Performance Modeling and Analysis of a Massively Parallel DIRECT—Part 1

Jian He[1]
Alex Verstak[1]
L. T. Watson[2]
M. Sosonkina[3]

Abstract

Modeling and analysis techniques are used to investigate the performance of a massively parallel version of DIRECT, a global search algorithm widely used in multidisciplinary design optimization applications. Several high-dimensional benchmark functions and real world problems are used to test the design effectiveness under various problem structures. Theoretical and experimental results are compared for two parallel clusters with different system scale and network connectivity. The present work aims at studying the performance sensitivity to important parameters for problem configurations, parallel schemes, and system settings. The performance metrics include the memory usage, load balancing, parallel efficiency, and scalability. An analytical bounding model is constructed to measure the load balancing performance under different schemes. Additionally, linear regression models are used to characterize two major overhead sources—interprocessor communication and processor idleness, and also applied to the isoefficiency functions in scalability analysis. For a variety of high-dimensional problems and large scale systems, the massively parallel design has achieved reasonable performance. The results of the performance study provide guidance for efficient problem and scheme configuration. More importantly, the generalized design considerations and analysis techniques are beneficial for transforming many global search algorithms to become effective large scale parallel optimization tools.

Keywords: DIRECT, global search algorithms, load balancing, parallel optimization, performance modeling, scalability analysis

1 INTRODUCTION

Global search algorithms have been increasingly applied to large scale optimization problems in many fields. Compared to local methods, these global approaches are more likely to discover the global optimum instead of being trapped at local minimum points for complex nonconvex or nonlinear problems with irregular design domain. The DIRECT algorithm (Jones et al. 1993) is one such global optimization algorithm that has been applied to large scale engineering design problems such as aircraft design (Baker et al. 2000), pipeline design (Carter et al. 2001), routing (Bartholomew-Biggs et al. 2003), surface optimization (Zhu et al. 2002), transmitter placement (He et al. 2004b), molecular genetic mapping (Ljungberg et al. 2004), and cell cycle modeling (Zwolak et al. 2005 and Panning et al. 2006). The complexity of these applications ranges from low dimensional with 3–20 variables to high dimensional with up to 143 variables.

Compared to local methods, global optimization methods generally have higher computational cost and memory requirement, which often lead to solutions that involve data-distributed parallel computing techniques. Since the birth of the first Beowulf 16-node cluster (Becker 1995), numerous commodity-based cluster systems have been developed in academia and industry to provide cost-effective alternatives to expensive mainframe supercomputers. With the rapid expansion of problem scale and system size, adaptive and dynamic algorithms are being actively proposed to optimally harness the abundant computing resources. There are a few such examples in the field of global optimization algorithms, including parallel versions of direct search (Dennis et al. 1991), branch-and-bound (Clausen 1997), Tabu search (Talbi et al. 1998), and genetic algorithms (McMahon et al. 2000). These sophisticated parallel schemes take into account both algorithm characteristics and system attributes, thus producing reasonable parallel efficiency and scalability.

In the past decade, the parallel schemes of DIRECT evolved from a classical master-slave paradigm (Gablonsky 2001a), to a fully distributed version with dynamic load balancing (Watson et al. 2001), and recently to a multilevel scheme combining global addressing and message passing models (pDIRECT_I,

[1]Departments of Computer Science and [2]Mathematics, Virginia Polytechnic Institute and State University, Blacksburg, VA 24061. (jihe@vt.edu)

[3]Ames Laboratory, Iowa State University, 236 Wilhelm Hall, Ames, IA 50011.

He et al. 2006). To further improve program portability, execution robustness, and parallel performance, a massively parallel version (pDIRECT_II, He et al. 2006) has been developed with several dynamic features, which have been evaluated on System X, a 2200-processor Apple G5 cluster. He et al. (2006) analyzed its performance in terms of data structure efficiency and load balancing. The present work characterizes its performance sensitivity to problem dimension, task granularity, domain partition, and computing environment. A 400-processor 64-bit Opteron Linux cluster (Anantham) is used as the second computing environment differing from System X in many aspects, some of which, such as network connectivity, are particularly interesting for the present study. The goal is to (1) ensure the design effectiveness of pDIRECT_II on a variety of problems and systems, (2) guide the proper choice of optimization parameter inputs specified by users, and (3) describe the design considerations and analysis techniques that can be generalized to apply to parallelizing other global optimization algorithms challenged by large scale applications on massively parallel systems.

The paper is organized as follows. Section 2 gives an overview of DIRECT, discusses the design issues for its parallel version, and describes the test problems used for the present work. In Section 3, the massively parallel design is briefly discussed with an overall scheme and high-level implementation details. Performance modeling and analysis on selected problem parameters, parallel schemes, and system characteristics are presented in detail in Section 4. Section 5 assesses whether the above stated goals were achieved.

2 OVERVIEW OF DIRECT

DIRECT is a deterministic global search algorithm for solving optimization problems subject to certain assumptions. The global convergence is contingent on the properties of the objective function and the nature of the constraints (Finkel et al. 2004b). When the objective function is Lipschitz continuous around the global optimum point, the global convergence is guaranteed. The general optimization problem considered here is to find the point $\bar{x} \in D$ that minimizes the given objective function $f(x)$ defined in the N-dimensional domain $D = \{x \in E^N \mid \ell \leq x \leq u\}$, where ℓ and u are lower and upper bounds on x. The computed solution may or may not approximate a global minimum point depending on the specified stopping condition—a budget of computational cost (i.e., the maximum number of iterations or evaluations) or a semi-global optimization goal (i.e., the relative function value improvement between iterations or the minimum diameter of the subregion centered at $\bar{x}$).

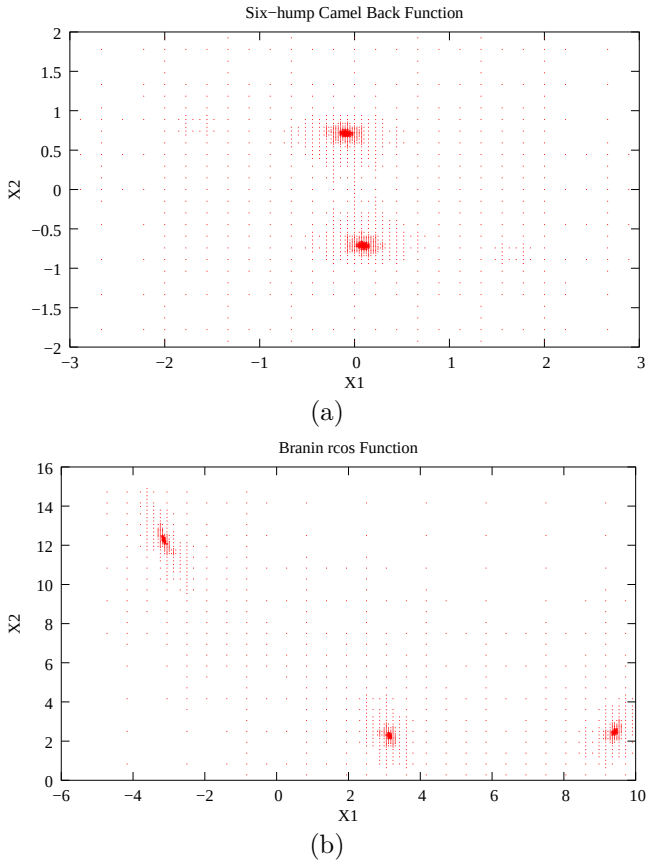


Fig. 2.1 The sample points of DIRECT for (a) Six-hump Camel Back function (two global solutions) and (b) Branin rcos function (three global solutions).

Before the DIRECT search satisfies the stopping condition, it iterates through a few steps to select "potentially optimal" subregions (SELECTION), to sample candidate points in these subregions (SAMPLING), and to subdivide D accordingly (DIVISION). "Potentially optimal" is the key concept defined in the original paper (Jones et al. 1993). If the objective function value within a subregion is potentially smaller than that in any other subregions for some Lipschitz constant, the subregion is deemed potentially optimal. This unique selection strategy explores the design space intelligently toward multiple promising subregions, thus avoiding being trapped by local minimum points. Figure 2.1 shows the sample points by DIRECT for the Six-hump Camel Back (SB) and Branin rcos (BR) two-dimensional functions

Table 2.1 Test functions selected from GEATbx (Pohlheim 1996).

