Journal of Parallel and Distributed Computing **62**, 173–192 (2002)
doi:10.1006/jpdc.2001.1773, available online at http://www.idealibrary.com on IDEAL®

Scalability versus Execution Time in Scalable Systems[1]

Xian-He Sun

Department of Computer Science, Illinois Institute of Technology, Chicago, Illinois 60616
E-mail: sun@cs.iit.edu

Received July 28, 1997; revised April 13, 2001; accepted May 7, 2001

Parallel programming is elusive. The relative performance of different parallel implementations varies with machine architecture, system and problem size. How to compare different implementations over a wide range of machine architectures and problem sizes has not been well addressed due to its difficulty. Scalability has been proposed in recent years to reveal scaling properties of parallel algorithms and machines. In this paper, the relation between scalability and execution time is carefully studied. The concepts of crossing point analysis and range comparison are introduced. Crossing point analysis finds slow/fast performance crossing points of parallel algorithms and machines. Range comparison compares performance over a wide range of ensemble and problem size via scalability and crossing point analysis. Three algorithms from scientific computing are implemented on an Intel Paragon and an IBM SP2 parallel computer. Experimental and theoretical results show how the combination of scalability, crossing point analysis, and range comparison provides a practical solution for scalable performance evaluation and prediction. While our testings are conducted on homogeneous parallel computers, the proposed methodology applies to heterogeneous and network computing as well. © 2002 Elsevier Science (USA)

Key Words: parallel processing; performance evaluation and measurement; scalability; execution time; numerical algorithms.

1. INTRODUCTION

In sequential computing, algorithms are well characterized in terms of operation count and memory requirement. Assuming sufficient memory is available, the execution time of a sequential algorithm is proportional to the work performed. While problem size[2] and memory requirements remain essential factors in parallel

[1] This research was supported in part by NSF under Grants ASC-9720215 and CCR-9972251, and by ONR under Navoceano PET/Logicon.

[2] Some authors refer to problem size as the parameter that determines the work, for instance, the order of matrices. In this paper, problem size refers to the work to be performed, and we will use problem size and work alternatively.

0743-7315/02 $35.00
© 2002 Elsevier Science (USA)
All rights reserved.

computing, communication overhead and load balance complicate determining parallel execution time. In general, load balance over processors decreases with the ensemble size (the number of processors available) while communication overhead increases with ensemble size; both may reduce performance considerably. More importantly, this "decrease" in load balancing and "increase" in communication vary with algorithms, machines, and algorithm–machine combinations (AMCs). They are functions of system ensemble size and problem size. An initially fast parallel implementation may slow when system and problem size increase. A superior algorithm may only be superior for a given architecture and only over a limited range of system and problem sizes. Finding the range of superiority is inherently difficult. Superiority/inferiority is determined in terms of execution time, whereas execution time is set at a given parallel platform and at a specified system and problem size. The lack of a performance evaluation mechanism for range comparison is a current barrier of parallel/distributed programming.

Scalability is a property that exhibits performance linearly proportional to the number of processors employed. During the past few years, scalability has received intensive attention. Depending on how the performance power is defined and measured, different scalability metrics have been proposed [1–6]. Compared with execution time, an advantage of scalability is its "dimensionless": it depends on the initial state only. Scalability has different applications. One important application of the "dimensionless" is in performance prediction, where premeasured performance data are collected, stored, and used for predicting the future performance[3]. Execution time is the ultimate measure of interest in practice. Performance prediction should be in terms of execution. Since execution time depends on problem size and there are numerous possible combinations of problem size and data distribution, execution time is hardly likely to be collected in a manageable way for performance prediction. A practical solution seems to be a combination of scalability and execution time, stored performance in terms of scalability, and use the prestored scalability to predict the performance in terms of execution time.

Integrating scalability into performance prediction requires an in-depth understanding of the relation between scalability and execution time. Scalability, however, has traditionally been studied separately. Its relation to execution time has not been well understood. In this paper, we carefully study the relation between scalability and time. Based on newly uncovered relations, the concept of range comparison is introduced. Unlike conventional execution time comparison in which performance is compared at a given parallel platform and at a specified system and problem size, range comparison compares performance over a wide range of ensemble and problem size via performance crossing point analysis. The idea of range comparison is straightforward: find the first performance superior/inferior

[3] In many case studies, the performance of the algorithm under study can be formulated in terms of machine parameters. The performance of the algorithm, therefore, can be predicted based on the performance formula. This approach of performance prediction has two shortcomings in scalable computing practice: the value of the machine parameters may vary with problem size and machine ensemble size; performance formulation is not widely available.

crossing point. Before meeting the first crossing point, over a wide range of system and problem sizes, a fast program will remain fast and a slow program will remain slow. The key question is how to find the crossing point. This question is addressed in this study. Theoretical foundation is built to find the crossing point via *isospeed scalability* [3]. Since the relation between isospeed scalability and other scalabilities has been studied [7], results presented in this paper can be extended to other scalabilities as well.

In Section 2, we present the relation between scalability and execution time. In Section 3, we introduce the concepts of crossing point analysis and range comparison. Two algorithms are also given to determine the crossing point and performance range comparison. We present three parallel algorithms in Section 4. Comparison and scalability analysis of the three algorithms are also performed. Experimental results of the three algorithms are given in Section 5 to confirm our analytical findings. Section 6 summarizes the work.

2. ISOSPEED SCALABILITY AND ITS RELATION WITH TIME: AN OVERVIEW

A goal of high-performance computing is to solve large problems fast. Considering both execution time and problem size, what we seek from parallel processing is *speed*, which is defined as work divided by time. Average speed is the achieved speed divided by the number of processors used. Average speed is a quantity that ideally would be unchanged with scaled system size. For a large class of AMCs, the average speed can be maintained by increasing the problem size. The necessary problem size increase varies with algorithm–machine combinations. This variation provides a quantitative measurement for scalability. Let W be the amount of work of an algorithm when p processors are employed in a machine, and let W' be the amount of work of the algorithm when $p' > p$ processors are employed to maintain the average speed. Following the isospeed scalability [3], the scalability from system size p to system size p' of the algorithm–machine combination is

$$\psi(p, p') = \frac{p' \cdot W}{p \cdot W'}, \tag{1}$$

where the work W' is determined by the isospeed constraint. A formula has been given in [8] to compute W',

$$W' = \frac{a \cdot p' \cdot T_o(W')}{1 - a\Delta}, \tag{2}$$

where a is the average speed, Δ is the sustained computing capacity of a single processor (reciprocal of speed), and $T_o(W')$ is the parallel processing overhead on p' processors. When parallel degradation does exist (i.e., $T'_o(W') > 0$), $a \cdot \Delta < 1$ and, therefore, Eq. (2) is traceable. $T'_o > 0$ is a necessary and sufficient condition of equation 2. When $T'_o = 0$, ideal scalability is achieved with $\psi(p, p') = 1$.

