Performance Range Comparison Via Crossing Point Analysis

Xian-He Sun
Department of Computer Science
Louisiana State University
Baton Rouge, LA 70803-4020
sun@bit.csc.lsu.edu

Abstract.

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 never 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, based on scalability analysis, 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 the combination of scalability, crossing point analysis, and range comparison provides a practical solution for scalable performance evaluation. While our testings are conducted on homogeneous parallel computers, the proposed methodology applies to heterogeneous and network computing as well.

1 Introduction

In a parallel and distributed environment, load balance over processors generally decreases with the ensemble size (the number of processors available) while communication overhead increases with ensemble size. The decrease of load balance and increase of communication overhead may reduce the performance considerably and lead to a much longer execution time than expected when problem and system size increase. More importantly, the "decrease" and "increase" vary with algorithms, machines, and Algorithm-Machine Combinations (AMCs). They are also functions of system ensemble size and problem size. An initially fast parallel implementation may become 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.

Execution time is the ultimate and most used measure in practice. System ensemble and problem size vary in a scalable computing environment. How to compare execution time over a range of system and problem size, however, eludes researchers still. Scalability analysis has been introduced in recent years to identify the scaling properties of parallel algorithms and machines [2, 1, 3, 8, 4]. Based on newly revealed relations between scalability and execution time, in this paper, the concept of range comparison is introduced. Unlike conventional execution time comparison, range comparison compares performance over a wide range of ensemble and problem size via scalability and performance crossing point analysis. The idea of range comparison is straightforward: find the first performance superior/inferior 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 well addressed in this study. Theoretical foundation is built to find the crossing point via *isospeed scalability* [8]. Since the relation between isospeed scalability and other scalabilities has been studied [9], results presented in this paper can be extended to other scalabilities as well.

2 Background

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, therefore, is *speed* which is defined as work divided by time. In general, how work should be defined is debatable. For scientific applications, it is commonly agreed that the floating-point (flop) operation count is a good estimate of work. *Average unit speed* is the achieved speed of the given computing system divided by p, the number of processors. It represents a quantity or efficiency that ideally would be constant with the system size. *Isospeed scalability* has been formally defined in [8] as the ability to maintain the average unit speed based on this observation.

Definition 1 *An* `algorithm-machine combination` *is* `scalable` *if the achieved average speed of the algorithm on the given machine can remain constant with increasing numbers of processors, provided the problem size can be increased with the system size.*

For a large class of algorithm-machine combinations, the average speed can be maintained by increasing problem size [8]. The necessary increase of problem size varies with algorithms, machines, and their 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 needed to maintain the average speed when $p' > p$ processors are employed. We define the *scalability from ensemble size p to ensemble size p'* of an algorithm-machine combination as follows:

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

The work W' is determined by the isospeed constraint. When $W' = \frac{p'}{p}W$, that is when average speed is maintained with work per processor unchanged, the scalability equals one. This is the ideal case. In general, work per processor may have to be increased to achieve the fixed average speed, and scalability is less than one.

Since the average speed is fixed, the isospeed scalability (1) also can be equivalently defined in terms of execution time [8]:

$$\psi(p,p') = \frac{T_p(W)}{T_{p'}(W')} \tag{2}$$

where $T_{p'}(W')$ is the corresponding execution time of solving W' on p' processors.

By the definition of isospeed scalability, scalability can be predicted if and only if the scaled work size, W', can be predicted. A prediction formula has been given in [10] to compute W':

$$W' = \frac{a \cdot p' \cdot T_o(W')}{1 - a\Delta} \tag{3}$$

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.

Theorem 1 and 2 show that isospeed scalability favors systems with better run-time and characterizes the run-time well when problem size scales up with system size. The proofs and some direct results of Theorem 1 and 2 can be found in [6].

Theorem 1 *If algorithm-machine combinations 1 and 2 have execution time $\alpha \cdot T$ and T, respectively, at the same initial state (the same initial ensemble and problem size), then combination 1 has a higher scalability than combination 2 if and only if the execution time of combination 1 is smaller than the α multiple of the execution time of combination 2 for solving W', where W' is the scaled problem size of combination 1.*

Theorem 1 shows that if two AMCs have some initial performance difference, in terms of execution time, then the faster AMC will remain faster on scaled problem sizes if it has a larger scalability. Initial performance difference can be presented in terms of execution time, as given in Theorem 1, or in terms of problem size needed for obtaining the desired average unit speed, as in most scalability studies [8]. Theorem 2 shows the relation of scalability and execution time when the initial performance difference is given in terms of problem size.

