MEASURING THE SCALABILITY OF PARALLEL COMPUTER SYSTEMS

J.R. Zorbas, D.J. Reble, and R.E. VanKooten

Myrias Research Corporation
#900, 10611 - 98 Avenue
Edmonton, Alberta, Canada T5K 2P7

Abstract

This paper discusses scalability and outlines a specific approach to measuring the scalability of parallel computer systems. The relationship between scalability and speedup is described. It is shown that a parallel system is scalable for a given algorithm if and only if its speedup is unbounded. A technique is proposed that can be used to help determine whether a candidate model is correct, that is, whether it adequately approximates the system's scalability. Experimental results illustrate this technique for both a poorly scalable and a very scalable system.

Keywords: scalability, measuring scalability, modeling scalability, speedup, parallel computer systems, performance evaluation, Myrias, operating systems

1. Introduction

Inherent in the notion of a parallel computer system is that the number of processors making up the system can be chosen arbitrarily. This extra degree of freedom complicates the evaluation of a parallel system's performance. Measuring the performance of a particular number of processors provides a partial understanding; however, a full evaluation of the system requires an understanding of how performance changes as the configuration size is increased.

The scalability of a parallel computer system relates the system's performance to its size. A system is scalable if an increase in the system's size produces an analogous increase in its processing power [1]. In order to measure a system's scalability a more precise definition is needed. The main goal of this paper is to develop such a definition.

Scalability is considered an important attribute of parallel systems [2], and has been adopted as a design goal for several systems [1, 3-5]. Verifying that this design goal has been achieved is an important problem which has received little attention in the literature. In part, this involves measuring how a system scales. Such measurements have recently been reported for two parallel systems [6, 7].

In this paper we develop a theoretical framework for measuring and modeling the scalability of a parallel system. We assume that information about the system can be used to suggest a "candidate model" of scalability. Measurements are then taken to either help support or refute the correctness of the candidate model. Of course, scalability measurements by themselves cannot be used to prove that a candidate model is correct since scalability is a property of an infinite collection of configurations, and measurements can only be made on a finite subset of this collection. This problem is discussed further in sections 4 and 5 of this paper.

Various aspects of scalability have been defined and studied in recent papers. Deshpande and Jenevein have introduced the notions of resource scalability and application scalability [8]. *Resource scalability* deals with the dependence of various architectural properties and their associated costs on the size of the parallel system. For example, the growth rate of the diameter of the network, used to connect a P-processor system, as P is increased, helps determine the resource scalability of the system. *Application scalability* deals with how well a particular application utilizes a parallel system as the size of the system is increased. Ma and Shea [9] have introduced the notion of *downward scalability*, which deals with how the performance of an application, which uses a fixed portion of the parallel system's resources, is affected by increasing the system's size.

In section 2 of this paper we propose a definition of scalability and illustrate this definition with two simple examples. The relationship between scalability and speedup is described in section 3. We show that a parallel system is scalable, as defined in this paper, if and only if one can obtain an arbitrarily large speedup by choosing a large enough configuration size for the system. Section 4 describes a technique which can be used to help determine whether a candidate model of scalability is correct. Section 5 contains an example of how this technique can be applied.

2. Scalability

In this section we give a definition of scalability that is precise enough to allow the scalability of a parallel computer system to be measured. This definition is algorithm dependent and corresponds to application scalability as defined by Deshpande and Jenevein [8].

Consider a parallel algorithm which is used to solve a problem on a P-processor system. We require that the amount of work done by the algorithm, denoted W, is an increasing function of P. To see why this is necessary, suppose that the work done by the parallel algorithm is fixed. Since there is only a fixed amount of work that can be done in parallel, increasing the system size beyond some maximum number of processors will not increase the system's processing power. Note, this argument assumes that the system's processing power is measured using a single algorithm.

