

Parallel Algorithms for Large-scale Nonlinear Optimization

PAUL KANG-HOH PHUA, WEIGUO FAN* and YUELIN ZENG†

Department of Information Systems and Computer Science, National University of Singapore, Lower Kent Ridge Road, Singapore, 119260 Singapore

Multi-step, multi-directional parallel variable metric (PVM) methods for unconstrained optimization problems are presented in this paper. These algorithms generate several VM directions at each iteration, different line search and scaling strategies are then applied in parallel along each search direction. In comparison to some serial VM methods, computational results show that a reduction of 200% or more in terms of number of iterations and function/gradient evaluations respectively could be achieved by the new parallel algorithm over a wide range of 63 test problems. In particular, when the complexity, or the size of the problem increases, greater savings could be achieved by the proposed parallel algorithm. In fact, the speedup factors gained by our PVM algorithms could be as high as 28 times for some test problems. © 1998 IFORS. Published by Elsevier Science Ltd.

1. INTRODUCTION

Variable metric (VM), or quasi-Newton (QN) methods for unconstrained optimization are a class of numerical techniques for solving the following problem

$$\min_x f(x) \quad (1.1)$$

where $f(x)$ is a nonlinear real-valued function and x is an n -dimensional real vector. At the k th iteration, an approximation point x_k and an $n \times n$ matrix H_k are available. The methods proceed by generating a sequence of approximation points via the equation

$$x_{k+1} = x_k + \alpha_k d_k \quad (1.2)$$

where $\alpha_k > 0$ is calculated to satisfy certain line search conditions and d_k is a descent direction.

One important feature of VM methods is the choice of the matrix H_k . The methods require H_k to be positive definite and satisfy the QN equation

$$H_{k+1} y_k = \lambda_k \delta_k, \quad \lambda_k > 0 \quad (1.3)$$

where $\delta_k = x_{k+1} - x_k$ and $y_k = \nabla f(x_{k+1}) - \nabla f(x_k)$. One of the best known VM methods is the BFGS method that was proposed independently by Broyden [4], Fletcher [8], Goldfarb [9] and Shanno [17]. The BFGS update is defined by the equation

$$H_{k+1} = H_k + \frac{1}{\delta_k^T y_k} \left(\left(1 + \frac{y_k^T H_k y_k}{\delta_k^T y_k} \right) \delta_k \delta_k^T - \delta_k y_k^T H_k - H_k y_k \delta_k^T \right) \quad (1.4)$$

with $\lambda_k \equiv 1$.

Based upon non-quadratic models, Biggs [1] first suggested a self-adjustable value for the parameter λ_k , that is,

$$\lambda_k = \frac{1}{t_k} \quad (1.5)$$

*E-mail: fanweigu@iscs.nus.sg.

†E-mail: zengyuel@iscs.nus.sg.

Correspondence: P. K.-H. Phua, Department of Information Systems and Computer Science, National University of Singapore, Lower Kent Ridge Rd, Singapore, 119260 Singapore (e-mail: phuakh@iscs.nus.sg).

where

$$t_k = \frac{6}{\delta_k^T y_k} (f(x_k) - f(x_{k+1}) + \delta_k^T g_{k+1}) - 2 \quad (1.6)$$

Hence, a modified BFGS update can be defined by

$$H_{k+1} = H_k + \frac{1}{\delta_k^T y_k} \left((\lambda_k + \frac{y_k^T H_k y_k}{\delta_k^T y_k}) \delta_k \delta_k^T - \delta_k y_k^T H_k - H_k y_k \delta_k^T \right). \quad (1.7)$$

A method that updates H_k by adding only a symmetric rank-one matrix was proposed by Davidon [6]. This symmetric rank-one (SR1) update is defined by

$$H_{k+1} = H_k + \frac{(\delta_k - H_k y_k)(\delta_k - H_k y_k)^T}{(\delta_k - H_k y_k)^T y_k}. \quad (1.8)$$

Numerical results on the performance of different updates can be found in Luksan [11].

Our interest here is the parallel extension to the VM methods which will suit for the solution of large-scale optimization problems. The parallelization of the VM methods has been considered by several researchers in the past decades. The earliest work was done for the symmetric rank-one method by Straeter [19] and later was modified by van Laarhoven [21]. At each step, this algorithm updates the quasi-Newton matrix along n independent directions by parallelly evaluating values of the function and the gradient at n points. For a positive definite quadratic function, it can be shown that the method converges in one iteration regardless of the initial starting point. Computational results in [21] for a set of well-known problems with no more than four variables indicates that the algorithm improves the total number of parallel function evaluations by a factor of 2.5 and the total number of iterations by a factor of 1.5. However, it is assumed that as many processors as needed are available to perform all the computations that can be done in parallel. This may not be feasible for large problems that arise in many applications. Also, the numerical viability of the method for larger problems remains to be investigated. An excellent summary of this method as well as other parallel methods for general nonlinear optimization problems is provided by Lootsma and Ragsdell [10].

