JOURNAL OF PARALLEL AND DISTRIBUTED COMPUTING 19, 27–37 (1993)

Scalable Problems and Memory-Bounded Speedup*

XIAN-HE SUN

ICASE, NASA Langley Research Center, Hampton, Virginia 23681-0001

AND

LIONEL M. NI

Computer Science Department, Michigan State University, East Lansing, Michigan 48824-1027

In this paper three models of parallel speedup are studied. They are *fixed-size speedup*, *fixed-time speedup*, and *memory-bounded speedup*. The latter two consider the relationship between speedup and problem scalability. Two sets of speedup formulations are derived for these three models. One set considers uneven workload allocation and communication overhead and gives more accurate estimation. Another set considers a simplified case and provides a clear picture on the impact of the sequential portion of an application on the possible performance gain from parallel processing. The simplified fixed-size speedup is *Amdahl's law*. The simplified fixed-time speedup is *Gustafson's scaled speedup*. The simplified memory-bounded speedup contains both Amdahl's law and Gustafson's scaled speedup as special cases. This study leads to a better understanding of parallel processing. © 1993 Academic Press, Inc.

1. INTRODUCTION

Although parallel processing has become a common approach for achieving high performance, there is no well-established metric to measure the performance gain of parallel processing. The most commonly used performance metric for parallel processing is *speedup*, which gives the performance gain of parallel processing versus sequential processing. Traditionally, speedup is defined as the ratio of uniprocessor execution time to execution time on a parallel processor. There are different ways to define the metric "execution time." In *fixed-size speedup*, the amount of work to be executed is independent of the number of processors. Based on this model, Ware [17] summarized Amdahl's [1] arguments to define a speedup formula which is known as *Amdahl's law*. However, in many applications, the amount of work to be performed increases (as the number of processors increases) in order to obtain a more accurate or better result. The concept of *scaled speedup* was proposed by Gustafson *et al.* at Sandia National Laboratory [6]. Based on this concept, Gustafson suggested a *fixed-time speedup* [5], which fixes the execution time and is interested in how the problem size can be scaled up. In scaled speedup, both sequential and parallel execution times are measured based on the same amount of work defined by the scaled problem.

Both Amdahl's law and Gustafson's scaled speedup use a single parameter, the sequential portion of a parallel algorithm, to characterize an application. They are simple and give much insight into the potential degradation of parallelism as more processors become available. Amdahl's law has a fixed problem size and is intended in how small the response time could be. It suggests that massively parallel processing may not gain high speedup. Gustafson [5] approaches the problem from another point of view. He fixes the response time and is interested in how large a problem could be solved within this time. This paper further investigates the scalability of problems. While Gustafson's scalable problems are constrained by the execution time, the capacity of main memory is also a critical metric. For parallel computers, especially for distributed-memory multiprocessors, the size of scalable problems is often determined by the memory available. Shortage of memory is paid for in problem solution time (due to the I/O or message-passing delays) and in programmer time (due to the additional coding required to multiplex the distributed memory) [3]. For many applications, the amount of memory is an important constraint to scaling problem size [6, 10]. Thus, *memory-bounded speedup* is the major focus of this paper.

We first study three models of speedup: *fixed-size speedup*, *fixed-time speedup*, and *memory-bounded speedup*. With both uneven workload allocation and communication overhead considered, speedup formulations will be derived for all three models. When communication overhead is not considered and the workload only

* This research was supported in part by the NSF Grant ECS-88-14027 and by the National Aeronautics and Space Administration under NASA Contract NAS1-18605.

0743-7315/93 $5.00

consists of sequential and perfectly parallel portions, the simplified fixed-size speedup is Amdahl's law; the simplified fixed-time speedup is Gustafson's scaled speedup; and the simplified memory-bounded speedup contains both Amdahl's law and Gustafson's speedup as special cases. Therefore, the three models of speedup, which represent different points of view, are unified.

Based on the concept of scaled speedup, intensive research has been conducted in recent years in the area of performance evaluation. Some other definitions of speedup have also been proposed, such as generalized speedup, cost-related speedup, and superlinear speedup. Interested readers can refer to [14, 9, 16, 7, 18, 2, 8] for details.

This paper is organized as follows. In Section 2 we introduce the program model and some basic terminologies. More generalized speedup formulations for the three models of speedup are presented in Section 3. Speedup formulations for simplified cases are studied in Section 4. The influence of communication/memory tradeoff is studied in Section 5. Conclusions and comments are given in Section 6.

2. A MODEL OF PARALLEL SPEEDUP

To measure different speedup metrics for scalable problems, the underlying machine is assumed to be a *scalable multiprocessor*. A multiprocessor is considered scalable if, as the number of processors increase, the memory capacity and network bandwidth also increase. Furthermore, all processors are assumed to be homogeneous. Most distributed-memory multiprocessors and multicomputers, such as commercial hypercube and mesh-connected computers, are scalable multiprocessors. Both message-passing and shared-memory programming paradigms have been used in such multiprocessors. To simplify the discussion, our study assumes homogeneous distributed-memory architectures.

The parallelism in an application can be characterized in different ways for different purposes [15]. For simplicity, speedup formulations generally use very few parameters and consider very high level characterizations of the parallelism. We consider two main degradations of parallelism, *uneven allocation* (load imbalance) and *communication latency*. The former degradation is application dependent. The latter degradation depends on both the application and the parallel computer under consideration. To obtain an accurate estimate, both degradations need to be considered. Uneven allocation is measured by *degree of parallelism*.

DEFINITION 1. The *degree of parallelism* of a program is an integer which indicates the maximum number of processors that can be busy computing at a particular instant in time, given an unbounded number of available processors.

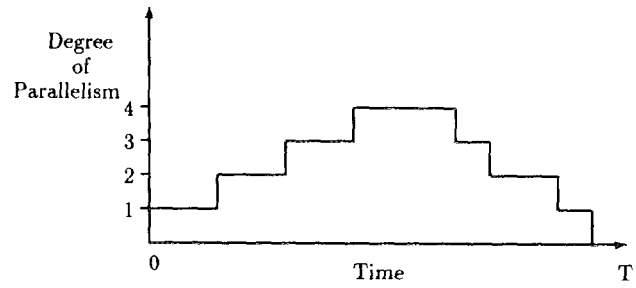


FIG. 1. Parallelism profile of an application.

