k} \hat{H} y_k - \left(1 + \frac{y_k^T \hat{H}_k y_k}{p_k^T y_k}\frac{p_k^T g_{k+1}}{p_k^T y_k} - \frac{y_k^T \hat{H}_k g_{k+1}}{p_k^T y_k}\right) p_k. \tag{26}$$

Here, $\hat{H}$ is an approximation to the inverse Hessian, and the vectors $\hat{H}_k g_{k+1}$ and $\hat{H}_k y_k$ are defined by

$$\hat{H}_k g_{k+1} = \frac{p_t^T y_t}{y_t^T y_t} g_{k+1} - \frac{p_t^T g_{k+1}}{y_t^T y_t} y_t + \left(\frac{p_t^T g_{k+1}}{p_t^T y_t} - \frac{y_t g_{k+1}}{y_t^T y_t}\right) p_t, \tag{27}$$

$$\hat{H}_k y_k = \frac{p_t^T y_t}{y_t^T y_t} y_k \frac{p_t^T y_k}{y_t^T y_t} y_t + \left(2\frac{p_t^T y_k}{p_t^T y_t} - \frac{y_t y_k}{y_t^T y_t}\right) p_t. \tag{28}$$

In this method H_{k+1}, the approximation to the inverse Hessian, is a matrix obtained by updating the identity matrix with a limited number of quasi-Newton corrections. The storage of an $(n \times n)$ matrix is avoided by storing only the vectors that define the rank-two corrections. Consequently, Ref.

30 calls the method the memoryless quasi-Newton method. See also Ref. 38 as well as Refs. 35 and 36.

As suggested in Ref. 39, the search vector s_{k+1} is scaled by

$$\hat{s}_{k+1}=(2(f_{k+1}-f_k)/g_{k+1}^T)s_{k+1}. \tag{29}$$

Go to Step 2.

3.2. Storage Requirements for CONMIN. From the description of the CONMIN algorithm (see also Ref. 20), it is evident that the implementation of this algorithm requires the storage of the following vectors:

x = current estimate of the minimum;
g = gradient evaluated at the current point;
s = current search direction;
x^* = new estimate of the minimum;
g^* = gradient evaluated at $x=x^*$;
s_r = Beale restart search direction;
y_r = Beale restart vector.

Notice that no extra storage is required to store the vector y, since this vector can be stored into the vector x^* after the vector x is replaced by x^*. Consequently the CONMIN subroutine requires $7n$ single/double precision real words of storage, in addition to the storage of various auxiliary scalar products.

3.3. Computational Complexity of the CONMIN Subroutine. Our practical experience (see also Section 4) showed that this conjugate-gradient algorithm required p function and gradient evaluations per iteration, with $2\le p\le 3$. The computational effort of function and gradient evaluation is problem dependent, but as a rule becomes the most expensive part of the conjugate-gradient algorithm as the number of variables increases, i.e., for large-scale unconstrained minimization.

As shown in Ref. 40, the basic formula for CPU time consumption in an optimization code is

$$T=t_fn_f+t_gn_g+t_in_i=t_f(n_f+nn_g)+t_in_i; \tag{30}$$

here, t_f and t_g are the times required per function and gradient calls, respectively; t_i is the average overhead execution time per iteration; n_f is the number of function evaluations; n_g is the number of gradient evaluations; $t_g=nt_f$; and n_i is the number of iterations.

The computational complexity of our test problems will be further discussed in the next section.

We shall now attempt to analyze the computational complexity of CONMIN in terms of the number of operations (multiplications and additions) required per iteration. For the above description, we notice that a

Table 1.

Step	Multiplications	Additions
2	pn	pn
3	n	$2n$
4	$7n$	$7n$
5	$19n$	$16n$

complete cycle of an iteration in CONMIN involves the execution of Steps 2-5. The number of operations required to perform each of these steps can be summarized in Table 1. Since $2 \leq p \leq 3$, due to the particular line-search method used (Ref. 20), CONMIN requires at most $3n$ multiplications and additions to perform Step 2. There are two distinct types of iterations in CONMIN, namely a restart iteration and a normal (nonrestart) iteration.

