To appear in Journal of Parallel and Distributed Computing, October 1993.
A preliminary version of this paper appears in the Proceedings of the 26th Hawaii International Conference on System Sciences, 1993.

Performance Properties of Large Scale Parallel Systems*

Anshul Gupta and Vipin Kumar

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

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

TR 92-32, September 1992 (Revised June 1993)

Abstract

There are several metrics that characterize the performance of a parallel system, such as, parallel execution time, speedup and efficiency. A number of properties of these metrics have been studied. For example, it is a well known fact that given a parallel architecture and a problem of a fixed size, the speedup of a parallel algorithm does not continue to increase with increasing number of processors. It usually tends to saturate or peak at a certain limit. Thus it may not be useful to employ more than an optimal number of processors for solving a problem on a parallel computer. This optimal number of processors depends on the problem size, the parallel algorithm and the parallel architecture. In this paper we study the impact of parallel processing overheads and the degree of concurrency of a parallel algorithm on the optimal number of processors to be used when the criterion for optimality is minimizing the parallel execution time. We then study a more general criterion of optimality and show how operating at the optimal point is equivalent to operating at a unique value of efficiency which is characteristic of the criterion of optimality and the properties of the parallel system under study. We put the technical results derived in this paper in perspective with similar results that have appeared in the literature before and show how this paper generalizes and/or extends these earlier results.

*This work was supported by IST/SDIO through the Army Research Office grant # 28408-MA-SDI to the University of Minnesota and by the University of Minnesota Army High Performance Computing Research Center under contract # DAAL03-89-C-0038.

1 Introduction

Massively parallel computers employing hundreds to thousands of processors are commercially available today and offer substantially higher raw computing power than the fastest sequential supercomputers. Availability of such systems has fueled interest in investigating the performance of parallel computers containing a large number of processors [23, 8, 7, 27, 37, 30, 6, 33, 18, 32, 17, 38, 34, 4, 5, 29].

The performance of a parallel algorithm cannot be studied in isolation from the parallel architecture it is implemented on. For the purpose of performance evaluation we define a **parallel system** as a combination of a parallel algorithm and a parallel architecture on which it is implemented. There are several metrics that characterize the performance of a parallel system, such as, parallel execution time, speedup and efficiency. A number of properties of these metrics have been studied. 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 but 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 for a fixed size problem is bounded by $\frac{1}{s}$, no matter how many processors are used. Gustafson, Montry and Benner [18, 16] experimentally demonstrated that the upper bound on speedup can be overcome by increasing the problem size as the number of processors is increased. Worley [37] showed that for a class of parallel algorithms, if the parallel execution time is fixed, then there exists a problem size which cannot be solved in that fixed time no matter how many processors are used. Flatt and Kennedy [8, 7] derived some important upper bounds related to the performance of parallel computers in the presence of synchronization and communication overheads. They show that if the parallel processing overhead for a certain computation satisfies certain properties, then there exists a unique value p_0 of the number of processors for which the parallel execution time is minimum (or the speedup is maximum) for a given problem size. However, at this point, the efficiency of the parallel execution is rather poor. Hence they suggest that the number of processors should be chosen to maximize the product of efficiency and speedup. Flatt and Kennedy, and Tang and Li [33] also suggest maximizing a weighted geometric mean of efficiency and speedup. Eager *et. al.* [6] proposed that an optimal operating point should be chosen such the efficiency of execution is roughly 0.5.

Many of the results presented in this paper are extensions, and in some cases, generalizations of the results of the above mentioned authors. Each parallel system has a unique overhead function, the value of which depends on the size of the problem being attempted and the number of processors being employed. Moreover, each parallel algorithm has an inherent degree of concurrency that determines the maximum number of processors that can be simultaneously kept busy at any given time while solving the problem of a given size. In this paper we study the effects of the overhead function and the degree of concurrency on performance measures such as speedup, execution time and efficiency and determine the optimal number of processors to be used under various optimality criteria.

We show that if the overhead function of a parallel system does not grow faster than $\Theta(p)$, where p is the number of processors in the parallel ensemble, then speedup can be maximized by using as many processors as permitted by the degree of the concurrency of the algorithm. If the overheads grow faster than $\Theta(p)$, then the number of processors that should be used to maximize speedup is determined either by the degree of concurrency or by the overhead function. We derive the exact expressions for maximum speedup, minimum execution time, the number of processors that yields maximum speedup, and the efficiency at the point of maximum speedup. We also show that for a class of overhead functions, given any problem size, operating at the point of maximum speedup is equivalent to operating at a fixed efficiency and the relation between the problem size and the number of processors that yields maximum speedup is given by the isoefficiency metric of scalability [22, 10, 23]. Next, a criterion of optimality is described that is more general than just maximizing the speedup and similar results are derived under this new condition for choosing the optimal operating point.

The organization of the paper is as follows. In Section 2 we define the terms to be used later in the paper. In Sections 3 and 4 we derive the technical results. In Section 5, we put these results in perspective with similar results that have appeared earlier in the literature and demonstrate that many of the results derived here are generalizations and/or extensions of the earlier results. Throughout the paper, examples are used to illustrate these results in the context of parallel algorithms for practical problems such as FFT, Matrix Multiplication, Shortest Paths, etc. Although, for the sake of ease of presentation, the examples are restricted to simple and regular problems, the properties of parallel systems studied here apply to general parallel systems as well.

A preliminary version of this paper appears in [13].

2 Definitions and Assumptions

In this section, we formally describe the terminology used in the rest of the paper.

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.

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 $C(W)$**:** 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 can not use more than $C(W)$ processors. $C(W)$ 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 [9], $W = N^3$ and $C(W) = N^2 = W^{2/3}$. It is easily seen that if the processor-time product [1] is $\Theta(W)$ (*i.e.*, the algorithm is **cost-optimal**), then $C(W) \leq \Theta(W)$.

Maximum Number of Processors Usable, p_{max}: The number of processors that yield maximum speedup S^{max} for a given W. This is the maximum number of processors one would like to use because using more processors will not increase the speedup.

3 Minimizing the Parallel Execution Time

In this section we relate the behavior of the T_P verses p curve to the nature of the overhead function T_o. As the number of processors is increased, T_P either asymptotically approaches a minimum value, or attains a minimum and starts rising again. We identify the overhead functions which lead to one case or the other. We show that in either case, the problem can be solved in minimum time by using a certain number of processors which we call p_{max}. Using more processors than p_{max} will either have no effect or will degrade the performance of the parallel system in terms of parallel execution time.

