VECTORIZATION OF CONJUGATE-GRADIENT METHODS FOR LARGE-SCALE MINIMIZATION

I. M. Navon,
Supercomputer Computations Research Institute and Dept. of Mathematics
Florida State University
Tallahassee, FL 32306-4052

P. K. H. Phua,
Dept. of Info. Systems and Computer Science
National University of Singapore

and M. Ramamurthy
Dept. of Atmospheric Sciences
University of Illinois
Urbana-Champagne, IL 61801

ABSTRACT

Vectorization techniques are applied here for the vectorization of the non-linear conjugate-gradient method for large-scale unconstrained minimization. Until now the main thrust of vectorization techniques has been directed towards vectorization of linear conjugate-gradient methods designed to solve symmetric linear systems of algebraic equations. Computational results are presented using a robust limited-memory Quasi-Newton-like conjugate-gradient algorithm due to Shanno and Phua [21], applied to large-scale unconstrained minimization problems arising in meteorological applications. The vectorization of the non-linear conjugate-gradient method results in speed-ups up to a factor of 21 compared to the performance of the scalar code, when minimizing non-linear functions of $10^4 - 10^5$ variables. A sizable reduction in the CPU, time required for the minimization of large-scale non-linear functions is obtained-pointing to the advantages of vectorization for large-scale numerical unconstrained minimization problems, where local minima of non-linear functions is to be found using the non-linear conjugate-gradient method.

1. INTRODUCTION

During the last few years conjugate-gradient methods were found to be the best available tool for large-scale non-linear minimization of functions occurring in geophysical applications. We consider here the case when function evaluation is expensive and gradients are evaluated by finite differences. The objective function $F(x)$ is often described by a complex computer code and the derivatives of $F(x)$ are not available analytically. Generally in this case no higher derivatives are used. Therefore for large-scale unconstrained minimization with expensive function evaluations and in the absence of analytical gradients, each gradient evaluation requires N function evaluations.

The cost of solving the optimization problem i.e.

$$\begin{array}{c} \min F : R^n \rightarrow R^n \\ x \in R^n \end{array} \tag{1}$$

is dominated by the function and finite difference gradient evaluations.

In the present paper we present the results of the vectorization of the memoryless Quasi-Newton conjugate-gradient code due to Shanno and Phua [26] applied for finding the local minimizer of large-scale meteorological problems involving $10^4 - 10^5$ variables. In such cases it is not unusual for each evaluation of $F(x)$ to require up to a minute on a single processor of the CYBER-205. The results point to the vectorization of the function and gradient evaluation code, as well as the vectorization of the conjugate-gradient routine itself to result in a significant speed-up, i.e. in a sizable reduction in the CPU time required for the minimization.

Our approach may be then used in a multiple processor architecture using concurrent evaluations as proposed by Schnabel [20]. It is concluded that vector processors are advantageous for large-scale non-linear unconstrained minimization problems where the local minima of non-linear functionals is found using the conjugate-gradient method.

As for the non-linear conjugate-gradient method which constitutes the topic of the present research paper – a thorough review of the available literature points out to the fact that the absolute totality of the research activity carried by a small number of researchers was directed towards efforts in parallelizing the conjugate-gradient method – and to our best knowledge – no effort was directed towards vectorizing the method.

Parallelization of the non-linear 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 gridpoints along the search direction.

Several authors (Chazan and Miranker [2], Sloboda [22], Sutti [23,24], and Schendel and Schyska [19]) designed parallel versions of Powell's non-gradient method [15] 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 Lootsma [8]).

The Hatfield Polytechnic Group has investigated conjugate-gradient methods of Nazareth [12] which generate conjugate search directions without exact linear searches (see Dixon, Ducksbury and Sing [4], Dixon and Patel [3], Dixon, Patel, and Ducksbury [5], Patel [13], and Schnabel [18]). Other work on parallel optimization is reported in Straeter and Markos [26]. Housos and Wing [6,7] used pseudo-conjugate directions for the solution of the non-linear unconstrained op-

CH2617-9/88/0000/0410$01.00 © 1988 IEEE

timization problem on a parallel computer using the Powell [15] non-gradient method. Other efforts involved Mukai [9], Pierre [14], Van-Laarhoven [25], etc. No report appears to be available of speeding up the non-linear 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 by Schnabel [18] 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.