Intuitively, average speed should increase with problem size for any scalable AMC. This intuition may not be generally true due to hardware limitations. Definition 1 gives a definition of problem size scalability.

DEFINITION 1 (DATA SCALABLE). We say an algorithm–machine combination is data scalable if this algorithm–machine combination has a nonzero parallel processing overhead ($T_o(W) > 0$), and for any fixed ensemble size $p > 1$ its average speed increases with its problem size.

Theorem 1 gives a basic relation between scalability and execution time: a better scalability leads to a better (scaled) execution time. Scalability is supposed to be in favor of execution time. Theorem 1 confirms the legitimacy of the isospeed scalability. More importantly, it gives a quantitative measurement of the relation between scalability and execution time. Theorem 1 is very general. Let the constant α equal 1 and AMC 1 and 2 have the same scalability, or let α equal 1 and AMC 1 and 2 have the same execution time, etc.; then we have different corollaries. Using the fact that Corollary 2 holds for both scaled problem size W' and W^*, Corollary 1 gives a relation between the scaled execution time of AMC 1 and AMC 2. Initial performance differences can be presented in terms of execution time, or in terms of problem size needed for obtaining the desired average speed. Theorem 2 shows the relation of scalability and execution time when the initial performance difference is given in terms of problem size. For the sake of brevity, only part of the corollaries are listed here. The proofs of the listed corollaries are also not given. Interested readers are refered to [9] for more information.

In Theorem 1 and throughout this paper, the scaled problem size is the scaled problem size under isospeed scalability.

THEOREM 1. *If algorithm–machine combinations* 1 *and* 2 *are data scalable and have execution time* $\alpha \cdot T(p, W)$ *and* $T(p, W)$*, respectively, at the same initial state (the same initial ensemble and problem sizes), then combination* 1 *has a higher scalability than combination* 2 *at scaled ensemble size* p' *if and only if the execution time of combination* 1 *is smaller than* α *multiplied by the execution time of combination* 2 *for solving* W' *or* W^**, where* W' *and* W^* *are the scaled problem sizes of combinations* 1 *and* 2*, respectively.*

Corollary 1 is a direct result of Theorem 1.

COROLLARY 1. *If algorithm–machine combinations* 1 *and* 2 *are data scalable and have execution time* $\alpha \cdot T(p, W)$ *and* $T(p, W)$*, respectively, at the same initial state, then the execution time of combination* 1 *is smaller than* α *multiplied by the execution time of combination* 2 *for solving* W' *at scaled ensemble size* p' *if and only if the execution time of combination* 1 *is smaller than* α *multiplied by the execution time of combination* 2 *for solving* W^**, where* W' *and* W^* *are the scaled problem sizes of combinations* 1 *and* 2 *respectively.*

When two AMCs have the same initial performance, or the same scalability, we have the following corollaries.

COROLLARY 2. *If algorithm–machine combinations* 1 *and* 2 *are data scalable and have the same performance at the same initial state, then combination* 1 *has a higher scalability than that of combination* 2 *at scaled ensemble size* p' *if and only if combination* 1 *has an execution time smaller than that of combination* 2 *for solving* W' *or* W^**, where* W' *and* W^* *are the scaled problem sizes of combination* 1 *and combination* 2 *respectively.*

COROLLARY 3. *If algorithm–machine combinations* 1 *and* 2 *are data scalable and have execution time* $\alpha \cdot T$ *and* T*, respectively, at the same initial state, then combinations* 1 *and* 2 *have the same scalability at scaled ensemble size* p' *if and only if the execution time of combination* 1 *is equal to* α *multiplied by the execution time of combination* 2 *for solving* W' *or* W^**, where* W' *and* W^* *are the scaled problem sizes of combination* 1 *and combination* 2 *respectively.*

Theorem 2 is an alternative of Theorem 1 with the initial performance difference given in terms of problem size.

THEOREM 2. *If algorithm–machine combinations* 1 *and* 2 *are data scalable and achieve the same average speed with problem size* W *and* $\alpha \cdot W$*, respectively, at the same initial ensemble size, then* α *multiplied by the scalability of combination* 1 *is greater than the scalability of combination* 2 *at scaled ensemble size* p' *if and only if combination* 1 *has an execution time smaller than that of combination* 2 *for solving* W' *or* W^**, where* W' *and* W^* *are the scaled problem sizes of combination* 1 *and combination* 2 *respectively.*

When AMC 1 and 2 have the same scalability, Theorem 2 leads to the following corollary.

COROLLARY 4. *If algorithm–machine combinations* 1 *and* 2 *are data scalable and achieve the same average speed with problem size* W *and* $\alpha \cdot W$*, respectively, at the initial ensemble size, then* α *multiplied by the scalability of combination* 1 *is the same as the scalability of combination* 2 *at scaled ensemble size* p' *if and only if combination* 1 *has the same execution time as that of combination* 2 *for solving* W' *or* W^**, where* W' *and* W^* *are the scaled problem sizes of combination* 1 *and combination* 2 *respectively.*

3. RANGE COMPARISON AND CROSSING POINT ANALYSIS

When the system ensemble size scales up, an originally faster code with smaller scalability can become slower than code that has a better scalability. The definition of crossing point is problem size-dependent. It depends on the view of scalable computing: does problem size scale up? and if so, then how? Definition 2 gives a formal definition of scaled crossing point based on isospeed scalability. The correctness of the definition is given by Theorem 3. An alternative definition of scaled crossing point in terms of parallel processing overhead can be found in [9].

DEFINITION 2 (SCALED CROSSING POINT). For any $\alpha > 1$, if algorithm–machine combinations 1 and 2 have execution time αT and T respectively at the same

initial state, then we say a scaled ensemble size p' is a scaled crossing point of combinations 1 and 2 if the ratio of the isospeed scalability of combination 1 and combination 2 is greater than α at p'.

