SIAM J. OPTIM.
Vol. 13, No. 1, pp. 264–282

A CLASS OF TRUST-REGION METHODS FOR PARALLEL OPTIMIZATION*

P. D. HOUGH† AND J. C. MEZA†

Abstract. We present a new class of optimization methods that incorporates a parallel direct search (PDS) method within a trust-region Newton framework. This approach combines the inherent parallelism of PDS with the rapid and robust convergence properties of Newton methods. Numerical tests have yielded favorable results for both standard test problems and engineering applications. In addition, the new method appears to be more robust in the presence of noisy functions, which are inherent in many engineering simulations.

Key words. parallel optimization, trust-region methods, direct search methods, nonlinear programming

AMS subject classifications. 65Y05, 68W15, 90C30, 90C53, 90C90

PII. S1052623498343799

1. Introduction.

Optimization of functions derived from the modeling and simulation of some physical process constitutes an important class of problems in many engineering and scientific applications. Often, the computer simulation entails the solution of a system of nonlinear partial differential equations (PDE) in two or three dimensions. Other applications include particle dynamics simulations or problems in chemical kinetics. The main characteristic of these types of problems is that the function evaluation is computationally expensive and dominates the total cost of the optimization problem. Depending on the nature of the application and the solution method employed, there can also be noise associated with the evaluation of the objective function. This noise can usually be reduced, but only at the cost of making the computation time even greater. In many of these applications, derivative information is also not available or must be computed using finite differences, thereby generating noisy gradients. Fortunately, the dimension of the optimization problem in many of these optimal design problems is small (usually on the order of tens of parameters). In this study, we will concentrate on the development of parallel unconstrained optimization algorithms for the solution of these types of problems on small-scale shared memory processors (SMPs), where the number of available processors is comparable to the number of optimization parameters. The rationale for this decision is that, although massively parallel computers are available, the majority of computational power in most industrial or scientific settings consists of small-scale clusters of SMPs or networks of workstations (NOWs) that can be used in a similar capacity.

There have been many attempts at parallelizing nonlinear optimization methods. In the area of Newton methods, one of the earliest attempts at parallelization was the work of Straeter [22], who developed a parallel rank-one updating formula for

*Received by the editors August 19, 1998; accepted for publication (in revised form) December 21, 2001; published electronically August 28, 2002.

http://www.siam.org/journals/siopt/13-1/34379.html

†Computational Sciences and Mathematics Research Department, Sandia National Laboratories, P.O. Box 969, MS 9217, Livermore, CA 94551 (pdhough@ca.sandia.gov, meza@ca.sandia.gov).

the Hessian approximations used in variable metric methods. This formula was later extended by Laarhoven [16] to more general updating formulas. Byrd, Schnabel, and Shultz [2] also proposed parallel quasi-Newton methods based on speculative gradient and Hessian evaluations. Schnabel [21] gave an excellent review of the challenges and limitations in parallel optimization. In that review, Schnabel identified three major levels for introducing parallelism: (i) parallelize the function, gradient, and constraint evaluations; (ii) parallelize the linear algebra; and (iii) parallelize the optimization algorithm at a high level.

In this study, due to the characteristics of the problems mentioned above, we choose to focus on the third option. In particular, the first option is not usually available to us because for many situations we do not have access to the source code for the function or the constraints. In addition, the dimension of the optimization problems of interest is usually small, and therefore parallelizing the linear algebra would not yield any benefits.

The third option, that of parallelizing optimization at a high level, has recently received more attention. Some attempts that fit into this category include methods such as parallel direct search methods [7], genetic algorithms [14], and simulated annealing [13]. These methods are inherently parallel and extremely popular in engineering optimization. Although these search methods can be powerful tools, they suffer from slow convergence and thus may require many function evaluations. In a setting in which each function evaluation may take several CPU hours to compute, this is highly undesirable.

Newton-based methods, on the other hand, have good convergence properties, but there are few options for parallelizing a standard Newton method at a high level. For the purposes of this paper, we will assume that the gradient of the objective function is not available and that finite differences are used to compute any necessary derivative information. This calculation is trivially parallelized, so we focus our attention on finding less apparent opportunities. Another approach to parallelization is the work by Phua and Zeng [19] in which they use a multiple line search, multiple direction algorithm to introduce parallelism into a Newton method. However, it is not clear how robust a line search method would be in a situation where the function and gradient are noisy. Carter [3] has addressed the issue of inexact gradients for another class of algorithms known as trust-region methods and has given conditions under which these algorithms will converge. In a separate paper, Carter presented various numerical results [4] for this class of algorithms.

In this paper, we consider a new class of methods that combines the parallel direct search (PDS) method and the trust-region method to produce a new class of algorithms that takes advantage of the best properties of each approach. In particular, we will show that the rapid convergence rates typical of Newton-type methods are preserved while the advantage of parallelism inherent in the PDS methods is gained. In section 2, we describe the new class of algorithms and in section 3 consider their convergence properties. In section 4, we give numerical results from a set of test problems and an application in optimal design. We conclude in section 5 with a summary and a brief discussion of future research directions.

2. The trust-region PDS algorithm. Before describing the new class of algorithms, we first give a brief overview of the standard trust-region method and the PDS method of Dennis and Torczon [7]. In each iteration of a trust-region method, a quadratic model of the objective function, f, is formed, and a region in which the model is trusted to approximate the actual function accurately is determined. A trial

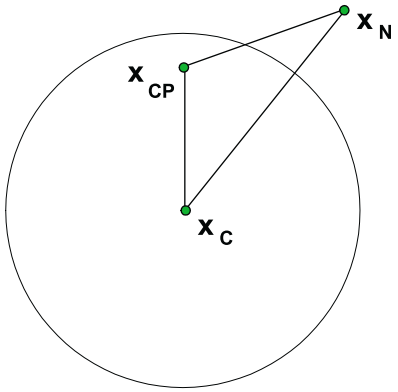

FIG. 1. *Standard trust-region method.*

step is then computed by approximately solving the following subproblem:

$$\min_{\mathbf{s}\in\mathbb{R}^n} \ \psi(\mathbf{s}) = g(\mathbf{x}_c)^T\mathbf{s} + \frac{1}{2}\mathbf{s}^T H_c\mathbf{s}, \tag{1}$$

$$\text{s.t.} \ \ \|\mathbf{s}\|_2 \leq \delta_c,$$

where $\mathbf{x}_c$ is the current point, $\mathbf{s}$ is the step, $g(\mathbf{x}_c)$ is the gradient of f at the current point, $H_c \approx \nabla^2 f(x_c)$ is the Hessian approximation at the current point, and δ_c is the size of the trust region. We will refer to this as the *trust-region subproblem.* It is well known that the step generated at each iteration, k, can be computed using any method, as long as it satisfies a fraction of Cauchy decrease condition according to the quadratic model. In particular, there must exist constants, $\beta > 0$ and $C > 0$, independent of k, for which the step s_k taken at iteration k satisfies

$$\psi_k(\mathbf{s}_k) \leq \beta \left\|g(\mathbf{x}_k)\right\|_2 \min\left(\delta_k, \frac{\|g(\mathbf{x}_k)\|_2}{C}\right). \tag{2}$$

There are several well-known procedures for computing the solution to (1) that satisfy (2). One such example is the dogleg step (see [6, p. 139]), which is a convex combination of the steepest descent direction and the Newton direction. This approach is illustrated in Figure 1.

