Available online at www.sciencedirect.com

ScienceDirect

European Journal of Operational Research 182 (2007) 514–535

EUROPEAN JOURNAL OF OPERATIONAL RESEARCH

www.elsevier.com/locate/ejor

Continuous Optimization

Parallel radial basis function methods for the global optimization of expensive functions

Rommel G. Regis [a], Christine A. Shoemaker [b,*]

[a] *Cornell Theory Center and School of Operations Research & Industrial Engineering, Cornell University, Ithaca, NY 14853, USA*
[b] *School of Civil & Environmental Engineering and School of Operations Research & Industrial Engineering, Cornell University, 210 Hollister Hall, Ithaca, NY 14853, USA*

Received 7 February 2005; accepted 21 August 2006
Available online 17 November 2006

Abstract

We introduce a master–worker framework for parallel global optimization of computationally expensive functions using response surface models. In particular, we parallelize two radial basis function (RBF) methods for global optimization, namely, the RBF method by Gutmann [Gutmann, H.M., 2001a. A radial basis function method for global optimization. Journal of Global Optimization 19(3), 201–227] (Gutmann-RBF) and the RBF method by Regis and Shoemaker [Regis, R.G., Shoemaker, C.A., 2005. Constrained global optimization of expensive black box functions using radial basis functions, Journal of Global Optimization 31, 153–171] (CORS-RBF). We modify these algorithms so that they can generate multiple points for simultaneous evaluation in parallel. We compare the performance of the two parallel RBF methods with a parallel multistart derivative-based algorithm, a parallel multistart derivative-free trust-region algorithm, and a parallel evolutionary algorithm on eleven test problems and on a 6-dimensional groundwater bioremediation application. The results indicate that the two parallel RBF algorithms are generally better than the other three alternatives on most of the test problems. Moreover, the two parallel RBF algorithms have comparable performances on the test problems considered. Finally, we report good speedups for both parallel RBF algorithms when using a small number of processors.
© 2006 Elsevier B.V. All rights reserved.

Keywords: Global optimization; Parallel optimization; Radial basis function; Response surface model; Surrogate model; Function approximation; Expensive function

1. Introduction

1.1. Motivation and problem definition

In this paper, we explore the use of parallel processing to solve global optimization problems involving computationally expensive functions. Here, *computationally expensive* means that each function evaluation

* Corresponding author.
E-mail addresses: rgr6@cornell.edu (R.G. Regis), cas12@cornell.edu (C.A. Shoemaker).

0377-2217/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.ejor.2006.08.040

takes a long time, possibly several hours of computing time. These expensive functions can be found in many engineering applications including those that involve numerical solutions of partial differential equations (PDEs). For such global optimization problems, only a limited number of function evaluations is feasible when using only a single processor. However, global optimization of a nonconvex function generally requires a large number of function evaluations in order to obtain a high quality solution. Hence, our goal is to develop optimization algorithms that allow simultaneous function evaluations on multiple processors.

More precisely, let $\mathscr{D}$ be a compact subset of $\mathbb{R}^d$ and let $f : \mathscr{D} \to \mathbb{R}$ be a deterministic continuous function. The *global optimization problem* (GOP) is to find $x^* \in \mathscr{D}$ such that $f(x^*) \leqslant f(x)$ for all $x \in \mathscr{D}$. Note that the continuity of f on the compact set $\mathscr{D}$ guarantees the existence of a global minimum point for f on $\mathscr{D}$. A good introduction to GOP can be found in Horst et al. (2000) and Törn and Žilinskas (1989). In this paper, we assume that $\mathscr{D}$ is a closed hypercube in $\mathbb{R}^d$ and that f is a black box that results from an expensive simulation. We also assume that the derivatives of f are unavailable. Furthermore, we assume that the time to evaluate f is constant. In many practical applications, this assumption is reasonable and it makes load balancing in parallel processing straightforward. Our main objective is to find an approximate global minimizer for f on $\mathscr{D}$ quickly using parallel processors.

1.2. Serial algorithms for global optimization

First, we provide some background on serial global optimization methods for expensive functions. Since the derivatives of f are not available, one can perform either finite-differencing or automatic differentiation to obtain derivatives and then use standard nonlinear programming techniques combined with global search strategies (e.g. multistart). However, finite-differencing may be unreliable in the presence of noise. Moreover, automatic differentiation does not always produce accurate derivatives and it is not always applicable. For instance, automatic differentiation may yield inaccurate derivatives because of truncation error in numerical solutions of PDEs or because of the presence of branching in the simulation code (Nocedal and Wright, 1999). Finally, automatic differentiation is not an option if the source code for the expensive function is not available, which might be the case when using commercial simulation codes.

Because of the problems with the use of derivatives in some optimization problems, practitioners sometimes use *derivative-free methods*. Examples are pattern search algorithms (Torczon, 1997) and derivative-free trust-region methods (Conn et al., 1997; Marazzi and Nocedal, 2002; Powell, 2002, 2003). These local optimization methods can be used for global optimization via a multistart approach. Finally, heuristic global optimization methods like evolutionary algorithms and simulated annealing are also popular in the literature.

Another approach for the optimization of expensive functions relies on *response surface (RS) models* (also known as *surrogate models* or *metamodels*) for the expensive function. Examples of RS models are multivariate polynomials, which are used in traditional response surface methodology (Box and Draper, 1987; Myers and Montgomery, 1995), radial basis functions (RBFs) (Buhmann, 2003; Powell, 1992, 1999), neural networks, kriging models (Sacks et al., 1989; Cressie, 1993) and regression splines (Friedman, 1991). The derivative-free trust-region methods mentioned above rely on local linear or quadratic models of the expensive function. Brekelmans et al. (2005) also proposed a variant of the derivative-free trust-region method of Conn et al. (1997) for constrained nonlinear optimization that uses local linear approximations. Booker et al. (1999) used kriging interpolation to reduce the computational effort in pattern search algorithms. El-Beltagy et al. (1999) also used kriging to approximate fitness functions in an evolutionary algorithm. Finally, researchers in the field of multidisciplinary design optimization have used polynomial and kriging models for aerospace design (e.g. see Kaufman et al., 1996 or Giunta et al., 1997).

We focus on algorithms that utilize global RS models (i.e. RS models that approximate the expensive function globally over the entire domain $\mathscr{D}$). A typical strategy for these algorithms begins by evaluating the expensive function at the points of a space-filling experimental design. Then a global RS model is fit using the evaluated points and it is used to identify promising points for subsequent function evaluations. The process may be iterated where the RS model is updated with the addition of newly evaluated points. The selection of points for function evaluations is typically guided by the dual goals of finding a global minimizer for the RS model and sampling regions for which little information exists in order to improve the current RS model. It usually involves solving an *auxiliary global optimization subproblem* where the objective function is cheap to evaluate, and in many cases, the derivatives are easy to obtain. Examples of such algorithms are the EGO method by Jones et al. (1998), the RBF method by Gutmann (Björkman and Holmström, 2000; Gutmann, 2001a), and the CORS-RBF method by Regis and Shoemaker (2005).

1.3. Parallel algorithms for global optimization

Schnabel (1995) noted three levels at which parallelism can be incorporated in a nonlinear local optimization algorithm: (i) parallelization of individual evaluations of the objective function, constraints, and/or their derivatives; (ii) parallelization of the linear algebra in each iteration; and (iii) parallelization of the optimization process at a high level (i.e. perform multiple function, constraint, and/or derivative evaluations concurrently on multiple processors). We will focus on (iii) in this paper. Our approach leads to a *coarse-grained parallel algorithm*, which is a parallel algorithm where each processor performs a substantial amount of computation between points where it synchronizes or communicates with other processors.

For global optimization algorithms that are multistart versions of local optimization algorithms, a natural approach to parallelization is to assign starting points to processors and perform independent local optimizations asynchronously on the different processors (e.g. see Byrd et al., 1990; Dixon and Jha, 1993; Schnabel, 1995). In addition, we can also parallelize the local optimization component of these parallel multistart algorithms giving rise to parallel algorithms with the potential to use a large number of processors.

For other global optimization algorithms, parallelization can be achieved simply by performing multiple function evaluations concurrently on multiple processors in each iteration. This approach results in coarse-grained parallel algorithms when dealing with expensive functions. For instance, in the case of evolutionary algorithms, multiple fitness evaluations of offspring solutions can be carried out simultaneously on multiple processors in each generation (e.g. Yoon, 1997; Cantu-Paz, 2000). Also, in the case of global optimization algorithms based on RS models, we can use the RS model and other information from previously evaluated data points to select several points for simultaneous evaluation on multiple processors in each iteration. For example, in Sóbester et al. (2004), multiple evaluation points are generated by considering the best local maxima of the expected improvement surface of a kriging model for the expensive function.

1.4. Goal of investigation

In this paper, we parallelize two radial basis function (RBF) methods for global optimization: the RBF method by Gutmann (2001a) (Gutmann-RBF) and the CORS-RBF method by Regis and Shoemaker (2005). We assume that the time to evaluate the expensive function is roughly constant. Given P processors, we modify each of these RBF methods so that they can generate P distinct points for simultaneous function evaluation in parallel in each iteration. Because the objective function is expensive to evaluate, these parallel algorithms have a coarse-grained structure. We run our parallel RBF algorithms using 1, 2, 4, 6, 8 and 12 processors on eleven test problems and on a 6-dimensional groundwater bioremediation application and we compare them to three alternatives: (1) a parallel multistart derivative-based local optimization algorithm; (2) a parallel multistart derivative-free trust-region algorithm; and (3) a parallel evolutionary algorithm.

The results indicate that the two parallel RBF methods are generally better than the other three alternatives on most of the test problems. Moreover, the two parallel RBF methods have comparable performances on the test problems considered. The results also indicate good performance for these parallel RBF methods when using up to six processors on most of the test problems. Finally, preliminary comparisons indicate that our parallel RBF methods are competitive with the derivative-free variant of the parallel RBF method by Sóbester et al. (2004).

2. Synchronous master–worker parallel response surface methods for global optimization

2.1. General framework

We now present a general framework for parallel global optimization using response surface (RS) models. This framework includes the derivative-free variant of the parallel algorithm presented by Sóbester et al.

(2004). Assume that there are P available processors and that any two function evaluations take the same amount of time. The most natural idea is to fit/update an RS model at the beginning of each iteration using all available *data points* (i.e. points where the function values are known) and select P distinct points for simultaneous evaluation in parallel using the P processors.

