# The Relation of Scalability and Execution Time*

*Xian-He Sun*

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

**Abstract**

Scalability has been used extensively as a de facto performance criterion for evaluating parallel algorithms and architectures. However, for many, scalability has theoretical interests only since it does not reveal execution time. In this paper, the relation between scalability and execution time is carefully studied. Results show that the isospeed scalability well characterizes the variation of execution time: smaller scalability leads to larger execution time, the same scalability leads to the same execution time, etc. Three algorithms from scientific computing are implemented on an Intel Paragon and an IBM SP2 parallel computer. Experimental and theoretical results show that scalability is an important, distinct metric for parallel and distributed systems, and may be as important as execution time in a scalable parallel and distributed environment.

* This research was supported in part by the National Aeronautics and Space Administration under NASA contract NAS1-19480 and NAS1-1672 while the author was in residence at the Institute for Computer Applications in Science and Engineering (ICASE), NASA Langley Research Center, Hampton, VA 23681-0001, and by Louisiana Education Quality Support Fund.

# 1 Introduction

Parallel computers, such as the Intel Paragon, IBM SP2, and Cray T3D, have been successfully used in solving certain of the so-called "grand-challenge" applications. However, despite initial success, parallel machines have not been widely accepted in production engineering environment. Reasons for the limited acceptance include lack of program portability, lack of suitable performance metrics, and the two-to-three year gap in technology between the microprocessors used on parallel computers and on serial computers, due to the long design process of parallel computers. Appropriate performance metrics are essential for providing a general guide-line of efficient parallel algorithm and architecture design, for finding bottlenecks of a given algorithm-architecture combination, and for choosing an optimal algorithm-machine pair for a given application.

In sequential computing, an algorithm is well characterized in terms of work, which is measured in terms of operation count and memory requirement. Assuming sufficient memory available, execution time of a sequential algorithm is proportional to work performed. This simple relation between time and work makes performance of sequential algorithms easily understood, compared, and predicted. While problem size[1] and memory requirement remain as essential factors in parallel computing, another two parameters, communication overhead and load balance, enter the complexity of 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. The decrease of load balance and increase of communication overhead may reduce the performance considerably and lead to a much longer execution time than the peak performance when problem and system size increase. Large parallel systems are very difficult to use efficiently for solving small problems. The well-known Amdahl's law [1] shows a limitation of parallel processing due to insufficient parallel work, when problem size does not increase with ensemble size. Large parallel systems are designed for solving large problems. The concept of scalable computing has been proposed [2] in which problem size scales up with ensemble size, and is well accepted. With the scaled problem size, however, time is no longer an appropriate metric to evaluate performance between a small and a larger parallel system. In addition to time, scalability has emerged as a key measurement of scalable computing.

Intensive research has been conducted in recent years in scalability. The commonly used parallel performance metric, speedup, defined as sequential processing time over parallel processing time, has been extended for scalable computing. Scaled speedups such as *fixed-time speedup* [3], *memory-bounded speedup* [4], and *generalized speedup* [3] have been proposed for different scaling constraints. Simply speaking, the scalability of an Algorithm-Machine Combination (AMC) is the ability of the AMC to deliver high performance power when system and problem sizes are large. Depending

[1] 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.

on how the performance power is defined and measured, different scalability metrics have been proposed and have been used regularly in evaluating parallel algorithms and architectures [5, 6, 7, 8, 9].

Execution time is the ultimate measure of interest in practice. Scalability, however, has been traditionally studied separately. Its relation to time has not been revealed. For this reason, though scalability measurement has been used extensively in the parallel processing community, some scientists consider it only of theoretical interests. In this paper, we carefully study the relation between scalability and time. We show that the isospeed scalability characterizes execution time well. If two AMCs have the same execution time at an initial scale, then the AMC with smaller scalability has a larger execution time for the scaled problem. (This is also true if the AMC with the smaller scalability has a larger initial time.) If two AMCs have the same scalability, then smaller initial time leads to a smaller execution time for scaled problems. Since the relation between isospeed scalability and other scalabilities has been studied [10], results presented in this paper can be extended to other scalabilities as well.

In Section 2, we first review the isospeed scalability and, then, present the main results of the study, the relation between scalability and execution time. We introduce three parallel algorithms, the Parallel Partition LU (PPT), the Parallel Diagonal Dominant (PDD), and the Reduced Parallel Diagonal Dominant (RPDD) algorithms, in Section 3. Comparison and scalability analysis of the three algorithms are also performed. Experimental results of the three algorithms on an Intel Paragon and on an IBM SP2 are presented in Section 4 to confirm our findings. Section 5 summarizes the work.

## 2 Isospeed Scalability and Its Relation with Time

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. 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. The average unit speed (or average speed, in short) is a good measure of parallel processing speed.

**Definition 1** *The* `average unit speed` *is the achieved speed of the given computing system divided by p, the number of processors.*

Theoretical peak performance is usually based on ideal situation where the average speed remains constant when system size increase. If problem size is fixed, however, the ideal situation is unlikely to happen, since the communication/computation ratio typically increases with the number of processors, and therefore, the average unit speed will decrease with increased system size. On the other hand, if system size is fixed, communication/computation ratio is likely to decrease

with increased problem size for most practical algorithms. For these algorithms, increasing problem size with the system size may keep the average speed constant. The *isospeed scalability* has been formally defined as the ability to maintain the average speed in [7] based on this observation.