A more straightforward approach to parallelize the VM methods is to approximate the derivatives of the function $f(x)$ by using finite difference when the calculation of the exact gradient of the function is prohibitively expensive or is not analytically available. Schnabel [16] suggested to concurrently compute the function values needed by both the gradient approximation and linear search on parallel computers. As the dimensions of the problem grow larger, for a fixed number of processors, this finite difference method will result in a speedup factor that is approaching optimal. Further, in Ref. [5], Byrd *et al.* incorporated the columns of the finite difference Hessian into the BFGS update. Two ways of this incorporation were described and the convergence of the resulting algorithms was established in Ref. [5].

In this paper, we consider a class of multi-step, multi-directional parallel VM methods. A detailed description of these techniques is presented in Section 2. Section 3 presents numerical results obtained by applying the proposed parallel algorithms in solving a wide range of standard test problems. For comparison purposes, numerical results obtained by various serial VM methods are also included in this section. Discussions and conclusions are given in Section 4.

2. PARALLEL VARIABLE-METRIC ALGORITHMS

Consider the class of VM updates (see Shanno [17]) given by

$$H^* = H + t \frac{\delta \delta^T}{\delta^T y} + \frac{[(1-t)\delta - Hy][(1-t)\delta - Hy]^T}{[(1-t)\delta - Hy]^T y}, \quad (2.1)$$

where t is the scalar parameter, and the subscripts have been omitted by dropping (k) and replacing ($k+1$) by the subscript (*). It is evident that the SR1 and BFGS updates defined by equations (1.8) and (1.4) corresponds to $t = 0$ and $t = \infty$ respectively in equation (2.1).

Consequently, we shall denote the SR1 and BFGS updates by $H^*(0)$ and $H^*(\infty)$ respectively. By the same analogy, the Biggs' update given in equations (1.5), (1.6) and (1.7) will be denoted by $H^*(t_k)$.

In practice, we observe from our numerical results and results of many other papers that a particular VM algorithm may be "good" in solving certain types of minimization problems, but its efficiency degenerates when it is applied to solve other categories of problems (see Phua and Setiono [15], for instance). In particular, we notice that when the SR1 update (without modifications) is applied to solve a practical problem, two things are likely to happen. First, if it solves that problem at all, its efficiency is usually better than the other VM updates, such as DFP and BFGS, (see Phua and Chew [14], for instance). Second, it may fail to solve the problem due to the difficulties mentioned above for this update. Any subsequent modifications made to this update may force it to solve the problem, but its efficiency degenerates.

For this reason, we propose in this paper some parallel algorithms for VM methods so that when these new algorithms are applied to solve practical problems, the parallel mechanism will be able to explore different search directions generated by various VM updates during the minimization process. In addition, different line search strategies will be employed simultaneously in the process of locating the line minimum along each direction. This leads us to the following parallel VM algorithm.

2.1. PVM (parallel variable metric) algorithm

2.1.1. *Initialization.* Let $k := 0$, and x_0 be the initial guess of the minimum and $H_0 = I$ be the identity matrix. Let $\epsilon > 0$ be the required accuracy.

2.1.2. *Compute the function and gradient values at x_k .* Let

$$f_k := f(x_k)$$

and

$$g_k := \nabla f(x_k)$$

2.1.3. *Compute the parallel search directions.* Let $m_1 > 0$ be the number of processors available for computing the search directions in parallel. Compute

$$d_k^{(j)} = -H_k(t_j)g_k, \quad j = 1, 2, \dots, m_1 \quad (2.2)$$

For instance, t_j can be chosen as 0, 1, ∞ in equation (2.1), t_k in equations (1.5), (1.6) and (1.7), or any other appropriate values.

2.1.4. *Apply the parallel line search algorithm.* Call the PLS routine (see below) to perform the line search procedure in parallel along each search direction $d_k^{(j)}$, $j = 1, 2, \dots, m_1$. Terminate the execution of this routine once a point α_k has been found satisfying the following Wolfe conditions:

$$f(x_k + \alpha_k d_k^{(j)}) - f(x_k) \leq 0.0001 \times \alpha_k g_k^T d_k^{(j)} \quad (2.3)$$

and

$$\nabla f(x_k + \alpha_k d_k^{(j)})^T d_k^{(j)} \geq 0.9 \times g_k^T d_k^{(j)} \quad (2.4)$$

along any search direction $d_k^{(j)}$. Let d_k^* be the search direction such that α_k has been found successfully, and we shall denote this point as a successful point. If successful points are found from more than one search direction, then d_k^* is chosen to be the search direction that attains the lowest function value. The Wolfe conditions equations (2.3) and (2.4) will assure that certain QN updates, such as the BFGS update, will be positive definite, and hence descent directions will be generated by these updates. Therefore the BFGS update is suggested in step 3. In step 4, at least one descent direction must be included in the set of m_1 parallel search directions. In any

case, if no successful points could be found in this step, d_k^* will be chosen as the direction which attains the lowest function value.

2.1.5. *Compute the new point.* Let

$$x_{k+1} := x_k + \alpha_k d_k^*, \quad g_{k+1} := \nabla f(x_{k+1})$$

2.1.6. *Test for convergence.* If $\|g_{k+1}\| \leq \epsilon \cdot \max\{1, \|x_{k+1}\|\}$, then stop; otherwise, proceed to step 7.