We will utilize a master–worker framework for parallelization in which there is one master task and P worker tasks. Although there are really $P+1$ tasks, we will implement this using only P processors since for truly costly functions the workload for the master task will be much less than that of any worker task. Below is an outline for a parallel global optimization algorithm using RS models. In the notation below, n is the number of previously evaluated points, $\mathscr{A}_n = \{x_1, \ldots, x_n\}$ is the set of n previously evaluated points, and $s_n(x)$ is the RS model constructed using the points in $\mathscr{A}_n$ and their known function values.

```
Inputs:
    (1) A continuous real-valued function $f$ defined on a closed hypercube $\mathscr{D} \subseteq \mathbb{R}^d$.
    (2) The number of processors $P$.
    (3) A particular RS model, e.g. thin plate spline RBF augmented by a linear polynomial.
    (4) A set of initial evaluation points $\mathscr{I} = \{x_1, \ldots, x_{k \cdot P}\} \subseteq \mathscr{D}$, e.g. points from a space-filling experimental
        design (see Koehler and Owen, 1996). (Note that $|\mathscr{I}| = k \cdot P$ is divisible by $P$.)
    (5) The maximum number of function evaluations allowed denoted by maxeval.

Output: The best solution $x^* \in \mathscr{D}$ encountered by the algorithm.
Step 1. (Initialize tasks) Initialize the master task and each of the $P$ worker tasks.
Step 2. (Evaluate initial points) The master evenly distributes the initial evaluation points $\mathscr{I}$ to the $P$ workers.
        Each worker performs function evaluations at the points it received from the master and sends its
        results back to the master. For each result returned by a worker, the master updates the best function
        value encountered so far. The master waits for all workers to finish.
Step 3. The master sets $n := k \cdot P$ and $\mathscr{A}_n := \mathscr{I}$. While the termination condition is not satisfied do
    Step 3.1. (Fit/update response surface model) The master fits/updates a response surface model $s_n(x)$
              using the data points $\mathscr{B}_n = \{(x, f(x)) : x \in \mathscr{A}_n\}$.
    Step 3.2. (Generate evaluation points) Using the information in $\mathscr{B}_n$ and $s_n(x)$, the master generates a
              set $\mathscr{E}_n \subseteq \mathscr{D}$ of $P$ points for function evaluations and evenly distributes them to the $P$
              workers.
    Step 3.3. (Perform function evaluations) Each worker performs a function evaluation at the point it
              received from the master and sends its result back to the master.
    Step 3.4. (Update information) For each result returned by a worker, the master updates the best
              function value encountered so far. The master waits for all workers to finish, and then,
              it sets $\mathscr{A}_{n+P} := \mathscr{A}_n \cup \mathscr{E}_n$ and resets $n := n + P$.
end.
```

Note that we require the number of initial evaluation points to be divisible by P. This is reasonable since we have P processors and the function evaluation time is assumed to be constant. Because the objective function is expensive to evaluate, this parallel framework has a coarse-grained structure. The parallelization is straightforward and the main problem is simply how to select P distinct points for simultaneous evaluation in parallel. The RBF algorithms that we shall discuss below all have this general structure and they differ only in the procedure for selecting the P evaluation points in each iteration (Step 3.2 above).

Another possibility for parallelization is to perform multiple independent runs of a serial global optimization algorithm based on RS models. That is, given P processors, we can perform P simultaneous runs of the serial algorithm where different runs are initialized by different space-filling experimental designs. The best result obtained among the P independent runs is the output of this parallel algorithm. However, a major disadvantage of this approach is that the data points are not shared by the processors. Hence, the RS model that is constructed on each processor is not the most accurate possible since it does not utilize the data points that are available on other processors. Thus, we did not investigate this approach since we expect its performance to be worse than the above framework. On the other hand, if we require each processor to share its data points with every other processor so that the same RS model is constructed on each processor in every iteration, then we are essentially following the above framework.

2.2. Computing wall clock time

We can measure the performance of a parallel global optimization algorithm by determining the wall clock time it takes to get a solution with a specified level of accuracy. For test problems where the global minimum value is known, getting a solution with a specified level of accuracy means getting a solution with a relative error that is less than some user-specified value, say 1%. If f^* is the global minimum value for some test function and f_{best} is the best value obtained by an algorithm, then the *relative error* is given by $|f_{\text{best}} - f^*|/|f^*|$ provided that $f^* \neq 0$. When $f^* = 0$ or when the global minimum value is not known, we can simply measure the wall clock time required to get a solution whose function value is less than some user-specified value. From now on, whenever we talk about the wall clock time for a global optimization algorithm on a particular problem, we mean the wall clock time required to get a solution with a function value or relative error that is less than some pre-specified value.

The wall clock times for parallel global optimization algorithms that follow the above framework depend heavily on the choice of the initial evaluation points (i.e. the experimental design used). Hence, for a fixed number of processors, it is more appropriate to treat the wall clock time as a random variable whose value depends on the random set of initial evaluation points used. Let τ be the time to evaluate f on a single processor. (Recall from Section 1.1 our assumption that the time to evaluate f is constant.) Consider a parallel global optimization algorithm that follows the above framework. Fix the number of processors P and the experimental design (whose cardinality depends on P). The wall clock time for this parallel algorithm when using P processors is given by $W(P) = (A(P)/P)\tau + B(P)$, where $A(P)$ is the number of function evaluations required for the algorithm to get a specified level of accuracy while $B(P)$ is the wall clock time spent by the algorithm on everything else except function evaluations. In particular, $B(P)$ includes all the communication times between the master and the workers. Note that $W(P)$, $A(P)$ and $B(P)$ are all random variables. Hence, we consider the *expected wall clock time for P processors*:

$$E[W(P)] = (E[A(P)]/P)\tau + E[B(P)]. \tag{1}$$

In the computational experiments, multiple trials will be performed and $E[W(P)]$, $E[A(P)]$ and $E[B(P)]$ will be estimated by the corresponding average values over all trials.

In the above framework, note that the choice of the P evaluation points in any iteration is not influenced by whether the P evaluation points in the previous iteration were evaluated sequentially in serial or simultaneously in parallel. Moreover, it is reasonable to assume that the total time spent on communication between the master and the workers is negligible compared to the computation times for the expensive function evaluations. Hence, it is possible to estimate the wall clock times for an algorithm that follows the above framework by running it sequentially on a serial machine, i.e. by doing the function evaluations in Steps 2 and 3.3 sequentially instead of simultaneously. This is particularly useful for estimating wall clock times of parallel RBF methods on the test problems later.

2.3. Speedup and efficiency

Two commonly used measures of parallel performance are *speedup* and *efficiency*. The *speedup when using P processors* (Barr and Hickman, 1993) is given by:

$$\text{Speedup}(P) = \frac{\text{time required by the fastest serial algorithm to solve the problem on 1 processor}}{\text{time required by the parallel algorithm to solve the same problem using } P \text{ processors}}. \tag{2}$$

Also, the *efficiency when using P processors* is defined by: Efficiency(P) = Speedup$(P)/P$. In this context, solving the problem means getting a solution with a specified level of accuracy. Moreover, when the wall clock times depend on random initial conditions, then expected wall clock times are generally used for the numerator and denominator in (2).

The problem with the above definition of speedup is that it is not always possible to know the fastest serial algorithm for a given problem. However, in some cases, the numerator in (2) may be estimated using the time required by the parallel algorithm to solve the same problem using 1 processor. The resulting speedup is called *relative speedup* and the corresponding efficiency is called *relative efficiency*. In the computational experiments below, we will use relative speedup and relative efficiency to measure parallel performance.

3. A radial basis function model

We now present the radial basis function (RBF) interpolation model (Buhmann, 2003; Powell, 1992, 1999) that we used in our serial and parallel RBF algorithms. This RBF model was used as the basis of the RBF method by Gutmann (2001a).

Assume that we are given n distinct points $x_1, \dots, x_n \in \mathbb{R}^d$ where the function values $f(x_1), \dots, f(x_n)$ are known. We use an interpolant of the form

$$s_n(x) = \sum_{i=1}^{n} \lambda_i \phi(\|x - x_i\|) + p(x), \quad x \in \mathbb{R}^d, \tag{3}$$

where $\|\cdot\|$ is the Euclidean norm, $\lambda_i \in \mathbb{R}$ for $i = 1, \dots, n$, $p \in \Pi_m^d$ (the space of polynomials in d variables of degree less than or equal to m), and ϕ is one of the following forms: (1) *surface splines*: $\phi(r) = r^\kappa$, $\kappa \in N$, κ odd, or $\phi(r) = r^\kappa \log r$, $\kappa \in N$, κ even; (2) *multiquadrics*: $\phi(r) = (r^2 + \gamma^2)^\kappa$, $\kappa > 0$, $\kappa \notin N$; (3) *inverse multiquadrics*: $\phi(r) = (r^2 + \gamma^2)^\kappa$, $\kappa < 0$; (4) *Gaussians*: $\phi(r) = \mathrm{e}^{-\gamma r^2}$; where $r \geqslant 0$ and γ is a positive constant. Special cases lead to the *cubic spline* ($\phi(r) = r^3$) and the *thin plate spline* ($\phi(r) = r^2 \log r$).

Select a particular ϕ. Define the matrix $\Phi \in \mathbb{R}^{n \times n}$ by: $\Phi_{ij} := \phi(\|x_i - x_j\|)$, $i,j = 1, \dots, n$. Moreover, define m_ϕ to be $\lfloor \kappa/2 \rfloor$ if ϕ is a surface spline, $\lfloor \kappa \rfloor$ if ϕ is a multiquadric, and -1 if ϕ is an inverse multiquadric or a Gaussian. Let $m \geqslant m_\phi$ and let $\widehat{m}$ be the dimension of the linear space Π_m^d. (It is easy to check that $\widehat{m} = \binom{m+d}{d}$.) Also, let $p_1, \dots, p_{\widehat{m}}$ be a basis of Π_m^d, and define the matrix $\Gamma \in \mathbb{R}^{n \times \widehat{m}}$ as follows: $\Gamma_{ij} := p_j(x_i)$, $i = 1, \dots, n$; $j = 1, \dots, \widehat{m}$. Now, the RBF that interpolates $(x_1, f(x_1)), \dots, (x_n, f(x_n))$ is obtained by solving

$$\begin{pmatrix} \Phi & \Gamma \\ \Gamma^{\mathrm{T}} & 0 \end{pmatrix} \begin{pmatrix} \lambda \\ c \end{pmatrix} = \begin{pmatrix} F \\ 0_{\widehat{m}} \end{pmatrix}, \tag{4}$$

where $F = (f(x_1), \dots, f(x_n))^{\mathrm{T}}$, $\lambda = (\lambda_1, \dots, \lambda_n)^{\mathrm{T}} \in \mathbb{R}^n$ and $c = (c_1, \dots, c_{\widehat{m}})^{\mathrm{T}} \in \mathbb{R}^{\widehat{m}}$. In the case where $m = -1$, Γ is omitted from Eq. (4).

Powell (1992) showed that the coefficient matrix in Eq. (4) is invertible if and only if $\mathrm{rank}(\Gamma) = \widehat{m}$, where Γ is defined above and $\widehat{m} = \dim(\Pi_m^d)$. Hence, we must have $n \geqslant \widehat{m}$. Moreover, if the coefficient matrix in Eq. (4) is invertible, then it will remain invertible with the addition of new data points that are distinct from the previous ones.

4. Radial basis function methods for global optimization

4.1. Gutmann-RBF method

The RBF method by Gutmann (2001a) was based on a general idea proposed by Jones (1996). Let $x_1, \dots, x_n \in \mathcal{D}$ be the previously evaluated points and assume that we have a guess f_n^* of the global minimum value of f. We refer to f_n^* as a *target value*. Furthermore, for each $y \notin \{x_1, \dots, x_n\}$, assume that there is a unique function s_n^y, belonging to some linear space of functions, that interpolates the data points $(x_1, f(x_1)), \dots, (x_n, f(x_n))$ and the additional data point (y, f_n^*). Jones (1996) proposed that the next evaluation point $y \in \mathcal{D} \setminus \{x_1, \dots, x_n\}$ be chosen such that s_n^y is "most reasonable". Gutmann (2001a) interpreted "most reasonable" to mean "less bumpy" and found a suitable measure of bumpiness for RBFs given by the semi-norm σ defined by $\sigma(s_n) = \langle s_n, s_n \rangle := (-1)^{m_\phi + 1} \sum_{i=1}^{n} \lambda_i s_n(x_i)$, where s_n and m_ϕ are defined in Section 3. That is, the next evaluation point x_{n+1} is chosen to be the point $y \in \mathcal{D} \setminus \{x_1, \dots, x_n\}$ such that $\sigma(s_n^y)$ is a minimum. We shall refer to this method as *Gutmann-RBF*.

