Speedup and Scalability of SPMD Computations on Workstation Clusters

K. Efe
Department of Computer Engineering
and Information Sciences
Bilkent University
Bilkent, Ankara, Turkey

P. Uthayopas
Department of Computer Science
Kasetsart University
Thailand

A. Broadwater
Department of Computer Science
University of Southwestern Louisiana
Lafayette, LA

Abstract *Scalabiltity is among the most important advantages attributed to workstation clusters. A new application may be initially started with a few workstations, and new workstations may be added as demand increases for computing power. However, designing scalable systems requres fine-tuning of several hardware and software parameters. This paper introduces the* maximum-speedup scalability *as a performance metric to be used when evaluating design alternatives for a hardware/software combination, and demonstrates the model with measurements taken from real implementations.*

Keywords: Workstation clusters, scalability, speedup, parallel computing

1 Introduction

Ideally, we would like to double the speed of parallel computations when we double the number of processors, but this is not possible with the exception of a few trivial cases. Often, parallel speed depends on a complex interplay between the system architecture and software, as well as special parameters arising from the hardware. Beyond a certain level of parallelism, the speed of a parallel computation may actually deteriorate. This phenomenon has been studied intensively and various measures of speedup and scalability have been defined. These models help better understand the couses of overheads and allow the system designer to fine-tune the system for better performance, but these models are inadequate when we consider certain price/performance issues. In this paper, the *maximum-speedup scalability* is introduced and illustrated with experimentation on a workstation clusters Grad and Basin at the Center for Advanced Computer Studies at the University of Southwestern Louisiana.

Related Work One factor that limits the parallel performance pointed out by Gene Amdahl [1] is the sequential bottleneck. If f is the fraction of code that can not be parallelised, the maximum speedup that can be obtained by parallel execution is limited by an amount proportional to $1/f$. Typically, increasing the level of parallelism increases the amount of speedup until a certain point, beyond which increased parallelism becomes detrimental to speedup due to increased overheads. Speedup in the context of specific applications such as elliptic partial differential equations, grid iteration problems, and parallel VLSI test pattern generations have been studied in [11, 15, 2]. Gustafson, Montry, and Benner at Sandia National Laboratory [6] experimentally demonstrated that the sequential part, which usually consists of the initialization code, and the program loading and preparation, did not scale with the problem size. Therefore, speedup can be increased by increasing the problem size. Among other speedup models, the scaled speedup model [5, 7, 16, 17] focuses on real-time applications where the amount of time allowed for completion of a computation may be fixed.

Different scalability models have also been presented. The isoefficiency scalability metric has been proposed by Grama, Gupta and Kumar [3]. According to this metric, the scalability of parallel systems is indicated by the ability to maintain a fixed efficiency when the problem size and number of processors are increased proportionally. Further discussions and applications of the isoefficiency analysis can be found in [10, 4, 3, 9]. Recently, Sun and Rover [13, 14] have proposed the concept of isospeed scalability. Some relations concerning this scalability metric and isoefficiency metric have been discussed in [14].

Relationship to This Work Although various models described above can provide useful information, the issues of communication overhead and the effect of limited memory size have not been addressed satisfactorily. The proposed model is an attempt to fill this gap.

For a given problem size n, there is an optimum number p of workstations that maximizes the speedup S. Conversely, for a given number of workstations, there is a problem size that admits this number of workstations as its optimum number. In case the same optimal speedup is attained by different problem sizes, we are interested in the smallest problem size that satisfies this optimum level. When the number of workstations is increased from p to p', the problem size must also increase from n to n' to reach the new minimum problem size that admits p' as its optimum number of workstations.

The proposed scalability metric measures the amount $\Delta n = n' - n$ by which the problem size needs to be increased in order to maintain the speedup at its optimum level. We note that optimum speedup may be different for different p values. We say that an algorithm/system combination with a small Δn value is more scalable than another one with large Δn value. As the memory requirement of a computation increases with increasing problem size, and the system has a finite memory, there is a limit beyond which problem size cannot be increased. This fact emphasizes the need for algorithm/system combinations that are scalable according to the measure proposed here.