Theorem 2 *If algorithm-machine combinations 1 and 2 achieve the same average speed with problem size W and $\alpha \cdot W$, respectively, at the same initial ensemble size, then the α multiple of the scalability of combination 1 is greater than the scalability of combination 2 if and only if combination 1 has a smaller execution time than that of combination 2 for solving W', where W' is the scaled problem size of combination 1.*

3 Range Comparison and Crossing Point Analysis

Theorems 1 and 2 give the basic relation of scalability and execution time: better scalability leads to better execution time. Range comparison becomes more challenging when the initial faster AMC has a smaller scalability. When the system ensemble size scales up, an originally faster code with smaller scalability can become slower than code that has a better scalability. Finding the fast/slow crossing point is critical to optimizing performance. Finding the superior/inferior crossing point, however, is very difficult and has yet to be fully studied. 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 crossing point based on isospeed scalability. The correctness of the definition is proved by Proposition 3. The relation given by Proposition 2 and 3 is important by itself. It not only contributes to the concept of crossing point, but of equal importance, it demonstrates the uniqueness of scalability measurement in evaluation and benchmarking of algorithms and parallel computers.

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 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 the crossing point of AMC 1 and 2 if and only if

$$\frac{\Phi(p,p')}{\Psi(p,p')} > \alpha. \tag{4}$$

In fact, as given by Theorem 3, when $\Phi(p,p') > \alpha\Psi(p,p')$ we have $t_{p'}(W') < T_{p'}(W^*)$. Notice that since $\alpha > 1$ combination 2 has a smaller execution time at the initial state, $t_p(W) > T_p(W)$. This superior/inferior change in execution time illustrates the meaning of crossing point. For the sake of showing the relation between parallel processing overhead and scaled crossing point, in the following we do not prove the correctness of Definition 2 directly. Instead, we prove the correctness through an alternative equivalent definition which is defined in terms of parallel processing overhead. In Definition 3 and through out this paper, the scaled problem size is the scaled problem size under isospeed scalability.

Definition 3 (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 the scaled ensemble size p' is a crossing point of combinations 1 and 2 if p' is a scaled ensemble size at which the ratio of the scaled parallel overhead of combination 1 and 2 is less than $\frac{1-a\Delta/\alpha}{1-a\Delta}$, where a is the average speed of combination 2, Δ is the computing rate.*

Proposition 1 *Definition 2 and Definition 3 are equivalent.*

Proof: Let AMC 1 have execution time t, parallel processing overhead t_o, scalability $\Phi(p,p')$, and scaled problem size W'. Let AMC 2 have execution time T, parallel overhead T_o, average speed a, scalability $\Psi(p,p')$, and scaled problem size W^*. Since $t_p(W) = \alpha T_p(W)$ at the initial state, the average speed of combination 1 is a/α. By the definition of isospeed scalability, we have

$$\frac{\alpha W'}{p' t_{p'}(W')} = \frac{W^*}{p' T_{p'}(W^*)}.$$

Thus,

$$\frac{W^*}{W'} = \frac{\alpha T_{p'}(W^*)}{t_{p'}(W')}. \tag{5}$$

Also, by equation (3)

$$\frac{W^*}{W'} = \frac{a \cdot p' \cdot T_o(W^*)}{(1-a\Delta)} \cdot \frac{\alpha(1-a\Delta/\alpha)}{a \cdot p' \cdot t_o(W')} = \frac{\alpha(1-a\Delta/\alpha)}{1-a\Delta} \cdot \frac{T_o(W^*)}{t_o(W').} \tag{6}$$

Combining the equations (5) and (6) and using the relation (2), we get

$$\frac{T_o(W^*)}{t_o(W')} = \frac{1-a\Delta}{1-a\Delta/\alpha} \cdot \frac{\Psi^{-1}(p,p') \cdot T_p(W)}{\Phi^{-1}(p,p') \cdot t_p(W)} = \frac{1-a\Delta}{\alpha(1-a\Delta/\alpha)} \cdot \frac{\Phi(p,p')}{\Psi(p,p').} \tag{7}$$

