To appear in Journal of Parallel and Distributed Computing, 1994.

Analyzing Scalability of Parallel Algorithms and Architectures*

Vipin Kumar and Anshul Gupta

Department of Computer Science
University of Minnesota
Minneapolis, MN - 55455

kumar@cs.umn.edu and *agupta@cs.umn.edu*

TR 91-18, November 1991 (Revised July 1993)

Abstract

The scalability of a parallel algorithm on a parallel architecture is a measure of its capacity to effectively utilize an increasing number of processors. Scalability analysis may be used to select the best algorithm-architecture combination for a problem under different constraints on the growth of the problem size and the number of processors. It may be used to predict the performance of a parallel algorithm and a parallel architecture for a large number of processors from the known performance on fewer processors. For a fixed problem size, it may be used to determine the optimal number of processors to be used and the maximum possible speedup that can be obtained. The objective of this paper is to critically assess the state of the art in the theory of scalability analysis, and motivate further research on the development of new and more comprehensive analytical tools to study the scalability of parallel algorithms and architectures. We survey a number of techniques and formalisms that have been developed for studying scalability issues, and discuss their interrelationships. For example, we derive an important relationship between time-constrained scaling and the isoefficiency function. We point out some of the weaknesses of the existing schemes for measuring scalability, and discuss possible ways of extending them.

*This work was supported by IST/SDIO through the Army Research Office grant # 28408-MA-SDI to the University of Minnesota and by the Army High Performance Computing Research Center at the University of Minnesota. An earlier version of this paper appears in the Proceedings of the 1991 International Conference on Supercomputing, Cologne, Germany, June 1991. A short version also appeared as an invited paper in the Proceedings of the 29th Annual Allerton Conference on Communication, Control and Computing, Urbana, IL, October 1991

1 Introduction

At the current state of technology, it is possible to construct parallel computers that employ thousands of processors. Availability of such systems has fueled interest in investigating the performance of parallel computers containing a large number of processors. While solving a problem in parallel, it is reasonable to expect a reduction in execution time that is commensurable with the amount of processing resources employed to solve the problem. The scalability of a parallel algorithm on a parallel architecture is a measure of its capacity to effectively utilize an increasing number of processors. Scalability analysis of a parallel algorithm-architecture combination can be used for a variety of purposes. It may be used to select the best algorithm-architecture combination for a problem under different constraints on the growth of the problem size and the number of processors. It may be used to predict the performance of a parallel algorithm and a parallel architecture for a large number of processors from the known performance on fewer processors. For a fixed problem size, it may be used to determine the optimal number of processors to be used and the maximum possible speedup that can be obtained. The scalability analysis can also predict the impact of changing hardware technology on the performance and thus help design better parallel architectures for solving various problems.

The objective of this paper is to critically assess the state of the art in the theory of scalability analysis, and to motivate further research on the development of new and more comprehensive analytical tools to study the scalability of parallel algorithms and architectures. We survey a number of techniques and formalisms that have been developed for studying the scalability issues, and discuss their interrelationships. We show some interesting relationships between the technique of isoefficiency analysis [29, 13, 31] and many other methods for scalability analysis. We point out some of the weaknesses of the existing schemes, and discuss possible ways of extending them.

The organization of this paper is as follows. Section 2 describes the terminology that is followed in the rest of the paper. Section 3 surveys various metrics that have been proposed for measuring the scalability of parallel algorithms and parallel architectures. Section 4 reviews the literature on the performance analysis of parallel systems. Section 5 describes the relationships among the various scalability metrics discussed in Section 3. Section 6 analyzes the impact of technology dependent factors on the scalability of parallel systems. Section 7 discusses scalability of parallel systems when the cost of scaling up a parallel architecture is also taken into account. Section 8 contains concluding remarks and directions for future research.

2 Definitions and Assumptions

Parallel System: The combination of a parallel architecture and a parallel algorithm implemented on it. We assume that the parallel computer being used is a homogeneous ensemble of processors; *i.e.*, all processors and communication channels are identical in speed.

Problem Size W: The size of a problem is a measure of the number of basic operations needed to solve the problem. There can be several different algorithms to solve the same problem. To keep the problem size unique for a given problem, we define it as the number of basic operations required by the fastest known sequential algorithm to solve the problem on a single processor. Problem size is a function of the size of the input. For example, for the problem of computing an N-point FFT, $W = \Theta(N \log N)$.

According to our definition, the sequential time complexity of the fastest known serial algorithm to solve a problem determines the size of the problem. If the time taken by an optimal (or the fastest known) sequential algorithm to solve a problem of size W on a single processor is T_S, then $T_S \propto W$, or $T_S = t_c W$, where t_c is a machine dependent constant.

Serial Fraction s**:** The ratio of the serial component of an algorithm to its execution time on one processor. The serial component of the algorithm is that part of the algorithm which cannot be parallelized and has to be executed on a single processor.

Parallel Execution Time T_P**:** The time elapsed from the moment a parallel computation starts, to the moment the last processor finishes execution. For a given parallel system, T_P is normally a function of the problem size (W) and the number of processors (p), and we will sometimes write it as $T_P(W, p)$.

Cost: The cost of a parallel system is defined as the product of parallel execution time and the number of processors utilized. A parallel system is said to be cost-optimal if and only if the cost is asymptotically of the same order of magnitude as the serial execution time (i.e., $pT_P = \Theta(W)$). Cost is also referred to as **processor-time product**.

Speedup S**:** The ratio of the serial execution time of the fastest known serial algorithm (T_S) to the parallel execution time of the chosen algorithm (T_P).

Total Parallel Overhead T_o**:** The sum total of all the overhead incurred due to parallel processing by all the processors. It includes communication costs, non-essential work and idle time due to synchronization and serial components of the algorithm. Mathematically, $T_o = pT_P - T_S$.

In order to simplify the analysis, we assume that T_o is a non-negative quantity. This implies that speedup is always bounded by p. For instance, speedup can be superlinear and T_o can be negative if the memory is hierarchical and the access time increases (in discrete steps) as the memory used by the program increases. In this case, the effective computation speed of a large program will be slower on a serial processor than on a parallel computer employing similar processors. The reason is that a sequential algorithm using M bytes of memory will use only $\frac{M}{p}$ bytes on each processor of a p-processor parallel computer. The core results of the paper are still valid with hierarchical memory, except that the scalability and performance metrics will have discontinuities, and their expressions will be different in different ranges of problem sizes. The flat memory assumption helps us to concentrate on the characteristics of the parallel algorithm and architectures, without getting into the details of a particular machine.

For a given parallel system, T_o is normally a function of both W and p and we will often write it as $T_o(W, p)$.

Efficiency E**:** The ratio of speedup (S) to the number of processors (p). Thus, $E = \frac{T_S}{pT_P} = \frac{1}{1+\frac{T_o}{T_S}}$.

Degree of Concurrency $\Gamma(W)$**:** The maximum number of tasks that can be executed simultaneously at any given time in the parallel algorithm. Clearly, for a given W, the parallel algorithm cannot use more than $\Gamma(W)$ processors. $\Gamma(W)$ depends only on the parallel algorithm, and is independent of the architecture. For example, for multiplying two $N \times N$ matrices using Fox's parallel matrix multiplication algorithm [12], $W = N^3$ and $\Gamma(W) = N^2 = W^{2/3}$. It is easily seen that if the processor-time product [1] is $\Theta(W)$ (i.e., the algorithm is **cost-optimal**), then $\Gamma(W) \leq \Theta(W)$.

3 Scalability Metrics for Parallel Systems

It is a well known fact that given a parallel architecture and a problem instance of a fixed size, the speedup of a parallel algorithm does not continue to increase with increasing number of processors. The speedup tends to saturate or peak at a certain value. As early as in 1967, Amdahl [2] made the observation that if s is the serial fraction in an algorithm, then its speedup is bounded by $\frac{1}{s}$, no matter how many processors are used. This statement, now popularly known as **Amdahl's Law**, has been used by Amdahl and others to argue against the usefulness of large scale parallel computers. Actually, in addition to the serial fraction, the speedup obtained by a parallel system depends upon a number of factors such as the degree of concurrency and overheads due to communication, synchronization, redundant work etc. For a fixed problem size, the speedup saturates either because the overheads grow with increasing number of processors or because the number of processors eventually exceeds the degree of concurrency inherent in the algorithm. In the last decade, there has been a growing realization that for a variety of parallel systems, given any number of processors p, speedup arbitrarily close to p can be obtained by simply executing the parallel algorithm on big enough problem instances [47, 31, 36, 45, 22, 12, 38, 41, 40, 58].