The degree of parallelism is a function of time. By drawing the degree of parallelism over the execution time of an application, a graph can be obtained. We refer to this graph as the *parallelism profile*. Figure 1 is the parallelism profile of a hypothetical divide-and-conquer computation [13]. By accumulating the time spent at each degree of parallelism, the profile can be rearranged to form the *shape* (see Figure 2) of the application [12].

Let W be the amount of work of an application. Work can be defined as arithmetic operations, instructions, or whatever is needed to complete the application. Formally, the speedup with N processors and with the total amount of work W is defined as

$$S_N(W) = \frac{T_1(W)}{T_N(W)}, \tag{1}$$

where $T_i(W)$ is the time required to complete W amount of work on i processors. Let W_i be the amount of work executed with degree of parallelism i, and let m be the maximum degree of parallelism. Thus, $W = \sum_{i=1}^{m} W_i$. Assuming each computation takes a constant time to finish on a given processor, the execution time for computing W_i with a single processor is

$$t_1(W_i) = \frac{W_i}{\Delta}, \tag{2}$$

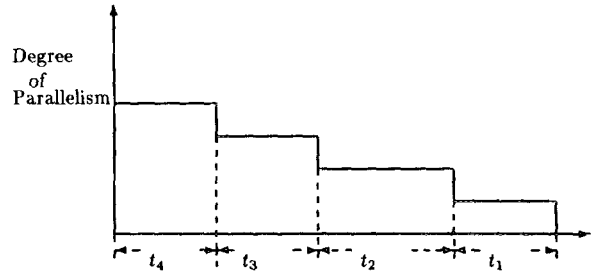


FIG. 2. Shape of the application.

where Δ is the computing capacity of each processor. If there are i processors available, the execution time is

$$t_i(W_i) = \frac{W_i}{i\Delta}.$$

With an infinite number of processors available, the execution time will not be further decreased and is

$$t_\infty(W_i) = \frac{W_i}{i\Delta} \quad \text{for } 1 \le i \le m.$$

Therefore, without considering communication latency, the execution times on a single processor and on an infinite number of processors are

$$T_1(W) = \sum_{i=1}^{m} \frac{W_i}{\Delta}, \tag{3}$$

$$T_\infty(W) = \sum_{i=1}^{m} \frac{W_i}{i\Delta}. \tag{4}$$

The maximum speedup, with work W and an infinite number of processors, is

$$S_\infty(W) = \frac{T_1(W)}{T_\infty(W)} = \frac{\sum_{i=1}^{m} \frac{W_i}{\Delta}}{\sum_{i=1}^{m} \frac{W_i}{i\Delta}} = \frac{\sum_{i=1}^{m} W_i}{\sum_{i=1}^{m} W_i/i}. \tag{5}$$

Average parallelism is an important factor for speedup and efficiency. It has been carefully examined in [4]. Average parallelism is equivalent to the maximum speedup S_∞ [4, 15]. S_∞ gives the best possible speedup based on the inherent parallelism of an algorithm. There are no machine dependent factors considered. With only a limited number of available processors and with communication latency considered, the speedup will be less than the best speedup, $S_\infty(W)$. If there are N processors available and $N < i$, then some processors have to do $W_i \lceil i/N \rceil / i$ work and the rest of the processors will do $W_i \lfloor i/N \rfloor / i$ work. By the definition of degree of parallelism, W_i and W_j cannot be executed simultaneously for $i \neq j$. Thus, the elapsed time will be

$$t_N(W_i) = \frac{W_i}{i\Delta} \left\lceil \frac{i}{N} \right\rceil.$$

Hence,

$$T_N(W) = \sum_{i=1}^{m} \frac{W_i}{i\Delta} \left\lceil \frac{i}{N} \right\rceil, \tag{6}$$

and the speedup is

$$S_N(W) = \frac{T_1(W)}{T_N(W)} = \frac{\sum_{i=1}^{m} W_i}{\sum_{i=1}^{m} \frac{W_i}{i} \left\lceil \frac{i}{N} \right\rceil}. \tag{7}$$

Communication latency is another degradation factor of performance. Unlike degree of parallelism, communication latency is machine dependent. It depends on the communication network topology, the routing scheme, the adopted switching technique, and the dynamics of the network traffic. Let $Q_N(W)$ be the communication overhead when N processors are used to complete W amount of work. The actual formulation for $Q_N(W)$ is difficult to derive as it is dependent on the communication pattern and the message sizes of the algorithm itself as well as the system-dependent communication latency. Note that $Q_N(W)$ is encountered when there are N processors ($N > 1$). Assuming that the degree of parallelism does not change due to communication overhead, the speedup becomes

$$S_N(W) = \frac{T_1(W)}{T_N(W)} = \frac{\sum_{i=1}^{m} W_i}{\left(\sum_{i=1}^{m} \frac{W_i}{i} \left\lceil \frac{i}{N} \right\rceil\right) + Q_N(W)}. \tag{8}$$

3. SPEEDUP OF SCALED PROBLEMS

In the last section we developed a general speedup formula and showed how the number of processors and degradation parameters influence the performance. However, speedup is not dependent only on these parameters. It is also dependent on how we view the problem. With different points of view, we get different models of speedup and different speedup formulations. One viewpoint emphasizes shortening the time it takes to solve a problem by parallel processing. With more and more computation power available, the problem can, in principle, be solved in less and less time. With more processors available, the system will provide a fast turnaround time and the user will have a shorter waiting time. A speedup formulation based on this philosophy is called *fixed-size speedup*. In the previous section, we implicitly adopted fixed-size speedup. Equation (8) is the speedup formula for fixed-size speedup. Fixed-size speedup is suitable for many algorithms in which the problem size cannot be scaled.