Each restart iteration involves the execution of steps 2-5, whereas a normal iteration consists of Steps 2, 3, 5. In summary, we have Table 2. In other words, the amount of operations required in performing a restart iteration of CONMIN is at most $10n$ multiplications and additions, whereas $20n$ additions and $22n$ multiplications are required to perform a normal iteration.

One may wish to find out how frequently a restart iteration is performed in comparison to a normal iteration. The runs of the CONMIN subroutine were closely investigated when it was applied to solve our problems. We found that in general a restart was being made every two or three iterations. It was extremely rare for a given direction to be used for more than ten iterations.

4. Vectorization Techniques

In this section, we describe the various steps taken to speed up the performance of the quasi-Newton conjugate-gradient algorithm for the CYBER 205 vector computer. Because of its memory-to-memory architecture, the CYBER 205 has a longer vector start-up time than, say, a register-to-register supercomputer such as a CRAY X-MP. Hence, in order to achieve

Table 2.

Iteration	Multiplications	Additions
Restart	$(8+p)n$	$(8+p)n$
Normal	$(20+p)n$	$(18+p)n$

top performance, it becomes necessary to increase the vector length on the CYBER 205 to fairly long vectors. For the CYBER 205, the half-performance length is about 100, whereas on the CRAY X-MP it is around 10 elements. The half-performance length can be defined as the vector length needed to achieve one-half the asymptotic peak vector operation rate.

The conjugate-gradient algorithm involves two principal sections of code that consume most of the CPU time: (a) function and gradient evaluation; (b) actual minimization step, including the linear search.

We included routines from BLAS-2 and BLAS-3 in order to vectorize the vector inner products as well as using machine call Q8-SDOT.

As we shall see later, the ratio of the CPU times spent in (a) and (b) will vary from problem to problem, depending on several factors such as the complexity of the objective function, the number of independent variables, and the total number of degrees of freedom in the problem, among others.

As was evident from the results, the vectorization of the problem-independent parts of the minimization routine results in promising benefits only for large problems (a speed-up factor of 7 for the largest problem), whereas only a factor of 3 was obtained for the medium-size problems. For small problems, due to the slower start-up time of the CYBER-205, no benefit is to be expected. This could be different on a CRAY supercomputer.

As a first step, the minimization routine CONMIN was restructured so that all DO-loops could be vectorized. The bulk of the DO-loops in this routine perform inner product and summation operations. On the CYBER 205, the aforementioned tasks are initiated by a certain type of machine instructions called vector macros. Although both of these computations are reduction operations, they are vectorizable because of their hardware implementation.

The floating-point add and multiply units on the CYBER 205 have feedback connections for accumulative add or multiply operations. Additionally, the result from any of the functional units can be routed directly to the input of other units wihout stopping in some intermediate registers or referencing of memory. This process, known as short stopping, gives an effective stream rate of one result per cycle. The timing information for summation and inner product is as follows:

Q8-SSUM,	$96+N$ cycles,
Q8-SDOT,	$107+N$ cycles.

After the initial vector start-up time of 96 and 107 cycles, respectively, a new result becomes available after each cycle. Hence, the larger N, the lesser the impact of the start-up time on the final performance of the two operations.

Having vectorized the minimization routine, the next task is to vectorize the often computationally intensive FUNCT routine, which calculates the cost function and its corresponding gradients for subsequent use in CONMIN. The number of function and gradient evaluations within each iteration in CONMIN is dependent on the rate of convergence, the stepsize, and the number of restarts.

We have applied the conjugate-gradient method to two separate meteorological problems. However, for the sake of brevity, we will describe the vectorization techniques and detail the modifications for the Lamb wave problem only. The Lamb wave problem is a multi-dimensional boundary-value problem, and as a result the minimization is done only in the interior of the domain. This necessitates collapsing of the two-dimensional and three-dimensional arrays into one-dimensional arrays to achieve top performance on the CYBER 205. The computation of the cost function and the gradients can be vectorized over all three dimensions by collapsing the DO loops in the three spatial directions into a one-dimensional DO loop and making use of the control bit vectors. Such collapsing is done very efficiently in a bit-addressable computer such as the CYBER 205, with help of WHERE statements that enable us to mask the results along the boundary grid points by initializing those addresses to zero bits with Q8VMKO calls. Also, the largest loop range was made the innermost loop in those loops where collapsing was not possible due to the iterative nature of the computations.