The PDS algorithm belongs to a class of optimization methods that do not compute derivatives. The PDS algorithm can be briefly described as follows. Starting from an initial simplex, S_0, the function value at each of the vertices in S_0 is computed, and the vertex corresponding to the lowest function value, v_0, is determined. Using an underlying grid structure, S_0 is reflected about v_0, and the function values at the vertices of this rotation simplex, S_r, are compared against the function value at v_0. If one of the vertices in S_r has a function value less than the function value corresponding to v_0, then an expansion step to form a new simplex, S_e, is attempted in which the size of S_r is expanded by some multiple, usually 2. The function values at the vertices of S_e are compared against the lowest function value found in S_r. If a lower function value is encountered, then S_e is accepted as the starting simplex for the next iteration; otherwise, S_r is accepted for the next iteration. If no function value lower than the one corresponding to v_0 is found in S_r, then a contraction simplex is created by reducing the size of S_0 by some multiple, usually 1/2, and is accepted for the next iteration.

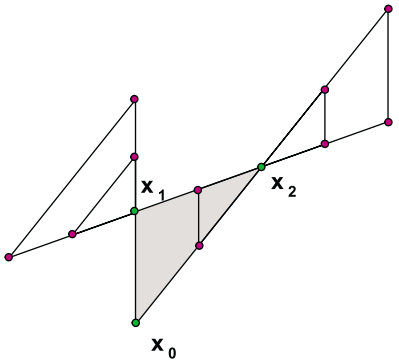

FIG. 2. *PDS method.*

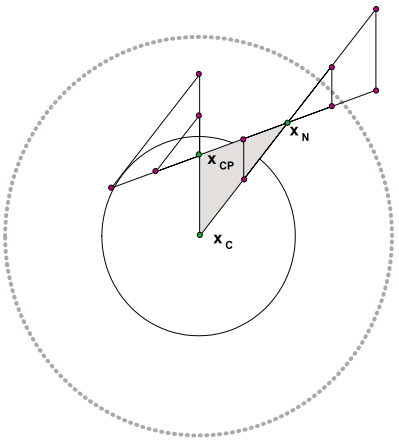

FIG. 3. *TRPDS method.*

Because PDS uses only function comparisons, it is easy to implement and use. Since the rotation, expansion, and contraction steps are all well determined, it is also possible to precompute a set of grid points corresponding to the vertices of the simplices constructed from various combinations of rotations, expansions, and contractions. Given this set of grid points, called a search scheme, the PDS algorithm can compute the function values at all of these vertices in parallel and determine the vertex corresponding to the lowest function value. The number of points used in the search scheme is referred to as the *search scheme size*, and it usually is adjusted to be at least equal to the number of processors available. Figure 2 demonstrates one possible PDS iteration.

Both the trust-region and the PDS methods have advantages and disadvantages, as described in section 1. In order to combine the strengths of these methods, we propose a new class of algorithms which uses the PDS method within a trust-region framework. This type of algorithm, which we will refer to as TRPDS, is illustrated in Figure 3 and is described below. The controlling framework is the same as that for standard trust-region algorithms, but the method of computing the new step is different. Rather than solving the trust-region subproblem, the TRPDS method

approximately solves the following problem:

$$\begin{aligned} &\min_{\mathbf{s}\in\mathbb{R}^n} f(\mathbf{x}_c+\mathbf{s}), \\ &\text{s.t. } \|\mathbf{s}\|_2 \le 2\delta_c, \\ &\qquad \psi(\mathbf{s}) \le \beta \|g(\mathbf{x}_c)\|_2 \min\left(\delta_c, \frac{\|g(\mathbf{x}_c)\|_2}{C}\right), \end{aligned} \tag{3}$$

where $\beta > 0$, $C > 0$, and ψ is defined as in (1). We will refer to this as the *PDS subproblem*. There are several notable differences between this and the standard trust-region approach. The first is that the actual objective function, as opposed to a quadratic model of the objective function, is being minimized. Second, this subproblem is not solved to optimality; only a small amount of decrease is required from the PDS method. Third, the step length is allowed to be twice the size of the trust region to allow for the possibility of taking a step longer than the Newton step, as is sometimes done to accelerate local convergence of singular problems. Further discussions of this acceleration idea can be found in [20] and [15]. Finally, because by design the steps computed by PDS do not satisfy the fraction of Cauchy decrease condition (2), we must include an explicit constraint to enforce this condition.

An overview of the TRPDS algorithm appears below, followed by a discussion of the critical steps.

ALGORITHM 1 (TRPDS).
Given $\mathbf{x}_0$, $\mathbf{g}_0$, H_0, δ_0, and $\eta \in (0,1)$,
for $k = 0, 1, \ldots$ until convergence **do**
  1. Solve $H_k \mathbf{s}_N = -\mathbf{g}_k$.
  for $i = 0, 1, \ldots$ until step accepted **do**
    2. Form an initial simplex using $\mathbf{s}_N$.
    3. Find an approximate solution $\mathbf{s}_i$ to (3) using PDS.
    4. Compute $\rho_i = (f(\mathbf{x}_k + \mathbf{s}_i) - f(\mathbf{x}_k))/\psi_k(\mathbf{s}_i)$.
    if $\rho_i > \eta$, **then**
      5. Accept step and set $\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{s}_i$, eval $\mathbf{g}_{k+1}$ and H_{k+1}.
    else
      6. Reject step.
    end if
    7. Update δ.
  end for
end for

Here we use the notation $\mathbf{g}_k = g(\mathbf{x}_k)$ and $H_k = H(\mathbf{x}_k)$, where $H(\cdot)$ is the Hessian approximation. There are several points to consider within this framework. The initial simplex formed in step 2 needs to be chosen carefully. While there is a lot of freedom in the choice of the initial simplex, it will have an impact on the solution to (3) and on the performance of the algorithm. There is also a question as to how accurately we should solve (3). In many applications it may be reasonable to ask for only a small fraction of decrease in the function since each function evaluation is so expensive. This also has a bearing on the decision to accept or reject the new step. Finally, the updating of the trust region must be addressed within the context of this framework. In the following sections, we address each one of these issues.