**Definition 2** *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 [7]. 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. It 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 [7]:

$$\psi(p, p') = \frac{T(p, W)}{T(p', W')} \tag{2}$$

where $T(p', W')$ is the corresponding execution time of solving $W'$ on $p'$ processors.

Execution time is the ultimate measure of parallel processing. In Theorem 1 and 2, we show that isospeed scalability favors systems with better run time and characterizes the run time well when problem size scales up with system size.

**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 $\alpha$ multiple of the execution time of combination 1 is smaller than the execution time of combination 2 for solving $W'$, where $W'$ is the scaled problem size of combination 1.*

**Proof:** Let $t$, $T$ be the execution time of algorithm-machine combinations 1 and 2, respectively. Since combinations 1 and 2 have the relation $t(p, W) = \alpha \cdot T(p, W)$ for the initial problem

size $W$ at ensemble size $p$,

$$\alpha \cdot a = \frac{W}{p \cdot t(p, W)} \quad and \quad a = \frac{W}{p \cdot T(p, W)}.$$

Let $W'$ and $W^*$ be the scaled problem sizes to maintain the initial average speed of combinations 1 and 2, respectively. We have

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

Therefore,

$$\frac{\alpha \cdot t(p', W')}{T(p', W^*)} = \frac{W'}{W^*}. \tag{3}$$

Let $a'$ be the achieved average speed of combination 2 with the scaled problem size $W'$. $a' = \frac{W'}{p' \cdot T(p', W')}$. By Eq. (3),

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

Thus,

$$\frac{T(p', W')}{\alpha \cdot t(p', W')} = \frac{a}{a'}. \tag{4}$$

Let $\Phi$, $\Psi$ be the scalability of AMC 1 and 2 respectively. By the definition of isospeed scalability,

$$\Phi(p, p') = \frac{p' \cdot W}{p \cdot W'} \tag{5}$$

and

$$\Psi(p, p') = \frac{p' \cdot W}{p \cdot W^*}. \tag{6}$$

Equations (5) and (6) show that $\Psi(p, p') < \Phi(p, p')$ if and only if $W^* > W'$. Under the general assumption that the speed increases with the problem size, $a > a'$ if and only if $W^* > W'$. By Eq. (4), $T(p', W') > \alpha \cdot t(p', W')$ if and only if $a > a'$. Combining these three if and only if conditions, we have

$$\Psi(p, p') < \Phi(p, p') \quad \text{if and only if} \quad \alpha \cdot t(p', W') < T(p', W'),$$

which proves the theorem. □

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

**Corollary 1** *If algorithm-machine combinations 1 and 2 have the same performance at the same initial state, then combination 1 has a higher scalability than that 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.*

**Proof:** Take $\alpha = 1$ in Theorem 1. □

**Corollary 2** *If algorithm-machine combinations 1 and 2 have execution time $\alpha \cdot T$ and $T$, respectively, at the same initial state, then combination 1 and 2 have the same scalability if and only if the $\alpha$ multiple of the execution time of combination 1 is equal to the execution time of combination 2 for solving $W'$, where $W'$ is the scaled problem size of combination 1.*

**Proof:** Similar to the proof of Theorem 1. The only difference is that, from Eqs. (5) and (6), combinations 1 and 2 have the same scalability if and only if $W' = W^*$. By Eq. (4) and the definition of $a'$, $W' = W^*$ if and only if $\alpha \cdot t(p', W') = T(p', W')$. □

Corollary 3 is a direct result of Corollary 1 and 2.

**Corollary 3** *If algorithm-machine combinations 1 and 2 have the same performance at the same initial state, then combinations 1 and 2 have the same scalability if and only if combinations 1 and 2 have the same execution time for solving $W'$, where $W'$ is the scaled problem size of combination 1.*

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 [7]. 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 $\alpha$ 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.*

**Proof:** We define $a$, $a'$, $W'$, $W^*$, $t$, $T$, $p$, and $p'$ similarly as in Theorem 1. We let $W$ be the initial problem size of combination 1. By the given condition,

$$a = \frac{W}{p \cdot t(p, W)}, \qquad a = \frac{\alpha \cdot W}{p \cdot T(p, \alpha \cdot W)},$$

and

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

Thus,

$$\frac{\alpha \cdot t(p', W')}{T(p', W^*)} = \frac{W'}{W^*}. \tag{7}$$

Following a similar deduction as that used in deriving Eq. (4), we have

$$\frac{T(p', W')}{t(p', W')} = \frac{a}{a'.} \tag{8}$$

Let $\Phi$, $\Psi$ be the scalability of AMC 1 and 2 respectively. By the definition of isospeed scalability,

$$\Phi(p, p') = \frac{p' \cdot W}{p \cdot W'}$$

and

$$\Psi(p, p') = \frac{p' \cdot \alpha \cdot W}{p \cdot W^*}.$$

Thus,

$$\frac{\alpha \cdot \Phi(p, p')}{\Psi(p, p')} = \frac{W^*}{W'}. \tag{9}$$

By Eq. (9), $\alpha \cdot \Phi(p, p') > \Psi(p, p')$ if and only if $W^* > W'$. Under the general assumption that speed increases with problem size, $W^* > W'$ if and only if $a > a'$. By Eq. (8), $T(p', W') > \alpha \cdot t(p', W')$ if and only if $a > a'$. We have obtained the desired result:

$$\alpha \cdot \Psi(p, p') < \Phi(p, p') \quad \text{if and only if} \quad t(p', W') < T(p', W').$$