Name	Description
GR	Griewank $f = 1 + \sum_{i=1}^{N} {x_i}^2/500 - \prod_{i=1}^{N} \cos(x_i/\sqrt{i}))$, $-20.0 \le x_i \le 30.0$, $f(0,\ldots,0) = 0.0$
QU	Quartic $f = \sum_{i=1}^{N} 2.2 \times (x_i + 0.3)^2 - (x_i - 0.3)^4$, $-2.0 \le x_i \le 3.0$, $f(3,\ldots,3) = -29.816N$
RO	Rosenbrock's Valley $f = \sum_{i=1}^{N} 100(x_{i+1} - x_i^2)^2 + (1 - x_i)^2$, $-2.048 \le x_i \le 2.048$, $f(1,\ldots,1) = 0$
SC	Schwefel $f = -\sum_{i=1}^{N} x_i \sin(\sqrt{\lvert x_i \rvert})$, $-500 \le x_i \le 500$, $f\big(420.9(1,\ldots,1)\big) \approx -418.9N$
MI	Michalewicz $f = -\sum_{i=1}^{N} \sin(x_i) \times \sin(\frac{ix_i^2}{\pi})^{20}$, $0 \le x_i \le \pi$, $f(\bar{x}) = 0$ for $\bar{x} \in \{0,\pi\}^N$
SB	Six-hump Camel Back $f = (4 - 2.1 * x_1^2 + x_1^4/3)x_1^2 + x_1x_2 +$ $(-4 + 4 * x_2^2)x_2^2$, $-3 \le x_1 \le 3$, $-2 \le x_2 \le 2$, $f(\bar{x}) = -1.0316$ at $\bar{x} = (-0.0898, 0.7126)$ and $(0.0898, -0.7126)$
BR	Branin rcos $f = (x_2 - \frac{5.1}{4-\pi^2}x_1^2 + \frac{5}{\pi}x1 - 6)^2$ $+10(1 - \frac{1}{8-\pi})\cos(x_1) + 10$, $-5 \le x_1 \le 10$, $0 \le x_2 \le 15$, $f(\bar{x}) = 0.397887$ at $\bar{x} = (-\pi, 12.275)$, $(\pi, 2.275)$, and $(9.42478, 2.475)$

(see Table 2.1 for a list of artificial test functions used in the present work).

Ljungberg et al. (2004) report that DIRECT performs faster and more accurate than exhaustive grid search and a genetic algorithm. Zhu et al. (2002) also found that DIRECT converges faster to a global optimum point than adaptive simulated annealing. The secret may lie in the parameter ϵ defined in DIRECT to balance the global and local search efforts. Nevertheless, DIRECT still converges slowly compared to local or gradient-based algorithms, because a significant amount of time is consumed in exploring the entire design space, which is a common tradeoff between the search broadness and the convergence rate. Several modifications to DIRECT (Nelson et al. 1998, Gablonsky et al. 2001b, Cox et al. 2002, Finkel et al. 2004a, Sergeyev et al. 2006) have been proposed to speed up the convergence. Additionally, combining DIRECT with other methods has been found to be fairly successful to address this issue (Nelson et al. 1998, Zwolak et al. 2005, Panning et al. 2006). The emphasis of the present work is on the performance analysis of the parallel design, so the original DIRECT with $\epsilon = 0$ and additional stopping rules is considered.

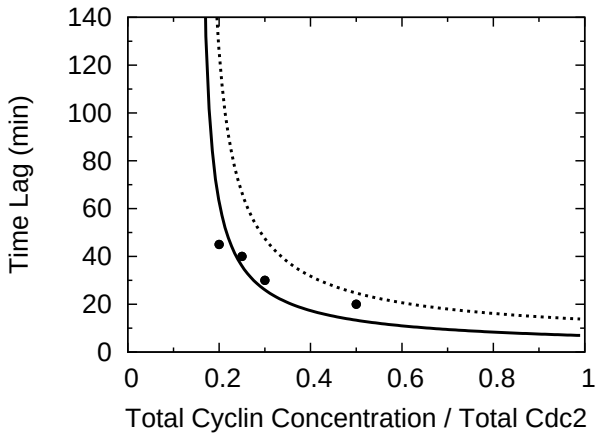


Fig. 2.2 An example of matching model predictions (curves) and experimental data (dots) for the frog egg extract.

In addition to the artificial test functions, the cell cycle parameter estimation problems for frog egg extracts (FE) by Zwolak et al. (2005) and budding yeast (BY) by Panning et al. (2006) are used here as real world applications. The cell cycle modeling aims at decoding the physiological behavior of a living cell. The models used here are systems of ordinary differential equations (ODEs) with unknown parameters. The parameters are determined by orthogonal distance regression with the nonlinear ODE model and experimental data. Figure 2.2 shows an example of matching model predictions (curves) with the known experimental data (dots) for the frog egg extract (He et al. 2004a).

The complexity of solving a parameter estimation problem depends on the scale of the ODE model, the size of the parameter space, and the process of the error function evaluation for experiments. The FE objective function costs about 3 seconds per function evaluation with three ODEs containing 16 parameters and the BY objective function takes approximately 11 seconds per evaluation with 36 ODEs containing 143 parameters. A sequential DIRECT would require a few days and even a few weeks to fully explore the parameter space. On a single machine, DIRECT will eventually fail due to limited memory capacity for the fast-growing

intermediate data. These computationally intensive and memory demanding applications motivated a massively parallel design to cope with the memory and speed demand, and efficiently utilize modern large scale parallel systems.

3 DESIGN AND IMPLEMENTATION

The main high level steps of DIRECT for each iteration are:

1. SELECTION identifies a set of "potentially optimal" boxes that represent subregions inside a normalized design domain.
2. SAMPLING evaluates new points sampled around the centers of all "potentially optimal" boxes along their longest dimensions.
3. DIVISION subdivides "potentially optimal" boxes according to the function values at the newly sampled points.

Note that the original DIRECT starts with a single domain, so there is only one center point for SAMPLING and DIVISION at the first iteration. Unavoidably, a load imbalance occurs at the early stage. To mitigate this problem, an optional step of domain decomposition can be specified by the user to create multiple staring points, one per subdomain. The performance analysis and results in Section 4 show that the multiple subdomain approach not only improves the load balancing, but also converges faster for the real world applications FE and BY.

From the second iteration, SELECTION outputs multiple "potentially optimal" boxes to SAMPLING, which in turn passes the function values of new sample points to DIVISION. The inherent concurrency gives rise to a natural task parallelism. Unfortunately, the same seemingly advantageous step also comes with a disadvantage—data dependency, since any step during an iteration needs to wait for the results of all the previous steps. This inherently sequential nature favors a parallel scheme that decouples SELECTION and SAMPLING into two different roles—master and worker, respectively.

On a master processor, SELECTION involves convex hull computations because the "potentially optimal" boxes are on the convex hull of coordinate pairs containing box center function values and box diameters (Jones et al. 1993). The present work also incorporates an "aggressive" implementation (Watson and Baker 2001) of SELECTION that bypasses the convex hull computation to produce more function

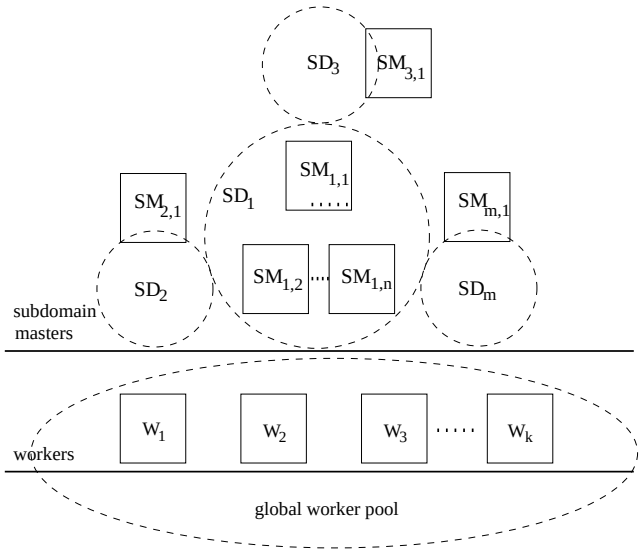


Fig. 3.1 The parallel scheme.

evaluation tasks, but similar experiments and analysis are not repeated for the present work.

