Scalability of Parallel Algorithm–Machine Combinations

Xian-He Sun and Diane T. Rover

Abstract—**Scalability has become an important consideration in parallel algorithm and machine designs. The word *scalable*, or *scalability*, has been widely and often used in the parallel processing community. However, there is no adequate, commonly accepted definition of scalability available. Scalabilities of computer systems and programs are difficult to quantify, evaluate, and compare. In this paper, scalability is formally defined for algorithm–machine combinations. A practical method is proposed to provide a quantitative measurement of the scalability. The relation between the newly proposed scalability and other existing parallel performance metrics is studied. A harmony between speedup and scalability has been observed. Theoretical results show that a large class of algorithm–machine combinations is scalable and the scalability can be predicted through premeasured machine parameters. Two algorithms have been studied on an nCUBE 2 multicomputer and on a MasPar MP-1 computer. These case studies have shown how scalabilities can be measured, computed, and predicted. Performance instrumentation and visualization tools also have been used and developed to understand the scalability related behavior.**

Index Terms—**Scalable high performance computing, performance metrics, performance evaluation of parallel algorithms and machines, scalability, scientific computation, visualization.**

I. Introduction

In 1988, researchers at Sandia National Laboratory achieved speedup greater than 1000 on a 1024-processor multicomputer [1], [2]. The Sandia work [1] is based on the *scaled up* principle: problem size increases with the size of the given computer system. Three applications, in wave mechanics, fluid dynamics, and beam-strain analysis, were implemented on a hypercube architecture multicomputer [1]. The implementation results show that when the problem size is scaled up with the system size the speedup increases linearly with the system size. Each of these applications provides an ideal situation. For massively parallel computing, we would like the performance to increase linearly with the system size. In general, because of design restrictions and physical limitations, the efficiency provided by a given computer for a given algorithm will decrease when the system size is increased. Problem size in general has to be increased to achieve a linear increase of performance. The ratio of system size and problem size increases is closely related to the computation and communication patterns of the application, and to the hardware support of the underlying architecture.

The word *scalable*, or *scalability*, has been widely used in practice to describe how the system size and problem size will influence the performance of parallel machines and algorithms. Scalability is an important issue in parallel processing. It measures the ability of a parallel architecture to support parallel processing at different machine sizes, and measures the inherent parallelism of a parallel algorithm. Scalability can be used to predict the performance of large system and problem sizes based on the performance of small system and problem sizes. It suggests which computer systems could be built with more processors and which algorithms might be more suitable for larger computer systems. Despite the fact that scalability is important and has been widely used in practice, there is no adequate, commonly accepted definition of scalability available. Indeed, scalabilities of computer systems and programs are difficult to quantify, evaluate, and compare. Intensive research has been conducted in this area during recent years [3]. Some metrics have been proposed to measure the scalability of algorithms and computer systems [4], [5], and a formal definition for the scalability of parallel machines also has been proposed [6]. Although these proposed metrics provide ways to measure some special properties of algorithms and architectures, each has certain deficiencies for measuring the scalability of an algorithm-machine combination. In this paper, the *isospeed* metric is proposed. Based on the new metric, the scalability of a parallel algorithm-machine combination is formally defined. The validity of the new definitions is discussed and studied. Three approaches for obtaining the scalability of an algorithm-machine combination are proposed, implemented, and compared. A scalability analysis is applied to actual algorithms. In addition to analytical results, the scalability of these algorithms is experimentally measured on an nCUBE 2 multicomputer and on a MasPar MP-1 data parallel computer. The analytical and experimental results show that the newly defined scalability metric provides a unique quantitative measurement to describe the behavior of a parallel algorithm-machine combination as sizes are varied, which cannot be provided by any other performance metric.

The remainder of this paper is organized as follows. In Section II we present a historical background and the requirements of scalability. The isospeed metric and formal definitions of scalable and scalability are proposed in Section

Manuscript received August 13, 1991; revised June 1, 1993. This work was supported in part by the Applied Mathematical Sciences Program of Ames Laboratory, which is operated for the U.S. Department of Energy under Contract W-7405-ENG-82 while the authors were with Ames Laboratory, Iowa State University, Ames, IA, and in part by the National Aeronautics and Space Administration under NASA Contract NAS1-19480.

X.-E. Sun was with ICASE, NASA Langley Research Center, Hampton, VA 23681–0001, USA. He is now with the Department of Computer Science, Louisana State University, Baton Rouge, LA 70803, USA.

D. T. Rover is with the Department of Electrical Engineering, Michigan State University, East Lansing, MI 48824–1226, USA.

IEEE Log Number 9216782.

III. The relationships between scalability and execution time and between scalability and speedup are studied. Performance prediction is considered. A formula based on scalability is derived for a large class of algorithm-machine combinations. In Section IV, three approaches to obtain scalabilities are proposed. The *Burg* algorithm [7], which is a popular and computationally efficient signal processing procedure, is studied in detail to illustrate the concept of scalability and to compare the three approaches for obtaining scalabilities. The behavior and scalability of a more complicated program, solving a radiosity application [8], is also visualized and studied in Section IV. This visualization approach provides a unique way to reveal characteristics which correspond to the scalability of an algorithm-machine combination. The scalability of the two algorithms are measured on the nCUBE 2 and MasPar MP-1 parallel computers. The observed results are compared, and the influence of hardware design on scalability are discussed. Section V gives the conclusion and comments.

For the purpose of this study, we assume the underlying parallel machine is homogeneous, i.e., all processors are identical. The isospeed scalability proposed in this paper can be applied to any machine architecture. However, the scalability prediction of heterogeneous computing would be more sophisticated than that of homogeneous computing studied here.

II. Background and Preliminary

Scalability has been used in practice as a property that describes the demand for proportionate changes in performance with adjustments in system size. So, a simple, intuitive definition of scalability might be given as follows.

Definition 1: Scalability is a property which exhibits performance linearly proportional to the number of processors employed.

While Definition 1 is very understandable and acceptable, it does not provide enough information. Two questions remain open. First, what performance metric should be chosen to measure the scalability? Second, having selected a metric, how should it be measured? Since *speedup* is one of the most commonly used metrics for parallel processing, and the Sandia work has shown a linear increasing of speedup, speedup seems to be a natural choice. If we choose speedup as the metric, then we need to decide how to measure the speedup. There are three known notions of speedup, *fixed-size speedup*, *fixed-time speedup*, and *memory-bounded speedup* [9], [10]. Fixed-size speedup fixes the problem size and emphasizes how fast a problem can be solved. Amdahl's law [11] is based on the fixed-size speedup model. Therefore, fixed-size speedup is bounded by the reciprocal of the serial fraction of the algorithm. This hard performance limitation implies that fixed-sized speedup is inadequate [6]. Fixed-time speedup argues that parallel computers are designed for otherwise intractably large problems. It fixes the execution time and emphasizes how much more work can be done with parallel processing within the same time. Memory-bounded speedup assumes that a physical limitation of the machine, its memory capacity, is the primary constraint on larger problem sizes. It allows memory capacity to increase linearly with the number of processors available. Both fixed-time and memory-bounded speedup are forms of scaled speedup. Each allows problem size to increase with the system size. The difference between the two speedups is that fixed-time speedup uses execution time to limit the problem size, and memory-bounded speedup uses the memory capacity to limit the problem size. The term "scaled speedup" is used for memory-bounded speedup by many authors [1], [6].

In general, operational complexity increases faster than storage complexity for a given application. Thus, memory-bounded speedup typically yields higher performance than fixed-time speedup. Unfortunately, even with the most liberal speedup, i.e., memory-bounded speedup, no parallel program can possibly exhibit speedup which is linearly proportional to the number of processors available unless the program contains no sequential portion (see [9], [15], and [6]). Speedup is defined to measure the performance gain of parallel processing. It compares the parallel performance over the sequential performance. Operating on a scaled problem size, sequential execution could be impossible or could be very slow, and, therefore, yield a very high speedup. Speedup is a tool for analysis, not the goal of parallel processing. It is not an acceptable metric for representing scalability.

To measure the scalability of parallel algorithms, Kumar *et al.* [4] proposed the *isoefficiency* concept. Isoefficiency fixes the efficiency and measures how much work must be increased to keep the efficiency unchanged. An *isoefficiency function*, $f(N)$, is defined to measure the scalability of parallel algorithms, where N is the number of processors and $f(N)$ is the amount of work needed to maintain the efficiency. The value of $f(N)$ could be arbitrarily large. *Efficiency* is defined as speedup divided by N. So, constant efficiency means that speedup increases linearly with system size. Thus, Kumar *et al.* still use speedup as the performance metric. The significant improvement of their work is its independence of any of the notions of the three speedup models. By using an isoefficiency function, they allow the problem size to increase without bound to attain the requisite efficiency. This approach not only lets algorithms meet the isoefficiency requirement on a given architecture but also prescribes a quantitative measurement. Although the isoefficiency approach is more advanced than the speedup approaches, it is deficient because of its inherent ties to parallel speedup.