Equation (7) shows that $\frac{t_o(W')}{T_o(W^*)} = \frac{1-a\Delta/\alpha}{1-a\Delta}$ if and only if $\frac{\Phi(p,p')}{\Psi(p,p')} = \alpha$, and $\frac{t_o(W')}{T_o(W^*)} < \frac{1-a\Delta/\alpha}{1-a\Delta}$ if and only if $\frac{\Phi(p,p')}{\Psi(p,p')} > \alpha$. That is Definition 2 and 3 are equivalent. □

Proposition 2 *If algorithm-machine combinations 1 and 2 have execution time αT and T respectively at the initial state, then combinations 1 and 2 have the same scaled execution time at scaled ensemble size p' if and only if combinations 1 and 2 have scaled parallel overhead $\frac{1-a\Delta/\alpha}{1-a\Delta}T_o$ and T_o respectively at ensemble size p'.*

Proof: We use the same notations as used in the proof of Proposition 1. By relation (2), we have

$$\begin{aligned} t_{p'}(W') &= \Phi^{-1}(p,p') \cdot t_p(W), & (8)\\ T_{p'}(W^*) &= \Psi^{-1}(p,p') \cdot T_p(W). & (9) \end{aligned}$$

By equation (1)

$$\Phi^{-1}(p, p') = \frac{pW'}{p'W} = \frac{pW^*}{p'W} \cdot \frac{W'}{W^*} = \Psi^{-1}(p, p') \frac{W'}{W^*.}$$

Thus, by equation (8)

$$t_{p'}(W') = \Psi^{-1}(p, p') \cdot \frac{W'}{W^*} \cdot t_p(W) = \frac{W'}{W^*} \cdot \frac{t_p(W)}{T_p(W)} \cdot T_{p'}(W^*). \tag{10}$$

Also, by equation (6)

$$\frac{W^*}{W'} = \frac{\alpha(1 - a\Delta/\alpha)}{1 - a\Delta} \cdot \frac{T_o(W^*)}{t_o(W').}$$

Substitute relation (6) into equation (10), we get the final relation

$$t_{p'}(W') = \frac{1 - a\Delta}{\alpha(1 - a\Delta/\alpha)} \cdot \frac{t_o(W')}{T_o(W^*)} \cdot \frac{t_p(W)}{T_p(W)} T_{p'}(W^*) = \frac{1 - a\Delta}{1 - a\Delta/\alpha} \cdot \frac{t_o(W')}{T_o(W^*)} \cdot T_{p'}(W^*).$$

That is $\frac{t_o(W')}{T_o(W^*)} = \frac{1-a\Delta/\alpha}{1-a\Delta}$ if and only if $t_{p'}(W') = T_{p'}(W^*)$, which completes the proof. □

Proposition 3 *For any $\alpha > 1$, if algorithm-machine combinations 1 and 2 have execution time αT and T respectively at the initial state, then combination 1 has a smaller scaled execution time than that of combination 2 at scaled ensemble size p' if and only if combination 1's scaled parallel overhead is less than the $\frac{1-a\Delta/\alpha}{1-a\Delta}$ multiple of combination 2's scaled parallel overhead at p', where a is the average speed of combination 2.*

Proof: Similar as the proof of Proposition 2. □

Theorem 3 is a direct result of Proposition 3.

Theorem 3 *If algorithm-machine combination 1 has a larger execution time than 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 smaller scaled execution time than that of combination 2.*

Since two different algorithm-machine combinations may have different scalabilities, they may cross performances at crossing point p' with different problem sizes. Scaled crossing point is different from the equal-size crossing point where performance crosses with the same problem size. Proposition 4 gives a relation between the scaled crossing point and equal-size crossing point. It shows that in the range of (p, p'), combination 2 is superior than combination 1 under the condition of Proposition 2, in terms of both scaled and equal-size performance.

Proposition 4 *Under the assumption of Proposition 2, we have*

$$T_{p'}(W') < T_{p'}(W^*) = t_{p'}(W')$$

where W' is the scaled problem size of combination 1 and W^* is the scaled problem size of combination 2.

Proof: $t_{p'}(W') = T_{p'}(W^*)$ is a direct result of Proposition 2.