Kumar and Rao [31] developed a scalability metric relating the problem size to the number of processors necessary for an increase in speedup in proportion to the number of processors. This metric is known as the **isoefficiency function**. If a parallel system is used to solve a problem instance of a fixed size, then the efficiency decreases as p increases. The reason is that T_o increases with p. For many parallel systems, if the problem size W is increased on a fixed number of processors, then the efficiency increases because T_o grows slower than W. For these parallel systems, the efficiency can be maintained at some fixed value (between 0 and 1) for increasing p, provided that W is also increased. We call such systems **scalable**[1] parallel systems. This definition of scalable parallel algorithms is similar to the definition of parallel effective algorithms given by Moler [41].

For different parallel systems, W should be increased at different rates with respect to p in order to maintain a fixed efficiency. For instance, in some cases, W might need to grow as an exponential function of p to keep the efficiency from dropping as p increases. Such parallel systems are poorly scalable. The reason is that on these parallel systems, it is difficult to obtain good speedups for a large number of processors unless the problem size is enormous. On the other hand, if W needs to

[1]For some parallel systems (e.g., some of those discussed in [48] and [34]), the maximum obtainable efficiency E^{max} is less than 1. Even such parallel systems are considered scalable if the efficiency can be maintained at a desirable value between 0 and E^{max}.

grow only linearly with respect to p, then the parallel system is highly scalable. This is because it can easily deliver speedups proportional to the number of processors for reasonable problem sizes.

The rate at which W is required to grow w.r.t. p to keep the efficiency fixed can be used as a measure of scalability of the parallel algorithm for a specific architecture. If W must grow as $f_E(p)$ to maintain an efficiency E, then $f_E(p)$ is defined to be the isoefficiency function for efficiency E and the plot of $f_E(p)$ vs. p is called the isoefficiency curve for efficiency E. Equivalently, if the relation $W = f_E(p)$ defines the isoefficiency curve for a parallel system, then p should not grow faster that $f_E^{-1}(W)$ if an efficiency of at least E is desired.

In Kumar and Rao's framework, a parallel system is considered scalable if its isoefficiency function exists; otherwise the parallel system is unscalable. The isoefficiency function of a scalable system could, however, be arbitrarily large; i.e., it could dictate a very high rate of growth of problem size w.r.t. the number of processors. In practice, the problem size can be increased asymptotically only at a rate permitted by the amount of memory available at each processor. If the memory constraint does not allow the size of the problem to increase at the rate necessary to maintain a fixed efficiency, then the parallel system should be considered unscalable from a practical point of view.

Isoefficiency analysis has been found to be very useful in characterizing the scalability of a variety of parallel systems [17, 24, 19, 31, 32, 34, 46, 48, 56, 55, 18, 15, 33, 14, 30]. An important feature of isoefficiency analysis is that in a single expression, it succinctly captures the effects of characteristics of a parallel algorithm as well as the parallel architecture on which it is implemented. By performing isoefficiency analysis, one can test the performance of a parallel program on a few processors, and then predict its performance on a larger number of processors. For a tutorial introduction to the isoefficiency function and its applications, one is referred to [29, 13].

Gustafson, Montry and Benner [22, 20] were the first to experimentally demonstrate that by scaling up the problem size one can obtain near-linear speedup on as many as 1024 processors. Gustafson et al. introduced a new metric called **scaled speedup** to evaluate the performance on practically feasible architectures. This metric is defined as the speedup obtained when the problem size is increased linearly with the number of processors. If the scaled-speedup curve is good (e.g., close to linear w.r.t. the number of processors), then the parallel system is considered scalable. This metric is related to isoefficiency if the parallel algorithm under consideration has linear or near-linear isoefficiency function. In this case the scaled-speedup metric provides results very close to those of isoefficiency analysis, and the scaled-speedup is linear or near-linear with respect to the number of processors. For parallel systems with much worse isoefficiencies, the results provided by the two metrics may be quite different. In this case, the scaled-speedup vs. number of processors curve is sublinear.

Two generalized notions of scaled speedup were considered by Gustafson et al. [20], Worley [58] and Sun and Ni [50]. They differ in the methods by which the problem size is scaled up with the number of processors. In one method, the size of the problem is increased to fill the available memory on the parallel computer. The assumption here is that aggregate memory of the system will increase with the number of processors. In the other method, the size of the problem grows with p subject to an upper-bound on execution time. Worley found that for a large class of scientific problems, the time-constrained speedup curve is very similar to the fixed-problem-size speedup curve. He found that for many common scientific problems, no more than 50 processors can be used effectively in current generation multicomputers if the parallel execution time was to be kept fixed. For some other problems, Worley found time-constrained speedup to be close to linear, thus indicating that arbitrarily large instances of these problems can be solved in fixed time by simply increasing p. In [16], Gupta and Kumar identify the classes of parallel systems which yield linear and sublinear time-constrained speedup curves.

Karp and Flatt [27] introduced experimentally determined serial fraction f as a new metric for measuring the performance of a parallel system on a fix-sized problem. If S is the speedup on a p-processor system, then f is defined as $\frac{1/S-1/p}{1-1/p}$. The value of f is exactly equal to the serial fraction s of the algorithm if the loss in speedup is only due to serial component (i.e., if the parallel program has no other overheads). Smaller values of f are considered better. If f increases with the number of processors, then it is considered as an indicator of rising communication overhead, and thus an indicator of poor scalability. If the value of f decreases with increasing p, then Karp and Flatt [27] consider it to be an anomaly to be explained by phenomena such as superlinear speedup effects or cache effects. On the contrary, our investigation shows that f can decrease for perfectly normal programs. Assuming that the serial and the parallel algorithms are the same, f can be approximated by $\frac{T_o}{pT_S}$.[2] For a fixed W (and hence fixed T_S), f will decrease provided T_o grows slower than $\Theta(p)$. This happens for some fairly common algorithms such as parallel FFT on a SIMD hypercube [18][3]. Also, the parallel parallel algorithms for which f increases with p for a fixed W are not uncommon. For instance, for computing vector dot products on a hypercube [19] $T_o > \Theta(p)$ and hence f increases with p if W is fixed. But as shown in [19], this algorithm-architecture combination has an isoefficiency function of $\Theta(p \log p)$ and can be considered quite scalable.

Zorbas et al. [61] developed the following framework to characterize the scalability of parallel systems. A parallel algorithm with a serial component W_{serial} and a parallelizable component $W_{parallel}$, when executed on one processor, takes $t_c(W_{serial}+W_{parallel})$ time. Here t_c is a positive constant. The ideal execution time of the same algorithm on p processors should be $t_c(W_{serial}+\frac{W_{parallel}}{p})$. However, due to the overheads associated with parallelization, it always takes longer in practice. They next introduce the concept of an overhead function $\Phi(p)$. A p-processor system scales with overhead $\Phi(p)$ if the execution time T_P on p processors satisfies $T_P \leq t_c(W_{serial}+\frac{W_{parallel}}{p}) \times \Phi(p)$. The smallest function $\Phi(p)$ that satisfies this equation is called the systems overhead function and is defined by $\frac{T_P}{t_c(W_{serial}+W_{parallel}/p)}$. A parallel system is considered ideally scalable if the overhead function remains constant when the problem size is increased sufficiently fast w.r.t. p.

For any parallel system, if the problem size grows faster than or equal to the isoefficiency function, then $\Phi(p)$ will be a constant making the system (according to Zorbas' definition) ideally scalable. Thus all parallel systems for which the isoefficiency function exists, are scalable according to Zorbas' definition. If $\Phi(p)$ grows as a function of p, then the rate of increase of the overhead function determines the degree of unscalability of the system. The faster the increase, the more unscalable the system is considered. Thus, in a way, this metric is complimentary to the isoefficiency function. It distinguishes scalable parallel systems from unscalable ones, but does not provide any information

[2] Speedup $= S = \frac{T_S}{T_P} = \frac{T_S p}{T_S+T_o}$. For large p, $f = \frac{1}{S} - \frac{1}{p} = \frac{T_o+T_S}{pT_S} - \frac{1}{p} = \frac{T_o}{pT_S}$.

