Performance Modeling and Analysis of a Massively Parallel DIRECT—Part 2

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

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. In this second part of a two-part work, theoretical and experimental results are compared for two parallel clusters with different system scale and network connectivity. The first part studied performance sensitivity to important parameters for problem configurations and parallel schemes, using performance metrics such as memory usage, load balancing, and parallel efficiency. Here 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 design considerations and analysis techniques generalize to the transformation of other global search algorithms into effective large scale parallel optimization tools.
Keywords: DIRECT, global search algorithms, isoefficiency, load balancing, parallel optimization, performance modeling, scalability analysis

1 INTRODUCTION

Part 1 (He et al. 2007) of this work describes the serial version of the deterministic global optimization algorithm DIRECT, sets a context for the applications of massively parallel global search, and outlines a recently developed massively parallel version of DIRECT. This paper continues the performance modeling and analysis of Part 1 by specifically addressing the computing environment and communication overhead issues that were ignored in Part 1. A detailed comparison between an Apple Xserve G5-based system with 2,200 processors (System X) and an AMD Opteron-based system with 400 processors (Anantham) is conducted.

Table 1.1 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

Both System X and Anantham are cluster systems of dual CPU nodes located at Virginia Tech. They provide different types of computing environments for the present study. On System X, 10 Gbps InfiniBand switches connect 1,100 nodes, each of which has two 2.3 GHz processors and 4 GB of main memory. On top of the 32-bit Mac OS X operating system, MVAPICH (MPI-1 implementation over VAPI (Liu et al. 2004b)) software is used to communicate via the InfiniBand interconnect. On Anantham, 200 nodes running the 64-bit Linux operating system, each with two 1.4 GHz processors and 1 GB of main memory, are interconnected over 2 Gbps Myrinet (Boden et al. 1995) interfaces underlying a GM (general messaging) communication platform. Both System X and Anantham have Gigabit Ethernet as a secondary network. A detailed performance

[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.

comparison of VAPI/InfiniBand and Myrinet/GM on a 32-node cluster can be found in Liu et al. (2004a).

Part 2 studies the impact of parallel system parameters on the performance of pDIRECT_II (He et al. 2006) on large-scale clusters, focusing on the latency modeling, overhead identification, and scalability analysis. The parallel system parameters include the objective function cost T_f and two network characteristics T_{cp} and T_{ca} as shown in Table 1.1, which recounts the relevant parameters under consideration for both Part 1 and Part 2.

The rest of Part 2 is organized as follows. Section 2 addresses the parameters T_f, T_{cp}, and T_{ca}. Section 3 offers a scalability analysis based on isoefficiency. The conclusions given in Part 1 apply to the entire body of work, so Section 4 here gives only those conclusions and future work pertinent to this paper, Part 2.

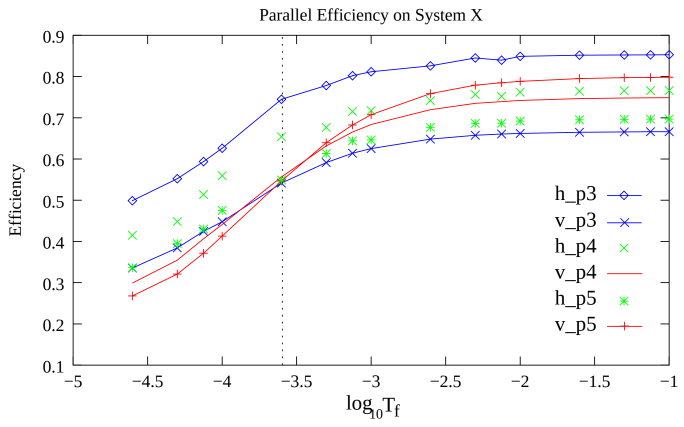
2 Parallel System Parameters

The overall scheme of pDIRECT_II consists of a single master or multiple masters with or without a global pool of shared workers. When masters take on function evaluation tasks locally without workers, they form a horizontal scheme. A vertical scheme is formed when workers are used to carry out the function evaluation tasks distributed by a master. Depending on the memory requirements, the horizontal scheme of multiple masters can be mixed with the vertical scheme to share the memory burden and dispatch tasks to workers. These schemes are the basic components that map data and tasks onto a cluster system of processors.

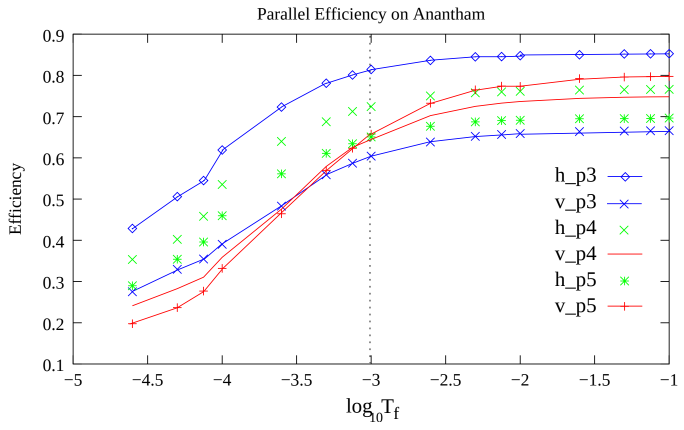
2.1 Objective Function Cost

The objective function cost T_f is the key parameter that affects parallel performance under different parallel schemes. An empirical study by He et al. (2006) concluded that the horizontal scheme outperforms the vertical scheme when $T_f \approx 0.0$, but performs worse than the vertical scheme when $T_f = 0.1$. The performance impact of $T_f \in (0.0, 0.1)$ was further investigated using the 150-dimensional GR function (Appendix) under both the vertical and horizontal schemes on System X and Anantham.

The first experiment used a small number of processors $p = 3$, 4, 5 with T_f being adjusted in 15 small time intervals from $T_1 = 0.1$ to $T_{16} = 2.5$E-05. Since the 150-dimensional GR function originally costs about 1.5E-05 on System X and 2.5E-05 on Anantham, T_{16} is chosen as the smallest objective function cost to make the test cases comparable on System X and Anantham. Figure 2.1 shows that the parallel efficiency increases sharply as T_f grows from T_{16} to T_{mid} (called the "performance boundary",



(a) System X.



(b) Anantham.

Fig. 2.1 Comparison of parallel efficiency as T_f changes in the range $(0, 0.1)$ using $p = 3$, 4, and 5 processors for the 150-dimensional GR function with $I_{\max} = 90$. "v_" stands for the vertical scheme and "h_" for the horizontal scheme.