5. Discussion of Numerical Results

By running the conjugate-gradient large-scale minimization, it became evident that the bulk of the CPU time was spent in the function and gradient evaluation procedures. This is particularly true for the scalar versions of the minimization code for the two large-scale problems tested in this study. As such, our vectorization effort was mainly directed toward the performance improvement of the function and gradient evaluation routines, while the rest of the code was vectorized by an automatic vectorizer procedure (VAST2) as well as by using adequate BLAS routines for linear algebra.

As a starting point of this effort, we began with automatic vectorization. Further improvement in the speed-up due to vectorization was achieved by using manual vectorization, hereby referred to as *supervectorization*. We found out that, in both problems, the improvement due to automatic vectorization was only marginal; only after performing manual vectorization was an impressive speed-up achieved. The manual vectorization included loop-collapsing, in-line machine calls, and eliminating or reordering code so as to allow for optimal vectorization.

Table 3. Speed-up ratios for the AUGLAG minimization problem.

	Quantity	Mesh	Funct	Minimiz	Total
(a)	Scalar to super vector ratio	46×46	64.8	3.35	21.00
(a)	Scalar to super vector ratio	23×23	31.9	2.97	15.03
(b)	Scalar to auto vector ratio	46×46	1.76	3.35	1.86
(b)	Scalar to auto vector ratio	23×23	1.72	2.91	1.80

(a) Speed-up ratio (ratio of corresponding CPU timings) between scalar code and manual refined vectorization performed after the code was initially vectorized by the automatic vectorizer procedure VAST2. The manual vectorization is referred to as supervectorization since it achieves a high percentage of vectorization (~90%).

(b) Speed-up ratio (ratio of corresponding CPU timings) between scalar code and code vectorized by the automatic vectorizer procedure VAST2.

In the first problem (AUGLAG), using a 46×46 mesh, the speed-up due to vectorization was a factor of almost 65 in the function and gradient evaluation routine. Since all the DO loops in the conjugate-gradient minimization routine (CONMIN) were already vectorized by the automatic vectorizing compiler, no further speed-up could be achieved for that routine. The net speed-up for the first minimization problem was an impressive factor of 21, as shown in Table 3.

For the coarser mesh version of AUGLAG (23×23 mesh), the improvement was relatively smaller. This is due to the fact that the CYBER 205 has a slower vector start-up time compared to the CRAY computers (see Fig. 1) and the performance efficiency of the CYBER 205 has a strong dependence on the vector length.

On the other hand, for the second problem treated in this study (GUSTAF), the speed-up due to vectorization was a factor of about 7 for

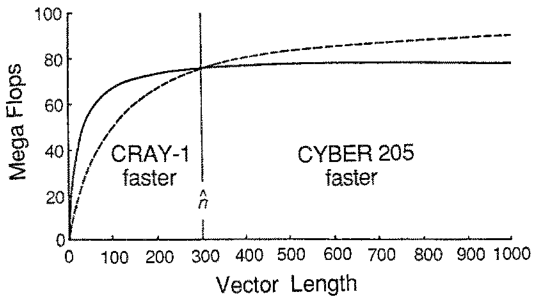

Fig. 1. Performance of the 2-pipe CYBER 205 (broken line) and the CRAY-1 (full line) as a function of vector length (see R. W. Hockney and C. R. Jesshope, *Parallel Computers: Architecture, Programming, and Algorithms*, Taylor and Francis, 1981).

Table 4. Speed-up ratios for the GUSTAF minimization problem.

Quantity	Mesh	Funct	Minimiz	Total
(a) Scalar to super vector ratio	111×150	6.71	7.03	6.73
(a) Scalar to super vector ratio	12×15	1.90	1.05	1.71
(b) Scalar to auto vector ratio	111×150	1.20	7.03	1.25
(b) Scalar to auto vector ratio	12×15	1.11	1.00	1.1

See Table 3 for Explanations (a) and (b).

the fine mesh case (111×150 mesh). This clearly reflects the problem-dependent nature of the computational cost for the function and gradient evaluation routines. The total speed-up for the second problem was also a factor of 7. For a very coarse mesh version of GUSTAF (12×15 mesh), the speed-up due to vectorization was only by a factor of less than 2, again reflecting on the longer breakeven point for vector computations on the CYBER 205 supercomputer. These results are also detailed in Table 4.