Nussbaum and Agarwal [6] proposed a definition of scalability of parallel machines. They define the scalability of a parallel machine, $\Psi(W)$, where W is the problem size, as the best speedup of the given architecture over the best speedup of an ideal parallel machine. This definition of $\Psi(W)$ still relies on speedup as the performance metric; however it measures the speedup differently. It determines the best speedup over the number of processors used, given an unbounded number of processors. While $\Psi(W)$ provides a way to evaluate the design of an architecture, what it represents is the ratio of the best possible performance of the given architecture to the best possible performance of an ideal architecture. In other words, it gives the achieved design efficiency of the architecture. Observe that a given architecture could achieve

its best performance at N equals 10, and the ideal architecture could achieve its best performance at N equals 100. $\Psi(W)$ does not give us any information on how increases in system size will influence algorithm performance. Because of this, it does not match our intuitive understanding of scalability (see Definition 1).

To develop a good definition, we must identify the essential characteristics of scalability. First, based on the intuitive Definition 1, scalability should provide information about how the system size will influence the performance. Can the performance be scaled up with the system size? And, what price must we pay to achieve this scaled performance? Scalability should be a function of the variation of system size.

Second, in general, scalability is a function of the pair of parallel algorithm and machine. Both algorithms and architectures have parallel overhead, and when large system size is employed, the overhead may significantly degrade performance. The sources of degradation are sometimes specified with respect to algorithm and architecture. The commonly considered algorithmic degradation is *uneven allocation*, or *load imbalance* [9]. The commonly considered architecture degradation is *communication cost* which contains the communication latency and other delay incurred by interprocessor communication. These two degradations cannot be clearly separated. On the one hand, the communication delay will influence the achieved degree of parallelism of a given algorithm. On the other hand, unless we have synchronous communication, the communication cost will vary with the inherent degree of parallelism of a given algorithm, even for algorithms with the same communication requirement. With some simplified assumptions, we could distinguish between algorithmic scalability and architectural scalability. For instance, we could use idealized parallel machines and algorithms [5], [6]. However, in general, when we talk of algorithm scalability, we mean the algorithm's scalability with respect to a given architecture. If the architecture represents a large class of architectures, then the algorithm scalability is a general scalability suitable for the class of architectures. Similarly, architecture scalability is a scalability with respect to a given algorithm unless this algorithm represents a class of algorithms. Scalability is an inherent property of algorithms, architectures, and their combinations. Scalability should first be considered for algorithm-machine combinations. Algorithm scalability and architecture scalability could then be defined based on the scalability of the algorithm-machine combination.

Third, scalability should be a *meaningful* and *quantitative* performance metric that can be evaluated and compared. Its meaning should be driven by the motivations and goals of parallel processing. Its value must be calculated by a method that is consistent with the meaning we assign to scalability.

We do not consider physical constraints of the hardware. If the performance cannot scale up with the system size, we say that the machine, or the algorithm-machine combination, is unscalable even if it achieves the best performance using today's technology. Distinguishing such performance degradations will motivate the need for new technologies. For instance, constrained by the three dimensionality of space and today's technologies, systems having more processors require longer wires for interprocessor connection. Therefore, the communication cost will increase at least logarithmically with the system size for a large class of algorithms. This degradation has spurred the development of new technologies, e.g., optical communication, for the next generation parallel machines.

III. Definition and Analysis

There are three main driving forces behind parallel processing: faster execution time; solving otherwise intractably large problems; and providing a better system cost-performance ratio. If we focus on the first two factors, then the performance that we seek from parallel processing involves both execution time and problem size. What we seek from parallel processing is *speed*, where speed is defined as work divided by time. For scientific applications, work is generally measured as floating point operations performed. A parallel algorithm may increase parallelism by sacrificing mathematical efficiency. For this reason, the floating point operations count of parallel processing is usually based on a conventional sequential algorithm. Problem size sometimes has been referred to as a parameter which determines the floating point operations, e.g., the order of a matrix. For this study, we will use the term work and problem size interchangeably without distinction. While we agree with others that this measure of work is less than adequate in some contexts, we measure work in terms of floating point operations in this study because of its prevalence. Our objective here is to define *scalability* not *work*. The scalability metric we define is not biased toward any particular work measure and can adopt a better work measure when one is developed. With speed as a goal, we seek the power to solve problems of some magnitude in a reasonably short amount of time. Speed is a quantity that ideally would increase linearly with system size. So, it is consistent with the scalability property proposed in Definition 1. Based on this reasoning, we propose the isospeed approach.

Definition 2: The `average unit speed` is the achieved speed of the given computing system divided by N, the number of processors.

We refer to average unit speed as *average speed* when the context is clear. Saying that the speed of a computing system is linearly proportional to the system size is the same as saying that the average speed is a constant number independent of system size. Using average speed, following Definition 1, we give the following definition for any algorithm-machine combination in which the parallel machine is homogeneous.

Definition 3: 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.

By Definition 3, scalability is expressed in terms of system size. In general, increasing the problem size will increase the computation/overhead ratio, and therefore, increase the speed. This is especially true for parallel processing where the computation/communication ratio increases with problem size for most algorithms. For a large class of algorithm-

machine combinations, the average speed can be maintained by increasing problem size (see case studies in Section IV). The necessary problem size increase varies with algorithms, machines, and their combination. This variation provides a quantitative measurement for scalability. Let W be the amount of work of an algorithm when N processors are employed in a machine, and let W' be the amount of work of the algorithm when $N' > N$ processors are employed to maintain the average speed, then we define the *scalability from system size N to system size N'* of the algorithm-machine combination as follows.

$$\psi(N, N') = \frac{N'W}{NW'}. \tag{1}$$

The work W' is determined by the isospeed (or, to be accurate, the iso-average-speed) constraint. In the ideal situation, we have trivial parallelism, or the local computation model [12]. There is no communication necessary and work is replicated on each processor. In this case, $W' = \frac{N'W}{N}$ and $\psi(N, N') = 1$. In general, $W' > \frac{N'W}{N}$ and $\psi(N, N') < 1$. The initial work W is a problem size which can achieve the initial speed. It is determined by the average speed constraint. To have a unique scalability for an algorithm-machine combination, we need to define the initial average speed uniquely for each algorithm-machine pair. Borrowing the asymptotic performance idea from Hockney [13], we define the asymptotic average unit speed of an algorithm-machine combination, r_∞, as the maximum speed reached by the single processor execution of the algorithm on the machine when problem size goes to infinity. The initial average speed is a fraction of the asymptotic speed. Following the $n_{1/2}$ defined in [13], we choose the initial average speed, $r_{1/2}$, as half of the asymptotic speed.

When $N = 1$, we denote $\psi(N, N')$ by $\psi(N')$. Recall that speed is equal to work divided by time, and average speed is equal to speed divided by the number of processors. Let T_N, $T_{N'}$ be the execution time when N and N' processors are employed, respectively; then when the average speeds are the same, we have

$$\frac{W}{NT_N} = \frac{W'}{N'T_{N'}}$$
$$\frac{N'W}{NW'} = \frac{T_N}{T_{N'}},$$

and

$$\psi(N, N') = \frac{T_N}{T_{N'}}. \tag{2}$$

When N equals one,

$$\psi(N') = \frac{T_1}{T_{N'}} = \frac{\textit{sequential execution time with problem size } W}{\textit{parallel execution time with problem size } W'}. \tag{3}$$

Equation (3) has a representation similar to the traditional definition of speedup. The difference is that in traditional speedup the problem size is fixed and in scalability the speed is fixed. Speedup measures the performance gain of parallel processing versus sequential processing (see Fig. 1). Scalability measures the performance degradation of larger parallel systems versus smaller parallel systems (see Fig. 2). From (3), we can see the harmony and discord between speedup and scalability.

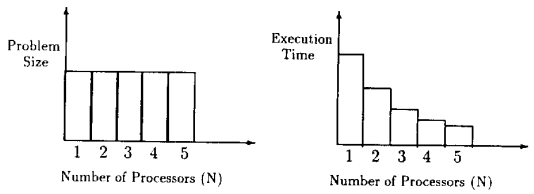


Fig. 1. Traditional speedup: Fix problem size, measure execution time.

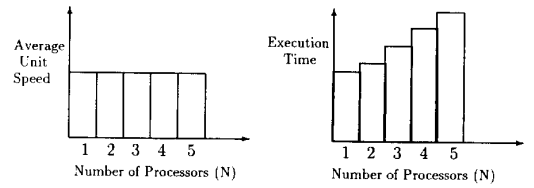


Fig. 2. Scalability: Fix average speed, measure execution time.

- Both speedup and scalability depend on the initial status (initial problem size and number of processors) and the "distance", i.e., the incremental change in the number of processors. In general, as "distance" increases, the speedup will increase and the scalability will decrease.
- Both speedup and scalability are dimensionless quantities, unlike performance metrics such as execution time or speed. However, each relates to these dimensioned performance metrics in some way. Speedup provides the variation ratio of the chosen dimensioned performance metric when the system size is increased. Scalability indicates the ability of the system to maintain the chosen dimensioned performance metric when the system size is increased.

Traditionally, speedup has been defined as sequential execution time over parallel execution time. A generalized speedup was recently introduced by Gustafson and Sun [14]. The relationship between generalized speedup and traditional speedup has been studied in [14]. The generalized speedup is defined as

$$\textit{Generalized speedup} = \frac{\textit{parallel speed}}{\textit{sequential speed}}. \tag{4}$$

If generalized speedup is used and assuming the sequential speed is independent of problem size (which is not true in general), then two implementations have the same efficiency if and only if they have the same average speed. In this case, the isospeed approach is the same as the isoefficiency approach.