Let AMC 1 have execution time t, scalability $\Phi(p, p')$, and scaled problem size W'. Let AMC 2 have execution time T, scalability $\Psi(p, p')$, and scaled problem size W^*. By Definition 2, p' is a scaled crossing point of AMC 1 and 2 if and only if

$$\frac{\Phi(p, p')}{\Psi(p, p')} > \alpha. \tag{3}$$

In fact, by Eq. (1), when $\Phi(p, p') \geqslant \alpha\Psi(p, p')$ we have $t(p', W') \leqslant T(p', W^*)$ [3]. Note that since $\alpha > 1$ combination 2 has a smaller execution time at the initial state, $t(p, W) > T(p, W)$. This fast/slow change in execution time illustrates the meaning of scaled crossing point. Theorem 3 confirms the correctness of Definition 2. However, scaled crossing point is not good enough for crossing point analysis. With different scalabilities, the scaled problem size W' and W^* could be different. For performance crossing point analysis, we are interested in performance crossing with the same problem size. The results given in Section 2 cannot apply directly for crossing point analysis. Theorem 4 presents a basic result for performance comparison with the same scaled problem sizes. Theorem 5 extends the result to a range of problem sizes. Theorems 4 and 5 build the theoretical foundation for crossing point analysis. The proof of Theorem 3 can be found in [9].

THEOREM 3. *If algorithm–machine combination* 1 *has an execution time larger than that of algorithm–machine combination* 2 *at the initial state, then, for any scaled ensemble size* p', p' *is a scaled crossing point if and only if combination* 1 *has a scaled execution time smaller than that of combination 2.*

Scaled crossing point is different from the crossing point where performance crosses with the same problem size. We call the latter *equal-size crossing point*, or simply *crossing point*, if the content is clear. Theorem 4 gives a relation between the scaled crossing point and equal-size crossing point.

THEOREM 4. *Assume algorithm–machine combinations* 1 *and* 2 *are data scalable and combination* 1 *has an execution time larger than that of combination* 2 *at the initial state, then the scaled ensemble size* p' *is not a scaled crossing point if and only if combination* 1 *has an execution time larger than that of combination* 2 *for solving both* W' *and* W^* *at* p', *where* W' *and* W^* *are the scaled problem sizes of combination* 1 *and combination* 2 *respectively.*

Proof. Let AMC 1 have execution time $t(p, W)$, scalability $\Phi(p, p')$, and scaled problem size W'. Let AMC 2 have execution time $T(p, W)$, scalability $\Psi(p, p')$, and scaled problem size W^*.

Proof of the "only if" condition. Since p' is not a scaled crossing point, by Theorem 3, $T(p', W^*) \leqslant t(p', W')$.

Case 1. If $W' < W^*$, then

$$T(p', W') < T(p', W^*) \leqslant t(p', W').$$

and

$$T(p', W^*) \leqslant t(p', W') < t(p', W^*).$$

Case 2. If $W' \geqslant W^*$, then by the definition of isospeed scalability we have the scalability of combination 1 is equal to or smaller than the scalability of combination 2, $\Phi(p, p') \leqslant \Psi(p, p')$. Thus combination 1 has a larger initial time and equal or smaller scalability, by Theorem 1, $T(p', W') < t(p', W')$ and $T(p', W^*) < t(p', W^*)$.

Proof of the "if" condition.

Case 1. If $W' \geqslant W^*$, then

$$T(p', W') \geqslant T(p', W^*),$$

and, by the "if" assumption,

$$t(p', W') > T(p', W').$$

Combining these two inequalities, we have

$$t(p', W') > T(p', W^*).$$

By Theorem 3, p' is not a scaled crossing point.

Case 2. If $W' < W^*$, then, since combination 2 is data scalable, we have

$$a = \frac{W'}{p' \cdot t(p', W')} = \frac{W^*}{p' \cdot T(p', W^*)} > \frac{W'}{p' \cdot T(p', W')}.$$

Thus,

$$t(p', W') < T(p', W'). \tag{4}$$

Inequality (4) contradicts with the "if" assumption that $t(p', W') > T(p', W')$. This concludes that the case of $W' < W^*$ does not exist under the "if" assumption. ▌

A more interesting result is given by Theorem 5.

THEOREM 5. *Assume algorithm–machine combinations* 1 *and* 2 *are data scalable and combination* 1 *has an execution time larger than that of combination* 2 *at the initial state. Then the scaled ensemble size p' is not a scaled crossing point if and only if combination* 1 *has an execution time larger than that of combination* 2 *for solving any scaled problem $W^\dagger$ such that $W^\dagger$ is between W' and W^* at p', where W' and W^* are the scaled problem size of combination* 1 *and combination* 2 *respectively.*

Proof. We use the same notations as used in the proof of Theorem 4.

Proof of the "only if" condition.

Case 1. If $W^\dagger = W'$ or $W^\dagger = W^*$, then, by Theorem 4, we are done.

Case 2. If $W' < W^*$ and $W' < W^\dagger < W^*$, then, since p' is not a scaled crossing point, by Theorem 3, $T(p', W^*) \leqslant t(p', W')$:

$$t(p', W^\dagger) > t(p', W') \geqslant T(p', W^*) > T(p', W^\dagger).$$

Case 3. If $W' > W^*$ and $W^* < W^\dagger < W'$, then, since combinations 1 and 2 are data scalable,

$$a = \frac{W'}{p' \cdot t(p', W')} > \frac{W^\dagger}{p' \cdot t(p', W^\dagger)},$$

and

$$a = \frac{W^*}{p' \cdot T(p', W^*)} < \frac{W^\dagger}{p' \cdot T(p', W^\dagger)}.$$

Combining the two inequalities, we have

$$\frac{W^\dagger}{p' \cdot t(p', W^\dagger)} < \frac{W^\dagger}{p' \cdot T(p', W^\dagger)}.$$

That is

$$t(p', W^\dagger) > T(p', W^\dagger).$$

Proof of the "if" condition. It is a direct result of Theorem 4. ▮

Theorems 4 and 5 give the necessary condition for range comparison of scalable computing: *p' is not a scaled crossing point of p if and only if it is not an equal-size crossing point of p for any scaled problem size within the scalable range of the two compared algorithm–machine combinations.* The ranking of execution time can be computed from the ranking of scalability. Based on this theoretical finding, Fig. 1 gives the range comparison algorithm in terms of finding the smallest scaled crossing point via scalability comparison. Figure 2 gives an alternative range comparison algorithm in terms of finding the smallest scaled crossing point via execution time comparison. In general, there could have been more than one scaled crossing point over the consideration range for a given pair of AMCs. These algorithms may be used iteratively to find successive scaled crossing points.