Most problems have a serial component W_s — the part of W that has to be executed sequentially. In this paper the sequential component of an algorithm is not considered as a separate entity, as it can be subsumed in T_o. While one processor is working on the sequential component, the remaining $p-1$ are ideal and contribute $(p-1)W_s$ to T_o. Thus for any parallel algorithm with a nonzero W_s, the analysis can be performed by assuming that T_o includes a term equal to $(p-1)W_s$. Under this assumption, the parallel execution time T_P for a problem of size W on p processors is given by the following relation:

$$T_P = \frac{W + T_o(W,p)}{p} \tag{1}$$

We now study the behavior of T_P under two different conditions.

3.1 Case I: $T_o \le \Theta(p)$

From Equation (1) it is clear that if $T_o(W,p)$ grows slower than $\Theta(p)$, then the overall power of p in the R.H.S. of Equation (1) is negative. In this case it would appear that if p is increased, then T_P will continue to decrease indefinitely. If $T_o(W,p)$ grows as fast as $\Theta(p)$ then there will be a lower bound on T_P, but that will be a constant independent of W. But we know that for any parallel system, the maximum number of processors that can be used for a given W is limited by $C(W)$. So the maximum speedup is bounded by $\frac{WC(W)}{W+T_o(W,C(W))}$ for a problem of size W and the efficiency at this point of peak performance is given by $\frac{W}{W+T_o(W,C(W))}$. Figure 1 illustrates the curve of T_P for the case when $T_o \le \Theta(p)$.

There are many important natural parallel systems for which the overhead function does not grow faster than $\Theta(p)$. One such system is described in Example 1 below. Such systems typically arise while using shared memory or SIMD machines which do not have a message startup time for data communication.

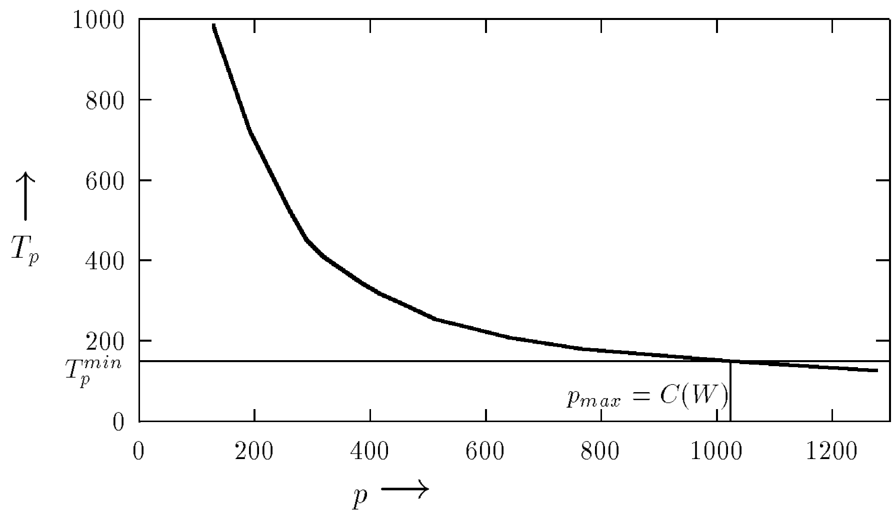

Figure 1: *A typical T_P verses p curve for $T_o \leq \Theta(p)$.*

Example 1: Parallel FFT on a SIMD Hypercube

Consider a parallel implementation of the FFT algorithm [14] on a SIMD hypercube connected machine (*e.g.*, the CM-2 [20]). If an N point FFT is being attempted on such a machine with p processors, $\frac{N}{p}$ units of data will be communicated among directly connected processors in $\log p$ of the $\log N$ iterations of the algorithm. For this parallel system $W = N \log N$. As shown in [14], $T_o = t_w \times \frac{N}{p} \log p \times p = t_w N \log p$, where t_w is the message communication time per word. Clearly, for a given W, $T_o < \Theta(p)$. Since $C(W)$ for the parallel FFT algorithm is N, there is a lower bound on parallel execution time which is given by $(1 + t_w) \log N$. Thus, p_{max} for an N point FFT on a SIMD hypercube is N and the problem cannot be solved in less than $\Theta(\log N)$ time.

3.2 Case II: $T_o > \Theta(p)$

When $T_o(W, p)$ grows faster than $\Theta(p)$, a glance at Equation (1) will reveal that the term $\frac{W}{p}$ will keep decreasing with increasing p, while the term $\frac{T_o}{p}$ will increase. Therefore, the overall T_P will first decrease and then increase with increasing p, resulting in a distinct minimum. Now we derive the relationship between W and p such that T_P is minimized. Let p_0 be the value of p for which the mathematical expression on the R.H.S of Equation (1) for T_P attains its minimum value.

At $p = p_0$, T_P is minimum and therefore $\frac{d}{dp} T_P = 0$.

$$\Rightarrow \frac{d}{dp}(\frac{W + T_o(W,p)}{p}) = 0$$

$$\Rightarrow \frac{-W}{p^2} - \frac{T_o(W,p)}{p^2} + \frac{\frac{d}{dp}T_o(W,p)}{p} = 0$$

$$\Rightarrow \frac{d}{dp}T_o(W,p) = \frac{W}{p} + \frac{T_o(W,p)}{p}$$

$$\Rightarrow \frac{d}{dp}T_o(W,p) = T_P \tag{2}$$

For a given W, we can solve the above equation to find p_0. A rather general form of the overhead is one in which the overhead function is a sum of terms where each term is a product of a function of W and a function of p. In most real life parallel systems, these functions of W and p are such that T_o can be written as $\Sigma_{i=1}^{i=n} c_i W^{y_i} (\log W)^{u_i} p^{x_i} (\log p)^{z_i}$, where c_i's are constants and $x_i \geq 0$ and $y_i \geq 0$ for $1 \leq i \leq n$, and u_i's and z_i's are 0's or 1's. The overhead functions of all architecture-algorithm combinations that we have come across fit this form [24, 25, 31, 14, 12, 15, 36, 35, 11]. As illustrated by a variety of examples in this paper, these include important algorithms such as Matrix Multiplication, FFT, Parallel Search, finding Shortest Paths in a graph, etc., on almost all parallel architectures of interest.