The dependence of the algorithm's work on P can be determined by a theoretical analysis of the algorithm. In many instances, the amount of work done by an algorithm depends on a single parameter, say n. In some cases n gives the input size, while in other cases n may define the accuracy of the solution obtained by the parallel algorithm. For a particular value of n, the algorithm's "intrinsic parallelism" determines the maximum size of the parallel system that can effectively be used to solve the problem. The following parallel algorithm for multiplying two n by n matrices provides a simple example:

```
PARDO I = 1, n
   PARDO J = 1, n
      C[I, J] = 0
      DO K = 1, n
         C[I, J] = C[I, J] + A[I, K] * B[K, J]
      ENDDO
   ENDPARDO
ENDPARDO
```

This algorithm makes use of at most n^2 processors, so to make effective use of a P-processor system we choose $n \propto \lceil P^{1/2} \rceil$.

Our definition of scalability requires that the algorithm make effective use of the P-processor system for every $P \geq 1$. This implies that W is chosen to be an increasing function of P, and each processor in the system has a share of the work.

The work done by the algorithm can be expressed as $W = W(P) = W_s(P) + W_p(P)$, where $W_s(P)$ is the serial part of the work, and $W_p(P)$ is the parallel part. We assume that the time needed to run the parallel algorithm on a single processor system is proportional to the amount of work. In this paper, "run an algorithm" means execute a program corresponding to the algorithm. Thus, the time needed to run the parallel algorithm on a single processor system is given by

$$\textit{sequential run time} = C * [W_s(P) + W_p(P)]$$

where C is a positive constant. The ideal run time of the algorithm on a P-processor system is defined by

$$\textit{ideal run time} = C * [W_s(P) + W_p(P)/P]$$

This is the ideal run time of the parallel algorithm for the chosen dependence of W on P. Note, the exact dependence of work on system size is not very important. What is important is that a particular choice is made. This choice determines the corresponding ideal run time curve. By measuring how the actual run time curve compares to the ideal curve we can determine how well the system scales.

We now introduce the concept of an overhead function.

Definition 1: Assume that a parallel algorithm makes effective use of a P-processor parallel system. A parallel computer system *scales with overhead* $\Phi(P)$ for the parallel algorithm if the actual run time, T, on a P-processor system, satisfies

$$T \leq C * [W_s(P) + W_p(P) / P] * \Phi(P), \quad P \geq 1 \tag{1}$$

A function $\Phi(P)$ which satisfies (1) is called an *overhead function.*

The smallest overhead function which satisfies (1) will be referred to as the *system's overhead function* for the given algorithm and is defined by

$$\Phi_S(P) = \frac{T}{C * [W_s(P) + W_p(P) / P]}$$

For this overhead function, we refer to the right side of (1) as the *correct model of scalability.* We attempt to understand the system's scalability by producing an overhead function, referred to as a "candidate overhead function", which approximates the system's overhead function. In this case we will refer to the right side of (1) as a *candidate model of scalability.*

Rewriting (1) as follows

$$T \leq C * [W_s(P) * \Phi(P) + (W_p(P) * \Phi(P)) / P]$$

suggests that the overhead function represents an increase in the amount of parallel and serial work which must be performed to run the algorithm. This increase is due to the system overhead needed to run the parallel algorithm. For an ideal system, the overhead function is constant and we say such a system is *ideally scalable* for the algorithm. For a more realistic system, the overhead will grow as P is increased. The rate of increase of the overhead function determines how well the system scales; the slower the increase, the better it scales.

There is always some benefit to using a P-processor system if the actual run time is less than the sequential run time, that is

$$C * [W_s(P) + W_p(P) / P] * \Phi(P) < C * [W_s(P) + W_p(P)], \; P \geq 1 \tag{2}$$

Let $\alpha(P)$ be the proportion of serial work making up the parallel algorithm, that is

$$\alpha(P) = W_s(P) / (W_s(P) + W_p(P))$$

then (2) can be written as

$$\Phi(P) < \frac{1}{\alpha(P) + (1 - \alpha(P)) / P}, \quad P \geq 1 \tag{3}$$

The benefit increases if, as P increases, the right side of (2) increases faster than the left side. This observation motivates our definition of scalability.