In general, sequential execution speed varies with problem size. Overhead exists, and achieved speed rarely matches peak speed. Excluding any virtual memory effects, it is likely that increasing the problem size will increase the sequential execution speed. We use $\mu(W)$ to represent the sequential execution speed when W amount of work is done. When N processors are used concurrently, the parallel execution

time can be divided into two parts, $T_N = t_{cN} + t_{oN}$, where t_{cN} (c stands for computing) is the computation time, and t_{oN} (o stands for overhead) is the cumulative time spent on communication, synchronization and other overhead caused by parallel processing. The following theorem shows that a large class of algorithm-machine combinations is scalable and the scalability can be calculated from precomputed parameters.

Theorem 1: If an algorithm has a balanced load on each processor, its communication cost is independent of problem size, and the single processor speed of the underlying machine increases with work load, then the algorithm-machine combination is scalable and the scalability

$$\psi(N, N') = \frac{t_{cN}}{t_{oN'}}\left[1 - \frac{\mu(W/N)}{\mu(W'/N')}\right] + \frac{t_{oN}}{t_{oN'}}. \quad (5)$$

Proof: If N processors are used to implement W amount of work, and W' is the amount of work required to maintain the average unit speed when $N' > N$ processors are employed, then we have

$$\frac{W}{NT_N} = \frac{W'}{N'T_{N'}}$$
$$\frac{W}{N(t_{cN} + t_{oN})} = \frac{W'}{N'(t_{cN'} + t_{oN'})}.$$

Since the work is distributed evenly among all processors,

$$\frac{W}{N(\frac{W}{N\mu(W/N)} + t_{oN})} = \frac{W'}{N'(\frac{W'}{N'\mu(W'/N')} + t_{oN'})}$$
$$\cdot\left(\frac{1}{\mu(W/N)} - \frac{1}{\mu(W'/N')}\right) + \frac{Nt_{oN}}{W} = \frac{N't_{oN'}}{W'}. \quad (6)$$

Since $(\frac{1}{\mu(W/N)} - \frac{1}{\mu(W'/N')}) + \frac{Nt_{oN}}{W} > 0$ and $t_{oN'}$ is independent of problem size, by (6), the algorithm-machine combination is scalable and the amount of work required to maintain the average unit speed is

$$W' = \frac{N't_{oN'}}{(\frac{1}{\mu(W/N)} - \frac{1}{\mu(W'/N')}) + \frac{Nt_{oN}}{W}}. \quad (7)$$

In this case

$$\psi(N, N') = \frac{N'W}{NW'} = \frac{W[(\frac{1}{\mu(W/N)} - \frac{1}{\mu(W'/N')}) + \frac{Nt_{oN}}{W}]}{Nt_{oN'}}$$
$$= \frac{(\frac{W}{\mu(W/N)} - \frac{W}{\mu(W'/N')}) + Nt_{oN}}{Nt_{oN'}}$$
$$= \frac{t_{oN}}{t_{oN'}} + \frac{1}{t_{oN'}}\left[t_{cN} - t_{cN}\frac{\mu(W/N)}{\mu(W'/N')}\right].$$

Therefore,

$$\psi(N, N') = \frac{t_{cN}}{t_{oN'}}\left[1 - \frac{\mu(W/N)}{\mu(W'/N')}\right] + \frac{t_{oN}}{t_{oN'}}. \quad (8)$$

□

Based on the proof of Theorem 1, we can see that the conditions "communication cost is independent of problem size" and "single processor speed of the underlying machine increases with work load" are only used to guarantee Equation (7) having a feasible solution. Replacing the two conditions by (7), we have the following.

Theorem 2: If an algorithm has a balanced load on each processor, and the equation

$$W' = \frac{N't_{oN'}(W')}{(\frac{1}{\mu(W/N)} - \frac{1}{\mu(W'/N')}) + \frac{Nt_{oN}(W)}{W}}$$

has a positive solution for any positive number W, then the algorithm-machine combination is scalable and the scalability

$$\psi(N, N') = \frac{t_{cN}(W)}{t_{oN'}(W')}\left[1 - \frac{\mu(W/N)}{\mu(W'/N')}\right] + \frac{t_{oN}(W)}{t_{oN'}(W')}. \quad (9)$$

While its condition is more difficult to verify, Theorem 2 has a weaker condition than Theorem 1. The following corollary was used in our case studies.

Corollary 1: Under the assumptions of Theorem 1 (or of Theorem 2), if the communication cost goes to infinity as the system size goes to infinity, then for any fixed N, $\lim_{N'\to\infty} \psi(N, N') = 0$.

Proof: By Theorem 1,

$$\psi(N, N') = \frac{t_{cN}(W)}{t_{oN'}(W')}\left[1 - \frac{\mu(W/N)}{\mu(W'/N')}\right] + \frac{t_{oN}(W)}{t_{oN'}(W')}.$$

By the definition of asymptotic speed r_∞, $\mu(W'/N') < r_\infty$. Therefore,

$$1 - \frac{\mu(W/N)}{\mu(W'/N')} < 1 - \frac{\mu(W/N)}{r_\infty},$$
$$0 \le \psi(N, N') \le \frac{t_{cN}(W)}{t_{oN'}(W')}\left[1 - \frac{\mu(W/N)}{r_\infty}\right] + \frac{t_{oN}(W)}{t_{oN'}(W')}.$$

Since $t_{oN'} \to \infty$, when $N' \to \infty$, we have

$$\lim_{N'\to\infty}\left(\frac{t_{cN}(W)}{t_{oN'}(W')}\left[1 - \frac{\mu(W/N)}{r_\infty}\right] + \frac{t_{oN}(W)}{t_{oN'}(W')}\right) = 0.$$

Thus,

$$\lim_{N'\to\infty} \psi(N, N') = 0$$

for any fixed N. □

In Theorem 1 and Theorem 2, we do not consider the physical limitations of the underlying machine. We assume that the target architecture has adequate memory capacity to sustain problem size increases. This assumption may not be generally applicable. If the memory capacity of a machine becomes a factor, the condition $W' \le W_0$ (or $\frac{W'}{N'} \le W_0$ for distributed memory multiprocessors), for some constant number W_0, should be included in the theorems. In general the scalability of an algorithm-machine combination is constrained by memory and communication requirements. That is, certain combinations of problem and system sizes can result in memory bound or communication bound program execution. Fig. 3 diagrams the general scalability space of algorithm-machine combinations.

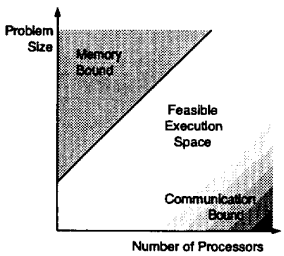


Fig. 3. Problem-ensemble space.

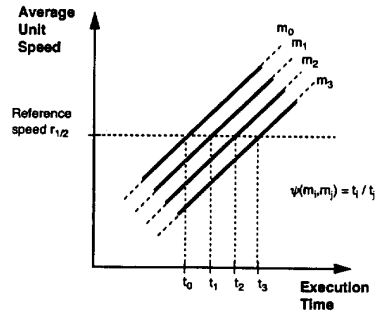


Fig. 4. Illustration of approach 2.

IV. Case Studies: Measurement and Visualization

In this section, we present four case studies that illustrate the preceding definitions relating to scalability and show how the scalability could be measured, computed, and predicted. Each case study involves a parallel program having a structure that is to some extent scalable on the parallel machine. That is, each is executable over a range of problem sizes on small scale to large scale system sizes. In each case, we briefly describe the algorithm-machine combination, do a scalability analysis, and highlight program characteristics via performance visualization.

The two programs under study are: the Burg algorithm and the modified SLALOM benchmark program. The Burg program is a linear signal processing procedure for fitting an autoregressive model to a time series data set [7]. Time (and work, measured as the number of floating point operations performed) varies linearly with the number of data points in the series; storage also varies linearly. SLALOM (Scalable Language-independent Ames Laboratory One-minute Measurement) [15] is a program designed as a scalable computer benchmark. It solves a radiosity problem by calculating the equilibrium radiation given off by a coupled set of diffuse surfaces that emit and absorb radiation. The surfaces are decomposed into patches, which are the basis for a radiosity matrix. The core computation of SLALOM is solving the radiosity matrix by Gaussian elimination, resulting in at most order n^3 time (or work) and order n^2 storage, where n is the number of patches.

The two machines under study are: the nCUBE 2 computer and the MasPar MP-1 computer. The nCUBE 2 is a distributed memory message-passing MIMD computer with a hypercube interconnection topology. Each node is a 32-bit custom CISC processor having a minimum of one megabyte of memory and operating at 20 megahertz clock rate. To contrast, while double-precision multiply and add require 6 clocks and 7 clocks, respectively, message overhead consumes approximately 1600 clocks. At the beginning of this study, we used a 64-node system in our laboratory (now, 128 nodes). The largest installed system has 1024 nodes. The MasPar MP-1 is a distributed memory massively parallel SIMD computer with a high-speed two-dimensional toroidal mesh topology. A control unit issues instructions at 12.5 megahertz clock rate. Each processing element in the array is a 4-bit custom load/store processor having a minimum of 16 kilobytes of memory. While double-precision multiply and add require 530 and 180 clocks, respectively, the mesh network overhead is approximately 12–30 clocks. We used an 8192-processor system in our laboratory. The system now has been upgraded to 16384 processors.