[3] The analysis in [18] can be adapted for SIMD hypercube by making the message startup time equal to zero.

on the degree of scalability of an ideally scalable system. On the other hand, the isoefficiency function provides no information on the degree of unscalability of an unscalable parallel system. A limitation of this metric is that $\Phi(p)$ captures overhead only due to communication, and not due to sequential parts. Even if a parallel algorithm is poorly scalable due to large sequential components, $\Phi(p)$ can be misleadingly low provided that the parallel system's communication overheads are low. For example, if $W_{serial} = W_S$ and $W_{parallel} = 0$, $\Phi(p) = \frac{T_P}{t_c(W_{serial}+W_{parallel}/p)} = \frac{1}{t_c} = \Theta(1)$ (i.e., the parallel system is perfectly scalable!).

Chandran and Davis [8] defined **processor efficiency function** (PEF) as the upper limit on the number of processors p than can be used to solve a problem of input size N such that the execution time on p processors is of the same order as the ratio of the sequential time to p; i.e., $T_P = \Theta(\frac{W}{p})$. An inverse of this function called **data efficiency function** (DEF) is defined as the smallest problem size on a given number of processors such that the above relation holds. The concept of data efficiency function is very similar to that of the isoefficiency function.

Kruskal et al. [28] defined the concept of **Parallel Efficient** (**PE**) problems which is related to the concept of isoefficiency function. The **PE** class of problems have algorithms with a polynomial isoefficiency function for some efficiency. The class PE makes an important delineation between algorithms with polynomial isoefficiencies and those with even worse isoefficiencies. Kruskal et al. proved the invariance of the class PE over a variety of parallel computation models and interconnection schemes. An important consequence of this result is that an algorithm with a polynomial isoefficiency on one architecture will have a polynomial isoefficiency on many other architectures as well. There can however be exceptions; for instance, it is shown in [18] that the FFT algorithm has a polynomial isoefficiency on a hypercube but an exponential isoefficiency on a mesh. As shown in [18, 48, 19], isoefficiency functions for a parallel algorithm can vary across architectures and understanding of this variation is of considerable practical importance. Thus, the concept of isoefficiency helps us in further sub-classifying the problems in class PE. It helps us in identifying more scalable algorithms for problems in the PE class by distinguishing between parallel systems whose isoefficiency functions are small or large polynomials in p.

Eager, Zahorajan and Lazowska [9] introduced the **average parallelism** measure to characterize the scalability of a parallel software system that consists of an acyclic directed graph of subtasks with a possibility of precedence constraints among them. The average parallelism A is defined as the average number of processors that are busy during the execution time of the parallel program, provided an unlimited number of them are available. Once A is determined, analytically or experimentally, the speedup and efficiency on a p-processor system are lower bounded by $\frac{pA}{(p+A-1)}$ and $\frac{A}{(p+A-1)}$ respectively. The reader should note that the average parallelism measure is useful only if the parallel system incurs no communication overheads (or if these overheads can be ignored). It is quite possible that a parallel algorithm with a large degree of concurrency is substantially worse than one with smaller concurrency and minimal communication overheads. Speedup and efficiency can be arbitrarily poor if communication overheads are present.

Marinescu and Rice [39] argue that a single parameter that depends solely on the nature of the parallel software is not sufficient to analyze a parallel system. The reason is that the properties of the parallel hardware can have a substantial effect on the performance of a parallel system. As an example, they point out the limitations of the approach of Eager et al. [9] in which a parallel system is characterized with the average parallelism of the software as a parameter. Marinescu and Rice develop a model to describe and analyze a parallel computation on a MIMD machine in terms of the number of threads of control p into which the computation is divided and the number of communication acts or events $g(p)$ as a function of p. An event is defined as an act of communication or synchronization. At a given time, a thread of control could either be actively performing the computation for which the parallel computer is being used, or it could be communicating or blocked. The speedup of the parallel computation can therefore be regarded as the average number of threads of control that are active in useful computation at any time. The authors conclude that for a fixed problem size, if $g(p) = \Theta(p)$, then the asymptotic speedup has a constant upper-bound. If $g(p) = \Theta(p^m)(m > 1)$, then the optimal speedup is obtained for a certain number of processors p_{opt} and asymptotically approaches zero as the number of processors is increased. For a given problem size, the value of p_{opt} is dependent upon $g(p)$. Hence $g(p)$ provides a signature of a parallel computation and p_{opt} can be regarded as a measure of scalability of the computation. The higher the number of processors that can be used optimally, the more scalable the parallel system is. The authors also discuss scaling the problem size linearly with respect to p and derive similar results for the upper-bounds on attainable speedup. When the number of events is a convex function of p, then the number of threads of control p_{smax} that yields the maximum speedup can be derived from the equation, $p = \frac{(\frac{W}{\theta}+g(p))}{g'(p)}$, where W is the problem size and θ is the work associated with each event. In their analysis, the authors regard θ, the duration of each event, as a constant. For many parallel algorithms, it is a function of p and W. Hence for these algorithms, the expression for p_{smax} derived under the assumption of a constant θ might yield a result quite different from the actual p_{smax}.

Van-Catledge [53] developed a model to determine the relative performance of parallel computers for an application w.r.t the performance on selected supercomputer configurations for the same application. The author presented a measure for evaluating the performance of a parallel computer called the **equal performance condition**. It is defined as the number of processors needed in the parallel computer to match its performance with that of a selected supercomputer. It is a function of the serial fraction of the base problem, scaling factor and the base problem size on which the scaling factor is applied. If the base problem size is W (in terms of the number of operations in the parallel algorithm), and s is the serial fraction, then the number of serial and parallel operations are sW and $(1-s)W$, respectively. Assume that by scaling up the problem size by a factor k, the number of serial and parallel operations becomes $G(k)sW$ and $F(k)(1-s)W$. If p' is the number of processors in the reference machine and t'_c is its clock period (in terms of the time taken to perform a unit operation of the algorithm), then the execution time of the problem on the reference machine is $T'_P = t'_c W(G(k)s + \frac{F(k)(1-s)}{p'})$. Similarly, on a test machine with clock period t_c, using p processors will result in an execution time of $T_P = t_c W(G(k)s + \frac{F(k)(1-s)}{p})$. Now the relative performance of the test machine w.r.t. the reference machine will be given by $\frac{T_P}{T'_P}$. The value of p for which the relative performance is one is used as a scalability measure. The fewer the number of processors that are required to match the performance of the parallel system with that of the selected supercomputer, the more scalable the parallel system is considered.

Van-Catledge considers a case in detail for which $G(k) = 1$ and $F(k) = k$. For this case, he computes the equal performance curves for a variety of combinations of s, k and t_c. For example it is shown that a uniprocessor system is better than a parallel computer having 1024 processors, each of which is 100 times slower, unless s is less than .01 or the scaling factor is very large. In another example it is shown that the equal performance curve for a parallel system with 8 processors is uniformly better than that of another system with 16 processors, each of which is half as fast. From these results, it is inferred that it is better to have a parallel computer with fewer faster processors than one with many slower processors. We discuss this issue further in Section 7 and show that this inference is valid only for a certain class of parallel systems.

In [49, 21], Sun and Gustafson argue that traditional speedup is an *unfair* measure, as it favors slow processors and poorly coded programs. They derive some *fair* performance measures for parallel programs to correct this. The authors define **sizeup** as the ratio of the size of the problem solved on the parallel computer to the size of the problem solved on the sequential computer in a given fixed amount of time. Consider two parallel computers M_1 and M_2 for which the cost of each serial operation is same but M_1 executes the parallelizable operations faster than M_2. They show that M_1 will attain poorer speedups (even if the communication overheads are ignored), but according to sizeup, M_1 will be considered better. Based on the concept of fixed time sizeup, Gustafson [21] has developed the SLALOM benchmark for a distortion-free comparison of computers of widely varying speeds.

Sun and Rover [51] define a scalability metric called the **isospeed** measure, which is the factor by which the problem size has to be increased so that the average unit speed of computation remains constant if the number of processors is raised from p to $p^{'}$. The average unit speed of a parallel computer is defined as its achieved speed ($\frac{W}{T_P}$) divided by the number of processors. Thus, if the number of processors is increased to $p^{'}$ from p, then isospeed$(p, p^{'}) = \frac{p^{'}W}{pW^{'}}$. The problem size $W^{'}$ required for $p^{'}$ processors is determined by the isospeed. For a perfectly parallelizable algorithm with no communication, isospeed$(p, p^{'}) = 1$ and $W^{'} = \frac{p^{'}W}{p}$. For more realistic parallel systems, isospeed$(p, p^{'}) < 1$ and $W^{'} > \frac{p^{'}W}{p}$.