Definition 2: A parallel system is *scalable* for a parallel algorithm if the system scales with an overhead $\Phi(P)$ which satisfies

$$\Phi(P) * [\alpha(P) + (1 - \alpha(P)) / P] \rightarrow 0 \text{ as } P \rightarrow +\infty \tag{4}$$

In order for a system to be scalable for a parallel algorithm, the proportion of serial work must go to zero fast enough, i.e., $\Phi(P) * \alpha(P) \rightarrow 0$ as $P \rightarrow +\infty$, and the overhead function cannot increase too fast, i.e., $\Phi(P) / P \rightarrow 0$ as $P \rightarrow \infty$. We give two simple examples to illustrate this definition.

First, consider the parallel algorithm for matrix multiplication, given at the start of this section. In this case, $n \propto \lceil P^{1/2} \rceil$ and the theoretical run time is given by $C * [W_p(P) / P]$, where $W_p(P) \propto \lceil P^{1/2} \rceil$. The serial portion of this parallel algorithm is zero, since the program corresponds exactly to the pseudo-code description of the matrix multiplication algorithm. Thus, $\alpha(P) = 0$, and a parallel system is scalable for this parallel algorithm if $\Phi(P) / P \rightarrow 0$ as $P \rightarrow +\infty$.

A second example is provided by the usual parallel algorithm for computing the Fast Fourier Transform [10] of a n element vector.

```
DO stage = 1, log(n)
   PARDO I = 1, n
      — do a "butterfly" transform
   ENDPARDO
ENDDO
```

For this parallel algorithm we make effective use of a P-processor system if we chose $n \propto P$. In this case the ideal run time is given by $C * [W_s(P) + W_p(P) / P]$, where, $W_s(P) \propto log(P)$ and $W_p(P) \propto P * log(P)$. It is easy to verify that a parallel system is scalable for this parallel algorithm if $\Phi(P) / P \rightarrow 0$ as $P \rightarrow +\infty$.

For the two examples given in this section, the system is scalable if $\Phi(P)/P \rightarrow 0$ as $P \rightarrow +\infty$. How well the system scales is governed by how fast $\Phi(P)/P$ goes to zero as P is increased. Thus, a system with overhead $\Phi(P) = log(P)$ scales better than a system with overhead $\Phi(P) = P^{1/2}$.

3. Speedup

One important measure of the performance of a parallel algorithm is given by the *speedup*, denoted by $s(P)$, and defined by

$$s(P) = \text{sequential run time / parallel run time}$$

The "parallel run time" is the time needed to run the parallel algorithm on a P-processor system. The "sequential run time" is usually taken to be the time of the best sequential algorithm which solves the problem on a single processor system. In this paper we will take the "sequential run time" to mean the time taken by the parallel algorithm on a single processor system. This form of speedup is sometimes referred to as "rough speedup" [11]. Examples of how speedup is used to measure performance and further references to the literature dealing with this topic can be found in two recent papers [12, 13].

In this section we show that a parallel system is scalable for a parallel algorithm if and only if an arbitrarily large speedup can be obtained by choosing a large enough system. This result makes precise the intuitive notion that for a scalable system one can obtain an increase in the processing power of the system by increasing the system's size.

Arguments have been made that Amdahl's law [1], given by

$$s(P) \leq \frac{1}{\alpha(P) + (1 - \alpha(P)) / P} \leq 1/\alpha(P)$$

restricts the maximum possible speedup for a given parallel algorithm. The restriction on performance imposed by this law has recently been reexamined by Gustafson, et. al. [14, 15]. They argue that the work done by a parallel algorithm should be increased as the size of the system is increased, and thus the resulting "scaled speedup" can be arbitrarily large. We also have adopted this approach.

The relationship between scalability and speedup is given by the following result.

Theorem 1: Assume that a parallel algorithm makes effective use of a P-processor system. The system is scalable for the parallel algorithm if and only if its speedup is unbounded, i.e., $s(P) \longrightarrow +\infty$ as $P \longrightarrow +\infty$.