□

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

**Corollary 4** *If algorithm-machine combinations 1 and 2 achieve the same average speed with problem size $W$ and $\alpha \cdot W$, respectively, at the initial ensemble size, then the $\alpha$ multiple of the scalability of combination 1 is the same as the scalability of combination 2 if and only if combination 1 has the same execution time as that of combination 2 for solving $W'$, where $W'$ is the scaled problem size of combination 1.*

**Proof:** Similar to the proof of Corollary 3. □

# 3 Tridiagonal Solvers: A case study

Solving tridiagonal systems is one of the key issues in scientific computing [11]. Many methods used for the solution of partial differential equations (PDEs) rely on solving a sequence of tridiagonal systems. In addition to PDE's, tridiagonal systems also arise in many other applications [12]. Three parallel tridiagonal solvers, which are used to confirm the analytical results, are introduced in the following four sectons. Interested readers may refer to [13] and [12] for details of the algorithms, especially for accuracy analysis and for extending these algorithms for solving periodic systems and for general banded linear systems.

## 3.1 A Partition Method for Parallel Processing

A tridiagonal system is a linear system of equations

$$Ax = d, \tag{10}$$

where $x = (x_1, \cdots, x_n)^T$ and $d = (d_1 \cdots, d_n)^T$ are $n$-dimensional vectors, and A is a tridiagonal matrix of order $n$:

$$A = \begin{bmatrix} b_0 & c_0 & & & & & \\ a_1 & b_1 & c_1 & & & & \\ & \cdot & \cdot & \cdot & & & \\ & & & \cdot & \cdot & \cdot & \\ & & & a_{n-2} & b_{n-2} & c_{n-2} \\ & & & & a_{n-1} & b_{n-1} \end{bmatrix} = [a_i, b_i, c_i]. \tag{11}$$

To solve Eq. (10) efficiently on parallel computers, we partition $A$ into submatrices. For convenience we assume that $n = p \cdot m$, where $p$ is the number of processors available. The matrix $A$ in Eq. (10) can be written as

$$A = \tilde{A} + \Delta A,$$

where $\tilde{A}$ is a block diagonal matrix with diagonal submatrices $A_i (i = 0, \cdots, p-1)$. The submatrices $A_i (i = 0, \cdots, p-1)$ are $m \times m$ tridiagonal matrices. Let $e_i$ be a column vector with its $i$th

$(0 \le i \le n-1)$ element being one and all the other entries being zero. We have

$$\Delta A = [a_m e_m, c_{m-1} e_{m-1}, a_{2m} e_{2m}, c_{2m-1} e_{2m-1}, \cdots, c_{(p-1)m-1} e_{(p-1)m-1}] \begin{bmatrix} e_{m-1}^T \\ e_m^T \\ \cdot \\ \cdot \\ e_{(p-1)m-1}^T \\ e_{(p-1)m}^T \end{bmatrix} = V E^T,$$

where both $V$ and $E$ are $n \times 2(p-1)$ matrices. Thus, we have

$$A = \tilde{A} + V E^T.$$

Based on the matrix modification formula originally defined by Sherman and Morrison [14] for rank-one changes, and the assumption that all $A_i$'s are invertible, Eq. (10) can be solved by

$$x = A^{-1} d = (\tilde{A} + V E^T)^{-1} d, \tag{12}$$

$$x = \tilde{A}^{-1} d - \tilde{A}^{-1} V (I + E^T \tilde{A}^{-1} V)^{-1} E^T \tilde{A}^{-1} d. \tag{13}$$

Let

$$\tilde{A}\tilde{x} = d \tag{14}$$

$$\tilde{A} Y = V \tag{15}$$

$$h = E^T \tilde{x} \tag{16}$$

$$Z = I + E^T Y \tag{17}$$

$$Z y = h \tag{18}$$

$$\Delta x = Y y. \tag{19}$$

Equation (13) becomes

$$x = \tilde{x} - \Delta x. \tag{20}$$

In Eqs. (14) and (15), $\tilde{x}$ and $Y$ are solved by the LU decomposition method. Based on the structure of $\tilde{A}$ and $V$, this is equivalent to solving

$$A_i [\tilde{x}^{(i)}, v^{(i)}, w^{(i)}] = [d^{(i)}, a_{im} e_0, c_{(i+1)m-1} e_{m-1}], \tag{21}$$

$i = 0, \cdots, p-1$. Here $\tilde{x}^{(i)}$ and $d^{(i)}$ are the $i$th block of $\tilde{x}$ and $d$, respectively, and $v^{(i)}, w^{(i)}$ are possible nonzero column vectors of the $i$th row block of $Y$. Equation (21) implies that we only need to solve

three linear systems of order $m$ with the same LU decomposition for each $i$ ($i = 0, \cdots, p-1$).