The class **PC*** of problems defined by Vitter and Simons [54] captures a class of problems with efficient parallel algorithms on a PRAM. Informally, a problem in class **P** (the polynomial time class) is in PC* if it has a parallel algorithm on a PRAM that can use a polynomial (in terms of input size) number of processors and achieve some minimal efficiency ϵ. Any problem in PC* has at least one parallel algorithm such that for some efficiency E, its isoefficiency function exists and is a polynomial. Between two efficient parallel algorithms in this class, obviously the one that is able to use more processors is considered superior in terms of scalability.

Nussbaum and Agarwal [43] defined scalability of a parallel architecture for a given algorithm as the ratio of the algorithm's asymptotic speedup when run on the architecture in question to its corresponding asymptotic speedup when run on an EREW PRAM. The asymptotic speedup is the maximum obtainable speedup for a fixed problem size given an unlimited number of processors. This metric captures the communication overheads of an architecture by comparing the performance of a given algorithm on it to the performance of the same algorithm on an ideal communication free architecture. Informally, it reflects *"the fraction of the parallelism inherent in a given algorithm that* *can be exploited by any machine of that architecture as a function of problem size"* [43]. Note that this metric cannot be used to compare two algorithm-architecture pairs for solving the same problem or to characterize the scalability (or unscalability) of a parallel algorithm. It can only be used to compare parallel architectures for a given parallel algorithm. For any given parallel algorithm, its optimum performance on a PRAM is fixed. Thus, comparing two architectures for the problem is equivalent to comparing the minimum execution times for a given problem size on these architectures. In this context, this work can be categorized with that of other researchers who have investigated the minimum execution time as a function of the size of the problem on an architecture with an unlimited number of processors available [58, 16, 11].

According to Nussbaum and Agarwal, the scalability of an algorithm is captured in its performance on the PRAM. Some combination of the two metrics—the asymptotic speedup of the algorithm on the PRAM, and the scalability of the parallel architecture—can be used as a combined metric for the algorithm-architecture pair. A natural combination would be the product of the two, which is equal to the maximum obtainable speedup (lets call it $S^{max}(W)$) of the parallel algorithm on the architecture under consideration for a problem size W given arbitrarily many processors. In the best case, $S^{max}(W)$ can grow linearly with W. In the worst case, it could be a constant. The faster it grows with respect to W, the more scalable the parallel system should be considered. This metric can favor parallel systems for which the processor-time product is worse as long as they run faster. For example, for the all-pairs shortest path problem, this metric will favor the parallel algorithm [1] based upon an inefficient $\Theta(N^3 \log N)$ serial algorithm over the parallel algorithms [34, 25] that are based on Floyd's $\Theta(N^3)$ algorithm. The reason is that $S^{max}(W)$ is $\Theta(N^3)$ for the first algorithm and is $\Theta(N^2)$ for the second one.

4 Performance Analysis of Large Scale Parallel Systems

Given a parallel system the speedup on a problem of fixed size may peak at a certain limit because the overheads may grow faster than the additional computation power offered by increasing the number of processors. A number of researchers have analyzed the optimal number of processors required to minimize parallel execution time for a variety of problems [11, 42, 52, 38].

Flatt and Kennedy [11, 10] derive some important upper-bounds related to the performance of parallel computers in the presence of synchronization and communication overheads. They show that if the overhead function satisfies certain properties, then there exists a unique value p_0 of the number of processors for which T_P, for a given W, is minimum. However, for this value of p, the efficiency of the parallel system is rather poor. Hence, they suggest that p should be chosen to maximize the product of efficiency and speedup[4] (which is equivalent to maximizing the ratio of efficiency to parallel execution time) and analytically compute the optimal values of p. A major assumption in their analysis is that the per-processor overhead $t_o(W, p)$ ($t_o(W, p) = \frac{T_o(W,p)}{p}$) grows faster than $\Theta(p)$. As discussed in [16], this assumption limits the range of the applicability of the analysis. Further, the

[4]They later suggest maximizing a combination of the number of processors and the parallel execution time. They consider the weighted geometric mean F_x of E and S. For a given problem size, $F_x(p) = (E(p))^x(S(p))^{2-x}$, where $0 < x < 2$.

analysis in [16] reveals that the better a parallel algorithm is (i.e., the slower t_o grows with p), the higher the value of p_0. For many algorithm architecture combinations, p_0 exceeds the limit imposed by the degree of concurrency on the number of processors that can be used [16]. Thus the theoretical value of p_0 and the efficiency at this point may not be useful in studying many parallel algorithms.

Flatt and Kennedy [11] also discuss scaling up the problem size with the number of processors. They define scaled speedup as k times the ratio of $T_P(W, p)$ and $T_P(kW, kp)$. Under the simplifying assumptions that t_o depends only on the number of processors[5] and the number of serial steps in the algorithm remain constant as the problem size is increased, they prove that there is an upper-bound of $\frac{k}{T_P(kW,kp)}$ on the scaled speedup. They argue that since logarithmic per-processor overhead is the best possible, the upper-bound on the scaled speedup is $\Theta(\frac{k}{\log k})$. The reader should note that there are many important natural parallel systems where the per-processor overhead t_o is smaller than $\Theta(\log p)$. For instance, in two of the three benchmark problems used in the Sandia experiment [22], the per-processor overhead was constant.

Eager et al. [9] use the the average parallelism measure to locate the position (in terms of number of processors) of the **knee** in the execution time-efficiency curve. The knee occurs at the same value of p for which the ratio of efficiency to execution time is maximized. Determining the location of the knee is important when the same parallel computer is being used for many applications so that the processors can be partitioned among the applications in an optimal fashion.

Tang and Li [52] prove that maximizing the efficiency to T_P ratio is equivalent to minimizing $p(T_P)^2$. They propose minimization of a more general expression; i.e., $p(T_P)^r$. Here r determines the relative importance of efficiency and the parallel execution time. The choice of a higher r implies that reducing T_P is given more importance. Consequently, the system utilizes more processors, thereby operating at a low efficiency. A low value of r means greater emphasis on improving the efficiency than on reducing T_P; hence, the operating point of the system will use fewer processors and yield a high efficiency.

In [16], for a fairly general class of overhead functions, Gupta and Kumar analytically determine the optimal number of processors to minimize T_P as well as the more general metric $p(T_P)^r$. It is then shown that for a wide class of overhead functions, minimizing any of these metrics is asymptotically equivalent to operating at a fixed efficiency that depends only on the overhead function of the parallel system and the value of r.

As a refinement of the model of Gustafson et al., Zhou [60] and Van Catledge [53] present models that predict performance as a function of the serial fraction of the parallel algorithm. They conclude that by increasing the problem size, one can obtain speedups arbitrarily close to the number of processors. However, the rate at which the problem size needs to be scaled up in order to achieve this depends on the serial component of the algorithm used to solve the problem. If the serial component is a constant, then the problem size can be scaled up linearly with the number of processors in order to get arbitrary speedups. In this case, isoefficiency analysis also suggests that a linear increase in problem size w.r.t. p is sufficient to maintain any arbitrary efficiency. If the serial component grows with the problem size, then the problem size may have to be increased faster than linearly w.r.t. the number of processors. In this case, isoefficiency analysis not only suggests that a higher rate of

[5] Very often, apart from p, t_o is also a function of W.

growth of W w.r.t p will be required, but also provides an expression for this rate of growth, thus answering the question posed in [60] as to how fast the problem should grow to attain speedups close to p.

The analysis in [60] and [53] does not take the communication overheads into account. The serial component W_{serial} of an algorithm contributes $W_{serial}(p-1) \approx pW_{serial}$ to the total overhead cost of parallel execution using p processors. The reason is that while one processor is busy on the sequential part, the remaining $(p-1)$ are idle. Thus the serial component can model the communication overhead as long as the overhead grows linearly with p. However, if the overhead due to communication grows faster or slower than $\Theta(p)$, as is the case with many parallel systems, the models based solely upon sequential vs. parallel component are not adequate.

Carmona and Rice [6, 7] provide new and improved interpretations for the parameters commonly used in the literature such as serial fraction, and the portion of time spent on performing serial work on a parallel system, etc. The dynamics of parallel program characteristics like speedup and efficiency are then presented as a function of these parameters for several theoretical and empirical examples.