Proof: First, we show that scalability implies unbounded speedup. The time needed to run the parallel algorithm on one processor is given by

$$\text{sequential run time} = C * [W_s(P) + W_p(P)]$$

Since the system is scalable for the parallel algorithm, we have

$$\text{parallel run time} \leq C * [W_s(P) + W_p(P) / P] * \Phi(P)$$

Thus

$$s(P) \geq \frac{C * [W_s(P) + W_p(P)]}{C * [W_s(P) + W_p(P) / P] * \Phi(P)} \tag{5}$$

$$= \frac{1}{[\alpha(P) + (1 - \alpha(P)) / P] * \Phi(P)}$$

Since, by Definition 2, we have

$$[\alpha(P) + (1 - \alpha(P)) / P] * \Phi(P) \longrightarrow 0 \text{ as } P \longrightarrow +\infty$$

we conclude that $s(P) \longrightarrow +\infty$ as $P \longrightarrow +\infty$.

We now show that unbounded speedup implies scalability. By definition of the system's overhead function, the system scales with overhead $\Phi_S(P)$ and we have

$$\text{parallel run time} = C * [W_s(P) + W_p(P) / P] * \Phi_S(P)$$

Thus

$$s(P) = \frac{C * [W_s(P) + W_p(P)]}{C * [W_s(P) + W_p(P) / P] * \Phi_S(P)}$$

$$= \frac{1}{[\alpha(P) + (1 - \alpha(P)) / P] * \Phi_S(P)}$$

Since we are assuming $s(P) \longrightarrow +\infty$ as $P \longrightarrow +\infty$, we have

$$[\alpha(P) + (1 - \alpha(P)) / P] * \Phi_S(P) \longrightarrow 0 \text{ as } P \longrightarrow +\infty$$

Thus, we have exhibited an overhead function which satisfies the definition of scalability.

Knowing that a parallel system is scalable for a parallel algorithm has very important implications. By making the system large enough, and appropriately increasing the size of the problem, we can obtain any desired speedup.

4. Determining the Correct Model

Determining the correct scalability model for a given parallel computer system and a particular parallel algorithm is a complex problem. In most instances, an understanding of how the system executes the algorithm will suggest a candidate for the correct model. For example, if the system is message–based, a count of the number of messages needed to execute the problem will suggest an overhead function. Of course, such an approach will not always yield the correct model since the performance of the system does not only depend on the number of messages being sent. In fact, a variety of factors play a role in determining the scalability of a parallel computer system. These include: the connectivity of the processors, how the communication system is used, the distribution of operating system work across the processors, memory contention [16], and task synchronization [16, 17]. We will assume henceforth that an educated guess has been made to obtain a candidate for the correct model of scalability.

To determine whether a candidate model is an adequate approximation to the correct model, one must try to fit run time versus configuration size data with the candidate model. In some cases, it is obvious that the model does not fit the data. Then, one must revise the candidate model. This revision will probably require a better understanding of how the system executes the program, since the analysis for the candidate model must not have considered some aspects of the parallel computer system and algorithm.

In many cases, deciding whether the candidate model fits the data is not so straightforward. There are at least three reasons for this. First, experimental data is noisy. Second, any candidate model is, at best, an approximation to the correct model. Most candidate models are many–times differentiable, increasing functions of the number of processors. However, it is very unlikely that a graph for the correct model is differentiable and it may not even be increasing. Thus the best we can hope for is to obtain a "good" approximation to the correct model of scalability. Third, a good approximation may involve one or more extra terms not appearing in the candidate model. The constants appearing in these terms may be relatively small, and thus the missing terms do not dominate for small configurations of the parallel system. Of these three problems, the third is the most serious.

The first problem can be reduced to an acceptable level by repeating the measurement of each data point and averaging the resulting values. We believe that the second problem is not serious in practice. Essentially, we are assuming that the run times do not behave erratically as the number of processors is increased. The third problem would not be serious if one could configure arbitrarily large parallel systems. If this were possible, one could perform a sequence of experiments using increasingly large systems to determine statistically whether the candidate model is a suitable approximation to the correct model. Unfortunately, in most instances the hardware resources are limited to some fixed number of processors, denoted by $P_{\max}$. Thus, we must decide whether a candidate model is a suitable approximation to the correct model of scalability, using only data for configurations having at most $P_{\max}$ processors. In the following text, we suggest one approach that can be used to deal with the third problem.