Solving Eq. (18) is the major computation involved in the conquer part of our algorithms. Different approaches have been proposed for solving Eq. (18), which results in different algorithms for solving tridiagonal systems [13].

### 3.2 The Parallel Partition LU (PPT) Algorithm

Based on the matrix partitioning technique described previously, using $p$ processors, the PPT algorithm to solve (10) consists of the following steps:

Step 1. Allocate $A_i, d^{(i)}$, and elements $a_{im}, c_{(i+1)m-1}$ to the $i$th node, where $0 \leq i \leq p-1$.

Step 2. Solve (21). All computations can be executed in parallel on $p$ processors.

Step 3. Send $\tilde{x}_0^{(i)}, \tilde{x}_{m-1}^{(i)}, v_{m-1}^{(i)}, v_0^{(i)}, w_{m-1}^{(i)}$, and $w_0^{(i)}$ to all the other nodes from the $i$th node to form the matrix $Z$ and vector $h$ (see Eqs. (16) and (17)) on each node. Here and throughout the subindex indicates the component of the vector.

Step 4. Use the LU decomposition method to solve $Zy = h$ (see Eq. (18)) on all nodes simultaneously. Note that $Z$ is a $2(p-1)$ dimensional tridiagonal matrix.

Step 5. Compute (19) and (20). We have

$$\Delta x^{(i)} = [v^{(i)}, w^{(i)}] \begin{pmatrix} y_{(2i-1)} \\ y_{2i} \end{pmatrix}$$

$$x^{(i)} = \tilde{x}^{(i)} - \Delta x^{(i)}$$

Step 3 requires a global total-data-exchange communication[2].

### 3.3 The Parallel Diagonal Dominant (PDD) Algorithm

The matrix $Z$ in Eq. (18) has the form

[2] The all-to-all global communication can be replaced by one data-gathering communication plus one data-scattering communication. However, on most communication topologies (including 2-D mesh, multi-stage Omega network, and hypercube), the latter has a higher communication cost than the former [6].

$$Z = \begin{bmatrix} 1 & w_{m-1}^{(0)} & 0 & & & & & \\ v_0^1 & 1 & 0 & w_0^{(1)} & & & & \\ v_{m-1}^{(1)} & 0 & 1 & w_{m-1}^{(1)} & 0 & & & \\ & \cdot & \cdot & \cdot & \cdot & \cdot & & \\ & & \cdot & \cdot & \cdot & \cdot & \cdot & \\ & & & \cdot & \cdot & 1 & 0 & w_0^{(p-2)} \\ & & & & v_{m-1}^{(p-2)} & 0 & 1 & w_{m-1}^{(p-2)} \\ & & & & & 0 & v_0^{(p-1)} & 1 \end{bmatrix}$$

In practice, for a diagonally dominant tridiagonal system, the magnitude of the last component of $v^{(i)}$, $v_{m-1}^{(i)}$, and the first component of $w^{(i)}$, $w_0^{(i)}$, may be smaller than machine accuracy when $p \ll n$. In this case, $w_0^{(i)}$ and $v_{m-1}^{(i)}$ can be dropped, and $Z$ becomes a diagonal block system consisting of $(p-1)$ $2 \times 2$ independent blocks. Thus, Eq.(18) can be solved efficiently on parallel computers, which leads to the highly efficient *parallel diagonal dominant* (PDD) algorithm [13].

Using $p$ processors, the PDD algorithm consists of the following steps:

Step 1. Allocate $A_i, d^{(i)}$, and elements $a_{im}, c_{(i+1)m-1}$ to the $i$th node, where $0 \le i \le p-1$.

Step 2. Solve (21). All computations can be executed in parallel on $p$ processors.

Step 3. Send $\tilde{x}_0^{(i)}, v_0^{(i)}$ from the $i$th node to the $(i-1)$th node, for $i = 1, \cdots, p-1$.

Step 4. Solve

$$\begin{bmatrix} 1 & w_{m-1}^{(i)} \\ v_0^{(i+1)} & 1 \end{bmatrix} \begin{pmatrix} y_{2i} \\ y_{2i+1} \end{pmatrix} = \begin{pmatrix} \tilde{x}_{m-1}^{(i)} \\ \tilde{x}_0^{(i+1)} \end{pmatrix}$$

in parallel on the $i$th node for $0 \le i \le p-2$. Then send $y_{2i}$ from the $i$th node to the $(i+1)$th node, for $i = 0, \cdots, p-2$.

Step 5. Compute (19) and (20). We have

$$\Delta x^{(i)} = [v^{(i)}, w^{(i)}] \begin{pmatrix} y_{(2i-1)} \\ y_{2i} \end{pmatrix},$$

$$x^{(i)} = \tilde{x}^{(i)} - \Delta x^{(i)}.$$

In all of these, there are only two neighboring communications.

### 3.4 The Reduced PDD Algorithm

The PDD algorithm is very efficient in communication. However, the PDD algorithm has a larger operation count than the conventional sequential algorithm, the Thomas algorithm [15]. The Reduced PDD algorithm is proposed in order to further enhance computation [12].