For the sake of simplicity of the following analysis, we assume $z_i = 0$ and $u_i = 0$ for all i's. Analysis similar to that presented below can be performed even without this assumption and similar results can be obtained (Appendix A). Substituting $\Sigma_{i=1}^{i=n} c_i W^{y_i} p^{x_i}$ for $T_o(W,p)$ in Equation (2), we obtain the following equation:

$$\Sigma_{i=1}^{i=n} c_i x_i W^{y_i} p^{x_i - 1} = \frac{W + \Sigma_{i=1}^{i=n} c_i W^{y_i} p^{x_i}}{p}$$

$$\Rightarrow W = \Sigma_{i=1}^{i=n} c_i (x_i - 1) W^{y_i} p^{x_i} \tag{3}$$

For the overhead function described above, Equation (3) determines the relationship between W and p for minimizing T_P provided that T_o grows faster than $\Theta(p)$. Because of the nature of Equation (3), it may not always be possible to express p as a function of W in a closed form. So we solve Equation (3), considering one R.H.S. term at a time and ignoring the rest. If the ith term is being considered, the relation $W = c_i(x_i - 1)W^{y_i} p^{x_i}$ yields $p = (\frac{W^{1-y_i}}{c_i(x_i-1)})^{\frac{1}{x_i}} = \Theta(W^{\frac{1-y_i}{x_i}})$. It can be shown (Appendix B) that among all the i solutions for p obtained in this manner, the speedup is maximum for any given W when $p = \Theta(W^{\frac{1-y_j}{x_j}})$ where $\frac{1-y_j}{x_j} \leq \frac{1-y_i}{x_i}$ for all i $(1 \leq i \leq n)$. We call the jth term of T_o the **dominant term** if the value of $\frac{1-y_j}{x_j}$ is the least among all values $\frac{1-y_i}{x_i}$ $(1 \leq i \leq n)$ because this is the term that determines the order of p_0 -

the solution to Equation (2) for large values of W and p. If jth term is the dominant term of T_o, then solving Equation (3) with respect to the jth term on the R.H.S. yields the following approximate expression for p_0 for large values of W:

$$p_0 \approx \left(\frac{W^{1-y_j}}{c_j(x_j-1)}\right)^{\frac{1}{x_j}} \tag{4}$$

The value of p_0 thus obtained can be used in the expression for T_P to determine the minimum parallel execution time for a given W. The value of p_0, when plugged in the expression for efficiency, yields the following:

$$E_0 = \frac{W}{W+T_o}$$

$$\Rightarrow \; E_0 \approx \frac{W}{W + c_j W^{y_j}\left(\frac{W^{\frac{1-y_j}{x_j}}}{(c_j x_j - c_j)^{\frac{1}{x_j}}}\right)^{x_j}}$$

$$\Rightarrow \; E_0 \approx 1 - \frac{1}{x_j} \tag{5}$$

Note that the above analysis holds *only* if x_j, the exponent of p in the dominant term of T_o is greater than 1. If $x_j \leq 1$, then the asymptotically highest term in T_o (*i.e.*, $c_j W^{y_j} p^{x_j}$) is less than or equal to $\Theta(p)$ and the results for the case when $T_o \leq \Theta(p)$ apply.

Equations (4) and (5) yield the mathematical values of p_0 and E_0 respectively. But the derived value of p_0 may exceed $C(W)$. So in practice, at the point of peak performance (in terms of maximum speedup or minimum execution time), the number of processors p_{max} is given by $min(p_0, C(W))$ for a given W. Thus it is possible that $C(W)$ of a parallel algorithm may determine the minimum execution time rather than the mathematically derived conditions. The following example illustrates this case:

Example 2: Floyd's Striped Shortest Path Algorithm on Mesh

A number of parallel Shortest Path algorithms are discussed in [25]. Consider the implementation of Floyd's algorithm in which the $N \times N$ adjacency matrix of the graph is striped among p processors such that each processor stores $\frac{N}{p}$ full rows of the matrix. The problem size W here is given by N^3 for finding all to all shortest paths on an N-node graph. In each of the N iterations of this algorithm, a processor broadcasts a row of length N of the adjacency matrix of the graph to every other processor. As shown in [25], if the p processor are connected in a mesh configuration with cut-through routing, the total overhead due to this communication is given by $T_o = t_s N p^{1.5} + t_w(N + \sqrt{p})Np$. Here t_s and t_w are constants related to message startup time and the speed of message transfer respectively. Since t_w is often very small compared to t_s, $T_o = (t_s + t_w)Np^{1.5} + t_w N^2 p \approx t_s N p^{1.5} + t_w N^2 p = t_s W^{1/3} p^{1.5} + t_w W^{2/3} p$. From Equation (3), p_0 is equal to $(\frac{W^{2/3}}{.5t_s})^{2/3} \approx \frac{1.59N^{4/3}}{t_s^{2/3}}$. But since at most N processors can be used in this algorithm,

$p_{max} = min(C(W), p_0) = N$. The minimum execution time for this parallel system is therefore $N^2 + t_s N^{1.5} + t_w N^2$ for $p_{max} = N$.

If working on a 100 node graph, then the speedup will peak at $p = N = 100$ and for $t_s = 1$ and $t_w = 0.1$, the speedup will be 83.33 resulting in an efficiency of 0.83 at the point of peak performance.

It is also possible for two parallel systems to have the same T_o (and hence the same p_0) but different $C(W)$s. In such cases, an analysis of the overhead function might mislead one into believing that the two parallel systems are equivalent in terms of maximum speedup and minimum execution time. Example 3 below illustrates the case when the speedup peaks at $p = p_o$. The algorithm in Example 2 has exactly the same T_o and hence the same p_o, but the speedup peaks at $p = C(W)$ because $C(W) < p_o$. Thus the two parallel systems described in Examples 2 and 3 are grossly different in terms of their peak performances, although their overhead functions are the same.