In Section 2 we will describe the relevant large-scale meteorological problems where the constrained non-linear 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 non-linear minimization. The robust (see Navon and Legler [11]) conjugate-gradient solver, its structure and computational complexity will be described in Section 3. Numerical results concerning the vectorization of the conjugate-gradient code and in particular the vectorization of the function/gradient evaluation 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-ups 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 non-linear conjugate-gradient code for a particular vector supercomputer, e.g., (the CYBER-205) will be discussed.

Section 6 will include a summary and conclusion remarks where implications for vectorization of different non-linear conjugate-gradient methods applied to large and very large-scale unconstrained minimization.

2. TYPICAL LARGE-SCALE METEOROLOGICAL PROBLEMS

To exemplify the large-scale problems encountered in meteorology we will briefly present two such problems.

A) ENFORCING CONSERVATION OF INTEGRAL INVARIANTS USING THE AUGMENTED-LAGRANGIAN METHOD

An Augmented-Lagrangian method is applied to enforce conservation of total mass, total energy and potential enstrophy for a limited-area model of the two dimensional shallow-water equations. The non-linear equality constrained problem is transformed in a series of unconstrained minimization problems (Bertsekas [1]).

The functional F is defined by:

$$\sum_{j=1}^{N_x}\sum_{k=1}^{N_y}\alpha(U-U_0)^2+\alpha(V-V_0)^2+\beta(H-H_0)^2 \qquad (2)$$

where $N_x\Delta x = L$ and $N_y\Delta y = D$, where L and D are the dimensions of the rectangular domain over which the shallow water equations are being solved.

The functional was solved subject to three equality constraints :

$$\begin{aligned} &E_n - E_0\\ &Z_n - Z_0\\ &H_n - H_0 \end{aligned} \qquad (3)$$

where n and 0 denote the values of the integral invariants at time $t_n = n\Delta t$ and $t_0 = 0$ respectively. Here the vector x has $3N_xN_y$ components where x is given by:

$$x = \left(u_{11},\ldots,u_{N_xN_y},v_{11},\ldots,v_{N_xN_y},h_{11},\ldots,h_{N_xN_y}\right)^T. \quad (4)$$

For a coarse resolution of a grid of 12×15 we had a function with 540 variables, i.e. the unconstrained minimization was carried out on a non-linear function of 540 variables.

A refined mesh test was also carried out for a mesh space increment of:

$$\Delta x = \Delta y = 40 \text{ km and } \Delta t = 360 \text{ sec} \qquad (5)$$

whereas the original grid had a mesh spacing of 400km and a time step of numerical integration of 3600 sec.

In the refined mesh test we had $150 \times 111 \times 3$ variables, i.e. 50000 variables in the non-linear unconstrained minimization.

2.1. Constrained adjustment to suppress Lamb waves

In meteorological applications one is often interested in suppressing external gravity waves in a 3-D model by modifying the observed wind field in such a way that the vertical motions at the lowest level of a three-dimensional atmospheric model vanish.

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 $\frac{dp_s}{dt}$ is zero everywhere and p_s is the surface pressure.

The continuity equation in pressure coordinates is given by

$$\frac{\partial u}{\partial x}+\frac{\partial v}{\partial y}+\frac{\partial w}{\partial p}=0. \qquad (6)$$

Integrating this equation from the top to the bottom of the atmosphere and assuming the vertical velocity $w = 0$ at both end points we obtain (see Ramamurthy [16], Ramamurthy and Carr [17]).

$$\int_0^{p_s} \left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} \right) dp = 0. \tag{7}$$

Using this equation as a constraint will ensure that

$$\frac{dp_s}{dt} = 0. \tag{8}$$

In other words, using the continuity equation as a strong constraint will enable us to suppress Lamb waves (Ramamurthy [16]).