When the amount of intermediate data may potentially grow beyond the memory capacity on a single machine, multiple masters should be used to share the data and collaborate on SELECTION in parallel. Also, the multiple masters update the intermediate results at the end of each iteration and check whether the stopping condition is met. Several studies (Banino et al. 2004 and Aida et al. 2003) have shown that an appropriate configuration of multiple masters can improve the overall performance. The present work recommends that masters evaluate functions locally if the objective function cost is lower than the communication round trip cost between two nodes on the parallel system. This forms the horizontal 1-D scheme described by He et al. (2006), in contrast with the vertical 1-D scheme with a single master distributing the function evaluation tasks to remote workers. Stacking function evaluations is another approach to reduce the communication overhead for distributing cheap objective function evaluations under the vertical scheme. Because an unpredictable number of boxes and sample points are generated at each iteration, these two schemes can be used separately or together to achieve the best performance under particular circumstances discussed in Section 4.

The overall parallel scheme is shown in Figure 3.1 with the user configurable m subdomains (SDs), n subdomain masters (SMs), and k globally shared workers (Ws) that request tasks from randomly selected SMs. The above design is implemented purely in Fortran 95 to support high accuracy computation

and dynamic data structures that expand at run time for rapidly growing intermediate data. It can be executed on either a single processor or multiple processors depending on the space and computation complexity of the optimization problem. If multiple processors are used, a small set of the MPI library functions are called to establish the interprocessor communication and synchronization. In addition, practical checkpointing methods were implemented to enhance fault tolerance in case of system failure. The key contributions of the massively parallel design are the important techniques developed to reduce the local memory requirement, minimize the network traffic, and balance the workload. Section 4 applies a few modeling and analysis tools to study the performance impact of these techniques and presents convincing experimental results to support the theoretical results.

4 PERFORMANCE ANALYSIS

Several optimization parameters and system characteristics listed in Table 4.1 are selected for the performance sensitivity analysis. In fact, more input parameters are required to define the optimization problem. For example, the upper and lower bounds of the design domain define the search region. Varying the bounds certainly affects the convergence rate as well as the total computational cost to reach the solution, so the proper bounding for a particular problem deserves a thorough application-oriented study by researchers in that field. In the present work, upper and lower bounds are fixed (see Table 2.1) for each problem.

Table 4.1 Parameters and characteristics under study.

#	Description
N_d	Problem dimension
$I_{\max}$	maximum iterations
N_b	number of evaluations per task
m	number of subdomains
n	number of masters per subdomain
k	number of workers
T_f	objective function cost, seconds
T_{cp}	point-to-point round trip cost, microseconds
T_{ca}	one-to-all broadcast cost, microseconds

Table 4.2 Comparison of I_{out} (the number of iterations before the program halts on a single processor) without (NON-LBC) and with LBC for all test problems. $I_{\max} = 1000$.

#	NON-LBC	LBC
GR	206	467
QU	90	1000
RO	105	122
SC	95	453
MI	82	936
FE	316	316
BY	185	477

The first two parameters N_d and $I_{\max}$ directly affect the problem scale. $I_{\max}$ is also required to enable the local memory reduction technique discussed in Section 4.1. The parallel scheme parameters N_b, m, n, and k have great impact on both minimization performance and parallel performance such as load balancing. In Section 4.2, these performance metrics are analyzed theoretically and experimentally. The next subset $\{T_f,\ T_{cp},\ T_{ca}\}$ determines the task granularity defined as the ratio of the computation time to communication time per task. These parameters are addressed in Part 2, where detailed discussions of the influence of these parameters on overhead and scalability are given.

4.1 Problem Configuration

As N_d grows, the memory requirement imposed by the intermediate data increases dramatically as shown in Figure 4.1. A local memory reduction technique—LBC (limiting box column)—was developed in He et al. (2006) to take advantage of $I_{\max}$, which limits the number of boxes stored in memory for remaining iterations at run time, thus reducing the box memory usage. The bar plot in Figure 4.2 compares the memory allocated for holding boxes with and without LBC for all test problems. The first five artificial functions set $N_d = 150$ to generate an amount of intermediate data comparable to that for the BY problem. Memory usage for the FE problem is the smallest, because it has the lowest N_d. All tests were run on a single machine until (1) the stopping condition $I_{\max} = 1000$ was reached, or (2) the diameter of the box holding the best solution became smaller than the problem precision, or (3) the run crashed due to memory allocation failure. Table 4.2 compares the number of iterations I_{out} for all test problems on a single processor before it halts due to one of the above three conditions. The LBC technique reduces the memory usage by 10–70% for the test problems so that the program can run longer without memory allocation failure on a single processor.

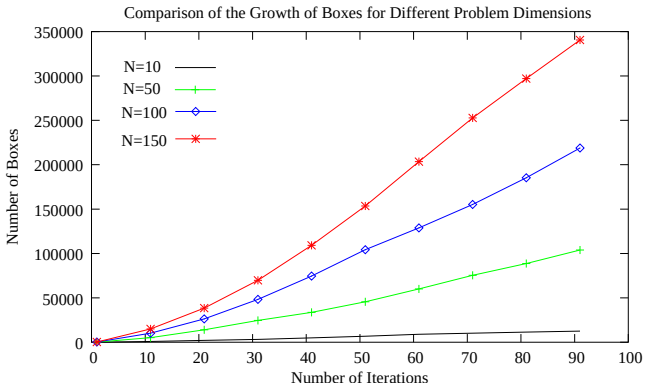


Fig. 4.1 The growth of boxes for the RO test function. Dimensions $N_d = 10$, 50, 100, and 150. $I_{\max} = 100$.

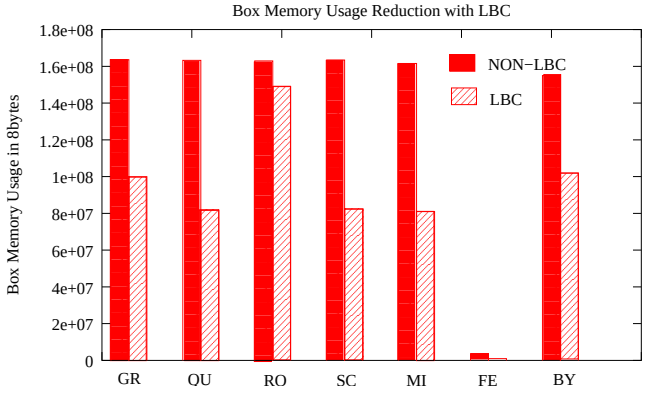


Fig. 4.2 Comparison of box memory usage without (NON-LBC) and with LBC for all the test problems.

Large scale problems with higher dimensions and larger $I_{\max}$ are intended to generate more work that keeps a large number of processors busy. Figure 4.3 shows the comparison of parallel efficiencies with various N_d and $I_{\max}$ for the RO function using $p = 100$ processors. The parallel efficiency E is the ratio of (algorithmic) speedup $S = T_s/T_p$ to p, where T_s is the execution time with a single processor and T_p is the parallel execution time with p processors. E is improved with both increasing N_d and increasing $I_{\max}$, because higher N_d and larger $I_{\max}$ yield more function evaluations. Better load balancing is the real reason behind the improved parallel efficiency as the problem scale grows. A workload model bound is discussed in Section 4.2 to analyze the performance influence of the parallel scheme parameters.

4.2 Scheme Configuration

4.2.1 Parameter N_b

He et al. (2006) propose a theoretical lower bound on the parallel execution time T_t for the parallel scheme

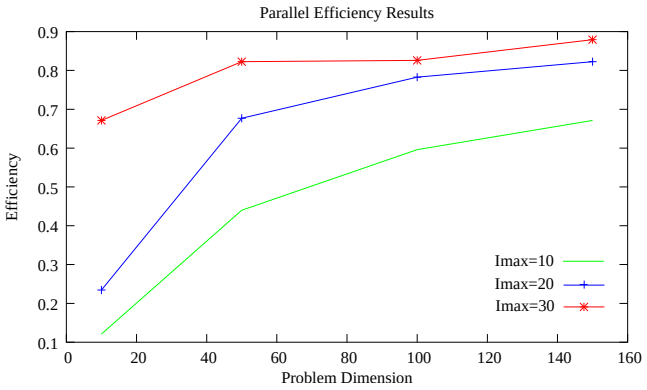


Fig. 4.3 Parallel efficiency comparison for the RO function using 100 processors. $N_d = 10$, 50, 100, and 150. $I_{\max} = 10$, 20, and 30.

with k workers:

$$T_t(N_b) = \sum_{i=1}^{I_{\max}} \left\lceil \frac{F_i}{k} \right\rceil (N_b T_f), \tag{4.1}$$

where F_i is the number of function evaluation tasks at iteration i, and N_b is the number of point evaluations per task. Implicitly, such a lower bound assumes $T_f > T_{cp}$, a necessary condition for achieving reasonable speedup when distributing tasks to remote workers. It considers the computation time of function evaluations and the idle time of workers waiting for new tasks to become available at the beginning of each iteration.