Next, we discuss the global minimization of $\sigma(s_n^y)$ over all $y \in \mathscr{D} \setminus \{x_1, \ldots, x_n\}$ in the case of RBFs. We refer to this as the *Bumpiness Minimization Subproblem (BMS)*. We recall the notation in Section 3. If $f_n^* < \inf_{x \in \mathscr{D}} s_n(x)$, Gutmann (2001b) showed that minimizing $\sigma(s_n^y)$ over all $y \in \mathscr{D} \setminus \{x_1, \ldots, x_n\}$ is equivalent to maximizing

$$h_n(y) := \frac{(-1)^{m_\phi+1}}{[s_n(y) - f_n^*]^2} \left[\phi(0) - \begin{pmatrix} u_n(y)^{\mathrm{T}} & \pi(y)^{\mathrm{T}} \end{pmatrix} \begin{pmatrix} \Phi & \Gamma \\ \Gamma^{\mathrm{T}} & 0 \end{pmatrix}^{-1} \begin{pmatrix} u_n(y) \\ \pi(y) \end{pmatrix} \right], \quad y \in \mathscr{D}, \tag{5}$$

where ϕ, m_ϕ, Φ and Γ are defined in Section 3, and $u_n(y) := (\phi(\|y - x_1\|), \ldots, \phi(\|y - x_n\|))^{\mathrm{T}}$ and $\pi(y) := (p_1(y), \ldots, p_{\widehat{m}}(y))^{\mathrm{T}}$. However, if $f_n^* = \inf_{x \in \mathscr{D}} s_n(x)$, then any global minimizer of $s_n(x)$ that is not a previously evaluated point is also a global minimizer of $\sigma(s_n^y)$ over all $y \in \mathscr{D} \setminus \{x_1, \ldots, x_n\}$ (Gutmann, 2001a).

The target values are selected in the interval $[-\infty, \inf_{x \in \mathscr{D}} s_n(x)]$ and are set by performing cycles of $N+1$ iterations, where each cycle starts with a low target value (global search) and ends with a high target value equal or close to $\inf_{x \in \mathscr{D}} s_n(x)$ (local search). We refer to N as the *cycle length*. When ϕ is linear, cubic and the thin plate spline, Gutmann (2001a) showed that, by choosing target values in a particular manner, convergence to the global minimum is guaranteed for any continuous function defined over a compact set.

In this investigation, we adopt the procedure used by Gutmann (2001a) and by Björkman and Holmström (2000) in setting the target values. Let α be a permutation of $\{1, \ldots, n\}$ such that $f(x_{\alpha(1)}) \leqslant \cdots \leqslant f(x_{\alpha(n)})$. Also, let n_0 be the number of initial evaluation points (i.e. number of points in the space-filling experimental design). Set the cycle length to $N = 5$, and for $n \geqslant n_0$, set

$$f_n^* = \inf_{x \in \mathscr{D}} s_n(x) - W_n \cdot \left(\max_{1 \leqslant i \leqslant k_n} f(x_{\alpha(i)}) - \inf_{x \in \mathscr{D}} s_n(x) \right) = \inf_{x \in \mathscr{D}} s_n(x) - W_n \cdot \left(f(x_{\alpha(k_n)}) - \inf_{x \in \mathscr{D}} s_n(x) \right), \tag{6}$$

where

$$W_n = \left[\frac{\mathrm{mod}(N - (n - n_0),\ N+1)}{N} \right]^2 \quad \text{and} \quad k_n = \begin{cases} n & \text{if } \mathrm{mod}(n - n_0, N+1) = 0, \\ k_{n-1} - \lfloor (n - n_0)/N \rfloor & \text{otherwise.} \end{cases} \tag{7}$$

Gutmann (2001b) noted that $s_n, h_n \in C^{\kappa-1}(\mathbb{R}^d)$ in the case of the surface splines while $s_n, h_n \in C^\infty(\mathbb{R}^d)$ in the multiquadric, inverse multiquadric and Gaussian cases. Hence, we can use standard gradient-based optimization techniques to find the global minimum of s_n (needed in setting the target value) and the global maximum of h_n. However, Björkman and Holmström (2000) found that maximizing $h_n(y)$ could lead to numerical difficulties. Hence, we adopted their procedure of minimizing $-\log(h_n(y))$ instead when selecting the next evaluation point x_{n+1}.

4.2. CORS-RBF method

As before, let $x_1, \ldots, x_n \in \mathscr{D}$ be the previously evaluated points. We define the *maximin distance* in $\mathscr{D}$ relative to $x_1, \ldots, x_n$ to be $\Delta_n = \max_{\tilde{x} \in \mathscr{D}} \min_{1 \leqslant j \leqslant n} \|\tilde{x} - x_j\|$. Clearly, for any $x \in \mathscr{D}$, the distance between x and any previously evaluated point is at most Δ_n. In the CORS-RBF method developed by Regis and Shoemaker (2005), the next evaluation point is chosen to be the point $y \in \mathscr{D}$ that solves the following constrained optimization problem:

$$\text{Minimize } \{s_n(x) : x \in \mathscr{D},\ \|x - x_j\| \geqslant \beta_n \Delta_n,\ j = 1, \ldots, n\}, \tag{8}$$

where Δ_n is given above and $0 \leqslant \beta_n \leqslant 1$ is a parameter to be set by the user. That is, the next evaluation point is chosen to be the point $y \in \mathscr{D}$ that minimizes the RBF model subject to the constraints that y be of distance at least $\beta_n \Delta_n$ from each previously evaluated point. We shall refer to the parameter β_n as a *distance factor*. Also, we shall refer to the constrained optimization problem (8) as the *CORS-RBF auxiliary optimization subproblem*. A procedure for solving this subproblem is described in Regis and Shoemaker (2005). If $\limsup_{n \to \infty} \beta_n > 0$, then CORS-RBF converges to the global minimum of an arbitrary continuous function defined over the closed hypercube $\mathscr{D}$ (Regis and Shoemaker, 2005).

In the implementation of CORS-RBF, we balance global and local search by performing cycles of $N+1$ iterations, where each cycle employs a range of values for β_n, starting with a high value close to 1 (global search) and ending with $\beta_n = 0$ (local search). More precisely, if n_0 is the number of initial evaluation points, we have $\beta_n = \beta_{n+N+1}$ for all $n \geqslant n_0$ and $1 \geqslant \beta_{n_0} \geqslant \beta_{n_0+1} \geqslant \cdots \geqslant \beta_{n_0+N} = 0$. We refer to N as the *cycle length* and we refer to the sequence $\langle \beta_{n_0}, \beta_{n_0+1}, \ldots, \beta_{n_0+N} = 0 \rangle$ as the *search pattern*.

5. Parallelization of radial basis function methods

5.1. Parallel Gutmann-RBF method

Gutmann-RBF requires a target value (guess of the global minimum) in order to generate the evaluation point in each iteration. Given P processors, the most natural way to parallelize this method is to choose P distinct target values in $[-\infty, \inf_{x\in\mathscr{D}} s_n(x)]$ and solve the corresponding BMS in a given parallel iteration.

Recall from Section 4.1 that the target values are set in cycles of $N+1$ iterations, where N is the cycle length. Let N_s be the cycle length for the standard implementation of Gutmann-RBF. (Recall also that $N_s = 5$.) We consider two different parallelization strategies depending on whether $P \leqslant N_s + 1$ or $P > N_s + 1$.

When $P \leqslant N_s + 1$, we set the target values in each iteration as follows. Assume that at the beginning of an iteration there are exactly n previously evaluated points (note that n is divisible by P). As before, we let n_0 be the number of initial evaluation points and define the permutation α as in Section 4.1. We then set the P target values $f_n^*, f_{n+1}^*, \ldots, f_{n+P-1}^*$ in one iteration in a manner similar to that in the serial version:

$$f_{n+i}^* = \inf_{x\in\mathscr{D}} s_n(x) - W_{n+i} \cdot \left(f(x_{\alpha(k_{n+i})}) - \inf_{x\in\mathscr{D}} s_n(x)\right), \quad i = 0, 1, \ldots, P-1, \tag{9}$$

where the sequences $\{W_n\}_{n\geqslant n_0}$ and $\{k_n\}_{n\geqslant n_0}$ are defined as before (in Eq. (7)) and $N = N_s = 5$.

Now, if we follow the above strategy for $P > N_s + 1$, then two of the target values $f_n^*, f_{n+1}^*, \ldots, f_{n+P-1}^*$ could become identical since $\{W_n\}_{n\geqslant n_0}$ is a cyclic sequence and this could result in the same evaluation point. To get distinct target values when $P > N_s + 1$, we set $N = P - 1$ and define the target values as in Eq. (9). By setting N this way, note that an iteration of the parallel version now corresponds to a cycle of $N + 1 = P$ iterations of the serial version.

However, even if the target values are distinct, there is no guarantee that we will obtain distinct evaluation points since a solution to the BMS (i.e. finding $\inf_{y\in\mathscr{D}} \sigma(s_n^y)$) corresponding to some target value might also be a solution to the BMS corresponding to a different target value. One way around this problem is to remove the redundant points generated by the different target values and consider additional target values at random. To get more diversity in the evaluation points, we consider random target values that are close to the global minimum of the RBF model (i.e. high random target values) and also low random target values. We will continue considering random target values until we get exactly P distinct evaluation points. More precisely, each additional target value will be set as follows: $f^* = \inf_{x\in\mathscr{D}} s_n(x) - W \cdot (\max_{1\leqslant i\leqslant n} f(x_i) - \inf_{x\in\mathscr{D}} s_n(x))$, where W is set in the following manner. First, generate a uniform random number ω_1 between 0 and 1. Next, generate a standard normal random number ω_2 and set $W = 0.05|\omega_2|$ if $\omega_1 \leqslant 0.5$ or $W = |\omega_2|$ otherwise.

Note that it is possible that there are only a few solutions to the BMS for the entire range of possible target values in some iteration. To prevent the above procedure from getting trapped in an infinite loop, we keep track of the total number of *consecutive failures* in each iteration. Here, a *failure* is defined to be a target value that did not yield a point that is distinct from previously selected points within that iteration. When some threshold number of consecutive failures have occurred in a given iteration, we simply successively generate enough points, each one of which is as far away as possible from all previously evaluated points and previously selected points.

5.2. Parallel CORS-RBF method

In the original (serial) implementation of CORS-RBF, two search patterns were used: $\langle 0.95, 0.25, 0.05, 0.03, 0 \rangle$ and $\langle 0.9, 0.75, 0.25, 0.05, 0.03, 0 \rangle$. For comparison purposes with Gutmann-RBF, we parallelize the latter search pattern that has a cycle length of 5. As before, we parallelize CORS-RBF depending on whether $P \leqslant N_s + 1$ or $P > N_s + 1$, where N_s is the cycle length for the standard implementation, which we have now chosen to be 5.

Given P processors, we need to generate P distinct evaluation points in each iteration. Suppose that in a given iteration we consider P *distinct* distance factors (i.e. β's). Note that there is no guarantee that the resulting auxiliary optimization problems (see (8)) will yield distinct points. In a given iteration, the maximin distance is fixed and an optimal solution for an auxiliary problem corresponding to a given distance factor β might also be an optimal solution for another auxiliary problem corresponding to some distance factor $\beta' < \beta$. To get around this problem, we can solve the auxiliary problems sequentially while requiring that each selected point be far from previously evaluated points and *previously selected points within that iteration*. With this approach, we are always guaranteed to get distinct points even when some of the β values considered in an iteration are not distinct.

More precisely, let $\beta_n, \beta_{n+1}, \ldots, \beta_{n+P-1}$ be the (not necessarily distinct) values of β that we are considering in a given iteration. The P distinct evaluation points $y_1, \ldots, y_P$ in this iteration are recursively determined as follows: For $i = 1, \ldots, P$, y_i is a solution to the following optimization problem:

$$\begin{aligned} &\min \ s_n(x) \\ &\text{subject to} \\ &\quad \|x - x_j\| \geqslant \beta_{n+i-1}\Delta_{n,i}, \quad j = 1, \ldots, n, \\ &\quad \|x - y_j\| \geqslant \beta_{n+i-1}\Delta_{n,i}, \quad j = 1, \ldots, i-1, \\ &\quad x \in \mathscr{D}, \end{aligned} \tag{10}$$

where

$$\Delta_{n,i} = \max_{\tilde{x} \in \mathscr{D}} \min \left\{ \min_{1 \leqslant j \leqslant n} \|\tilde{x} - x_j\|, \min_{1 \leqslant j \leqslant i-1} \|\tilde{x} - y_j\| \right\}.$$

That is, for $i = 1, \ldots, P$, y_i is required to be of distance at least $\beta_{n+i-1}\Delta_{n,i}$ from the previously evaluated points $x_1, \ldots, x_n$ and also from the previously selected points $y_1, \ldots, y_{i-1}$ in the current iteration. Here, $\Delta_{n,i}$ is the maximin distance in $\mathscr{D}$ relative to $x_1, \ldots, x_n, y_1, \ldots, y_{i-1}$.