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:

$$\begin{aligned} f = & \int_x \int_y \int_p \left[(u - \tilde{u})^2 + (v - \tilde{v}) \right]^2 dx\, dy\, dp \\ & + \int_x \int_y \left[\lambda \int_0^{P_s} \left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} \right) dp \right] dx\, dy, \end{aligned} \tag{9}$$

where $\tilde{u}$ and $\tilde{v}$ are the analyzed horizontal wind components - while u and v are the observed horizontal wind components - and λ is the Lagrange multiplier.

In a discrete Augmented-Lagrangian formulation we obtained

$$\begin{aligned} L = & \sum_i \sum_j \sum_k \Big[\left(u_{ijk} - \tilde{u}_{ijk} \right)^2 \\ & + \left(v_{ijk} - v_{ijk} \right)^2 \Big] \Delta x \Delta y \Delta p \\ & + \sum_i \sum_j \lambda_{ij} \Bigg[\sum_k \Bigg(\frac{u_{i+1,j,k} - u_{i-1,j,k}}{2\Delta x} \\ & + \frac{v_{i,j+1,k} - v_{i,j-1,k}}{2\Delta y} \Bigg) \Delta p \Bigg] \Delta x \Delta y \\ & + \frac{1}{2} \sum_i \sum_j C_{ij} \Bigg[\sum_k \Bigg(\frac{u_{i+1,j,k} - u_{i-1,j,k}}{2\Delta x} \\ & + \frac{v_{i,j+1,k} - v_{i,j-1,k}}{2\Delta y} \Bigg) \Delta p \Bigg]^2 \Delta x \Delta y. \end{aligned} \tag{10}$$

where

C_{ij} are the penalty terms.

λ_{ij} are the Lagrange multipliers.

Our model domain is rectangular with a resolution of (46×46) grid points while in the vertical we have 10 discrete levels resulting in this application in a function of $46 \times 46 \times 10 \times 2$ components *i.e.* ≈ 42320 components. A coarser mesh case where the mesh spacing was increased by a factor of 2 in the horizontal resulted in a function of $23 \times 23 \times 10 \times 2$ components ≈ 10000 variables. The gradient of the discrete Augmented-Lagrangian function L with respect to $\underline{x}$ where $\underline{x}$ is given by

$$\underline{x} = \left(u_{111}, \ldots, u_{N_x N_y N_p}, v_{111}, \ldots, v_{N_x N_y N_p} \right)^T \tag{11}$$

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

$$\begin{aligned} \left. \frac{\partial L}{\partial u} \right|_{ijk} = & 2 \left(u_{ijk} - \tilde{u}_{ijk} \right) \Delta x \Delta y \Delta p \\ & + \sum_k \Bigg(\frac{u_{i+1,j,k} - u_{i-1,j,k}}{2\Delta x} \\ & + \frac{v_{i,j+1,k} - v_{i,j-1,k}}{2\Delta y} \Bigg) \Delta p \\ & \cdot \left(\frac{C_{i-1,j} - C_{i+1,j}}{2\Delta x} \right) \Delta x \Delta y \end{aligned} \tag{12}$$

$$\begin{aligned} \left. \frac{\partial L}{\partial v} \right|_{ijk} = & 2 \left(v_{ijk} - \tilde{v}_{ijk} \right) \Delta x \Delta y \Delta p \\ & + \left(\frac{\lambda_{i,j-1} - \lambda_{i,j+1}}{2\Delta y} \right) \Delta x \Delta y \Delta p \\ & + \sum_k \Bigg(\frac{u_{i+1,j,k} - u_{i_1,j,k}}{2\Delta x} \\ & + \frac{v_{i,j+1,k} - v_{i,j-1,k}}{2\Delta y} \Bigg) \Delta p \\ & \cdot \left(\frac{c_{i,j-1} - C_{i,j+1}}{2\Delta y} \right) \Delta x \Delta y. \end{aligned} \tag{13}$$

The same inexact minimization of the Augmented-Lagrangian of Bertsekas [1] is applied using the same rules for updating the multipliers and penalties.