4. TRIDIAGONAL SOLVERS: A CASE STUDY

Solving tridiagonal systems is one of the fundamental problems in scientific computing [10]. Many methods used for the solution of partial differential Eqs. (PDEs) rely on solving a sequence of tridiagonal systems. In addition to PDEs,

`Assumption of the Algorithm:` Assume algorithm-machine combinations 1 and 2 have execution time $t(p, W)$ and $T(p, W)$ respectively, and $t(p, W) = \alpha T(p, W)$ at the initial state, where $\alpha > 1$.

`Objective of the Algorithm:` Find the superior range of combination 2 starting at the ensemble size p

Range Comparison
Begin
$p' = p$;
Repeat
$p' = p' + 1$;
Compute the scalability of combination 1 $\Phi(p, p')$;
Compute the scalability of combination 2 $\Psi(p, p')$;
Until($\Phi(p, p') > \alpha\Psi(p, p')$ **or** p' = the limit of ensemble size)
If $\Phi(p, p') > \alpha\Psi(p, p')$ **then**
p' is the smallest scaled crossing point;
Combination 2 is superior at any ensemble size $p^\dagger$, $p \le p^\dagger < p'$;
Else
Combination 2 is superior at any ensemble size $p^\dagger$, $p \le p^\dagger \le p'$
End{If}
End{Range Comparison }

FIG. 1. Range comparison via crossing point analysis.

tridiagonal systems also arise in many other applications [11]. Three parallel tridiagonal solvers, the Parallel Partition LU (PPT), the Parallel Diagonal Dominant (PDD), and the Reduced Parallel Diagonal Dominant (RPDD, Reduced PDD) algorithms, are used to confirm the analytical results. Interested readers may refer to [12] and [11] for details of the algorithms, especially for accuracy analysis and extension of these algorithms to solve periodic systems and general banded linear systems. Only computation and communication count of the algorithms are presented here for scalability analysis.

4.1. Operation Comparison

Tridiagonal systems arising in many applications are multiple right-side systems. They are usually "kernels" in much larger codes. The computation and communication counts for solving multiple right-side systems are listed in Table 1, in which the factorization of the matrix is not considered. Parameter n_1 is the number of right-hand-sides (RHS). Note that for multiple RHS systems, the communication cost increases with the number of right-hand-sides. For the PPT algorithm, the communication cost also increases with the ensemble size. In the Reduced PDD algorithm, only j elements of the partition vectors need to be computed for the final modification. Formulas for computing the integer j can be found in [11], depending on particular circumstances.

`Assumption of the Algorithm:` Assume algorithm-machine combinations 1 and 2 have execution time $t(p, W)$ and $T(p, W)$ respectively, and at the initial state, $t(p, W) > T(p, W)$.

`Objective of the Algorithm:` Find the superior range of combination 2 starting at the ensemble size p

Range Comparison
Begin
$p' = p$;
Repeat
$p' = p' + 1$;
Compute the scalability of combination 1 $\Phi(p, p') = \frac{p' \cdot W}{p \cdot W'}$;
Until$(t(p', W') \leq T(p', W')$ **or** $p' =$ the limit of ensemble size)
If $t(p', W') \leq T(p', W')$ **then**
p' is the smallest scaled crossing point;
Combination 2 is superior at any ensemble size $p^{\dagger}$, $p \leq p^{\dagger} < p'$;
Else
Combination 2 is superior at any ensemble size $p^{\dagger}$, $p \leq p^{\dagger} \leq p'$
End{If}
End{Range Comparison }

FIG. 2. An alternative range comparison algorithm.

Communication cost has great impact on overall performance. For most distributed-memory computers, the time of a processor to communicate with its nearest neighbors is found to vary linearly with problem size. Let S be the number of bytes to be transferred. Then the transfer time to communicate with a neighbor can be expressed as $\rho + S\beta$, where ρ is a fixed startup time and β is the incremental transmission time per byte. Assuming 4 bytes are used for each real number, Steps 3 and 4 of the PDD and Reduced PDD algorithm take $\rho + 8\beta$ and $\rho + 4\beta$ time respectively on any architecture that supports single-array topology. The communication cost of the total-data-exchange communication is highly architecture-dependent. The listed communication cost of the PPT algorithm is based on a square 2-D torus with p processors (i.e., 2-D mesh, wraparound, square) [13]. If a hypercube topology or a multistage Omega network is assumed the communication cost would be $\log(p)\,\rho + 12(p-1)\,\beta$ and $\log(p)\,\rho + 8(p-1)\,n_1 \cdot \beta$ for single systems and systems with multiple right sides, respectively [12, 14].

4.2. Scalability Analysis

The scalability analysis of the PDD algorithm for solving single systems can be found in [11]. In the following, we give a scalability analysis of the PDD algorithm for solving systems with multiple right sides, where the number of right sides does not increase with the ensemble size and the LU factorization of the matrix is not considered. Scalability analysis of the PPT and the Reduced PDD algorithms is presented under the same assumption.

TABLE 1

Comparison of Computation and Communication

System	Algorithm	Computation	Communication
Multiple right sides	Best sequential	$(5n-3)\cdot n_1$	0
	PPT	$\left(9\frac{n}{p}+10p-11\right)\cdot n_1$	$(2\rho+8p\cdot n_1\cdot\beta)(\sqrt{p}-1)$
	PDD	$\left(9\frac{n}{p}+1\right)\cdot n_1$	$(2\rho+8n_1\cdot\beta)$
	Reduced PDD	$\left(5\frac{n}{p}+4j+1\right)\cdot n_1$	$(2\rho+8n_1\cdot\beta)$