For a message–based parallel system, the execution time of a program depends, in part, on the number of messages needed by the operating system to execute the program. Thus, the growth in the number of messages with configuration size partially determines the correct model of scalability for the system. It may be that the candidate model has not adequately accounted for the growth in the number of messages with configuration size. For example, we might adopt a candidate model of scalability with overhead function

$$\Phi(P) = a_1 + b_1\, log(P) \tag{6}$$

whereas in reality, the correct model of scalability may correspond to the following overhead function

$$\Phi(P) = a_2 + b_2\, log(P) + c_2\, P$$

where the constant c_2 is quite small. The constant term represents that portion of the execution time which does not depend on the configuration size, and should not depend strongly on the number of messages processed. For example, going from one to two processors increases the number of messages sent, but does not change the constant term. Neither should it depend strongly on the average time to send and process a message, denoted by T_{avg}. The constants appearing in the remaining terms are expected to depend strongly on T_{avg}. For example, if T_{avg} were to approach zero, we would expect these constants to approach zero. If messages are processed instantly, the overhead of executing tasks on the processors making up the system would be zero, independent of the number of processors used. Note, this argument is only valid if the system and problem are such that resource contention does not occur. Thus, if we increase the value of T_{avg} we would expect both b_2 and c_2 to increase. As T_{avg} becomes sufficiently large, the $c_2\, P$ term should dominate the $b_2\, log(P)$ term for P approaching $P_{\max}$.

Now, consider the candidate model corresponding to the overhead function (6). By appropriately slowing down the message passing system we can obtain run time versus configuration size data for a sequence of values of T_{avg}, say $T_{avg}=t_1,t_2,\cdots,t_k$. For each value of T_{avg}, we divide the actual run times by the corresponding ideal run times and obtain a least squares fit, which yields values for a_1 and b_1. Thus, we obtain a sequence of values for a_1 and b_1. If the candidate model is not a suitable approximation to the correct model, we expect that this sequence of values will not be consistent, i.e., the values for a_1 will not be constant as T_{avg} is increased.

5. Experimental Results

Using the approach outlined in the previous section, we ran a test program, called *ising*, on a prototype parallel computer system developed at Myrias Research Corporation (see [5] for a general description of the prototype system). This test program is a simplified version of code used to solve the Ising Model in statistical physics [18]. The *ising* program is completely parallelizable, and the amount of work contained in this program was chosen to be directly proportional to the number of processors used to run the program.

The test program *ising* is based on the following algorithm:

```
PARDO sample = 1, n
   for each lattice point (x,y,z)
      — perform spin-flip on (x,y,z)
      — cumulate change in magnetism
ENDPARDO
```

We ran the program on configurations from 2 to 256 processors, using a preliminary version of the Myrias control mechanism. This control mechanism suffered from several problems. In particular, certain control messages were used excessively, and there were "hot spots", that is, concentrations of operating system work on certain processors. We will refer to this preliminary control mechanism as PCM (Preliminary Control Mechanism).

The message system was degraded by placing a busy loop in the routine used to send messages. By varying the *loop_count*, we could control the severity of the degradation. Figure 1 gives run time versus configuration size data for four values of *loop_count*.