In the last step, Step 5, of the PDD algorithm, the final solution, $x$, is computed by combining the intermediate results concurrently on each processor,

$$x^{(k)} = \tilde{x}^{(k)} - y_{(2k-1)} v^{(k)} - y_{2k} w^{(k)},$$

which requires $4(n-1)$ sequential operations and $4m$ parallel operations, if $p = n/m$ processors are used. The PDD algorithm drops the first element of $w, w_0$, and the last element of $v$, $v_{m-1}$, in solving Eq. (18). In [12], we have shown that, for symmetric Toeplitz tridiagonal systems, when $m$ is large enough, we may drop $v_i, i = j, j+1, \cdots, m-1$, and $w_i, i = 0, 1, \cdots, j-1$, for some integer $j > 0$, while maintaining the required accuracy. If we replace $v_i$ by $\tilde{v}_i$, where $\tilde{v}_i = v_i$, for $i = 0, 1, \cdots, j-1$, $\tilde{v}_i = 0$, for $i = j, \cdots, m-1$; and replace $w$ by $\tilde{w}$, where $\tilde{w}_i = w_i$ for $i = j, \cdots, m-1$, and $\tilde{w}_i = 0$, for $i = 0, 1, \cdots, j-1$; and use $\tilde{v}, \tilde{w}$ in Step 5, we have

Step 5'

$$\Delta x^{(k)} = [\tilde{v}, \tilde{w}] \begin{pmatrix} y_{(2k-1)} \\ y_{2k} \end{pmatrix},$$

$$x^{(k)} = \tilde{x}^{(k)} - \Delta x^{(k)}. \tag{22}$$

This requires only $4j/p$ parallel operations. Replacing Step 5 of the PDD algorithm by Step 5', we get the Reduced PDD algorithm [12]. In general, $j$ is quite small. For instance, when error tolerance $\epsilon$ equals $10^{-4}$, $j$ equals either 10 or 7 when $\lambda$, the magnitude of the off diagonal elements, equals $\frac{1}{3}$ or $\frac{1}{4}$ respectively, the diagonal elements being equal to 1. The integer $j$ reduces to 4 for $0 < \lambda \leq \frac{1}{9}$.

### 3.5 Operation Comparison

Table 1 gives the computation and communication count of the tridiagonal solvers under consideration. 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 matrix $\tilde{A}$ and computation of $Y$ are not considered (see Eqs. (14) and (15) in Section 3.1). Parameter $n_1$ is the number of right-hand-sides. Note that for multiple right-side 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. The computational saving of the Reduced PDD algorithm is not only in step 5, the final modification step, but in other steps as well. Since we only need

| System | Algorithm | Computation | Communication |
|---|---|---|---|
| Single system | best sequential | $8n-7$ | 0 |
| | PPT | $17\frac{n}{p}+16p-23$ | $(2\alpha+8p\beta)(\sqrt{p}-1)$ |
| | PDD | $17\frac{n}{p}-4$ | $2\alpha+12\beta$ |
| | Reduced PDD | $11\frac{n}{p}+6j-4$ | $2\alpha+12\beta$ |
| Multiple right sides | best sequential | $(5n-3)\cdot n_1$ | 0 |
| | PPT | $(9\frac{n}{p}+10p-11)\cdot n_1$ | $(2\alpha+8p\cdot n_1\cdot\beta)(\sqrt{p}-1)$ |
| | PDD | $(9\frac{n}{p}+1)\cdot n_1$ | $(2\alpha+8n_1\cdot\beta)$ |
| | Reduced PDD | $(5\frac{n}{p}+4j+1)\cdot n_1$ | $(2\alpha+8n_1\cdot\beta)$ |

Table 1. Comparison of Computation and Communication

$j$ elements of vector $v$ and $w$ for the final modification in the Reduced PDD algorithm (see Eq. (22) in Section 3), we only need to compute $j$ elements for each column of $V$ in solving Eq. (15). Formulas for computing the integer $j$ can be found in [12] depending on particular circumstances. The listed sequential operation count is based on Thomas algorithm.

Communication cost has a 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 $\alpha + S\beta$, where $\alpha$ is a fixed startup time and $\beta$ 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 $\alpha+8\beta$ and $\alpha+4\beta$ time respectively on any architecture which 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) [16]. If a hypercube topology or a multi-stage Omega network is assumed the communication cost would be $\log(p)\alpha + 12(p-1)\beta$ and $\log(p)\alpha + 8(p-1)n_1\cdot\beta$ for single systems and systems with multiple right sides respectively [13, 17].

### 3.6 Scalability Analysis

The scalability analysis of the PDD algorithm for solving single systems can be found in [12]. 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 are also presented under the same assumption.

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 (see definitions (1) and (2)) would be that when both the number of processors and the amount of work are scaled up by a factor of $N$, the execution time remains unchanged:

$$T(N \times p, N \times W) = T(p, W) \tag{23}$$

To eliminate 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

$$W' = (N \times 5n) \cdot n_1 = (5n') \cdot n_1, \tag{24}$$

$$n' = N \cdot n. \tag{25}$$

**The PDD Algorithm**

Let $\tau_{comp}$ represent the unit of a computation operation normalized to the communication time. The time required to solve (10) by the PDD algorithm with $p$ processors is

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

and

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

Thus the PDD algorithm is perfectly scalable. Its scalability equals one under our assumption. Notice that in the above analysis we assume $T(p, W)$ contains the communication cost. The perfect scalability may 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 a smaller operation count 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 under our assumption.