5 Relation Between Some Scalability Measures

After reviewing these various measures of scalability, one may ask whether there exists one measure that is better than all others [23]? The answer to this question is no, as different measures are suitable for different situations.

One situation arises when the problem at hand is fixed and one is trying to use an increasing number of processors to solve it. In this case, the speedup is determined by the serial fraction in the program as well as other overheads such as those due to communication and due to redundant work. In this situation choosing one parallel system over the other can be done using the standard speedup metric. Note that for any fixed problem size W, the speedup on a parallel system will saturate or peak at some value $S^{max}(W)$, which can also be used as a metric. Scalability issues for the fixed problem size case are addressed in [11, 27, 16, 42, 52, 58].

Another possible scenario is that in which a parallel computer with a fixed number of processors is being used and the best parallel algorithm needs to be chosen for solving a particular problem. For a fixed p, the efficiency increases as the problem size is increased. The rate at which the efficiency increases and approaches one (or some other maximum value) with respect to increase in problem size may be used to characterize the quality of the algorithm's implementation on the given architecture.

The third situation arises when the additional computing power due to the use of more processors is to be used to solve bigger problems. Now the question is how should the problem size be increased with the number of processors?

For many problem domains, it is appropriate to increase the problem size with the number of processors so that the total parallel execution time remains fixed. An example is the domain of weather forecasting. In this domain, the size of the problem can be increased arbitrarily provided that the problem can be solved within a specified time (e.g., it does not make sense to take more than 24 hours to forecast the next day's weather). The scalability issues for such problems have been explored by Worley [58], Gustafson [22, 20], and Sun and Ni [50].

Another extreme in scaling the problem size is to try as big problems as can be handled in the memory. This is investigated by Worley [57, 58, 59], Gustafson [22, 20] and by Sun and Ni [50], and is called the memory-constrained case. Since the total memory of a parallel computer increases with increasing p, it is possible to solve bigger problems on parallel computer with bigger p. It should also be clear that any problem size for which the memory requirement exceeds the total available memory cannot be solved on the system.

An important scenario is that in which one is interested in making efficient use of the parallel system; i.e., it is desired that the overall performance of the parallel system increases linearly with p. Clearly, this can be done only for scalable parallel systems, which are exactly the ones for which a fixed efficiency can be maintained for arbitrarily large p by simply increasing the problem size. For such systems, it is natural to use isoefficiency function or related metrics [31, 28, 8]. The analyses in [60, 61, 11, 38, 42, 52, 9] also attempt to study the behavior of a parallel system with some concern for overall efficiency.

Although different scalability measures are appropriate for rather different situations, many of them are related to each other. For example, from the isoefficiency analysis, one can reach a number of conclusions regarding the time-constrained case (i.e., when bigger problems are solved on larger parallel computers with some upper-bound on the parallel execution time). It can be shown that for cost-optimal algorithms, the problem size can be increased linearly with the number of processors while maintaining a fixed execution time if and only if the isoefficiency function is $\Theta(p)$. The proof is as follows. Let $\Gamma(W)$ be the degree of concurrency of the algorithm. Thus, as p is increased, W has to be increased at least as $\Theta(p)$, or else p will eventually exceed $\Gamma(W)$. Note that $\Gamma(W)$ is upper-bounded by $\Theta(W)$ and p is upper-bounded by $\Gamma(W)$. T_P is given by $\frac{T_S}{p} + \frac{T_o(W,p)}{p} = \frac{t_c W}{p} + \frac{T_o(W,p)}{p}$. Now consider the following two cases. Let the first case be when $\Gamma(W)$ is smaller than $\Theta(W)$. In this case, even if as many as $\Gamma(W)$ processors are used, the term $\frac{t_c W}{\Gamma(W)}$ of the expression for T_P will diverge with increasing W, and hence, it is not possible to continue to increase the problem size and maintain a fixed parallel execution time. At the same time, the overall isoefficiency function grows faster than $\Theta(p)$ because the isoefficiency due to concurrency exceeds $\Theta(p)$. In the second case in which $\Gamma(W) = \Theta(W)$, as many as $\Theta(W)$ processors can be used. If $\Theta(W)$ processors are used, then the first term in T_P can be maintained at a constant value irrespective of W. The second term in T_P will remain constant if and only if $\frac{T_o(W,p)}{p}$ remains constant when $p = \Theta(W)$ (in other words, $\frac{T_o}{W}$ remains constant while p and W are of the same order). This condition is necessary and sufficient for linear isoefficiency.

A direct corollary of the above result is that if the isoefficiency function is greater than $\Theta(p)$, then the minimum parallel execution time will increase even if the problem size is increased as slowly as linearly with the number of processors. Worley [57, 58, 59] has shown that for many algorithms used in the scientific domain, for any given T_P, there will exist a problem size large enough so that it cannot be solved in time T_P, no matter how many processors are used. Our above analysis shows that for these parallel systems, the isoefficiency curves have to be worse than linear. It can be easily shown that the isoefficiency function will be greater than $\Theta(p)$ for any algorithm-architecture combination for which $T_o > \Theta(p)$ for a given W. The latter is true when any algorithm with a global operation (such as broadcast, and one-to-all and all-to-all personalized communication [5, 26]) is implemented on a parallel architecture that has a message passing latency or message startup time. Thus, it can be concluded that *for any cost-optimal parallel algorithm involving global communication, the problem size cannot be increased indefinitely without increasing the execution time on a parallel computer having a startup latency for messages, no matter how many processors are used (up to a maximum of W).* This class of algorithms includes some fairly important algorithms such as matrix multiplication (all-to-all/one-to-all broadcast) [15], vector dot products (single node accumulation) [19], shortest paths (one-to-all broadcast) [34], and FFT (all-to-all personalized communication) [18], etc. The readers should note that the presence of a global communication operation in an algorithm is a sufficient but not a necessary condition for non-linear isoefficiency on an architecture with message passing latency. Thus, the class of algorithms having the above mentioned property is not limited to the algorithms with global communication.

If the isoefficiency function of a parallel system is greater than $\Theta(p)$, then given a problem size W, there is as lower-bound on the parallel execution time. This lower-bound (lets call it $T_P^{min}(W)$) is a non-decreasing function of W. The rate at which the $T_P^{min}(W)$ for a problem (given arbitrarily many processors) must increase with the problem size can also serve as a measure of scalability of the parallel system. In the best case, $T_P^{min}(W)$ is constant; *i.e.*, larger problems can be solved in a fixed amount of time by simply increasing the number of processors. In the worst case, $T_P^{min}(W) = \Theta(W)$. This happens when the degree of effective parallelism is constant. The slower $T_P^{min}(W)$ grows as a function of the problem size, the more scalable the parallel system is. T_P^{min} is closely related to $S^{max}(W)$ defined in the context of Nussbaum and Agarwal's work in Section 3. For a problem size W, these two metrics are related by $W = S^{max}(W) \times T_P^{min}(W)$. Let $\xi(W)$ be the number of processors that should be used for obtaining the minimum parallel execution time $T_P^{min}(W)$ for a problem of size W. Clearly, $T_P^{min}(W) = \frac{W}{\xi(W)} + \frac{T_o(W,\xi(W))}{\xi(W)}$. Using $\xi(W)$ processors leads to optimal parallel execution time $T_P^{min}(W)$, but may not lead to minimum pT_P product (or the cost of parallel execution). Now consider the cost-optimal implementation of the parallel system (i.e., when the number of processors used for a given problem size is governed by the isoefficiency function). In this case, if $f(p)$ is the isoefficiency function, then T_P is given by $\frac{W}{f^{-1}(W)} + \frac{T_o(W,f^{-1}(W))}{f^{-1}(W)}$ for a fixed W. Let us call this $T_P^{iso}(W)$. Clearly, $T_P^{iso}(W)$ can be no better than $T_P^{min}(W)$. In [16], it is shown that for a fairly wide class of overhead functions, the relation between problem size W and the number of processors p at which the parallel run time T_P is minimized, is given by the isoefficiency function for some value of efficiency. Hence, $T_P^{iso}(W)$ and $T_P^{min}(W)$ have the same asymptotic growth rate w.r.t. to W for these types of overhead functions. For these parallel systems, no advantage (in terms of asymptotic parallel execution time) is gained by making inefficient use of the parallel processors.