Following the notation given in Section 2, we let $T(p, W)$ be the execution time for solving a system with W work (problem size) on p processors. By the definition of isospeed scalability, the ideal situation would be when both the number of processors and the amount of work are scaled up by a factor of N, the execution time remaining unchanged:

$$T(N\times p, N\times W) = T(p, W). \tag{5}$$

To incorporate the effect of numerical inefficiencies in parallel algorithms, in practice, the flop count is based upon some practical optimal sequential algorithm. In our case, Thomas' algorithm [15] was chosen as the sequential algorithm. It takes $(5n-3)\cdot n_1$ floating point operations for multiple right sides, where the number 3 can be neglected for large n. As the problem size W increases N times to W', we have

$$W' = (N\times 5n)\cdot n_1 = (5n')\cdot n_1, \tag{6}$$

$$n' = N\cdot n. \tag{7}$$

THE PDD ALGORITHM. Let τ_{comp} represent the unit of a computation operation normalized to the communication time. The time required by the PDD algorithm with p processors is

$$T(p, W) = \left(9\frac{n}{p}+1\right)n_1\cdot\tau_{\text{comp}} + 2(\rho+8\cdot n_1\cdot\beta),$$

and

$$\begin{aligned} T(N\times p, N\times W) &= \left(9\frac{n'}{N\cdot p}+1\right)n_1\cdot\tau_{\text{comp}} + 2(\rho+4n_1\cdot\beta) \\ &= \left(9\frac{N\cdot n}{N\cdot p}+1\right)n_1\cdot\tau_{\text{comp}} + 2(\rho+4n_1\cdot\beta) \\ &= \left(9\frac{n}{p}+1\right)n_1\cdot\tau_{\text{comp}} + 2(\rho+4n_1\cdot\beta) \\ &= T(p, W). \end{aligned}$$

Thus the PDD algorithm is perfectly scalable. Its scalability equals 1. Note that in the above analysis we assume $T(p, W)$ contains the communication cost. The perfect scalability does not apply for the special case where $p = 1$.

THE REDUCED PDD ALGORITHM. The Reduced PDD algorithm has the same computation and communication pattern as the PDD algorithm, but has an operation count smaller than that of the PDD algorithm. Similar arguments can be applied to the Reduced PDD algorithm as well. Therefore, the PDD and the Reduced PDD algorithm have the same scalability. They are perfectly scalable.

THE PPT ALGORITHM. The PPT algorithm is not perfectly scalable. Its scalability analysis requires more discussion. The prediction formula (2) is needed for the scalability analysis,

$$W' = \frac{a \cdot p' \cdot T'_o(W')}{1 - a \cdot \Delta}, \tag{8}$$

where W', p', a, and Δ are as defined in (2). Parameters a and Δ do not vary with the number of processors. For a given AMC and a given initial average speed, $c = \frac{a}{1 - a \cdot \Delta}$ is a constant. Therefore, Eq. (8) can also be written as

$$W' = c \cdot p' \cdot T'_o(W') \tag{9}$$

By Eq. (6), W' is a linear function of n. The computing time can be represented as

$$T(p, n) = T_s(p, n) + T_o(p, n), \tag{10}$$

where $T_s(p, n)$ is the computing time with ideal parallelism and $T_o(p, n)$ represents the degradation of parallelism. For the particular problem discussed here, the run time model is (see Table 1)[4]

$$T(p, n) = \left(9\frac{n}{p} + 10p\right) \cdot n_1 \cdot \tau_{\text{comp}} + (2\rho + 8 \cdot n_1 \cdot p \cdot \beta)(\sqrt{p} - 1). \tag{11}$$

By Eq. (6),

$$T_s(p, n) = \frac{5n}{p} \cdot n_1 \cdot \tau_{\text{comp}}. \tag{12}$$

Therefore,

$$T_o(p, n) = \left(4\frac{n}{p} + 10p\right) \cdot n_1 \cdot \tau_{\text{comp}} + (2\rho + 8 \cdot n_1 \cdot p \cdot \beta)(\sqrt{p} - 1).$$

Using the prediction formula (9), we have

$$W' = c \cdot p' \cdot T'_o = c \cdot p' \left[\left(4\frac{n'}{p'} + 10p'\right) \cdot n_1 \cdot \tau_{\text{comp}} + (2\rho + 8 \cdot n_1 \cdot p' \cdot \beta)(\sqrt{p'} - 1) \right].$$

[4] The constant number 11 is eliminated for convenience, since it is independent of parameters n and p.

Substituting $W' = 5 \cdot n' \cdot n_1$ into the above equation,

$$5 \cdot n' \cdot n_1 \cdot \tau_{\text{comp}} = c \cdot p' \left[\left(4 \frac{n'}{p'} + 10p' \right) \cdot n_1 \cdot \tau_{\text{comp}} + (2\rho + 8 \cdot n_1 \cdot p' \cdot \beta)(\sqrt{p'} - 1) \right],$$

which eventually leads to

$$n' = c'[10p'^2 \cdot n_1 \cdot \tau_{\text{comp}} + (2\rho p' + 8 \cdot n_1 \cdot p'^2 \cdot \beta)(\sqrt{p'} - 1)], \tag{13}$$

where $c' = c/(5-4c) \cdot n_1 \cdot \tau_{\text{comp}}$. Equation (13) is true for any work-processor pair that maintains the fixed average speed. In particular:

$$n = c'[10 \cdot p^2 \cdot n_1 \cdot \tau_{\text{comp}} + (2 \cdot \rho \cdot p + 8 \cdot n_1 \cdot p^2 \cdot \beta)(\sqrt{p} - 1)] \tag{14}$$

Combining Eqs. (13) and (14), we have

$$n' - n = c'[10 \cdot n_1 \cdot (p'^2 - p^2) \cdot \tau_{\text{comp}} + 2\rho(p'^{3/2} - p^{3/2}) \tag{15}$$

$$+ 8 \cdot n_1 \cdot \beta(p'^{5/2} - p^{5/2}) - 2\rho(p' - p) - 8 \cdot n_1 \cdot \beta(p'^2 - p^2)]. \tag{16}$$

If the communication start-up time is the dominant factor of the overhead, then

$$n' - n \approx 2c' \cdot \rho \cdot (p'^{3/2} - p^{3/2}),$$

which shows that the variation of n is in direct proportion to the $3/2$ power of the variation of ensemble size. By Eq. (6), W, the work, is in direct proportion to the order of matrix n, therefore, the scalability of this AMC can be estimated as

$$\psi(p, p') = \psi(p, Np) = \frac{NpW}{pW'} \approx \frac{N \cdot W}{N^{3/2}W} = \frac{1}{\sqrt{N}}. \tag{17}$$

Similarly, if the computing is the dominant factor of the overhead, then

$$n' - n \approx 10c' \cdot n_1 \cdot (p'^2 - p^2) \cdot \tau_{\text{comp}},$$

and

$$\psi(p, p') = \psi(p, Np) = \frac{NpW}{pW'} \approx \frac{N \cdot W}{N^2 W} = \frac{1}{N}; \tag{18}$$

if the transmission delay is the dominant factor of the overhead, then

$$n' - n \approx 8c' \cdot n_1 \cdot \beta(p'^{5/2} - p^{5/2}),$$

and

$$\psi(p, p') = \psi(p, Np) = \frac{NpW}{pW'} \approx \frac{N \cdot W}{N^{5/2}W} = \frac{1}{\sqrt{N^3}}. \tag{19}$$

In any case, the PPT algorithm is far from ideally scalable. Its scalability decreases with the increase of ensemble size and the rate of the decrease varies with machine parameters.