To complete the parallelization of CORS-RBF, we show how to select the distance factors $\beta_n, \beta_{n+1}, \ldots, \beta_{n+P-1}$ in a given iteration. Let $\langle \zeta_1, \ldots, \zeta_{N_s+1} \rangle = \langle 0.9, 0.75, 0.25, 0.05, 0.03, 0 \rangle$. As before, we assume that n_0 is the number of initial evaluation points. We consider two cases: $P \leqslant N_s + 1$ and $P > N_s + 1$.

When $P \leqslant N_s + 1$, we first define the cyclic sequence

$$j(\ell) = \begin{cases} N_s + 1 & \text{if } \operatorname{mod}(\ell - n_0 + 1, N_s + 1) = 0, \\ \operatorname{mod}(\ell - n_0 + 1, N_s + 1) & \text{otherwise}, \end{cases} \qquad \ell \geqslant n_0.$$

Now, set $\beta_{n+i} := \zeta_{j(n+i)}$, $i = 0, 1, \ldots, P-1$. In other words, we successively take P consecutive distance factors from the search pattern $\langle \zeta_1, \ldots, \zeta_{N_s+1} = 0 \rangle$, going back to the beginning whenever we reach the end of the cycle. For example, when $P = 4$, the distance factors for the first iteration are $\{0.9, 0.75, 0.25, 0.05\}$, the distance factors for the second iteration are $\{0.03, 0, 0.9, 0.75\}$, and so on.

Next, assume that $P > N_s + 1$. If we follow the procedure for $P \leqslant N_s + 1$, then some iterations will have two distance factors with the value 0. In these iterations, we do not expect to get distinct candidate points since a distance factor of 0 results in the global minimization of $s_n(x)$ over $\mathscr{D}$ with no distance constraints. Hence, we adopt a different strategy. First, define $\zeta'_1, \ldots, \zeta'_{N_s}$ as follows: For $i = 1, \ldots, N_s$, $\zeta'_i = \zeta_{N_s - ((i-1)/2)}$ if i is odd while $\zeta'_i = \zeta_{(i/2)}$ if i is even. Next, define $\xi'_1 = \zeta_{N_s+1} = 0$ and, for $i = 2, \ldots, P$, define $\xi'_i = \zeta'_{h(i)}$, where $h(i) = N_s$ if $\operatorname{mod}(i-1, N_s) = 0$ and $h(i) = \operatorname{mod}(i-1, N_s)$ otherwise. Let $\xi_1, \ldots, \xi_P = 0$ be the values of $\xi'_1, \ldots, \xi'_P$ sorted in descending order. Now, set $\beta_{n+i} := \xi_{i+1}$, $i = 0, 1, \ldots, P-1$.

The above procedure works as follows. As more processors $> N_s + 1$ are added, we will include additional distance factors to $\langle \zeta_1, \ldots, \zeta_{N_s+1} \rangle = \langle 0.9, 0.75, 0.25, 0.05, 0.03, 0 \rangle$ beginning with the smallest positive distance factor in the above serial search pattern (i.e. $\zeta_5 = 0.03$), followed by the largest distance factor (i.e. $\zeta_1 = 0.9$), then followed by the second smallest positive distance factor (i.e. $\zeta_4 = 0.05$), then followed by the second largest distance factor (i.e. $\zeta_2 = 0.75$), and so on. Once we have chosen all the distance factors, we then arrange them in descending order to form the search pattern for each iteration of Parallel CORS-RBF. For example, the distance factors for $P = 8$ and 12 are $\{0.9, 0.9, 0.75, 0.25, 0.05, 0.03, 0.03, 0\}$ and $\{0.9, 0.9, 0.75, 0.75, 0.25, 0.25, 0.05, 0.05, 0.03, 0.03, 0.03, 0\}$, respectively.

5.3. Modifications of the RBF methods

In the computational experiments, our serial and parallel RBF methods will be initialized by using symmetric Latin hypercube designs (SLHD) (Ye et al., 2000). Regis and Shoemaker (2006) found that the average-case performance of Gutmann-RBF and CORS-RBF when initialized by SLHDs can be improved by implementing a complete restart strategy whenever the algorithm fails to make any substantial progress after some user-specified threshold number of consecutive iterations. Here, we define *substantial progress* to mean an improvement of at least some percentage (0.1% in this study) of the absolute value of the previous best function value. In addition, Regis and Shoemaker (2006) noted that we can ensure a better balance between local and global search by restricting the BMS in Gutmann-RBF (see Section 4.1) to a smaller hypercube centered at a global minimizer of the RBF model $s_n(x)$.

The complete restart strategy is implemented as follows (Regis and Shoemaker, 2006): Restart the algorithm from scratch using a new SLHD whenever it fails to make any substantial progress after $5(N+1)$ consecutive iterations, where N is the cycle length of the RBF method. When measuring the performance of these algorithms (in terms of number of function evaluations or wall clock time), we also take into account all the iterations done prior to any restarts.

The restriction on the BMS in Gutmann-RBF is implemented as follows (Regis and Shoemaker, 2006): Find the point $y \in \mathscr{D} = [a, b] \subseteq \mathbb{R}^d$ that minimizes $\sigma(s_n^y)$ in Section 4.1 subject to the constraint that $y \in [z_n^* - \rho_n(b-a), z_n^* + \rho_n(b-a)] \cap \mathscr{D}$, where z_n^* is a global minimum point of $s_n(x)$ and $\rho_n = \rho_n(W_n)$ is defined as follows: $\rho_n = v\sqrt{W_n}$ if $0 \leqslant W_n \leqslant U$ and $\rho_n = 1$ otherwise. Here, $0 < v < 1$ and $U > 0$ are parameters to be specified and W_n is the weight parameter from Eqs. (6) and (7) in Section 4.1. In the numerical experiments, we will set $v = 0.5$ and $U = 0.25$.

The parallel RBF algorithms described above are implemented using these modifications. Hence, we refer to these parallel algorithms as *ParCGRBF-R (Parallel Controlled Gutmann-RBF with Restart)* and *ParCORSRBF-R (Parallel CORS-RBF with Restart)*. These parallel RBF algorithms are restarted from scratch using a new SLHD whenever they fail to make any substantial progress after $\max(5, \lceil 5(N+1)/P \rceil)$ consecutive iterations, where N is the cycle length of the RBF method and P is the number of processors.

6. Alternative parallel global optimization methods

To assess the significance of our parallel RBF algorithms, we compare them to three alternative parallel global optimization methods, namely, a parallel evolutionary algorithm, a parallel multistart derivative-based algorithm and a parallel multistart derivative-free trust-region algorithm. In addition, we also perform some preliminary comparisons with the derivative-free variant of the parallel RBF method by Sóbester et al. (2004). We discuss these algorithms below.

6.1. Parallel differential evolution

Among the many possible evolutionary algorithms for continuous optimization, we parallelize *Differential Evolution* (Storn and Price, 1997) since it proved to be the fastest genetic type algorithm during the First International IEEE Competition on Evolutionary Optimization in 1996. It may not be the best evolutionary algorithm for our test problems but it is certainly a reasonable candidate since it works well on continuous optimization problems.

Differential Evolution (DE) maintains a population of constant size N_{pop} in each generation, and so, its parallelization can be easily achieved by choosing N_{pop} to be divisible by the number of processors P so that no processor becomes idle in each generation. (Again, recall our assumption that function evaluations take roughly the same time.) Moreover, we also required that $N_{\text{pop}} \geqslant 5d$, where d is the dimension of the problem. Thus, we set N_{pop} to be the smallest multiple of P that is $\geqslant 5d$, i.e. $N_{\text{pop}} := \lceil 5d/P \rceil P$. We used the matlab code for DE which is available in the DE homepage http://www.icsi.berkeley.edu/~storn/code.html and we also used the default parameters for the algorithm. We refer to the resulting parallel algorithm as *ParDE*.

6.2. Parallel multistart

We use an asynchronous master–worker parallel multistart procedure where the master selects the starting points for the local minimization runs while each worker performs a local minimization run from a starting point provided by the master. The basic idea is that the starting point for any local minimization run by any worker is chosen by the master to be as far away as possible from all the previous and current local minimization trajectories by all the workers.

A derivative-based method and a derivative-free trust-region method will be used for the underlying local minimization algorithm of our multistart procedure. For the derivative-based method, the gradient information will be obtained by finite differencing. The derivative-based method that will be used for all test problems, except the groundwater problem, is sequential quadratic programming (SQP), which is implemented in the FMINCON routine from the Matlab Optimization Toolbox (The Mathworks, 2004). The groundwater bioremediation problem proved to be a disaster for FMINCON because the problem turned out to be nonsmooth and nondifferentiable in certain regions, an unfortunate feature of many real world problems that is typically due to the inaccuracies in simulation codes. Hence, we will use the implicit filtering (ImFil) algorithm (Gilmore and Kelley, 1995) as implemented by Kelley (1999) for the groundwater problem. The derivative-free trust-region method that will be used is UOBYQA (Powell, 2002), which is implemented in Matlab by Vanden Berghen and Bersini (2005). We refer to these parallel algorithms as *Parallel Multistart SQP/ImFil (ParMult-SQP/ImFil)* and *Parallel Multistart UOBYQA (ParMult-UOBYQA)*.

6.3. Alternative parallel RBF method

The Parallel Gradient-Enhanced RBF (ParGERBF) method by Sóbester et al. (2004) was designed for problems where all the gradient components are available at very little cost (e.g. via a perturbation analysis of a finite element model) or at the cost of around one function evaluation (e.g. in the case of an adjoint CFD model). However, their method can also be used when derivatives are not available. We compare our parallel RBF methods with the derivative-free variant of ParGERBF, which we refer to as *ParDFRBF*.

In ParGERBF, a Gaussian RBF is used and the function evaluation points in each parallel iteration are chosen to be the local maximizers of the *expected improvement function* (Jones et al., 1998). Hence, ParGERBF is essentially a parallel version of the EGO method by Jones et al. (1998) except that its RBF model is equivalent to a simpler version of the kriging model used by Jones et al. (1998). We are not aware of any convergence proof for the EGO method except in the 1-dimensional case (Jones et al., 1998; Locatelli, 1997). In contrast, Gutmann-RBF and CORS-RBF have convergence proofs for the multi-dimensional case.

7. Computational experiments

7.1. Synthetic test problems

We compare the performance of the different parallel algorithms on eleven synthetic test problems with dimensions ranging from 2 to 6. The test problems include five of the Dixon and Szegö (1978) functions (GP, HA3, SH7, SH10, HA6), three Schoen (1993) functions (SC3, SC4, SC5), the 2-dimensional Camel

Table 1
Test functions for the computational experiments

Test function	Domain	No. of local min	Global min value	Test function	Domain	No. of local min	Global min value
GP	$[-2,2]^2$	4	3	SC4	$[0,1]^4$	$\geqslant 4$	13.3611
CA	$[-5,5]^2$	6	−1.0316285	MRO5	$[-1,1]^5$	>50	≈−3.2541
HA3	$[0,1]^3$	4	−3.86	MAC5	$[-2.048,2.048]^5$	>500	−7.125
SC3	$[0,1]^3$	$\geqslant 5$	5.9303	SC5	$[0,1]^5$	$\geqslant 4$	21.5615
SH7	$[0,10]^4$	7	−10.4029	HA6	$[0,1]^6$	4	−3.32
SH10	$[0,10]^4$	10	−10.5364				

(CA) function (Branin, 1972), and the 5-dimensional Modified Rosenbrock (MRO5) and Modified Ackley (MAC5) functions from Sóbester et al. (2004). Table 1 summarizes the characteristics of these test problems.