2.1.7. *Compute the new VM updates.* Let

$$H_{k+1} := H_{k+1}(\infty)$$

That is, update the approximate inverse Hessian matrix H_k by using the BFGS update defined in equation (1.7). Repeat the whole process from step 2. The parallel line search algorithm will be described as follows.

2.2. PLS (parallel line search) routine

Let m_2 be the number of parallel processors available for locating the line minimum along a particular search direction $d_k^{(j)}$ ($j = 1, 2, \dots, m_1$). Let α_{\max} be the maximum allowable step-size, and denote $\psi(x) = f(x_k + \alpha d_k^{(j)})$. The main iteration of this routine consists of the following steps:

2.2.1. *Choose the step sizes.* Let

$$0 < \alpha^{(i)} < \alpha_{\max}, \quad i = 1, 2, \dots, m_2$$

where $\alpha^{(1)} < \alpha^{(2)} < \dots < \alpha^{(m_2)}$ are m_2 different approximately chosen step sizes. For instance, we may choose $\alpha^{(1)} = 0.5$, $\alpha^{(2)} = 1.0$, \dots , $\alpha^{(m_2)} = \alpha_{\max}$. Let Φ be the set of these step sizes.

2.2.2. *Compute the function/gradient values concurrently.* For $i = 1, 2, \dots, m_2$, compute concurrently

$$x_k^{(i)} = x_k + \alpha^{(i)} d_k^{(j)}, \quad f_k^{(i)} = f(x_k^{(i)}), \quad g_k^{(i)} = \nabla f(x_k^{(i)})$$

2.2.3. *Test for successful points.* Let Φ^* be the set of step sizes $\alpha^{(i)}$ such that for each $\alpha^{(i)} \in \Phi^*$, $\alpha^{(i)}$ satisfies the Wolfe conditions equations (2.3) and (2.4). If $\Phi^* \neq \phi$ (empty set) and $\alpha_k^{(i)} \in \Phi^*$ is the stepsize which corresponds to the minimum functional value, that is,

$$\psi(\alpha_k^{(i)}) = \min_{\alpha^{(i)} \in \Phi^*} f(x_k + \alpha^{(i)} d_k^{(j)})$$

then set $\alpha_k := \alpha_k^{(i)}$ and return to the main PVM routine; otherwise, proceed to step 4.

2.2.4. *Choose interpolation points.* Let Φ^+ be the set of stepsizes such that for each $\alpha^{(i)} \in \Phi^+$, $\alpha^{(i)}$ satisfies

$$d_k^{(j)T} g_k^{(i)} > 0.$$

Let $\Phi^- = \Phi - \Phi^+$. Choose $\alpha_1 \in \Phi^-$ such that

$$\psi(\alpha_1) = \min_{\alpha^{(i)} \in \Phi^-} f(x_k + \alpha^{(i)} d_k^{(j)})$$

and choose $\alpha_2 \in \Phi^+$ such that

$$\psi(\alpha_2) = \min_{\alpha^{(i)} \in \Phi^+} f(x_k + \alpha^{(i)} d_k^{(j)}).$$

If $\Phi^- = \phi$, then choose $\alpha_1 = 0$. If $\Phi^+ = \phi$, then choose $\alpha_2 = \alpha^{(m)}$, where $\alpha^{(m)} \in \Phi^-$ such that

$$\psi(\alpha^{(m)}) = \max_{\alpha^{(i)} \in \Phi^-} f(x_k + \alpha^{(i)} d_k^{(j)}).$$

2.2.5. Apply the cubic interpolation technique. Let $\varphi(\alpha)$ be the cubic spline passing through the two points α_1 and α_2 . Let α^* be the minimum of $\varphi(\alpha)$. Compute

$$\psi(\alpha^*) = f(x_k + \alpha^* d_k^{(j)})$$

and

$$h^* := \nabla \psi(\alpha^*).$$

If α^* satisfies the Wolfe conditions (equations (2.3) and (2.4)), then set $\alpha_k = \alpha^*$ and return to the main PVM routine. Otherwise, if $d_k^{(j)T} \cdot \nabla \psi(\alpha^*) > 0$, then α^* replaces α_1 ; if $d_k^{(j)T} \cdot \nabla \psi(\alpha^*) \leq 0$, then α^* replaces α_2 . Repeat step 5.

3. IMPLEMENTATION AND NUMERICAL RESULTS

To implement the proposed parallel VM methods, we have chosen three parallel processes ($m_1 = 3$) for computing the following VM directions concurrently in step 3 of the PVM routine

$$d_k^{(1)} = -H_k(0)g_k \quad (3.1)$$

$$d_k^{(2)} = -H_k(\infty)g_k \quad (3.2)$$

and

$$d_k^{(3)} = -H_k(t_k)g_k \quad (3.3)$$

That is, three parallel search directions are generated by SR1, BFGS and Biggs' updates defined in equation (1.8), equations (1.4), (1.5), (1.6) and (1.7) respectively. Similarly, three parallel processes ($m_2 = 3$) are chosen for the implementation of step 1 of the PLS (parallel line search) routine. In particular, $\alpha^{(1)} = \frac{1}{2}$, $\alpha^{(2)} = 1$, and $\alpha^{(3)} = 2$ are chosen here. We refer to this particular implementation as PVM3.3 algorithm.