The same non-linear conjugate-gradient unconstrained minimization method, `CONMIN` [26], is used to minimize the Augmented-Lagrangian discrete functional.

3. THE LIMITED MEMORY QUASI-NEWTON CONJUGATE-GRADIENT METHOD

In our application we used the limited memory Quasi-Newton conjugate-gradient method of Shanno and Phua [26] implementing a restructured version of `CONMIN`, which was found to be robust and performing for a wide series of meteorological and geophysical applications [11]. This routine allows the user to use either a conjugate-gradient method or, if memory is available the Quasi-Newton BFGS algorithm. The Beale restarted memoryless Q.N. conjugate-gradient requires $7N$ single/double precision words of working space.

For solving our large-scale non-linear unconstrained optimization problems, memory considerations mandate using the conjugate-gradient method.

As shown by Le [10] the basic formula for CPU consumption in an optimization code is:

$$T = t_f(n_f + nn_g) + t_i n_i \tag{14}$$

where t_f and t_g are the times required for function and gradient calculation respectively, while t_i is the average overhead execution time per iteration.

Here n_f is the number of function evaluations, n_g is the number of gradient evaluations, where:

$$t_g = nt_g \tag{15}$$

and n_i is the number of iterations.

The computational cost of CONMIN depends on the number of function and gradient evaluations more than on the number of additions and multiplications within CONMIN itself which is about $20N$ additions and $22N$ multiplications for a normal iteration of the conjugate-gradient code.

The frequency of restart iterations in comparison with normal iterations was one to three and it was extremely rare for a given direction to be used for more than 10 iterations.

4. VECTORIZATION TECHNIQUES

In this section we describe the various steps taken to speed up the CONMIN conjugate-gradient algorithm for the CYBER-205 vector processor. 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 the CRAY X/MP. To achieve top performance it is necessary to increase the vector length on the CYBER-205 to fairly long vectors.Its half performance vector length is about 100. Most of the CPU consumption in the conjugate-gradient algorithm was in two sections :

a) Function and gradient evaluation,

b) actual minimization step.

The actual ratio between the two is problem dependent, depending on the complexity of the objective function, number of variables, etc. As a first step we restructured CONMIN so that all the DO loops could be vectorized, where the bulk of the DO loops perform inner products and summations. We used the Q8 SSUM and Q8 SDOT operations which have an initial vector start-up of 96 and 107 cycles respectively.

Hence the larger the N in the DO loop, the lesser the impact of the start-up time on the final performance of the two operations. Having vectorized the minimization routine, the main task is to vectorize the often heavy compute-intensive FUNCT routine which performs the function and gradient evaluation for subsequent use in CONMIN.

The number of function evaluations depends amongst others on the rate of convergence, the step-size requirements and the number of Beale restarts.

In general vectorization in FUNCT necessitated collapsing two and three dimensional arrays into one dimensional arrays to extract top performance from the CYBER-205.

Use of control bit vectors was also necessary.Such collapsing is done very efficiently on the bit addressable CYBER-205 with the help of WHERE statements that enable us to mask the results along the boundary grid points by pre-initializing those addresses to zero bits with Q8 VMKO 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 timing the runs of the conjugate-gradient large-scale minimization code (using SPY for instance) it became immediately evident that effectively the bulk of the CPU time was spent in the FUNCT routine for function and gradient evaluation.

As such our vectorization effort was mainly directed towards performance improvement of the FUNCT routine. We first applied automatic vectorization (such as VAST-2) for instance.

Further improvement in the speed-up due to vectorization was achieved by using manual vectorization hereby referred to as supervectorization.

In both large-scale problems the improvement due to automatic vectorization was rather marginal-and it was only after performing manual vectorization that impressive speed-ups were achieved. In the first problem (AUGLAG) the speed-up due to vectorization was a factor of almost 65 in the FUNCT routine. The net speed-up for this problem was an impressive factor of 21. For the coarser mesh version of AUGLAG (23×23) the improvement was already reduced since the CYBER-205 has a slower vector start-up time compared to the CRAY computer and the performance efficiency of the CYBER-205 has a strong dependence on the vector length.