Our test problems are multimodal but they are not really expensive to evaluate. However, it is still possible to conduct meaningful studies of parallel performance for the different algorithms by assuming that these functions are computationally expensive. It is reasonable to expect that the relative performance of parallel algorithms on these functions will mimic performance on truly expensive functions with similar surfaces.

7.2. Groundwater bioremediation

We apply our parallel RBF methods to the problem of minimizing the cost of groundwater bioremediation (Minsker and Shoemaker, 1998; Yoon and Shoemaker, 1999). Groundwater bioremediation involves using injection wells that will supply electron acceptors (e.g. oxygen) or nutrients (e.g. nitrogen) into the groundwater in order to promote the growth of the soil bacteria that can transform the contaminants into harmless substances. The cost of cleaning up polluted groundwater could run into tens of millions of dollars, and so, an efficient global optimization algorithm could result in huge savings for environmental cleanup.

In this investigation, we assume that the locations of the injection wells have been fixed and that we wish to determine the pumping rates for these injection wells. There are also monitoring wells for measuring the concentrations of the contaminants at specific locations. We use a 2-dimensional finite element simulation model called Bio2D (Taylor, 1993), which describes groundwater flow and the changes in the concentrations of the contaminant, oxygen and biomass. We evenly divide the entire planning horizon into management periods. Our problem is to determine the pumping rates for each injection well at the beginning of each management period that minimize the total pumping cost subject to some constraints on the contaminant concentration at the monitoring wells. This problem was formulated by Yoon and Shoemaker (1999) as a box-constrained global optimization problem by introducing a penalty term for the nonlinear constraints into the total pumping cost objective function.

More specifically, we consider a hypothetical contaminated aquifer whose characteristics are symmetric about a horizontal axis. There are 6 injection wells and 84 monitoring wells that are also symmetrically arranged. Because of the symmetry, we only need to make pumping decisions for the 3 injection wells on one side of the axis of symmetry. There are two management periods, giving us six decision variables for our optimization problem. The maximum pumping rate was set to 1 so the domain of the problem is $[0,1]^6$. We shall refer to this groundwater bioremediation problem as GWB6.

The simulation times for groundwater bioremediation problems can take a long time. Shoemaker et al. (2001) report simulation times that vary between minutes and hours depending on the size of the modeled region. The example used in this investigation is a much smaller grid and requires only about 0.45 seconds to run each simulation on a 2.4 GHz Pentium 4 desktop. However, this problem is reflective of the type of function used in more complex groundwater bioremediation problems.

7.3. Experimental setup

Each of the five parallel algorithms (ParCGRBF-R, ParCORSRBF-R, ParDE, ParMult-SQP/ImFil, ParMult-UOBYQA) are applied to each of the 11 test problems and to the 6-dimensional groundwater application using $P = 1, 2, 4, 6, 8, 12$ processors. Moreover, since the performances of these parallel algorithms depend on some random initial conditions, we perform 30 trials of each parallel algorithm on each problem and for each processor setting. For the parallel RBF methods, each run uses a different randomly generated symmetric Latin hypercube design (SLHD) (Ye et al., 2000) for the initial evaluation points. The size of the SLHD is equal to the smallest multiple of P that is greater than or equal to $(d+1)(d+2)/2$, where P is the number of processors and d is the dimension of the problem. Recall from Section 5.3 that an RBF algorithm will be restarted completely from scratch, using a new randomly generated SLHD, whenever the algorithm fails to make any substantial progress after $\max(5, \lceil 30/P \rceil)$ consecutive iterations.

The different algorithms are compared in terms of average wall clock time required to get a solution with relative error <1%. In the case of GWB6, where the global minimum value is not known, we measure the wall clock time required to get a solution whose function value is less than or equal to 500. Since the test problems are not really expensive to evaluate, we present the results as though the test functions are computationally expensive and express the average wall clock time as a function of a parameter τ which is the time it takes to do one function evaluation.

In the two RBF methods, we adopt the strategy used by Gutmann (2001a), Björkman and Holmström (2000), and Regis and Shoemaker (2005) of replacing large function values by the median of all available function values whenever these values exceed the median. This transformation prevents oscillations in the RBF interpolant that are due to large differences in function values.

All numerical computations are performed in Matlab using several 2.4 GHz Processor Pentium 4 machines. As noted in Section 2.2, the wall clock times for the different parallel optimization methods can be estimated reasonably well by running the parallel algorithms sequentially. For the parallel RBF algorithms, the global optimization subproblems for generating the iterates are solved using a multistart implementation of FMINCON from the Matlab Optimization Toolbox (The Mathworks, 2004). In addition, we supply FMINCON with the gradients of the objective and constraint functions of these optimization subproblems.

8. Results and discussion

8.1. Comparison of average wall clock times

The average wall clock times for the different parallel algorithms on the test problems using different processor settings are shown in Table 2. The average wall clock times are given in the form $C(P)\tau + D(P)$, where τ is the wall clock time in seconds for one function evaluation and $C(P)$ and $D(P)$ are given in the table. From Eq. (1) in Section 2.2, $C(P)$ is the average number of function evaluations needed to get a relative error of $<1\%$ divided by the number of processors P. For the GWB6 problem, the values of $C(P)\tau + D(P)$ represent the average wall clock times required to get a total pumping and penalty cost of $\leqslant 500$.

Fig. 1 shows bar graphs of the average wall clock times versus the number of processors for ParCGRBF-R, ParCORSRBF-R and the two parallel multistart algorithms assuming that each function evaluation takes 1 hour, i.e. $\tau = 3600$ seconds. The error bars represent 95% confidence intervals for the mean, i.e. each side of the error bar has length equal to about 1.96 times the standard error of the mean. We did not include ParDE in the comparison plots because its performance is much worse than the other parallel methods and we want to highlight the differences between the other methods.

From Table 2, it is clear that ParCGRBF-R and ParCORSRBF-R are much better than ParDE. For example, assuming that $\tau = 3600$ seconds, the estimated wall clock time for ParDE when using 12 processors on the HA6 test problem is about 15.6 times the wall clock time of ParCORSRBF-R when using 12 processors on the same problem. These results suggest that evolutionary algorithms are not really appropriate for expensive functions even if parallel processors are available.

From Table 2 and Fig. 1, we can see that ParCGRBF-R and ParCORSRBF-R are much better or at least comparable to ParMult-SQP/ImFil on all test problems and processor settings, except on HA3 ($P = 1, 2$) and HA6 ($P = 1$). ParCGRBF-R and ParCORSRBF-R are also better or at least comparable to ParMult-UOBYQA on all test problems and processor settings, except when $P = 1$ on GP, HA3, HA6, GWB6 and when $P = 2$ on HA3. Observe that the instances where the parallel multistart algorithms performed better than our parallel RBF methods correspond to the test problems with the fewest number of local minima. Note also that the two parallel RBF methods are much better than the two parallel multistart methods on the MRO5 and MAC5 test problems, which have a large number of local minima. In addition, the two parallel RBF algorithms are much better than ParMult-ImFil on the groundwater bioremediation application. Overall, ParCGRBF-R and ParCORSRBF-R are generally better than ParMult-SQP/ImFil and ParMult-UOBYQA on the test problems.

From the results, ParCGRBF-R and ParCORSRBF-R have comparable performances on the test problems. ParCORSRBF-R is better than ParCGRBF-R on GWB6 and it is slightly better than ParCGRBF-R on GP, CA, HA3, SH7 and MRO5. On the other hand, ParCGRBF-R is better than ParCORSRBF-R on SC3, SC5 and HA6. On the remaining problems, ParCORSRBF-R is better than ParCGRBF-R for some processor settings while the opposite is true for other processor settings. Finally, ParCORSRBF-R seems to be slightly better than ParCGRBF-R when $P = 12$ on many of the test problems.

Table 2
Average wall clock times (seconds) to get a relative error of <1% for 12 test problems and five alternative parallel global optimization methods using $P = 1, 2, 4, 6, 8, 12$ processors for the global optimization of a costly function that requires τ seconds for one evaluation. The average wall clock times are represented in the form $C(P)\tau + D(P)$, where $C(P)$ and $D(P)$ are given below. For GWB6, the values of $C(P)\tau + D(P)$ represent the average wall clock times required to get a cost of $\leqslant 500$

Test function	P	ParCGRBF-R		ParCORSRBF-R		ParDE		ParMult-SQP/ImFil		ParMult-UOBYQA	
		$C(P)$	$D(P)$	$C(P)$	$D(P)$	$C(P)$	$D(P)$	$C(P)$	$D(P)$	$C(P)$	$D(P)$
GP	1	58.40	91.49	55.90	64.30	581.00	0.10	57.87	0.12	46.47	0.19
	2	36.37	95.69	30.50	73.07	290.50	0.09	43.73	0.06	33.93	0.11
	4	17.47	76.45	17.47	86.90	157.80	0.10	29.93	0.05	22.47	0.11
	6	12.17	81.09	12.70	97.20	105.20	0.10	27.03	0.05	18.63	0.09
	8	11.67	116.51	10.30	108.51	108.00	0.13	26.17	0.05	17.83	0.08
	12	8.77	146.96	8.13	124.06	52.60	0.10	24.33	0.05	16.63	0.07
CA	1	43.40	63.38	35.07	41.70	285.67	0.05	50.47	0.11	94.67	0.36
	2	22.87	52.98	18.53	48.51	142.83	0.05	39.53	0.10	39.03	0.04
	4	12.80	52.53	12.07	65.11	88.80	0.06	24.37	0.08	18.10	0.02
	6	8.87	54.13	8.17	74.11	59.20	0.06	23.13	0.05	15.33	0.01
	8	8.20	76.76	7.00	81.22	53.33	0.07	20.93	0.05	14.43	0.02
	12	7.83	127.02	5.63	91.42	29.60	0.06	19.10	0.07	13.57	0.02
HA3	1	59.97	174.09	56.47	119.81	259.00	0.06	44.20	0.11	30.83	0.05
	2	36.63	196.83	32.83	151.00	130.67	0.06	25.27	0.09	19.80	0.04
	4	19.13	214.07	17.17	164.44	65.33	0.06	21.80	0.09	15.73	0.03
	6	13.63	227.71	10.70	146.58	57.60	0.07	20.10	0.09	14.67	0.04
	8	15.77	477.60	12.43	250.27	32.67	0.05	18.43	0.09	13.50	0.04
	12	11.87	914.66	6.80	215.44	34.67	0.09	17.30	0.05	12.87	0.05
SC3	1	42.33	100.13	66.87	132.25	1338.50	0.39	98.23	0.25	91.67	0.23
	2	24.77	106.86	37.00	156.06	598.13	0.37	48.43	0.14	69.53	0.20
	4	16.93	150.40	18.57	159.38	299.07	0.35	44.27	0.12	41.77	0.13
	6	13.53	182.23	14.67	188.15	236.50	0.43	41.83	0.13	35.87	0.13
	8	10.27	194.84	11.83	217.06	149.53	0.37	40.77	0.14	29.27	0.12
	12	9.10	319.34	8.20	233.11	114.80	0.39	38.83	0.21	25.10	0.13
SH7	1	177.10	862.19	142.80	1261.64	2884.00	0.45	263.80	0.39	300.57	1.31
	2	144.87	1162.36	131.90	2122.08	1442.00	0.43	152.03	0.17	126.67	0.36
	4	66.87	1043.50	74.53	2460.56	721.00	0.43	119.10	0.12	73.30	0.13
	6	53.60	1488.83	51.47	2526.49	667.73	0.59	104.77	0.11	65.40	0.11
	8	46.13	2146.61	39.90	2551.99	500.80	0.58	95.90	0.11	66.87	0.18
	12	40.27	3712.04	26.03	2743.65	333.87	0.57	84.87	0.08	53.03	0.08
SH10	1	161.20	774.81	119.60	1006.73	3216.00	0.47	289.23	0.47	260.47	0.94
	2	82.77	656.69	123.73	2007.30	1608.00	0.50	161.57	0.24	141.03	0.46
	4	45.87	730.95	61.73	2068.41	804.00	0.48	116.40	0.13	72.93	0.14
	6	28.27	768.60	48.13	2388.96	617.20	0.55	110.13	0.12	72.37	0.21
	8	28.30	1274.31	32.30	2105.09	462.90	0.55	97.67	0.11	64.23	0.12
	12	32.10	2805.28	24.00	2543.64	308.60	0.53	88.03	0.11	54.53	0.08
SC4	1	85.20	336.20	86.70	340.20	1077.33	0.33	113.80	0.29	76.97	0.22
	2	52.33	378.25	57.27	479.85	538.67	0.32	87.10	0.23	58.43	0.19
	4	29.77	414.13	29.07	503.11	269.33	0.35	72.17	0.21	40.63	0.14
	6	17.13	373.61	19.07	494.87	215.07	0.38	70.60	0.21	33.73	0.14
	8	21.27	764.48	16.77	618.94	161.30	0.40	69.67	0.22	33.03	0.15
	12	15.07	1051.53	10.67	592.16	107.53	0.39	66.73	0.24	31.03	0.17
MRO5	1	31.97	71.67	30.27	95.38	885.83	0.16	1190.40	84.31	912.80	37.76
	2	31.33	226.76	32.43	315.83	454.57	0.18	373.83	21.29	304.93	8.69
	4	17.40	255.61	17.60	369.10	253.87	0.18	81.00	0.40	56.33	0.24
	6	13.90	366.53	12.27	397.84	179.67	0.18	60.17	0.13	61.37	2.00
	8	14.37	634.72	10.03	471.34	142.27	0.21	54.23	0.11	32.83	0.07
	12	11.13	1063.75	7.33	559.70	108.30	0.22	50.63	0.12	31.33	0.07