In our implementation, the programs are all written in FORTRAN 90 language on top of the PVM (parallel virtual machine), which are run under the CRAY J916 supercomputing environment. We also use the master-slave computing model to implement the PVM3.3 main algorithm and the PLS routine. The main program acts as the "master", who controls the three "slaves". Each "slave" is delegated the task of performing the line search along a particular VM search direction.

To improve the performance of BFGS updates, the scaling strategy suggested by Oren [15] is adopted in our implementation. We note that Oren's scaling algorithms help to stabilize the performance of our algorithm as the size of the problem increases. The required accuracy is set as: $\epsilon = 10^{-5}$. That is, convergence is assumed if the following criterion is satisfied at the point x_k

$$\|g_k\| \leq 10^{-5} \times \max\{1, \|x_k\|\} \quad (3.4)$$

where $\|\cdot\|$ is the l_2 (Euclidean) norm. In practice, one may wish to adopt the convergence criterion as proposed by Dennis and Schnabel [7] on the relative gradient of f at $x = x^{(k)}$, that is, when equation (3.5) defined as follows is satisfied:

$$\max_i |g_i^{(k)}| \times \max\{|x_i^{(k)}|, |x_i^*|\} \leq \epsilon \times \max\{|f(x^{(k)})|, |f^*|\}, \quad 1 \leq i \leq n, \quad (3.5)$$

where x^* and f^* are the users estimate of typical magnitudes of x and f , respectively. The test condition (equation (3.5)) is independent of any change in the units of f and x .

A total of 12 standard functions have been chosen for evaluation purposes. Some of these functions are: Power's function given in Oren [13], the Broyden-Toint function given in Toint [20], Hilbert's function given in Brent [3], and the remaining functions are described in More, Garbow and Hillstrom [12]. Most of these test problems are also available in the MINPACK and CUTE libraries [2]. Each function, except for Wood and Penalty Function II, is tested with dimensions varied from 20 to 1000 variables. Wood's function is tested with $n = 4$,

while the Penalty Function II is tested with $n = 20$ and 50 , respectively. This represents a total of 63 test problems.

For the purpose of comparison, two serial VM methods are also evaluated over the same set of test problems. These include the CONMIN, developed by Shanno and Phua [18] for the BFGS algorithm (equation (1.3)) and E04DGE, the routine in the NAG library for solving nonlinear unconstrained problems. Numerical results obtained by PVM3.3, CONMIN and E04DGE algorithms are summarized in Table 1. Table 2 shows the differences in performance of these three algorithms.

In these tables, “Iter” denotes the number of iterations and “Ifun” denotes the number of function/gradient evaluations. For PVM3.3, whenever two or three function/gradient evaluations can be done concurrently, they are counted as one function/gradient evaluation. The improvement (differences) and the speedup of PVM3.3 over CONMIN and E04DGE in terms of total

Table 1. Numerical results obtained by PVM3.3, CONMIN and E04DGE programs

Problems	PVM3.3 Iter/Ifun	CONMIN Iter/Ifun	E04DGE Iter/Ifun
Rosenbrock			
$n = 20$	36/45	34/50	31/42
$n = 100$	37/52	36/50	28/50
$n = 200$	43/57	34/45	28/45
$n = 400$	39/56	34/45	31/53
$n = 800$	36/49	36/45	28/54
$n = 1000$	38/50	35/46	26/48
Powell			
$n = 20$	44/48	51/52	40/44
$n = 100$	45/49	67/68	64/73
$n = 200$	59/70	52/53	53/73
$n = 400$	41/44	69/70	117/182
$n = 800$	56/66	55/58	64/102
$n = 1000$	43/49	49/53	222/364
Power			
$n = 20$	37/38	280/281	100/105
$n = 100$	55/56	867/868	201/225
$n = 200$	72/73	1403/1404	159/186
$n = 400$	94/95	2207/2208	157/191
$n = 800$	120/121	3356/3357	460/473
$n = 1000$	137/138	3857/3858	183/209
Watson			
$n = 20$	98/103	115/119	709/715
$n = 100$	157/163	186/195	1101/1138
$n = 200$	336/347	228/233	1184/1257
$n = 400$	410/429	252/265	1594/1749
$n = 800$	649/676	298/313	2371/2656
$n = 1000$	751/792	F/F^*	2304/2678
Broyden-Tridiagonal			
$n = 20$	19/20	21/22	31/40
$n = 100$	21/22	22/23	45/84
$n = 200$	23/24	25/26	67/126
$n = 400$	22/24	28/30	90/176
$n = 800$	21/28	45/47	97/186
$n = 1000$	26/32	60/65	98/187
Variable-dimension			
$n = 20$	20/21	24/25	24/25
$n = 100$	31/32	36/37	36/38
$n = 200$	37/38	41/42	41/42
$n = 400$	41/42	46/47	45/46
$n = 800$	46/47	51/52	50/51
$n = 1000$	47/48	53/54	F/F^{**}
Subtotal:	3787/4044	14053/14206	11879/13713
(36 Problems)		+ $F/$ + F	+ $F/$ + F
Trigonometry			
$n = 20$	40/47	44/50	62/71
$n = 100$	37/48	42/52	58/72
$n = 200$	46/53	47/55	65/79
$n = 400$	41/51	48/58	56/63
$n = 800$	43/52	58/67	58/72
$n = 1000$	43/50	56/63	60/65