For some applications we may have a time limitation, but we may not want to obtain the solution in the shortest possible time. If we have more computation power, we may want to increase the problem size, carry out more operations, and get a more accurate solution. Various finite difference and finite element algorithms for the so-

lution of Partial Differential Equations (PDEs) are typical examples of such scalable problems.

An important issue in scalable problems is the identification of *scalable constraints*. One scalable constraint is to keep the execution time unchanged with respect to uniprocessor execution time. This viewpoint leads to a different model of speedup, called *fixed-time speedup*. For fixed-time speedup the workload is scaled up with the number of processors available. Let $W' = \sum_{i=1}^{m'} W_i'$ be the total amount of scaled work, where W_i' is the amount of scaled work executed with degree of parallelism i, and m' be the maximum degree of parallelism of the scaled problem where N processors are available. Note that the maximum degree of parallelism can change as the problem is scaled. In order to keep the same turnaround time as the sequential version, the condition $T_1(W) = T_N(W')$ must be satisfied for W'. That is, the following scalable constraint must be satisfied,

$$\sum_{i=1}^{m} W_i = \sum_{i=1}^{m'} \frac{W_i'}{i} \left\lceil \frac{i}{N} \right\rceil + Q_N(W'). \tag{9}$$

Thus, the general speedup formula for fixed-time speedup is

$$S_N(W') = \frac{T_1(W')}{T_N(W')} = \frac{\sum_{i=1}^{m'} W_i'}{\sum_{i=1}^{m'} \frac{W_i'}{i} \left\lceil \frac{i}{N} \right\rceil + Q_N(W')} = \frac{\sum_{i=1}^{m'} W_i'}{\sum_{i=1}^{m} W_i}. \tag{10}$$

In many parallel computers, the memory size plays an important role in performance. Many large scale multiprocessors with local memory architecture do not support virtual memory due to insufficient I/O network bandwidth. When solving an application with one processor, the problem size is more often bounded by the memory limitation than by the execution time limitation. With more processors available, instead of keeping the execution time fixed, we may want to meet the memory size constraint. In other words, if you have adequate memory space and the scaled problem meets the time limit imposed by fixed-time speedup, will you further increase the problem size to yield an even better or more accurate solution? If the answer is yes, the appropriate model is *memory-bounded speedup*. Like fixed-time speedup, memory-bounded speedup is a scaled speedup. The problem size scales up with memory size. The difference is that in fixed-time speedup execution time is the limiting factor and in memory-bounded speedup memory size is the limiting factor.

With memory size considered as a factor of performance, the requirements of an algorithm consist of two parts. One is the computation requirement, which is the workload, and the other is the memory (capacity) requirement. For a given algorithm, these two requirements are related to each other, and the workload might be viewed as a function of the memory requirement. Let M represent the memory size of each processor. Let g be a function such that $W = g(M)$, or $M = g^{-1}(W)$, where g^{-1} is the inverse function of g. An example of function g and g^{-1} can be found in Section 5. In a homogeneous, scalable, parallel computer, the memory capacity on each node is fixed and the total memory available increases linearly with the number of processors available. If $W = \sum_{i=1}^{m} W_i$ is the workload for execution on a single processor, the maximum scaled workload with N processors, $W^* = \sum_{i=1}^{m^*} W_i^*$ must satisfy the following scalable constraint,

$$W^* = g(NM) = g(Ng^{-1}(W)), \tag{11}$$

where m^* is the maximum degree of parallelism of the scaled problem and g is determined by the algorithm. The memory limitation can be stated as: *the memory requirement for any active processor is less than or equal to* $M = g^{-1}(\sum_{i=1}^{m} W_i)$. Here the main point is that the memory occupied on each processor is limited. By considering the communication overhead, Eq. (12) is the general speedup formula for memory-bounded speedup.

$$S_N(W') = \frac{\sum_{i=1}^{m^*} W_i^*}{\sum_{i=1}^{m^*} \frac{W_i^*}{i} \left\lceil \frac{i}{N} \right\rceil + Q_N(W^*)}. \tag{12}$$

4. SIMPLIFIED MODELS OF SPEEDUP

The three general speedup formulations contain both uneven allocation and communication latency degradations. They give better upper bounds on the performance of parallel applications. On the other hand, these formulations are problem dependent and difficult to understand. They give detailed information for each application, but lose the global view of possible performance gains. In this section, we make some simplifying assumptions. We assume that the communication overhead is negligible; i.e., $Q_N = 0$, and the workload only contains two parts, a sequential part and a perfectly parallel part. That is, $W_i = 0$, for $i \neq 1$ and $i \neq N$. We also assume that the sequential part is independent of the system size; i.e., $W_1 = W_1'$ and W_1^*.

Under this simplified case, the general fixed-size speedup formulation (Eq. 8) becomes

$$S_N(W) = \frac{W_1 + W_N}{W_1 + \frac{W_N}{N}}. \tag{13}$$

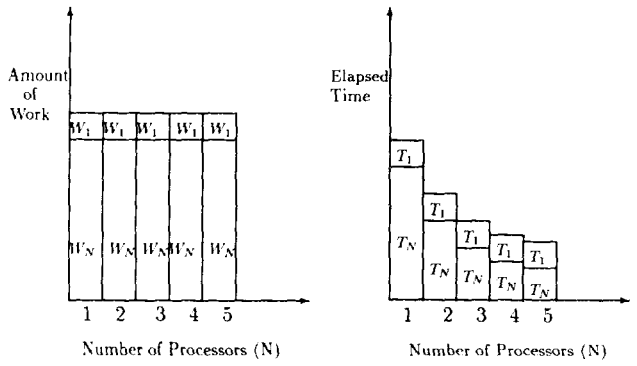


FIG. 3. Amdahl's law.

Equation (13) is known as Amdahl's law. Figure 3 shows that when the number of processors increases the load on each processor decreases. Eventually, the sequential part will dominate the performance and the speedup is bounded by $(W_1 + W_N)/W_1$. In Figure 3, T_1 is the execution time for the sequential portion of the work, and T_N is the execution time for the parallel portion of the work.