(*continued on next page*)

Table 2 (*continued*)

Test function	P	ParCGRBF-R		ParCORSRBF-R		ParDE		ParMult-SQP/ ImFil		ParMult-UOBYQA	
		$C(P)$	$D(P)$	$C(P)$	$D(P)$	$C(P)$	$D(P)$	$C(P)$	$D(P)$	$C(P)$	$D(P)$
MAC5	1	21.00	0.20	21.00	30.77	1593.33	0.25	427.37	1.15	1629.60	99.62
	2	30.80	259.43	30.97	361.19	825.93	0.27	273.67	0.79	976.40	74.65
	4	14.53	221.47	15.23	355.19	429.57	0.28	141.10	0.32	252.77	7.11
	6	11.13	287.73	10.80	386.91	304.50	0.29	129.47	0.33	282.27	20.14
	8	10.03	448.33	9.50	483.06	243.87	0.31	110.70	0.22	129.20	4.71
	12	7.57	705.55	6.23	507.31	179.70	0.34	108.50	0.23	90.73	3.88
SC5	1	60.67	254.97	95.73	489.23	1630.83	0.49	143.07	0.36	126.67	0.43
	2	31.00	231.73	65.53	716.71	852.80	0.52	98.30	0.30	83.20	0.34
	4	18.77	286.38	26.43	594.27	443.80	0.55	78.80	0.25	57.07	0.24
	6	14.20	367.14	20.73	664.10	324.67	0.61	75.07	0.26	48.10	0.20
	8	13.87	576.81	18.20	884.76	244.27	0.59	74.30	0.26	46.60	0.21
	12	11.83	1052.61	10.67	741.35	186.70	0.66	70.63	0.29	44.97	0.24
HA6	1	146.70	1159.07	214.07	1874.07	8846.00	1.29	130.07	0.24	111.57	0.28
	2	79.43	1055.93	101.30	1802.56	4423.00	1.33	108.03	0.13	81.57	0.21
	4	36.30	944.86	41.17	1461.71	1989.60	1.14	80.10	0.10	56.00	0.13
	6	39.07	1881.29	41.37	2438.69	1474.33	1.30	76.77	0.09	53.37	0.12
	8	28.87	2380.46	33.63	2743.97	994.80	1.16	74.40	0.09	52.50	0.13
	12	28.67	6090.47	33.43	5089.45	542.20	1.03	69.93	0.10	50.40	0.13
GWB6	1	183.03	1846.38	104.57	1165.52	1156.00	0.89	811.80	5.38	74.33	0.17
	2	79.57	1309.00	58.20	1402.09	578.00	0.88	480.80	3.37	62.60	0.96
	4	36.30	1022.77	31.13	1606.03	327.73	1.01	238.73	3.28	45.53	2.56
	6	24.27	1144.82	27.90	2138.23	192.67	0.85	175.37	4.61	41.77	4.24
	8	21.47	1727.71	21.67	2412.05	163.87	1.04	154.60	6.14	38.77	5.87
	12	18.10	2926.17	15.70	2858.78	119.90	1.10	105.03	9.08	35.60	9.35

8.2. Comparison of relative speedups

The relative speedups for the different parallel algorithms are summarized in Table 3. Before we proceed, we note that a better relative speedup does not necessarily mean better wall clock time. Relative speedup measures how well a parallel algorithm scales relative to its serial version with the addition of more processors. For two different parallel algorithms, the times required to solve a problem using 1 processor are usually different. Hence, we cannot determine which parallel algorithm has a better parallel wall clock time by simply comparing the values of their relative speedups.

We also note that parallel RBF methods could exhibit some *speedup anomalies* that are similar to those encountered in branch-and-bound methods (Lai and Sahni, 1984; Barr and Hickman, 1993). An example of a speedup anomaly is when the speedup for P_1 processors is greater than that for P_2 processors even when $P_1 < P_2$. Another example is *superlinear speedup*, which occurs when Relative Speedup(P) > P. Such anomalies can occur in branch-and-bound methods because the set of nodes encountered during the search is usually different for different processor settings. Similarly, in the case of parallel RBF methods, the set of points encountered during the search for the global minimum could be different for different processor settings even if the same experimental design is used for initialization.

Another possible cause of speedup anomalies is the stochastic nature of the results, i.e. the results depend on the random experimental design used. When $P_1 < P_2$, it is reasonable to expect that the average wall clock time of a parallel RBF method should be better when using P_2 processors than when using P_1 processors. However, insufficient number of trials, due to limited computational resources, could also lead to anomalous results where the average wall clock time (and hence the relative speedup) for P_2 processors is worse than that for P_1 processors even if $P_1 < P_2$.

Now, from Table 3, observe that the relative speedups for ParDE are generally very good and better than the relative speedups for the other parallel algorithms on most of the test problems. In some cases (see the

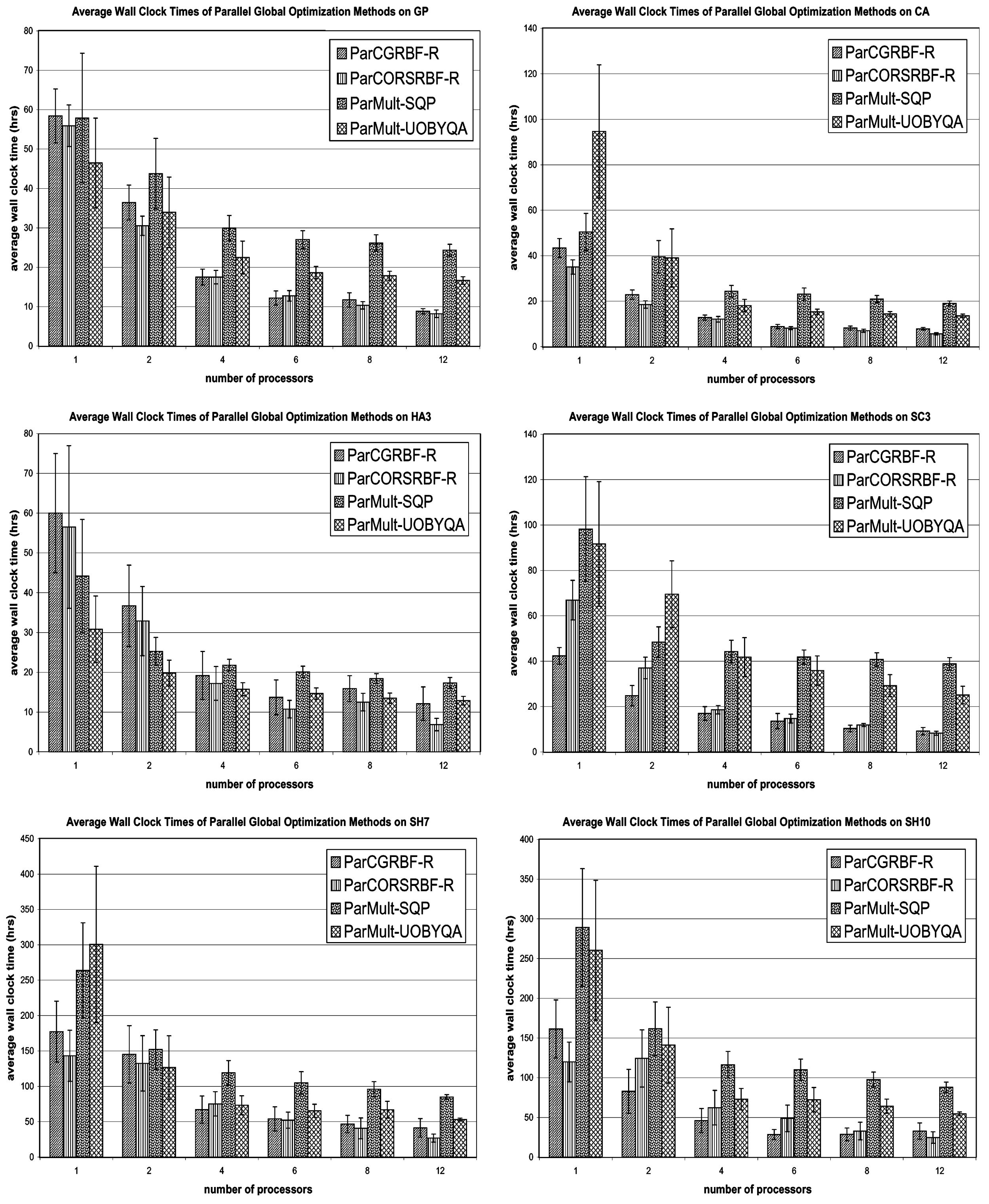
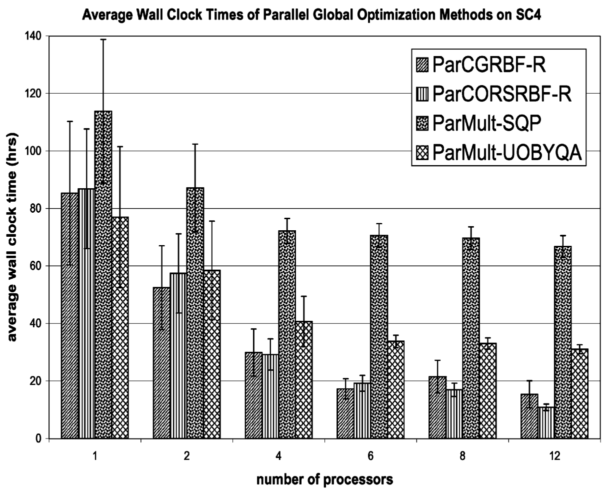
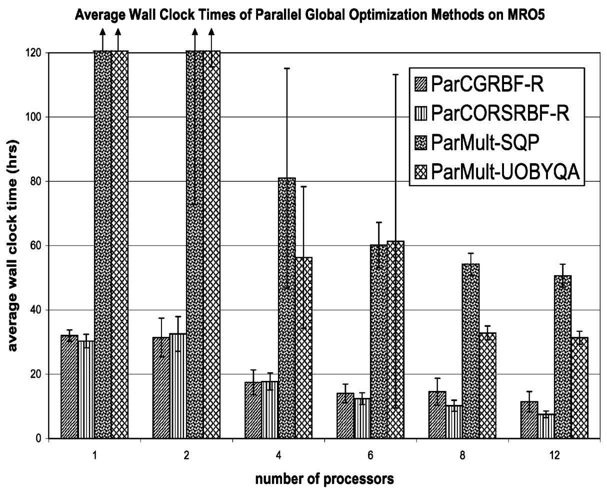
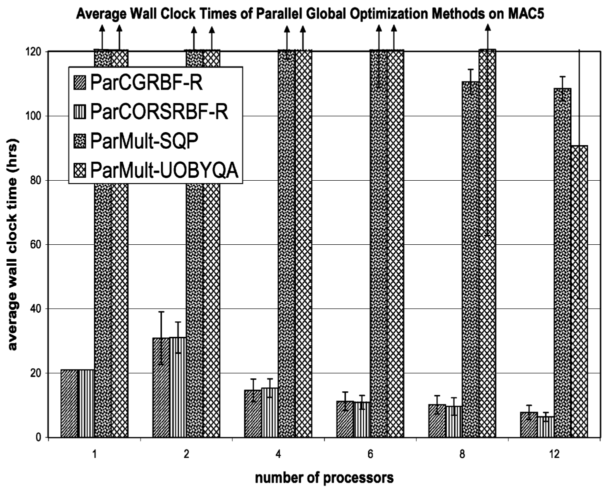
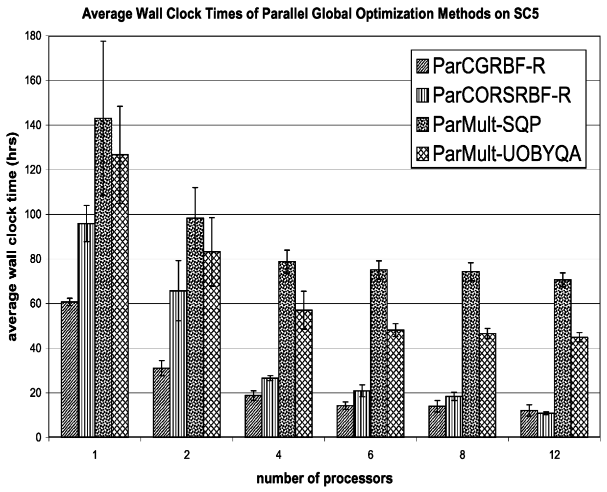