A. Calculation of Scalability

Three different approaches to obtain the scalabilities of an algorithm-machine combination have been noticed and compared in our study.

1) The scalability can be *measured* in software by a control program that invokes the application program and searches for the run having the desired fixed average unit speed.
2) The scalability can be *computed* by first finding the relation between average unit speed and execution time (or work) and then using Equation 2 (or Equation 1).
3) The scalability can be *predicted* by deriving a general scalability formula.

The first approach is the most direct and accurate. It is easy to understand and implement. The only point requiring mention is that we do not include redundant work (extra work added for parallel processing) in the work calculation. The second approach provides more information than the first approach. It shows the influence of initial speed. If we assume the derived relation remains true for larger system size and problem size or if we can predict the relation for large system and problem size based on the performance of small system and problem size, it also can be used to predict scalability. A graphical method of calculating scalability based on approach 2 is shown in Fig. 4. The method uses a set of measured data to find the relation between average unit speed and execution time. One curve (relation) is plotted for each machine size, with execution time on the horizontal axis and average unit speed on the vertical axis. Execution time varies as the problem size is varied. The third approach is the simplest one if a formula can be derived. However, in general, a formula may not be available or only can be found under certain simplified conditions.

In the sample plot of Fig. 4, four curves are drawn, one per machine size (typically m_0 is the smallest size), in order to illustrate approach 2. Each curve is generated by measuring average unit speed and execution time for runs of increasing problem size. The maximum average unit speed is calculated by taking the asymptotic limit of average unit speed with

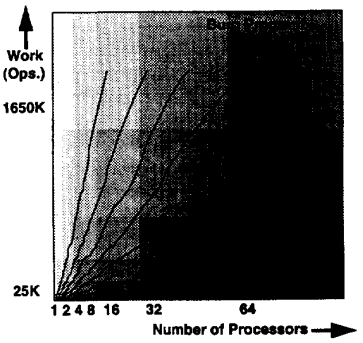


Fig. 5. Isospeed curves for the Burg algorithm.

respect to time as time (or work) approaches infinity. The constant average unit speed used as the reference point could be any fraction of this maximum: we choose half of the maximum of single processor execution in our study. This reference speed is marked in the figure with the horizontal dashed line. The intersection of this speed line with each curve determines the program execution time delivering that speed. The ratios of these times, as defined in Section 3, yields a set of scalability numbers. The relative horizontal time positions indicate time (or work) variation and provide visual cues for the scalabilities.

The Burg-nCUBE combination is used to compare the three approaches.

B. Burg-nCUBE Combination

The Burg program employs a block decomposition form of data parallelism on the nCUBE 2 computer. Each processor does a number of calculations dependent primarily on its block size, and so for evenly distributed data there is a balanced load. The main loop of the Burg program can be approximately divided into three phases [7]: 1) a linear shift operation to align arrays, followed by local array calculations; 2) a global sum operation; and 3) local array update calculations. The number of communication steps required for the linear shift operation is constant, and the number for the global sum is logarithmic with respect to machine size. Communication in the Burg algorithm is independent of problem size. The Burg algorithm fits the conditions of Theorem 1 well.

The measured isospeed (i.e., constant average unit speed) curves are shown in the contour plot of Fig. 5. Each curve represents system size-problem size combinations that yield approximately equal average speed per processor. The desired average unit speed used as the reference point for calculating scalabilities corresponds to a curve within this contour plot. Considering Definition 3, we can see that the Burg algorithm is scalable: a constant average speed can be identified as system size increases. The background shading indicates the value of the average speed, where lighter shading corresponds to relatively higher speeds.

Selecting the isospeed corresponding to half of the asymptotic speed and applying approach 2, we derive a set of scalabilities, listed in Table I. The asymptotic speed calculated for this algorithm-machine combination is 1.7 MFLOPS (millions of floating point operations per second), giving a reference speed of 0.85 MFLOPS. As in Fig. 4, a set of average unit speed (as a function of time) curves is plotted based on measured data, and from this, a set of times is obtained. The execution times (in milliseconds) used to compute the scalabilities are shown in Table II. The variation in scalability can be shown pictorially by representing the tabular data (of Table I) as a three-dimensional surface plot, as in Fig. 6. The table rows and columns map to two dimensions, and scalability, to the third dimension. Higher scalability values also are represented by darker shading for emphasis. This plot portrays the range and trends of algorithm-machine scalabilities at a glance.

TABLE I
COMPUTED SCALABILITY OF BURG-nCUBE COMBINATION (APPROACH 2)

$\psi(N, N')$	1	2	4	8	16	32	64	128
1	1.00	0.441	0.296	0.215	0.188	0.157	0.136	0.121
2		1.00	0.670	0.488	0.426	0.357	0.308	0.274
4			1.00	0.728	0.635	0.532	0.460	0.408
8				1.00	0.827	0.693	0.599	0.531
16					1.00	0.837	0.724	0.642
32						1.00	0.865	0.767
64							1.00	0.887
128								1.00

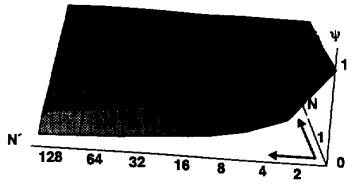


Fig. 6. Variation in scalability: Three-dimensional surface plot of computed scalability for the Burg-nCUBE combination (see Table I).

Another useful pictorial representation of the tabular data is shown in Fig. 7. A set of curves is drawn, one curve per machine size N (or per row of Table I). Scalability ψ is on the vertical axis, and machine size N' (columns of Table I) is on the horizontal axis. The variation in scalability is shown for 1) each initial machine size with respect to larger machine sizes (i.e., N with respect to N') and 2) increasing initial machine sizes N.

The average unit speed requirement makes the execution time unique for each system size. Notice that by the definition of scalability (1) $\psi(N, N'') = \psi(N, N') * \psi(N', N'')$ for any $0 < N < N' < N''$. The scalability matrix of Table I can be abbreviated as a vector array as shown in Table III. All the scalabilities in Table I can be computed directly from the scalabilities in Table III.

By Theorem 1, the scalability of the Burg algorithm also can be predicted via (5). In hypercube multicomputers, message transfer time between two adjacent processors can be expressed as $\alpha + \beta S$, where α is the communication latency, β is the transmission time per byte, and S is the number of bytes in the message. We have measured $\alpha = 130$ (microsecond) and $\beta = 0.5$ (microsecond) for our nCUBE 2 multicomputer. Substituting the communication cost of the Burg algorithm

TABLE II
TIME VARIATION OF BURG-nCUBE COMBINATION

system size	1	2	4	8	16	32	64	128
Time	0.004029	0.00913	0.01362	0.01774	0.02144	0.02561	0.0296	0.03338

TABLE III
COMPUTED SCALABILITY OF BURG-nCUBE COMBINATION (APPROACH 2)

$\psi(1,2)$	$\psi(2,4)$	$\psi(4,8)$	$\psi(8,16)$	$\psi(16,32)$	$\psi(32,64)$	$\psi(64,128)$
0.441	0.670	0.728	0.827	0.837	0.865	0.887

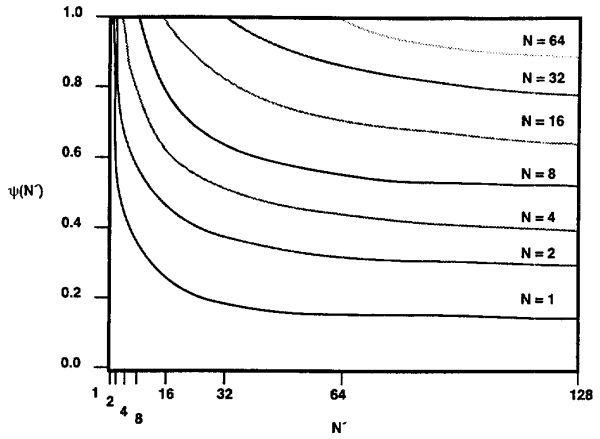


Fig. 7. Variation in scalability: Interpolated plot of $\psi(N')$ for each N (see Table I).

into (5), we have

$$\psi(N,N') = \frac{t_{cN}}{[\alpha+4\beta]+\log(N')[\alpha+8\beta]}\left[1-\frac{\mu(W/N)}{\mu(W'/N')}\right] + \frac{[\alpha+4\beta]+\log(N)[\alpha+8\beta]}{[\alpha+4\beta]+\log(N')[\alpha+8\beta]}. \quad (10)$$

The set of predicted scalabilities obtained by applying approach 3 and the prediction formula from (10) is shown in Table IV. Finally, the set of directly measured scalabilities obtained by applying approach 1 is given in Table V.