5. EXPERIMENTAL RESULTS

The PPT, PDD, and Reduced PDD algorithms were implemented on an IBM SP2 and an Intel Paragon. Both SP2 and Paragon machines are distributed-memory

parallel computers that adopt message-passing communication paradigm and support virtual memory. Each processor (node) of the SP2 is functionally equivalent to a RISC System/6000 desktop system (thin node) or a RISC System/6000 deskside system (wide node). The Paragon XP/S supercomputer uses the i860 XP microprocessor, which includes an RISC integer core processing unit and three separate on-chip caches for page translation, data, and instructions. The SP2 High-Performance Switch is a multistage packet-switched Omega network that provides a minimum of four paths between any pair of nodes in the system. The processors of Intel Paragon are connected in a two-dimensional rectangular mesh topology. For SP2, the measured latency is 45 μs and the bandwidth is 35 Mbytes per second. For Paragon, the measured latency is 46 μs and the bandwidth is 80 Mbytes per second. The SP2, available at NASA Langley Research Center, has 48 wide nodes with 128-Mbyte local memory each. The Paragon available at the center has 72 nodes with 32-Mbyte local memory each.

As an illustration of the algorithms and theoretical results given in previous sections, a sample matrix is tested. This sample matrix is a diagonal dominant, symmetric, Toeplitz system

$$A = \begin{bmatrix} 1 & \frac{1}{3} & & & & & \\ \frac{1}{3} & 1 & \frac{1}{3} & & & & \\ & \ddots & \ddots & \ddots & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & \ddots & \ddots & \ddots & \\ & & & & \frac{1}{3} & 1 & \frac{1}{3} \\ & & & & & \frac{1}{3} & 1 \end{bmatrix} = [\tfrac{1}{3}, 1, \tfrac{1}{3}]$$

arising in CFD applications [11]. $j = 17$ has been chosen for the Reduced PDD algorithm to reach the single precision accuracy, 10^{-7}.

The nodes of SP2 and Paragon have different processing powers and local memory sizes. For a fixed 1024 right sides, following the asymptotic speed concept [7], the order of the matrix for the SP2 has been chosen to be 6400 and the order of the matrix for the Paragon has been chosen to be 1600 for uniprocessor processing. Execution time is measured in seconds. Speed is given in MFLOPS (millionth floating-point operation per second). Tables 2 through 6 list the measured results on the SP2 and Paragon machines. Measurements begin with two processors, since uniprocessor processing does not involve communication on SP2 and Paragon and, therefore, the uniprocessor performance is not suitable for the analytical results. From Tables 2 and 4, we can see that the execution time of the PDD and Reduced PDD remains unchanged, except for some minor measuring perturbations when the order of the matrix doubles with the number of processors. Since problem size increases linearly with the order of the matrix for our applications, the constant timing indicates that the PDD and Reduced PDD algorithms are ideally scalable. This indication is confirmed by Tables 3 and 5, which show that the average speed of these two algorithms are unchanged on both SP2 and Paragon machine. By the definition of isospeed scalability, the four algorithm–machine combinations,

TABLE 2

Measured Execution Time (in Seconds) on the SP2 Machine

	Number of Processors				
	2	4	8	16	32
Order of matrix	12800	25600	51200	102400	204800
PDD algorithm	0.8562	0.8561	0.8564	0.8564	0.8569
Reduced PDD alg.	0.5665	0.5666	0.5668	0.5673	0.5659
PPT algorithm	0.7810	0.9826	1.004	1.103	1.288

PDD–SP2, PDD–Paragon, RPDD–SP2, and RPDD–Paragon, are perfectly scalable, with scalability equal to 1.

Since the PDD and Reduced PDD algorithms have the same scalability, these two algorithms satisfy the condition of Corollary 3. Their performance can be used to verify this corollary. Observing the timing given in Tables 2 and 4, we can see that the measured results confirm the theoretical results. For instance, based on Table 2, the initial timing ratio between the PDD and the Reduced PDD algorithm, α, remains unchanged when the problem size is scaled up with the ensemble size. The Reduced PDD algorithm is superior to the PDD algorithm over the scalable computing range. Similarly, since the scalability of the PPT algorithm is less than the scalability of the PDD and the Reduced PDD algorithms, the performance comparison of these three algorithms can be used to verify Theorems 1 and 3. By Theorem 1, since the PPT algorithm is slow at the initial state on the Paragon machine, it is inferior to the computing range, and the timing difference between the PPT algorithm and the PDD and Reduced PDD algorithms should be enlarged when problem size is scaled up with ensemble size. This claim is supported by the measured data on the Paragon machine. Performance on the SP2 is more interesting. The PPT algorithm is faster than the PDD algorithm at the initial state. The initial time difference ratio is $0.8562/0.781 = 1.0963$. By Eq. (17), the scalability of

TABLE 3

Measured Average Speed (in MFLOPS) on the SP2 Machine

	Number of Processors				
	2	4	8	16	32
Order of matrix	12800	25600	51200	102400	204800
PDD algorithm	38.292	38.2975	38.2850	38.285	38.2625
Reduced PDD alg.	57.875	57.865	57.845	57.795	57.9375
PPT algorithm	41.979	35.9275	32.6562	29.7250	25.455

TABLE 4

Measured Execution Time (in Seconds) on the Paragon Machine

	Number of Processors					
	2	4	8	16	32	64
Order of Matrix	3200	6400	12800	25600	51200	102400
PDD Alg.	0.7379	0.7388	0.7387	0.7397	0.7388	0.7393
Reduced PDD Alg.	0.5452	0.5524	0.5539	0.5550	0.5521	0.5563
PPT Alg.	0.8317	0.9115	1.066	1.462	2.008	3.095

the PPT algorithm $\Phi(2, 4) = \frac{1}{\sqrt{2}} = 0.7$. The PDD is ideally scalable, $\Psi(2, 4) = 1$. Therefore, following the range comparison algorithm given in Fig. 1, the first scaled crossing point is at ensemble size 4. This predicted scaled crossing point is confirmed by experimental measurement (see Table 2). In summary, the following range comparison principles have been confirmed by experimental results.