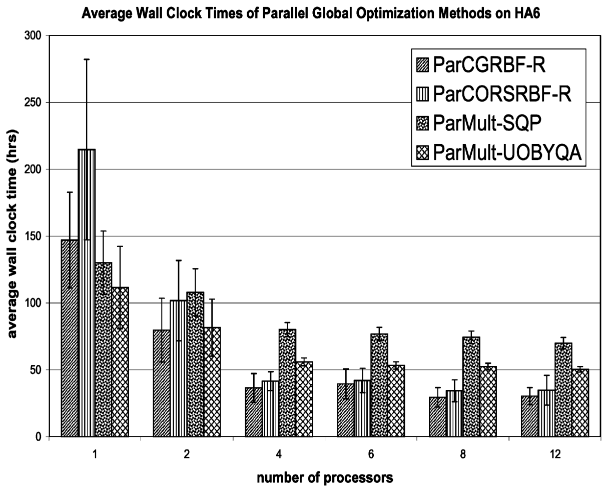
Fig. 1. Average wall clock times of parallel global optimization methods assuming $\tau = 1$ hour. The error bars represent 95% confidence intervals for the means.











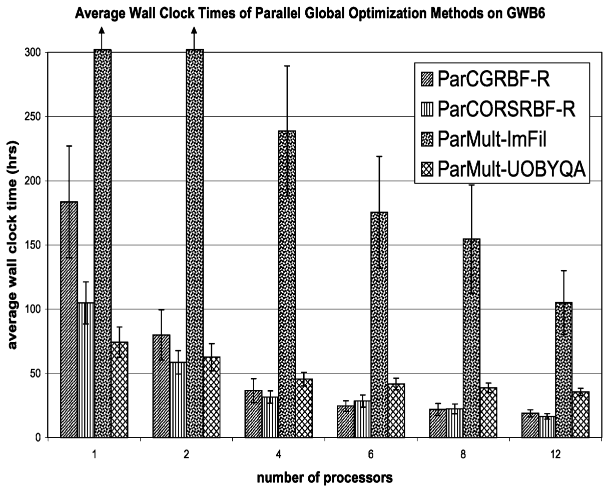


Fig. 1 (*continued*)

Table 3
Relative speedups of parallel global optimization methods on the test problems

Test prob	P	ParCG-RBF-R	ParCORS-RBF-R	ParDE	ParMult-SQP	ParMult-Uobyqa	Test prob	P	ParCG-RBF-R	ParCORS-RBF-R	ParDE	ParMult-SQP/ImFil	ParMult-Uobyqa
GP	2	1.61	1.83	2.00	1.32	1.37	SC4	2	1.63	1.51	2.00	1.31	1.32
	4	3.34	3.20	3.68	1.93	2.07		4	2.85	2.97	4.00	1.58	1.89
	6	4.79	4.39	5.52	2.14	2.49		6	4.95	4.52	5.01	1.61	2.28
	8	4.99	5.41	5.38	2.21	2.61		8	3.97	5.12	6.68	1.63	2.33
	12	6.63	6.85	11.05	2.38	2.79		12	5.55	8.01	10.02	1.71	2.48
CA	2	1.90	1.89	2.00	1.28	2.43	MRO5	2	1.02	0.93	1.95	3.18	2.99
	4	3.39	2.90	3.22	2.07	5.23		4	1.83	1.71	3.49	14.70	16.20
	6	4.89	4.28	4.83	2.18	6.17		6	2.28	2.45	4.93	19.79	14.87
	8	5.28	5.00	5.36	2.41	6.56		8	2.20	2.98	6.23	21.95	27.80
	12	5.52	6.20	9.65	2.64	6.98		12	2.80	4.05	8.18	23.51	29.13
HA3	2	1.64	1.72	1.98	1.75	1.56	MAC5	2	0.68	0.68	1.93	1.56	1.67
	4	3.13	3.28	3.96	2.03	1.96		4	1.44	1.37	3.71	3.03	6.45
	6	4.38	5.26	4.50	2.20	2.10		6	1.87	1.93	5.23	3.30	5.77
	8	3.77	4.52	7.93	2.40	2.28		8	2.07	2.18	6.53	3.86	12.61
	12	4.95	8.24	7.47	2.55	2.40		12	2.71	3.30	8.87	3.94	17.96
SC3	2	1.71	1.81	2.24	2.03	1.32	SC5	2	1.96	1.46	1.91	1.46	1.52
	4	2.50	3.59	4.48	2.22	2.19		4	3.22	3.60	3.67	1.82	2.22
	6	3.12	4.55	5.66	2.35	2.56		6	4.25	4.58	5.02	1.91	2.63
	8	4.10	5.63	8.95	2.41	3.13		8	4.33	5.20	6.68	1.93	2.72
	12	4.61	8.10	11.66	2.53	3.65		12	5.01	8.82	8.74	2.03	2.82
SH7	2	1.22	1.08	2.00	1.74	2.37	HA6	2	1.84	2.11	2.00	1.20	1.37
	4	2.64	1.90	4.00	2.21	4.10		4	4.02	5.16	4.45	1.62	1.99
	6	3.28	2.74	4.32	2.52	4.60		6	3.71	5.10	6.00	1.69	2.09
	8	3.80	3.53	5.76	2.75	4.50		8	4.98	6.24	8.89	1.75	2.13
	12	4.29	5.34	8.64	3.11	5.67		12	4.84	6.16	16.32	1.86	2.21
SH10	2	1.95	0.96	2.00	1.79	1.85	GWB6	2	2.30	1.79	2.00	1.69	1.19
	4	3.50	1.92	4.00	2.48	3.57		4	5.02	3.32	3.53	3.40	1.63
	6	5.67	2.46	5.21	2.63	3.60		6	7.47	3.68	6.00	4.63	1.78
	8	5.63	3.65	6.95	2.96	4.06		8	8.36	4.70	7.05	5.25	1.92
	12	4.91	4.85	10.42	3.29	4.78		12	9.70	6.36	9.64	7.73	2.09

results for SC3 and HA6), superlinear speedup occurs. These results are somewhat expected because of the inherently parallel nature of DE.

The relative speedups for ParMult-UOBYQA are somewhat better than those for ParMult-SQP except on HA3 and GWB6. However, the relative speedups for these parallel multistart algorithms are generally poor compared to the relative speedups for the other parallel algorithms on the test problems with relatively few local minima. This is because the addition of more processors will generally not make the search for the global minimum faster on such problems.

As noted above, speedup anomalies can occur when using parallel RBF methods. For instance, from Table 3, superlinear speedup occurs for ParCGRBF-R on HA6 when $P = 4$, and on GWB6 when $P = 2,4,6,8$. Superlinear speedup also occurs for ParCORSRBF-R on HA6 when $P = 2,4$. A speedup anomaly can also be seen in Table 3 where the relative speedups of ParCGRBF-R and ParCORSRBF-R on MAC5 when using 2 processors are both <1, implying that their wall clock times for 2 processors are worse than their serial wall clock times. However, this anomaly is expected for MAC5 because the center of the hypercube domain, which happens to be the global minimizer of MAC5, is always contained in all of the 21-point SLHDs used in the serial version. (Note that an SLHD with an odd number of points always includes the center.) On the other hand, all of the 22-point SLHDs for $P = 2$ do not contain the global minimum point. Finally, another example is that the speedup of ParCGRBF-R on HA3 drops as the number of processors increases from 6 to 8.

From Table 3, note that the two parallel RBF algorithms exhibit generally good relative speedups when using up to six processors on most test problems. Moreover, the relative speedups of ParCORSRBF-R are generally better or comparable to the relative speedups for ParCGRBF-R except on SH7, SH10 and GWB6. However, we can also see that the performance of the two parallel RBF methods do not scale well with the addition of more processors. For instance, in most of the cases considered, the relative efficiency of a parallel RBF method drops to below 60% when using 12 processors. A possible reason for this could be that with more processors we are selecting the evaluation points based on less information. In the serial case, we choose the evaluation point based on the information provided by all the previously evaluated points. On the other hand, when using 12 processors, we choose an evaluation point in an iteration without knowledge of the function values of the other 11 evaluation points in that iteration.

8.3. Comparison with an alternative parallel RBF method

Finally, we compare ParCGRBF-R and ParCORSRBF-R with ParDFRBF, which is the derivative-free variant of ParGERBF (Sóbester et al., 2004). The code used by Sóbester et al. (2004) is not publicly available so we perform an indirect comparison by running our parallel algorithms on the test problems they used, namely, MRO5 and MAC5. The results for ParDFRBF are originally expressed in terms of the number of function evaluations needed to reach convergence. To facilitate the comparison, we converted these results in terms of number of parallel iterations for convergence and these can be compared directly to the values of $C(P)$ in Table 2 for MRO5 and MAC5. Sóbester et al. (2004) did not specify their convergence criterion but their method settled on a value that is close to the global minimum for each of their test problems. The results for ParDFRBF in terms of the number of parallel iterations for convergence on MRO5 when using $P = 1,4,8$ processors are 47, 12 and 8, respectively. The corresponding results for ParDFRBF on MAC5 when using $P = 1,4,8$ processors are 66, 16 and 10, respectively.

Before we proceed, we note that the results of this comparison will be somewhat preliminary because Sóbester et al. (2004) performed only one trial of their parallel RBF methods for each processor setting ($P = 1,4,8$) on each of their two synthetic test problems. Although they are using optimal Latin hypercube designs (LHDs), multiple trials are still important since optimal LHDs are not unique. In fact, a simple rotation and/or reflection of an optimal LHD yields a possibly different optimal LHD of the same size. Regis and Shoemaker (2006) observed highly significant performance variability in two RBF methods when using different SLHDs (which on average have better space-filling properties than ordinary LHDs) on many test problems. In fact, the results for our parallel RBF methods on the 12 test problems show significant variability. Hence, it is reasonable to expect that such variability also exists when using optimal LHDs.