2.1. Choosing the initial simplex. In step 2 of the TRPDS algorithm we require the formation of an initial simplex. The choice of this simplex is important to the performance of the PDS algorithm and deserves careful consideration. There

are three points that must be included in the simplex: the current point, the Cauchy point, and the Newton point. The Cauchy point is defined to be the minimizer of the quadratic model along the steepest descent direction. Likewise, we define the Newton point, $\mathbf{s}_N$, to be the minimizer of the quadratic model along the Newton direction. The current point must be in the simplex so that PDS can determine whether or not it has found a descent direction. The Cauchy point is required to ensure convergence of the algorithm. Finally, the Newton point is necessary to allow for the possibility of rapid convergence in the limit. In practice, all three of these points are not always included, but instead related points are used. A discussion of these substitutions follows.

Recall that, in the standard trust-region algorithm, the step length is limited by the size of the trust region. When the Cauchy point or the Newton point falls outside of the trust region, it is projected onto the trust region. As we are seeking to preserve as much of the trust-region framework as possible, the construction of the initial simplex includes similar features. There are three different scenarios that must be addressed.

1. *Both the Cauchy point and the Newton point are inside the trust region.* This case is straightforward. The Cauchy point and the Newton point are used in the initial simplex.
2. *The Cauchy point is inside the trust region, and the Newton point is outside the trust region.* The Cauchy point is used in the initial simplex. The dogleg point is computed and replaces the Newton point in the initial simplex.
3. *Both the Cauchy point and the Newton point are outside the trust region.* Both points are projected onto the trust region, and the resulting points are used in the initial simplex.

For a problem of dimension n, PDS requires the initial simplex to have $n+1$ vertices. We have described the selection of only three. The question of how to pick the remaining $n-2$ vertices remains. While there are many logical ways to choose these points, the only real restriction on them is that they be chosen such that the initial simplex is not degenerate. Our current implementation uses $n-2$ vertices from a right angle simplex constructed around the Newton point; i.e., vertices $4, \ldots, n+1$ are defined as follows:

$$\mathbf{v}_{i+3} = \mathbf{x}_N + \delta_c \mathbf{e}_i, \quad i = 1, \ldots, n-2,$$

where $\mathbf{x}_N$ is the Newton point, δ_c is the current trust-region radius, and $\mathbf{e}_i$ is the ith column of the $n \times n$ identity matrix.

When the simplex is constructed in the manner described here, there are two situations in which it may be degenerate. One case can arise in the first iteration of the trust-region method. If the initial Hessian is a multiple of the identity, then the Newton direction and the Cauchy direction will be the same, so the simplex needs to be constructed in a slightly different manner in the first iteration. We use the current point, the Newton point, and $n-1$ of the vertices from the right angle simplex around the Newton point. The other case of degeneracy arises if the edges of the simplex are badly scaled. This is easily corrected by rescaling all of the edges to be the same length as the Newton edge. Note that this allows longer steps than if all edges were rescaled to be the same length as the Cauchy edge.

2.2. Solving the PDS subproblem. One way to think about the TRPDS algorithm is to imagine using an optimization algorithm within an optimization algorithm. As such, the PDS method needs algorithmic parameters in order to solve the

PDS subproblem. In particular, PDS needs information about the search space, and it needs stopping criteria. Recall that the PDS method evaluates the function at a set of predetermined reflection, contraction, and expansion points in order to determine a trial step. This search scheme can be determined ahead of time and need only be generated once; however, during the optimization phase, the PDS method must know how many points in that search scheme to evaluate at each iteration. One possible choice in a parallel setting is to set this number equal to the number of processors that are available. PDS also needs to be aware of the constraints on the step. Clearly, it must know the size of the trust region, as that constrains the step length. As noted earlier, we relax the trust region by a factor of two in order to allow for the possibility of taking a step longer than the Newton step. Finally, PDS must have access to the quadratic model used by the trust-region framework in order to ensure that it generates trial steps that satisfy the fraction of Cauchy decrease constraint.

Since the PDS subproblem is not solved to optimality, we use four criteria to determine when to return a trial step. The first is a simple decrease requirement. If f_c is the function value at the current point, then we return a step when

$$f_t \leq pdstol * f_c, \tag{4}$$

where f_t is the function value at the trial point, and $pdstol < 1$ is the amount of decrease desired. The second is a restriction on how much PDS is allowed to decrease the step length. Ideally, the trust-region framework should maintain control of the step length; however, if stopping criteria for PDS are not chosen appropriately, it is possible for PDS to "hijack" control of the step length. Recall that PDS, in its effort to find decrease, may reduce the size of the simplex. If allowed too many opportunities to shrink the simplex, PDS can return a step that is significantly smaller than the current trust region. As we will see in section 2.4, the trust-region update is based on this step length. As a result, the trust region will become unacceptably small, causing the algorithm to halt prematurely. In order to prevent that from happening, we return a trial step when

$$\|E_t\|_2 \leq etol * \|E_0\|_2 ,$$

where E_t is the longest edge in the simplex producing the trial point, E_0 is the longest edge in the initial simplex, and $etol \leq 1$ is the edge reduction tolerance. If PDS cannot find a trial point that satisfies either of these criteria, then it returns when it has exceeded either the maximum number of function evaluations or the maximum number of iterations allowed.

2.3. Acceptance/rejection of step. Once a step has been computed, it is necessary to determine whether or not it is acceptable. This is handled by the trust-region framework. If the step yields sufficient decrease, then the step will be accepted. Otherwise, the step is rejected. In a standard trust-region method, sufficient decrease is determined by computing ρ_k as given in step 4 of the algorithm and comparing it to some tolerance. If ρ_k is greater than the tolerance, then the decrease is sufficient. There is some flexibility in the choice of this tolerance, and computational expense of the function plays a role in determining the appropriate choice. There is one situation that arises in the TRPDS algorithm that requires a minor modification to this scheme. It is possible that PDS will find no decrease, and thus return a step of zero length. In this case, ρ_k is not computed, and the step is rejected immediately.

2.4. Updating the trust region. The procedure for updating the trust region is based on the strategy proposed in [3]. At each iteration, the trust region is updated as follows:

$$\delta_{k+1} = \begin{cases} \dfrac{\min(\delta_k, \|E\|_2)}{10} & \text{if } \rho_k < \eta_1 \text{ or } \|\mathbf{s}_k\|_2 = 0, \\ \dfrac{\|\mathbf{s}_k\|_2}{2} & \text{if } \eta_1 \le \rho_k \le \eta_2, \\ 2 \cdot \delta_k & \text{if } \eta_3 \le \rho_k \le 2 - \eta_3, \\ \max(2 \cdot \|\mathbf{s}_k\|_2, \delta_k), & \text{otherwise.} \end{cases}$$

Here E is the longest edge of the final PDS simplex, and $\eta_1, \eta_2, \eta_3 \in (0, 1)$. Notice that when the trust-region size is reduced, we incorporate information about the size of the final simplex in order to reduce the possibility of rechecking points that are already known to be unacceptable. In our algorithm, we also impose a maximum on the trust-region size that is allowed at any iteration.

