ANOTHER VIEW ON PARALLEL SPEEDUP*

Xian-He Sun and *Lionel M. Ni*

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

Abstract: In this paper three models of parallel speedup are studied. They are *fixed-size speedup*, *fixed-time speedup* and *memory-bounded speedup*. Two sets of speedup formulations are derived for these three models. One set requires more information and gives more accurate estimation. Another set considers a simplified case and provides a clear picture of possible performance gain of 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 its special cases. This study proposes a new metric for performance evaluation and leads to a better understanding of parallel processing.

1 Introduction

Although parallel processing has become a common approach for achieving high performance, performance evaluation techniques of parallel processing are weak. There is no well established metric to measure the performance gain. This evaluation weakness leads to confusion and limits the healthy growth of parallel computation. The frequently used metrics include elapsed time and *parallel speedup*. A new metric, experimentally determined serial fraction, was recently proposed to reveal aspects of the performance that are not easily discerned from the other metrics [1]. In this paper, we will study issues of performance and give a better understanding of one important metric, *parallel speedup*.

The most commonly used performance metric for parallel processing is *speedup*, which gives the performance gain of parallel processing versus sequential processing. However, with different emphases, the speedup has been defined differently. One definition focuses on how much faster a problem can be solved with N processors. Thus, it compares the best sequential algorithm with the parallel algorithm under consideration. This definition is referred as *absolute speedup*. Absolute speedup has two different definitions depending on considering or not considering the machine resources. In the case of machine dependent, this speedup is defined as the ratio of the elapsed time of the best sequential algorithm on one processor over the elapsed time of the parallel algorithm on N processors. In the case of machine independent, the absolute speedup is defined as the ratio of the elapsed time of the best sequential algorithm on the fastest sequential machine over the elapsed time of the parallel algorithm on the parallel machine [2]. Another speedup, called *relative speedup*, deals with the inherent parallelism of the parallel algorithm under consideration. It is defined as the ratio of elapsed time of the parallel algorithm on one processor to elapsed time of the parallel algorithm on N processors. The reason for using relative speedup is that the performance of parallel algorithms varies with the number of available processors. Comparing the algorithm itself with different number of processors gives information on the variations and the degradations of parallelism. Absolute speedup and relative speedup are two commonly used speedup measures. Other definitions of speedup also exist. For instance, considering the constructing cost of processors, a cost-related speedup is defined in [3].

Among all of the defined speedups, the relative speedup probably is the one which has had the most influence on parallel processing. Two well known speedup formulations have beed proposed based on relative speedup. One is *Amdahl's law* [4] and another is *Gustafson's scaled speedup* [5]. Both of these two speedup formulations use a single parameter, the sequential portion of a parallel algorithm. They are simple and give much insight into the degradations of parallelism. Amdahl's law has a fixed problem

*This research was supported in part by the NFS grant ECS-88-14027

CH2916-5/90/0000/0324/$01.00 © IEEE

size and is interested in how fast the response time could be. It suggests that massively parallel processing may not gain high speedup. Under the influence of Amdahl's law many parallel computers have been built with a small number of processors. Gustafson [5] approachs 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. The argument of Gustafson is that the problem size should be increased to meet the available computation power for better results. Experimental results show that the speedup could increase linearly with the number of processors available based on his argument [6].

In this paper we will give a careful study on relative speedup. We first study three models of speedup, *fixed-size speedup*, *fixed-time speedup*, and *memory-bounded speedup*. All of the three models are based on relative speedup. With both uneven allocation and communication overhead considered, general speedup formulations have been derived for all of the three models. When the communication overhead is not considered and the workload only 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, with one more parameter, the simplified memory-bounded speedup contains both Amdahl's law and Gustafson's speedup as its special cases. Therefore, from different point of views, the three models of speedup are unified.