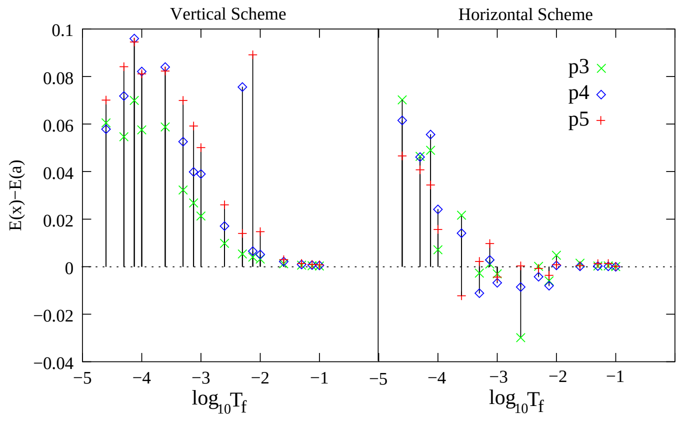


Fig. 2.2 Parallel efficiency differences from that on System X to that on Anantham (marked as "E(x)-E(a)") corresponding to the experimental results in Figure 2.1.

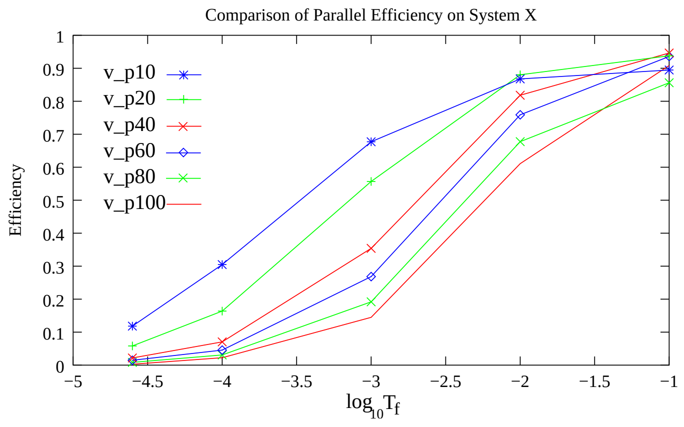
marked as dotted lines with the x coordinates being

2.5E-04 and 1.0E-03 on System X and Anantham, respectively), then grows slightly up to T_1, which gives the highest efficiency for a particular setting specified by the parallel scheme, the number of processors, and the computing platform system. Since $T_f < T_{mid}$ is comparable with the communication cost under either the vertical or the horizontal scheme, the communication overhead dominates, degrading the efficiency significantly. When $p = 3$ and 4, the horizontal scheme outperforms the vertical scheme regardless of the value of T_f because one processor is dedicated as the master in the vertical scheme. When $p = 5$ and $T_f \geq T_{mid}$, the vertical scheme starts to perform better than the horizontal scheme. The turning point T_{mid} is different for System X and Anantham due to the different underlying network properties characterized by T_{cp} and T_{ca}, which are considered in Section 2.2.

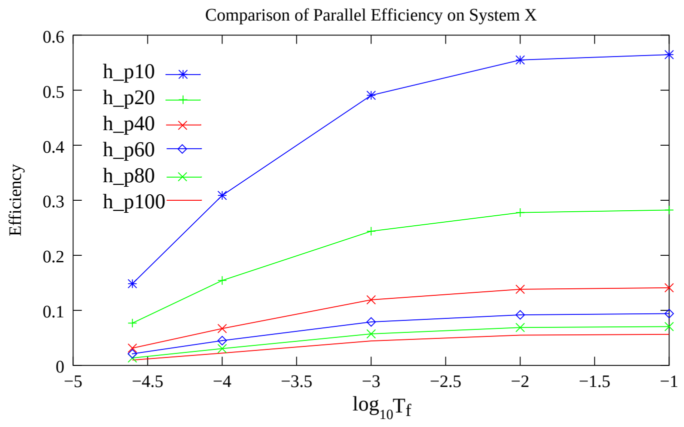
A smaller value of T_{mid} suggests a lower T_{cp} and/or a higher T_{ca}. The point-to-point communication, characterized by T_{cp}, happens more often in the vertical scheme than in the horizontal scheme, while the global one-to-all communication, with the cost T_{ca}, is a feature of the horizontal scheme. Moreover, the common overhead caused by data dependency and problem structure becomes more dominant than the communication overhead when $T_f > T_{mid}$ and eventually levels off the efficiency growth to the highest possible value for a particular setting.

The difference from $E(x)$ (the efficiency on System X) to $E(a)$ (the efficiency on Anantham) is denoted $E(x) - E(a)$ in Figure 2.2. Observe that $E(x) \geq E(a)$ for the vertical scheme, but for horizontal scheme, there are a few cases when $E(x)$ is worse than $E(a)$ for $T_f \in$ (2.5E-04, 1.0E-02). More importantly, the difference becomes very small as T_f grows beyond 1.0E-02. However, no inference should be drawn without further study that compares the performance impact of T_f using a larger number of processors.

The second experiment uses $p = 10, 20, 40, \ldots, 100$ processors with T_f being adjusted from the largest value $T_1 = 0.1$ to $T_i = T_{i-1} \times$ 1.0E-01 ($i = 2, 3, 4$) to the smallest value of $T_5 =$ 2.5E-05. As expected, the results on System X (Figure 2.3) show that the parallel efficiency of the vertical scheme is greatly improved, reaching 80–90% for all numbers of processors p when T_f grows from T_5 to T_1. However, the efficiency of the horizontal scheme is improved slightly except for $p = 10$, when it still does not surpass 60% for the largest value T_1. For the vertical scheme on System X, smaller p results in higher efficiency when $T_f \in (T_5, T_2)$. For the rest of the T_f values, reasonable efficiency is attained ranging from the lowest $E = 0.85$ to the highest $E = 0.94$ on 80 and 40 processors,



(a) The vertical scheme.



(b) The horizontal scheme.

Fig. 2.3 Comparison of parallel efficiency as T_f changes in the range $(0, 0.1)$ using $p = 10, \ldots, 100$ processors on System X for the 150-dimensional GR function with $I_{\max} = 90$.

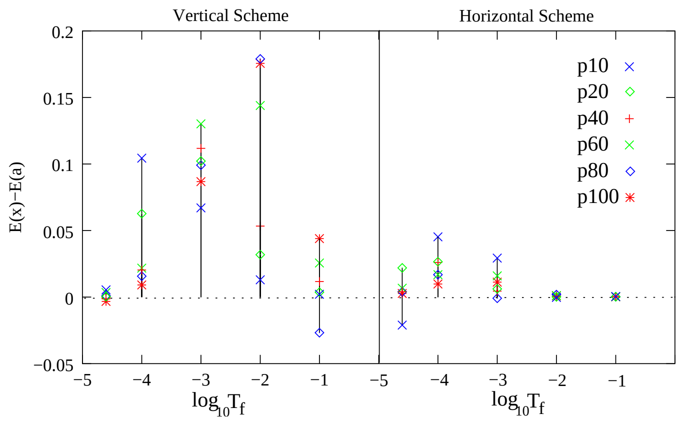


Fig. 2.4 The parallel efficiency differences on System X and Anantham (denoted "E(x)-E(a)") corresponding to the experimental results in Figure 2.3.

respectively. In the horizontal scheme, all the cases perform less efficiently than in the vertical scheme, especially when a large number of processors are used. On Anantham, similar results were produced as observed in Figure 2.4, which plots the differences of the parallel efficiency for both vertical and horizontal schemes on System X and Anantham. For most cases, System X gives a higher or equal efficiency. The next subsection studies the system-dependent parameters that may cause the performance variance on different systems.