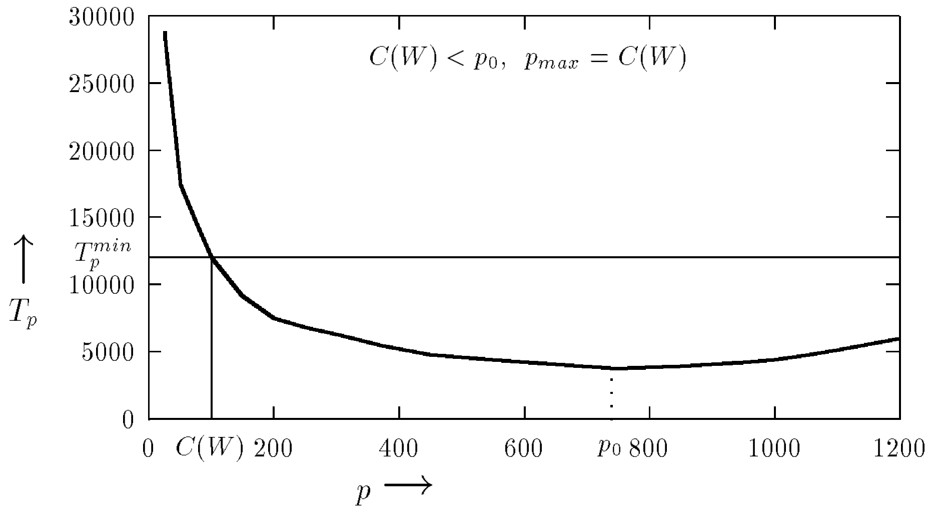

Figure 2: T_P *verses* p *curve for* $T_o > \Theta(p)$ *showing* T_P^{min} *when* $C(W) < p_0$.

Example 3: Floyd's Checkerboard Shortest Path Algorithm on Mesh

In this example, we consider a different parallel system consisting of another variation of Floyd's algorithm discussed in [25] and a wrap-around mesh with store-and-forward routing. In this algorithm, the $N \times N$ adjacency matrix is partitioned into p sub-blocks of size $\frac{N}{\sqrt{p}} \times \frac{N}{\sqrt{p}}$ each, and these sub-blocks are mapped on a p processor mesh. In this version of Floyd's algorithm, a processor broadcasts $\frac{N}{\sqrt{p}}$ elements among $\sqrt{p}$ processors in each of the N iterations. As shown

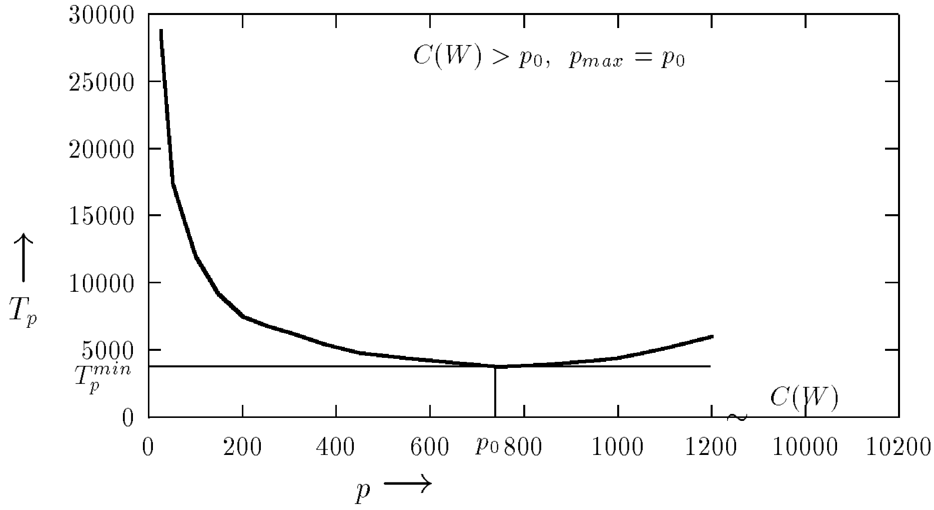

Figure 3: T_P *verses* p *curve for* $T_o > \Theta(p)$ *showing* T_P^{min} *when* $C(W) > p_0$.

in [25], this results in a total overhead of $T_o = t_s N p^{1.5} + t_w N^2 p$. Since the expression for T_o is same as that in Example 2, $p_0 = \frac{1.59 N^{4/3}}{t_s^{2/3}}$ again. But $C(W)$ for the checkerboard version of the algorithm is $W^{2/3} = N^2$. Therefore $p_{max} = p_0$ in this case as $p_0 < C(W)$.

For $t_s = 1$ and $t_w = 0.1$, Equation (3) yields a value of $p_o = 738$ for a 100 node graph. The speedup peaks with 738 processors at a value of 246, but the efficiency at this peak speedup is only 0.33.

Figures 2 and 3 graphically depict T_P as a function of p corresponding to Examples 2 and 3 respectively.

3.3 Minimizing T_P and the Isoefficiency Function

In this section we show that for a wide class of overhead functions, studying a parallel system at its peak performance in terms of the speedup is equivalent to studying its behavior at a fixed efficiency. The isoefficiency metric [22, 10, 23] comes in as a handy tool to study the fixed efficiency characteristics of a parallel system. The isoefficiency function relates the problem size to the number of processors necessary for an increase in speedup in proportion to the number of processors used. If a parallel system incurs a total overhead of $T_o(W,p)$ while solving a problem of size W on p processors, the efficiency of the system is given by $E = \frac{1}{1+\frac{T_o(W,p)}{W}}$. In order to maintain a constant efficiency, $W \propto T_o(W,p)$ or $W = KT_o(W,p)$ must be satisfied, where $K = \frac{E}{1-E}$ is a constant depending on the efficiency to be maintained. This is the central relation that is used to determine isoefficiency as a function of p. From this equation, the problem size

W can usually be obtained as a function of p by algebraic manipulations. If the problem size W needs to grow as fast as $f_E(p)$ to maintain an efficiency E, then $f_E(p)$ is defined to be the isoefficiency function of the parallel algorithm-architecture combination for efficiency E.

We now show that unless $p_{max} = C(W)$ for a parallel system, a unique efficiency is attained at the point of peak performance. This value of E depends only on the characteristics of the parallel system (*i.e.*, the type of overhead function for the algorithm-architecture combination) and is independent of W or T_P. For the type of overhead function assumed in Section 3.2, the following relation determines the isoefficiency function for an efficiency E.

$$W = \frac{E}{1-E}\Sigma_{i=1}^{i=n} c_i W^{y_i} p^{x_i} \tag{6}$$

Clearly, this equation has the same form as Equation (3), but has different constants. The dominant term on the R.H.S will yield the relationship between W and p in a closed form in both the equations. If this is the jth term, then both the equations will become equivalent asymptotically if their jth terms are same. This amounts to operating at an efficiency that is given by the following relation obtained by equating the coefficients of the jth terms of Equations (3) and (6).

$$\frac{E}{1-E}c_j = c_j(x_j - 1)$$

$$\Rightarrow \; E = 1 - \frac{1}{x_j}$$

The above equation is in conformation with Equation (5). Once we know that working at the point of peak performance amounts to working at an efficiency of $1-\frac{1}{x_j}$, then, for a given W, we can find the number of processors at which the performance will peak by using the relation $1-\frac{1}{x_j} = \frac{W}{W+T_o(W,p)}$.