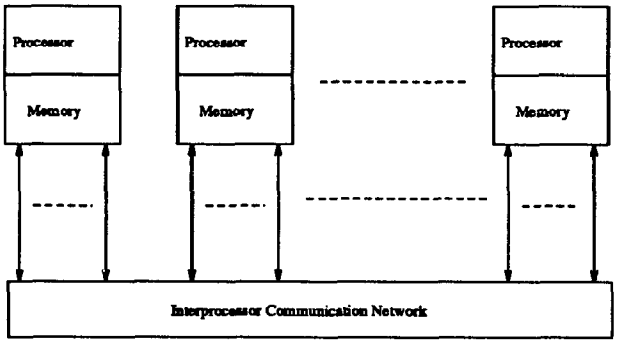

Figure 1. A Generic Multicomputer Architecture

To simplify the discussion, the parallel systems considered in this paper are *multicomputers*. Multicomputers are distributed-memory multiprocessors. They are organized as an ensemble of individual programmable computers, called *nodes*, and communicate through an interprocessor communication network. The memory is distributed and associated with each node. When the number of processors increases, the memory capacity also increases. Some of the most powerful and large scale parallel computing systems such as NCUBE multicomputer, Intel's iPSC series, and Ametek's 2010 are all belong to the class of multicomputers. A generic architecture of multicomputers is depicted in Figure 1.

This paper is organized as follows. In Section 2 we will introduce some preliminary knowledge and terminologies. General speedup formulations of the three models of speedup will be presented in Section 3. Speedup formulations for simplified cases are studied in Section 4. Conclusion and comments are given in Section 5.

2 Preliminary

From *data dependency graph* [7] to *task precedence graph* [8], from *Petri Net* [9] to *average parallelism* [10], the parallelism in an application can be characterized in different ways for different purposes. For simplicity, speedup formulations generally use very few parameters and consider very high level characterizations of the parallelism. In our study we consider two main degradations of parallelism, *uneven allocation* and *communication latency*. The former degradation is application dependent. The later degradation depends on both the application and the parallel computer under consideration. To give an accurate estimate, both of the degradations need to be considered. Uneven allocation is measured by *degree of parallelism*.

Definition 1 *The degree of parallelism is an integer which indicate the number of processors that are busy during the execution of the program in question, given an unbounded number of available processors.*

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*. Some software tools are available to determine the parallelism profile of large scientific and engineering applications [11]. Figure 2 is the parallelism profile of a hypothetical divide-and-conquer computation [12]. By accumulating the time spent at each degree of parallelism, the profile can be rearranged to form the *shape* of the application [13].

Definition 2 *The average parallelism is the average number of processors that are busy during the execution of the program in question, given an unbounded number of available processors.*

By definition, average parallelism is the ratio of the total service demand to the execution time with

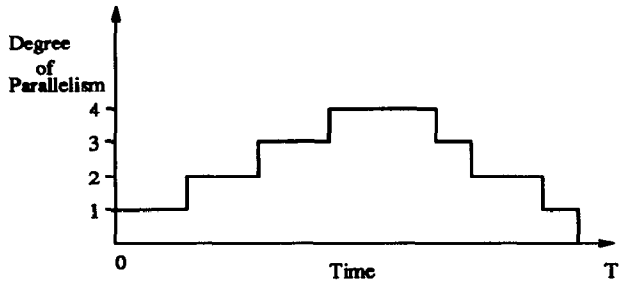

Figure 2. Parallelism Profile of an Application

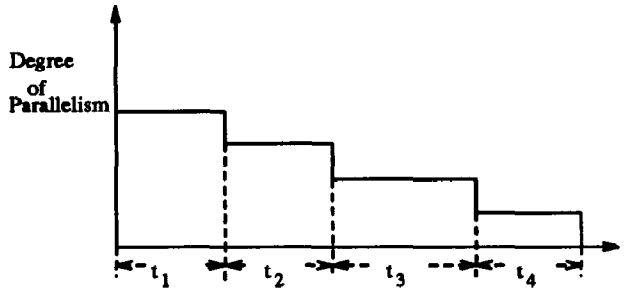

Figure 3. Shape of the Application

an unbounded number of available processors. This is equal to the speedup, given unbounded number of available processors and without considering the communication latency. Therefore, average parallelism can be defined equivalently as follows [10].