2.2 Network Characteristics

Network connections are affected by many factors and characterized mostly by the classical metrics such as point-to-point latency, broadcast latency, and bandwidth. Here empirical studies of the network performance are presented for the point-to-point round trip cost T_{cp}. Although the one-way broadcast cost T_{ca} is also recognized as a major network characteristic (see, e.g., Wu et al. (2005)), its detailed modeling is beyond the scope of this paper, which contains only a few observations regarding T_{ca}.

The OSU benchmark library (Panda et al. 2006) provided in the MVAPICH software distribution was used to measure the MPI-level point-to-point and broadcast latencies. Small simulation programs were also used to quantize the other delays (i.e., host overhead, bottleneck, and network contention) involved in the messaging via MPI. Based on the benchmark and simulation results, both network characteristics T_{cp} and T_{ca} may be modeled as linear functions of the message size and the number of processors. For all the experiments in this section, the timing results were measured in microseconds. For small messages (less than 8 Kbytes), the results were the average of 10,000 runs, before which 1,000 runs were skipped as the network warm-up period; for large messages, 110 runs were done with the first 10 runs skipped.

2.2.1 Round Trip Cost

T_{cp}, the message point-to-point round trip cost, can be construed as the time for a master to receive from and reply to a worker. All the point-to-point messages for a given problem have the same size, so the transmission time for these messages over the network theoretically is the same.

Figure 2.5 shows the latency benchmark results between two processors on System X and Anantham as the message size S_m grows up to 2048 bytes. The two processors are either "coupled" on the same node or "decoupled" on two different nodes. The latency benchmark is a typical ping-pong test, where the sender calls MPI_Send to send a certain size message to the receiver and waits for a reply; the receiver

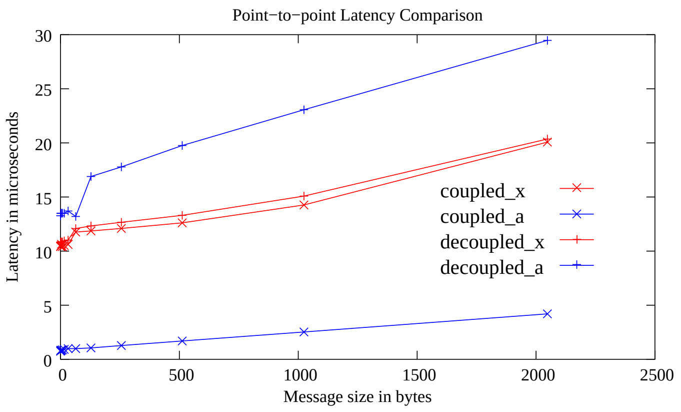
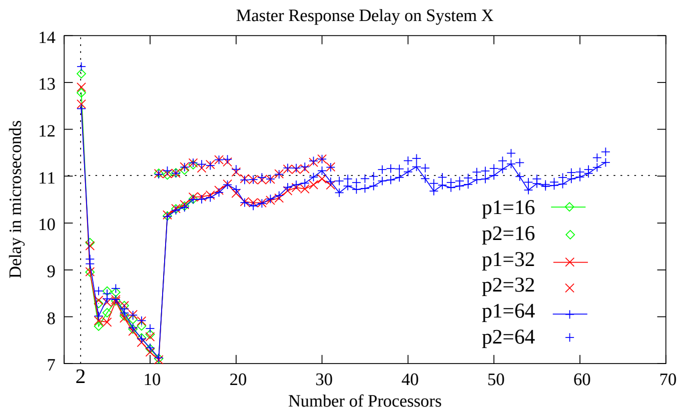


Fig. 2.5 Comparison of point-to-point latency test results. The processors are either coupled or decoupled on System X ("_x") and Anantham ("_a").

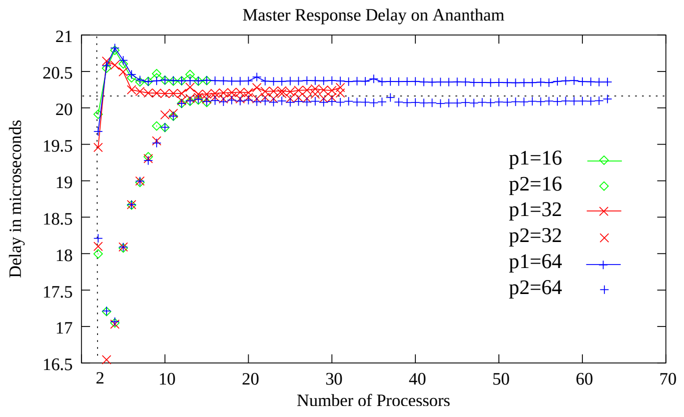
calls MPI_Recv to receive the message and sends it back as the reply. The average latency results shown in Figure 2.5 are the one-way message transmission delays.

The timing results show that using decoupled processors yields a much higher latency on Anantham than on System X. Furthermore, using coupled processors reduces the latency significantly on Anantham but causes almost no change on System X. Observe that the Myrinet intra-node latency is lower than that of VAPI for InfiniBand, which has been also pointed out in Liu et al. (2004a). The reason behind such poor performance of MVAPICH is that its particular Mac OS X implementation used on System X takes no advantage of the shared memory available to the coupled processors, and all the communications may involve the network interface on a given node. In fact, the disregard or poor usage of the shared memory may actually *increase* communication latencies as has been shown, e.g., in Chen et al. (2003). On the other hand, the MPI implementation over Myrinet is more mature now and has been well tuned to exploit the shared memory access provided by the Linux operating system.

On System X, InfiniBand is still an attractive communication medium compared with using the Gigabit Ethernet over TCP/IP, because InfiniBand delivers much better communication performance for both decoupled and coupled processors. For instance, with the message size $S_m = 1024$ bytes, $T_{cp} = 37.58$ and 225.19 when MPICH-2 is used between two coupled (over the shared memory) and decoupled processors (over the Gigabit Ethernet), respectively.



(a) System X



(b) Anantham

Fig. 2.6 Comparison of master response delay using $p = 16$, 32, and 64 processors on System X and Anantham. "pi" corresponds to i processors per node.

Conversely, $T_{cp} = 14.26\ \mu s$ when MVAPICH is used via VAPI/InfiniBand instead.