Several researchers have proposed to use an operating point where the value of $p(T_P)^r$ is minimized for some constant r and for a given problem size W [11, 9, 52]. It can be shown [52] that this corresponds to the point where ES^{r-1} is maximized for a given problem size. Note that the location of the minima of $p(T_P)^r$ (with respect to p) for two different algorithm-architecture combinations can be used to choose one between the two. In [16], it is also shown that for a fairly large class of overhead functions, the relation between the problem size and p for which the expression $p(T_P)^r$ is minimum for that problem size, is given by the isoefficiency function (to maintain a particular efficiency E which is a function of r) for the algorithm and the architecture being used.

6 Impact of Technology Dependent Factors on Scalability

A number of researchers have considered the impact of the changes in CPU and communication speeds on the performance of a parallel system. It is clear that if the CPU or the communication speed is increased, the overall performance can only become better. But unlike a sequential computer, in a parallel system, a k-fold increase in the CPU speed may not necessarily result in a k-fold reduction in execution time for the same problem size.

For instance, consider the implementation of the FFT algorithm on a SIMD hypercube [18]. The efficiency of a N-point FFT computation on a p-processor cube is given by $E = \frac{1}{1+\frac{t_w}{t_c}\frac{\log p}{\log N}}$. Here t_w is the time required to transfer one word of data between two directly connected processors and t_c is the time required to perform a unit FFT computation. In order to maintain an efficiency of E, the isoefficiency function of this parallel system is given by $\frac{E}{1-E} t_w p^{\frac{E}{1-E}\frac{t_w}{t_c}} \log p$. Clearly, the scalability of the system deteriorates very rapidly with the increase in the value of $\frac{E}{1-E}\frac{t_w}{t_c}$. For a given E, this can happen either if the computation speed of the processors is increased (i.e., t_c is reduced) or the communication speed is decreased (i.e., t_w is increased). For example, if $t_w = t_c = 1$, the isoefficiency function for maintaining an efficiency of 0.5 is $p \log p$. If p is increased 10 times, then the problem size needs to grow $10\sqrt{10}$ times to maintain the same efficiency. On the other hand, a 10-fold increase in t_w or a 10-fold decrease in t_c will require one to solve a problem of size nearly equal to W^{10} if W was the problem size required to get the same efficiency on the original machine. Thus, technology dependent constants like t_c and t_w can have a dramatic impact on the scalability of a parallel system.

In [48] and [15], the impact of these parameters is discussed on the scalability of parallel shortest path algorithms, and matrix multiplication algorithms, respectively. In these algorithms, the isoefficiency function does not grow exponentially (as in the case of FFT) with respect to the $\frac{t_w}{t_c}$ ratio, but is a polynomial function of this ratio. For example, in the best-scalable parallel implementations of matrix multiplication algorithm on the mesh [15], the isoefficiency function is proportional to $(\frac{t_w}{t_c})^3$. The implication of this is that for the same communication speed, using ten times faster processors in the parallel computer will require a 1000 fold increase in the problem size to maintain the same efficiency. On the other hand, if p is increased by a factor of 10, then the same efficiency can be obtained on $10\sqrt{10}$ times bigger problems. Hence, for parallel matrix multiplication, it is better to have a parallel computer with k-fold as many processors rather than one with the same number of processors, each k-fold as fast (assuming that the communication network and the bandwidth etc. remain the same).

Nicol and Willard [42] consider the impact of CPU and communication speeds on the peak performance of parallel systems. They study the performance of various parallel architectures for solving Elliptic Partial Differential Equations. They show that for this application, a k-fold increase in the speed of the CPUs results in an improvement in the optimal execution time by a factor of $k^{\frac{1}{3}}$ on a bus architecture if the communication speed is not improved. They also show that improving only the bus speed by a factor of k reduces the optimal execution time by a factor of $k^{\frac{2}{3}}$.

Kung [35] studied the impact of increasing CPU speed (while keeping the I/O speed fixed) on the memory requirement for a variety of problems. It is argued that if the CPU speed is increased without a corresponding increase in the I/O bandwidth, the system will become imbalanced leading to idle time during computations. An increase in the CPU speed by a factor of α w.r.t. the I/O speed requires a corresponding increase in the size of the memory of the system by a factor of $H(\alpha)$ in order to keep the system balanced. As an example it has been shown that for matrix multiplication, $H(\alpha) = \alpha^2$ and for computations such as relaxation on a d-dimensional grid, $H(\alpha) = \alpha^d$. The reader should note that similar conclusions can be shown to hold for a variety of parallel computers if only the CPU speed of the processors is increased without changing the bandwidth of the communication channels.

The above examples show that improving the technology in only one direction (i.e., towards faster CPUs) may not be a wise idea. The overall execution time will, of course, reduce by using faster processors, but the speedup with respect to the execution time on a single fast processor will decrease. In other words, increasing the speed of the processors alone without improving the communication speed will result in diminishing returns in terms of overall speedup and efficiency. The decrease in performance is highly problem dependent. For some problems it is dramatic (e.g., parallel FFT [18]), while for others (e.g., parallel matrix multiplication) it is less severe.

7 Study of Cost Effectiveness of Parallel Systems

So far we have confined the study of scalability of a parallel system to investigating the capability of the parallel algorithm to effectively utilize an increasing number of processors on the parallel architecture. Many algorithms may be more scalable on more costly architectures (i.e., architectures that are cost-wise less scalable due to high cost of expanding the architecture by each additional processor). For example, the cost of a parallel computer with p processors organized in a mesh configuration is proportional to $4p$, whereas a hypercube configuration of the same size will have a cost proportional to $p \log p$ because a p processor mesh has $4p$ communication links while a p processor hypercube has $p \log p$ communication links. It is assumed that the cost is proportional to the number of links and is independent of the length of the links. In such situations, one needs to consider whether it is better to have a larger parallel computer of a cost-wise more scalable architecture that is underutilized (because of poor efficiency), or is it better to have a smaller parallel computer of a cost-wise less scalable architecture that is better utilized. For a given amount of resources, the aim is to maximize the overall performance which is proportional to the number of processors and the efficiency obtained on them. It is shown in [18] that under certain assumptions, it is more cost-effective to implement the FFT algorithm on a hypercube rather than a mesh despite the fact that large scale meshes are cheaper to construct than large hypercubes. On the other hand, it is quite possible that the implementation of a parallel algorithm is more cost-effective on an architecture on which the algorithm is less scalable. Hence this type of cost analysis can be very valuable in determining the most suitable architecture for a class of applications.

Another issue in the cost vs. performance analysis of parallel systems is to determine the tradeoffs between the speed of processors and the number of processors to be employed for a given budget.

From the analysis in [3] and [53], it may appear that higher performance is always obtained by using fewer and faster processors. It can be shown that this is true only for those parallel systems in which the overall overhead T_o grows faster than or equal to $\Theta(p)$. The model considered in [53] is one for which $T_o = \Theta(p)$, as they consider the case of a constant serial component. As an example of a parallel system where their conclusion is not applicable, consider the matrix multiplication on a SIMD mesh architecture [1]. Here $T_o = t_w N^2 \sqrt{p}$ for multiplying $N \times N$ matrices. Thus $T_P = t_c \frac{N^3}{p} + t_w \frac{N^2}{\sqrt{p}}$. Clearly, a better speedup can be obtained by increasing p by a factor of k (until p reaches N^2) rather than reducing t_c by the same factor.

Even for the case in which T_o grows faster than or equal to $\Theta(p)$, it might not always be cost-effective to have fewer faster processors. The reason is that the cost of a processor increases very rapidly with respect to speed once a technology dependent threshold is reached. For example, the cost of a processor with 2 ns clock speed is far more than 10 times the cost of a processor with 20 ns clock speed given today's technology. Hence, it may be more cost effective to get higher performance by having a large number of relatively less utilized processors than to have a small number of processors whose speed is beyond the current technology threshold.

Some analysis of the cost-effectiveness of parallel computers is given by Barton and Withers in [3]. They define cost C of an ensemble of p processors to be equal to dpV^b, where V is the speed of each individual processor in FLOPS (floating point operations per second), d and b are constants, b is typically greater than 1. Now for a given fixed cost and a specific problem size, the actual delivered speed in FLOPS is given by $\frac{Kp^r}{1+(p-1)f+\frac{Kp^r t_c(p)}{W}}$, where s is the serial fraction, $K = (\frac{C}{d})^{\frac{1}{b}}$ is a constant, W is the problem size in terms of number of floating point operations, $r = \frac{b-1}{b}$ and $t_c(p)$ denotes the time spent in interprocessor communication etc. From this expression, it can be shown that for a given cost and a fixed problem instance, the delivered FLOPS peaks for a certain number of processors.