Compared to the measured scalability of Table V, both the predicted scalability of Table IV and the computed scalability of Table I provide good approximations for the Burg-nCUBE combination. The predicted scalability tends to be slightly higher than the measured scalability due to the fact that in the actual algorithm, in addition to the communication cost $\alpha + \beta S$, extra operations are involved to prepare variables for communication. For the computed scalability, its function fitting method has included this extra cost in the fitting function.

Taking advantage of the fitting function method used for the computed scalability, we can predict the performance of a larger system based on the performance of smaller systems. For example, the predicted execution time for a 128-node system, based on the computed execution times calculated for systems up to 64 nodes, is 0.03409 sec. (only $\approx 2\%$ greater than the number listed in Table II). Using this time with the others yields the following scalabilities for $N < 128$ and $N' = 128$: 0.118, 0.269, 0.400, 0.520, 0.629, 0.751, and 0.868. Comparing these numbers with the rightmost column of Table IV, which is the scalability predicted by approach 3, we observe that they more closely approximate the actual numbers in the rightmost column of Table V, which is the measured scalability. Here, prediction via computed values from approach 2 is better than prediction via approach 3. Whereas approach 2 uses all the information available, approach 3 only uses the single node execution information.

TABLE IV
PREDICTED SCALABILITY OF BURG-nCUBE COMBINATION (APPROACH 3)

$\psi(N,N')$	1	2	4	8	16	32	64	128
1	1.00	0.410	0.336	0.281	0.238	0.207	0.182	0.162
2		1.00	0.743	0.581	0.478	0.405	0.352	0.311
4			1.00	0.783	0.643	0.545	0.474	0.419
8				1.00	0.826	0.691	0.589	0.536
16					1.00	0.837	0.713	0.652
32						1.00	0.852	0.769
64							1.00	0.885
128								1.00

TABLE V
MEASURED SCALABILITY OF BURG-nCUBE COMBINATION (APPROACH 1)

$\psi(N,N')$	1	2	4	8	16	32	64	128
1	1.00	0.451	0.319	0.240	0.193	0.161	0.138	0.121
2		1.00	0.707	0.533	0.427	0.357	0.306	0.269
4			1.00	0.753	0.604	0.504	0.433	0.380
8				1.00	0.802	0.670	0.575	0.504
16					1.00	0.835	0.717	0.629
32						1.00	0.859	0.753
64							1.00	0.876
128								1.00

The communication cost of the Burg-nCUBE combination increases slowly with the system size. If $N' - N = constant$, the second term of Equation (10), the ratio of communication, will increase with N and approaches 1 as N approaches infinity. As N grows, the Burg-nCUBE combination should exhibit better scalability. This is confirmed by the measured results (see Table V).

Another trend observed via the measured results can be explained analytically. Note in Table V the lower scalabilities for small N and large N'. This is consistent with Corollary 1 in Section III, which says that as N' grows for fixed N, scalability should approach zero. How fast it approaches zero depends on the computation time versus communication time ratio of the system and on the initial system size. Since this ratio is small for the nCUBE 2 (e.g., 7 clock cycles/1600 clock

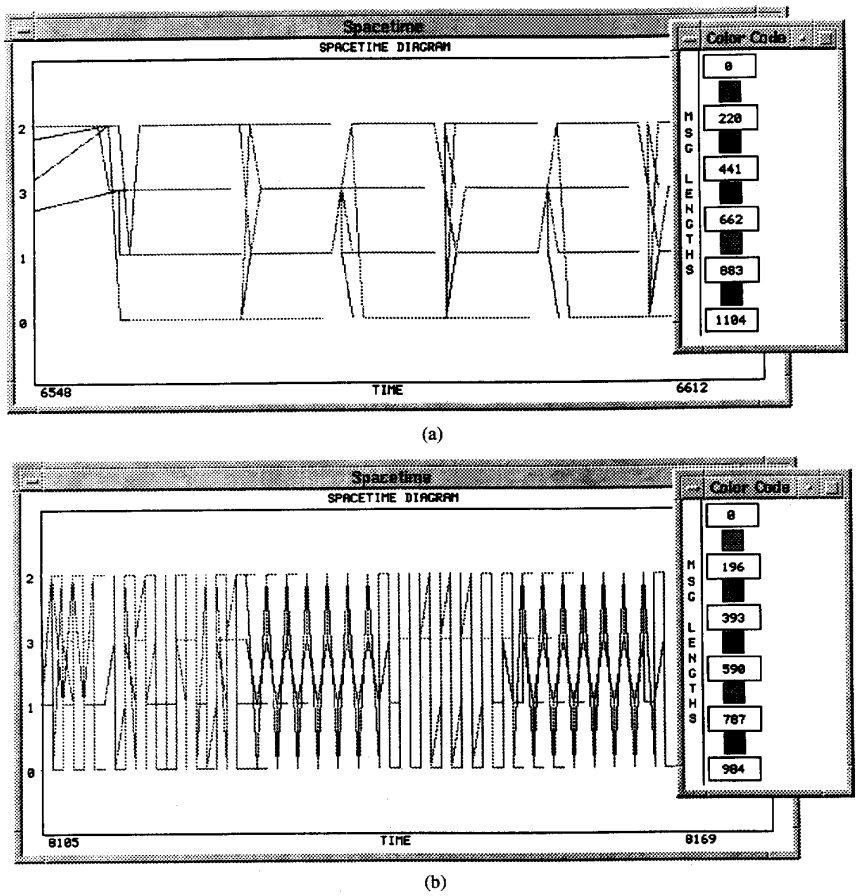


Fig. 8. Spacetime diagrams of SLALOM program execution during matrix solution (4 processors).

cycles), the reduction in scalability for small initial system sizes is steep. That is, despite increasing slowly with system size, communication cost can be relatively high compared to time spent in computation. The effort (in terms of increasing problem size) needed to offset the overhead and to reach the isospeed is then considerable.

C. SLALOM-nCUBE Combination

The SLALOM program does not satisfy the condition of Theorem 1. Thus, approach 3 for calculating scalability is not applicable. Results from applying approaches 1 and 2 are presented in this section. Tables VI and VII give sets of computed and measured scalabilities for the SLALOM-nCUBE combination, respectively. This version of SLALOM for the nCUBE computer only executes on "square" hypercubes (hence the N and N' used in the tables). The asymptotic speed calculated for the SLALOM-nCUBE combination is 1.5 MFLOPS, giving a reference speed of 0.75 MFLOPS.

The SLALOM algorithm has a more complex structure than the Burg algorithm. The computed scalability of the SLALOM-nCUBE combination does not match the measured scalability as closely as in the Burg-nCUBE combination. Performance instrumentation and visualization tools are especially useful for these more complex cases. The tools—by capturing and depicting the dynamic behavior of parallel program execution—can contribute to and expand our perception of scalability. We applied the PICL (Portable Instrumented Communication Library) [16] and ParaGraph [17] toolset to the SLALOM-nCUBE combination for this purpose. PICL produces an execution trace during an actual run of an in-

TABLE VI
COMPUTED SCALABILITY OF SLALOM-nCUBE COMBINATION (APPROACH 2)

$\psi(N, N')$	1	4	16	64
1	1.00	0.65	0.54	0.38
4		1.00	0.83	0.59
16			1.00	0.71
64				1.00

TABLE VII
MEASURED SCALABILITY OF SLALOM-nCUBE COMBINATION (APPROACH 1)

$\psi(N, N')$	1	4	16	64
1	1.00	0.625	0.504	0.344
4		1.00	0.806	0.549
16			1.00	0.681
64				1.00

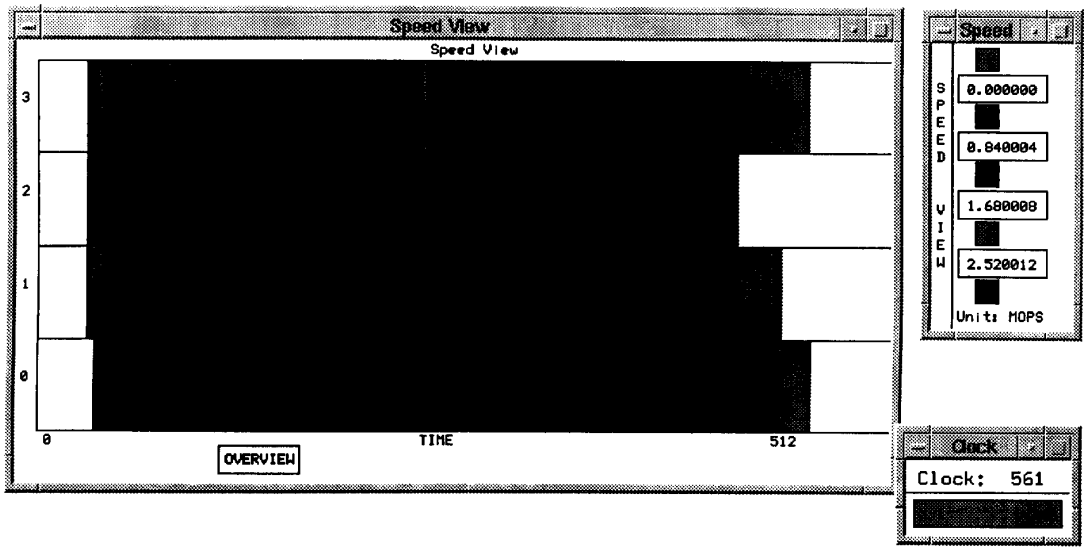


Fig. 9. Average speed chart and clock for SLALOM program execution (4 processors).