As discussed in [22, 10], the relation between the problem size and the maximum number of processors that can be used in a cost-optimal fashion for solving the problem is given by the isoefficiency function. Often, using as many processors as possible results in a non-cost-optimal system. For example, adding n numbers on an n-processor hypercube takes $\Theta(\log n)$ time, which is the minimum execution time for this problem. This is not a cost optimal parallel system because $W = \Theta(N) < pT_P = \Theta(n \log n)$. An important corollary of the result presented in this section is that for the parallel systems for which the relationship between the problem size and the number of processors for maximum speedup (minimum execution time) is given by the isoefficiency function, the asymptotic minimum execution time can be attained in a cost-optimal fashion. For instance, if $\Theta(\frac{n}{\log n})$ processors are used to add n numbers on a hypercube, the parallel system will be cost-optimal and the parallel execution time will still be $\Theta(\log n)$.

Note that the correspondence between the isoefficiency function and the relation between W and p for operating at minimum T_P will fail if the x_j in the dominant term is less than or equal to 1. In this case, a term other than the one that determines the isoefficiency function will determine the condition for minimum T_P.

3.4 Summary of Results

At this point we state the important results of this section.

- *For parallel algorithms with* $T_o \leq \Theta(p)$*, the maximum speedup is obtained at* $p = C(W)$ *and for algorithms with* $T_o > \Theta(p)$*, the maximum speedup is obtained at* $p = min(p_o, C(W))$*, where* p_0 *for a given* W *is determined by solving Equation (3).*

- *For the parallel algorithms with* T_o *of the form described in Section 3.2, if the* j*th term is the dominant term in the expression for* T_o *and* $x_j > 1$*, then the efficiency at the point of maximum speedup always remains the same irrespective of the problem size, and is given by* $E = 1 - \frac{1}{x_j}$*.*

- *For the parallel algorithms satisfying the above conditions, the relationship between the problem size and the number of processors at which the speedup is maximum for that problem size, is given by the isoefficiency function for* $E = 1 - \frac{1}{x_j}$*, unless* $p_{max} = C(W)$*.*

4 Minimizing $p(T_P)^r$

From the previous sections, it is clear that operating at a point where T_P is minimum might not be a good idea because for some parallel systems the efficiency at this point might be low. On the other hand, the maximum efficiency is always attained at $p = 1$ which obviously is the point of minimum speedup. Therefore, in order to achieve a balance between speedup and efficiency, several researchers have proposed to operate at a point where the value of $p(T_P)^r$ is minimized for some constant r ($r \geq 1$) and for a given problem size W [8, 6, 33]. It can be shown [33] that this corresponds to the point where ES^{r-1} is maximized for a given problem size.

$$p(T_P)^r = pT_P(\frac{W}{S})^{r-1} = \frac{W^r}{ES^{r-1}}$$

Thus $p(T_P)^r$ will be minimum when ES^{r-1} is maximum for a given W and by minimizing $p(T_P)^r$, we are choosing an operating point with a concern for both speedup and efficiency, their relative weights being determined by the value of r.

Now let us locate the point where $p(T_P)^r$ is minimum.

$$p(T_P)^r = p(\tfrac{T_e+T_o}{p})^r = p^{1-r}(T_e + T_o)^r$$

Again, as in the previous section, the following two cases arise:

4.1 Case I: $T_o \leq \Theta(p^{\frac{r-1}{r}})$

Since $p(T_P)^r = p^{1-r}(T_e + T_o)^r = (T_e p^{\frac{1-r}{r}} + T_o p^{\frac{1-r}{r}})^r$, if $T_o \leq \Theta(p^{\frac{r-1}{r}})$ then the overall power of p in the expression for $p(T_P)^r$ will become negative and hence its value will mathematically tend to some lower bound as $p \rightsquigarrow \infty$. Thus using as many processors as are feasible will lead to minimum $p(T_P)^r$. In other words, for this case, $p(T_P)^r$ is minimum when $p = C(W)$.

4.2 Case II: $T_o > \Theta(p^{\frac{r-1}{r}})$

If T_o grows faster than $\Theta(p^{\frac{r-1}{r}})$, then we proceed as follows. In order to minimize $p(T_P)^r$, $\frac{d}{dp}\, p(T_P)^r$ should be equal to zero.

$$(1-r)p^{-r}(T_e+T_o)^r + rp^{(1-r)}(T_e+T_o)^{(r-1)}\frac{d}{dp}T_o = 0$$

$$\Rightarrow \; \frac{d}{dp}T_o = \frac{r-1}{r}T_P \tag{7}$$

We choose the same type of overhead function as in Section 3.2. Substituting $\Sigma_{i=1}^{i=n} c_i W^{y_i} p^{x_i}$ for T_o in Equation (7), we get the following equation:

$$\Sigma_{i=1}^{i=n} c_i x_i W^{y_i} p^{x_i-1} = \frac{r-1}{rp}(W + \Sigma_{i=1}^{i=n} c_i W^{y_i} p^{x_i})$$

$$\Rightarrow \; W = \Sigma_{i=1}^{i=n} c_i(\frac{rx_i}{r-1} - 1)W^{y_i} p^{x_i} \tag{8}$$

Now even the number of processors for which $p(T_P)^r$ is minimum could exceed the value of p that is permitted by the degree of concurrency of the algorithm. In this case the minimum possible value for $p(T_P)^r$ will be obtained when $C(W)$ processors are used. The following example illustrates this case.