With P_i being the number of point evaluations required at iteration i, the number of tasks $F_i = \lceil P_i/N_b \rceil$, and thus the theoretical lower bound for iteration i is

$$\left\lceil \frac{F_i}{k} \right\rceil (N_b T_f) = \left\lceil \frac{\lceil P_i/N_b \rceil}{k} \right\rceil (N_b T_f).$$

This is clearly minimal when both $N_b \mid P_i$ and $k \mid (P_i/N_b)$, but lacking this divisibility, the minimum occurs for $N_b = 1$, since

$$\begin{aligned} \left\lceil \frac{\lceil P_i/N_b \rceil}{k} \right\rceil N_b &= \left(\frac{P_i}{N_b k} + \frac{a}{N_b k} + \frac{b}{k} \right) N_b \\ &= \frac{P_i}{k} + \frac{a}{k} + \frac{b N_b}{k}, \end{aligned}$$

where $0 \le a < N_b$ and $0 \le b < k$, is clearly minimal for $N_b = 1$. (Note: (4.1) is valid except for the special case where $F_i \bmod k = 1$ and one task has $N_{b-} = P_i \bmod N_b < N_b$ point evaluations. In this case the optimal choice for N_b is the value that minimizes $\big((k+1)N_{b-} - N_b\big)/k$. Since for $k \gg 1$,

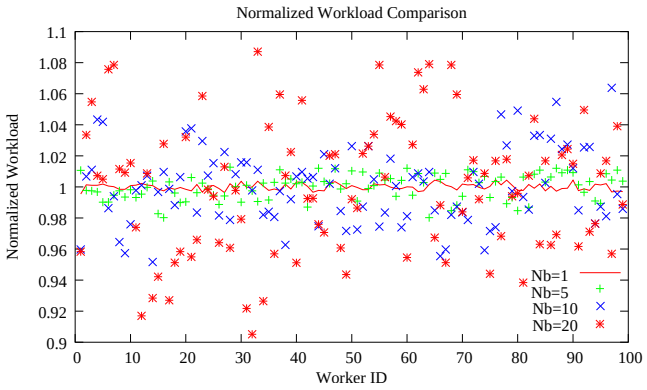
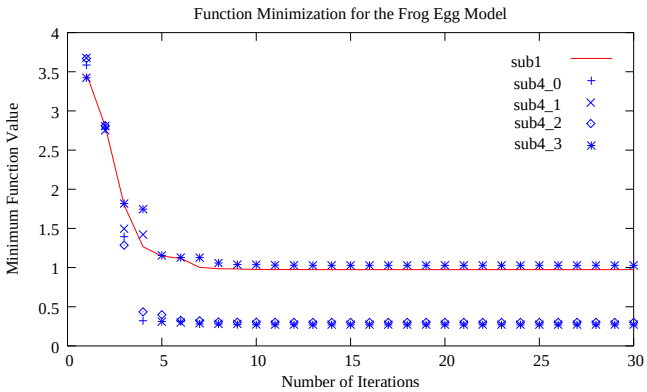


Fig. 4.4 Comparison of the normalized workload on 99 workers with $N_b = 1$, 5, 10, and 20 for the 150-dimensional GR function.



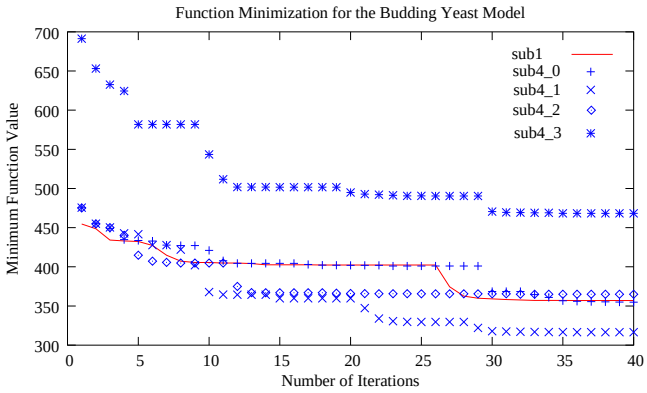
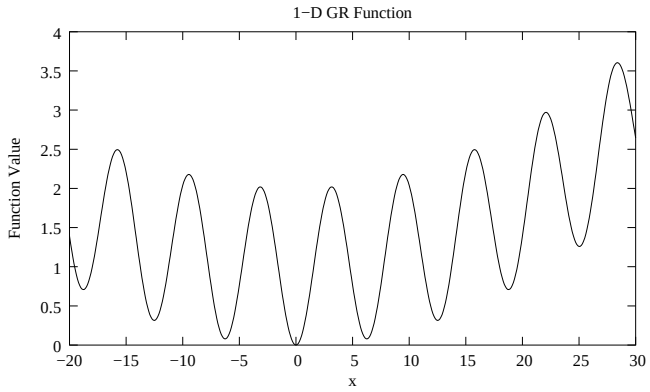


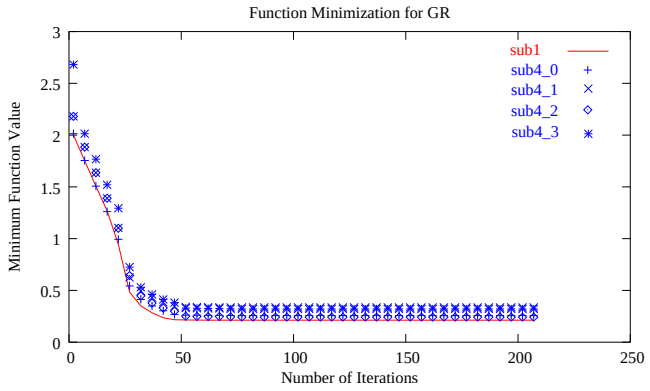
Fig. 4.5 Comparison of function minimization for a single domain (sub1) and four subdomains (sub4_i), where i is the subdomain ID.

$F_i \bmod k = 1$ is a rare event, this special case is ignored here.)

Figure 4.4 compares the normalized workload on 99 workers with increasing $N_b = 1$, 5, 10, and 20. A better load balance is achieved when $N_b = 1$, so $N_b > 1$ should only be used to stack cheap function evaluations to achieve $N_b T_f > T_{cp}$. Detailed analysis for load balancing is given in the following subsection.



(a)



(b)

Fig. 4.6 (a) The 1-D view of the GR function. (b) Comparison of function minimization on a 150-dimensional GR function for a single domain (sub1) and four subdomains (sub4_i), where i is the subdomain ID.

4.2.2 Subdomain parameter m

The scheme parameter m, the number of subdomains, is specified according to the computational budget and optimization goal. With more function evaluations generated across multiple subdomains, the same $I_{\max}$ likely yields a better solution than the single domain search. Figure 4.5 compares the function minimization progress with a single domain ($m = 1$) and four subdomains ($m = 4$) for FE and BY, the cell cycle parameter estimation problems discussed in Section 2. In both cases, the single subdomain search results in higher minimum function values. This behavior indicates an irregular or asymmetric problem structure, a common case for many science and engineering problems. If the problem structure is symmetric or the global minimum is near the center, a single domain DIRECT search may converge to the target faster than the multiple subdomain search. The GR function is such an example shown in Figure 4.6. The single domain search progresses slightly better than the four subdomain search, since the global solution is located close to the center of the design domain.

The second advantage of the multiple subdomain search is improved load balancing among workers. He et al. (2006) show that function evaluation tasks are distributed more efficiently to workers with a multiple subdomain search than a single domain search, especially when subdomains have unequal amounts of work. The globally shared workers have better chance to find work if the subdomain masters are not synchronized due to the data dependency in a single domain. An analytical workload model is constructed here to interpret the experimental observations. For simplicity, the objective function cost T_f is assumed to be constant. Again, $T_f > T_{cp}$ is assumed so that the workload model can ignore all the communication overhead.