As noted in [3], this analysis is valid only for a fixed problem size. Actually, the value of p for peak performance also increases as the problem size is increased. Thus if the problem is scaled properly, it could still be beneficial to use more and more processors rather than opting for fewer faster processors. The relation between the problem size and the number of processors at which the peak performance is obtained for a given C, could also serve as a measure of scalability of the parallel system under the fixed cost paradigm. For a fixed cost, the slower the rate at which the problem size has to increase with increasing the number of processors that yield maximum throughput, the more scalable the algorithm-architecture combination should be considered.

The study of cost related issues of a parallel system is quite complicated because the cost of a parallel computer depends on many different factors. Typically, it depends on the number of processors used, the configuration in which they are connected (this determines the number and the length of communication links), the speed of processors and the speed of the communication channels, and usually it is a complex non-linear function of all these factors. For a given cost, its optimal distribution among the different components (e.g., processors, memory, cache, communication network, etc.) of the parallel computer depends on the computation and communication pattern of the application and the size of the problem to be solved.

While studying the performance of a parallel system, its efficiency E is often considered as an important attribute. Obtaining or maintaining a good efficiency as the parallel system is scaled up is an important concern of the implementors of the parallel system and often the ease with which this objective can be met determines the degree of scalability of the parallel system. In fact the term commonly known as efficiency is a loose term and it should more precisely be called processor-efficiency because it is measure of the efficiency with which the processors in the parallel computer are being utilized. Usually a parallel computer using p processors is not p times as costly as a sequential computer with a similar processor. Therefore, processor-efficiency is not a measure of the efficiency with which the money invested in building a parallel computer is being utilized. If using p processors results in a speedup of S over a single processor, then $\frac{S}{p}$ gives the processor-efficiency of the parallel system. Analogously, if an investment of $C(p)$ dollars yields a speedup of S, then assuming unit cost for a single processor system, the ratio $\frac{S}{C(p)}$ should determine the efficiency with which the *cost* of building a parallel computer with p processors is being utilized. We can define $\frac{S}{C(p)}$ as the *cost-efficiency* E_C of the parallel system, which, from a practical point of view, might be a more useful and insightful measure to compare two parallel systems.

8 Concluding Remarks

Significant progress has been made in identifying and developing measures of scalability in the last few years. These measures can be used to analyze whether parallel processing can offer desired performance improvement for problems at hand. They can also help guide the design of large-scale parallel computers. It seems clear that no single scalability metric would be better than all others. Different measures will be useful under different contexts and further research is needed along several directions. Nevertheless, a number of interesting properties of several metrics and parallel systems have been identified. For example, we show that a cost-optimal parallel algorithm can be used to solve arbitrarily large problem instances in a fixed time if and only if its isoefficiency is linear. Arbitrarily large instances of problems involving global communication cannot be solved in constant time on a parallel machine with message passing latency. To make efficient use of processors while solving a problem on a parallel computer, it is necessary that the number of processors be governed by the isoefficiency function. If more processors are used, the problem might be solved faster, but the parallel system will not be utilized efficiently. For a wide class of parallel systems identified in [16], using more processors than determined by the isoefficiency function does not help in reducing the parallel time complexity of the algorithm.

The introduction of hardware cost factors (in addition to speedup and efficiency) in the scalability analysis is important so that the overall cost-effectiveness can be determined. The current work in this direction is still very preliminary. Another problem that needs much attention is that of partitioning of processor resources among different problems. If two problems are to be solved on a multicomputer and one of them is poorly scalable while the other one has very good scalability characteristics on the given machine, then clearly, allocating more processors to the more scalable problem tends to optimize the efficiency, but not the overall execution time. The problem of partitioning becomes more complex as the number of different computations is increased. Some early work on this topic has been reported in [4, 44, 37, 38].

References

[1] S. G. Akl. *The Design and Analysis of Parallel Algorithms.* Prentice-Hall, Englewood Cliffs, NJ, 1989.

[2] G. M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *AFIPS Conference Proceedings*, pages 483–485, 1967.

[3] M. L. Barton and G. R. Withers. Computing performance as a function of the speed, quantity, and the cost of processors. In *Supercomputing '89 Proceedings*, pages 759–764, 1989.

[4] Krishna P. Belkhale and Prithviraj Banerjee. Approximate algorithms for the partitionable independent task scheduling problem. In *Proceedings of the 1990 International Conference on Parallel Processing*, pages I72–I75, 1990.

[5] D. P. Bertsekas and J. N. Tsitsiklis. *Parallel and Distributed Computation: Numerical Methods.* Prentice-Hall, Englewood Cliffs, NJ, 1989.

[6] E. A. Carmona and M. D. Rice. A model of parallel performance. Technical Report AFWL-TR-89-01, Air Force Weapons Laboratory, 1989.

[7] E. A. Carmona and M. D. Rice. Modeling the serial and parallel fractions of a parallel algorithm. *Journal of Parallel and Distributed Computing*, 1991.

[8] S. Chandran and Larry S. Davis. An approach to parallel vision algorithms. In R. Porth, editor, *Parallel Processing.* SIAM, Philadelphia, PA, 1987.

[9] D. L. Eager, J. Zahorjan, and E. D. Lazowska. Speedup versus efficiency in parallel systems. *IEEE Transactions on Computers*, 38(3):408–423, 1989.

[10] Horace P. Flatt. Further applications of the overhead model for parallel systems. Technical Report G320-3540, IBM Corporation, Palo Alto Scientific Center, Palo Alto, CA, 1990.

[11] Horace P. Flatt and Ken Kennedy. Performance of parallel processors. *Parallel Computing*, 12:1–20, 1989.

[12] G. C. Fox, M. Johnson, G. Lyzenga, S. W. Otto, J. Salmon, and D. Walker. *Solving Problems on Concurrent Processors: Volume 1.* Prentice-Hall, Englewood Cliffs, NJ, 1988.

[13] Ananth Grama, Anshul Gupta, and Vipin Kumar. Isoefficiency: Measuring the scalability of parallel algorithms and architectures. *IEEE Parallel and Distributed Technology*, 1(3):12–21, August, 1993. Also available as Technical Report TR 93-24, Department of Computer Science, University of Minnesota, Minneapolis, MN.

[14] Ananth Grama, Vipin Kumar, and V. Nageshwara Rao. Experimental evaluation of load balancing techniques for the hypercube. In *Proceedings of the Parallel Computing '91 Conference*, pages 497–514, 1991.

[15] Anshul Gupta and Vipin Kumar. The scalability of matrix multiplication algorithms on parallel computers. Technical Report TR 91-54, Department of Computer Science, University of Minnesota, Minneapolis, MN, 1991. A short version appears in *Proceedings of 1993 International Conference on Parallel Processing*, pages III-115–III-119, 1993.

[16] Anshul Gupta and Vipin Kumar. Performance properties of large scale parallel systems. *Journal of Parallel and Distributed Computing*, 19:234–244, 1993. Also available as Technical Report TR 92-32, Department of Computer Science, University of Minnesota, Minneapolis, MN.

[17] Anshul Gupta and Vipin Kumar. A scalable parallel algorithm for sparse matrix factorization. Technical Report 94-19, Department of Computer Science, University of Minnesota, Minneapolis, MN, 1994. A short version appears in *Supercomputing '94 Proceedings.* TR available in *users/kumar* at anonymous FTP site *ftp.cs.umn.edu.*

[18] Anshul Gupta and Vipin Kumar. The scalability of FFT on parallel computers. *IEEE Transactions on Parallel and Distributed Systems*, 4(8):922–932, August 1993. A detailed version available as Technical Report TR 90-53, Department of Computer Science, University of Minnesota, Minneapolis, MN.

[19] Anshul Gupta, Vipin Kumar, and A. H. Sameh. Performance and scalability of preconditioned conjugate gradient methods on parallel computers. *IEEE Transactions on Parallel and Distributed Systems*, 1995 (To Appear). Also available as Technical Report TR 92-64, Department of Computer Science, University of Minnesota, Minneapolis, MN. A short version appears in *Proceedings of the Sixth SIAM Conference on Parallel Processing for Scientific Computing*, pages 664–674, 1993.