On the other hand for the conservation of integral invariants treated in this study only a factor of about 7 speed-up was attained for the fine mesh (111×115). This clearly reflects the problem dependent nature of the computational cost for the gradient and function evaluation routines.

The total speed-up for this problem was also a factor of 7. For a very coarse mesh version of GUSTAF (12×15) the speed-up due to the vectorization was only by a factor of about 2, again reflecting on the longer break-even point for vector computations on the CYBER-205 supercomputer.

A more detailed break-down of the computational cost and the overheads associated with the large-scale minimization using CONMIN are illustrated in the following Tables. The relative percentages of the CPU time spent in the various parts of the minimization code (namely CONMIN itself and the function and gradient evaluation routines) are depicted in the accompanying figures.

These figures display the relative percentages for scalar code, automatic vectorization and super (manual) vectorization 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 FUNCT –the function and gradient evaluation routine versus the CPU time spent in CONMIN itself is noticed.

This fact is even more evident in the fine mesh case ($46 \times 46 \times 10$) which involves minimization over a very long vector (40000 variables).

In contrast for the second problem (GUSTAF), the function and gradient evaluation routine dominates by far the bulk of the computational cost, and this trend persists in all three versions of the code (scalar, auto and supervector) for both short and long vectors. However a speed-up factor of 7 was achieved due to manual vectorization for the entire minimization code.

6. SUMMARY AND CONCLUSIONS

Vectorization of the non linear limited memory Q.N. like conjugate-gradient method applied to large-scale minimization problems using the CYBER-205 vector processor has been implemented. For the problems considered the function and gradient evaluation 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 non-linear functions of $10^4 - 10^5$ variables.

This confirms the Schnabel [20] hypothesis that vector computers are advantageous in the case of large-scale unconstrained optimization. This approach could prove vital in further applications in meteorology, for instance for the adjoint 4-D data assimilation problem where each iteration of the conjugate-gradient method requires 2-3 model integrations for the length of the assimilation period and the application is not easily amenable to concurrent calculations. The speed-up is, however, both problem and also computer dependent and the results are more impressive for large-scale

Problems where the number of variables is of the order of 10^4. Use of multiple vector processors would allow multiple evaluations of $F(x)$ to be performed concurrently with each evaluation performed optimally by a vector processor. This is the case for ETA^{10} and the ALLIANT.

REFERENCES

[1] D. P. Bertsekas, "Constrained Optimization and Lagrange Multiplier Methods," Academic Press, 395pp, 1982.

[2] D. Chazan and W. L. Miranker, "A nongradient and parallel algorithm for unconstrained minimization," *SIAM Jour. on Control*, **8**, 207–217, 1970.

[3] L. C. W. Dixon and K. Patel, "The place of parallel computation in numerical optimization." IV Parallel algorithms for nonlinear optimization, Tech. Rep. 125, Numerical Optimization Centre, Hatfield Polytechnic, 1982.

[4] L. C. W. Dixon, P. G. Ducksbury, and P. Sing, "A parallel version of the conjugate-gradient algorithm for finite-element problems," Tech. Rep. 1132, Numerical Optimization Centre, Hatfield Polytechnic, 1982.

[5] L. C. W. Dixon, K. D. Patel, and P. G. Ducksbury, "Experience running optimization algorithms on parallel processing systems," NOC Tech. Rep. 138, Numerical Optimization Centre, Hatfield Polytechnic, 1983.

[6] E. C. Housos and O. Wing, "Parallel nonlinear minimization by conjugate-gradient " in Proceedings of the International Conference on Parallel Processing, Los Alamitos, IEEE proceedings, 157–158, 1980.

[7] E. C. Housos and O. Wing, "Pseudo-conjugate directions for the solution of the nonlinear unconstrained optimization problem on a parallel computer," *Jour. Optim. Theory and Appls.* **42**, 2, 169–180, 1984.