The model is developed from the simplest case with one subdomain $m = 1$, one master $n = 1$ and extended to cases with $m = 1$, $n > 1$ and $m > 1$, $n > 1$. The potential communication bottleneck at masters is not considered in this section assuming the message buffers on master processors are sufficiently long and the service time for each message is negligible. Due to the variability of workload distribution among workers, a bounding analysis is applied to capture the lower and upper workload bounds. Bounding techniques require less computation and weaker assumptions than exact solution techniques, but sufficiently characterize the system performance under variability and uncertainties (Luthi et al. 1997).

At every iteration, each master dispatches function evaluation tasks to workers upon request during SAMPLING, each task defined by the coordinates of N_b points. As before, let P_i be the total number of point evaluations during iteration i, and $F_i = \lceil P_i/N_b \rceil$ the total number of function evaluation tasks. When P_i is not exactly a multiple of N_b, the last task has $N_{b-} = P_i - \lfloor P_i/N_b \rfloor N_b < N_b$ points, but all the remaining tasks have N_b points. A task costs either $T_b = N_b T_f$ or $T_{b-} = N_{b-} T_f$. At iteration i, each of k workers obtains either $\delta_{i-} = \lfloor F_i/k \rfloor$ or $\delta_{i+} = \lceil F_i/k \rceil$ tasks. The workload on each worker is defined as the total computation time for function evaluations. When F_i is a multiple of k and P_i is not a multiple of N_b, the worker that obtains the last task with N_{b-} points reaches the lower bound $WL_{li} = (\delta_{i-} - 1)T_b + T_{b-}$. In other cases, the workload lower bound is $WL_{li} = \delta_{i-} T_b$. Note that $\delta_{i-} = 0$ when $F_i < k$. Summing over all $I_{\max}$ iterations, the workload lower bound is

$$WL_l = \sum_{i=1}^{I_{\max}} WL_{li},$$

where $WL_{li} = (\delta_{i-} - 1)T_b + T_{b-}$ if $k \mid F_i$ and N_b does not divide P_i; $WL_{li} = \delta_{i-} T_b$ otherwise.

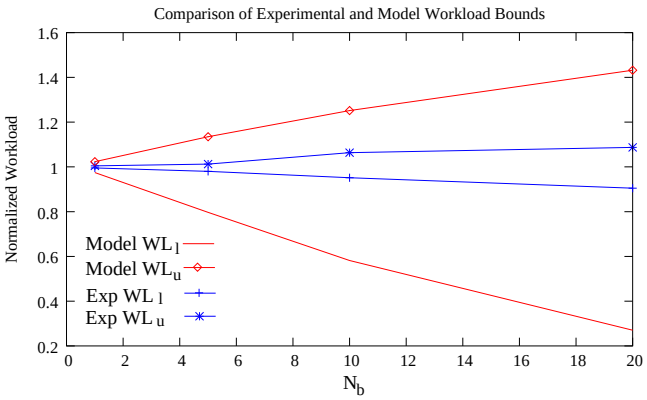


Fig. 4.7 Comparison of the normalized workload bounds based on the model and the experiments with 99 workers and $N_b = 1$, 5, 10, and 20 for the 150-dimensional GR function.

Following the same reasoning, the workload reaches the upper bound when a worker obtains δ_{i+} tasks, all of which have N_b points to compute or one of which has $N_{b-} > 0$ points if the remainder of F_i/k is 1. So the upper bound

$$WL_u = \sum_{i=1}^{I_{\max}} WL_{ui},$$

where $WL_{ui} = (\delta_{i+} - 1)T_b + T_{b-}$ if $F_i - \lfloor F_i/k \rfloor k = 1$ and $N_{b-} > 0$; otherwise, $WL_{ui} = \delta_{i+} T_b$.

To compare workload balance for different problems, the workload is normalized by dividing by the average workload $\overline{wl} = \sum_{i=1}^{I_{\max}} (P_i/k)T_f$. Hence, the normalized workload WL_j on worker j ($j = 1, 2, \ldots, k$) is within the range $(WL_l/\overline{wl}, WL_u/\overline{wl})$. Let $WR = WL_u/\overline{wl} - WL_l/\overline{wl}$ be the measure of workload balance, so smaller WR is better. Observe that WL_l would be closer to WL_u with $N_b = 1$, because then $N_b \mid P_i$ always, yielding a larger WL_l. Recall that $N_b = 1$ also gives the smallest possible $WL_u = \sum_{i=1}^{I_{\max}} \lceil P_i/k \rceil T_f \le \sum_{i=1}^{I_{\max}} \lceil \lceil P_i/N_b \rceil / k \rceil N_b T_f$.

Figure 4.7 plots the workload ranges estimated with the model and the actual workload ranges measured from experiments for different N_b values. All the experimental workload bounds are within the model estimation, but the difference between the experimental bounds and the model estimation becomes larger as N_b increases. The model sums up the lowest workload among workers over all iterations, while in reality, the workload on a particular worker has a fair chance to be the lowest at every iteration. Therefore, the load balancing measurement is better

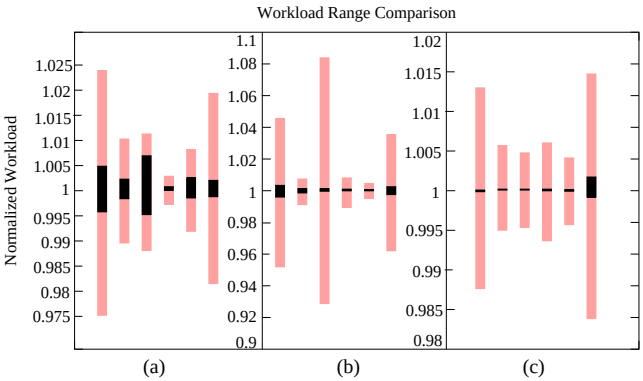


Fig. 4.8 Comparison of the workload ranges based on the model (grey) and the experiments (black) for (a) a single domain search with 1 master and 99 workers, (b) a single domain search with 4 masters and 196 workers, and (c) a four subdomain search with 1 master per subdomain and totally 196 workers. $N_b = 1$**. The vertical bars, left to right, correspond respectively to problems GR, QU, RO, SC, MI, BY.** $I_{max} = 90$ **and** $N_d = 150$ **for the artificial problems.** $I_{\max} = 40$ **and** $N_d = 143$ **for the BY problem.**

than the model estimation because of the inherent randomness of the parallel scheme.

For high dimensional test problems, Figure 4.8(a) compares the workload ranges based on the model and the experiments for a single domain search using one master and 99 workers. Consider only $N_b = 1$ for simplicity in the remaining analysis and discussion. The model estimations in Figure 4.8(a) are all within 2.1% of the measured workload bounds from experiments.

The above model can be easily extended to the case $n > 1$ with multiple masters sharing the memory burden and the computation involved in the parallel SELECTION. Since workers randomly select masters to request work, each master initially has an approximately equal number (k/n) of workers to compute its function evaluation tasks. Although masters may have different numbers of tasks, the global shared worker pool balances the workload by assigning workers to masters with work. A weight variable $w_{i,j} = F_{i,j}/F_i$ is introduced, where $F_{i,j} = w_{i,j}F_i$ is the number of tasks on master j for iteration i. Master j has $w_{i,j}F_i$ tasks, thus $w_{i,j}k$ workers are assigned to it. Each worker obtains $\delta_{i-} = \lfloor w_{i,j}F_i/(w_{i,j}k)\rfloor$ or $\delta_{i+} = \lceil w_{i,j}F_i/(w_{i,j}k)\rceil$ tasks. It turns out that $w_{i,j}$ is canceled out for the workload estimation. Hence, the number of masters n in a single domain search has little impact on the workload bounds. Table 4.3 shows the experimental results for the parallel execution time T_k and the normalized workload range (WR_k) using $k = 100$, 200 workers in a single domain search with the number of masters n varying from one to eight. The timing and load balancing results are very close to each other for all n.