When a large number of processors are used, T_{cp} may also include a certain amount of response delay due to the host buffer latency, bottleneck effect, and network contention at the receiver. Such a delay is called "master response delay" because it is more likely to happen at a master that handles the messages exchanged with workers and other masters than at a worker that only communicates to one master at a time. A small MPI program was written to simulate the same communication pattern between a master processor and a group of processors, either as other masters or workers that communicate with this master simultaneously. With p processors, the master processor repeats the receive-and-reply of a small handshaking message ($S_m = 112$ bytes, a typical message size for a 10-dimensional problem with $N_b = 1$ function evaluation per task) 11,000 times initiated by $p - 2$ other processors from within the total $p > 2$ processors. The first 1,000 handshakes

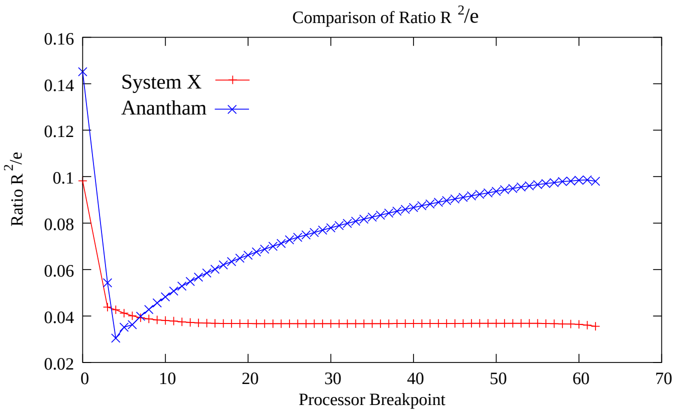
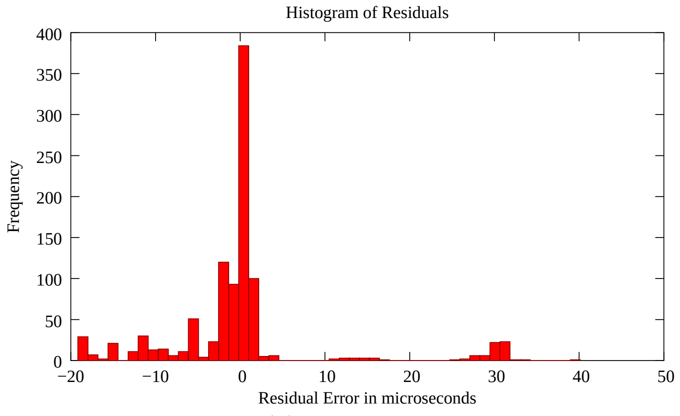


Fig. 2.7 Comparison of the ratio R^2/e for different breakpoints of the piecewise linear model of T_{cp} on System X and Anantham.

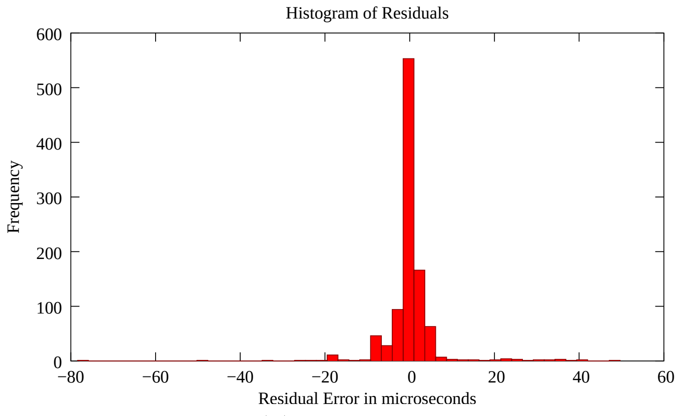
are considered as the network warm-up period as in the other experiments, thus not counted in the timing results. Six groups of simulation results shown in Figure 2.6 are the average timing results with $p = 16$, 32, and 64 processors either coupled or decoupled on System X and Anantham. The x-axis refers to the number of processors within the $p-2$ workers/masters that are communicating with the specified master.

On both systems, the curves for different numbers p are overlapped very well. Similarly to the latency benchmark results, using coupled processors reduces the master response delay significantly on Anantham especially for small numbers of processors, but slightly increases the delay on System X. On System X, no matter whether the processors are coupled or decoupled, the delay drops dramatically at the beginning until $p = 10$ (coupled) or $p = 11$ (decoupled), then suddenly reaches up to a saturated level $\approx 11.0\ \mu s$. On Anantham, the delay using coupled processors drops slightly at the beginning, then increases significantly until $p = 10$, from which point it starts leveling off at the saturated level $\approx 20.0\ \mu s$. If using decoupled processors instead, the delay increases slightly at the beginning from $p = 2$ to $p = 4$, then drops to a level $\approx 20.3\ \mu s$. The narrower bandwidth on Anantham yields a longer delay because the network contention occurs when a large number of processors are communicating to each other simultaneously. In the following experiments, coupled processors are used primarily because the cluster is utilized more efficiently (when decoupled processors are used, 50% of the processors are idle). Secondarily, it makes a minor difference on System X and even reduces the latency on Anantham.

The above experiments indicate that the T_{cp} between a master and a group of processors is affected by two factors: (1) the message size S_m and (2) the



(a) System X



(b) Anantham

Fig. 2.8 Histogram of residual errors of the linear regression models for T_{cp} on System X and Anantham.

number of processors p that communicate with the master. T_{cp} increases as S_m grows, but varies when p is small and levels off at a certain value when p is sufficiently large depending on the computing system. Therefore, T_{cp} is modeled as a multivariate function of S_m and p as

$$T_{cp}(p, S_m) = T_L(S_m) + T_d(p),$$

where $T_L(S_m)$ is defined as the point-to-point latency function in terms of S_m without the response delay and $T_d(p)$ as the pure response delay function in terms of p. In Figure 2.5, the data points for decoupled processors essentially depict $T_{cp}(S_m, p)$ as the point-to-point latency $T_L(S_m)$ with a response delay $T_d(p = 2)$, so $T_L(S_m)$ is fairly likely a linear function of S_m. In addition, the data points for coupled processors in Figure 2.6 show that the delay function $T_d(p > 2)$ with $T_L(S_m = 112\text{bytes})$ may be a piecewise linear function with one breakpoint around $p = 10$, which disconnects the two pieces on System X but connects them on Anantham. The optimal breakpoints of a piecewise linear model can be determined systematically as in Vieth (1989). For the present work, a series of piecewise linear models of T_{cp} were fitted using the statistics package R (Dalgaard 2002) at every possible breakpoint that forms either a single linear model or a two-piece piecewise linear model, each piece containing at least two different p numbers from the experimental data. For each piecewise linear model, define the ratio of the average correlation coefficient $R^2 = (R_1^2 + R_2^2)/2$ to the sum of the residual standard errors $e = e_1 + e_2$ as the objective function to be maximized to find the optimal breakpoint. Clearly, the ratio is simply R_1^2/e_1 for the single linear model case. When either R_1 or R_2 is below 0.5, the model is considered invalid and the objective function value is set to zero. The shortest point-to-point message is 48 bytes in the present work, so the data points were collected with message size $S_m = 2^i$ bytes ($i = 6, \ldots, 14$) using $p = 64$ coupled processors on System X and Anantham.