Example 4: Matrix Multiplication on Mesh

Consider a simple algorithm described in [12] for multiplying two $N \times N$ matrices on a $\sqrt{p} \times \sqrt{p}$ wrap-around mesh. As the first step of the algorithm, each processor acquires all those elements of both the matrices that are required to generate the $\frac{N^2}{p}$ elements of the product matrix which are to reside in that processor. For this parallel system, $W = N^3$ and $T_o = t_s p\sqrt{p} + t_w N^2\sqrt{p}$. For determining the operating point where $p(T_P)^2$ is minimum, we substitute $n = 2$, $r = 2$, $c_1 = t_s$, $c_2 = t_w$, $x_1 = 1.5$, $x_2 = 0.5$, $y_1 = 0$ and $y_2 = \frac{2}{3}$ in Equation (8). This substitution yields the relation $W = 2t_s p^{1.5}$ for determining the required operating point. In other words, the number of processors p_0 at which $p(T_P)^2$ is minimum is given by $p_0 = (\frac{W}{2t_s})^{2/3} = \frac{N^2}{(2t_s)^{2/3}}$. But the maximum number of processors that this algorithm can use is only N^2. Therefore, for $t_s < .5$, $p_0 > C(W)$ and hence $C(W)$ processors should be used to minimize pT_P^2.

4.3 Minimizing $p(T_P)^r$ and the Isoefficiency Function

In this subsection we show that for a wide class of parallel systems, even minimizing $p(T_P)^r$ amounts to operating at a unique efficiency that depends only on the overhead function and the value of r. In other words, for a given W, $p(T_P)^r$ is minimum for some value of p and the relationship between W and this p for the parallel system is given by its isoefficiency function for a unique value of efficiency that depends only on r and the type of overhead function. Equation

(8), which gives the relationship between W and p for minimum $p(T_P)^r$, has the same form as Equation (6) that determines the isoefficiency function for some efficiency E. If the jth terms of the R.H.S.s of Equations (6) and (8) dominate (and $x_j > \frac{r-1}{r}$), then the efficiency at minimum $p(T_P)^r$ can be obtained by equating the corresponding constants; *i.e.*, $\frac{Ec_j}{1-E}$ and $c_j(\frac{rx_j}{r-1} - 1)$. This yields the following expression for the value of efficiency at the point where $p(T_P)^r$ is minimum:

$$E = 1 - \frac{r-1}{rx_j} \tag{9}$$

The following example illustrates how the analysis of Section 4 can be used for chosing an appropriate operating point (in terms of p) for a parallel algorithm to solve a problem instance of a given size. It also confirms the validity of Equation (9).

Example 5: FFT on a Hypercube

Consider the implementation of the FFT algorithm on an MIMD hypercube using the binary-exchange algorithm. As shown in [14], for an N point FFT on p processors, $W = N \log N$ and $T_o = t_s p \log p + t_w N \log p$ for this algorithm. Taking $t_s = 2$, $t_w = 0.1$ and rewriting the expression for T_o in the form described in Section 3.2, we get the following:

$$T_o \approx 2p \log p + 0.1 \frac{W}{\log W} \log p$$

Now suppose it is desired to minimize $p(T_P)^2$, which is equivalent to maximizing the ES product. Clearly, the first term of T_o dominates and hence putting $r = 2$ and $x_j = 1$ in Equation (9), an efficiency of 0.5 is predicted when $p(T_P)^2$ is minimized. An analysis similar to that in Section 4.2 will show that $p(T_P)^r$ will be minimum when $p \approx \frac{N}{2}$ is used.

If a 1024 point FFT is being attempted, then Table I shows that at $p = 512$ the ES product is indeed maximum and the efficiency at this point is indeed 0.5.

Again, just like in Section 3.3, there are exceptions to the correspondence between the isoefficiency function and the condition for minimum $p(T_P)^r$. If the jth term in Equation (6) determines the isoefficiency function and in Equation (8), $x_j < \frac{r-1}{r}$, then the coefficient of the jth term in Equation (8) will be zero or negative and some other term in Equation (8) will determine the relationship between W and p for minimum $p(T_P)^r$.

The following subsection summarizes the results of this section.

4.4 Summary of Results

- *For parallel algorithms with $T_o \leq \Theta(p^{\frac{r-1}{r}})$, the minimum value for the expression $p(T_P)^r$ is attained at $p = C(W)$ and for algorithms with $T_o > \Theta(p^{\frac{r-1}{r}})$, it is attained at $p = min(C(W), p_0)$, where p_0 for a given W is obtained by solving Equation (8).*

Table I

Performance of FFT on a hypercube with $N = 1024$, $t_s = 2$ and $t_w = 0.1$.

p	T_P	S	E	$E \times S$
128	99.6	103	.80	82.4
256	59.2	173	.68	117.6
384	46.1	222	.58	128.4
512	**39.8**	**257**	**.50**	**129.3**
640	36.1	284	.44	124.9
768	33.8	303	.39	116.2
896	32.2	318	.35	112.8
1024	31.0	330	.32	105.5

- *For the parallel algorithms with T_o of the form described in Section 3.2, if the jth term dominates in the expression for T_o and $x_j > \frac{r-1}{r}$, then the efficiency at the point of minimum $p(T_P)^r$ always remains same irrespective of the problem size and is given by $E = 1 - \frac{r-1}{rx_j}$.*
- *For the parallel algorithms satisfying the above conditions, the relationship between the problem size and the number of processors at which $p(T_P)^r$ is minimum for that problem size, is given by the isoefficiency function for $E = 1 - \frac{r-1}{rx_j}$, provided $C(W) > p_0$ determined from Equation (8).*

In fact the results pertaining to minimization of T_P are special cases of the above results when $r \rightsquigarrow \infty$, *i.e.*; the weight of p is zero with respect to T_P or the weight of E is zero with respect to S. Equation (3) can be derived from Equation (8) and Equation (5) from Equation (9) if $\frac{r-1}{r}$ is replaced by $lim_{r \rightsquigarrow \infty} \frac{r-1}{r} = 1$.

5 Significance in the Context of Related Research

In this section we discuss how this paper encapsulates several results which have appeared in the literature before and happen to be special cases of the more general results presented here.

Flatt and Kennedy [8, 7] show that if the overhead function satisfies certain mathematical properties, then there exists a unique value p_0 of the number of processors for which T_P is minimum for a given W. A property of T_o on which their analysis depends heavily is that $T_o > \Theta(p)$.[1] This assumption on the overhead function limits the range of the applicability of their analysis. As seen in Section 3.1 and Example 1, there exist parallel systems for which do

[1] T_o, as defined in [8], is the overhead incurred per processor when all costs are normalized with respect to $W = 1$. So in the light of the definition of T_o in this paper, the actual mathematical condition of [8], that T_o is an increasing nonnegative function of p, has been translated to the condition that T_o grows faster than $\Theta(p)$.

not obey this condition, and in such cases the point of peak performance is determined by the degree of concurrency of the algorithm being used.