Table 4.3 Comparison of the parallel execution time T_p **and the measured normalized workload range** WR_p **for the RO function with** $N_d = 150$**,** $T_f = 0.1$ **second, and** $I_{\max} = 90$ **using 100 and 200 workers.**

		Number of Masters			
k		1	2	4	8
100	T_k	348.56	346.90	351.39	348.04
	WR_k	0.013	0.015	0.018	0.019
200	T_k	200.30	183.00	184.20	182.59
	WR_k	0.018	0.026	0.024	0.032

The random worker assignment strategy plays an important role in balancing the workload among workers. If fixed worker assignment is used instead, each master would have a fixed number of workers (k/n) to compute the function evaluations for all iterations. Workers assigned to master j would obtain either $\delta'_{i+} = \lceil (w_{i,j}F_i)/(k/n)\rceil$ or $\delta'_{i-} = \lfloor (w_{i,j}F_i)/(k/n)\rfloor$ tasks. Note that $w_{i,j} = 1/n$ produces the same assignment as in the case of random worker assignment. With some $w_{i,j} < 1/n$ (implying $\delta'_{i-} \le \delta_{i-}$) and some $w_{i,j} > 1/n$ (implying $\delta'_{i+} \ge \delta_{i+}$) under the fixed worker assignment, the lower bound WL_l becomes smaller and the upper bound WL_u becomes greater, but the average workload $\overline{wl}$ is the same, so the workload range $(WL_u - WL_l)/\overline{wl}$ becomes larger, indicating a worse balanced workload.

In the previous two cases, the masters in a single subdomain $m = 1$ serve as a single work source for workers. Multiple subdomains $m > 1$ serve as multiple work sources, because the masters from different subdomains are not synchronized to update intermediate results or to compute the global convex hull boxes. A similar weight variable w'_j is used in this case to estimate the number of workers assigned for the subdomain j:

$$w'_j = \sum_{i=1}^{I_{\max}} F_{i,j} \Big/ \sum_{j=1}^{m}\sum_{i=1}^{I_{\max}} F_{i,j}, \qquad (4.2)$$

where $F_{i,j}$ denotes the number of tasks at iteration i for subdomain j. w'_j is based on the overall tasks from all iterations because subdomains are asynchronous. In contrast, $w_{i,j}$ in the single domain case is defined for each iteration due to the global synchronization inside a subdomain. For subdomain j, the average workload is $\overline{wl_j} = \sum_{i=1}^{I_{\max}} F_{i,j}/(w'_j k)T_b$. The workload lower bound

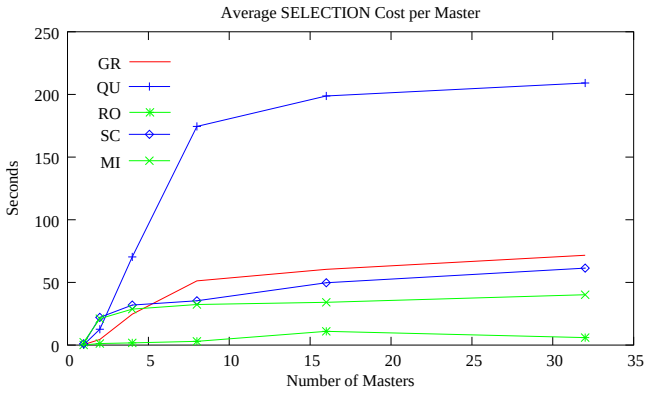


Fig. 4.9 The average SELECTION cost per master increases as n grows from 1 to 32 for test problems. $I_{\max} = I_{out}$ with LBC support listed in Table 4.2.

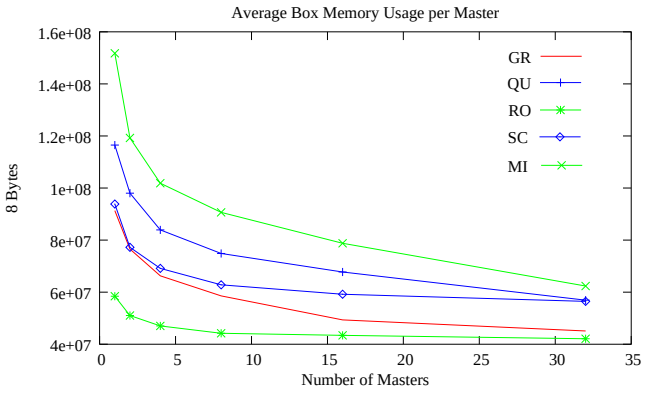


Fig. 4.10 The average box memory size decreases as n grows from 1 to 32 for test problems. $I_{\max} = I_{out}$ with LBC support listed in Table 4.2.

in subdomain j is $WL_{lj} = \sum_{i=1}^{I_{\max}} \lfloor F_{i,j}/(w'_j k) \rfloor T_b$, where $T_b = T_f$ because $N_b = 1$; the workload upper bound is $WL_{uj} = \sum_{i=1}^{I_{\max}} \lceil F_{i,j}/(w'_j k) \rceil T_b$. Hence, the overall normalized workload range is

$$\left(\min_{1 \le j \le m} (WL_{lj}/\overline{wl_j}), \max_{1 \le j \le m} (WL_{uj}/\overline{wl_j}) \right)$$

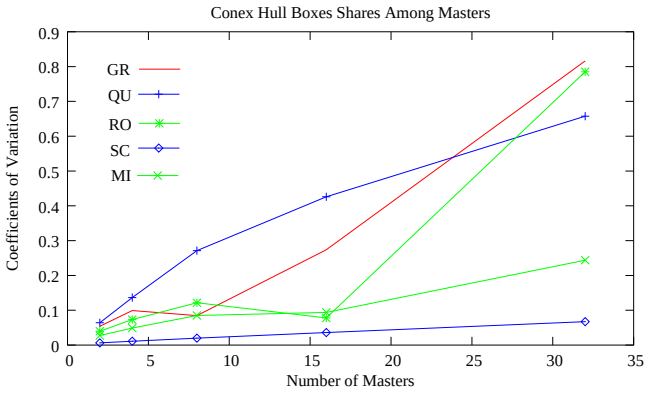
across all m subdomains. Figure 4.8(c) compares the model estimated workload ranges with the experimental results for a four subdomain search using 196 workers and four masters, one per subdomain. All the experimental workload results fall within the 1.5% ranges estimated by the model. Due to the randomness of workers' requests to masters, the experimental workload is better balanced than the (worst case) model estimation. For comparison, the workload ranges based on the model and experiments for a single domain search using the same number of masters and workers are shown in Figure 4.8(b). To keep the problem the same as in the four subdomain search, the single domain search is applied to each of the four subdomains sequentially using 49 workers and one master. The workload is normalized over the total workload of the four runs. The model estimated range for the single domain is wider than that for the four subdomains. This estimation is verified by the experimental results, which present smaller ranges for the four subdomain search (dark bars in Figure 4.8(c)) than for the single domain search (dark bars in Figure 4.8(b)).

4.2.3 Parameter n

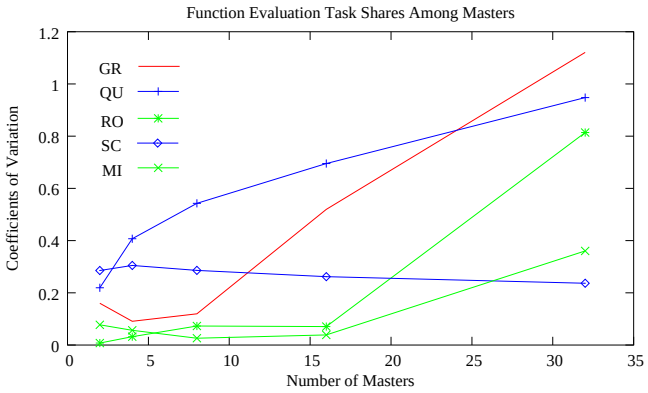
The number n of masters affects the efficiency of SELECTION and data distribution. Convex hull computation is the key component in SELECTION (cf. Section 3). The parallel SELECTION involves both local and global computation of convex hull boxes. First, each subdomain master computes the local convex hull boxes. Next, the root subdomain master gathers all the local convex hulls to find a global convex hull set of boxes. Finally, each subdomain master starts SAMPLING on its own portion of the convex hull boxes.