3. Convergence results. It is interesting to note that Algorithm 1 falls into the class of generalized trust-region methods described in [1]. As such, it offers a great deal of flexibility, and the convergence theory is straightforward. In order to apply the work of [1], we first demonstrate how TRPDS fits into the generalized trust-region framework.

The first feature of interest is the approximation model used by the trust-region framework. The model used to approximate the objective function can be any model suitable for the application in question as long as it satisfies the following mild conditions:

$$a_k(\mathbf{x}_k) = f(\mathbf{x}_k), \tag{5}$$

$$\operatorname{grad} a_k(\mathbf{x}_k) = \operatorname{grad} f(\mathbf{x}_k), \tag{6}$$

where a_k is the approximation model at iteration k. In the TRPDS setting, a_k is the quadratic model, so these conditions are clearly satisfied.

The second feature of interest in the generalized trust-region framework is that the step generated at each iteration can be computed using any method as long as it satisfies a fraction of Cauchy decrease condition according to the approximation model being used. In particular, there must exist constants, $\beta > 0$ and $C > 0$, independent of k, for which the step taken at iteration k, $\mathbf{s}_k$, satisfies

$$f(\mathbf{x}_k) - a_k(\mathbf{x}_k + \mathbf{s}_k) \ge \beta \|g(\mathbf{x}_k)\|_2 \min\left(\delta_k, \frac{\|g(\mathbf{x}_k)\|_2}{C}\right), \tag{7}$$

where a_k denotes the model being used to approximate the objective function. To obtain a trial step, we use PDS to solve the PDS subproblem (3), which includes a fraction of Cauchy decrease as an explicit constraint. Therefore, we know that the trial step will satisfy (7). We note that numerical results indicate that ignoring the fraction of Cauchy decrease condition rarely has undesirable effects on the convergence of the algorithm in practice. Given the computational expense of the function, we believe that excluding steps that do not satisfy a fraction of Cauchy decrease should be optional in practice.

With the conditions on the model and the steps satisfied, we can now apply standard trust-region theory.

THEOREM 3.1. *Assume that f is uniformly continuously differentiable, bounded below, and that the Hessian approximations are uniformly bounded. Furthermore, assume that the sequence of iterates generated by Algorithm* 1 *satisfies* (5)–(7). *Then*

$$\lim_{k\to\infty} \inf \|\text{grad}\, f(\mathbf{x}_k)\|_2 = 0.$$

In fact, since Algorithm 1 uses widely accepted criteria for updating the steps within the trust-region framework, it is easy to show that the stronger condition

$$\lim_{k\to\infty} \|\text{grad}\, f(\mathbf{x}_k)\|_2 = 0$$

holds.

This is satisfying, in that the new class of algorithms inherits all of the good theoretical convergence properties of the classical trust-region methods while incorporating all of the practical advantages of PDS for typical engineering optimization problems.

4. Numerical results. In order to evaluate the performance of the TRPDS algorithm, we chose a set of test problems from the literature. One set of 24 problems was obtained from papers by Moré, Garbow, and Hillstrom [18] and Byrd, Schnabel, and Shultz [2]. Another set of 22 problems was obtained from the CUTE test set developed by Conn, Gould, and Toint [5]. A list of the problems used can be found in the appendix. For comparison purposes, we also solved these problems using a standard BFGS trust-region algorithm.

To evaluate the effectiveness of the TRPDS algorithm, we ran a series of tests varying several of the algorithmic parameters. The first set of 24 problems allow a variable dimension, so we ran problems with dimensions of dim $= 4, 8, 16$. The search scheme size (sss) was then chosen so that sss $=$ dim $+ 1, 2 *$ dim, $4 *$ dim, $8 *$ dim. The function tolerance used for the inner PDS iteration was also varied using values of $0.1, 0.5, 0.9$, and 0.99995. The CUTE test problems were run using the default dimension and varying only the sss and the function tolerance for PDS. The combination of all of these experiments resulted in a total of 1504 test cases run, though we present only a representative subset of the results here. In addition, we also present results for the case of noisy functions, as well as the results from an engineering application problem based on a computer model of a chemical vapor deposition furnace.

All tests were run on a 64-processor SGI Origin 2000 with the IRIX 6.5 operating system. The starting points used for these problems were the same as those given in the references, and the gradients were computed using parallel central differences. The BFGS algorithm was run with the number of processors equal to the dimension of the problem, which is the maximal amount of parallelism for our implementation of the parallel central differences. The TRPDS algorithm was run with the number of processors equal to the sss, which we varied in our tests. Algorithmic parameters are listed in Table 1.

The step tolerance, the function tolerance, and the gradient tolerance are used as stopping criteria in termination tests like those found in [8, p. 306].

The next three sections contain the results for all of the test runs: (1) the noise-free case, (2) noisy functions, and (3) a furnace problem.

4.1. Noise-free tests. The results for the noise-free case are contained in Figures 4–7. While reporting the number of iterations may be useful in some settings,

TABLE 1
Algorithmic parameters.

Parameter	**Value**
Initial trust region	$0.1 \cdot \|\mathbf{g}_0\|_2$
Machine epsilon	$2.22045 \cdot 10^{-16}$
Maximum step	4000
Minimum step	$1.49012 \cdot 10^{-8}$
Maximum iter	500
Maximum fcn eval	$1,000,000$
Step tolerance	$1.49012 \cdot 10^{-8}$
Function tolerance	10^{-10}
Gradient tolerance	10^{-6}

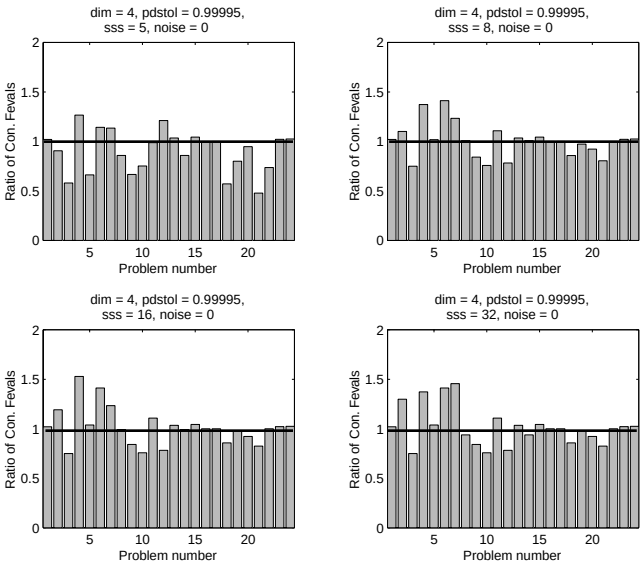

FIG. 4. *Ratios of the number of concurrent function evaluations for BFGS to that for TRPDS for dimension* 4. *Ratios greater than one indicate that the TRPDS algorithm takes fewer concurrent function evaluations and thus less overall time.*

we believe that it is not useful here. In our applications, the most important measure of performance is the total time to solution of the problem. Since the computational cost of the function evaluations dominates the cost of the algorithm, we base our comparison of TRPDS with BFGS on the number of function evaluations. However, since the two algorithms have different degrees of parallelism, comparing the total number of function evaluations required for each is not a fair method of comparison, as this would not reflect the amount of time required to solve the problems. Instead, we compare the number of *concurrent function evaluations*, which are defined as follows. Suppose that p processors are available, where $p \geq 1$. If p independent function evaluations are required, then each processor can be tasked to perform one of them. This means that p function evaluations can be done simultaneously. Thus, we define a *concurrent function evaluation* to be one instance of p function evaluations being