For fixed-time speedup and under the simplified conditions, the scalable constraint (Eq. 9) becomes

$$W_1 + W_N = W_1' + \frac{W_N'}{N}. \tag{14}$$

Since $W_1 = W_1'$, we have $W_N = W_N'/N$. That is, $W_N' = NW_N$. Eq. (10) becomes

$$S_N(W') = \frac{W_1 + NW_N}{W_1 + W_N}. \tag{15}$$

The simplified fixed-time speedup (Eq. 15) is known as Gustafson's scaled speedup [5]. From Eq. (15) we can see that the parallel portion of an application scales up linearly with the system size. The relation of workload and elapsed time for Gustafson's scaled speedup is depicted in Fig. 4.

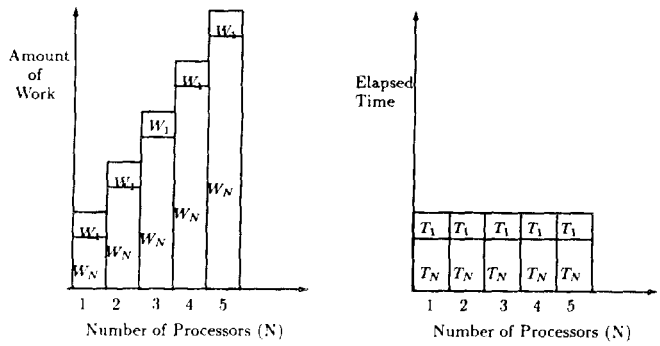


FIG. 4. Gustafson's scaled speedup.

We need some preparation before deriving the simplified formulation for memory-bounded speedup.

DEFINITION 2. A function g is a semihomomorphism if there exists a function $\bar{g}$ such that for any real number c and any variable x, $g(cx) = \bar{g}(c)g(x)$.

One class of semihomomorphism functions is the power function $g(x) = x^b$, where b is a rational number. In this case, $\bar{g}$ is the same as the function g. Another class of semihomomorphism functions is the single term polynomial $g(x) = ax^b$, where a is a real constant and b is a rational number. For this kind of semihomomorphism function, $\bar{g}(x) = x^b$, which is not the same as $g(x)$.

Under our assumptions, the sequential portion of the workload, W_1, is independent of the system size. If the influence of memory on the sequential portion is not considered; i.e., the memory capacity M is used for the parallel portion only, we have the following theorem.

THEOREM 1. *If $W = g(M)$ for some semihomomorphism function g, $g(cx) = \bar{g}(c)g(x)$, then, with all data being accessible by all available processors and using all available memory space, the simplified memory-bounded speedup is*

$$S_N(W^*) = \frac{W_1 + \bar{g}(N)W_N}{W_1 + \dfrac{\bar{g}(N)}{N} W_N}. \tag{16}$$

Proof. Assume that the maximum problem size will take the maximum available memory capacity of M when one processor is used. As mentioned before, when one processor is available, the parallel portion of the workload, W_N, can be expressed as $W_N = g(M)$. Since all data are accessible by all processors, there is no need to replicate the data. With N processors available, the total available memory capacity will be increased to NM. The parallel portion of the problem can be scaled up to use all available memory capacity NM. Thus, the scaled parallel portion, W_N^*, is expressed as $W_N^* = g(NM) = \bar{g}(N)g(M)$. Therefore, $W_N^* = \bar{g}(N)W_N$ and

$$S_N(W^*) = \frac{W_1^* + W_N^*}{W_1^* + W_N^*/N} = \frac{W_1 + \bar{g}(N)W_N}{W_1 + \dfrac{\bar{g}(N)}{N} W_N}. \quad \blacksquare \tag{17}$$

Note that in Theorem 1, we made two assumptions in the simplified case: (1) Since the communication latency is ignored, remote memory accesses take the same time as local memory accesses. This implies that the data is accessible by all available processors, and (2) all the available memory space is used for a better solution. These simplified speedup models are useful to demonstrate how the sequential portion of an application, W_1, will affect the maximum speedup that can be achieved

with different number of processors. Let $k = W_1/(W_1 + W_N)$. The simplified fixed-size speedup, fixed-time speedup, and memory-bounded speedup are, respectively,

$$S_N(W) = \frac{N}{1 + k(N - 1)}, \tag{18}$$

$$S_N(W') = N - k(N - 1) = k + N(1 - k), \tag{19}$$

and

$$S_N(W^*) = N\left(\frac{\bar{g}(N) + k(1 - \bar{g}(N))}{\bar{g}(N) + k(N - \bar{g}(N))}\right) \tag{20}$$

When the number of processors, N, goes to infinity, Eq. (18) is bounded by the reciprocal of k, which gives the maximum value of the fixed-size speedup. Equation (19) shows that the fixed-time speedup is a linear function of the number of processors with slope equal to $(1 - k)$. When N goes to infinity, this speedup can increase without bound. Memory-bounded speedup depends on the function $\bar{g}(N)$. When $\bar{g}(N) = 1$, memory-bounded speedup is the same as fixed-size speedup. When $\bar{g}(N) = N$, the memory-bounded speedup is the same as the fixed-time speedup. In general, the function $\bar{g}(N)$ is application dependent and $\bar{g}(N) \geq N$. It implies that when the problem size is increased by N, the amount of work increases more than N times. It is easy to verify that $S_N(W^*) > S_N(W')$ when $\bar{g}(N) > N$. Note that all data in memory is likely to be accessed at least once. Thus, for scaled problems, $\bar{g}(N) < N$ is unlikely to occur. The sequential portion of the work plays different roles in the three definitions of speedup. In fixed-size speedup, the influence of the sequential portion increases with system size and eventually dominates the performance. In fixed-time speedup, the influence of the sequential portion is unchanged which makes the speedup a linear function of system size. In the memory-bounded speedup, since in general $\bar{g}(N) > N$, the influence of the sequential portion is reduced when the system size increases, indicating that a better speedup could be achieved with a larger system size.