For the local convex hull computation, the gift-wrapping algorithm with complexity $\mathcal{O}(N^2)$ (Manber 1989) is preferred because it requires no extra space. A faster algorithm, Graham's scan (Graham 1972), is used for the global convex hull computation at the root subdomain master. It takes $\mathcal{O}(N)$ operations because the gathered local convex hull boxes are already sorted, saving $\mathcal{O}(N \log(N))$ operations required for sorting at the root subdomain master. Although the Graham's scan needs a stack for back-tracking, the buffer for merging local convex hull boxes also serves as a stack of boxes, which are indexed by integer pointers for back-tracking. As n increases, the number of sets of local convex hull boxes grows, and more boxes are merged at the root subdomain master for the final global computation. Since the root needs to gather from all and broadcast to all at each iteration, the communication overhead may overshadow the benefit of sharing the memory burden. Therefore, the value of n should be large enough to accommodate the intermediate data, yet no larger to minimize the network traffic for global communication.

To investigate the performance impact of n, other scheme parameters need to be fixed. The horizontal scheme with a single domain ($m = 1$), no workers ($k = 0$), and a varying number n of masters is employed to study the computational complexity of SELECTION and the balance of data distribution among masters. The analysis of n's influence on the communication overhead is presented in detail in Part 2 of this paper, which discusses parameters related to



(a)



(b)

Fig. 4.11 Coefficient of variation of (a) convex box shares and (b) function evaluation task shares as p grows from 2 to 32 for the horizontal scheme for test problems with $N = 150$.

system settings. Let LD_i be the number of different box diameters and LC_i the number of local convex hull boxes on master i ($i = 1, \ldots, n$). At the root subdomain master, $GD \leq \sum_{i=1}^{n} LC_i$ is the merged number of distinct box diameters and $GC \leq GD$ is the final number of convex hull boxes. The total convex hull computation includes n local gift-wrappings $\left(\mathcal{O}\left(\sum_{i=1}^{n}(LD_i)^2\right)\right)$, n global merges $\left(\mathcal{O}\left(\sum_{i=1}^{n} LC_i\right)\right)$, and one global Graham's scan ($\mathcal{O}(GD)$). Figure 4.9 shows how the average SELECTION cost per master grows as n increases for five test problems. Here, the cost includes the communication overhead and the convex hull computation. The motivation for using multiple masters is not to reduce the SELECTION cost, but to share the memory burden. As n grows from one to 32, the average memory storage for boxes is decreased significantly (shown in Figure 4.10).

4.2.4 Parameter k

In the previous section, the number of workers $k = 0$ forming a horizontal scheme of pure masters. In the vertical scheme, $k \geq 2$ workers are used with a single master. High objective function cost is one of the factors that favor the vertical scheme, where a large number of workers can be utilized efficiently. More importantly, the poor load balancing in the pure horizontal scheme becomes the major obstacle to using a large number of masters that compute function evaluations locally. At each iteration, convex hull boxes are assigned to masters using a simple balancing heuristic that apportions the boxes into approximately equal shares. However, the number of function evaluation tasks on a master depends on the number of longest dimensions contained by its share of the convex hull boxes. So the function evaluation tasks on masters are still very likely to be unbalanced even if all masters obtain the same number of convex hull boxes.

Finding an optimal convex hull box assignment that minimizes the variance of the combined number of longest dimensions on all masters is an NP-complete problem similar to the knapsack or bin packing problems (Manber, 1989). Although a polynomial time approximate solution is available (i.e., dynamic programming), it does not solve all the balancing problems. Even if a supposedly optimal box assignment is found, a load imbalance still occurs when (1) an insufficient number of convex hull boxes is generated, or (2) T_f varies at different sample points. The first cause is inevitable in early stages of the DIRECT search, and it also happens more often for low dimension problems or problems with certain structures that produce a small number of convex hull boxes. The second cause is also common in many engineering design problems, such as the ray-tracing technique and WCDMA simulation in wireless communication system design (He et al. 2004b).

Figure 4.11 shows the coefficient of variation (a) c_b for the convex hull box shares per processor and (b) c_f for the function evaluation task shares per processor with $p = 2$, 4, 8, 16, 32 for test problems with $N = 150$ and $I_{\max} = I_{out}$, the maximum number of iterations before the program halts on a single processor with the LBC support (see Table 4.2). The general trend is that both the convex hull and the function evaluation shares become less balanced as p grows except for the problem SC. For c_b shown in Figure 4.11(a), the problems GR, QU, and RO have a sharp increase, but the problems SC and MI increase slightly, because a larger number of convex hull boxes are generated per iteration for the last two problems (see Table 4.4, a list of the average convex hull boxes $\overline{N_{box}} = \sum_{i=1}^{I_{out}} C_i/I_{out}$ and function evaluation tasks per iteration $\overline{N_f} = \sum_{i=1}^{I_{out}} F_i/I_{out}$, where C_i is the number of convex hull boxes and F_i is the number of

Table 4.4 Comparison of the average number of convex hull boxes $\overline{N_{box}}$ and function evaluation tasks $\overline{N_f}$ per iteration for test problems.

#	$\overline{N_{box}}$	$\overline{N_f}$
GR	13.341	2990.2
QU	10.197	2007.8
RO	16.025	4234.7
SC	103.52	17152.0
MI	26.521	5942.0

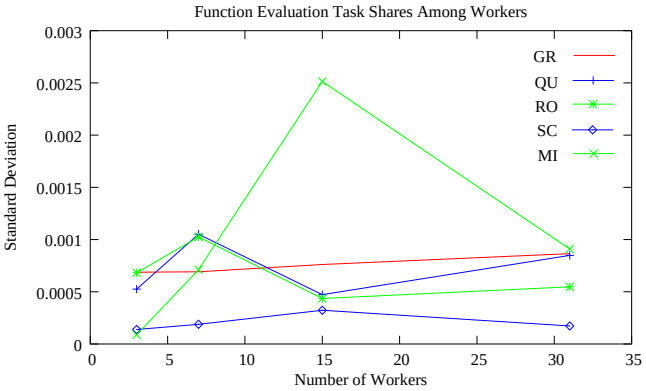


Fig. 4.12 Coefficient of variation of the function evaluation task shares as p grows from 4 to 32 for the vertical scheme for test problems with $N = 150$.

function evaluation tasks at iteration i). As the only exception, the problem SC has an improved function evaluation task balance even though its convex hull box balance becomes worse as p grows. However, if more than the necessary number of processors are used, i.e., $p > \overline{N_{box}}$, the problem SC will eventually become unbalanced on convex hull box and function evaluation task distribution among master processors.

For the vertical scheme, a similar set of experiments with $p \geq 4$ demonstrates a significant improvement on the function evaluation task balance among worker processors as shown in Figure 4.12. Not only does c_f decrease by two orders of magnitude, but the narrower range of values of c_f indicates better scalability. The drawback of the pure vertical scheme is the limited memory capacity on the single master processor that is dedicated to dispatching function evaluation tasks instead of carrying out the actual function evaluations as the masters do in the horizontal scheme. Therefore, the ultimate solution is the hybrid scheme that shares the memory burden on more than one master if necessary ($n \geq 1$), and dynamically assigns k workers to task-loaded masters. The number k of workers should be at least twice the number of masters, that is, $k \geq 2mn$.

4.3 System Configuration

All the analysis so far in this paper (Part 1) is based on an ideal computing environment ignoring any communication overhead. Part 2 addresses in detail the system configuration characterized by T_{cp}, T_{ca}, and T_f, followed by a scalability analysis.

5 CONCLUSION AND FUTURE WORK

Some performance factors for a massively parallel version of the global optimization algorithm DIRECT have been extensively studied using modeling and analysis tools such as bounding models, linear regression models, and isoefficiency functions. The analysis covered a broad range of performance metrics including memory usage, parallel efficiency, load balancing, and scalability.

The crucial factors proved to be the problem structure and configuration that directly affect the amount of data dependency overhead and memory requirement. The next most important factors are the scheme configuration parameters that determine four different parallel schemes—pure vertical, pure horizontal, hybrid single domain, and hybrid multiple subdomains. The vertical scheme outperforms the horizontal scheme in most cases, except for a small number of processors ($p < 5$) and cheap objective functions, in which case, stacking function evaluations ($N_b > 1$) should be considered to reduce the communication overhead. The hybrid schemes are recommended for large scale optimization applications with high computational cost and memory requirement. The hybrid scheme with multiple subdomains has demonstrated its superiority in balancing workload, thus reducing overhead and improving the overall scalability.

Moreover, several important design considerations for the massively parallel DIRECT can be generalized for global search algorithms, as follows.