For each system, Figure 2.7 shows that the single linear model maximizes the ratio of R^2/e, thus a single linear model is used for T_{cp} on both systems. Table 2.1 lists the model formula, coefficients, residual standard errors e, and correlation coefficient R^2.

Table 2.1 Model formula, coefficients, residual standard errors e, and correlation coefficient R^2 for T_{cp} on System X (X) and Anantham (A).

#	$T_{cp}(S_m, p)$ Linear Models
X	$9.917 + 0.007S_m + 0.001p$
	$e = 9.435,\ R^2 = 0.926$
A	$18.76 + 0.007S_m - 0.017p$
	$e = 6.587,\ R^2 = 0.956$

The values e and R^2 are very reasonable for both systems. The model coefficients suggest that T_{cp} is affected only slightly by S_m and p. Ignoring S_m and p, T_{cp} is about twice smaller on System X than on Anantham. The histograms of residual errors are plotted in Figure 2.8 using 50 bins for the total of 1,008 data points. The residual errors fall in a narrower range $[-20, 50]$ on System X than the range $[-80, 60]$ on Anantham. However, residuals are skewed with more data points located away from zero on System X than on Anantham, where residual errors follow an approximate normal distribution. Because System X has 2,200 processors, the maximum impact of the term $0.001p$ is 2.2 μs. Also, the maximum impact of $-0.017p$ on Anantham is 6.8 μs. Therefore, the impact of p on T_{cp} is negligible on both systems. Since the factor S_m has a large range of values depending on the problem dimension, the number of function evaluations per node, and the scheme configuration, S_m is kept in the final model of T_{cp}:

$$T_{cp}(S_m) = 0.007 S_m + L_{cp}, \qquad (2.1)$$

where $L_{cp} = 9.917$ on System X and 18.76 on Anantham.

In the DIRECT algorithm, all the collective communications may be modeled using broadcast operations. For example, the barrier occurring after all the masters prepare local convex hull boxes can be treated as an all-to-one gather followed by an one-to-all broadcast. Similarly to the point-to-point latency, T_{ca} is a linear function of the message size S_m and the number of processors p, i.e., $T_{ca}(S_m, p)$. It has been observed (Vadhlyar et al. 2004), however, that its dependence on p may be affected by such factors as the broadcast algorithm implemented on a particular number of processors and network topology and by switching modes of this algorithm triggered for certain processor numbers.

3 Scalability Analysis

Scalability is the capacity to increase speedup in proportion to the number of processors p for a parallel system, including a parallel implementation of an algorithm and a particular execution environment. The crucial step is to identify the overhead as a function of problem scale and system scale. The scalability is evaluated with a realistic tool—the isoefficiency function (Kumar et al. 1994) based on the characterized communication patterns and overhead sources.

Kumar et al. (1994) point out that if the parallel efficiency can be maintained constant as the total work grows at least linearly with p, then the parallel implementation is scalable. The authors have previously used the isoefficiency function to analyze the performance of the restarted generalized minimum residual (GMRES) method (Saad 2003) for the iterative solution of large scale sparse linear systems (Sosonkina et al. 2002). Decker and Krandick (2001) have generalized the concept of isoefficiency function for use in search algorithms, and showed its application in the analysis of a search algorithm that computes isolating intervals for polynomial real roots. In the present paper, the overhead terms are analyzed in great detail in order to identify the ones with significant weight under different circumstances. The derivation of the isoefficiency function is shown only for the vertical ($n = 1,\ k > 1$) scheme. Since one may proceed in a similar manner to develop the isoefficiency function for the horizontal ($n > 1,\ k = 0$) scheme, its derivation is omitted here while major findings are only summarized.

3.1 Vertical Scheme Scalability

In the vertical scheme, three sources of overhead are present. The first is the communication involved in distributing tasks to remote workers. The second is the idleness due to task imbalance among workers due to the data dependency of algorithm steps. Part 2 assumes $N_b = 1$ (one function evaluation per task), since Part 1 has proved that choice for N_b minimizes the data dependency overhead, especially for expensive objective functions. The third is the idle time when workers wait for DIVISION and SELECTION to be finished at the master. Since DIVISION and SELECTION are necessary steps of DIRECT, and the cost does not vary with k, the third source of overhead is considered as a constant for a particular problem with a specified stopping condition.

Let W be the total number of function evaluations over $I_{\max}$ iterations,

$$W = \sum_{i=1}^{I_{\max}} F_i, \qquad (3.1)$$

where F_i denotes all the function evaluation tasks at iteration i. The total overhead for k workers is

$$T_o(W, k) = k\ T_k - W\ T_f, \qquad (3.2)$$

where T_k is the parallel execution time with k workers. Define the parallel efficiency E_p as

$$E_p = \frac{W\ T_f}{k\ T_k}. \qquad (3.3)$$

If E_p can be maintained constant as W grows at least linearly with k, then the vertical scheme is scalable. The isoefficiency function is derived from Equation (3.2) and (3.3) as

$$W = \frac{E_p}{1 - E_p}\ (T_o(W, k)/T_f). \qquad (3.4)$$

It describes the total work W in terms of T_o, the total overhead function given a desired E_p and the constant objective function cost T_f. To find if W is a linear function of k, $T_o(W, k)$ as a function of W needs to be extended by accumulating over all $I_{\max}$ iterations. All the costs, except for the function evaluation, are included in $T_o(W, k)$. So it has three components: T^1_{oc} (communication overhead), T_{od} (data dependency overhead), and a constant T_{os} (the DIVISION and SELECTION cost).

The interaction between masters and workers is described in full detail by He et al. (2006). Here, the communication phases are only outlined.

(1) Each of k workers sends a nonblocking request to a randomly selected master.

(2) For each worker that has sent a nonblocking request or that is in the queue, a master sends a task, if any. If there are no more tasks, the master sends a "no point" message, or if there are no more iterations, sends a "all done" message.

(3) The workers who received tasks send values back to the master. Those who received the "no point" notice check with other masters, who may have tasks, and if none have tasks, send blocking requests and wait. Those who received the "all done" notice terminate. For each value received, a master sends a task, if any; if there are no more tasks, a master sends a "no point" notice; for each blocking request, a master queues up the worker.

Only one master distributes tasks, so if it runs out of tasks at the end of an iteration, all workers will block waiting in its queue. So, Phase (1) happens only at the first iteration, because the workers are all in the queue of the master from the second iteration onward. Phases (2) and (3) are repeated at every iteration. To sum up the communication overhead, consider the message types and the frequency of these messages exchanged between a master and a group of workers:

(a) $T_1 = k\ T_{cp}/2$ (k workers send nonblocking requests),

(b) $T_2 = F_i\ T_{cp}/2$ (workers receive F_i tasks),

(c) $T_3 = F_i\ T_{cp}/2 + k\ T_{cp}$ (the master receives all function values and sends "no point" messages at end of each iteration to k workers; all workers then send blocking requests to the master),

(d) $T_4 = k\ T_{cp}/2$ (workers receive "all done" messages from the master at the last iteration).