A more detailed breakdown of the computational cost and overheads associated with the minimization of the conjugate-gradient routine CONMIN and the function and gradient evaluations are illustrated in Tables 5-8. The relative percentages of CPU time spent in the various parts of the minimization program (namely, the minimization routine itself and the function and gradient evaluation routines) are depicted in Figs. 2-5.

Table 5. Timing details for the AUGLAG minimization problem (46×46 mesh) for various levels of vectorization (CPU times in sec).

	Vectorization level		
Quantity	Hand-vectorized	Automatic vectorization	Scalar
(c) Time spent in FUNCT	0.0334	1.2316	2.1649
(c) Time spent on FUNCT calls	0.1003	3.6949	6.4945
(d) Time spent in CONMIN (total)	0.3491	3.9436	7.3278
(e) Time spent in minimization	0.2487	0.2488	0.8333

(c) FUNCT is the subroutine where the function and gradient evaluations are performed.
(d) Total time spent in the conjugate-gradient unconstrained minimization subroutine CONMIN, which includes time spent in the subroutine FUNCT and in calls to subroutine FUNCT.
(e) Time spent in the conjugate-gradient unconstrained minimization subroutine CONMIN, excluding, however, time spent in function and gradient evaluation and/or calls to subroutine FUNCT.

Table 6. Timing details for the AUGLAG minimization problem (23×23 mesh) for various levels of vectorization (CPU times in sec).

Quantity	Vectorization level		
	Hand-vectorized	Automatic vectorization	Scalar
(c) Time spent in FUNCT	0.0131	0.2428	0.4178
(c) Time spent on FUNCT calls	0.0395	0.7291	1.2533
(d) Time spent in CONMIN (total)	0.0942	0.7849	1.4158
(e) Time spent in minimization	0.0547	0.0558	0.1625

See Table 5 for Explanations (c), (d), (e).

Table 7. Timing details for the GUSTAF minimization problem (111×150 mesh) for various levels of vectorization (CPU times in sec).

Quantity	Vectorization level		
	Hand-vectorized	Automatic vectorization	Scalar
(c) Time spent in FUNCT	0.08362	0.46648	0.56127
(d) Time spent in CONMIN (total)	0.08699	0.46985	0.58508
(e) Time spent in minimization	0.00337	0.00337	0.02371

See Table 5 for Explanations (c), (d), (e).

Table 8. Timing details for the GUSTAF minimization problem (12×15 mesh) for various levels of vectorization (CPU times in sec).

Quantity	Vectorization level		
	Hand-vectorized	Automatic vectorization	Scalar
(c) Time spent in FUNCT	0.00281	0.00481	0.00535
(d) Time spent in CONMIN (total)	0.00361	0.00564	0.00619
(e) Time spent in minimization	0.00080	0.00083	0.00084

See Table 5 for Explanations (c), (d), (e).

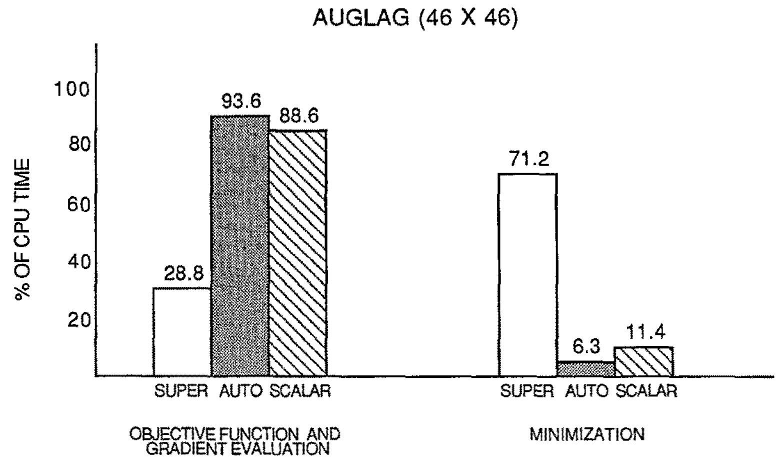

Fig. 2. Histograms of the relative percentages of scalar, automatic vectorization, and super (manual) vectorization code given as percentages of total CPU time spent in minimization code (AUGLAG problem, 46×46 mesh).