The function $\bar{g}(N)$ provides a metric to evaluate parallel algorithms. In general, $\bar{g}(N)$ may not be derivable for a given algorithm. Note that any single term polynomial is a semihomomorphism function, and most solvable algorithms have polynomial time computation and memory requirement. If we take an algorithm's computation and storage complexity (the term with the largest power) as its computation and memory requirement, for any algorithm with polynomial complexity there exists a semihomomorphism function g, such that $W = g(M)$. The approximated semihomomorphism function g will provide a good estimation on the memory-bounded speedup when the number of processors is large. More details case studies for the three models of speedup can be found in [13].

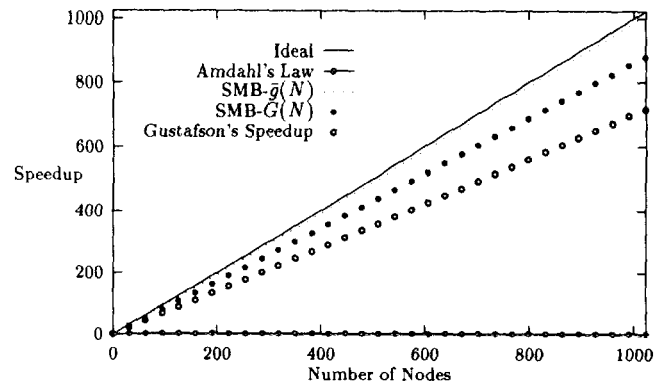


FIG. 5. Amdahl's law, Gustafson's speedup, and SMB speedup for $k = 0.3$.

Figure 5 demonstrates the difference between the three models of speedup when $k = 0.3$ and N ranges from 1 to 1024. For the simplified memory-bounded (SMB) speedup, we choose $\bar{g}(N) = N^{3/2}$, which is typical in many matrix operations to be described later. When $\bar{g}(N) = N$, it is Gustafson's scaled speedup. The case of $G(N) = (1 + \bar{g}[1 - \bar{g}^{-1}(N)/N])N$ will be studied in next section.

5. COMMUNICATION-MEMORY TRADEOFF

The simplified speedup formulations give the impact of the sequential portion of an application on the maximum speedup. The simplified memory-bounded speedup suggests that when data are shared by all processors, maximum speedup is obtained. However, in practice if communication overhead is considered, the data sharing approach may not lead to maximum speedup. In the design of efficient parallel algorithms, the communication cost plays an important role in deciding how a problem should be solved and scaled. One way to reduce the frequency of communication is to replicate some shared data to processors. Thus, a good algorithm design should consider the tradeoff between the maximum size that a problem can scale and the reduction of available memory due to the replication of shared data.

If data replication is allowed, the function $W = g(NM)$ will no longer hold. Motivated by Theorem 1, the function $G(N) = W_N^*/W_N$ is defined to represent the ratio of work increment when N processors are available. In terms of $G(N)$, the simplified memory-bounded speedup is generalized below.

THEOREM 2. *If W_1 is independent of system size, $W_i = 0$ for $1 < i < N$, and $W_N^* = G(N)W_N$ for some*

function $G(N)$, the memory-bounded speedup is

$$S_N(W^*) = \frac{W_1 + G(N)W_N}{W_1 + \frac{G(N)}{N} W_N + Q_N(W^*)}. \quad (21)$$

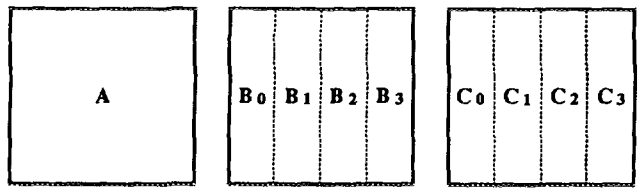
The proof of Theorem 2 is similar to the proof of Theorem 1. Equation (21) shows that the maximum speedup is not necessarily achieved when $G(N) = \bar{g}(N)$. Note that the communication cost $Q_N(W^*)$ is a unified communication cost. An optimal choice of the function $G(N)$ is both algorithm and architecture dependent and, in general, is difficult to obtain. Also, unlike $\bar{g}(N)$, $G(N)$ might be less than N. If $G(N) < N$, memory capacity is likely to be the scalable constraint when N is large. If $G(N) > N$, execution time is likely to be the scalable constraint. The function $G(N)$ indicates the possible scalable constraint of an algorithm. The proposed scaled speedup (Eq. 21) may not be easy to fully understand at first glance. Hence, we use matrix multiplication as an example to illustrate it.

A matrix often represents some discretized continuum. Enlarging the matrix size will generally lead to a more accurate solution for the continuum. For matrix multiplication $C = AB$, there are many ways to partition the matrices A and B to allow parallel processing [11]. Assume that there are N processors available, and A and B are $n \times n$ matrices when executing on a single processor. The computation requirement is $2n^3$ and the memory requirement is roughly $3n^2$. Thus, $W_N = 2n^3$ and $M = 2n^2$. Two extreme cases of memory-bounded scaled speedup are considered.

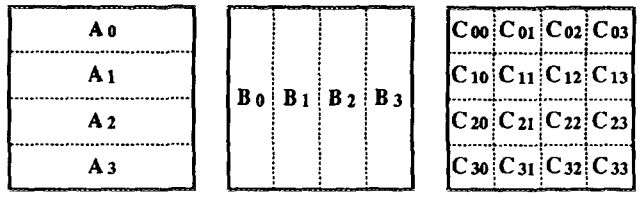
Local Computation

In the first case, we assume that the communication cost is extremely high. Thus, data should be replicated if possible to reduce communication. This can be achieved by partitioning the columns of matrix B into N submatrices, $B_0, B_1, \ldots, B_{N-1}$ and replicating the matrix A. Thus, B_i's are distributed among all the processors and matrix A is replicated on each processor. Processor i does the multiplication $AB_i = C_i$, $i = 0, \ldots, N-1$, independently. Since there is no need for communication, it is referred to as *local computation* approach. Figure 6a shows the partitioning of B for the case of $N = 4$.