- (PDD/Reduced PDD, Theorem 1) If two programs have the same scalability, then the initially faster program will remain faster over the considered scalable range.

- (Reduced PDD/PPT, Theorem 1) If the initially faster program has a larger scalability, then the initially faster program will be superior over the considered scalable range.

- (PDD/PPT, Theorem 4) If the initially faster program has a smaller scalability, then fast/slow performance will change when system size increases. The scaled crossing point can be predicted. In the PDD/PPT case, the scaled crossing point is 4.

TABLE 5

Measured Average Speed (in MFLOPS) on the Paragon Machine

	Number of Processors					
	2	4	8	16	32	64
Order of Matrix	3200	6400	12800	25600	51200	102400
PDD Alg.	11.1	11.0925	11.0950	11.0813	11.0938	11.0875
Reduced PDD Alg.	15.03	14.8375	14.8	14.7688	14.8469	14.7359
PPT Alg.	9.855	8.9925	7.6887	5.605	4.0812	2.6484

TABLE 6

Variation of the Reduced PDD Algorithm on the Paragon Machine

	Number of Processors				
	4	8	16	32	64
Order of Matrix	1000	2000	4000	8000	16000
Timing	0.1154	0.1155	0.1166	0.1159	0.1159
Average Speed	11.095	11.0875	10.9812	11.0469	11.0453
Order of Matrix	6400	12800	25600	51200	102400
Timing	0.5524	0.5539	0.5550	0.5521	0.5563
Average Speed	14.8375	14.8	14.7688	14.8469	14.7359

Theorem 2 provides the foundation of range comparison from another angle. Instead of using initial time difference, Theorem 2 uses initial efficiency (in terms of average speed) to predict the scaled performance. Table 6 shows the performance variation of the Reduced PDD algorithm on the Paragon. A small problem size, $n = 1000$, is chosen so that the Reduced PDD can reach the achieved average speed of the PDD algorithm with larger size (see Table 6). The initial ensemble size is chosen to be four because when the problem size is small the overall performance is highly dependent on communication delay. With two processors the PDD and Reduced PDD algorithms have one send and one receive communication. With more than two processors these algorithms require two send-and-receive communications. Although theoretically each processor on Paragon can send and receive messages concurrently, in practice the synchronization cost of concurrent sending and receiving may lead to noticeable performance differences when the problem size is small. The PDD algorithm and Reduced PDD algorithm reach the same average speed at ensemble size equal to 4 with problem size $W = (5n-3) * 1024 + 3n-4 = 32, 784, 124$ flops and $W = (5n-3) * 1024 + 3n-4 = 5, 119, 924$ flops respectively. The ratio of problem size difference, computed as $5, 119, 924$ over $32, 784, 124$, is 0.15617. That is, $\alpha = 0.15617$. The PDD and Reduced PDD algorithm have the same scalability. By Theorem 2, the execution time of the PDD algorithm on its scaled problem size should be greater than that of the Reduced PDD algorithm over the scalable computing range. Measured results given in Tables 6 and 4 confirm the theoretical statement.

The PPT algorithm is programmed using Fortran and the code is identical for both the SP2 and the Paragon except for communication commands. MPL is used on SP2 for message passing. The all-to-all communication is implemented by calling communication library calls, *gcol* is used on the Paragon, and *mp_concat* is used on the SP2. From Tables 2, 4, and 3, 5, we can see that the PPT algorithm has a smaller time increase and less average speed reduction on the SP2 than on the Paragon. This means the PPT algorithm has a better scalability on the SP2 than on the Paragon. The better scalability may be due to various reasons, including larger

memory and more efficient all-to-all communication subroutines available on the SP2. Interested readers may refer to [14] for more information on all-to-all communications. The emphasis here is that when an algorithm is not ideally scalable, its scalability does vary with machine parameters.

Range comparison is not only useful in algorithm or software development. It is also applicable in evaluating hardware variations. The best sequential tridiagonal solver, the Thomas algorithm, can be parallelized for systems with multiple right sides. The parallelized Thomas algorithm has less computing but more communication requirement than that of the PDD algorithm. Figures 3 and 4 demonstrate the performance range comparison of the PDD and parallelized Thomas algorithm, when the computing and communication capacity vary, respectively [8]. These figures are created based on scalability analysis formula with measured machine parameters. The number of processors used in both figures is fixed as 64. We can see that computing speed increases do not change the superiority. The PDD algorithm, its performance represented by the lower surface, remains superior in Fig. 3. However, communication capacity will change the superiority. As depicted in Fig. 4, when communication speed increases, the PDD algorithm, whose performance is given by the unshaded surface, changes from superior to inferior.

Scalability formulas are derived in Section 4 to verify the experimental results. In general, scalability also can be directly measured or computed from execution time [3]. We used the term algorithm–machine combinations through our analytical study. An algorithm could be implemented differently due to data distribution and other low-level implementation details. Different implementations could lead to different performance. The crossing point methodology applies to implementations as well. When scalabilities are derived from algorithm analysis, the crossing point is for the algorithm. When scalabilities are measured or computed from the implementations, the crossing point is for the implementations. The range comparison concept has been tested in restructuring compilation for data-parallel programming, where optimal data distribution needs to be determined for best performance [16]. Initial results are very encouraging.