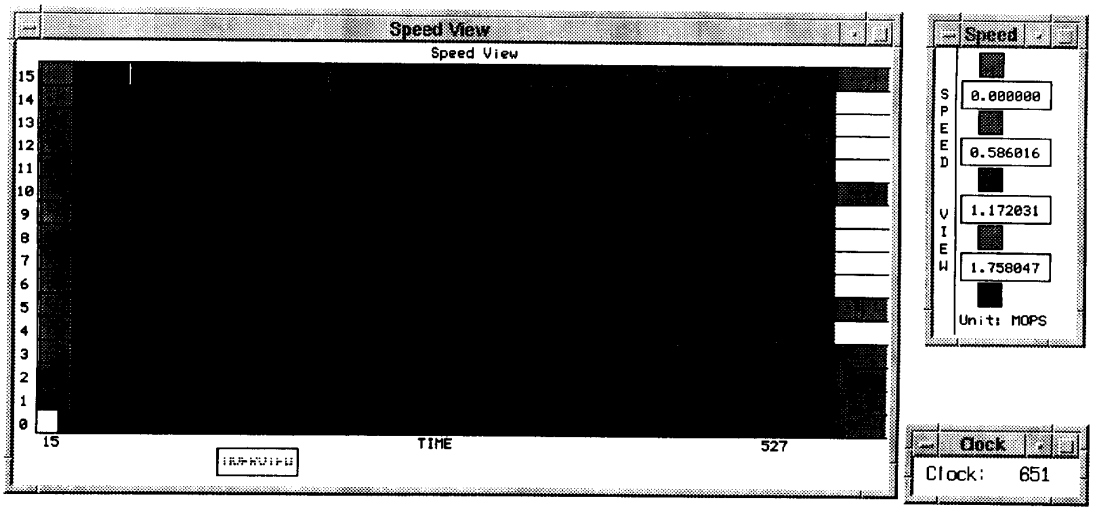


Fig. 10. Average speed chart and clock for SLALOM program execution (16 processors).

strumented program, and ParaGraph replays this trace using multiple views. Each was developed by researchers at Oak Ridge National Laboratory and is available via *netlib*. We selected appropriate views in ParaGraph to depict scalability-related information. Because scalability involves a comparison based on constant average unit speed, we compare views under those conditions. Two of these views are the Spacetime Diagram and the Average Speed Chart.

The Spacetime Diagram, supplied with ParaGraph, is patterned after the diagrams that are used in physics to depict interactions between particles. Processor activity is indicated by horizontal lines, one for each processor, while messages between processors are depicted by slanted lines between the sending and receiving processor activity lines, indicating the times at which each message was sent and received. A blank horizontal line indicates that a processor is idle waiting to receive a message. The color of a slanted line represents the relative size of the message in bytes. The Spacetime Diagram presents the structure of communication for the algorithm-machine combination. Comparing multiple Spacetime Diagram views, we observe how problem size and/or system size variations affect the structure. We can detect communication dependencies on problem size or system size, uniformities, periodicities, etc., all of which can influence scalability.

The structure of the SLALOM program is complicated. It contains different phases which have totally different computation and communication structures. Fig. 8 shows Spacetime Diagrams for one part of SLALOM, the core computation of matrix solution. Fig. 8(a) depicts the beginning of the

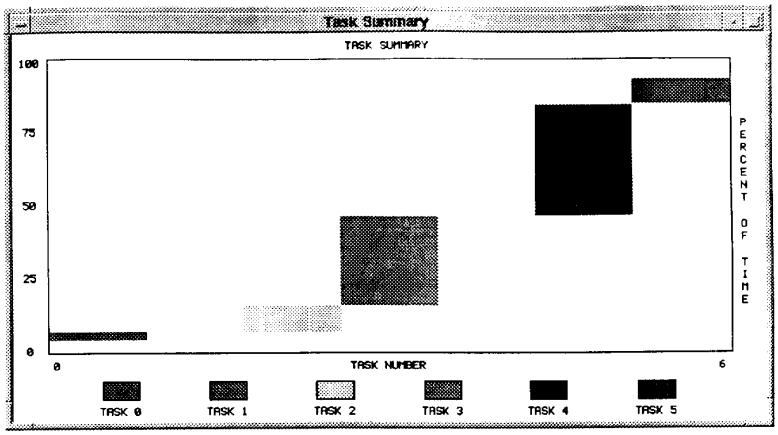


Fig. 11. Task summary view of SLALOM program execution (4 processors).

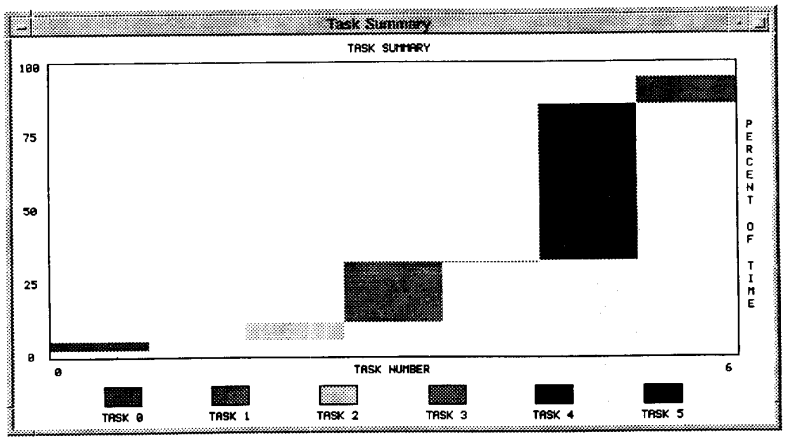


Fig. 12. Task summary view of SLALOM program execution (16 processors).

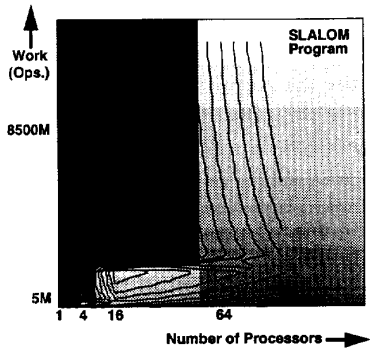


Fig. 13. Isospeed curves for SLALOM program.

factoring algorithm, and Fig. 8(b), the backsolving algorithm. The communication requirement is high, and it depends on both system size and problem size.

The Average Speed Chart, a view we designed and integrated into ParaGraph, depicts the speed of individual processors by a horizontal bar chart as a function of time. The color of each bar indicates the average speed (in MFLOPS) of the corresponding processor during each user-defined phase. The Average Speed Chart demonstrates that the isospeed conditions of scalability are being met and graphically displays the execution times for comparative analysis. Execution times for phases of the algorithm as well as the whole algorithm are shown.

Figs. 9 and 10 show Average Speed Charts for two complete runs of the SLALOM program. In each case, an average unit speed of 0.75 MFLOPS is achieved. Fig. 9 corresponds to a run using 4 nodes, and Fig. 10, 16 nodes. Problem sizes are 276 patches and 490 patches, respectively. SLALOM's core computation part comprises the widest shaded bar in each chart. Clock views, displaying simulated execution time, are also shown in the figures. An estimate of the scalability, $\psi(4, 16)$, can be obtained by reading the Clocks and taking the ratio of total execution times, $T_4 \,/\, T_{16} = 561/651 = 0.862$. (Note that the times are in Paragraph's simulated time units, which are related to real time units based on factors internal to Paragraph and entered by the user.)

From the idle processors in Fig. 8 and the non-uniform processor profiles in Fig. 10, we can see that the SLALOM

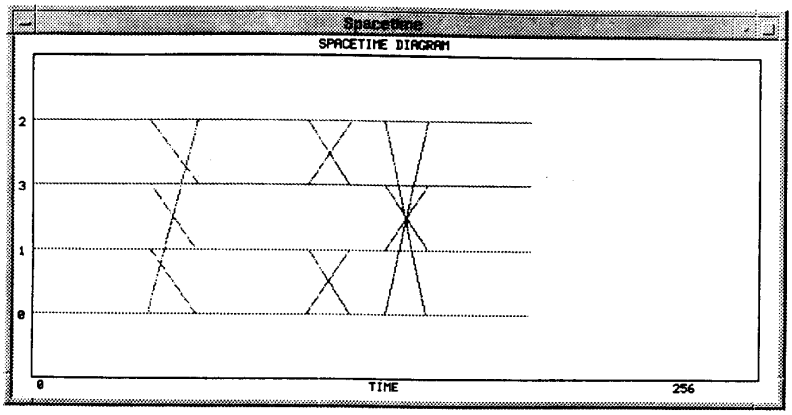


Fig. 14. Spacetime diagram of Burg program execution, first iteration (4 processors).