If both A and B are allowed to scale along any dimension and A and B are not necessary to be square matrices, the enlarged problem is $A^*B^* = C^*$, where A^* is an $l \times k$ matrix, B^* is a $k \times m$ matrix, and the resulting matrix C^* is an $l \times m$ matrix. Note that the local memory capacity is $M = 3n^2$. It is easy to see that the maximum memory-bound speedup will be achieved when $l = k = n$, and $m = nN$. In other words, both B and C are scaled up N times along their rows, and A is replicated but not scaled. The amount of computation on each processor is fixed,



(a) The matrix B is partitioned.



(b) Both matrices A and B are partitioned.

FIG. 6. Two partitioning schemes of matrices A and B.

$W_N = 2n^3$, and $W_N^* = NW_N$. Thus, we have $G(N) = N$. The memory-bounded scaled speedup is

$$S_N(W^*) = \frac{W_1 + NW_N}{W_1 + W_N},$$

which is Gustafson's scaled speedup. Thus, the best performance of memory-bounded speedup using the local computation model is the same as the Gustafson's scaled speedup. In general, the local computation model will lead to a speedup that is less than Gustafson's scaled speedup. For example, if both A and B are restricted to square matrices, the function $G(N)$ will be

$$G(N) = \left(\sqrt{\frac{3N}{N+2}}\right)^3,$$

which is less than N and is bounded by $3^{3/2}$ (see Appendix). Note that due to data replication, the memory capacity requirement increases faster than the computation requirement does.

Global Computation

In the second extreme case, we assume that the communication cost is negligible. Thus, there is no need to replicate the data. A bigger problem can be solved. We partition matrix A into N row blocks and B into N column blocks (see Fig. 6b). By assigning each pair of submatrices, A_i and B_i, to one processor initially, all main diagonal blocks of C can be computed. Then, the row blocks of A are rotated from one processor to another after each row–column submatrix multiplication. With N processors, $N - 1$ rotations are needed to finish the computa-

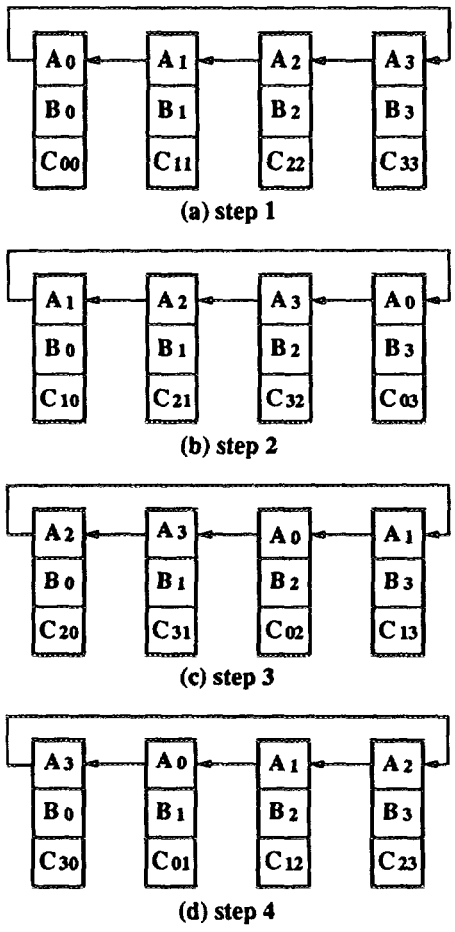


FIG. 7. Matrix multiplications without data replication.

tion as shown in Fig. 7 for the case of $N = 4$. This method is referred to as *global computations*.

For the global computation approach, the maximum scaled speedup is achieved when $l = k = m = n\sqrt{N}$ (see Appendix).

$$S_N(W^*) = \frac{W_1 + N^{3/2}W_N}{W_1 + N^{1/2}W_N}. \tag{22}$$

The corresponding function $G(N) = N^{3/2}$. Assuming $N \leq n^2$, we can write W_N as a function of M as follows,

$$W_N = g(M) = \left(\frac{2M}{3}\right)^{3/2}. \tag{23}$$

Increasing the total memory capacity to NM, we have

$$W_N^* = \left(\frac{2NM}{3}\right)^{3/2} = N^{3/2}\left(\frac{2M}{3}\right)^{3/2} = N^{3/2}W_N = \bar{g}(N)W_N. \tag{24}$$

The matrix multiplication problem has a semihomomorphism function between its memory requirement and computation requirement and $\bar{g}(N) = N^{3/2}$. Assuming a negligible communication cost, the global computation approach will achieve the best possible scaled speedup of the matrix multiplication problem.

We have studied two extreme cases of memory-bounded scaled speedup which are based on global computation and local computation. In general for most of the algorithms, part of the data may be replicated and part of the data may have to be shared. Deriving a speedup formulation for these algorithms is difficult, not only because we are facing a more complicated situation, but also because the ratio between replicated and shared data is uncertain. The replicated part may not increase as the system size is increased. In case the replicated part does increase, its speed of increase may be different from the speed that the shared part is increased. Also, an algorithm may start with global computation. When the system size is increased, replication may be needed as part of the effort to reduce communication overhead. A special combined case, $G(N) = (1 + \bar{g}[1 - \bar{g}^{-1}(N)/N])N$, has been carefully studied in [15]. The structure of that study can be used as a guideline for other algorithms.

The influence of communication overhead on the best performance of the memory-bounded speedup is studied. The study can be extended to fixed-time speedup, where redundant computation could be introduced to reduce the communication overhead. The function $G(N)$ determines the actual achieved speedup. We have shown how the partition and scale of the problem will influence the function $G(N)$. In general, finding an optimal function $G(N)$ is a nonlinear optimization problem.

6. CONCLUSIONS

It is known that the performance of parallel processing is influenced by the inherent parallelism and communication requirement of the algorithm, by the computation and communication power of the underlying architecture, and by the memory capacity of the parallel computer system. However, how these factors related to each other and how they influence the performance of parallel processing is generally unknown. Discovering the answers to these unknowns is important for designing efficient parallel algorithms. In this paper one model of speedup, *memory-bounded speedup*, is carefully studied. The model contains these factors as its parameters.