[8] F. A. Lootsma, "State-of-the-art in parallel unconstrained optimization" in Parallel Computing 85, M. Feilmeir, E. Joubert and V. Schendel (*eds.*), North-Holland, 157–163, 1986.

[9] H. Mukai, "Parallel algorithms for unconstrained optimization" in Proceedings of the 18th IEEE Conference on Decision and Control, 1 451–454, 1979.

[10] D. Le, "A fast and Robust Unconstrained Minimization Method Requiring Minimum Storage," Math Prog., **32**, 41–68, 1985.

[11] I. M. Navon and D. M. Legler, "Conjugate-gradient methods for large-scale minimization in meteorology," *Monthly Weather Review*, **115**, 1479–1502, 1987.

[12] L. Nazareth, "A conjugate-gradient algorithm without linear searches," JOTA **23**, 373–387, 1977.

[13] K. D. Patel, "Parallel computation and numerical optimization," Tech. Rep. No. 129, Numerical Optimization Centre, Hatfield Polytechnic, 1982.

[14] D. A. Pierre, "A nongradient minimization algorithm having parallel structure with implications for an array processor," *Computers and Electrical Engineering*, **1**, 3-21, 1973.

[15] M. J. D. Powell, "An efficient method for finding the minimum of a function of several variables without calculating derivatives," *Computer Jour.*, **7**, 155–162, 1964.

[16] M. Ramamurthy and F. Carr, "Four-dimensional data assimilation in the monsoon region, Part 1: Experiments with wind data," *Mon. Weath. Rev.*, **115**, 1987.

[17] M. Ramamurthy, "Four-dimensional data assimilation in a limited area model for the Monsoon Region," Ph. D. dissertation, University of Oklahoma, Norman, OK, 308 pp., 1986.

[18] R. B. Schnabel, "Parallel computing in optimization," *Computational Mathematical Programming*, **15**, Nato ASI Series F, K. Schittkowsky (*ed.*), Spring-Verlag, 357–381, 451 pp., 1985.

[19] U. Schendel and M. Schyska, "Parallelle Algorithmen in der Nicht-linearen Optimierung," Preprint 161/84. Fachbereich Mathematik, Freie Universitatät Berlin, 1984.

[20] R. B. Schnabel, "Concurrent Function Evaluations in Local and Global Optimization," *Comp. Methods in Appl Mech. Eng.*, **64**, 537–552, 1987.

[21] D. F. Shanno and K. H. Phua, "Matrix conditioning and nonlinear optimization," *Math. Program*, **14**, 149–160, 1978.

[22] F. Sloboda, "A conjugate direction method and its applications," Proceedings 8th IFIP Conference on Optimization Techniques, Wurtzburg, Springer-Verlag, 1977.

[23] C. Sutti, "Nongradient minimization methods for parallel processing computers, Part 1," JOTA, **39**, 4, 465–474, 1983.

[24] C. Sutti, "Nongradient minimization methods for parallel processing computers, Part 2," JOTA, **39**, 4, 475–488, 1983.

[25] P. J. M. Van Laarhoven, "Parallel variable metric algorithms for unconstrained optimization," *Math. Programming*, **33**, 68–81, 1985.

[26] T. A. Straeter and A. T. Markos, "A parallel Jacobsen-Okwan Optimization Algorithm," *NASA Tech. Note*, **D-8020**, NASA-Langley, VA, 1975.

[27] D. F. Shanno, K. H. Phua, "Remark on Algorithm 500," *ACM Trans. Math. Software*, **6**, 4, 618–622, 1980.