Definition 3 *Given an unbounded number of available processors and without considering the communication latency, the average parallelism is same as the speedup,*

Let W be the amount of work (computation) of an application. Let W_i be the amount of work executed with degree of parallelism i, and m be the maximum degree of parallelism. Thus, $W = \sum_{i=1}^{m} W_i$. The execution time for computing W_i with a single processor will be

$$t_i(1) = \frac{W_i}{\Delta},$$

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

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

With an infinite number of processors available, the execution time will be

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

Therefore, without considering communication latency, the response time on a single processor and on infinite number of processors will be

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

$$T(\infty) = \sum_{i=1}^{m} \frac{W_i}{i\Delta}. \tag{2}$$

And the speedup will be

$$S_\infty = \frac{T(1)}{T(\infty)} = \frac{\sum_{i=1}^{m} \frac{W_i}{\Delta}}{\sum_{i=1}^{m} \frac{W_i}{i\Delta}} = \frac{\sum_{i=1}^{m} W_i}{\sum_{i=1}^{m} W_i/i.} \tag{3}$$

The average parallelism, A, can be computed in terms of t_i,

$$A = \sum_{i=1}^{m} \frac{it_i}{\sum_{i=1}^{m} t_i} = \frac{\sum_{i=1}^{m} it_i}{\sum_{i=1}^{m} t_i.} \tag{4}$$

Notice that t_i is the time for executing W_i when an unbounded number of processors are available, $t_i = \frac{W_i}{i\Delta}$. Substituting $t_i = \frac{W_i}{i\Delta}$ into Eq. (3), we have

$$A = \frac{\sum_{i=1}^{m} it_i}{\sum_{i=1}^{m} t_i} = \frac{\sum_{i=1}^{m} W_i}{\sum_{i}^{m} W_i/i} = S_\infty. \tag{5}$$

This gives a formal proof for the equivalence of Definition 2 and Definition 3. Average parallelism is a very important factor for speedup and efficiency. It has been careful studied in [10]. S_∞ gives the best possible speedup based on the inherent parallelism of an application. There are no machine dependent factors considered. With only limited number of available processors and with the communication latency considered, the speedup will be less than the best speedup S_∞. If there are N processors available and $N < i$, then some processors have to do $\frac{W_i}{i}\lceil\frac{i}{N}\rceil$ work and the rest of the processors will do $\frac{W_i}{i}\lfloor\frac{i}{N}\rfloor$ work. In this case, assuming W_i and W_j cannot be solved simultaneously for $i \neq j$, the elapsed time will be

$$t_i(N) = \frac{W_i}{i\Delta}\lceil\frac{i}{N}\rceil$$

and

$$T(N) = \sum_{i=1}^{m} \frac{W_i}{i\Delta}\lceil\frac{i}{N}\rceil. \tag{6}$$

The speedup is

$$S_N = \frac{T(1)}{T(N)} = \frac{\sum_{i=1}^{m} W_i}{\sum_{i=1}^{m} \frac{W_i}{i}\lceil \frac{i}{N} \rceil}. \quad (7)$$

Communication latency is an important factor contributing to the complexity of a parallel algorithm. Unlike degree of parallelism, communication latency is machine dependent. It depends on the communication network, the routing scheme, and the adopted switching technique. For instance, the switching technique used in first generation multicomputers is *store-and-forward*. Second generation multicomputers adopt *circuit switching* or *wormhole routing* switching techniques. These new switching techniques reduce the communication cost considerably. Let Q_N be the communication overhead when N processors are used in parallel processing; the general speedup becomes

$$S_N = \frac{T(1)}{T(N)} = \frac{\sum_{i=1}^{m} W_i}{\sum_{i=1}^{m} \frac{W_i}{i}\lceil \frac{i}{N} \rceil + Q_N}. \quad (8)$$

3 Models of Speedup

In last section we developed a general speedup formula and showed how the number of processors and degradation parameters will influence the performance. However, the speedup is not only dependent on these parameters. It also depends on how we view the problem. With different points of view, we will get different models of speedup and will get different speedup formulations.

One viewpoint emphasizes shortening the time a problem takes to solve by parallel processing. With more and more computation power available, the problem can 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. Speedup formulation based on this philosophy is called *fixed-size speedup*. In the previous section, we adopt fixed-size speedup implicitly. Equation (8) is the general speedup formula for fixed-size speedup. Fixed-size speedup is suitable for many applications.