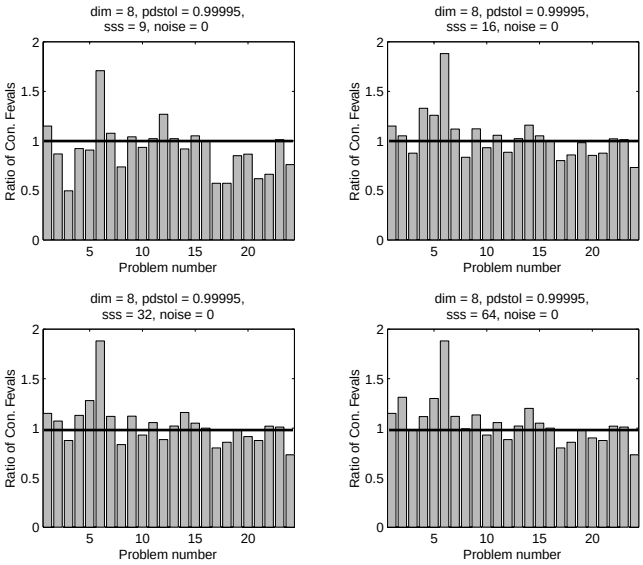

FIG. 5. *Ratios of the number of concurrent function evaluations for BFGS to that for TRPDS for dimension* 8. *Ratios greater than one indicate that the TRPDS algorithm takes fewer concurrent function evaluations and thus less overall time.*

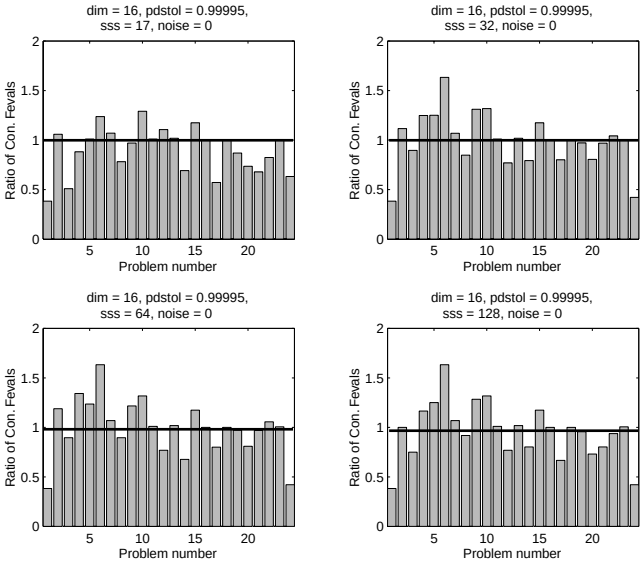

FIG. 6. *Ratios of the number of concurrent function evaluations for BFGS to that for TRPDS for dimension* 16. *Ratios greater than one indicate that the TRPDS algorithm takes fewer concurrent function evaluations and thus less overall time.*

performed simultaneously. Comparing the number of concurrent function evaluations is therefore roughly equivalent to comparing total wall clock time. The results that appear in Figures 4–7 are the ratios of the number of concurrent function evaluations taken by the trust-region method to the number of concurrent function evaluations

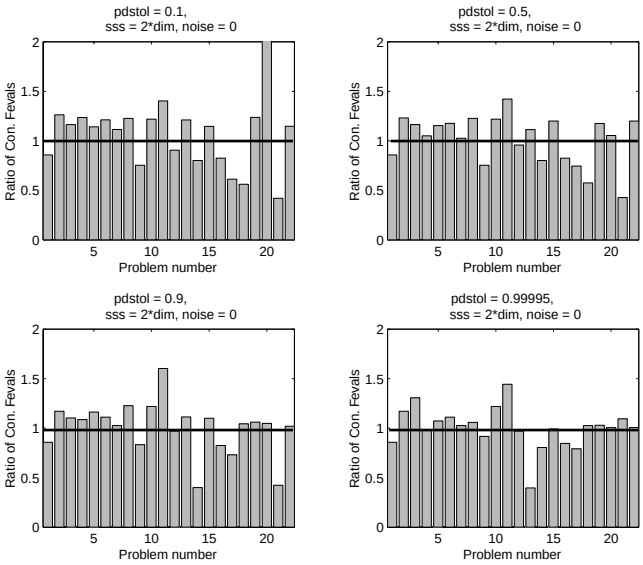

FIG. 7. *Ratios of the number of concurrent function evaluations for BFGS to that for TRPDS for CUTE problems. Ratios greater than one indicate that the TRPDS algorithm takes fewer concurrent function evaluations and thus less overall time.*

taken by the TRPDS algorithm. Ratios greater than one indicate that the TRPDS algorithm will take less overall time.

Overall, we find that TRPDS does at least as well as BFGS in 60% of the test cases. We now take a closer look at the effects of the parameters on the performance of the algorithm. The results shown in Figures 4–6 allow us to make some general observations about the effect of the sss. For a given problem, the size of the search scheme has very little effect on the performance of the algorithm in comparison to the standard trust-region algorithm. There are several factors that may be contributing to this effect. The first is that we have biased the search directions towards the Newton point through our method for constructing the initial simplex. A second factor is the order in which the search scheme points are chosen and evaluated. In the standard PDS algorithm, the reflection points are evaluated first, followed by the contraction points and the expansion points. Since the Newton point is often a good trial point we would not expect nearby points to have a lower function value, and we need to take a large step before finding a better point. Due to the construction of the search pattern, this requires a search scheme larger than those used in our tests. In a situation where we are far away from the solution, or where the function or gradient is noisy, the Newton point might not be a good trial point. In this case, the reflection and contraction points may yield better trial points than the Newton point, and thus the sss represented here might affect the performance of the algorithm. This is a point that will require further investigation. It would also be interesting to try different methods for creating the initial simplex to determine the effect on the algorithm's efficiency.

We have also examined the effects of changing the amount of function decrease requested from the inner PDS iteration by (4). The results are shown for the CUTE problems in Figure 7. We find that changing the amount of function decrease required has very little effect on the performance of TRPDS. The reason is that PDS satisfies

TABLE 2
Convergence tolerances for noisy functions.

Parameter	**Value**
Function tolerance	$\eta/10$
Gradient tolerance	$100\eta^{2/3}$

other stopping criteria before it attains the required amount of function decrease, particularly for larger amounts of decrease.