As part of the study on performance, two other models of speedup have also been studied. They are *fixed-size speedup* and *fixed-time speedup*. Two sets of speedup formulations have been derived for these two models of speedup and for memory-bounded speedup. Formulations in the first set give rise to generalized speedup formulas. The second set of formulations only considers a

special, simplified case. The simplified fixed-size speedup is *Amdahl's law*, the simplified fixed-time speedup is *Gustafson's scaled speedup*, and the simplified memory-bounded speedup contains both Amdahl's law and Gustafson's scaled speedup as special cases.

The three models of speedup, *fixed-size speedup*, *fixed-time speedup*, and *memory-bounded speedup*, are based on different viewpoints and are suitable for different classes of algorithms. However, algorithms exist which do not fit any of the models of speedup, but satisfy some combination of the models.

APPENDIX

When communication does not occur (local computation) or its cost is negligible, the memory-bounded speedup Eq. (21) becomes

$$S_N^* = \frac{W_1 + G(N)W_N}{W_1 + \frac{G(N)}{N} W_N}. \tag{25}$$

It is easy to verify that S_N^* increases with the function $G(N)$. Thus, for the two extreme cases considered in Section 5, the problem of how to reach the maximum speedup becomes how to scale the matrix A and B such that the function $G(N)$ reaches its maximum value. The matrix A and B can be scaled in any dimension. A general enlarged matrix multiplication problem is

$$A_{l*k} B_{k*m} = C_{l*m},$$

where both A and B are rectangular matrices. To achieve an optimal speedup, we need to decide the integers l, k, and m, for which that the function $G(N)$ reaches the maximum value. The following result gives the optimal l, k, and m for the global computation approach (Fig. 6b) given in Section 5. Recall that N is the number of processors.

PROPOSITION 1. *If A and B are $n \times n$ matrices when $N = 1$, then the global computation approach reaches the maximum $G(N)$ when $l = k = n$ and $m = n \times \sqrt{N}$, excluding the communication cost. The corresponding $G(N)$ equals $N^{3/2}$, and the maximum speedup is*

$$S_N^* = \frac{W_1 + N^{3/2} W_N}{W_1 + N^{1/2} W_N}. \tag{26}$$

Proof. By the partition schema of the global computation approach, the rows of matrix A and the columns of matrix B are distributed processors. The workload on each processor is

$$A_{(l/N)*k} B_{k*(m/N)} = C_{(l/N)*(m/N)}.$$

Since the memory is fully filled,

$$\frac{l}{N} * k + k * \frac{m}{N} + \frac{l}{N} * \frac{m}{N} = 3n^2.$$

Thus,

$$k = \frac{3n^2 - \frac{l}{N} * \frac{m}{N}}{\frac{l}{N} + \frac{m}{N}}. \tag{27}$$

The work of the scaled problem is

$$W_N^* = 2 * l * m * k = 2 * l * m * \left[\frac{3n^2 N - l * m}{l + m}\right]$$

$$= 2n^3 \frac{l * m}{n^2} \frac{3n^2 N - l * m}{l + m}$$

$$= \frac{(3n^2 - N - l * m)(l * m)}{(l + m) * n^3} W_N.$$

$$G(N) = \frac{(3n^2 - N - l * m)(l * m)}{(l + m) * n^3}. \tag{28}$$

Therefore, $G(N)$ reaches its maximum value if and only if the function

$$f(l, m) = \frac{(3n^2 - N - l * m)(l * m)}{l + m}$$

reaches its maximum value. At its maximum value, the derivatives of $f(l, m)$ satisfy

$$f_l' = -l^2 m^2 - 2lm^2 + 3n^2 m^2 N = 0,$$

$$f_m' = -l^2 m^2 - 2ml^2 + 3n^2 m^2 N = 0.$$

It leads to

$$l^2 + 2lm - 3n^2 N = 0, \tag{29}$$

$$m^2 + 2lm - 3n^2 N = 0. \tag{30}$$

This is

$$(l + m)^2 = m^2 + 3n^2 N.$$

$$(m + l)^2 = l^2 + 3n^2 N.$$

Thus, we have $m^2 = l^2$, i.e.

$$l = m \tag{31}$$

Combining the Eq. (31) and Eq. (29), we get

$$l = m = n\sqrt{N}.$$

From the Eq. (27), we have $k = n\sqrt{N}$. Thus, the enlarged A and B are still square matrices, with dimension $n\sqrt{N}$. By Eq. (28) the maximum $G(N)$ is

$$G(N) = \frac{(n\sqrt{N})^2(3n^2N - (n\sqrt{N})^2)}{n^3(2n\sqrt{N})} = N^{3/2},$$

which is equal to the memory-work function $\bar{g}(N)$ for the matrix multiplication problem (see Section 5), and the corresponding speedup is

$$S_N^* = \frac{W_1 + N^{3/2}W_N}{W_1 + N^{1/2}W_N}.$$

From Theorem 1, it is the best possible performance for the matrix multiplication problem. ■

Using similar arguments as in Proposition 1, we can find that the optimal dimension of the local computation approach is $l = k = n$, $m = nN$, and the maximum value of $G(N)$ is N (see Section 5). The scalability of matrix A and B is application dependent. If A and B should be maintained as square matrices, the following proposition shows the limitation of the local computation approach.