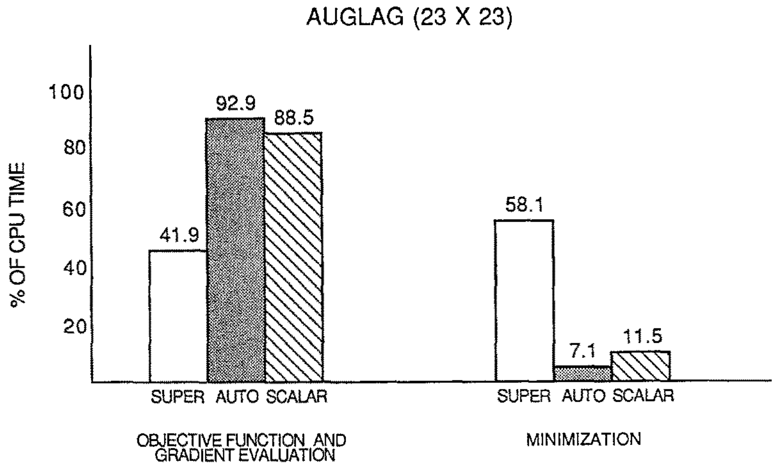

Fig. 3. Histograms of the relative percentages of scalar, automatic vectorization, and super (manual) vectorization code given as percentages of total CPU time spent in minimization code (AUGLAG problem, 23×23 mesh).

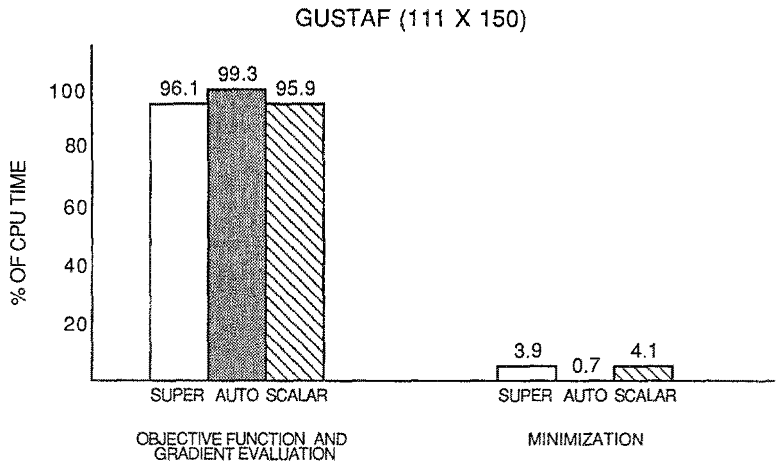

Fig. 4. Histograms of the relative percentages of scalar, automatic vectorization, and super (manual) vectorization code given as percentages of total CPU time spent in minimization code (GUSTAF problem, 111×150 mesh).

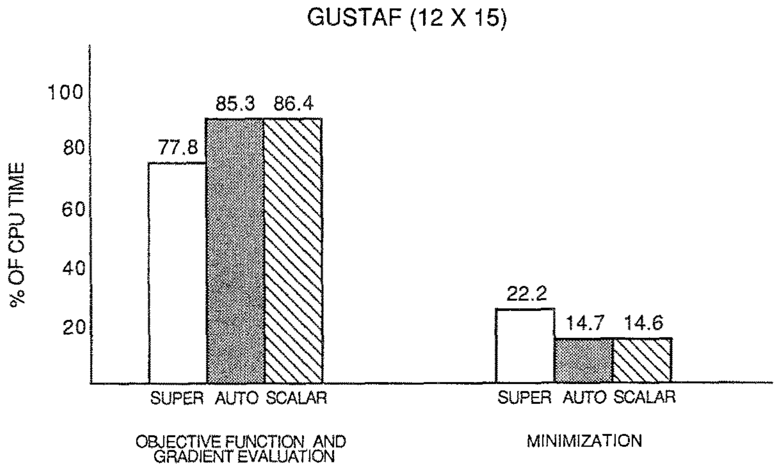

Fig. 5. Histograms of the relative percentages of scalar, automatic vectorization, and super (manual) vectorization code given as percentages of total CPU time spent in minimization code (GUSTAF problem, 12×15 mesh).