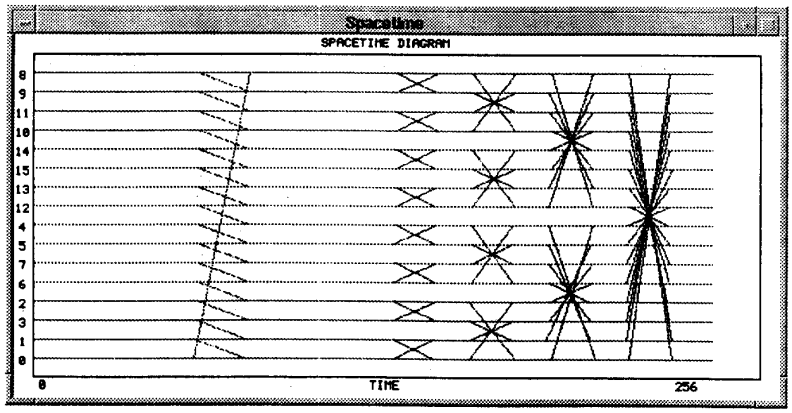


Fig. 15. Spacetime diagram of Burg program execution, first iteration (16 processors).

program has noticeable load imbalance degradation. Comparing Fig. 9 with Fig. 10, we can also see that the ratio of the execution time of each phase to the total execution time varies with system size. This variation makes the scalability of a SLALOM-nCUBE combination (or a SLALOM-MasPar combination) unpredictable with present analytical methods. The variation is specifically shown in the Task Summary views of Figs. 11 and 12 (4 and 16 node cases, respectively). These views show the percent of total execution time spent in each task, or phase, of the program. The tasks are identified with numbers (circularly) on the horizontal axis, and the slice of time taken by each task is displayed on the vertical axis. Seven tasks are delineated within SLALOM.

Fig. 13 shows the isospeed curves of SLALOM-nCUBE combinations. Comparing Fig. 13 with Fig. 5, we can see that for the domain under study these isospeed curves have a more complicated structure, including sharp turning points and peaks.

To contrast the SLALOM and Burg algorithms further, consider selected Paragraph views of Burg-nCUBE combinations. Figs. 14 and 15 show Spacetime Diagrams for two executions of the Burg algorithm. Fig. 14 corresponds to a run using 4 nodes, and Fig. 15, 16 nodes. Problem sizes are 440 data points and 2880 data points, respectively. In each case, an average unit speed of 0.85 MFLOPS is achieved. From these figures we can see that the Burg algorithm has only two communication patterns, and each is neatly structured. The first is independent of system size, and the second has a number of communication steps which increases logarithmically with system size (two send-receive pairwise exchanges in Fig. 14 and four pairwise exchanges in Fig. 15). Observe that no processors ever wait to receive a message; the load is evenly distributed.

Figs. 16 and 17 show Average Speed Charts and Clocks for the two executions of the Burg algorithm corresponding to Figs. 14 and 15, respectively. An estimate of the scalability, $\psi(4, 16)$, via reading the Clocks, is $T_4 \,/\, T_{16} = 195/301 = 0.648$. The three phases of the algorithm (see Section 4.2) are delineated within these Average Speed Charts. Taking note of the Speed Legends in the figures and comparing the two combinations, we see that the "phase speeds" (i.e., the speeds of processors during the same phase) are approximately equal. Differences in speed are due mainly to instrumentation artifact. We can also see that the ratio of the execution time of each phase to the total execution time is approximately unchanged. This means that the scalabilities of the parts (i.e., the phases) are equal to the scalabilities of the whole. This can be shown analytically. As the Burg-nCUBE combination scales, so does each particular phase of the algorithm.

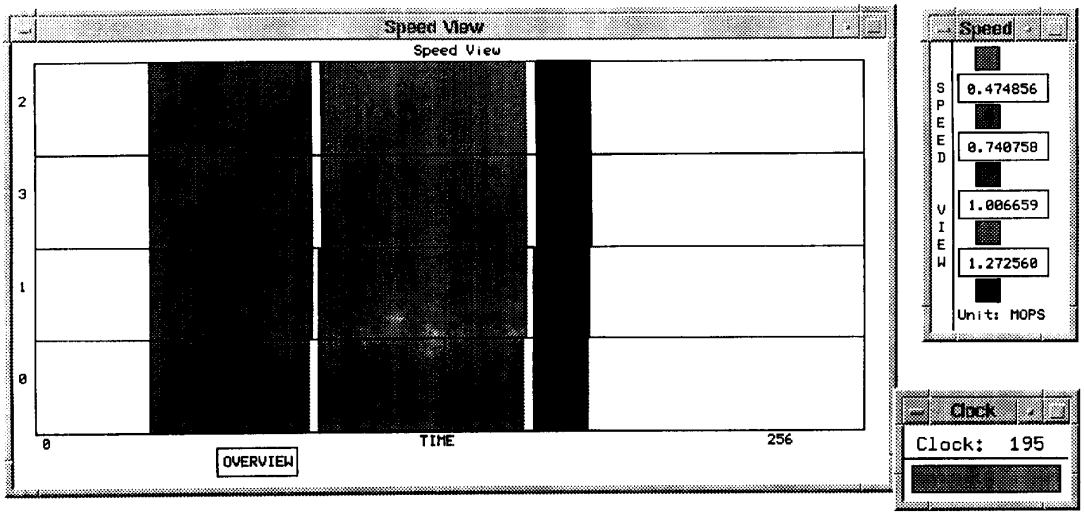


Fig. 16. Average speed chart and clock for Burg program execution, first iteration (4 processors).

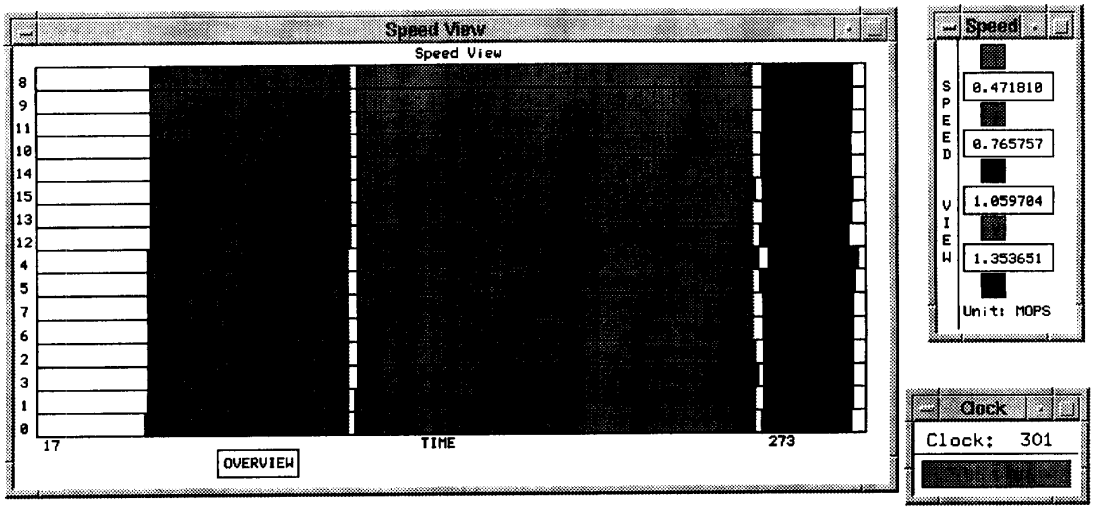


Fig. 17. Average speed chart and clock for Burg program execution, first iteration (16 processors).

The nice properties of the Burg algorithm make its scalability predictable. However, many practical algorithms behave like the SLALOM program in terms of having load imbalance degradation and communication cost varying with problem size. It is difficult to derive a general scalability formula for these algorithms. Performance measurement and visualization provide a feasible way to analyze and estimate the scalability of these algorithms.

D. Burg-Maspar and Slalom-Maspar Combinations

Next, we consider the scalability of the Burg and SLALOM programs on the MasPar computer. The Burg program employs a scattered decomposition form of data parallelism on the MasPar MP-1 computer [7]. The program has exhibited superior performance on the MasPar computer [7]. Machine sizes for this study include 1024 (or 1 K), 2048, 4096, and 8192 processors. The set of computed scalabilities (approach 2) for the Burg-MasPar combination is given in Table VIII. The asymptotic average unit speed calculated for the Burg–MasPar combination is 41 KFLOPS, giving a reference speed of 20.5 KFLOPS. The times used to compute the scalabilities are listed in Table IX. It is clear that while this scalability does vary similar to the Burg-nCUBE combination scalability, it is initially higher and decreases at a much slower rate. The more gradual decline is due to the larger computation to communication ratio of the MasPar system (e.g., 180 clock cycles / 30 clock cycles). Comparing the scalabilities of the Burg–nCUBE and the Burg–MasPar combinations, we can see the influence of the computing/communication ratio on scalability.