Schnabel [21] presents an argument for the merits of a line search algorithm with speculative gradient evaluation. This entails using extra processors to compute the components of a finite difference gradient at the current point while the function is being evaluated at that point. He argues that it is difficult to develop a parallel line search algorithm that will perform better than a speculative gradient algorithm, particularly when the dimension of the problem is not much larger than the number of available processors. A similar argument holds for a trust-region approach. Thus, we should also compare TRPDS to a speculative gradient implementation of a trust-region method. We have begun to address this work, and we refer the reader to [11] for an analytical comparison of the methods, and to [12] for preliminary numerical results.

4.2. Noisy functions. Since one of the motivations for proposing this new algorithm was to improve the robustness of Newton methods when applied to noisy functions, we also ran a set of test problems in which random noise was added to the functions. These test problems were generated by computing a function value such that

$$\hat{f}(x) = f(x) + \eta u,$$

where u is a random number from a uniform distribution, $U(0,1)$, and η is the noise level. We ran tests with $\eta = 10^{-9}, 10^{-6}$, and 10^{-3}. One of the more difficult issues in this set of test problems was choosing the stopping criteria. Since we are computing gradients using finite differences, the noise in the function will propagate into the gradients. As such, it is sometimes difficult to detect convergence using the standard stopping criteria. In our case, we used the tolerances given in Table 2.

The results shown in Figures 8–9 allow us to make some general observations about the effect of noise on the algorithms. For the first set of test problems, the TRPDS algorithm either beats or ties the BFGS algorithm in 66% of the test cases. More importantly, TRPDS fails to converge on only 3% of the tests while the BFGS algorithm fails to converge on 17% of the tests. The results from the CUTE test set are not quite as convincing with regard to robustness, although the TRPDS algorithm does win 70% of the time. As in the noise-free case, we suspect that a different choice of the initial simplex and perhaps a larger search scheme size will benefit the TRPDS algorithm in the noisy function case; we are pursuing this line of research.

4.3. Furnace design test problem. As another test case, we chose an optimization problem derived from the design of a vertical, multiwafer furnace. Vertical furnaces can process up to 200 silicon wafers in a single batch and have been used for thin film deposition, oxidation, and other thermal process steps. The evolution of vertical furnaces has been driven by the need for process uniformity (that is, wafer-to-wafer and within-wafer uniformity) and high wafer throughput. A recent variation

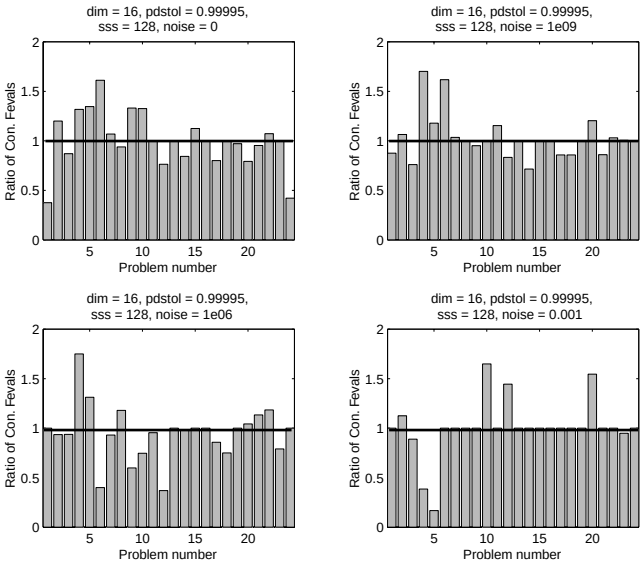

FIG. 8. *Ratios of the number of concurrent function evaluations for BFGS to that for TRPDS for noisy problems. Ratios greater than one indicate that the TRPDS algorithm takes fewer concurrent function evaluations and thus less overall time.*

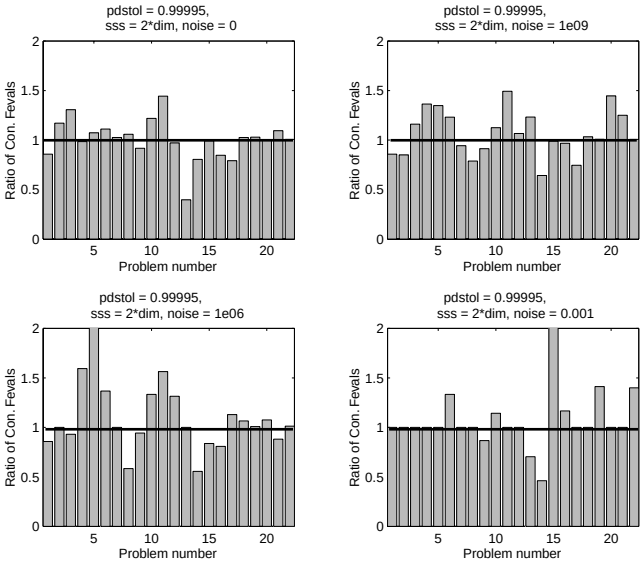

FIG. 9. *Ratios of the number of concurrent function evaluations for BFGS to that for TRPDS for noisy CUTE problems. Ratios greater than one indicate that the TRPDS algorithm takes fewer concurrent function evaluations and thus less overall time.*

of the multiwafer reactor design is the small-batch, fast-ramp (SBFR) furnace. The SBFR is designed to heat up and cool down quickly, thus reducing cycle time and thermal budget. The SBFR consists of a stack of silicon wafers eight inches in diameter enclosed in a vacuum-bearing quartz jar. The stack is radiatively heated by resistive

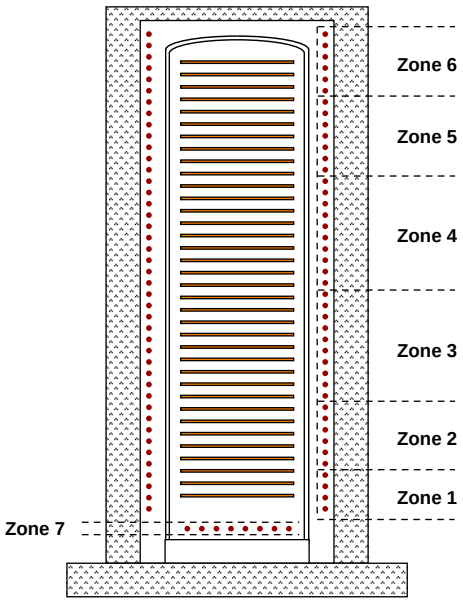

FIG. 10. *Vertical batch furnace with seven control zones.*

coil heaters contained in an insulated canister. The heating coils can be individually controlled or can be ganged together in zones to vary the emitted power along the length of the reactor; a seven-zone configuration is shown in Figure 10. There are six control zones (each containing several heating coils) along the length of the furnace and one heater zone in the base. The zones near the ends of the furnace are usually run hotter than the middle zones to make up for heat loss.