By Proposition 1, under the condition of Proposition 2 we have $\Phi(p,p') = \alpha\psi(p,p')$. Replace the scalability with relation (1), we get

$$\frac{p'W}{pW'} = \frac{\alpha p'W}{pW^*}.$$

Thus,

$$W^* = \alpha W',$$

and

$$T_{p'}(W') < T_{p'}(W^*) = t_{p'}(W').$$

□

A more general result is given by Theorem 4.

Theorem 4 *If algorithm-machine combination 1 has a larger execution time than algorithm-machine combination 2 at the initial state and the scaled ensemble size p' is not a scaled crossing point, then combination 1 has a larger execution time than that of combination 2 for solving W' at p', where W' is the scaled problem size of combination 1.*

Proof: We use the same notations as used in the proof of Proposition 1.

Since p' is not a scaled crossing point, by Theorem 3, $T_{p'}(W^*) \leq t_{p'}(W')$.

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

$$T_{p'}(W') < T_{p'}(W^*) \leq t_{p'}(W').$$

Case 2: If $W' \geq W^*$, then by the definition of isospeed scalability (1) we have the scalability of combination 1 is equal to or smaller than the scalability of combination 2, $\Phi(p,p') \leq \Psi(p,p')$. Thus combination 1 has a larger initial time and equal or smaller scalability, by Theorem 1, $T_{p'}(W') < t_{p'}(W')$.

□

Theorem 4 gives the necessary condition for equal-size performance crossing: for the initial ensemble size p, if p' is an equal-size crossing point of p it must be a scaled crossing point of p. In other words, if p' is not a scaled crossing point of p, it is not an equal-size crossing point of p. No performance crossing will occur before the scaled crossing point even in terms of equal-size performance. Theorem 4 provides the mean of range comparison. Based on the theoretical findings Figure 1 gives the range comparison algorithm.

`Assumption of the Algorithm:` Assume algorithm-machine combinations 1 and 2 have execution time αT and T respectively at the same initial state, where $\alpha > 1$.

`Objective of the Algorithm:` Predict if combination 2 is superior over the range of ensemble sizes from p to p', where $p' > p$.

```
Range Comparison
Begin
      Compute the Scalability of combination 1 $\Phi(p, p')$;
      Compute the Scalability of combination 2 $\Psi(p, p')$;
      If $\Phi((p, p') \leq \alpha\Psi(p, p')$ then
        Combination 2 is superior over the range $< p, p' >$;
      Else
        $p'$ is a performance crossing point
      End{If}
End{Range Comparison }
```

Fig. 1: Range Comparison Via Performance Crossing point

4 Experimental Testing

Three parallel tridiagonal solvers are used to confirm the analytical results. They are the PPT, PDD, and Reduced PDD algorithms. For the sake of brevity, the algorithms will not be re-introduced. Interested readers may refer to [5] and [7] for details of the algorithms and their corresponding scalability analyses. Only needed analytical and experimental results are presented here to confirm the range-comparison methodology.

The tridiagonal systems under consideration are diagonal dominant, symmetric, Toeplitz systems with multiple right sides. Both the PDD and Reduced PDD algorithms are ideally scalable with scalability equals one under our testing condition. The PPT algorithm is not perfect scalable. These three algorithms were implemented on an IBM SP2 and an Intel Paragon. Execution time is measured in seconds. Speed is given in MFLOPS (Milliones floating-point operation per second). Tables 1 through 3 list the measured results on the SP2 and Paragon machines. The measurement starts 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.

Observing the timing given in Tables 1 and 2, we can see that the measured result confirms the theoretical result. For instance, 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 Theorem 1 and 3. By Theorem 1, since the PPT algorithm is slow at the initial state on the Paragon machine, it is inferior over the computing

	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

Table 1: Measured Execution Time (in seconds) on the SP2 Machine

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 scalability analysis [7], the scalability of 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 Figure 1, the first performance crossing point is at ensemble size 4. This predicted 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 performance superiority/inferiority will change when system size increases. The crossing point can be predicted. In the PDD/PPT case, the performance crossing point is 4.