Another important issue is how to upgrade the system when the current configuration is not satisfactory for the needs of a computation. Based on the proposed scalability model, we present an example in Section 5 where we find that, for a particular matrix-vector multiplication algorithm running on a cluster of workstations, adding memory to each workstation can improve the speedup more than adding several workstations to the cluster. Various enhancement alternatives such as increasing the number of processors, increasing the memory size, or upgrading the communication network, that can be evaluated in the proposed model, are among the important issues not addressed in the earlier models.

2 The Speedup Model

The most suitable parallel programming model for workstation clusters is the SPMD (Single Program Multiple Data) model, where the same program runs on many workstations, but the data used to perform the computation are different.

In a typical SPMD parallel program The execution consists of 3 phases: (1) *Sequential phase* involves the execution of a sequential part of the code. (2) *Parallel phase* performs the computation of parallel part concurrently. (3) *Communication phase* involves the exchange of messages between processes. Phases (2) and (3) may repeat a number of times, and our model lumps together all execution activities and communication activities occurring in different iterations of these phases.

Let t_{seq} and t_{par} denote the unit execution time of a basic operation for the sequential part and the parallel part, respectively. The computation work can be defined as the number of times a basic operation is repeated to complete the task. This work is a function of the problem size n. Let $W_s(n)$ be the sequential work and $W_p(n)$ be the parallel work. The sequential execution time of a program for a problem size n is

$$T_s(n) = W_s(n)t_{seq} + W_p(n)t_{par} \tag{1}$$

Let $T_{comm}(n,p)$ be the communication time and p be the number of workstations used. The parallel execution time is given by

$$T_p(n,p) = W_s(n)t_{seq} + \frac{W_p(n)t_{par}}{p} + T_{comm}(n,p)$$

The communication time for a single message can be divided into two parts. One part depends on the message startup time and another part depends on the network bandwidth. Let $C_s(n,p)$ be the communication cost function that depends on the message startup time and $C_c(n,p)$ be the communication cost function that depends on the network bandwidth or the per-byte communication time. The communication time can be expressed as the sum of these two functions as follows:

$$T_{comm}(n,p) = C_s(n,p)t_s + C_c(n,p)t_c$$

Then, the parallel execution time becomes

$$T_p(n,p) = W_s(n)t_{seq} + \frac{W_p(n)t_{par}}{p} + C_s(n,p)t_s + C_c(n,p)t_c \tag{2}$$

The Communication-Aware Speedup Model: Before proceeding further, we need to introduce the following notation:

- $R_{sp}(n)$: is the ratio of sequential work to parallel work, $R_{sp}(n) = W_s(n)/W_p(n)$.
- r_{sq}: is the ratio of the unit execution time of sequential part to parallel part, $r_{sq} = t_{seq}/t_{par}$.
- r_s: is the ratio of the communication startup time to the unit execution time of the parallel part, $r_s = t_s/t_{par}$.
- r_c: is the ratio of the per-byte communication time to the unit execution time of the parallel part, $r_c = t_c/t_{par}$.

Formally, speedup of parallel processing with p processors for a problem of size n is defined as $S(n,p) = \frac{T_s(n)}{T_p(n,p)}$. Then, using equations 1 and 2, after simplification we obtain

$$S(n,p) = \frac{R_{sp}(n)r_{sq} + 1}{R_{sp}(n)r_{sq} + \frac{1}{p} + \frac{C_s(n,p)r_s}{W_p(n)} + \frac{C_c(n,p)r_c}{W_p(n)}}$$

From our model, we derive the optimum number of workstations in the following theorem.

Theorem 1 *[12] The number of workstations p that achieves the maximum speedup satisfies*

$$r_s\frac{dC_s(n,p)}{dp} + r_c\frac{dC_c(n,p)}{dp} = \frac{W_p(n)}{p^2}$$

There are two types of parameters used in our analysis. The parameters r_s and r_c are system dependent, which can be measured by experiments. The functions $W_p(n)$,$W_s(n)$, $C_s(n,p)$ and $C_c(n,p)$ are algorithm dependent. These functions can be formulated based on the running time function of the algorithm, and the amount of data exchanged between workstations.

The Memory-Aware Speedup Model: We now extend the proposed model to consider the effect of the memory access characteristics of a workstation. Let n be the problem size and p be the number of workstations. The memory requirement function M(n,p) is a function that represents the amount of memory requirement per workstation. The memory requirement function of the sequential program is represented by $M(n,1)$. For example, the sequential matrix multiplication requires the storage of $M(n,1) = 3n^2$. In case of a parallel program, the memory requirement depends on both the problem size and the number of workstations. One implementation of the parallel matrix multiplication that we will discuss later requires the memory storage of $M(n,p) = \frac{2n^2}{p} + n^2$ words for each workstation.