For some applications we may have a time limitation, but we may not want to solve the problem as soon as possible. If we have more computation power, we may want to increase the problem size, do more operation, get a more accurate solution and keep the execution time unchanged. This viewpoint leads to a new model of speedup, called *fixed-time speedup*. Many scientific and engineering applications can be represented by some partial differential equations, which can be discretized for different choices of grid spacing. Coarser grids demand less computation, but finer grids give more accurate solutions. If more accurate solutions are desired, this kind of application will fit the fixed-time speedup model. One good example is weather forecasting. With more computation power, we may not want to give the forecast earlier. Rather, we may wish to add more factors into the weather model – increasing the problem size and get a more accurate solution – giving a more precise forecast.

For fixed-time speedup the workload is scaled up with the number of processors available. Let W_i' be the amount of scaled up work executed with degree of parallelism i and m' be the maximum degree of parallelism of the scaled up problem when N processors are available. In order to keep the same turnaround time as the sequential version, we must have

$$\sum_{i=1}^{m} W_i = \sum_{i=1}^{m'} \frac{W'_i}{i}\lceil \frac{i}{N} \rceil + Q_N$$

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

$$S'_N = \frac{\sum_{i=1}^{m'} W'_i}{\sum_{i=1}^{m'} \frac{W'_i}{i}\lceil \frac{i}{N} \rceil + Q_N} = \frac{\sum_{i=1}^{m'} W'_i}{\sum_{i=1}^{m} W_i}. \quad (9)$$

From our experience in using multicomputers, we have found that the memory capacity plays a very important role on performance. Existing multicomputers do not support virtual memory and memory is distributed and associated with each node. The memory associated with each node is relative small. 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 nodes available, instead of keeping the execution time fixed, we may want to meet the memory capacity and increase the execution time. In general, the question is that, if you want to increase the problem size, do you have enough memory for the size increase? If you do have adequate memory space for the size increase, and after the problem size is increased to meet the time limit you still have memory space available, do you want to increase the problem size further by using this unused memory space and to get an even better solution? For memory-bounded speedup the answer is yes. Like fixed-time speedup, memory-bounded speedup is a scaled speedup. The problem size is scaled up with system size. The difference is that in fixed-time speedup the execution time is the dominant factor and in memory-bounded speedup the

memory capacity is the dominant factor. Most of the applications which fit fixed-time speedup will fit memory-bounded speedup when accurate solutions are the premier goal. A good application for memory-bounded speedup is simulation. If we simulate a nuclear power plant, obtaining an accurate solution probably will be the highest priority.

With memory capacity considered as a factor of performance, the requirement of solving an application contains two parts. One is the computation requirement, which is the workload, and another is the memory requirement. For a given application, these two requirements are related to each other, and the workload can be seen as a function of memory requirement. Let M represent the memory requirement and let g represent the relation, we have $W = g(M)$, or $M = g^{-1}(W)$, where g^{-1} is the inverse function of g. Under different architectures the memory capacity will change differently with the number of processors available. For multicomputers, the memory capacity increases linearly with the number of nodes available. If $W = \sum_{i=1}^{m} W_i$ is the workload for sequential execution, $W^* = \sum_{i=1}^{m^*} W_i^*$ is the scaled workload when N processors are available, m^* is the maximum degree of parallelism of the scaled problem, then the memory limitation for multicomputers can be stated as: *the memory requirement for any active node is less than or equal to* $g^{-1}(\sum_{i=1}^{m} W_i)$. Here the main point is that the memory occupation on each node is fixed. Equation (10) is the general speedup formula for memory-bounded speedup.

$$S_N^* = \frac{\sum_{i=1}^{m^*} W^*{}_i}{\sum_{i=1}^{m^*} W_i^* \lceil \frac{i}{N} \rceil + Q_N} \tag{10}$$

4 Simplified Models of Speedup

Three general speedup formulations have been proposed for three models of speedup. These formulations contain both uneven allocation and communication latency degradations. They are more close to actual speedup and give better upper bounds on the performance of parallel algorithms. On the other hand, these formulations are problem dependent and difficult to understand. They give more detailed information for each application, but lose the global view of the possible performance gain. In this section, we study a simplified case for speedup, which is the special case studied by Amdahl and Gustafson. We do not consider the communication overhead, $Q_N = 0$, and assume that the allocation only contains two parts, sequential part and 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, $W_1 = W_1' = W_1^*$.