**The PPT Algorithm**

The PPT algorithm is not perfectly scalable. Its scalability analysis needs more discussions. The

following prediction formula is needed for the scalability analysis ([18], Eq.(2)):

$$W' = \frac{ap'T'_o}{1 - a\tau}, \tag{26}$$

where $W'$ and $p'$ are as defined in (1), $a$ is the fixed average speed, $\tau$ is the substained computing rate (reciprocal of speed) of a single processor, and $T'_o$ is the parallel processing overhead. Parameters $a$ and $\tau$ do not vary with the number of processors. For a given AMC and a given initial average speed, $\frac{a}{1-a\tau}$ is a constant number. Therefore, Eq. (26) can also be written as:

$$W' = cp'T'_o \tag{27}$$

The computing time can be represented as

$$T(p,n) = T_C(p,n) + T_o(p,n), \tag{28}$$

where $T_C(p,n)$ 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[3])

$$T(p,n) = (9\frac{n}{p} + 10p) \cdot n_1 \cdot \tau_{comp} + (2\alpha + 8 \cdot n_1 \cdot p \cdot \beta)(\sqrt{p} - 1). \tag{29}$$

By Eq. (24),

$$T_C(p,n) = \frac{5n}{p} \cdot n_1 \cdot \tau_{comp}. \tag{30}$$

Therefore,

$$T_o(p,n) = (4\frac{n}{p} + 10p) \cdot n_1 \cdot \tau_{comp} + (2\alpha + 8 \cdot n_1 \cdot p \cdot \beta)(\sqrt{p} - 1). \tag{31}$$

Using the prediction formula (27), we have

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

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

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

which eventually leads to

$$n' = c'[10p'^2 \cdot n_1 \cdot \tau_{comp} + (2\alpha p' + 8 \cdot n_1 \cdot p'^2 \cdot \beta)(\sqrt{p'} - 1)], \tag{32}$$

[3]The constant number 11 is eliminated for convenience, since it is independent of parameter $n$ and $p$.

where $c' = \frac{c}{(5-4c)\cdot n_1 \cdot \tau_{comp}}$. Equation (32) is true for any work-processor pair which maintains the fixed average speed. In particular:

$$n = c'[10p^2 \cdot n_1 \cdot \tau_{comp} + (2\alpha p + 8 \cdot n_1 \cdot p^2 \cdot \beta)(\sqrt{p} - 1)] \tag{33}$$

Combining equations (32) and (33), we have

$$\begin{aligned} n' - n &= c'[10 \cdot n_1 \cdot (p'^2 - p^2) \cdot \tau_{comp} + 2\alpha(p'^{3/2} - p^{3/2}) + & (34) \\ & \quad 8 \cdot n_1 \cdot \beta(p'^{5/2} - p^{5/2}) - 2\alpha(p' - p) - 8 \cdot n_1 \cdot \beta(p'^2 - p^2)]. & (35) \end{aligned}$$

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

$$n' - n \approx 2c' \cdot \alpha \cdot (p'^{3/2} - p^{3/2}), \tag{36}$$

which shows that the variation of $n$ is in direct proportion to the $3/2$ power of the variation of ensembles size. By Eq. (24), $W$, the work, is in direct proportion to the order of matrix $n$, therefore, the scalability of this AMC can be estimated as

$$\psi(p, p') = \psi(p, Np) = \frac{NpW}{pW'} \approx \frac{N \cdot W}{N^{3/2}W} = \frac{1}{\sqrt{N}}. \tag{37}$$

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

$$n' - n \approx 10c' \cdot n_1 \cdot (p'^2 - p^2) \cdot \tau_{comp}, \tag{38}$$

and

$$\psi(p, p') = \psi(p, Np) = \frac{NpW}{pW'} \approx \frac{N \cdot W}{N^2 W} = \frac{1}{N}; \tag{39}$$

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

$$n' - n \approx 8c' \cdot n_1 \cdot \beta(p'^{5/2} - p^{5/2}), \tag{40}$$

and

$$\psi(p, p') = \psi(p, Np) = \frac{NpW}{pW'} \approx \frac{N \cdot W}{N^{5/2}W} = \frac{1}{\sqrt{N^3}}. \tag{41}$$

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.

# 4 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 a RISC integer core processing unit and three separate on-chip caches for page translation, data, and instructions. The heart of all distributed memory parallel computers is the interconnection network that links the processors together. The SP2 High-Performance Switch is a multi-stage 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 microseconds and bandwidth is 35 Mbytes per second. For Paragon, the measured latency is 46 microseconds and 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

$$A = \begin{bmatrix} 1 & \frac{1}{3} & & & & & \\ \frac{1}{3} & 1 & \frac{1}{3} & & & & \\ & \cdot & \cdot & \cdot & & & \\ & & \cdot & \cdot & \cdot & & \\ & & & \cdot & \cdot & \cdot & \\ & & & & \frac{1}{3} & 1 & \frac{1}{3} \\ & & & & & \frac{1}{3} & 1 \end{bmatrix} = [\frac{1}{3}, 1, \frac{1}{3}] \tag{42}$$

arising in CFD applications [12]. $j = 17$ has be chosen for the Reduced PDD algorithm to reach the single precision accuracy, $10^{-7}$.