Now comparing the results by Sóbester et al. (2004) with our results in Table 2 for the MRO5 and MAC5 test functions, we can see that our parallel RBF algorithms are competitive with ParDFRBF. In the serial case

($P = 1$), our RBF methods are much better on average than ParDFRBF. When $P = 4$ and 8, our parallel RBF methods are comparable to ParDFRBF for the MAC5 function while ParDFRBF seems to be better than our two parallel RBF methods on the MRO5 function. However, as mentioned above, it is hard to draw accurate conclusions from these comparisons but our results have the advantage that they are much more informative since they also provide some measure of performance variability.

9. Conclusions

We introduced a synchronous master–worker framework for parallel global optimization of expensive functions using response surface models. We parallelized CG-RBF-Restart and CORS-RBF-Restart, which are the improved variants of the RBF methods by Gutmann (2001a) and by Regis and Shoemaker (2005), respectively. We compared our parallel RBF algorithms with three alternatives: (1) Parallel Differential Evolution (DE), which is an evolutionary algorithm; (2) Parallel Multistart Sequential Quadratic Programming (SQP) (for the smooth problems) and Implicit Filtering (ImFil) (for the nonsmooth groundwater application); and (3) Parallel Multistart UOBYQA, which is a derivative-free trust-region algorithm. We also performed a preliminary comparison with the derivative-free variant of the parallel RBF method by Sóbester et al. (2004).

The experimental results indicate that ParCGRBF-R and ParCORSRBF-R are generally better than the three alternatives on most of the test problems considered. Moreover, the results indicate that ParCGRBF-R and ParCORSRBF-R have comparable performances on the test problems. However, for the 6-dimensional groundwater bioremediation application, ParCORSRBF-R is better than ParCGRBF-R. In addition, the results suggest good performance for these two parallel RBF methods when using up to 6 processors. Finally, preliminary comparisons indicate that ParCGRBF-R and ParCORSRBF-R are competitive with the derivative-free variant of the parallel RBF method by Sóbester et al. (2004).

Acknowledgements

This research was supported primarily by the CISE Directorate in NSF (grant ACI-0305583). Early work on the project was supported by the Intelligent Information Systems Institute (IISI) (AFOSR, grant F49620-01-1-0076), directed by Dr. Carla Gomes, by providing GRA funding for Rommel Regis. This research was conducted using the resources of the Cornell Theory Center, which receives funding from Cornell University, New York State, federal agencies, foundations, and corporate partners. Dr. Jae-Heung Yoon and Dr. Raju Rohde provided the Fortran simulation code for the groundwater bioremediation problem. Programming advice was given by Dr. John Zollweg of the Cornell Theory Center. Finally, we would like to thank the anonymous referees for their comments and suggestions.

References

Barr, R.S., Hickman, B.L., 1993. Reporting computational experiments with parallel algorithms: Issues, measures, and experts' opinions. ORSA Journal on Computing 5 (1), 2–18.

Björkman, M., Holmström, K., 2000. Global optimization of costly nonconvex functions using radial basis functions. Optimization and Engineering 1 (4), 373–397.

Booker, A.J., Dennis, J.E., Frank, P.D., Serafini, D.B., Torczon, V., Trosset, M.W., 1999. A rigorous framework for optimization of expensive functions by surrogates. Structural Optimization 17 (1), 1–13.

Box, G.E.P., Draper, N.R., 1987. Empirical Model-Building and Response Surfaces. John Wiley & Sons, Inc., New York.

Branin, F.H., 1972. Widely convergent methods for finding multiple solutions of simultaneous nonlinear equations. IBM Journal of Research Developments 16, 504–522.

Brekelmans, R., Driessen, L., Hamers, H., den Hertog, D., 2005. Constrained optimization involving expensive function evaluations: A sequential approach. European Journal of Operational Research 160, 121–138.

Buhmann, M.D., 2003. Radial Basis Functions. Cambridge University Press, U.K.

Byrd, R.H., Dert, C.L., Rinnooy Kan, A.H.G., Schnabel, R.B., 1990. Concurrent stochastic methods for global optimization. Mathematical Programming 46, 1–29.

Cantu-Paz, E., 2000. Efficient and Accurate Parallel Genetic Algorithms. Kluwer, Boston.

Conn, A.R., Scheinberg, K., Toint, Ph.L., 1997. Recent progress in unconstrained nonlinear optimization without derivatives. Mathematical Programming 79 (3), 397–414.

Cressie, N., 1993. Statistics for Spatial Data. John Wiley, New York.

Dixon, L.C.W., Jha, M., 1993. Parallel algorithms for global optimization. Journal of Optimization Theory and Applications 79 (2), 385–395.

Dixon, L.C.W., Szegö, G., 1978. The global optimization problem: An introduction. In: Dixon, L.C.W., Szegö, G. (Eds.), Towards Global Optimization 2. North-Holland, Amsterdam, pp. 1–15.

El-Beltagy, M.A., Nair, P.B., Keane, A.J., 1999. Metamodelling techniques for evolutionary optimization of computationally expensive problems: Promises and limitations. In: Banshaf, W., Daida, J., Eiben, A.E., Garzon, M.H., Honavar, V., Jakiela, M., Smith, R.E. (Eds.), Proceedings of the Genetic and Evolutionary Computation Conference (GECCO99). Morgan Kaufman, pp. 196–203.

Friedman, J., 1991. Multivariate adaptive regression splines. Annals of Statistics 19 (1), 1–141.

Gilmore, P., Kelley, C.T., 1995. An implicit filtering algorithm for optimization of functions with many local minima. SIAM Journal on Optimization 5, 269–285.

Giunta, A.A., Balabanov, V., Haim, D., Grossman, B., Mason, W.H., Watson, L.T., Haftka, R.T., 1997. Aircraft multidisciplinary design optimisation using design of experiments theory and response surface modelling. Aeronautical Journal 101 (1008), 347–356.

Gutmann, H.M., 2001a. A radial basis function method for global optimization. Journal of Global Optimization 19 (3), 201–227.

Gutmann, H.M., 2001b. Radial basis function methods for global optimization. Ph.D. Thesis, Department of Applied Mathematics and Theoretical Physics, University of Cambridge, UK.

Horst, R., Pardalos, P.M., Thoai, N.V., 2000. Introduction to Global Optimization, second ed. Kluwer, Dordrecht.

Jones, D.R., 1996. Global optimization with response surfaces. Presented at the Fifth SIAM Conference on Optimization, Victoria, Canada.

Jones, D.R., Schonlau, M., Welch, W.J., 1998. Efficient global optimization of expensive black-box functions. Journal of Global Optimization 13 (4), 455–492.

Kaufman, M., Balabanov, V., Burgee, S.L., Giunta, A.A., Grossman, B., Haftka, R.T., Mason, W.H., Watson, L.T., 1996. Variable-complexity response surface approximations for wing structural weight in HSCT design. Computational Mechanics 18 (2), 112–126.

Kelley, C.T., 1999. Iterative Methods for Optimization. SIAM, Philadelphia.

Koehler, J.R., Owen, A.B., 1996. Computer experiments. In: Ghosh, S., Rao, C.R. (Eds.), Handbook of Statistics, vol. 13: Design and Analysis of Computer Experiments, North-Holland, Amsterdam, pp. 261–308.

Lai, T.H., Sahni, S., 1984. Anomalies in parallel branch-and-bound algorithms. Communications of the Association for Computing Machinery 27, 594–602.

Locatelli, M., 1997. Bayesian algorithms for one-dimensional global optimization. Journal of Global Optimization 10, 57–76.

Marazzi, M., Nocedal, J., 2002. Wedge trust region methods for derivative free optimization. Mathematical Programming, Series A 91, 289–305.

Minsker, B.S., Shoemaker, C.A., 1998. Dynamic optimal control of in-situ bioremediation of groundwater. Journal of Water Resources Planning and Management 124 (3), 149–161.

Myers, R.H., Montgomery, D.C., 1995. Response Surface Methodology: Process and Product Optimization Using Designed Experiments. John Wiley & Sons Inc., New York.

Nocedal, J., Wright, S.J., 1999. Numerical Optimization. Springer, New York.

Powell, M.J.D., 1992. The theory of radial basis function approximation in 1990. In: Light, W. (Ed.), Advances in Numerical Analysis, vol. 2: Wavelets, Subdivision Algorithms and Radial Basis Functions, Oxford University Press, pp. 105–210.

Powell, M.J.D., 1999. Recent research at Cambridge on radial basis functions. In: Müller, M., Buhmann, M., Mache, D., Felten, M. (Eds.), New Developments in Approximation Theory, International Series of Numerical Mathematics, vol. 132. Birkhauser Verlag, Basel, pp. 215–232.

Powell, M.J.D., 2002. UOBYQA: Unconstrained optimization by quadratic approximation. Mathematical Programming 92, 555–582.

Powell, M.J.D., 2003. On trust region methods for unconstrained minimization without derivatives. Mathematical Programming 97, 605–623.

Regis, R.G., Shoemaker, C.A., 2005. Constrained global optimization of expensive black box functions using radial basis functions. Journal of Global Optimization 31, 153–171.

Regis, R.G., Shoemaker, C.A., 2006. Improved strategies for radial basis function methods for global optimization. Journal of Global Optimization. doi:10.1007/s10898-006-9040-1.

Sacks, J., Welch, W.J., Mitchell, T.J., Wynn, H.P., 1989. Design and analysis of computer experiments. Statistical Science 4 (4), 409–435.

Schnabel, R.B., 1995. A view of the limitations, opportunities, and challenges in parallel nonlinear optimization. Parallel Computing 21, 875–905.

Schoen, F., 1993. A wide class of test functions for global optimization. Journal of Global Optimization 3, 133–137.

Shoemaker, C.A., Willis, M., Zhang, W., Gossett, J., 2001. Model analysis of reductive dechlorination with data from Cape Canaveral field site. In: Magar, V., Vogel, T., Aelion, C., Leeson, A. (Eds.), Innovative Methods in Support of Bioremediation. Battelle Press, Columbus, OH, pp. 125–131.

Sóbester, A., Leary, S., Keane, A.J., 2004. A parallel updating scheme for approximating and optimizing high fidelity computer simulations. Structural and Multidisciplinary Optimization 27, 371–383.

Storn, R., Price, K., 1997. Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. Journal of Global Optimization 11, 341–359.

Taylor, S., 1993. Modeling enhanced in-situ biodegradation in groundwater: model response to biological parameter uncertaintyProceedings of the 1993 Groundwater Modeling Conference, International Groundwater Modeling Center, 51. Golden, Colorado, pp. (4-51)–(4-60).

The Mathworks, Inc., 2004. Optimization Toolbox for Use with MATLAB: User's Guide, Version 3, Natick, MA.

Torczon, V., 1997. On the convergence of pattern search algorithms. SIAM Journal on Optimization 7 (1), 1–25.

Törn, A., Žilinskas, A., 1989. Global Optimization, Lecture Notes in Computer Science, vol. 350. Springer Verlag, Berlin.

Vanden Berghen, F., Bersini, H., 2005. CONDOR, a new parallel, constrained extension of Powell's UOBYQA algorithm: Experimental results and comparison with the DFO algorithm. Journal of Computational and Applied Mathematics 181, 157–175.

Ye, K.Q., Li, W., Sudjianto, A., 2000. Algorithmic construction of optimal symmetric latin hypercube designs. Journal of Statistical Planning and Inference 90, 145–159.

Yoon, J.-H., 1997. Optimal design of groundwater in situ bioremediation using evolutionary algorithms. Ph.D. Thesis, School of Civil and Environmental Engineering, Cornell University, Ithaca, NY.

Yoon, J.-H., Shoemaker, C.A., 1999. Comparison of optimization methods for ground-water bioremediation. Journal of Water Resources Planning and Management 125, 54–63.