In our model, the unit execution time of an algorithm is not uniform but depends on the problem size. This is due to the effect of the memory access delay function. If this effect is considered, the unit execution time can be expressed by

$$t_{par}(n,p) = t_u D(M(n,p)) \tag{3}$$

where t_u is the unit execution time when the memory access is efficient (e.g., when the problem size is small), and $D(M(n,p))$ is a function that represents the increases in the unit execution time as the memory requirement increases. By using this new unit execution time, and after simplification, the resulting speedup is obtained by using equation 3 in equation 2, and then computing $S(n,p)$ as before. (We omit the steps due to space limitations.) This speedup model is referred to as the *memory-aware speedup model.* According to this model, the memory access delay in the sequential program contributes a positive effect to the speedup. Therefore, speedup tends to be unusually high when the sequential execution is slow due to large memory requirements.

Theorem 2 *[12] If the effect of memory access delay is considered, the number of workstations p that achieves the maximum speedup satisfies*

$$r_s \frac{dC_s(n,p)}{dp} + r_c \frac{dC_c(n,p)}{dp} = -W_p(n) \frac{d}{dp} \frac{D(M(n,p))}{p}$$

From this theorem, we can see that when the memory access time is fixed, the memory access delay function is $D(M) = 1$. So, the term $-W_p(n)\frac{d}{dp}\frac{D(M(n,p))}{p}$ is reduced to $\frac{W_p(n)}{p^2}$, a result which is similar to Theorem 1. In this point of view, the communication-aware speedup is a subset of the memory-aware speedup.

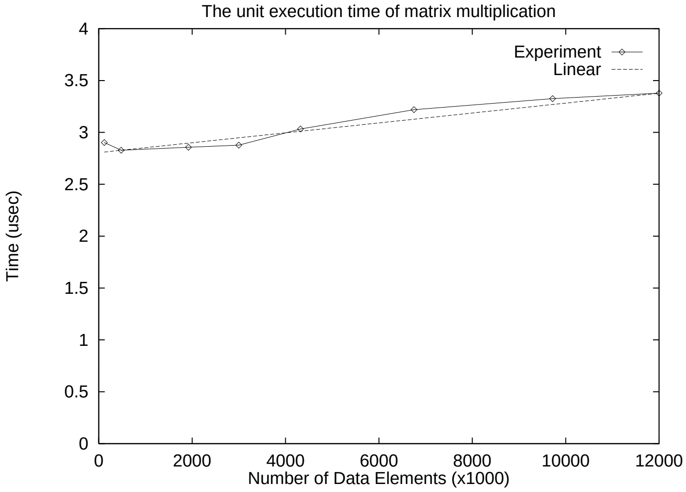

Figure 1: Unit Execution Time on Grad Cluster

There are two memory access delay functions that we have encountered in our systems. Therefore, we will use these two functions to illustrate the concept.

Case 1: $D(n)$ is a linear function: Figure 1 shows the unit execution time of the matrix multiplication program measured from one of our workstations that belong to the Grad system. In this case, a linear function can be used to approximate this function. The estimation is also shown in the same figure. Hence, the memory access delay function can be described as

$$D(n) = a + bM \tag{4}$$

where a,b are constant values and M is the memory requirement function. Since,

$$\begin{aligned}
\frac{d}{dp}\frac{D(M(n,p))}{p} &= \frac{d}{dp}(a/p + bM(n,p)/p) \\
&= -\frac{1}{p^2}(a + bM(n,p)) + bp\frac{dM(n,p)}{dp})
\end{aligned}$$

Using this information in Theorem 2, the optimum number of workstations can be obtained by solving the following condition:

$$\begin{aligned}
& r_s \frac{dC_s(n,p)}{dp} + r_c \frac{dC_c(n,p)}{dp} \\
= & \frac{W_p(n)}{p^2}(a + bM(n,p) + bp\frac{dM(n,p)}{dp})
\end{aligned}$$