Since execution time varies with communication/computation ratio on a parallel machine, the problem size is an important factor in performance evaluation, especially for machines supporting virtual memory. As studied in [10], a good choice of initial problem size is the problem size which reaches an appropriate portion of the asymptotic speed, the substained uniprocessor speed corresponding to main memory access [10]. 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, the order of matrix for SP2 has been chosen to be 6400 and the order of matrix for Paragon has been chosen to be 1600 for uniprocessor processing. Execution time is measured in seconds. Speed is

| | 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 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 | 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 3. Measured Average Speed on the SP2 Machine

given in MFLOPS (Millions floating-point operation per seconds). Tables 2 to 6 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. From Tables 2 and 4, we can see that the execution time of the PDD and Reduced PDD remain unchanged, except some minor measuring perturbations, when the order of matrix double with the number of processors. Since problem size increase linearly with the order of matrix for our applications, the constant timing indicts that the PDD and Reduced PDD algorithm 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, PDD-SP2, PDD-Paragon, RPDD-SP2, and RPDD-Paragon, are perfectly scalable, with scalability equals 1.

Since the PDD and Reduced PDD algorithms have the same scalability, these two algorithms satisfy the condition of Corollary 2. Their performance can be used to verify this corollary. Observing the timing given in Tables 2 and 4, we can see that the measured result confirms the theoretical result. For instance, based on Table 4, the initial timing ratio between the PDD and the Reduced PDD algorithm, $\alpha$, remains unchanged when the problem size is scaled up with the ensemble size. 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 Theorem 1. By Theorem 1, the timing difference between the PPT algorithm and the PDD

| | 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 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. | 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 5. Measured Average Speed on the Paragon Machine

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 both SP2 and Paragon machines.

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 achieve 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 theses algorithms 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 send and receive may lead to noticeable performance difference when problem size is small. The PDD algorithm and Reduced PDD algorithm reached 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 $\alpha$ 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 sizes should be greater (not smaller) than that of the Reduced PDD algorithm. 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

| | 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 6. Variation of the Reduced PDD Algorithm on the Paragon Machine

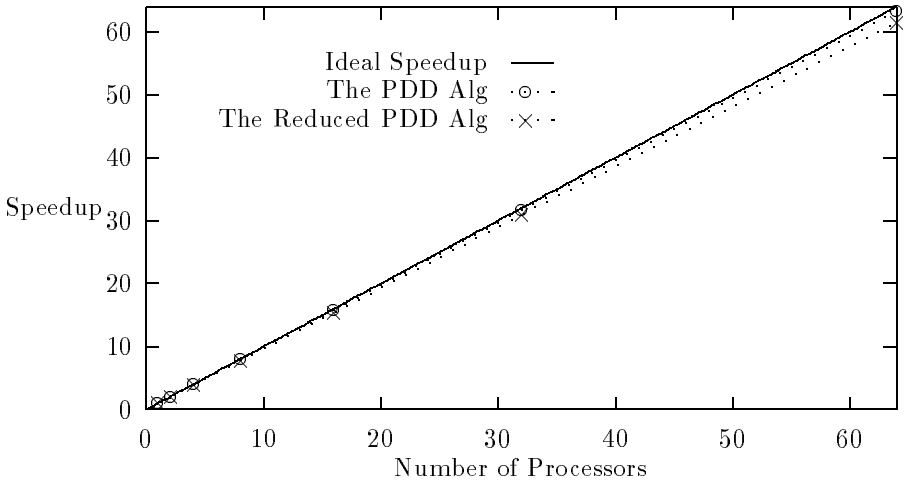


Figure 1. Measured Scaled Speedup on Intel Paragon
*1024 System of Order 1600*

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 that on the Paragon. This means the PPT algorithm has a better scalability on the SP2 than on the Paragon. The better scalability may due to various reasons, principally a larger memory and a more efficient all-to-all communication subroutine available on the SP2. Interested readers may refer to [17] 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.

It is known that constant scalability will lead to linear scalable-speedup (either in fixed-time or memory-bounded speedup) (Theorem 1 in [10]). Figure 1 shows the scaled speedup curves of the PDD and Reduced PDD algorithm on Paragon based on the measured data given in Table 4. To avoid inefficient uniprocessor processing on very large problem size [10], the sequential timing used in Figure 1 is predicted based on order of matrix equals 1600. From Figure 1 we can see that the two algorithms scale well. The speedup curves are a little below the ideal speedup, due to the communication that is not needed for uniprocessor processing. The speedup of the Reduced PDD algorithm is a little lower than the PDD algorithm, because the Reduced PDD algorithm has less computation and, therefore, a larger communication/computation ratio than the PDD algorithm.

# 5 Conclusion

Among other differences, speedup measures the parallel processing gain over sequential processing, scalability measures performance gain of large parallel system over small parallel system; speedup measures the final performance gain (usually in the form of time reduction), scalability measures the ability of an algorithm-machine combination in maintaining uniprocessor utilization. Scalability is a distinct metric for scalable computing.