Flatt and Kennedy show that the maximum speedup attainable for a given problem is upper-bounded by $\frac{1}{\frac{d}{dp}(pT_P)}$ at $p = p_0$. They also show that the better a parallel algorithm is (*i.e.*, the slower T_o grows with p), the higher is the value of p_0 and the lower is the value of efficiency obtained at this point. Equations (4) and (5) in this paper provide results similar to Flatt and Kennedy's. But the analysis in [8] tends to conclude the following - (i) if the overhead function grows very fast with respect to p, then p_0 is small, and hence parallel processing cannot provide substantial speedups; (ii) if the overhead function grows slowly (*i.e.*, closer to $\Theta(p)$), then the overall efficiency is very poor at $p = p_0$. Note that if we keep improving the overhead function, the mathematically derived value of p_0 will ultimately exceed the limit imposed by the degree of concurrency on the number of processors that can be used. Hence, in practice no more than $C(W)$ processors will be used. Thus, in this situation, the theoretical value of p_0 and the efficiency at this point does not serve a useful purpose because the point of peak performance efficiency cannot be worse than $\frac{W}{W+T_o(W,C(W))}$. For instance, Flatt and Kennedy's analysis will predict identical values of p_{max} and efficiency at this operating point for the parallel systems described in Examples 2 and 3 because their overhead functions are identical. But as we saw in these examples, this is not the case because the the value of $C(W)$ in the two cases is different.

In [27], 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 events $g(p)$ as a function of p. They consider the case where each event is of a fixed duration θ and hence $T_o = \theta g(p)$. Under these assumptions on T_o, they conclude that with increasing number of processors, the speedup saturates at some value if $T_o = \Theta(p)$, and it asymptotically approaches zero if $T_o = \Theta(p^m)$, where $m \geq 2$. The results of Sections 3.1 and 3.2 are generalizations of these conclusions for a wider class of overhead functions. In Section 3.1 we show that the speedup saturates at some maximum value if $T_o \leq \Theta(p)$, and in Section 3.2 we show that speedup will attain a maximum value and then it will drop monotonically with p if $T_o > \Theta(p)$.

Usually, the duration of an event or a communication step θ is not a constant as assumed in [27]. In general, both θ and T_o are functions of W and p. If T_o is of the form $\theta g(p)$, Marinescu and Rice [27] derive that the number of processors that will yield maximum speedup will be given by $p = (\frac{W}{\theta} + g(p))\frac{1}{g'(p)}$, which can be rewritten as $\theta g'(p) = \frac{W+\theta g(p)}{p}$. It is easily verified that this is a special case of Equation (2) for $T_o = \theta g(p)$.

Worley [37] showed that for certain algorithms, given a certain amount of time 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. In Section 3, we describe the exact nature of the overhead function for which a lower bound exists on the execution time for a given problem size. This is exactly the condition for which, given a fixed time, an upper bound will exist on the size of the problem that can be solved within this time. We show that for a class of parallel systems, the relation between problem size W and the number of processors p at which the parallel execution time T_P is minimized, is given by the isoefficiency function for a particular efficiency.

Several other researchers have used the minimum parallel execution time of a problem of a given size for analyzing the performance of parallel systems [28, 26, 30]. Nussbaum and Agarwal [30] define scalability of an 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 given problem size if an unlimited number of processors is available. For a fixed problem size, the scalability of the parallel system, according to their metric, depends directly on the minimum T_P for the system. For the class of parallel systems for which the correspondence between the isoefficiency function and the relation between W and p for minimizing T_P exists, Nussbaum and Agarwal's scalability metric will yield results identical to those predicted by the isoefficiency function on the behavior of these parallel systems.

Eager *et. al.* [6] and Tang and Li [33] have proposed a criterion of optimality different from optimal speedup. They argue that the optimal operating point should be chosen so that a balance is struck between efficiency and speedup. It is proposed in [6] that the "knee" of the execution time verses efficiency curve is a good choice of the operating point because at this point the incremental benefit of adding processors is roughly $\frac{1}{2}$ per processor, or, in other words, efficiency is 0.5. Eager *et. al.* and Tang and Li also conclude that for $T_o = \Theta(p)$, this is also equivalent to operating at a point where the ES product is maximum or $p(T_P)^2$ is minimum. This conclusion in [6, 33] is a special case of the more general case that is captured in Equation (9). If we substitute $x_j = 1$ in Equation (9) (which is the case if $T_o = \Theta(p)$), it can seen that we indeed get an efficiency of 0.5 for $r = 2$. In general, operating at the optimal point or the "knee" referred to in [6] and [33] for a parallel system with $T_o = \Theta(p^{x_j})$ will be identical to operating at a point where $p(T_P)^r$ is minimum, where $r = \frac{2}{2-x_j}$. This is obtained from Equation (9) for $E = 0.5$. Minimizing $p(T_P)^r$ for $r > \frac{2}{2-x_j}$ will result in an operating point with efficiency lower than 0.5 but a higher speedup. On the other hand, minimizing $p(T_P)^r$ for $r < \frac{2}{2-x_j}$ will result in an operating point with efficiency higher than 0.5 and a lower speedup.

In [21], Kleinrock and Huang state that the mean service time for a job is minimum for $p = \infty$, or for as many processors as possible. This is true only under the assumption that $T_o < \Theta(p)$. For this assumption to be true, the parallel system has to be devoid of any global operation (such as broadcast, and one-to-all and all-to-all personalized communication [3, 19]) with a message passing latency or message startup time. The reason is that such operations always lead to $T_o \geq \Theta(p)$. This class of algorithms includes some fairly important algorithms such as matrix multiplication (all-to-all/one-to-all broadcast) [12], vector dot products (single node accumulation) [15], shortest paths (one-to-all broadcast) [25], and FFT (all-to-all personalized communication) [14], 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 $T_o \geq \Theta(p)$.

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

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

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

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

[7] 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.

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

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] Anshul Gupta and Vipin Kumar. Analyzing performance of large scale parallel systems. In *Proceedings of the 26th Hawaii International Conference on System Sciences*, 1993. To appear in *Journal of Parallel and Distributed Computing*, 1993.