continued opposite

Broyden-Toint			
$n = 20$	26/27	36/37	61/72
$n = 100$	39/40	51/52	50/96
$n = 200$	36/37	47/48	58/117
$n = 400$	46/47	922/926	85/174
$n = 800$	1147/1189	1693/1702	1757/2109
$n = 1000$	1385/1444	2061/2071	F/F**
Wood			
$n = 4$	25/31	23/29	37/44
Hilbert			
$n = 20$	17/19	30/31	64/66
$n = 100$	26/29	46/47	168/172
$n = 200$	28/35	53/55	270/274
$n = 400$	32/40	62/64	272/274
$n = 800$	37/41	71/73	279/285
$n = 1000$	45/48	70/72	344/350
Penalty function I			
$n = 20$	53/65	62/73	498/729
$n = 100$	60/69	69/78	790/1230
$n = 200$	60/72	59/72	140/147
$n = 400$	62/71	69/80	113/115
$n = 800$	61/76	58/71	220/222
$n = 1000$	54/60	61/73	294/296
Penalty function II			
$n = 20$	97/121	307/359	181/254
$n = 50$	80/90	523/1051	113/132
Subtotal:	3666/3952	6668/7409	6153/7580
27 Problems		+ F/ + F	
Grand total:	7473/7996	20721/21615	18032/21293
(63 Problems)		+ F/ + F	+ 2F/ + 2F

* Maximum allowable CPU time is exceeded.

** Internal errors occurred.

Table 2. Comparison of results obtained by CONMIN, E04DGE with respect to PVM3.3s results

Problems	Difference2.1 CONMIN	Difference2.2 E04DGE	Speedup2.1 CONMIN	Speedup2.2 E04DGE
	Iter/Ifun	Iter/Ifun	Iter/Ifun	Iter/Ifun
Rosenbrock				
$n = 20$	-2/5	-5/ - 3	0.95/1.12	0.87/0.94
$n = 100$	-1/ - 2	-9/ - 2	0.98/0.97	0.76/0.97
$n = 200$	-9/ - 12	-15/ - 12	0.80/0.79	0.66/0.79
$n = 400$	-5/ - 11	-8/ - 3	0.88/0.81	0.80/0.95
$n = 800$	0/ - 4	-8/5	1.00/0.92	0.69/0.96
$n = 1000$	-3/ - 4	-12/ - 2	0.93/0.92	0.69/0.96
Powell				
$n = 20$	7/4	-4/ - 4	1.16/1.09	0.91/0.92
$n = 100$	22/19	19/24	1.49/1.39	1.43/1.49
$n = 200$	-7/ - 17	-6/3	0.89/0.76	0.90/1.05
$n = 400$	28/26	76/138	1.69/1.60	2.86/4.14
$n = 800$	-1/ - 8	8/36	0.99/0.88	1.15/1.55
$n = 1000$	6/4	179/315	1.14/1.09	5.17/7.43
Power				
$n = 20$	243/243	63/67	7.57/7.40	2.71/.77
$n = 100$	812/812	146/169	15.77/15.50	3.66/4.02
$n = 200$	1331/1331	87/113	19.49/19.24	2.21/2.55
$n = 400$	2113/2113	63/96	23.48/23.25	1.68/2.02
$n = 800$	3236/3236	340/352	27.97/27.75	3.84/3.91
$n = 1000$	3720/3720	46/71	28.16/27.96	1.34/1.52
Watson				
$n = 20$	17/16	611/612	1.18/1.16	7.24/6.95
$n = 100$	29/32	944/975	1.19/1.20	7.02/6.99
$n = 200$	-108/ - 114	848/910	0.68/0.68	3.53/3.53
$n = 400$	-158/ - 164	1184/1320	0.62/0.62	3.89/4.08
$n = 800$	-351/ - 363	1722/1980	0.46/0.47	3.66/3.93
$n = 1000$	F/F*	1553/1886	F/F*	3.07/3.39
Broyden-tridiagonal				
$n = 20$	2/2	12/20	1.11/1.10	1.64/2.00
$n = 100$	1/1	24/62	1.05/1.05	2.15/3.82
$n = 200$	2/2	44/102	1.09/1.09	2.92/5.25
$n = 400$	6/6	68/152	1.28/1.25	4.10/7.34
$n = 800$	24/19	76/158	2.15/1.68	4.62/6.65
$n = 1000$	34/33	72/155	2.31/2.04	3.77/5.85