These figures show the relative percentages for the scalar code, the automatic vectorization code, and the super (manual) vectorization code, given as percentages of the total CPU time spent in the minimization code. For the first problem (AUGLAG), a reversal of the relative percentage of CPU time spent in the function and gradient evaluation routines versus the time spent in CONMIN, the conjugate-gradient minimization code, is noticed. This fact is even more evident in the fine mesh case (46×46 mesh) which involves minimization over a much longer vector ($46 \times 46 \times 10 \times 2 \approx 40{,}000$ variables).

In contrast, for the second problem (GUSTAF), the function and gradient evaluation routine dominates by far the computational cost. This is noticed for all three versions of the code (scalar, auto, supervector) and for both short and long vectors. Despite this fact, a speed-up factor of 7 was achieved due to hand vectorization for the entire minimization code.

6. Summary and Conclusions

Vectorization of the nonlinear conjugate-gradient method applied to large-scale unconstrained minimization problems on a CYBER 205 supercomputer has been presented in the present research. Using the timing routines of the CYBER 205 (SPY), it became evident that, for the large-scale meteorological minimization problems, the gradient and function evaluation routines dominate the CPU time spent in minimization. By performing automatic and then hand vectorization, we succeeded in achieving a sizable reduction in the CPU time required for finding the local minimum of nonlinear functions of 10^4 to 10^5 variables. This confirms the hypothesis (Ref. 12) that vector computers are advantageous in the case of large-scale unconstrained minimization.

With the application of optimal control methods in meteorology (see Ref. 41) for 4-D data assimilation, large-scale minimization in meteorology becomes a current and frequent problem, and the present approach points to the use of vector supercomputers in speeding up the solution of such problems. The speed-up is to some extent computer dependent, and the results are more impressive for large-scale problems where the number of variables is of the order of 10^4. The conjugate-gradient algorithm used in the present study is an optimized version of CONMIN (Ref. 20) and forms the basis of the modern quasi-Newton-like limited-memory conjugate-gradient methods such as the variable storage method of Refs. 35 and 36 and the E04DGF algorithm of the NAG library (Ref. 34) due to Ref. 38. This method has been found to be extremely robust in a variety of applications in meteorology (Ref. 19), oceanography (Ref. 42), and molecular dynamics in chemistry (Ref. 43).

Further research should concentrate on other applications of vectorization of large-scale unconstrained minimization problems using the nonlinear conjugate-gradient method such as in chemistry, structural optimization, and network optimization. While efforts are being pursued in the direction of parallelization of the nonlinear conjugate-gradient method, the present study points out the benefits of computational economy and speed-up that can be achieved for large-scale unconstrained minimization using vector supercomputers. In the future, we would like to extend this effort to take advantage of the multiprocessing capabilities of the newly introduced ETA^{10} Supercomputer and exploit the inherent parallelism of the nonlinear conjugate-gradient algorithm.