PROPOSITION 2. *If A and B are $n \times n$ matrices when $N = 1$, and $l = k = m$ is required, then the maximum value of $G(N)$ of the local computation approach is $(\sqrt{(3N/N + 2)})^3$, which is bounded by $3^{3/2}$ and is smaller than N.*

Proof. When A and B are square matrices, the scaled problem is

$$A_{k*k}B_{k*k} = C_{k*k}.$$

If the load is balanced on each processor, and $m = k/N$ is an integer, then each processor does the work

$$A_{k*k}B_{k*m} = C_{k*m}.$$

When memory is fully used,

$$k^2 + 2k * m = 3n^2.$$

Since $m = k/N$,

$$k^2 + \frac{2k^2}{N} = 3n^2.$$

Thus,

$$k = \frac{3n^2}{1 + \frac{2}{N}} = \left(\sqrt{\frac{3N}{N + 2}}\right) n.$$

The scaled work

$$W_N^* = 2k^3 = \left(\sqrt{\frac{3N}{N + 2}}\right)^3 2n^3 = \left(\sqrt{\frac{3N}{N + 2}}\right)^3 W_N,$$

and

$$G(N) = \left(\sqrt{\frac{3N}{N + 2}}\right)^3.$$

Since

$$\frac{3N}{N + 2} = \frac{3N + 6}{N + 2} - \frac{6}{N + 2} = 3 - \frac{6}{N + 2},$$

the $G(N)$ is bounded by $3^{3/2}$ and is smaller than N. ■

ACKNOWLEDGMENTS

The authors would like to thank the referees for their valuable comments and suggestions which significantly improve the presentation of the paper.

REFERENCES

1. Amdahl, G. Validity of the single-processor approach to achieving large scale computing capabilities. *Proc. AFIPS Conference* 1967, pp. 483–485.
2. Barton, M., and Withers, G. Computing performance as a function of the speed, quantity, and cost of the processors. *Proc. Supercomputing '89.* 1989, pp. 759–764.
3. Dunigan, T. Performance of the Intel iPSC/860 and nCUBE 6400 hypercubes. *Parallel Computing* (Dec. 1991), 1285–1302.
4. Eager, D., Zahorjan, J., and Lazowska, E. Speedup versus efficiency in parallel system. *IEEE Trans. Comput.* (March 1989), 403–423.
5. Gustafson, J. Reevaluating Amdahl's law. *CACM* **31** (May 1988), 532–533.
6. Gustafson, J., Montry, G., and Benner, R. Development of parallel methods for a 1024-processor hypercube. *SIAM J. SSTC* **9**, 4 (July 1988).
7. Gustafson, J., Rover, D., Elbert, S., and Carter, M. The design of a scalable, fixed-time computer benchmark. *J. Parallel Distrib. Comput.* **11**, (Aug. 1991), 338–401.
8. Karp, A. H., and Flatt, H. P. Measuring parallel processor performance. *CACM* **33**, 5 (May 1990), 539–543.
9. Kumar, V., and Gupta, A. Analysis of scalability of parallel algorithms and architectures: A survey. *Proc. 1991 International Conference on Supercomputing.* June 1991.
10. Moler, C. Matrix computation on distributed memory multiprocessors. *Proc. First Conference on Hypercube Multiprocessors.* 1986, pp. 181–195.
11. Ni, L. M., and King, C.-T. On partitioning and mapping for hypercube computing. *Internat. J. Parallel Programming* **17**, (1988), 475–495.
12. Sevcik, K. Characterizations of parallelism in applications and their use in scheduling. *Proc. ACM SIGMETRICS and Performance '89.* May 1989.

13. Sun, X.-H. Parallel computation models: Representation, analysis and applications. Ph.D. dissertation, Computer Science Department, Michigan State University, 1990.
14. Sun, X.-H., and Gustafson, J. Toward a better parallel performance metric. *Parallel Comput.* **17,** (Dec 1991), 1093–1109.
15. Sun, X.-H., and Ni, L. M. Another view on parallel speedup. *Proc. of Supercomputing '90* New York, NY, 1990, pp. 324–333.
16. Sun, X.-H., and Rover, D. Scalability of parallel algorithm-machine combination. *IEEE Trans. Parallel Distrib. Systems*, to appear.
17. Ware, W. The ultimate computer. *IEEE Spectrum* **9,** (1972), 84–91.
18. Worley, P. H. The effect of time constraints on scaled speedup. *SIAM J. Sci. Stat. Comput.* **11,** 5 (Sept. 1990), 838–858.

XIAN-HE SUN received the B.S. degree in mathematics from Beijing Normal University, Beijing, China, in 1982 and the M.S. degree in mathematics and Ph.D. degree in computer science from Michigan State University, East Lansing, in 1985 and 1990, respectively. From September 1990 to August 1991, he was with the Ames Laboratory, Iowa State University and Department of Energy. During the year 1991–1992, he was a visiting assistant professor at Clemson University. He is currently a staff scientist in ICASE, NASA Langley Research Center. His research interests include parallel processing, performance evaluation, parallel numerical algorithms, and database systems. Dr. Sun is a member of the IEEE Computer Society and the Association for Computing Machinery, and is co-editor for the April 1994 issue of *Journal of Parallel and Distributed Computing* on Scalability of Parallel Algorithms and Architectures.

Received August 8, 1991; revised April 7, 1992; accepted September 21, 1992

LIONEL M. NI received the B.S. degree in electrical engineering from National Taiwan University in 1973, the M.S. degree in electrical and computer engineering from Wayne State University, Detroit, MI, in 1977, and the Ph.D. degree in electrical engineering from Purdue University, West Lafayette, IN, in 1980. In 1981 he joined the faculty of the Department of Computer Science, Michigan State University, East Lansing, where he is currently a professor and director of the Advanced Computer Systems Laboratory. During the summers of 1979 and 1981, he was a visiting researcher at the IBM San Jose Research Laboratories. During the year of 1987 to 1988, he was a visiting scientist in the Division of Mathematics and Computer Science at Argonne National Laboratory. His current research interests include computer architecture, parallel processing, and distributed computing. Dr. Ni is serving as a member of the editorial boards of *Journal of Parallel and Distributed Computing* and *IEEE Transactions on Computers*. He served as a distinguished visitor of the IEEE Computer Society from 1985 to 1988. He is a senior member of IEEE and a member of the Association for Computing Machinery and SIAM. He served in program committees of many conferences and was the Program Chairman of the Fifteenth IEEE Annual International Computer Software and Applications Conference. He received the Outstanding Paper Award at the 1992 International Conference on Parallel Processing.