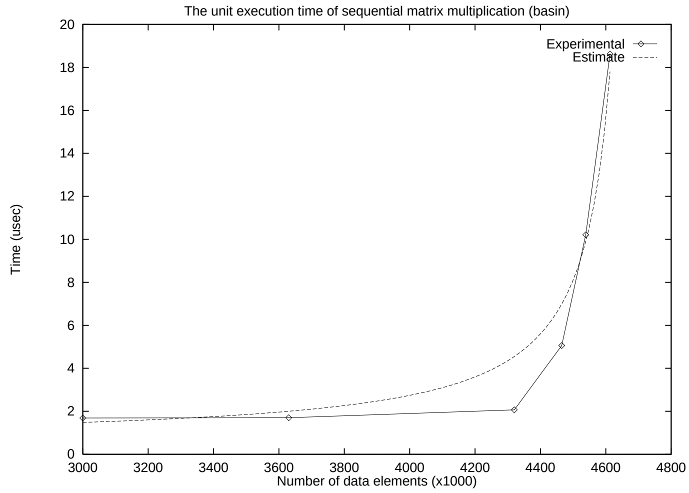

Figure 2: Unit Execution Time on Basin Cluster

Case 2: $D(n)$ is a reciprocal function: Figure 2 shows the unit execution time of the matrix multiplication program that we have observed from Basin cluster. It is more suitable to represent the memory access delay in this case with the reciprocal function such as

$$D(M) = a + \frac{b}{(c - dM)} \tag{5}$$

where a,b,c, and d are constants. The estimation of this memory access delay function is also depicted in Figure 2. In this case, we have

$$\begin{aligned} \frac{d}{dp}\frac{D(n/p)}{p} &= \frac{d}{dp}(a/p + b/(cp - dn) \qquad (6) \\ &= -(a/p^2 + bc/(cp - dn)^2) \end{aligned}$$

Again, by using this information in Theorem 2, the optimum number of workstations is obtained as:

$$r_s\frac{dC_s(n,p)}{dp} + r_c\frac{dC_c(n,p)}{dp} = K(M,n,p)$$

where

$$K(M,n,p) = \frac{W_p(n)}{p^2}(a + \frac{b(c + dp\frac{dM(n,p)}{dp} - dM(n,p))}{(c - dM(n,p))^2})$$

We can see that the consideration of the memory access delay results in additional terms in the optimality criterion. Although more complexity is involved, these additional terms provide more accurate optimization results.

3 Scalability of Workstation Clusters

As we have seen in the previous analysis, if the problem size is fixed, there exists an optimum number of workstations that yields the maximum speedup. In this point of view, the optimum number of workstations is the function of the problem size or $p = g(n)$. This function can be determined from Theorem 1 or Theorem 2, whichever applies for the needs of an application/system combination.

Conversely, there exist a problem size n_{opt} which admits p as its optimum number of workstations, and yields the speedup equal to S_{max}. Here we are interested in the problem size n_{opt} for each system size p since it indicates how the problem size must grow in order to maintain the speedup at S_{max} when the system size grows. We can see that n_{opt} is a function of the system size p, or $n_{opt} = h(p)$. This function can also be computed from Theorem 1 or Theorem 2. We refer to $h(p)$ as the *maximal-speedup scalability function.*

The maximal-speedup scalability function can be used to compare different algorithms or to evaluate different enhancement alternatives for the system. An algorithm with a small maximal-speedup scalability function requires only a small increase in the problem size to maintain the maximum speed on the larger system. In contrast, an algorithm with large maximal-speedup scalability function requires a much larger increase in the problem size. Thus, an algorithm with a smaller maximal-speedup scalability function is more scalable. In this section, we illustrate the proposed model by choosing a parallel matrix multiplication algorithm from the literature [8], and comparing different enhancement alternatives. Due to space limitations we only consider the memory-aware speedup model above.

In this algorithm there is a master process which reads the entire matrix B and sends it to each workstation. The matrix A is divided row-wise into p blocks each of size $\frac{n}{p} \times n$. After a workstation W_i receives a block a_i, $1 \leq i \leq p$, it performs the multiplication $C_i \leftarrow A_i\ X\ B$, and returns the sub-matrix C_i of size $\frac{n}{p} \times n$ to the master process. The master process then combines the sub-matrices to obtain C.