continued overleaf

Variable-dimension				
$n = 20$	4/4	4/4	1.20/1.20	1.20/1.20
$n = 100$	5/5	5/6	1.17/1.16	1.17/1.19
$n = 200$	4/4	4/4	1.11/1.11	1.11/1.11
$n = 400$	5/5	4/4	1.13/1.12	1.10/1.10
$n = 800$	5/5	4/4	1.11/1.11	1.09/1.09
$n = 1000$	6/6	F/F^{**}	1.13/1.13	F/F^{**}
Subtotal: (36 Problems)	10266/10162 + F	8092/9669 + F	3.72/3.52 + F	3.14/3.40 + F
Trigonometry				
$n = 20$	4/3	22/24	1.10/1.07	1.55/1.52
$n = 100$	5/4	21/24	1.14/1.09	1.57/1.50
$n = 200$	1/2	19/26	1.03/1.04	1.42/1.50
$n = 400$	7/7	15/12	1.18/1.14	1.37/1.24
$n = 800$	15/15	15/20	1.35/1.29	1.35/1.39
$n = 1000$	13/13	17/15	1.31/1.26	1.40/1.30
Broyden–Toint				
$n = 20$	10/10	35/45	1.39/1.38	2.35/2.67
$n = 100$	12/12	11/56	1.31/1.30	1.29/2.40
$n = 200$	11/11	22/80	1.31/1.30	1.62/3.17
$n = 400$	876/879	39/127	20.05/19.71	1.85/3.71
$n = 800$	546/513	610/920	1.48/1.44	1.54/1.78
$n = 1000$	676/627	F/F^{**}	1.49/1.44	F/F^{**}
Wood				
$n = 4$	-2/ - 2	12/13	0.92/0.94	1.48/1.42
Hilbert				
$n = 20$	13/12	47/47	1.77/1.64	3.77/3.48
$n = 100$	20/18	142/143	1.77/1.63	6.47/5.94
$n = 200$	25/20	242/239	1.90/1.58	9.65/7.83
$n = 400$	30/24	240/234	1.94/1.60	8.50/6.85
$n = 800$	34/32	242/244	1.92/1.79	7.55/6.96
$n = 1000$	25/24	299/302	1.56/1.50	7.65/7.30
Penalty function I				
$n = 20$	9/8	445/664	1.17/1.13	9.40/11.22
$n = 100$	9/9	730/1161	1.15/1.14	13.17/17.83
$n = 200$	-1/0	80/75	0.99/1.00	2.34/2.05
$n = 400$	7/9	51/44	1.12/1.13	1.83/1.62
$n = 800$	-3/ - 5	159/146	0.96/0.94	3.61/2.93
$n = 1000$	7/13	240/236	1.13/1.22	5.45/4.94
Penalty function II				
$n = 20$	210/238	84/133	3.17/2.97	1.87/2.10
$n = 50$	443/961	33/42	6.54/11.68	1.42/1.47
Subtotal: 27 Problems	3002/3457 + F	2487/3628	1.82/1.88 + F	1.68/1.92 + F
Grand total: (63 Problems)	13268/13619 + F	10579/13297 + 2 F	2.79/2.71 + F	2.42/2.67 + 2 F

* Maximum allowable CPU time is exceeded.

** Internal errors occurred.

number of iterations and function/gradient evaluations are given in Table 2. These results are listed under columns titled; “Difference 2.1”, “Difference 2.2”, “Speedup 2.1” and “Speedup 2.2”, respectively.

Since CONMIN has failed to solve the Watson function for $n = 1000$, as the maximum allowable CPU time is exceeded, the remaining 62 problems will be considered when the evaluation of PVM3.3 and CONMIN algorithms is performed. In solving this set of 62 test problems, the total number of iterations and function/gradient evaluations required by PVM3.3 and CONMIN are 6772 and 7204 vs 20271 and 21615 respectively. Thus, the savings gained by PVM3.3 in terms of number of iterations and function/gradient evaluations are 13949 and 14411, which represents the speedup factors of 3.06 and 3.00, respectively. The greatest savings are realized by PVM3.3 for the Power function when $n = 800$ and $n = 1000$, in this case, the speedup factor of up to 28 times is achieved.

As for E04DGE, it has failed to solve two problems, these are the Broyden–Toint function ($n = 1000$) and the variable-dimension function ($n = 1000$), due to the fact that internal errors occurred in some routines of this package. Thus, in comparing the performance of PVM3.3 and E04DGE, only 61 problems (excluding the above two problems) will be considered. For this set of problems, the total number of iterations and function/gradient evaluations requested by

PVM3.3 and E04DGE are 6041 and 6504 vs 18032 and 21293 respectively. Hence the savings gained by PVM3.3, in terms of the number of iterations and function/gradient evaluations, are 11991 and 14789 respectively, which represent the speedup factors of 2.98 and 3.27 respectively.

To evaluate the performance of PVM3.3, CONMIN, and E04DGE more closely, we compute the CPU time (seconds) required by these three optimization codes in solving each of the test problems. These results are summarized in Table 3. The speedup factors gained by PVM3.3 over CONMIN and E04DGE are computed and they are listed under the columns called "Speedup 3.1" and "Speedup 3.2" respectively.