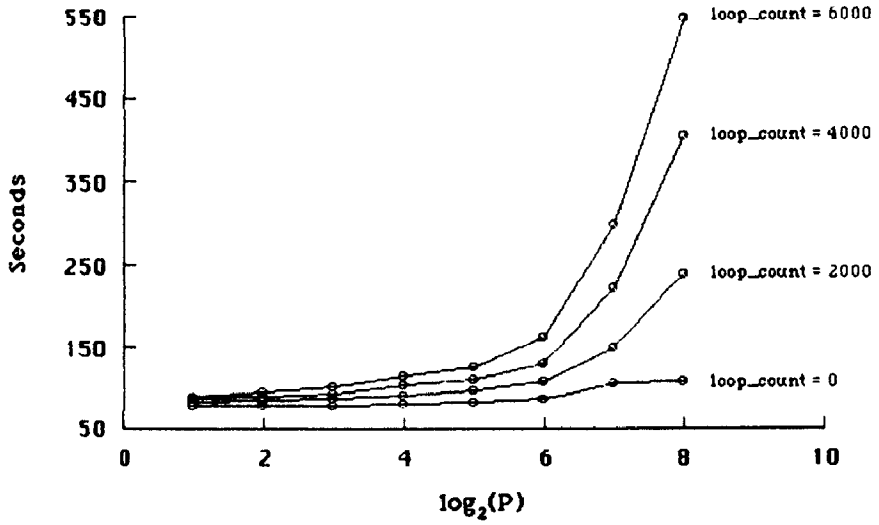

Figure 1: Execution times for the test program *ising* using the PCM.

Clearly, the candidate model corresponding to the overhead function (6) does not fit the data of Figure 1 for $loop\_count > 0$. Note, however, that this model does fit the data points for $loop\_count = 0$ reasonably well. The scalability problems are only apparent when the message–passing system is degraded. If one actually tries to fit all four collections of data points using (6), we obtain the results given in Table 1. Note, it is not necessary to divide the actual run times by the corresponding ideal run times since *ising* is completely parallelizable and the ideal run time is constant. Clearly, the a_1 values are not constant, refuting the model.

loop_count	0	2000	4000	6000
a_1	65.9	36.1	-6.3	-44.7
b_1	4.6	17.9	35.8	52.4

Table 1. Constants for the candidate model (6) using the data of Figure 1.

We next ran *ising* on the Myrias prototype system using the current version of the Myrias control mechanism, which we will refer to as MCM (Myrias Control Mechanism) [19]. This version does not make excessive use of control messages and has a fairly uniform distribution of control mechanism work. A simple message–counting argument suggests that the candidate model corresponding to the overhead function (6) is an adequate approximation to the correct model. To test this we again obtained run time versus work data for four values of *loop_count* (see Figure 2). These results are quite good, and a least squares fit using (6) yield the results of Figure 3. Table 2 gives the values for a_1 and b_1 which were obtained.

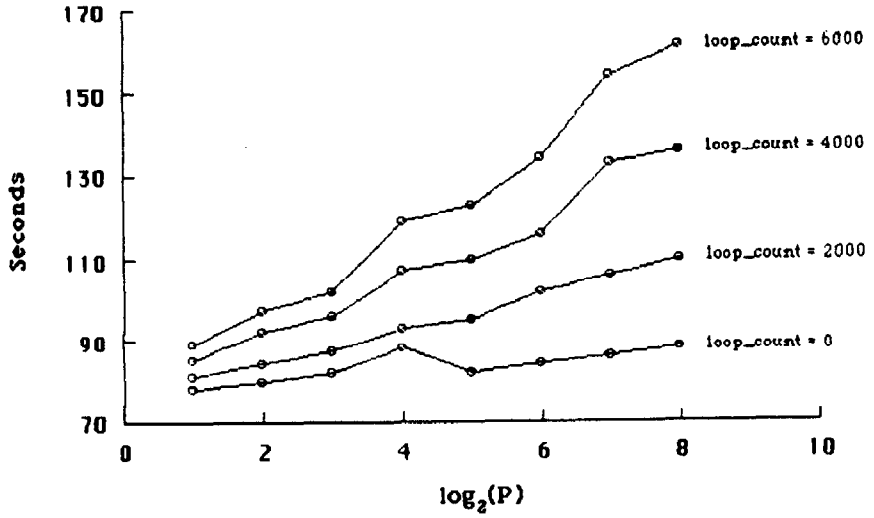

Figure 2: Execution times for the test program *ising* using the MCM.