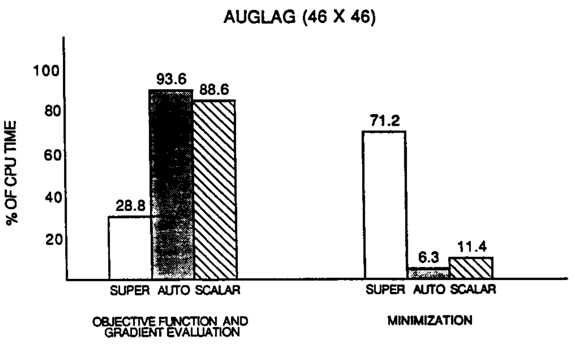


Figure 2. Histograms of the relative percentages of scalar, automatic vectorization and super (manual) vectorization code given as percentages of total CPU time spent in minimization code (AUGLAG (46 × 46).

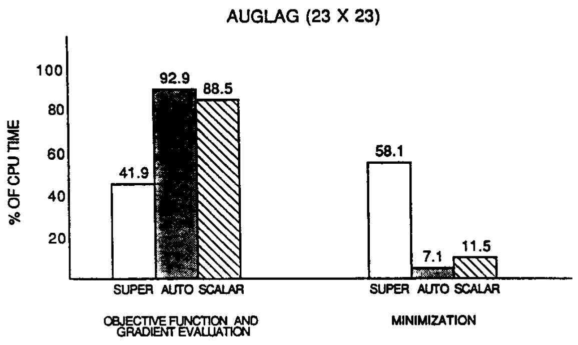
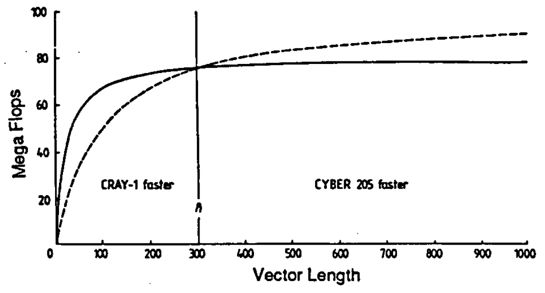


Figure 3. Same as Fig. 2 except for AUGLAG (23 × 23)



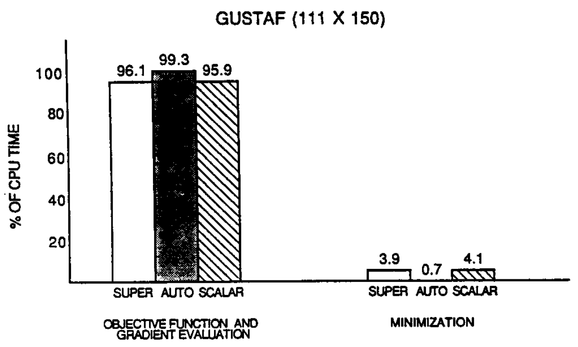


Figure 1. Performance of the 2-pipe CYBER 205 (broken line) and the CRAY-1 (full line) as a function of vector length (Source Hockney and Jesshope, Parallel Computers: Architecture, programming and algorithms, 1981.)

Figure 4. Same as Fig. 2 except for GUSTAF (111 × 150)

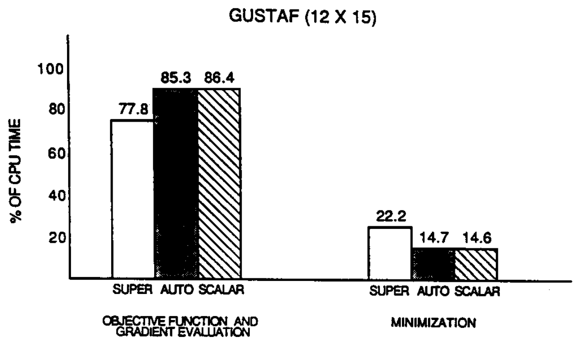


Figure 5. Same as Fig. 2 except for GUSTAF (12 × 15)

Table I: Speed-up ratios for the AUGLAG minimization problem

AUGLAG		FUNCT	MINIMIZ.	TOTAL
scalar to super vector ratio	(46 x 46)	64.8	3.35	21.00
scalar to super vector ratio	(23 x 23)	31.9	2.97	15.03
scalar to auto vector ratio	(46 x 46)	1.76	3.35	1.86
scalar to auto vector ratio	(23 x 23)	1.72	2.91	1.80