Since CONMIN has failed to solve the Watson function ($n = 1000$) for the reasons mentioned above, the remaining 62 test problems will be considered in evaluating its performance against that of PVM3.3. In this case, the total CPU time required by CONMIN and PVM3.3 in

Table 3. CPU time speedup of PVM3.3 over CONMIN and E04DGE

Problems	CPU Time PVM3.3	CPU Time CONMIN	CPU Time E04DGE	Speedup3.1 CONMIN	Speedup3.2 E04DGE
Rosenbrock					
$n = 20$	0.03	0.03	0.07	1.04	2.51
$n = 100$	0.07	0.45	0.11	6.20	1.51
$n = 200$	0.17	1.69	0.18	10.03	1.07
$n = 400$	0.43	6.53	0.32	15.25	0.75
$n = 800$	1.70	27.68	0.69	16.24	0.40
$n = 1000$	2.28	41.82	0.87	18.33	0.38
Powell					
$n = 20$	0.03	0.04	0.07	1.45	2.46
$n = 100$	0.09	0.85	0.16	9.22	1.74
$n = 200$	0.22	2.53	0.22	11.53	1.00
$n = 400$	0.60	13.53	0.49	22.73	0.82
$n = 800$	1.70	42.59	0.86	25.05	0.51
$n = 1000$	2.83	58.45	1.50	20.65	0.53
Power					
$n = 20$	0.03	0.24	0.13	8.31	4.59
$n = 100$	0.16	11.09	0.30	71.31	1.93
$n = 200$	0.45	69.77	0.35	153.78	0.77
$n = 400$	1.65	435.03	0.58	263.34	0.35
$n = 800$	6.85	2780.87	1.38	405.77	0.20
$n = 1000$	11.08	5121.32	1.74	462.01	0.16
Watson					
$n = 20$	0.25	0.16	1.14	0.65	4.58
$n = 100$	1.75	2.85	5.35	1.63	3.06
$n = 200$	3.67	12.32	12.58	3.36	3.43
$n = 400$	11.23	52.34	38.14	4.66	3.40
$n = 800$	33.28	238.50	126.97	7.17	3.81
$n = 1000$	45.39	F^*	178.52	F^*	3.93
Broyden-tridiagonal					
$n = 20$	0.01	0.02	0.12	1.47	9.02
$n = 100$	0.04	0.27	0.75	6.56	18.07
$n = 200$	0.10	1.19	2.52	11.70	24.73
$n = 400$	0.28	5.39	8.90	19.02	31.43
$n = 800$	0.98	34.45	31.60	34.99	32.09
$n = 1000$	1.54	72.68	48.06	47.05	31.11
Variable-dimension					
$n = 20$	0.01	0.02	0.12	1.47	9.02
$n = 100$	0.06	0.48	0.09	8.71	1.64
$n = 200$	0.15	2.12	0.14	14.56	0.96
$n = 400$	0.47	9.29	0.25	19.76	0.53
$n = 800$	1.77	41.36	0.45	23.31	0.25
$n = 1000$	2.79	67.78	F^{**}	24.28	F^{**}
Subtotal: (36 Problems)	134.16	9155.71	465.65 + F	68.24 + F	3.47 + $F + F$
Trigonometry					
$n = 20$	0.07	0.06	0.15	0.79	2.01
$n = 100$	0.68	0.80	2.20	1.17	3.23
$n = 200$	2.94	3.31	14.75	1.13	5.02
$n = 400$	9.83	13.31	100.19	1.35	10.19
$n = 800$	44.37	65.23	764.53	1.47	17.23
$n = 1000$	75.85	98.22	1459.36	1.29	19.24

continued overleaf

Broyden–Toint					
$n = 20$	0.05	0.04	0.14	0.86	2.79
$n = 100$	0.31	0.74	0.62	2.41	2.03
$n = 200$	0.61	2.48	1.88	4.06	3.08
$n = 400$	1.58	187.18	6.66	118.50	4.22
$n = 800$	122.72	1418.31	189.72	11.56	1.55
$n = 1000$	217.39	2685.92	F^{**}	12.36	F^{**}
Wood					
$n = 4$	0.01	0.01	0.29	0.82	31.52
Hilbert					
$n = 20$	0.02	0.03	0.12	1.91	8.00
$n = 100$	0.17	0.63	1.04	3.76	6.23
$n = 200$	0.58	2.79	5.54	4.78	9.50
$n = 400$	2.20	12.84	32.35	5.84	14.70
$n = 800$	6.82	57.92	214.24	8.50	31.42
$n = 1000$	14.54	89.79	406.07	6.18	27.93
Penalty function I					
$n = 20$	0.04	0.05	0.54	1.26	12.95
$n = 100$	0.09	0.89	1.28	9.66	13.96
$n = 200$	0.22	2.93	0.69	13.32	3.13
$n = 400$	0.70	13.45	1.75	19.18	2.50
$n = 800$	2.55	44.40	6.17	17.42	2.42
$n = 1000$	3.29	73.71	9.43	22.38	2.86
Penalty function II					
$n = 20$	0.08	0.27	0.83	3.35	10.31
$n = 50$	0.14	1.98	1.07	14.21	7.68
Subtotal:	507.84	4777.28	3221.61	9.41	6.34
27 Problems		$+F$		$+F$	
Grand total:	642.01	13933.00	3687.26	21.70	5.74
(63 Problems)		$+F$	$+2F$	$+F$	$+2F$

* Maximum allowable CPU time is exceeded.

** Internal errors occurred.

solving these 62 test problems are respectively 13933.00 and 596.62 seconds. Hence the average speedup factor (in terms of CPU time) of PVM3.3 over CONMIN is 23.35. The maximum speedup factor gained by PVM3.3 in this case is 462.01 for the Power function with $n = 1000$.