(1) Unnecessary data storage should be released dynamically to reduce the memory burden.
(2) Tasks should be dynamically distributed in the smallest possible chunks to ensure the best possible load balancing.
(3) Point sampling and evaluation can be decoupled to enhance the program concurrency.
(4) Domain decomposition not only results in better optimization solutions for problems with irregular structures, but also improves load balancing and scalability if using a large number of processors.

The new insights gained from the present work suggest a fresh research direction for conquering the biggest challenge—the data dependency of DIRECT. Advanced algorithm steps will be designed for

SAMPLING to prefetch enough function evaluation tasks, generated from selected boxes that are not on the convex hull, so that the current idle worker cycles would be put to use. The function values at these extra sampling points may not necessarily contribute to the optimization process, so an optimal selection strategy would balance such waste with the benefit of idle cycle computations that *do* further the DIRECT search. Also, the optimal strategy would preserve the determinism of DIRECT, contrasted with nondeterministic methods that produce different solutions on different runs. Of paramount importance is that the proposed modification does not destroy DIRECT's global convergence property. Given the recent interest in DIRECT both in the mathematics and computer science communities, the prospects for significant progress are bright.

ACKNOWLEDGMENTS

This work was supported in part by National Science Foundation Grant DMI-0355391, Department of Energy Grant DE-FG02-06ER25720, and NIGMS/NIH Grant 1 R01 GM078989-01. The authors also gratefully acknowledge access to System X provided by the Virginia Tech Terascale Computing Facility.

BIOGRAPHIES

REFERENCES

Aida, K., Natsume, W., and Futakata, Y. 2003. Distributed computing with hierarchical master-worker paradigm for parallel branch and bound algorithm. In Proc. 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID'03), Tokyo, Japan.

Baker, C.A., Watson, L.T., Grossman, B., Haftka, R.T., and Mason, W.H. 2000. Parallel global aircraft configuration design space exploration. In Proc. High Performance Computing Symposium 2000, A. Tentner (Ed.), Soc. for Computer Simulation Internat, San Diego, CA, pp. 101–106.

Banino, C., Beaumont, O., Carter, L., Ferrante, J., Legrand, A., and Robert, Y. 2004. Scheduling strategies for master-slave tasking on heterogeneous processor platforms. IEEE Trans. on Parallel and Distributed Systems, Vol. 15, No. 4, pp. 319–330.

Bartholomew-Biggs, M.C., Parkhurst, S.C., and Wilson, S.P. 2003. Global optimization approaches to an aircraft routing problem. EUR J. Operational Research, Vol. 146, No. 2, pp. 417–431.

Becker, D.J., Sterling, T., Savarese, D., Dorband J.E., Ranawake, U.A., and Packer, C.V. 1995. Beowulf:a parallel workstation for scientific computation. In Proc. International Conference on Parallel Processing, pp. 11–14.

Carter, R.G., Gablonsky, J.M., Patrick, A., Kelly, C.T., and Eslinger, O.J. 2001. Algorithms for noisy problems in gas transmission pipeline optimization. Optimization and engineering, Vol. 2, No. 2, pp. 139–157.

Clausen, J. 1997. Parallel branch and bound — principles and personal experiences. Parallel Computing in Optimization, Sverre Storoy (Ed.), Kluwer Academic Publishers, pp. 239–267.

Cox, S.E., Hart, W.E., Haftka, R., and Watson, L.T. 2002. DIRECT algorithm with box penetration for improved local convergence. In Proc. 9th AIAA/ISSMO Symposium and Exhibit on Multidisciplinary Analysis and Optimization, Atlanta, GA, AIAA Paper 2002–5581, 15 pp.

Dennis, J.E. and Torczon, V. 1991. Direct search methods on parallel machines. SIAM J. on Optimization, Vol. 1, pp. 448–474.

Finkel, D.E. and Kelly, C.T. 2004a. An adaptive restart implementation of DIRECT. Technical Report CRCS-TR04-30, Center for Research in Scientific Computation, North Carolina State University, Raleigh, NC, USA, August.

Finkel, D.E. and Kelley, C.T. 2004b. Convergence analysis of the DIRECT algorithm. Optimization On-line Digest, August.

Gablonsky, J.M. 2001a. Modifications of the DIRECT algorithm. Ph.D. thesis, Department of Mathematics, North Carolina State University, Raleigh, NC.

Gablonsky, J.M. and Kelly, C.T. 2001b. Locally-biased form of the DIRECT algorithm. J. of Global Optimization, Vol. 21, No. 1, pp. 27–37.

Graham, R. 1972. An efficient algorithm for determining the convex hull of a finite planar point set. Info. Proc. Letters, Vol. 1, pp. 132–133.

He, J., Sosonkina, M., Shaffer, C.A., Tyson, J.J., Watson, L.T., and Zwolak, J.W. 2004a. A hierarchical parallel scheme for global parameter estimation in systems biology. In Proc. 18th Internat. Parallel & Distributed Processing Symp., IEEE Computer Soc., Los Alamitos, CA, CD-ROM.

He, J., Verstak, A., Watson, L.T., and Sosonkina, M. 2006. Design and implementation of a massively parallel version of DIRECT. Computational Optimization and Applications, to appear.

He, J., Verstak, A., Watson, L. T., Stinson, C.A., Ramakrishnan, N., Shaffer, C.A., Rappaport, T.S., Anderson, C.R., Bae, K., Jiang, J., and Tranter, W.H. 2004b. Globally optimal transmitter placement for indoor wireless communication systems. IEEE Transactions on Wireless Communications, Vol. 3, No. 6, pp. 1906–1911.

He, J., Watson, L. T., Ramakrishnan, N., Shaffer, C. A., Verstak, A., Jiang, J., Bae, K., and Tranter, W. H. 2002. Dynamic data structures for a direct search algorithm. Computational Optimization and Applications, Vol. 23, No. 1, pp. 5–25.

Jones, D.R., Pertunen, C.D., and Stuckman, B.E. 1993. Lipschitzian optimization without the Lipschitz constant. J. Optimization Theory and Applications, Vol. 79, No. 1, pp. 157–181.

Ljungberg, K., Holmgren, S., and Carlborg, Ö. 2004. Simultaneous search for multiple QTL using the global optimization algorithm DIRECT. Bioinformatics (Oxford, England), Vol. 20, No. 12, pp. 1887–1895.

Luthi, J., Majumdar, S., Kotsis, G., and Haring, G. 1997. Performance bounds for distributed systems with workload variabilities and uncertainties. Parallel Computing, Vol 22, No. 13, pp. 1789–1806.

Manber, U. 1989. Introduction to Algorithms: a Creative Approach. Addison-Wesley, Reading, MA.

McMahon, M.T. and Watson, L.T. 2000. A distributed genetic algorithm with migration for the design of composite laminate structures. Parallel Algorithms and Applications, vol. 14, pp. 329–362.

Nelson, S.A.and Papalambros, P.Y. 1998. A modification to Jones' global optimization algorithm for fast local convergence. In Proc. 7th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization, St. Louis, MO, 2–4 Sept. 1998, pp. 341–348.

Panning, T.D., Watson, L.T., Allen, N.A., Chen, K.C., Shaffer, C.A., and Tyson, J.J. 2006. Deterministic global parameter estimation for a model of the budding yeast cell cycle. J. of Global Optimization, to appear.

Pohlheim, H. 1996. GEATbx: Genetic and Evolutionary Algorithm Toolbox for Use with Matlab–Documentation. Ph.D. thesis, Technical University Ilmenau, Germany.

Sergeyev, Ya.D. and Kvasov, D. 2006. Global search based on efficient diagonal partitions and a set of Lipschitz constants. SIAM J. on Optimization, Vol. 16, No. 3, pp. 910–937.

Talbi, E.G., Hafidi, Z. and Geib, J.M. 1998. A parallel adaptive tabu search approach. Parallel Computing, Vol. 24, pp. 2003–2019.

Watson, L.T. and Baker, C.A. 2001. A fully-distributed parallel global search algorithm. Engineering Computations, Vol. 18, No. 1/2, pp. 155–169.

Zhu, H. and Bogy, D.B. 2002. DIRECT algorithm and its application to slider air-bearing surface optimization. IEEE Transactions on Magnetics, Vol. 38, No. 5, pp. 2168–2170.

Zwolak, J.W., Tyson, J.J., and Watson, L.T. 2005. Globally optimised parameters for a model of mitotic control in frog egg extracts. IEE Systems Biology, Vol. 152, No. 2, pp. 81–92.