The communication in this algorithm consists of 3 steps:

1. Broadcasting of matrix B to p workstations. The communication time for this step is $p(t_s + n^2 t_c)$
2. Sending a block of matrix A to each of the p workstations. The communication time for this step is $p(t_s + \frac{n^2}{p} t_c)$
3. Gathering the results back. The communication time for this step is $p(t_s + \frac{n^2}{p} t_c)$

According to these communication steps, the communication cost function and the parallel work for this method are given by

$$\begin{aligned} C_s(n,p) &= 3p \quad (7)\\ C_c(n,p) &= (p+2)n^2 \\ W_p(n) &= n^3 \end{aligned}$$

Since there is no sequential work in this algorithm, we have $R_{sp}(n) = 0$. The sequential algorithm needs to store all three matrices A, B and C locally. If the size of each matrix is n, the memory requirement function for the sequential algorithm is

$$M_1(n,1) = 3n^2 \quad (8)$$

For the parallel algorithm, memory is needed for storing the entire matrix B and sub-matrices of A and C of size $\frac{n^2}{p}$ on each slave workstation. Thus, we have the memory requirement function

$$M_1(n,p) = n^2 + \frac{2n^2}{p} \quad (9)$$

Assume that we run this algorithm on a workstation cluster which consists of 16 workstations and the problem size is $n = 3755$. This problem size is selected such that the memory is almost full on all workstations. By using the speedup model above we can compute the effect of edding memory as opposed to increasing the number of workstations (see [12] for

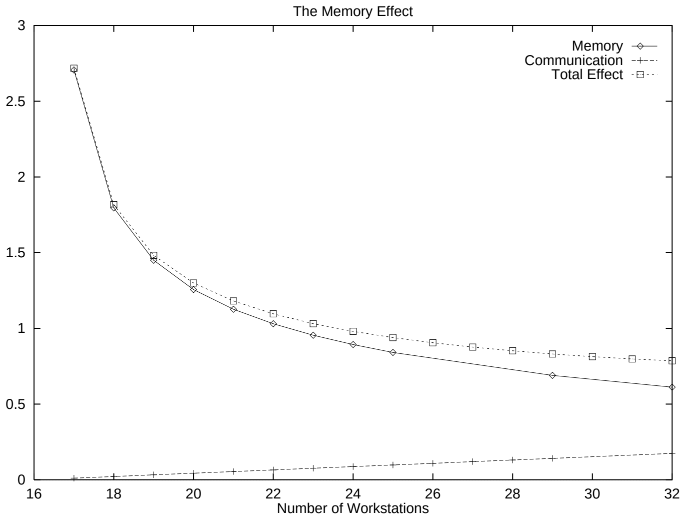

Figure 3: Plot of Each Term in Eq.41.

details). Increasing the memory size of workstations is more advantageous than increasing the number of workstations when:

$$\frac{p}{p'}\left(a + \frac{b}{c - dn^2(1+\frac{2}{p})}\right) + \frac{p(p'-p)r_c}{n} > 1 \quad (10)$$

where p is the initial number of workstations, p' is the increased number of workstations. The first term on the left-hand-side represents the reduction in memory access delay as a result of the additional workstations while the second term represents the increased communication. Figure 3 plots each term and their total values using parameter values measured from the Basin cluster.

We calculate the difference of the left-hand-side from the right-hand-side as shown in Figure 4. We can see that the condition presented in Eq. 10 is satisfied when the total number of workstations is less than 23 (The point where the graph goes below 0). Before this point, we can improve the speedup by adding more memory instead of more workstations. For example if we initially had 17 workstations, we could improve the speedup by adding sufficient amount of memory; and the resulting speedup will be same as if we added 6 more workstations to the cluster. However, if we have more than 23 workstations initially, it is not possible to gain a better speedup by adding more memory to