Similarly, E04DGE has failed to solve two problems, they are the Broyden–Toint function ($n = 1000$) and the variable-dimension function ($n = 1000$) for the reasons mentioned above. The remaining 61 test problems will be considered in evaluating the performance of E04DGE and PVM3.3. In this case, the total CPU time required by E04DGE and PVM3.3 in solving this set of test problems are respectively 3687.26 and 421.83 s. Hence the average speedup factor of PVM3.3 over E04DGE is 8.74. The maximum speedup factor gained by PVM3.3 is 32.09 for the Broyden-Tridiagonal function with $n = 800$.

4. CONCLUSIONS AND DISCUSSIONS

We have presented a class of multi-step, multi-directional parallel algorithms for solving unconstrained optimization problems. The proposed algorithms generate several quasi-Newton directions at each iteration and different line search strategies are then applied in parallel along each direction. The numerical results for a broad class of test problems show that a reduction of 200%, or more, in terms of the number of iterations and function/gradient evaluations can be obtained over the serial BFGS methods.

In practice, we also note that as the size and complexity of the problem increase, greater improvements could be realized by our PVM3.3 algorithm. The numerical results also indicate that the parallel algorithms are efficient and robust in solving large-scale problems. Our numerical results show that the speedup factor of up to 28 times could be achieved by the new parallel VM algorithm in solving certain large-scale test problems when it is compared with the performance of some serial VM algorithms.

As our implementation is presently limited to 9 parallel processors, that is, three parallel processors are required to find the line minimum along each of the three search directions simul-

taneously, further experiments can be carried out to include more parallel processors so that more search directions and line search strategies could be incorporated into the algorithm.

Acknowledgements—The authors are deeply indebted to the referees for many useful comments, suggestions, and modifications concerning the algorithms and test problems of this paper. Special thanks are extended to Professor Hugo D. Scolnik for his invaluable suggestions.

REFERENCES

- Biggs, M. C. (1973) A note on minimization algorithms which make use of non-quadratic properties of the objective function. *J. Inst. Maths. Applics.* **12**, 337–338.
- Bongartz, I., Conn, A. R., Gould, N. and Toint, Ph. L. (1995) CUTE: Constrained and unconstrained testing environment. *ACM Trans. Math. Software* **21**, 123–160.
- Brent, R. P. (1973) *Algorithms for Minimization without Derivatives*. Prentice–Hall, Inc., Englewood Cliffs, NJ.
- Broyden, C. G. (1970) The convergence of a class of double-rank minimization algorithms. *J. Inst. Maths. Applics.* **6**, 76–90.
- Byrd, R. H., Schnabel, R. B. and Shultz, G. A. (1988) Parallel quasi-Newton methods for unconstrained optimization. *Math. Program.* **42**, 273–306.
- Davidon, W. C. (1968) Variable metric method for minimization. *Comput. J.* **10**, 406–410.
- Dennis, J. E., Schnabel, J. R. and Schnabel, R. B. (1983) *Numerical methods for unconstrained optimization and nonlinear equations*. Prentice–Hall, Inc., Englewood Cliffs, NJ.
- Fletcher, R. (1970) A new approach to variable metric algorithms. *Comput. J.* **13**, 392–399.
- Goldfarb, D. (1970) A family of variable metric methods derived by variational means. *Math. Comput.* **24**, 23–26.
- Lootsma, F. A. and Ragsdell, K. M. (1988) State-of-the art in parallel nonlinear optimization. *Parallel Comput.* **6**, 133–155.
- Luksan, L. (1994) Computational experience with known variable metric updates. *J. Opt. Theory Appl.* **83**, 27–47.
- More, J. J., Garbow, B. S. and Hillstom, K. E. (1981) Testing unconstrained optimization software. *ACM Trans. Math. Software* **7**, 17–41.
- Oren, S. S. (1974) Self-scaling variable metric algorithms Part II. *Manage. Sci.* **20**, 863–874.
- Phua, K. H. and Chew, S. B. (1992) Symmetric rank-one update and quasi-Newton methods. In: *Optimization Techniques and Applications*, ed. K. H. Phua *et al.*, pp. 52–63. World Scientific, Singapore.
- Phua, K. H. and Setiono, R. (1993) Multi-step, multi-directional parallel algorithms for unconstrained optimization. In: *Optimization Techniques and Applications*, ed. K. H. Phua *et al.*, pp. 481–487. World Scientific, Singapore.
- Schnabel, R. B. (1987) Concurrent function evaluations in local and global optimization. *Comput. Methods Appl. Mech. Eng.* **64**, 537–552.
- Shanno, D. F. (1970) Conditioning of quasi-Newton methods for function minimization. *Math. Comput.* **24**, 647–656.
- F Shanno, D. and Phua, K. H. (1980) Remark on algorithm 500—a variable method subroutine for unconstrained nonlinear optimization. *ACM Trans. Math. Software* **6**, 618–622.
- Straeter, T. A. (1973) A parallel variable metric optimization algorithm. In: *NASA Technical Note*. NASA TN D-7329, Langley Research Center, Hampton, VA 23665.
- Toint, Ph. L. (1985) Some numerical results using a sparse matrix updating formula in unconstrained optimization. *Math. Comput.* **32**, 68–81.
- van Laarhoven, P. J. M. (1985) Parallel variable metric algorithms for unconstrained optimization. *Math. Program.* **33**, 68–81.