The communication overhead over all iterations is

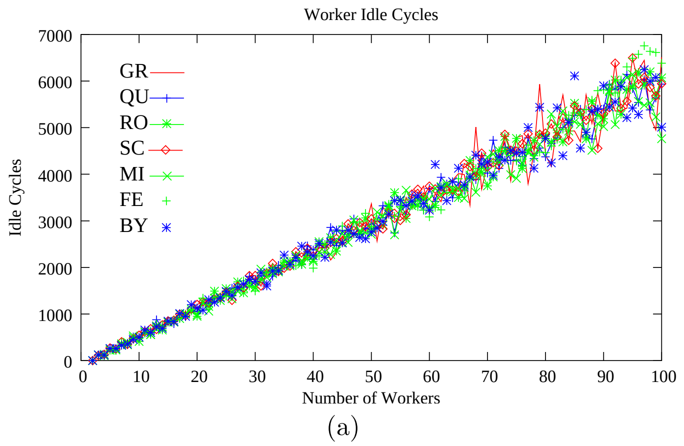
$$T^1_{oc} = T_1 + \sum_{i=1}^{I_{\max}} T_2 + \sum_{i=1}^{I_{\max}} T_3 + T_4.$$

Represent it as a function of W (Equation 3.1) and k with $T_{cp}(S_m)$ (Equation 2.1):

$$\begin{aligned} T^1_{oc}(W,k) &= (k(1+I_{\max}) + W)T_{cp}(S_m) \\ &= (0.007S_m + L_{cp})W \qquad (3.5) \\ &\quad + (1+I_{\max})(0.007S_m + L_{cp})k. \end{aligned}$$

For point-to-point messages, the message size S_m (in bytes) depends on the problem dimension N_d:

$$S_m = (N_d + 4)\ 8. \qquad (3.6)$$



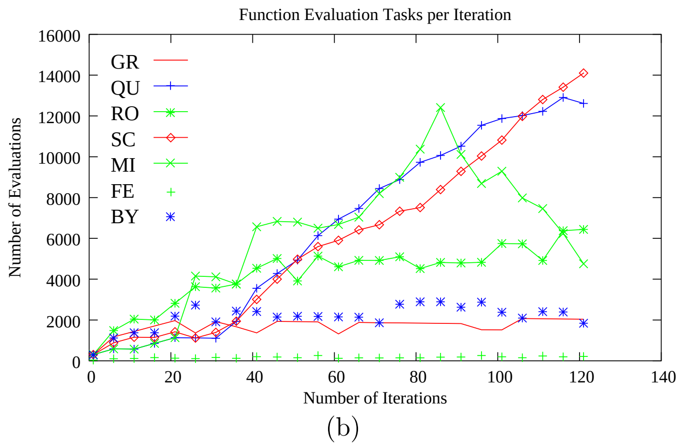


Fig. 3.1 All the worker idle cycles, shown in (a), are calculated using $k = 2, \ldots, 100$ workers and the function evaluations are shown in (b) on all the test problems on System X. $I_{\max} = 122$.

Therefore, $T_{cp}(S_m)$ is a constant for the same problem on a particular system. Because T^1_{oc} depends linearly on k, it grows linearly as $\mathcal{O}(k)$ for a fixed size problem specified by $I_{\max}$ and W. On System X, the twice smaller L_{cp} (Table 2.1) gives a slower growth of T^1_{oc} in terms of k than that on Anantham.

The second source of overhead—data dependency—is as important as the communication. It causes task imbalance among workers. At iteration i, if some workers obtain no tasks, when the number of tasks F_i is less than the number of workers k, or obtain fewer tasks than others, idle cycles will appear, each of which lasts about T_f. The number X_i of such cycles can be derived from the equation

$$\delta_{i+}\ (k - X_i) + \delta_{i-}\ (X_i) = F_i,$$

where $\delta_{i+} = \lceil F_i/k \rceil$ and $\delta_{i-} = \lfloor F_i/k \rfloor$ are the workers' task shares defined in the workload model of Part 1. Because $\delta_{i+} - \delta_{i-} = 1$,

$$X_i = \frac{\delta_{i+}\ k - F_i}{\delta_{i+} - \delta_{i-}} = \delta_{i+}\ k - F_i,$$

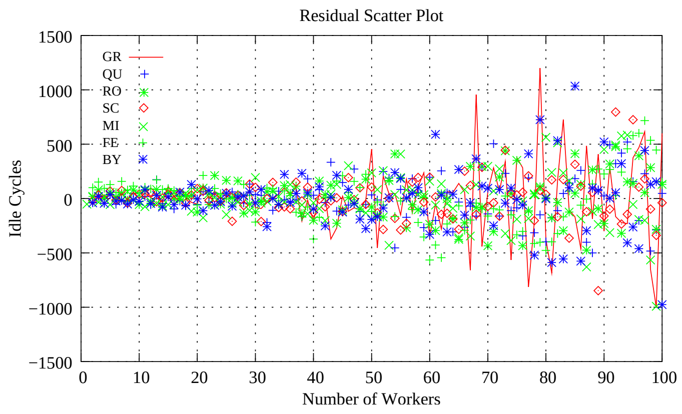


Fig. 3.2 Scatter plot of residuals $\delta_p(k)$ for test problems on System X.

yielding the total data dependency overhead

$$
\begin{aligned}
T_{od} &= \sum_{i=1}^{I_{\max}} X_i T_f \\
&= \sum_{i=1}^{I_{\max}} \left(\left\lceil \frac{F_i}{k} \right\rceil k - F_i \right) T_f .
\end{aligned} \tag{3.7}
$$

Observe that T_{od} becomes zero when F_i is a multiple of k. Figure 3.1(a) shows the results of $\sum_{i=1}^{I_{\max}} (X_{i,j})$, which is the total number of worker idle cycles over $I_{\max}$ iterations using $j = 2, \ldots, k$ workers. The test problems include the artificial functions described in the Appendix, cell cycle parameter estimation for frog egg extracts (FE) by Zwolak et al. (2005), and cell cycle parameter estimation for budding yeast (BY) by Panning et al. (2006) in systems biology.

Interestingly, each curve for the total idle cycles presents a linear function of k with a fluctuation, which falls in a larger range when more workers are used. Note that the wall clock duration of data dependency overhead is actually

$$\sum_{i=1}^{I_{\max}} \mathrm{sgn}(X_{i,j})\, T_f \geq \sum_{i=1}^{I_{\max}} (X_{i,j}) T_f / k,$$

and this average, rather than total, overhead per worker is considered for experimental verification here.