The thermal design optimization problem can be described as follows. Given a set number of fixed heating coils, how can the coils be grouped in the fewest number of control zones such that the temperature deviation about a fixed set-point is minimized? For this example, we concentrate on finding the optimal power settings and related temperature uniformity for a fixed zone configuration. The objective function, F, is defined by a least-squares fit of the N discrete wafer temperatures, $T_{w,i}$, to a prescribed profile, $T_{s,i}$,

$$F(p_j) = \sum_{i=1}^{N} (T_{w,i} - T_{s,i})^2, \tag{8}$$

where p_j are the unknown power parameters.

The engineering heat transfer model used in this example was developed by Houf, Grcar, and Breiland [10] specifically for the analysis of vertical furnaces. (The actual simulation code used in our experiments is called TWAFER.) Given a set of powers, p_j, each call to TWAFER produces a set of temperatures for the entire furnace, from which the wafer temperatures are extracted. The heat transfer formulation is simplified by using mass lumping and one-dimensional approximations. The nonlinear transport equations are solved using the TWOPNT solver [9], which uses a Newton method with a time evolution feature that computes the steady-state solution. By varying the tolerances for the TWOPNT solver, it is possible to increase the accuracy

TABLE 3
Number of nonlinear iterations and wall clock time.

	BFGS		**TRPDS**	
RTOL	**Iter**	**Time**	**Iter**	**Time**
10^{-2}	3*	154.471	100	5217.98
10^{-3}	30*	1603.75	100	6872.32
10^{-4}	64	4470.97	100	8857.03
10^{-5}	31	2729.82	25	2786.17
10^{-6}	39	4145.88	33	4468.22
10^{-7}	34	4516.57	24	3832.82
10^{-8}	31	4612.38	26	4548.66
10^{-9}	31	5152.05	25	4969.36
10^{-10}	31	6044.57	22	5101.78
10^{-11}	31	6576.52	28	6915.97
10^{-12}	31	6600.24	23	5774.78
$*$ indicates the method did not converge				

of the steady-state solution at the cost of increasing the computational time. In particular, we have chosen to vary a parameter that determines the relative convergence tolerance, RTOL, for the steady-state solution of the underlying PDE.

There are many different parameter combinations that have been considered in previous studies of the TWAFER code [17]. For this particular example we used only one configuration, namely, a design problem with seven heater zones: one bottom heater and six equally sized side heaters. Each simulation used a model that contained 100 wafers with ten discretization points per wafer. Our initial guess for the powers was $p_0 = \{100, 200, 300, 2700, 100, 400, 2000\}$.

Table 3 contains the total number of iterations as well as the total wall clock time taken by each method in computing a solution. With respect to the total wall clock time, the new method is competitive with the standard BFGS method in almost all cases. In addition, the new method is more robust for the larger values of RTOL. These values correspond to a very loose convergence tolerance for the PDE solver in the TWAFER modeling code. In these cases, the standard BFGS method did not converge to a solution, while the new method still managed to proceed; the BFGS algorithm terminated with the trust region shrinking below its minimum allowed size. This failure to converge is probably due to large inaccuracies in the gradient evaluations due to the loose tolerances in the PDE solutions.

The resulting power values were then given to the TWAFER simulation using a value of RTOL $= 10^{-12}$. Figures 11–12 show the wafer temperatures that result with the computed powers. The interesting point here is that, even with relatively large values of the parameter RTOL, the resulting temperatures are still quite reasonable. As we have already noted, for these same values of RTOL the BFGS algorithm did not converge.

5. Summary. We have described a new class of algorithms for parallel optimization. The general framework consists of a trust-region model in which a nonstandard subproblem is solved using a PDS method that takes advantage of parallelism to solve the problem more efficiently on multiple processors. This new algorithm can be shown to have the same convergence properties as standard trust-region methods. In addition, the practical properties of PDS methods can be used to solve engineering optimization problems that are noisy or lack analytic derivatives.

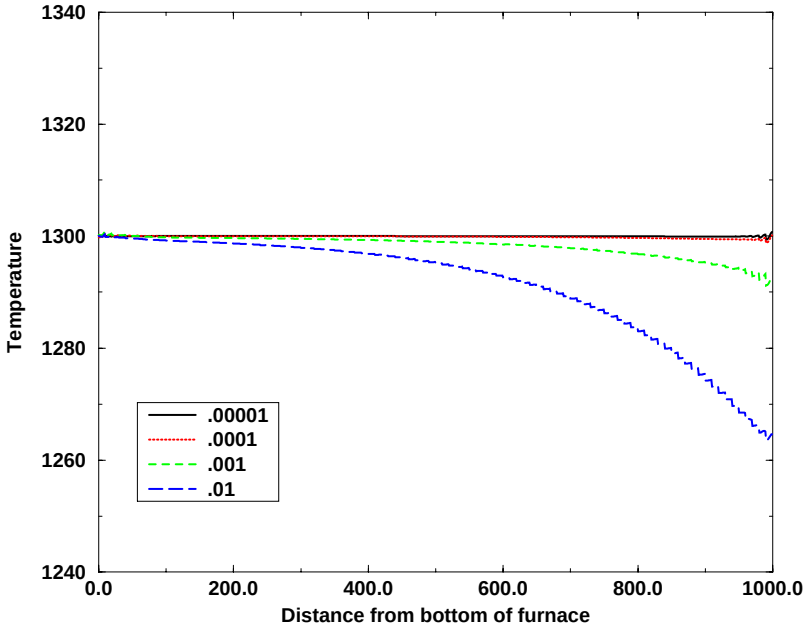

FIG. 11. *Computed temperatures for various values of RTOL using TRPDS.*

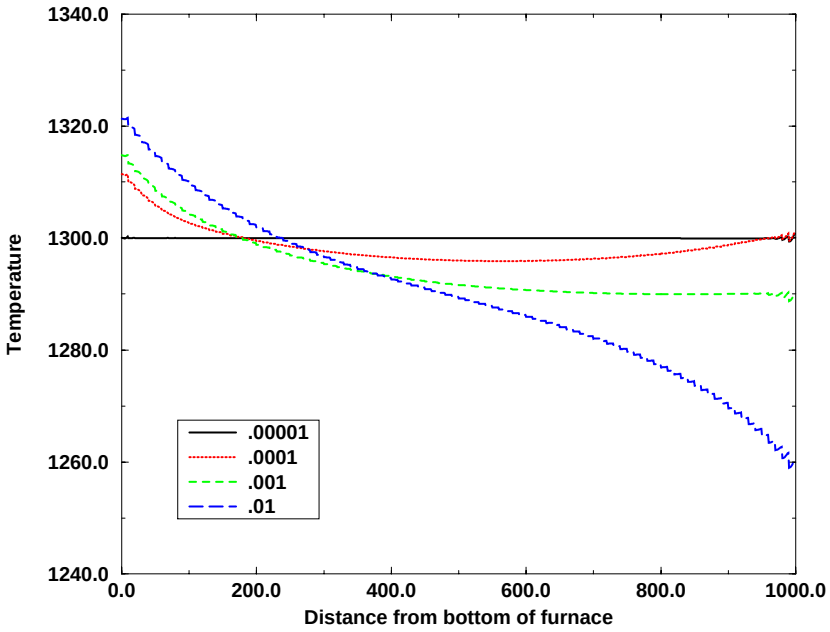

FIG. 12. *Computed temperatures for various values of RTOL using BFGS.*

This new class of algorithms was tested on a standard set of test problems, where it performed favorably against the traditional BFGS method. We also tested this new algorithm on a test case derived from an optimal design problem for a chemical vapor deposition furnace. The results indicate that the new method is competitive with the traditional BFGS method. In addition, the new method is more robust in the presence of noise that is generated by the use of less accurate PDE solvers. This is an important feature since many users would prefer to use less accurate PDE solvers in order to reduce the total computational time.