References

1. CHAZAN, D., and MIRANKER, W. L., *A Nongradient and Parallel Algorithm for Unconstrained Minimization,* SIAM Journal on Control, Vol. 8, pp. 207–217, 1970.
2. SLOBODA, F., *A Conjugate Direction Method and Its Applications,* Proceedings of the 8th IFIP Conference on Optimization Techniques, Wurtzburg, Germany, 1977; Springer-Verlag, Berlin, Germany, 1977.
3. SUTTI, C., *Nongradient Minimization Methods for Parallel Processing Computers, Part 1,* Journal of Optimization Theory and Applications, Vol. 39, pp. 465–474, 1983.
4. SUTTI, C., *Nongradient Minimization Methods for Parallel Processing Computers, Part 2,* Journal of Optimization Theory and Applications, Vol. 39, pp. 475–488, 1983.
5. POWELL, M. J. D., *An Efficient Method for Finding the Minimum of a Function of Several Variables without Calculating Derivatives,* Computer Journal, Vol. 7, pp. 155–162, 1964.
6. LOOTSMA, F. A., *State-of-the-Art in Parallel Unconstrained Optimization,* Parallel Computing 85, Edited by M. Feilmeir, E. Joubert, and V. Schendel, North-Holland, Amsterdam, Holland, pp. 157–163, 1986.
7. NAZARETH, L., *A Conjugate-Gradient Algorithm without Linear Searches,* Journal of Optimization Theory and Applications, Vol. 23, pp. 373–387, 1977.
8. DIXON, L. C. W., DUCKSBURY, P. G., and SING, P., *A Parallel Version of the Conjugate Gradient Algorithm for Finite-Element Problems,* Technical Report No. 1132, Numerical Optimization Centre, Hatfield Polytechnic, 1982.
9. DIXON, L. C. W., and PATEL, K. D., *The Place of Parallel Computation in Numerical Optimization, IV: Parallel Algorithms for Nonlinear Optimization,* Technical Report No. 125, Numerical Optimization Centre, Hatfield Polytechnic, 1982.
10. DIXON, L. C. W., PATEL, K. D., and DUCKSBURY, P. G., *Experience Running Optimization Algorithms on Parallel Processing Systems,* Technical Report No. 138, Numerical Optimization Centre, Hatfield Polytechnic, 1983.

11. PATEL, K. D., *Parallel Computation and Numerical Optimization*, Technical Report No. 129, Numerical Optimization Centre, Hatfield Polytechnic, 1982.
12. SCHNABEL, R. B., *Parallel Computing in Optimization*, Computational Mathematical Programming, Edited by K. Schittkowsky, Springer-Verlag, Berlin, Germany, pp. 357-381, 1985.
13. STRAETER, T. A., *A Parallel Variable-Metric Optimization Algorithm*, NASA, Technical Note No. D-7329, 1973.
14. HOUSOS, E. C., and WING, O., *Parallel Nonlinear Minimization by Conjugate Gradients*, Proceedings of the International Conference on Parallel Processing, Los Alamitos, California, pp. 157-158, 1980.
15. HOUSOS, E. C., and WING, O., *Pseudo-Conjugate Directions for the Solution of the Nonlinear Unconstrained Optimization Problem on a Parallel Computer*, Journal of Optimization Theory and Applications, Vol. 42, pp. 169-180, 1984.
16. MUKAI, H., *Parallel Algorithms for Solving Systems of Nonlinear Equations*, Computers and Mathematics with Applications, Vol. 7, pp. 235-250, 1981.
17. PIERRE, D. A., *A Nongradient Minimization Algorithm Having Parallel Structure with Implications for an Array Processor*, Computers and Electrical Engineering, Vol. 1, pp. 3-21, 1973.
18. VAN LAARHOVEN, P. J. M., *Parallel Variable-Metric Algorithms for Unconstrained Optimization*, Mathematical Programming, Vol. 33, pp. 68-81, 1985.
19. NAVON, I. M., and LEGLER, D. M., *Conjugate-Gradient Methods for Large-Scale Minimization in Meteorology*, Monthly Weather Review, Vol. 115, pp. 1479-1502, 1987.
20. SHANNO, D. F., and PHUA, K. H., *Remark on Algorithm 500*, ACM Transactions on Mathematical Software, Vol. 6, pp. 618-622, 1980.
21. LIU, D. C., and NOCEDAL, J., *On the Limited-Memory BFGS Method for Large-Scale Minimization*, Technical Report No. NAM-03, Department of Electrical Engineering and Computer Science, Northwestern University, 1988.
22. NAVON, I. M., and DE VILLIERS, R., *Combined Penalty-Multiplier Optimization Methods to Enforce Integral Invariants Conservation*, Monthly Weather Review, Vol. 111, pp. 1228-1243, 1983.
23. NAVON, I. M., and DE VILLIERS, R., *GUSTAF—A Quasi-Newton Nonlinear ADI FORTRAN IV Program for Solving the Shallow-Water Equations with Augmented Lagrangians*, Computers and Geosciences, Vol. 12, pp. 151-173, 1986.
24. NAVON, I. M., *FEUDX: A Two Stage, High-Accuracy, Finite-Element FORTRAN Program for Solving the Shallow-Water Equations*, Computers and Geosciences, Vol. 13, pp. 255-285, 1987.
25. BERTSEKAS, D. P., *Constrained Optimization and Lagrange Multiplier Methods*, Academic Press, New York, New York, 1982.
26. GUSTAFSSON, B., *An Alternating-Direction Implicit Method for Solving the Shallow-Water Equations*, Journal of Computational Physics, Vol. 7, pp. 239-254, 1971.
27. SASAKI, J., *Variational Design of Finite-Difference Schemes for Initial-Value Problems with an Integral Invariant*, Journal of Computational Physics, Vol. 21, pp. 270-278, 1976.