Table II: Speed-up ratios for the GUSTAF minimization problem

GUSTAF		FUNCT	MINIMIZ.	TOTAL
scalar to super vector ratio	(111 x 150)	6.71	7.03	6.73
scalar to super vector ratio	(12 x 15)	1.90	1.05	1.71
scalar to auto vector ratio	(111 x 150)	1.20	7.03	1.25
scalar to auto vector ratio	(12 x 15)	1.11	1.00	1.1

Table III: Timing details for the AUGLAG (46 $\times$ 46) minimization problem for various levels of vectorization

AUGLAG (46 x 46) - Hand Vectorized	
Time spent in FUNCT	= 0.0334 secs.
Time spent on FUNCT calls	= 0.1003 secs.
Time spent in CONMIN (total)	= 0.3491 secs.
Time spent in minimization	= 0.2487 secs.
AUGLAG (46 x46) - Automatic Vectorization	
Time spent in FUNCT	= 1.2316 secs.
Time spent on FUNCT calls	= 3.6949 secs.
Time spent in CONMIN (total)	= 3.9436 secs.
Time spent in minimization	= 0.2488 secs.
AUGLAG (46 x 46) - Scalar	
Time spent in FUNCT	= 2.1649 secs.
Time spent on FUNCT calls	= 6.4945 secs.
Time spent in CONMIN (total)	= 7.3278 secs.
Time spent in minimization	= 0.8333 secs.

Table IV: Timing details for the AUGLAG (23 $\times$ 23) minimization problem for various levels of vectorization

AUGLAG (23 x 23) - Hand Vectorized	
Time spent in FUNCT	= 0.0131 secs.
Time spent on FUNCT calls	= 0.0395 secs.
Time spent in CONMIN (total)	= 0.0942 secs.
Time spent in minimization	= 0.0547 secs.
AUGLAG (23 x 23) - Automatic Vectorization	
Time spent in FUNCT	= 0.2428 secs.
Time spent on FUNCT calls	= 0.7291 secs.
Time spent in CONMIN (total)	= 0.7849 secs.
Time spent in minimization	= 0.0558 secs.
AUGLAG (23 x 23) - Scalar	
Time spent in FUNCT	= 0.4178 secs.
Time spent on FUNCT calls	= 1.2533 secs.
Time spent in CONMIN (total)	= 1.4158 secs.
Time spent in minimization	= 0.1625 secs.

Table V: Timing details for the GUSTAF (111 × 150) minimization problem for various levels of vectorizatio

GUSTAF (111 x 150) - Hand Vectorized	
Time spent in FUNCT	= 0.08362 secs.
Time spent in CONMIN (total)	= 0.08699 secs.
Time spent in minimization	= 0.00337 secs.
GUSTAF (111 x 150) - Automatic Vectorization	
Time spent in FUNCT	= 0.46648 secs.
Time spent in CONMIN (total)	= 0.46985 secs.
Time spent in minimization	= 0.00337 secs.
GUSTAF (111 x 150) - scalar	
Time spent in FUNCT	= 0.56127 secs.
Time spent in CONMIN (total)	= 0.58508 secs.
Time spent in minimization	= 0.02371 secs.

Table VI: Timing details for the GUSTAF (12 × 15) minimization problem for various levels of vectorization

GUSTAF (12 x 15) - Hand Vectorized	
Time spent in FUNCT	= 0.00281secs.
Time spent in CONMIN (total)	= 0.00361 secs.
Time spent in minimization	= 0.00080 secs.
GUSTAF (12 x 15) - Automatic Vectorization	
Time spent in FUNCT	= 0.00481 secs.
Time spent in CONMIN (total)	= 0.00564 secs.
Time spent in minimization	= 0.00083 secs.
GUSTAF (12 x 15) - Scalar	
Time spent in FUNCT	= 0.00535 secs.
Time spent in CONMIN (total)	= 0.00619 secs.
Time spent in minimization	= 0.00084 secs.