[20] John L. Gustafson. Reevaluating Amdahl's law. *Communications of the ACM*, 31(5):532–533, 1988.

[21] John L. Gustafson. The consequences of fixed time performance measurement. In *Proceedings of the 25th Hawaii International Conference on System Sciences: Volume III*, pages 113–124, 1992.

[22] John L. Gustafson, Gary R. Montry, and Robert E. Benner. Development of parallel methods for a 1024-processor hypercube. *SIAM Journal on Scientific and Statistical Computing*, 9(4):609–638, 1988.

[23] Mark D. Hill. What is scalability? *Computer Architecture News*, 18(4), 1990.

[24] Kai Hwang. *Advanced Computer Architecture: Parallelism, Scalability, Programmability.* McGraw-Hill, New York, NY, 1993.

[25] Jing-Fu Jenq and Sartaj Sahni. All pairs shortest paths on a hypercube multiprocessor. In *Proceedings of the 1987 International Conference on Parallel Processing*, pages 713–716, 1987.

[26] S. L. Johnsson and C.-T. Ho. Optimum broadcasting and personalized communication in hypercubes. *IEEE Transactions on Computers*, 38(9):1249–1268, September 1989.

[27] Alan H. Karp and Horace P. Flatt. Measuring parallel processor performance. *Communications of the ACM*, 33(5):539–543, 1990.

[28] Clyde P. Kruskal, Larry Rudolph, and Marc Snir. A complexity theory of efficient parallel algorithms. Technical Report RC13572, IBM T. J. Watson Research Center, Yorktown Heights, NY, 1988.

[29] Vipin Kumar, Ananth Grama, Anshul Gupta, and George Karypis. *Introduction to Parallel Computing: Design and Analysis of Algorithms.* Benjamin/Cummings, Redwood City, CA, 1994.

[30] Vipin Kumar, Ananth Grama, and V. Nageshwara Rao. Scalable load balancing techniques for parallel computers. Technical Report 91-55, Computer Science Department, University of Minnesota, 1991. To appear in *Journal of Distributed and Parallel Computing*, 1994.

[31] Vipin Kumar and V. N. Rao. Parallel depth-first search, part II: Analysis. *International Journal of Parallel Programming*, 16(6):501–519, 1987.

[32] Vipin Kumar and V. N. Rao. Load balancing on the hypercube architecture. In *Proceedings of the Fourth Conference on Hypercubes, Concurrent Computers, and Applications*, pages 603–608, 1989.

[33] Vipin Kumar and V. N. Rao. Scalable parallel formulations of depth-first search. In Vipin Kumar, P. S. Gopalakrishnan, and Laveen N. Kanal, editors, *Parallel Algorithms for Machine Intelligence and Vision.* Springer-Verlag, New York, NY, 1990.

[34] Vipin Kumar and Vineet Singh. Scalability of parallel algorithms for the all-pairs shortest path problem. *Journal of Parallel and Distributed Computing*, 13(2):124–138, October 1991. A short version appears in the *Proceedings of the International Conference on Parallel Processing*, 1990.

[35] H. T. Kung. Memory requirements for balanced computer architectures. In *Proceedings of the 1986 IEEE Symposium on Computer Architecture*, pages 49–54, 1986.

[36] J. Lee, E. Shragowitz, and S. Sahni. A hypercube algorithm for the 0/1 knapsack problem. In *Proceedings of 1987 International Conference on Parallel Processing*, pages 699–706, 1987.

[37] Michael R. Leuze, Lawrence W. Dowdy, and Kee Hyun Park. Multiprogramming a distributed-memory multiprocessor. *Concurrency: Practice and Experience*, 1(1):19–33, September 1989.

[38] Y. W. E. Ma and Denis G. Shea. Downward scalability of parallel architectures. In *Proceedings of the 1988 International Conference on Supercomputing*, pages 109–120, 1988.

[39] Dan C. Marinescu and John R. Rice. On high level characterization of parallelism. Technical Report CSD-TR-1011, CAPO Report CER-90-32, Computer Science Department, Purdue University, West Lafayette, IN, Revised June 1991. To appear in *Journal of Parallel and Distributed Computing*, 1993.

[40] Paul Messina. Emerging supercomputer architectures. Technical Report C3P 746, Concurrent Computation Program, California Institute of Technology, Pasadena, CA, 1987.

[41] Cleve Moler. Another look at Amdahl's law. Technical Report TN-02-0587-0288, Intel Scientific Computers, 1987.

[42] David M. Nicol and Frank H. Willard. Problem size, parallel architecture, and optimal speedup. *Journal of Parallel and Distributed Computing*, 5:404–420, 1988.

[43] Daniel Nussbaum and Anant Agarwal. Scalability of parallel machines. *Communications of the ACM*, 34(3):57–61, 1991.

[44] Kee Hyun Park and Lawrence W. Dowdy. Dynamic partitioning of multiprocessor systems. *International Journal of Parallel Processing*, 18(2):91–120, 1989.

[45] Michael J. Quinn and Year Back Yoo. Data structures for the efficient solution of graph theoretic problems on tightly-coupled MIMD computers. In *Proceedings of the 1984 International Conference on Parallel Processing*, pages 431–438, 1984.

[46] S. Ranka and S. Sahni. *Hypercube Algorithms for Image Processing and Pattern Recognition.* Springer-Verlag, New York, NY, 1990.

[47] V. N. Rao and Vipin Kumar. Parallel depth-first search, part I: Implementation. *International Journal of Parallel Programming*, 16(6):479–499, 1987.

[48] Vineet Singh, Vipin Kumar, Gul Agha, and Chris Tomlinson. Scalability of parallel sorting on mesh multicomputers. *International Journal of Parallel Programming*, 20(2), 1991.

[49] Xian-He Sun and John L. Gustafson. Toward a better parallel performance metric. *Parallel Computing*, 17:1093–1109, December 1991. Also available as Technical Report IS-5053, UC-32, Ames Laboratory, Iowa State University, Ames, IA.

[50] Xian-He Sun and L. M. Ni. Another view of parallel speedup. In *Supercomputing '90 Proceedings*, pages 324–333, 1990.

[51] Xian-He Sun and Diane Thiede Rover. Scalability of parallel algorithm-machine combinations. Technical Report IS-5057, Ames Laboratory, Iowa State University, Ames, IA, 1991. To appear in *IEEE Transactions on Parallel and Distributed Systems.*

[52] Zhimin Tang and Guo-Jie Li. Optimal granularity of grid iteration problems. In *Proceedings of the 1990 International Conference on Parallel Processing*, pages I111–I118, 1990.

[53] Fredric A. Van-Catledge. Towards a general model for evaluating the relative performance of computer systems. *International Journal of Supercomputer Applications*, 3(2):100–108, 1989.

[54] Jeffrey Scott Vitter and Roger A. Simons. New classes for parallel complexity: A study of unification and other complete problems for P. *IEEE Transactions on Computers*, May 1986.

[55] Jinwoon Woo and Sartaj Sahni. Hypercube computing: Connected components. *Journal of Supercomputing*, 1991. Also available as TR 88-50 from the Department of Computer Science, University of Minnesota, Minneapolis, MN.

[56] Jinwoon Woo and Sartaj Sahni. Computing biconnected components on a hypercube. *Journal of Supercomputing*, June 1991. Also available as Technical Report TR 89-7 from the Department of Computer Science, University of Minnesota, Minneapolis, MN.

[57] Patrick H. Worley. *Information Requirements and the Implications for Parallel Computation.* PhD thesis, Stanford University, Department of Computer Science, Palo Alto, CA, 1988.

[58] Patrick H. Worley. The effect of time constraints on scaled speedup. *SIAM Journal on Scientific and Statistical Computing*, 11(5):838–858, 1990.

[59] Patrick H. Worley. Limits on parallelism in the numerical solution of linear PDEs. *SIAM Journal on Scientific and Statistical Computing*, 12:1–35, January 1991.

[60] Xiaofeng Zhou. Bridging the gap between Amdahl's law and Sandia laboratory's result. *Communications of the ACM*, 32(8):1014–5, 1989.

[61] J. R. Zorbas, D. J. Reble, and R. E. VanKooten. Measuring the scalability of parallel computer systems. In *Supercomputing '89 Proceedings*, pages 832–841, 1989.