[14] 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.

[15] Anshul Gupta, Vipin Kumar, and A. H. Sameh. Performance and scalability of preconditioned conjugate gradient methods on parallel computers. Technical Report TR 92-64, Department of Computer Science, University of Minnesota, Minneapolis, MN, 1992. A short version appears in *Proceedings of the Sixth SIAM Conference on Parallel Processing for Scientific Computing*, pages 664–674, 1993.

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

[17] 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.

[18] 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.

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

[20] S. L. Johnsson, R. Krawitz, R. Frye, and D. McDonald. A radix-2 FFT on the connection machine. Technical report, Thinking Machines Corporation, Cambridge, MA, 1989.

[21] L. Kleinrock and J.-H. Huang. On parallel processing systems: Amdahl's law generalized and some results on optimal design. *IEEE Transactions on Software Engineering*, 18(5):434–447, May 1992.

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

[23] Vipin Kumar and Anshul Gupta. Analyzing scalability of parallel algorithms and architectures. Technical Report TR 91-18, Department of Computer Science Department, University of Minnesota, Minneapolis, MN, 1991. To appear in *Journal of Parallel and Distributed Computing*, 1994. A shorter version appears in *Proceedings of the 1991 International Conference on Supercomputing*, pages 396-405, 1991.

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

[25] 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.

[26] 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.

[27] 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.

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

[29] Sam H. Noh, Dipak Ghosal, and Ashok K. Agrawala. An empirical study of the effect of granularity on parallel algorithms on the connection machine. Technical Report UMIACS-TR-89-124.1, University of Maryland, College Park, MD, 1989.

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

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

[32] 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.

[33] 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.

[34] 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.

[35] 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.

[36] 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.

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

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

Appendix A

Let $T_o(W,p) = \Sigma_{i=1}^{i=n} c_i W^{y_i} (\log W)^{u_i} p^{x_i} (\log p)^{z_i}$, where c_i's are constants and $x_i \geq 0$ and $y_i \geq 0$ for $1 \leq i \leq n$, and u_i's and z_i's are 0's or 1's. Now let us compute $\frac{d}{dp} T_o(W,p)$.

$$T_o(W,p) = \Sigma_{i=1}^{i=n} c_i W^{y_i} (\log W)^{u_i} p^{x_i} (\log p)^{z_i}$$

$$\frac{d}{dp} T_o(W,p) = \Sigma_{i=1}^{i=n} c_i W^{y_i} (\log W)^{u_i} (x_i p^{x_i - 1} (\log p)^{z_i} + z_i p^{x_i - 1} (\log p)^{z_i - 1})$$

If all z_i's are either 0 or 1, then the above equations can be rewritten as:

$$\frac{d}{dp} T_o(W,p) = \Sigma_{i=1}^{i=n} c_i W^{y_i} (\log W)^{u_i} (x_i p^{x_i - 1} (x_i \log p)^{z_i} + z_i)$$

$$\frac{d}{dp} T_o(W,p) \approx \Sigma_{i=1}^{i=n} c_i x_i W^{y_i} (\log W)^{u_i} x_i p^{x_i - 1}$$

Equating $\frac{d}{dp} T_o(W,p)$ to T_P according to Equation 2,

$$\Sigma_{i=1}^{i=n} c_i x_i W^{y_i} (\log W)^{u_i} x_i p^{x_i - 1} = \frac{W + \Sigma_{i=1}^{i=n} c_i W^{y_i} (\log W)^{u_i} p^{x_i} (\log p)^{z_i}}{p}$$

$$W = \Sigma_{i=1}^{i=n} c_i (x_i - 1) W^{y_i} (\log W)^{u_i} p^{x_i} (\log p)^{z_i} \tag{10}$$

The above equation determines the relation between W and p for which the parallel execution time is minimized. The equation determining the isoefficiency function for the parallel system with the overhead function under consideration will be as follows (see discussion in Section 3.3):

$$W = \frac{E}{1-E} \Sigma_{i=1}^{i=n} c_i W^{y_i} (\log W)^{u_i} p^{x_i} (\log p)^{z_i} \tag{11}$$

Comparing Equations 10 and 11, if the jth term in T_o is the dominant term and $x_j > 1$, then the efficiency at the point of minimum parallel execution time will be given by $E_0 \approx 1 - \frac{1}{x_j}$.

Appendix B

From Equation 3, the relation between W and p_0 is given by the solution for p from the following equation:

$$W = \Sigma_{i=1}^{i=n} c_i (x_i - 1) W^{y_i} p^{x_i}$$

If the jth term on R.H.S. of the above is the dominant term according to the condition described in Section 3.2, then we take $p_0 \approx (\frac{W^{1-y_j}}{c_j(x_j - 1)})^{\frac{1}{x_j}}$ as the approximate solution. Now we show that the speedup is indeed (asymptotically) maximum for this value of p_0.

$$S = \frac{Wp}{W + T_o(W,p)}$$

Since the maximum speedup condition is true in asymptotics, we will drop the constants and write order expressions only on the R.H.S..

$$S = O(\frac{W \times W^{\frac{1-y_j}{x_j}}}{W + \Sigma_{i=1}^{i=n}(W^{y_i} \times W^{\frac{1-y_j}{x_j}x_i})})$$

$$S = O(\frac{W^{1+\frac{1-y_j}{x_j}}}{W + \Sigma_{i=1}^{i=n} W^{y_i + \frac{1-y_j}{x_j}x_i}})$$

The summation $\Sigma_{i=1}^{i=n} W^{y_i+\frac{1-y_j}{x_j}x_i}$ in the denominator on the R.H.S. is at least $\Omega(W)$, because for $i = j$, $W^{y_i+\frac{1-y_j}{x_j}x_i} = W$. So we can ignore the first W in the denominator. Rewriting the expression for speedup, we get:

$$S = O(\frac{W^{1+\frac{1-y_j}{x_j}}}{\Sigma_{i=1}^{i=n} W^{y_i + \frac{1-y_j}{x_j}x_i}})$$

$$S = O(\frac{1}{\Sigma_{i=1}^{i=n} W^{y_i - 1 + (\frac{1-y_j}{x_j} - 1)x_i}})$$

Clearly, the above expression will be maximum when the denominator is minimum, which will happen for the minimum possible value of $\frac{1-y_j}{x_j}$.