TABLE VIII
COMPUTED SCALABILITY OF BURG–MASPAR COMBINATION (APPROACH 2)

$\psi(N, N')$	1 K	2 K	4 K	8 K
1 K	1.00	0.93	0.89	0.83
2 K		1.00	0.95	0.90
4 K			1.00	0.94
8 K				1.00

TABLE IX
TIME VARIATION OF BURG–MASPAR COMBINATION (SECONDS)

system size	1 K	2 K	4 K	8 K
Time	0.0135	0.0145	0.0152	0.0162

TABLE X
COMPUTED SCALABILITY OF SLALOM–MASPAR COMBINATION (APPROACH 2)

$\psi(N, N')$	1 K	2 K	4 K	8 K
1 K	1.00	0.74	0.75	0.34
2 K		1.00	1.02	0.46
4 K			1.00	0.46
8 K				1.00

TABLE XI
TIME VARIATION OF SLALOM–MASPAR COMBINATION (SECONDS)

system size	1 K	2 K	4 K	8 K
Time	8.42	11.37	11.20	24.60

Since the validation of Theorem 1 is only dependent on the parallel algorithm's properties, (5) also could be used for predicting the scalability of the Burg–MasPar combination.

Table X gives the computed scalabilities of the SLALOM–MasPar combination. The asymptotic average unit speed calculated for this combination is 18 KFLOPS, giving a reference speed of 9 KFLOPS. Table XI lists the times used to compute the scalabilities. Recall that the MasPar computer has a two-dimensional toroidal mesh connection. When 1-K processors or 4-K processors are used, the underlying machine topology is a 32×32 or 64×64 square mesh, respectively. When 2-K processors or 8-K processors are used, the connection topology is a 32×64 or 64×128 rectangular mesh, respectively. As previously noted, the core computation of SLALOM is solving a dense $n \times n$ matrix by Gaussian elimination. The work load is imbalanced during the core computation. When the underlying machine connection is non-symmetric—as for 2-K and 8-K processors—the load imbalance is magnified and leads to lower performance (relative to the symmetric configuration). This, for example, accounts for the ψ(2 K, 4 K) value in Table XI, which is the scalability of a square machine with respect to a rectangular one. The SLALOM–MasPar combination has shown how the scalability could be influenced by the shape of the machine configuration. More work is needed in this area.

V. CONCLUSION

As more and more people accept massively-parallel computing as a practical approach for achieving high performance, more and more people start using the word *scalable*. The term has a reputation as a nice property of parallel machines and algorithms. However, with the lack of a clear definition, it is impossible to compare scalability over algorithms, architectures, and algorithm-machine combinations. We all know that scalability is the *ability to scale*, i.e., the ability to adjust according to a proportion (Webster's Dictionary definition). But, adjust what? to what proportion? These are the complicated parts which had not been clearly defined and often lead to confusion. In our study, the scalability of algorithm-machine combinations is carefully defined and a quantitative measurement is proposed. We adjust the problem size (or equivalently execution time) with the system size and proportion the adjustment to obtain a fixed average unit processor speed. We have shown that all three quantities, system size, problem size (work), and speed, are inter-related. Their relation reflects the inherent parallel degradations of the algorithm and architecture under consideration, and provides the scalability information of the algorithm-machine combination. Scalability is a metric which reveals aspects of the performance which are not easily discerned from other metrics. Scalability itself is not a measurement of parallel processing gain. It is a factor that contributes to the ability of a system to deliver the expected performance. It has valid uses for predicting the behavior of large scale computations.

The scalability variation of algorithm-machine combinations is also carefully studied. Scalability formulas have been derived for a relatively large class of algorithm-machine combinations. Three different approaches have been proposed to measure, compute, and predict scalability. Two algorithms have been investigated on an nCUBE 2 multicomputer and on a MasPar MP-1 parallel computer. The first algorithm, the Burg algorithm, has a nice structure. It satisfies the preconditions of our theoretical results. The experimental results match our theoretical results closely. The second algorithm, the SLALOM algorithm, has a more sophisticated structure. It has load imbalance degradation and a complicated communication pattern. It contains several computing phases. Each phase has a different speed and the relative duration of each phase varies with the system size. The SLALOM algorithm represents another class of algorithms. These algorithms have an intricate system size, problem size, and speed relation. There is no easy way to derive a satisfiable scalability formula for this type of algorithm. To overcome the lack of theoretical guidelines, performance instrumentation and visualization tools for multicomputers are used and developed to observe the algorithm behavior. Our experience shows that visualization provides an accurate and perceptible indication of the full-scale behavior of a parallel algorithm-machine combination.

ACKNOWLEDGMENT

We are indebted to several individuals at Ames Laboratory for their help with this research. In particular, J. Gustafson offered thoughtful discussion about the concept of scalability. J. Kim implemented our custom views for Paragraph. M. Carter provided assistance with the systems, especially to gather the performance data.

REFERENCES

[1] J. Gustafson, G. Montry, and R. Benner, "Development of parallel methods for a 1024-processor hypercube," *SIAM J. SSTC*, vol. 9, July 1988.

[2] J. Gustafson, "Reevaluating Amdahl's law," *Commun. ACM*, vol. 31, pp. 532–533, May 1988.

[3] V. Kumar and A. Gupta, "Analysis of scalability of parallel algorithms and architectures: A survey," in *Proc. Int. Conf. on Supercomputing*, June 1991.

[4] V. Kumar and V. Singh, "Scalability of parallel algorithms for the all-pairs shortest path problem: A summary of results," in *Proc. of Conf. on Parallel Processing*, Chicago, IL, 1990, pp. III 136–140.

[5] M. Willebeek-LeMair, A. P. Reeves, and C. H. Ning, "Characterization of multicomputer system: a transfer ration approach," in *Proc. Int. Conf. on Parallel Processing*, Chicago, IL, 1990, pp. II 171–178.

[6] D. Nussbaum and A. Agarwal, "Scalability of parallel machines," *Commun. the ACM*, vol. 34, no. 3, pp. 57–61, 1991.

[7] D. Rover, V. Tsai, Y. Chow, and J. Gustafson, "Signal processing algorithms on parallel architectures: A performance update," *J. Parallel Distrib. Computing*, vol. 13, pp. 237–245, Nov. 1991.

[8] C. Goral, K. Torrance, D. Greenberg, and B. Battaile, "Modeling the interaction of light between diffuse surfaces," *Comput. Graphics*, vol. 18, pp. 213–222, July 1984.

[9] X.-H. Sun and L. Ni, "Another view on parallel speedup," in *Proc. of Supercomputing'90*, NY, 1990, pp. 324–333.

[10] ——, "Scalable problems and memory-bounded speedup," *J. Parallel Distrib. Computing*, vol. 19, pp. 27–37, Sept. 1993.

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

[12] X.-H. Sun, L. Ni, F. Salam, and S. Guo, "Compute-exchange computation for solving power flow problems: The model and application," in *Proc. 4th SIAM Conf. Parallel Processing for Scientific Computing*, 1989, pp. 198–203.

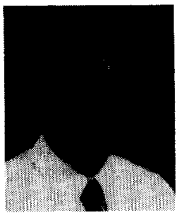
[13] R. W. Hockney, "Parametrization of computer performance," *Parellel Computing*, vol. 5, 1987.

[14] X.-H. Sun and J. Gustafson, "Toward a better parallel performance metric," *Parallel Computing*, vol. 17, Dec. 1991.

[15] J. Gustafson, D. Rover, S. Elbert, and M. Carter, "The design of a scalable, fixed-time computer benchmark," *J. Parallel Distrib. Computing*, vol. 11, Aug. 1991.

[16] G. Geist, M. Heath, B. Peyton, and P. Worley, "A machine-independent communication library," in *Proc. 4th Conf. Hypercubes, Concurrent Computers, and Applications*, 1989, pp. 565–568.

[17] M. T. Heath and J. A. Etheridge, "Visualizing the performance of parallel programs," *IEEE Software*, vol. 8, pp. 29–39, Sept. 1991.



Xian-He Sun received the B.S. degree in mathematics from Beijing Normal University, Beijing, China, the M.S. degree in mathematics and M.S. and Ph.D. degrees in computer science from Michigan State University.

After graduating from Michigan State, he joined the Ames Laboratory, operated for the Department of Energy by Iowa State University. During the year 1991–92, he was a visiting faculty member at Clemson University. He is currently a staff scientist at ICASE, NASA Langley Research Center. His research interests include parallel processing, parallel numerical algorithms, performance evaluation, and database systems.

Dr. Sun is a Guest Editor for the special issue of *Journal of Parallel and Distributed Computing* on Analyzing Scalability of Parallel Algorithms and Architectures. He is a member of the IEEE Computer Society and the Association for Computing Machinery.

Diane T. Rover received the B.S. degree in computer science in 1984, the M.S. degree in computer engineering in 1986, and the Ph.D. degree in computer engineering in 1989, all from Iowa State University.

From 1985 to 1988, she was awarded an IBM Graduate Fellowship. In 1986, she was an intern with McDonnell Douglas Corp. and in 1987, she was with the IBM Thomas J. Watson Research Center. Under a DOE Postdoctoral Fellowship from 1989 to 1991, she was a researcher in the Scalable Computing Laboratory at the Ames Laboratory. She is currently an Assistant Professor in the Department of Electrical Engineering at Michigan State University. Her research interests include parallel processing, computer architecture, reconfigurable hardware, and performance evaluation, instrumentation, and visualization.

With colleagues at the Ames Laboratory, Dr. Rover received an R&D 100 Award for the development of the Slalom benchmark. She is a member of the Program Committee for Supercomputing '94. She is a member of the IEEE Computer Society, ACM, ASEE, and the Society of Women Engineers.