The growth rate of the number of idle cycles was estimated by the same linear regression tool used in Section 2. Although the QU, SC, and MI problems generate more function evaluation tasks per iteration than other problems do (Figure 3.1(b)), the growth rates are very close to each other for these test problems as shown in Table 3.1. Hence, T_{od} grows approximately linearly with k with a fluctuation that can be described by the residual error function $\delta_p(k)$ (Figure 3.2).

Table 3.1 Comparison of the fitted linear growth rate θ, the residual standard error e, and the correlation coefficient R^2 for the fitted linear model of the worker idle cycles for test problems.

#	θ	e	R^2
GR	61.093	327.6	0.966
QU	60.724	202.5	0.986
RO	61.816	194.1	0.988
SC	61.879	200.1	0.987
MI	59.186	241.2	0.980
FE	64.613	234.9	0.984
BY	60.658	269.7	0.976

Table 3.2 Comparison of C_m, T^x_{oc}/k (on System X), T^a_{oc}/k (on Anantham), and T_{od}/k (in seconds) for all test problems using $k = 99$ workers. $I_{\max} = 122$ with the first 12 iterations skipped for the overhead computation. $N_b = 1$. $N_d = 150$ and $T_f = 0.1$ for artificial functions, $N_d = 16$ and $T_f = 2.66$ for FE, and $N_d = 143$ and $T_f = 11.02$ for BY.

#	C_m	T^x_{oc}/k	T^a_{oc}/k	T_{od}/k
GR	413332	0.039	0.057	4.450
QU	1598908	0.149	0.221	5.371
RO_x	1021480	0.097	–	5.101
RO_a	1017544	–	0.141	5.540
SC	1487836	0.139	0.206	5.368
MI	1508052	0.141	0.209	4.957
FE	59604	–	0.006	156.860
BY	528468	0.048	–	616.790

Table 3.2 compares the number of one-way messages C_m, the theoretical communication overheads per worker T^x_{oc}/k and T^a_{oc}/k on System X and Anantham, respectively, and the worker average data dependency overhead T_{od}/k for all test problems with $k = 99$ workers. On both System X and Anantham, $I_{\max} = 122$ iterations were run, with the first 12 iterations skipped for the overhead computation to eliminate startup effects, so the number of one-way messages is $C_m = 2\big(k(I_{\max} - 12) + \sum_{i=13}^{I_{\max}} F_i\big)$. Both T^x_{oc} and T^a_{oc} were estimated using the formula for T^1_{oc} in Equation 3.5 given C_m, F_i (the number of function evaluation tasks per iteration collected from the runs), the linear models of T_{cp} in Equation 2.1, and S_m in Equation 3.6. The worker average time T_{od}/k is calculated with Equation 3.7 given the collected data for F_i using 99 workers. For the BY problem the data were collected only on System X and, for the FE problem, only on Anantham, because the versions

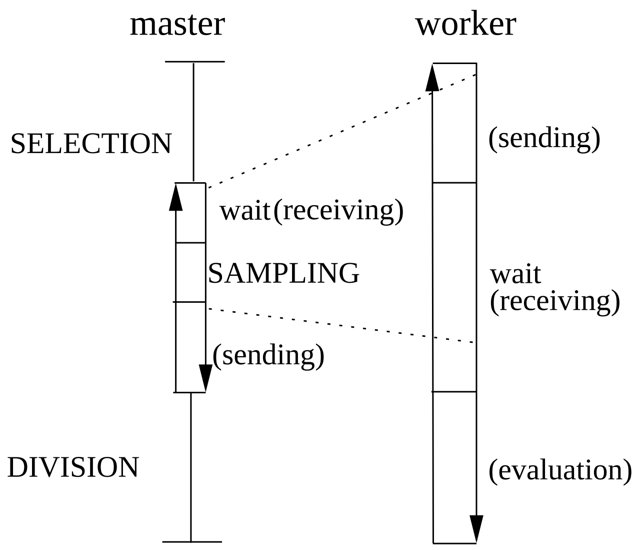


Fig. 3.3 Overhead measurement of sampling loop on a master and a worker.

of these applications used require specific execution environments. A reasonably large objective function cost $T_f = 0.1$ is assumed for artificial functions while $T_f \approx 2.66$ for the FE problem and $T_f \approx 11.02$ for the BY problem were measured with experiments. In addition, the RO problem generates different numbers of function evaluation tasks on System X and Anantham because the 64-bit registers on System X and the 80-bit registers on Anantham give slightly different computation precision. During the SELECTION step, the RO problem yields different results depending on the floating point hardware. Therefore, the estimation of the communication and data dependency overhead differs on these two systems.

When T_f is taken into account, the data dependency overhead becomes more significant for expensive functions, such as FE and BY, than for cheap objective functions. Although System X has lower communication overhead, the common overhead T_{od} still dominates because $T_f \geq 0.1$ is sufficiently large. On the other hand, if $T_f \leq 10^{-3}$, T_{od} would become less dominant than the communication overhead. This explains the observed values of the parallel efficiency as T_f changes (Section 2.1).

To verify the above theoretical analysis, T_{sk} (total sampling time with k workers) and T_e (total evaluation time) were measured by the phases depicted in Figure 3.3. T_{sk} is the same for the master and the worker. T^1_{oc} is measured by counting messages instead of directly timing the duration because it is hard to separate sending and receiving costs from SAMPLING. However, it is expected that the estimated overhead $T_{ot} = (T^1_{oc} + T_{od})/k$ (the sum of the theoretical communication and data dependency overhead per worker in Table 3.2) should be approximately equal to the experimental result $T_{oe} = T_{sk} - T_e/k$ as listed in Table 3.3.

Table 3.3 The experimental results of the SAMPLING cost T_{sk} and the evaluation cost T_e for all test problems on System X ('_x') and Anantham ('_a'). The combined experimental overhead $T_{oe} = T_{sk} - T_e/k$ is compared to the estimated overhead $T_{ot} = (T^1_{oc} + T_{od})/k$.

#	T_{sk}	T_e	T_{oe}	C_m	T_{ot}
GR_x	202.44	19578.0	4.68	413332	4.50
GR_a	203.28	19578.0	5.52	413332	4.51
QU_x	802.48	78857.0	5.94	1598908	5.52
QU_a	803.73	78858.0	7.18	1598908	5.59
RO_x	510.46	49987.0	5.54	1021480	5.20
RO_a	509.65	49789.0	6.73	1017544	5.68
SC_x	746.46	73306.0	5.99	1487836	5.51
SC_a	748.85	73304.0	8.41	1487836	5.57
MI_x	756.44	74316.0	5.77	1508052	5.10
MI_a	757.83	74316.0	7.16	1508052	5.17
FE_a	696.25	50608.0	185.06	59604	156.87
BY_x	29179.0	2788790	1009.40	528468	616.84

The SAMPLING cost T_{sk} is approximately the same on both systems. Note that the combined experimental overhead T_{oe} is greater than the theoretical estimation T_{ot}. Moreover, the scaled difference $(T_{oe} - T_{ot})/T_{sk}$ is larger on Anantham than that on System X. Nevertheless, C_m matches exactly (Table 3.2) the experimental results. One possibility is that the communication overhead could be underestimated because the small simulation program that generated the data points for modeling T_{cp} may incur less overhead than the full DIRECT optimization program does with a large amount of memory allocated. There might also exist another system dependent source of overhead other than the communication, such as memory access inefficiency, that may increase the overhead T_{oe} on Anantham.