There are many new options to explore. In particular, it would be useful to develop strategies for bound constrained and general inequality constrained problems. It is also necessary to address issues related to the distribution of function evaluations in order to improve the efficiency of the algorithm. It would also be interesting to explore more general approximation models within this new framework.

Appendix. Tables 4–5 list the problems corresponding to the "Problem number" referred to in the various plots in the paper. Table 4 contains the first set of 24 test problems. Table 5, which also includes the dimension, contains the 22 CUTE test problems.

TABLE 4
Test problems.

Number	Problem
1	almost
2	broyden1a
3	broyden1b
4	broyden2a
5	broyden2b
6	bv
7	chain_singular
8	chain_wood
9	chebyquad
10	cragg_levy
11	epowell
12	erosen
13	gen_brown
14	gen_wood
15	ie
16	lin
17	lin0
18	lin1
19	penalty1
20	penalty2
21	toint_trig
22	tointbroy
23	trig
24	vardim

TABLE 5
CUTE problems.

Number	Problem	Dimension
1	arglinb	10
2	brownal	10
3	cosine	10
4	dixmaana	15
5	dixmaanl	15
6	eigenals	6
7	engval1	2
8	fletcbv2	10
9	freuroth	2
10	mancino	10
11	morebv	10
12	msqrtals	4
13	nondia	10
14	nonmsqrt	9
15	power	10
16	schmvett	3
17	sensors	2
18	sinquad	5
19	sparsine	10
20	sparsqur	10
21	tquartic	5
22	vareigvl	10

Acknowledgments. We wish to thank John Dennis and Matthias Heinkenschloss for many helpful discussions and for pointing out the relationship between our new algorithm and the framework for approximation models. We would also like to thank Vicki Howle and Suzanne Shontz for their work on parallel finite differencing and the comparisons of TRPDS to a speculative gradient trust-region algorithm. Finally, we thank an anonymous referee for many helpful comments and suggestions.

REFERENCES

[1] N. Alexandrov, J. E. Dennis, Jr., R. M. Lewis, and V. Torczon, *A trust region framework for managing the use of approximation models in optimization*, Structural Optim., 15 (1998), pp. 16–12.

[2] R. H. Byrd, R. B. Schnabel, and G. A. Shultz, *Parallel quasi-Newton methods for unconstrained optimization*, Math. Programming, 42 (1988), pp. 273–306.

[3] R. G. Carter, *On the global convergence of trust region algorithms using inexact gradient information*, SIAM J. Numer. Anal., 28 (1991), pp. 251–265.

[4] R. G. Carter, *Numerical experience with a class of algorithms for nonlinear optimization using inexact function and gradient information*, SIAM J. Sci. Comput., 14 (1993), pp. 368–388.

[5] A. R. Conn, N. I. M. Gould, and P. L. Toint, *Testing a class of methods for solving minimization problems with simple bounds on the variables*, Math. Comp., 50 (1988), pp. 399–430.

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

[7] J. E. Dennis, Jr., and V. Torczon, *Direct search methods on parallel machines*, SIAM J. Optim., 1 (1991), pp. 448–474.

[8] P. E. Gill, W. Murray, and M. H. Wright, *Practical Optimization*, Academic Press, San Diego, CA, 1981.

[9] J. F. Grcar, *TWOPNT Program for Boundary Value Problems, Version* 3.10, Tech. report SAND91–8230, Sandia National Laboratory, Livermore, CA, 1992.

[10] W. G. Houf, J. F. Grcar, and W. G. Breiland, *A model for low pressure chemical vapor deposition in a hot-wall tubular reactor*, Materials Science Engineering B, Solid State Materials for Advanced Technology, 17 (1993), pp. 163–171.

[11] P. D. Hough and J. C. Meza, *A Class of Trust-Region Methods for Parallel Optimization*, Tech. report SAND98-8245, Sandia National Laboratories, Livermore, CA, 1999.

[12] V. E. Howle, S. M. Shontz, and P. D. Hough, *Some Parallel Extensions to Optimization Methods in OPT++*, Tech. report SAND2000-8877, Sandia National Laboratories, Livermore, CA, 2000.

[13] L. Ingber, *Simulated annealing: Practice versus theory*, Math. Comput. Modelling, 18 (1993), pp. 29–57.

[14] P. Jog, J. Y. Suh, and D. Van Gucht, *Parallel genetic algorithms applied to the traveling salesman problem*, SIAM J. Optim., 1 (1991), pp. 515–529.

[15] C. T. Kelley, *A Shamanskii-like acceleration scheme for nonlinear equations at singular roots*, Math. Comp., 47 (1986), pp. 609–623.

[16] P. J. M. Laarhoven, *Parallel variable metric algorithms for unconstrained optimization*, Math. Programming, 33 (1985), pp. 68–81.

[17] C. D. Moen, P. A. Spence, and J. C. Meza, *Automatic differentiation for gradient-based optimization of radiatively heated microelectronics manufacturing equipment*, in Proceedings of the 6th AIAA/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization, Bellevue, WA, 1996, pp. 1167–1175.

[18] J. J. Moré, B. S. Garbow, and K. E. Hillstrom, *Testing unconstrained optimization software*, ACM Trans. Math. Software, 7 (1981), pp. 17–41.

[19] P. K. Phua and Y. Zeng, *Parallel Quasi-Newton Algorithms for Large-Scale Optimization*, Tech. report TRB2/95, National University of Singapore, Singapore, 1995.

[20] L. B. Rall, *Convergence of the Newton process to multiple solutions*, Numer. Math., 9 (1966), pp. 23–37.

[21] R. B. Schnabel, *A view of the limitations, opportunities, and challenges in parallel nonlinear optimization*, Parallel Comput., 21 (1995), pp. 875–905.

[22] T. A. Straeter, *A Parallel Variable Metric Optimization Algorithm*, Tech. report NASA TN D-7329, NASA, Langley Research Center, Hampton, VA, 1973.