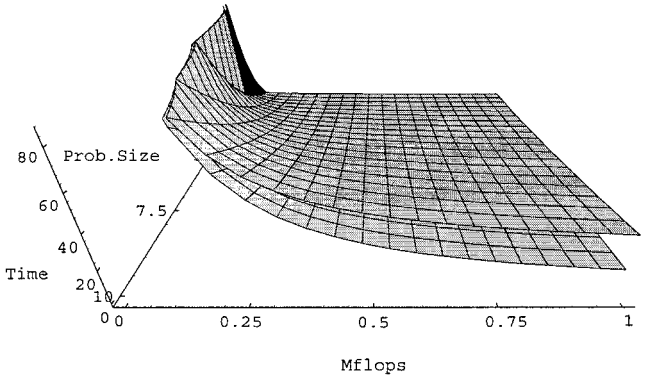

FIG. 3. Speed Influence on Testing Algorithms.

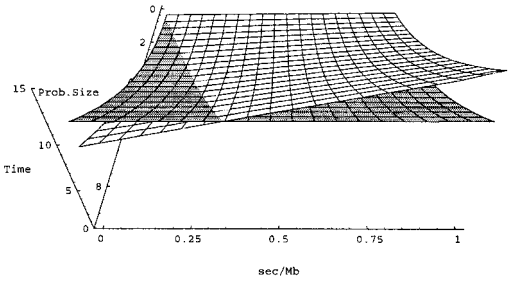

FIG. 4. Communication Influence on Testing Algorithms.

6. CONCLUSION

Different algorithms could solve a given application, and a given algorithm could have different implementations. The performance gain of parallel algorithms and implementations varies with problem size and system size. A slow algorithm with good scalability may become superior when system and problem size scale up. The system sizes for which the performance ranking of different algorithms/programs changes are called (performance) crossing points. Identifying crossing points is crucial for performance optimization and compiler-assisted parallelization. In this study, the relation between scalability and execution time is first carefully studied. Next, based on scalability analysis, the concepts of crossing point analysis and range comparison are proposed. Then a novel methodology for locating the crossing point via scalability and for evaluating the performance of different parallel algorithms and architectures over a range of system and problem sizes via crossing points is developed. Two algorithms for conducting range comparison are presented. Finally, experimental results are provided to confirm the analytical findings. Scalability makes range comparison. With initial execution time and scalability, the execution time of different algorithms can be compared and predicted over a range of problem size and system size. Unlike execution time, scalability is "dimensionless." It does not depend on problem size. The "dimensionless" and the range comparison methodology provided in this study makes a manageable database for performance prediction possible. Range comparison, in which execution time is compared over a range of system and problem sizes, opens new ways for performance evaluation. Its impact on scalable software development needs to be further explored.

ACKNOWLEDGMENTS

The author is grateful to S. Moitra of NASA Langley Research Center for help in gathering the performance data on the SP2 and to the anonymous referees for their constructive comments on the revision of the paper.

REFERENCES

1. P. Worley, The effect of time constraints on scaled speedup, *SIAM J. Sci. Stat. Comput.* **11** (Sept. 1990), 838–858.

2. A. Y. Grama, A. Gupta, and V. Kumar, Isoefficiency: Measuring the scalability of parallel algorithms and architectures, *IEEE Parallel Distrib. Technol.* **1** (Aug. 1993), 12–21.
3. X.-H. Sun and D. Rover, Scalability of parallel algorithm–machine combinations, *IEEE Trans. Parallel Distrib. Systems* **5** (June 1994), 599–613.
4. X. Zhang, Y. Yan, and K. He, Latency metric: An experimental method for measuring and evaluating parallel program and architecture scalability, *J. Parallel Distrib. Comput.* **22** (Sept. 1994), 392–410.
5. S. Sahni and V. Thanvantri, Performance metrics: Keeping the focus on runtime, *IEEE Parallel Distrib. Technol.* (Spring 1996), 43–56.
6. K. Hwang and Z. Xu, "Scalable Parallel Computing," McGraw–Hill, New York, 1998.
7. X.-H. Sun and J. Zhu, Performance considerations of shared virtual memory machines, *IEEE Trans. Parallel Distrib. Systems* (Nov. 1995), 1185–1194.
8. X.-H. Sun and J. Zhu, Performance prediction: A case study using a scalable shared-virtual-memory machine, *IEEE Parallel Distrib. Technol.* (Winter 1996), 36–49.
9. X.-H. Sun, "Scalability versus Execution Time in Scalable Systems," Louisiana State Uiversity, Computer Science TR-97-003, 1997. [Revised May 1998.]
10. C. Ho and S. Johnsson, Optimizing tridiagonal solvers for alternating direction methods on boolean cube multiprocessors, *SIAM J. of Sci. and Stat. Computing* **11** (1990), 563–592.
11. X.-H. Sun, Application and accuracy of the parallel diagonal dominant algorithm, *Parallel Computing* (Aug. 1995), 1241–1267.
12. X.-H. Sun, H. Zhang, and L. Ni, Efficient tridiagonal solvers on multicomputers, *IEEE Transactions on Computers* **41** (1992), 286–296.
13. V. Kumar *et al.*, "Introduction to Parallel Computing: Design and Analysis of Algorithms," Benjamin–Commings, Redwood City, CA, 1994.
14. V. Bala *et al.*, Ccl: A portable and tunable collective communication library for scalable parallel computers, *IEEE Trans. Parallel Distrib. Systems* **6** (Feb. 1995), 154–164.
15. C. Hirsch, "Numerical Computation of Internal and External Flows," Wiley, New York, 1988.
16. X.-H. Sun, M. Pantano, and T. Fahringer, Integrated range comparison for data-parallel compilation systems, *IEEE Trans. Parallel Distrib. Systems* **10** (May, 1999), 448–458.

XIAN-HE SUN received his Ph.D. in computer science from Michigan State University. He was a staff scientist at ICASE and NASA Langley Research Center and was an associate professor in the Computer Science Department at Louisiana State University. Currently he is an associate professor and the director of the Scalable Computing Software Laboratory in the Computer Science Department at the Illinois Institute of Technology (IIT) and a guest faculty member at the Argonne National Laboratory. Dr. Sun's research interests include parallel and distributed processing, performance evaluation, software systems, and scientific computing. He has published intensively in the field and his research has been supported by DoD, DoE, NASA, NSF, and other government agencies. He is a senior member of IEEE, a member of ACM, New York Academy of Science, and Phi Kappa Phi, and a distinguished visitor of the IEEE Computer Science society and has served and is serving as the chairman or on the program committee for more than 10 different international conferences and workshops. He received the ONR and ASEE Certificate of Recognition award in 1999 and the Best Paper Award from the International Conference on Parallel Processing (ICPP01) in 2001.