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 3 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 rearch the achieved average speed of the PDD algorithm with larger size (see Table 3). 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

	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

Table 2: Measured Execution Time (in seconds) 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
Speed/p	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
Speed/p	14.8375	14.8	14.7688	14.8469	14.7359

Table 3: Variation of the Reduced PDD Algorithm on the Paragon Machine

require two send-and-receive communications. Though 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 problem size is small. The PDD algorithm and Reduced PDD algorithm reach the same average speed at ensemble size equal four 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. Therefore, the α multiple of the scalability of PDD algorithm is less (not greater) than the scalability of the Reduced PDD algorithm. By Theorem 2, the execution time of the PDD algorithm on its scaled problem size should be greater (not smaller) than that of the Reduced PDD algorithm over the scalable computing range. Measured results given in Tables 3 and 2 confirm the theoretical statement.

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. Parallelized Thomas algorithm has less computing but more communication requirement than that of the PDD algorithm. Figure 2 and 3 demonstrate the performance range comparison of the PDD and parallelized

Thomas algorithm, when the computing and communication capacity varies, respectively [10]. 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 remains superior. However, communication capacity will change the superiority.

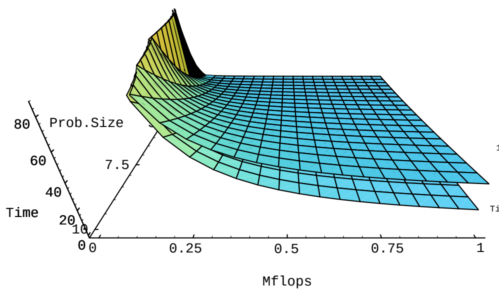

Fig. 2: Speed Influence

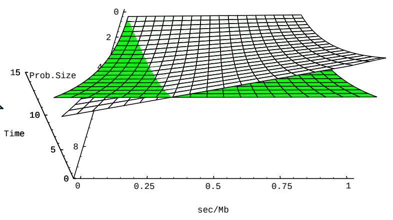

Fig. 3: Communication Influence

5 Conclusion

While scalability has been accepted as an important property of algorithms and architectures, execution time is the dominant metric of computing [4]. Scalability study would have little practical impact if it could not provide useful information on time variation in a scalable computing environment. The relation between scalability and execution time is identified in this study. The concepts of crossing point analysis and range comparison are proposed. A novel methodology is developed to compare/evaluate the performance of different parallel algorithms and architectures over a large range of system and problem size. Experimental and theoretical results show scalability is a unique indicator of time variation. An initially slow algorithm-machine combination can become superior if it has a better scalability. More importantly, the performance superior/inferior crossing point can be determined via scalability. Scalability makes range comparison possible. Range comparison, in which execution time is compared over a range of system and problem sizes, opens new ways of performance evaluation.

Acknowledgements

This research was supported in part by NSF under grant ASC-9720215, by NASA under contracts NAS1-1672, and by Louisiana Education Quality Support Fund.

References

1. Grama, A. Y., Gupta, A., and Kumar, V. Isoefficiency: Measuring the scalability of parallel algorithms and architectures. *IEEE Parallel & Distributed Technology 1*, 3 (Aug. 1993), 12–21.
2. Gustafson, J. Reevaluating Amdahl's law. *Communications of the ACM 31* (May 1988), 532–533.
3. Hwang, K. *Advanced Computer Architecture: Parallelism, Scalability, Programmability.* McGraw-Hill, 1993.
4. Sahni, S., and Thanvantri, V. Performance metrics: Keeping the focus on runtime. *IEEE Parallel & Distributed Technology* (Spring 1996), 43–56.
5. Sun, X.-H. Application and accuracy of the parallel diagonal dominant algorithm. *Parallel Computing* (Aug. 1995), 1241–1267.
6. Sun, X.-H. The relation of scalability and execution time. In *Proc. of the International Parallel Processing Symposium'96* (April 1996).
7. Sun, X.-H. Scalability versus execution time in scalable systems. Louisiana State Uiversity, Computer Science TR-97-003, 1997.
8. Sun, X.-H., and Rover, D. Scalability of parallel algorithm-machine combinations. *IEEE Transactions on Parallel and Distributed Systems* (June 1994), 599–613.
9. Sun, X.-H., and Zhu, J. Performance considerations of shared virtual memory machines. *IEEE Transactions on Parallel and Distributed Systems* (Nov. 1995), 1185–1194.
10. Sun, X.-H., and Zhu, J. Performance prediction: A case study using a scalable shared-virtual-memory machine. *IEEE Parallel & Distributed Technology* (Winter 1996), 36–49.