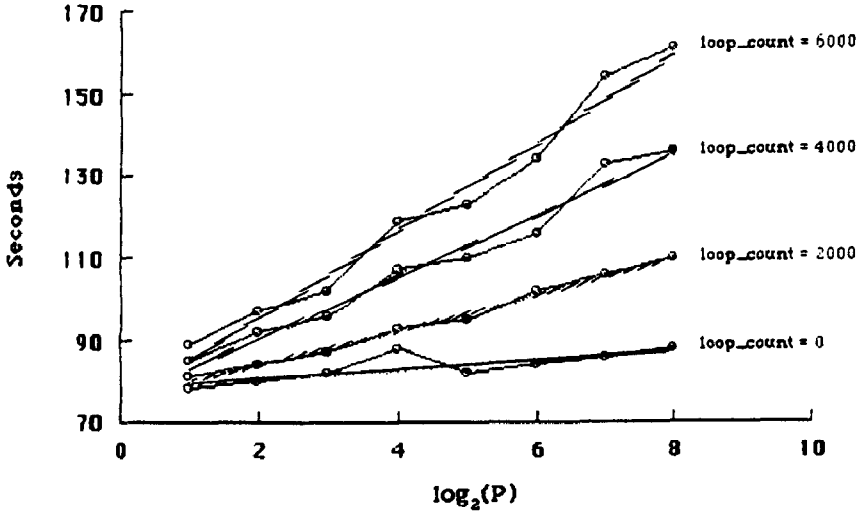

Figure 3: Data from Figure 1 fitted using model (6)

loop_count	0	2000	4000	6000
a_1	78.1	75.5	75.9	74.8
b_1	1.2	4.3	7.4	10.6

Table 2. Constants for the candidate model (6) using the data of Figure 2.

In Table 2, a_1 is essentially constant and the values of b_1 are linearly related to *loop_count* (see Figure 4).

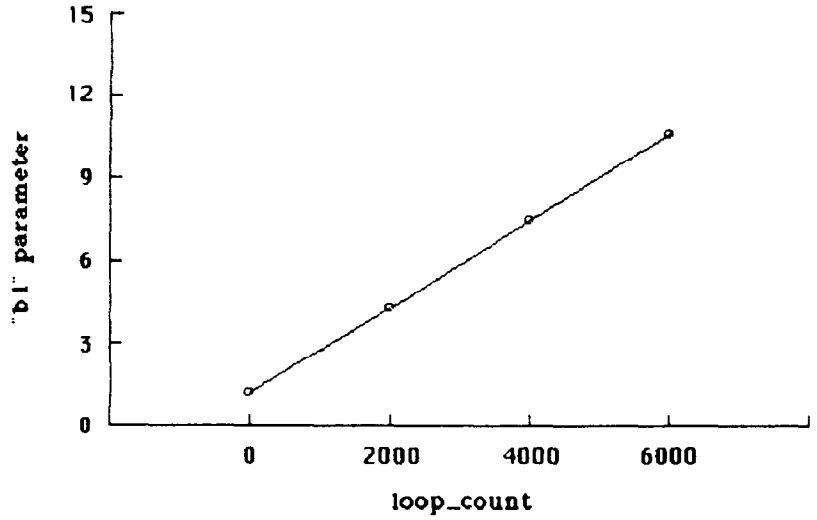

Figure 4: Least squares fit of the b_1 values of Table 2.

The results presented in this section provide further evidence of the scalability of the Myrias Research Corporation prototype parallel computer system. Additional scalability results for this system are given in [6].

6. Concluding Remarks

We emphasize that scalability, as defined in this paper, is an asymptotic property of parallel systems. This implicitly assumes that there is no upper bound on the size of a parallel system. In reality, various physical considerations, such as the finiteness of the speed of light, do limit the size of parallel systems. We ignore these physical limitations, since current systems would have to be greatly increased in size before these constraints limit performance.

From a more practical point of view, what is really important is how well the system scales, i.e., how slowly the overhead function increases as P increases for all values of P. This determines, as can be seen from inequality (5), the size of the speedup for each $P \geq 1$. The smaller the value of the overhead function, the larger the value of the speedup.

The approach described in section 4 of this paper is a very simple technique for checking the validity of a candidate model for scalability. We emphasize, however, that this approach cannot be used to verify that the candidate model is indeed an adequate approximation to the correct model. There are only two ways of verifying a model of scalability. First, develop a theoretical model which correctly takes into account those aspects of the parallel system that affect scalability, and deduce from this model that the system scales according to a particular model; this is a formidable task. Second, undertake a statistical analysis using a sequence of experiments with increasingly large configurations; but this may not be practical due to hardware limitations. If neither of these two approaches can be taken, one can develop techniques, such as the technique described in this paper, which can be used to help determine a system's scalability.