With all the overhead terms analyzed, the isoefficiency function (Equation 3.4) for the vertical scheme may be expressed as

$$W = \frac{E_p}{1 - E_p} \frac{T^1_{oc}(W,k) + T_{od}(k) + T_{os}}{T_f},$$

where $T^1_{oc}(W,k)$ is a linear function of k, $T_{od}(k)$ is an approximately linear function of k, and T_{os} is a constant for a particular problem with specified $I_{\max}$ and resulting W. Therefore, W becomes

$$W = \frac{E_p\big(T_{cp}k(1 + I_{\max}) + T_{od}(k) + T_{os}\big)/T_f}{1 - E_p - E_pT_{cp}/T_f}.$$

Firstly, observe that W grows linearly with k, meaning that the vertical scheme is scalable. Secondly, a larger ratio of T_{cp}/T_f on a particular system requires a faster growth of W to maintain the same E_p as more workers are used. Lastly, a larger T_f gives a better scalability.

3.2 Horizontal Scheme Scalability

When the objective function is computationally cheap, the horizontal scheme is more suitable than the vertical scheme as discussed in Section 2.1. Thus, $T_f \approx$ 2.6E-05 was considered for all the artificial test problems in the scalability analysis of the horizontal scheme with multiple masters ($n > 1$) and no workers ($k = 0$) in a single domain.

The overhead includes T_{oc}^2, the communication overhead involved in the global convex hull computation during the parallel SELECTION, and T'_{od}, the data dependency overhead among masters:

$$T'_o(H, W, n) = T_{oc}^2(H, n) + T'_{od}(H, W, n), \qquad (3.8)$$

where H stands for the total work in the convex hull computation. Part 1 has demonstrated that the horizontal scheme has load balancing of function evaluation tasks significantly worse than that of the vertical scheme. The analysis of $T'_o(H, W, n)$ reveals that it is mostly caused by the data dependency overhead T'_{od}. Specifically, the data dependency overhead is dominant for large data sets and is very problem dependent. On the other hand, the communication overhead dominates when T_f is cheap. Thus, the differences in communication subsystems, such as those on System X and Anantham, are not pronounced when function evaluations are expensive. In general, for the horizontal scheme, the isoefficiency function is linear but with the communication overhead growing faster than in the vertical scheme.

4 CONCLUSION

The performance impact of three parallel system parameters—T_f (the objective function cost), T_{cp} (the point-to-point round trip cost), and T_{ca} (the broadcast cost)—on pDIRECT_II, a massively parallel implementation of the DIRECT global optimization algorithm, has been explored in Part 2 of the present work. The scalability of basic parallel schemes—vertical and horizontal—were further analyzed here using linear models and isoefficiency functions.

On System X and Anantham, an empirical study with T_f varying in small time steps was conducted for both schemes to compare the resulting parallel performance under two types of computing environments differing in both hardware (i.e., CPU clock frequency, system architecture, and interconnect network) and software (i.e., operating systems, MPI supporting packages, and compilers). System X outperforms Anantham in all cases, except for a few cases under horizontal schemes when T_f is larger than 2.5E-04, the performance boundary value for the vertical and horizontal scheme on System X. A detailed analysis of overhead and scalability further supports the experimental observations.

Both theoretical and experimental studies have demonstrated that different system settings only matter when T_f is small. For expensive objective functions, the dominant impact on the parallel performance comes from the problem-dependent factors such as data dependency. However, higher network bandwidth and greater node availability on System X certainly provide a better computing environment for many large scale optimization applications.

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

Boden, N.J., Cohen, D., Felderman, R.E., Kulawik, A.E., Seizovic, J.N. 1995. Myrinet: a gigabit-per-second local area network. IEEE Micro, Vol. 15, No. 1, pp. 29–36.

Chen, X. and Turner, D. 2003. Efficient message-passing within SMP systems. Recent advances in Parallel Virtual Machine and Message Passing Interface, Lecture Notes in Computer Science, Springer, Vol. 2840, pp. 286–293.

Dalgaard, P.. 2002. Introductory Statistics with R. Springer, ISBN 0-387-95475-9.

Decker, T. and Krandick, W. 2001. Isoefficiency and the parallel Descartes method. Proceedings of Dagstuhl Seminar "Symbolic Algebraic Methods and Verification Methods", Lecture Notes in Computer Science. Springer, pp. 55–69.

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.

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.

Kumar, V. and Gupta, A. 1994. Analyzing scalability of parallel algorithms and architectures. J. of Parallel and Distributed Computing, Vol. 22, No. 3, pp. 379–391.

Liu, J., Chandrasekaran, B., Yu, W., Wu, J., Buntinas, D., Kini, S., Panda, D.K. 2004a. Microbenchmark performance comparison of high-speed cluster interconnects. IEEE Micro, Vol. 24, No. 1, pp. 42–51.

Liu, J., Wu, J., Kini, S.P., Wyckoff, P., and Panda, D.K. 2004b. High performance RDMA-based MPI implementation over InfiniBand. International J. of Parallel Programming, Vol. 32, No. 3, pp. 167–198.

Panda D. K. and MVAPICH team. 2006. MVAPICH 0.9.8 User and Tuning Guide. Network-based Computing Laboratory, Department of Computer Science and Engineering, the Ohio State University.

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.

Saad, Y. Iterative Methods for Sparse Linear Systems. 2003. SIAM, Philadelphia, PA, 2nd edition.

Sosonkina, M., Allison D.C.S., and Watson, L.T. 2002. Scalability analysis of parallel GMRES implementations. Parallel Algorithms and Applications, Vol. 17 pp. 285–299.

Vadhlyar, S.S., Fagg, G.E., and Dongarra, J.J. 2004. Towards an accurate model for collective communications. International J. of High Performance Computing Applications, Vol. 18, No. 1, pp. 159–167.

Vieth, E. 1989. Fitting piecewise linear regression functions to biological responses. Journal of applied physiology, Vol. 67, No. 1, pp. 390–396.

Wu, M.S., Kendall, R., Wright, K., and Zhang Z. 2005. Performance modeling and tuning strategies of mixed mode collective communications. SC '05: Proceedings of the 2005 ACM/IEEE conference on Supercomputing, pp. 45–55.

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

APPENDIX Test functions.

Table 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 $\bar{x} \in \{0,\pi\}^N$