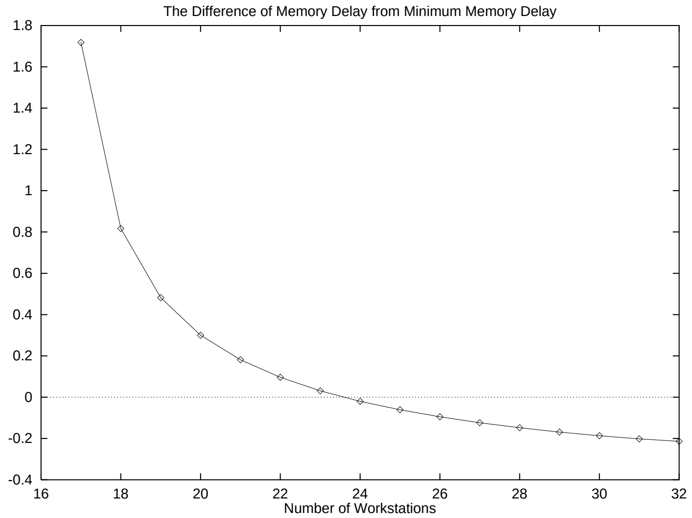

Figure 4: Difference of the left-hand-side from the right-hand-side in Eq. 10.

each workstation.

4 Summary and Conclusions

The main contribution of this paper is in the provision of comprehensive speedup models and a new scalability metric for workstation clusters. These models enhance the understanding about the behavior of parallel algorithms on workstation clusters.

Acknowledgements: This research was supported by a grant from the Turkish Scientific and Engineering Research Council, grant No: EEEAG-199E013; a grant from the Louisiana Board of Regents, grant no: LEQSF(1998-01)-RD-A-36; and a grant from the U.S. Department of Energy, grant no. DE-FG02-97ER1220.

References

[1] G. M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In A.S. Hoagland, editor, *American Federation of Information Processing Society Conference Proceedings*, volume 30, pages 483–485. Academic Press, London, 1967.

[2] Hideo Fujiwara and Tomoo Inoue. Optimal Granularity and Scheme of Parallel Test Generation in a Distributed System. *IEEE Transactions on Parallel and Distributed Systems*, 6(7):677–686, 1995.

[3] Anshul Gupta and Vipin Kumar. Performance Properties of Large Scale Parallel Systems. *Journal of Parallel and Distributed Computing*, 19:234–244, 1993.

[4] Anshul Gupta and Vipin Kumar. The Scalability of FFT on Parallel Computers. *IEEE Transactions on Parallel and Distributed Systems*, 4(8):922–932, 1993.

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

[6] 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.

[7] Michael Heath and Patrick Worley. Once Again Amdahl's Law. *Communications of the ACM*, 32(2):262–263, 1989.

[8] C.K. Koc and S.C. Gan. Parallel Matrix Multiplication on Networked Microcomputers. *Computers and Electrical Engineering*, 18(2):145–152, 1992.

[9] Vipin Kumar, Ananth Grama, Anshul Gupta, and George Karypis. *Introduction to Parallel Computing Design and Analysis of Algorithms*. Benjamin/Cummings, Menlo Park, California, 1994.

[10] Vipin Kumar and Vineet Singh. Scalability of Parallel Algorithms for the All-pairs Shortest-Path Problem. *Journal of Parallel and Distributed Computing*, 13:124–138, 1991.

[11] David M. Nicol and Frank H. Willard. Problem Size, Parallel Architecture, and Optimal Speedup. *Journal of Parallel and Distributed Computing*, 13:404–420, 1988.

[12] P. Uthayopas, Models of Speedup and Scalability for Workstation Clusters, PhD Dissertation, Center for Advanced Computer Studies, University of Southwestern Louisiana.

[13] Xian-He Sun and Diane T. Rover. Scalability of Parallel Algorithm-Machine Combinations. *IEEE Transactions on Parallel and Distributed Systems*, 5(6):1185–1194, 1994.

[14] Xian-He Sun and Jianping Zhu. Performance Considerations of Shared Virtual Memory Machines. *IEEE Transactions on Parallel and Distributed Systems*, 6(11):1185–1194, 1995.

[15] Zhimin Tang and Guo-Jie Li. Optimal Granularity of Grid Iteration Problems. In Benjamin W. Wah, editor, *Proceedings of the International Conference on Parallel Processing 1990*, pages I111–I118, St. Charles, Illinois, 1990.

[16] Patrick H. Worley. The Effect of Time Constraints On Scaled Speedup. *SIAM Journal on Scientific and Statistical Computing*, 11(5):838–858, 1990.

[17] Xiaofeng Zhou. Bridging the Gap between Amdahl's Law and Sandia Laboratory's Result. *Communications of the ACM*, 32(8):1014–1016, 1989.