Acknowledgements:

The authors of this paper wish to thank Kent McPhee and Dick Foster for the use of their *ising* code. As well, they would like to thank Monica Beltrametti for her support and encouragement and Al Covington for reading the final version of the manuscript.

7. References

1. M. J. Quinn, *Designing Efficient Algorithms for Parallel Computers*, McGraw-Hill, New York, 1987.
2. P. P. Patton, Multiprocessors: Architecture and Applications, *Computer June*, (1985), 29-40.
3. S. A. Ward, The MuNet: A Multiprocessor Message-Passing System Architecture, *Proc. Seventh Texas Conference on Computing Systems, , Nov., 1978, 21-24.*
4. W. D. Hillis, *The Connection Machine*, MIT Press, 1986.
5. A. M. Kobos, R. E. VanKooten and M. A. Walker, *The Myrias Parallel Computer System*, Myrias Research Corp., Edmonton, Alberta, 1986. *Proceedings of the Colloquium on Numerical Aspects of Vector and Parallel Computers* C.W.I. Amsterdam (1987).
6. C. Thomson and K. McPhee, Experience Scaling the Myrias System to Hundreds of Processors, *Proceedings of the Third International Conference on Supercomputers, Vol.II*, May 15-20, 1988, *Boston, 1988, 125-129.*
7. S. P. Kumar and S. M. Stern, A Scalability Analysis of the Butterfly Multiprocessor, *Proceedings of the Third International Conference on Supercomputers Vol. II, May 15-20*, (1988), 101-108.
8. S. R. Deshpande and R. M. Jenevein, Scalability of a Binary Tree on a Hypercube, *Proc. IEEE 1986 International Conference on Parallel Processing, , 1986, 661-668.*
9. Y. E. Ma and D. G. Shea, Downward Scalability of Parallel Architectures, *Proceedings of the Third International Conference on Supercomputing Vol. II, May 15-20*, (1988), 109-120.
10. D. F. Elliot and K. R. Rao, *Fast Transforms: Algorithms, Analyses, Applications*, Academic Press, New York, 1982.
11. R. A. Finkel, Large-grain Parallelism - Three Case Studies, in *The Characteristics of Parallel Algorithms*, L. H. Jamieson, D. B.Gannon and R. J. Douglas (ed.), MIT Press, Cambridge, MA, 1987, 21-63.
12. C. D. Polychronopoulous and U. Banerjee, Processor Allocation for Horizontal and Vertical Parallelism and Related Speedup Bounds, *IEEE Trans. on Computers Vol. C-36, No. 4*, (1987), 410-420.
13. Z. Cvetanovic, The Effects of Problem Partitioning, Allocation, and Granularity on the Performance of Multiple-Processor Systems, *IEEE Trans. on Computers Vol. C-36, No. 4*, (1987), 421-432.
14. J. L. Gustafson, Amdahl's Law Re-evaluated, *Comm. of the ACM 31*, (1988), 130-133.
15. J. L. Gustafson, G. R. Montry and R. E. Benner, Development of Parallel Methods for a 1024-Processor Hypercube, *SIAM J. on Sci. Stat. Computing Vol. 9, No. 4, July*, (1988), 609-638.
16. R. Mehrotra and E. F. Gehringer, Superlinear Speedup Through Randomized Algorithms, *Proceedings of the 1985 International Conference on Parallel Processing, ,* 1985, 291-300.
17. T. S. Axelrod, Effects of Synchronization Barriers on Multiprocessor Performance, *Parallel Computing Vol. 3*, (1986), 129-140, North-Holland.
18. K. Binder and D. Stauffer, A Simple Introduction to Monte Carlo Simulation and Some Specialized Topics, *Applications of the Monte Carlo Method in Statistical Physics*, New York, 1984.
19. M. Beltrametti, K. Bobey and J. R. Zorbas, The Control Mechanism for the Myrias Parallel Computer System, *Computer Architecture News Vol. 16, No. 4*, (1988), 21-30.