28. BERTSEKAS, D. P., *Combined Primal-Dual and Penalty Methods for Constrained Minimization*, SIAM Journal on Control and Optimization, Vol. 13, pp. 521-544, 1975.
29. RAMAMURTHY, M., and CARR, F., *Four-Dimensional Data Assimilation in the Monsoon Region, Part 1: Experiments with Wind Data*, Monthly Weather Review. Vol. 115, pp. 1678-1706, 1987.
30. SHANNO, D. F., *Conjugate-Gradient Methods with Inexact Searches*, Mathematics of Operations Research, Vol. 3, pp. 244-256, 1978.
31. SHANNO, D. F., *On the Convergence of a New Conjugate Gradient Method*, SIAM Journal on Numerical Analysis, Vol. 15, pp. 1247-1257, 1978.
32. SHANNO, D. F., and PHUA, K. H., *Matrix Conditioning and Nonlinear Optimization*, Mathematical Programming, Vol. 14, pp. 149-160, 1976.
33. POWELL, M. J. D., *Restart Procedures for the Conjugate Gradient Method*, Mathematical Programming, Vol. 12, pp. 241-254, 1977.
34. ANONYMOUS, N. N., *FORTRAN Library Reference Manual (Mark 12)*, Numerical Algorithm Group, E04DGF, Oxford, England, 1987.
35. BUCKLEY, A. G., and LENIR, A., *QN-Like Variable Storage Conjugate-Gradients*, Mathematical Programming, Vol. 27, pp. 155-175, 1983.
36. BUCKLEY, A. G., and LENIR, A., *Algorithm 630-BBVSCG: A Variable Storage Algorithm for Function Minimization*, ACM Transactions on Mathematical Software, Vol. 11, pp. 103-119, 1985.
37. BEALE, E. M. L., *A Derivation of Conjugate Gradients*, Numerical Methods for Nonlinear Optimization, Edited by F. A. Lootsma, Academic Press, London, England, pp. 39-43, 1972.
38. GILL, P. E., and MURRAY, W., *Conjugate-Gradient Methods for Large-Scale Optimization*, Technical Report No. SOL 79-15, Systems Optimization Laboratory, Department of Operations Research, Stanford University, 1979.
39. FLETCHER, R., *Conjugate Direction Methods*, Numerical Methods for Unconstrained Minimization, Edited by W. Murray, Academic Press, London, England, pp. 73-86, 1972.
40. LE, D., *A Fast and Robust Unconstrained Minimization Method Requiring Minimum Storage*, Mathematical Programming, Vol. 32, pp. 41-68, 1985.
41. LE DIMET, F. X., and TALAGRAND, O., *Variational Algorithms for Analysis and Assimilation of Meteorological Observations: Theoretical Aspects*, Tellus, Vol. 38A, pp. 97-110, 1986.
42. LEGLER, D. M., NAVON, I. M., and O'BRIEN, J. J., *Objective Analysis of Pseudostress over the Indian Ocean Using a Direct Minimization Approach*, Monthly Weather Review, Vol. 117, pp. 709-720, 1989.
43. ROBERTSON, D. H., BROWN, F. B., and NAVON, I. M., *Determination of the Structure of Mixed Argon-Xenon Clusters Using a Finite-Temperature, Lattice-Based Monte Carlo Method*. Journal of Chemical Physics, Vol. 90, pp. 3221-3229, 1989.