While scalability has been widely used as an important property in analyzing algorithms and architectures, execution time is the dominant metric of parallel processing [9]. 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 revealed in this study. Experimental and theoretical results show scalability is a good indicator of time variation when problem and system size scale up. For any pair of algorithm-machine combinations which have the same initial execution time, an AMC has a smaller scalability if and only if it has a larger execution time on scaled problems; the same scalability will lead to the same execution time, and vice versa. The relation is also extendible to more general situations where the two AMCs have different initial execution times. Scalability is an important companion and complement of execution time. Initial time and scalability together will describe the expected performance on

large systems.

Isospeed scalability is a dimensionless scalar. It is easy to understand and is independent of sequential processing. When an initial speed is chosen, average speed is independent of problem size, system size, and sequential processing. In other word, it is dimensionless [19]. Therefore, scalability shows the inherited scaling characteristics of AMCs. Two AMCs are said to be computational similar if they have the same scalability over a range. Two similar algorithms, the PDD algorithm and the Reduced PDD algorithm, are carefully examed in this study. We have shown, theoretically and experimentally, that the two algorithms have the same computation and communication structure and have the same scalability. A third algorithm, the PPT algorithm is also studied to show that different communication structure will lead to different scalability, and that machine parameters may influence the scalability considerably. Scalability and scaling similarity are very important in evaluation, benchmarking, and comparison of parallel algorithms and architectures. They have practical importance in performance debugging, compiler optimization, and selection of an optimal algorithm/machine pair for an application. Current understanding of scalability is very limited. This study is an attempt to lead to a better understanding of scalability and its practical impact.

## Acknowledgment

The author is grateful to J. Zhu of Mississippi State University and D. Keyes of ICASE for their valuable suggestions and comments that helped improve the presentation of the paper and grateful to S. Moitra of NASA Langley Research Center in help gathering the performance data on SP2.

## References

[1] G. Amdahl, "Validity of the single-processor approach to achieving large scale computing capabilities," in *Proc. AFIPS Conf.*, pp. 483–485, 1967.

[2] J. Gustafson, G. Montry, and R. Benner, "Development of parallel methods for a 1024-processor hypercube," *SIAM J. of Sci. and Stat. Computing*, vol. 9, pp. 609–638, July 1988.

[3] J. Gustafson, "Reevaluating Amdahl's law," *Communications of the ACM*, vol. 31, pp. 532–533, May 1988.

[4] X.-H. Sun and L. Ni, "Scalable problems and memory-bounded speedup," *J. of Parallel and Distributed Computing*, vol. 19, pp. 27–37, Sept. 1993.

[5] A. Y. Grama, A. Gupta, and V. Kumar, "Isoefficiency: Measuring the scalability of parallel algorithms and architectures," *IEEE Parallel & Distributed Technoloty*, vol. 1, pp. 12–21, Aug. 1993.

[6] K. Hwang, *Advanced Computer Architecture: Parallelism, Scalability, Programmability.* McGraw-Hill Book Co., 1993.

[7] X.-H. Sun and D. Rover, "Scalability of parallel algorithm-machine combinations," *IEEE Transactions on Parallel and Distributed Systems*, pp. 599–613, June 1994.

[8] X. Zhang, Y. Yan, and K. He, "Latency matric: An experimental method for measuring and evaluating parallel program and architecture scalability," *J. of Parallel and Distributed Computing*, Sept. 1994.

[9] S. Sahni and V. Thanvantri, "Parallel computing: Performance metrics and models." Research Report, Computer Science Department, University of Florida, May 1995.

[10] X.-H. Sun and J. Zhu, "Shared virtual memory and generalized speedup," in *Proc. of the Eighth International Parallel Processing Symposium*, pp. 637–643, April 1994.

[11] C. Ho and S. Johnsson, "Optimizing tridiagonal solvers for alternating direction methods on boolean cube multiprocessors," *SIAM J. of Sci. and Stat. Computing*, vol. 11, no. 3, pp. 563–592, 1990.

[12] X.-H. Sun, "Application and accuracy of the parallel diagonal dominant algorithm," *Parallel Computing*, vol. 21, 1995.

[13] X.-H. Sun, H. Zhang, and L. Ni, "Efficient tridiagonal solvers on multicomputers," *IEEE Transactions on Computers*, vol. 41, no. 3, pp. 286–296, 1992.

[14] J. Sherman and W. Morrison, "Adjustment of an inverse matrix corresponding to changes in the elements of a given column or a given row of the original matrix," *Ann. Math. Stat.*, vol. 20, no. 621, 1949.

[15] J. Ortega and R. Voigt, "Solution of partial differential equations on vector and parallel computers," *SIAM Review*, pp. 149–240, June 1985.

[16] V. Kumar and et.al, *Introduction to Parallel Computing: Design and Analysis of Algorithms*. The Benjamin/Commings Publishing Company, Inc., 1994.

[17] V. Bala and et. al, "Ccl: A portable and tunable collective communication library for scalable parallel computers," *IEEE Transactions on Parallel and Distributed Systems*, vol. 6, pp. 154–164, Feb. 1995.

[18] X.-H. Sun and J. Zhu, "Performance prediction of scalable computing: A case study," in *Proc. of the 28th Hawaii International Conference on System Sciences*, pp. 456–465, Jan. 1995.

[19] R. Hockney, "Computational similarity," *Concurrency: Practice and Experience*, vol. 7, pp. 147–166, Apr. 1995.