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

$$S_N = \frac{W_1 + W_N}{W_1 + \frac{W_N}{N}}. \tag{11}$$

which is known as Amdahl's law. From Eq.(11) and Fig. 4 we can see 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 the reciprocal of the sequential fraction W_1.

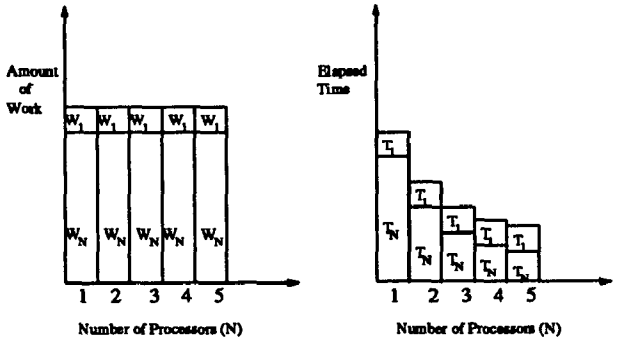

Figure 4. Amdahl's Law

Under the simplified condition, $\sum_{i=1}^{m} W_i = W_1 + W_N$ and $\sum_{i=1}^{m'} \frac{W'_i}{i} \lceil \frac{i}{N} \rceil + Q_N = W_1' + \frac{W_N'}{N}$. Therefore, for fixed-time speedup, we have $W_1 + W_N = W_1' + \frac{W_N'}{N}$. Since W_1 is fixed, we have $W_N = \frac{W_N'}{N}$. That is $W_N' = NW_N$. Equation (9) becomes

$$S_N' = \frac{\sum_{i=1}^{m'} W'_i}{\sum_{i=1}^{m} W_i} = \frac{W_1' + W_N'}{W_1 + W_N} = \frac{W_1 + NW_N}{W_1 + W_N}. \tag{12}$$

The simplified fixed-time speedup formula Eq.(12) is Gustafson's scaled speedup, which was proposed by Gustafson in 1988 [5]. From Eq.(12) we can see that the parallel portion of the application is scaled up linearly with the system size. And, therefore, the speedup increases linearly with the system size. The relation of workload and elapsed time for Gustafson's scaled speedup is depicted in Figure (5), where T_1 is the execution time for the sequential portion of work. T_N is the execution time for the parallel portion of load.

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

Definition 4 *A function g is homomorphism if*

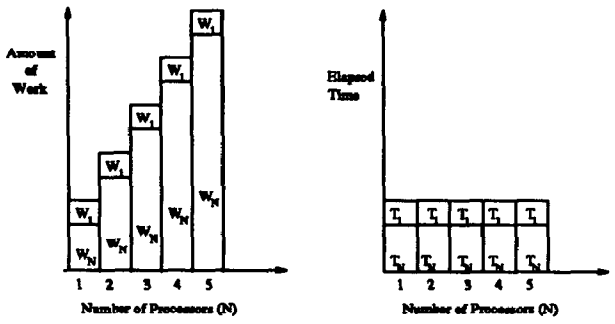

Figure 5. Gustafson's Scaled Speedup

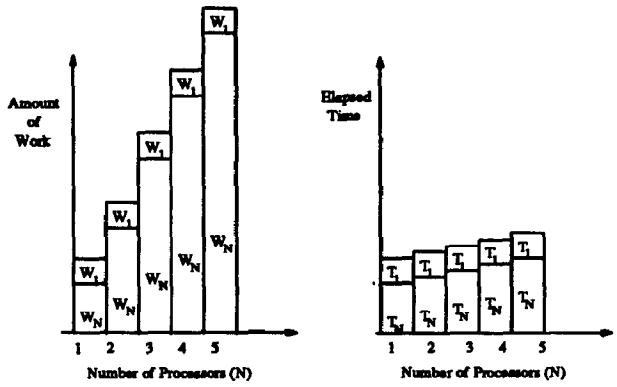

Figure 6. Simplified Memory-Bounded Scaled Speedup

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 homomorphism function 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 homomorphism function 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 homomorphism functions, $\bar{g}(x) = x^b$, which is not the same as $g(x)$. The sequential portion of the workload W_1 is independent of the system size. If we do not consider memory influence on the sequential portion we have the following theorem:

Theorem 1 *If $W = g(M)$ for some homomorphism function g, $g(cx) = \bar{g}(c)g(x)$, then, with all data being shared by all the available processors, the simplified memory-bounded speedup is*

$$S_N^* = \frac{W_1 + \bar{g}(N)W_N}{W_1 + \frac{\bar{g}(N)}{N}W_N} \tag{13}$$

Proof: As mentioned before, W_N is the parallel portion of the workload when one processor is used. Let the memory requirement of W_N be M, $W_N = g(M)$. M is the memory requirement when one node is available. With N nodes available, the memory capacity will increase to NM. Using all of the available memory, for the scaled parallel portion W_N^*, $W_N^* = g(NM) = \bar{g}(N)g(M)$. Therefore, $W_N^* = \bar{g}(N)W_N$ and

$$S_N^* = \frac{W_1^* + W_N^*}{W_1^* + W_N^*/N} = \frac{W_1 + \bar{g}(N)W_N}{W_1 + \frac{\bar{g}(N)}{N}W_N} \tag{14}$$

□

In the proof of Theorem 1, we claimed that $W_N^* = g(NM)$. This claim is true under two assumptions: 1) the data is shared by all available processors, and 2) all the available memory space are used for better solutions. A computation with the first property is called global computation. Equation (13) is the simplified memory-bounded speedup for global computation. In general, data may be duplicated on different nodes and the available memory may not be fully used for increased problem size. Based on Eq.(13), a more generalized speedup will be

$$S_N^* = \frac{W_1 + G(N)W_N}{W_1 + \frac{G(N)}{N}W_N}. \tag{15}$$

Equation (15) will be referred to as *simplified memory-bounded* (SMB) scaled speedup. SMB scaled speedup is determined by the function $G(N)$, which gives the information of how the memory change will influence the change of problem size. When the problem size is independent of the system, the problem size is fixed, $G(N) = 1$. In this case, SMB scaled speedup is the same as Amdahl's law, i.e., Eq.(15) and Eq.(11) are equivalent. *Local computation model* is one computation model studied in [12]. In the local computation model, when more processors are available, work will be replicated on these available processors. Computation is done locally on each node, and communication between nodes is not required. In this case, when memory is increased N times, the workload also increases N times, i.e., $G(N) = N$. And, in this case, SMB scaled speedup is same as Gustafson's scaled speedup. SMB scaled speedup contains both Amdahl's law and Gustafson's scaled speedup as its special cases. For most of the scientific and engineering applications, the computation requirement increases faster than the memory requirement. For these applications, $\bar{g}(N) > N$ and memory-bounded speedup will likely give a higher speedup than fixed-time speedup.

Equation (15), the proposed scaled speedup formulation, may be not easy to fully understood at first glance. Here we use matrix multiplication as an example to illustrate it. A matrix often represents some discretized continuum. Enlarging the matrix size generally will lead to a more accurate solution for the continuum. For matrices with dimension n, the computation requirement of matrix multiplication is $2n^3$ and the memory requirement is $3n^2$ (roughly). Thus,

$$W_N = 2n^3, \qquad M = 3n^2.$$

Writing W_N as a function of M, we have

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

This means that

$$W_N = g(M) = \left(\frac{2M}{3}\right)^{\frac{3}{2}}, \quad \bar{g}(N) = N^{\frac{3}{2}}. \qquad (16)$$

The simplified memory-bounded speedup for global computation will be

$$S_N^* = \frac{W_1 + N^{\frac{3}{2}} W_N}{W_1 + N^{\frac{1}{2}} W_N}. \qquad (17)$$

Global computation uses distributed local memories as a large shared memory [12]. All the data is distributed and shared. When these local memories are used locally without sharing, the computation is local computation and $W_N^* = Ng(M)$. This means that $\bar{g} = N$. The speedup is

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

which is Gustafson's scaled speedup. For matrix multiplication $C = AB$, let A_i be the i^{th} row of A, $i = 1, ..., n$, and let B_j be the j^{th} column of B, $i = 1, ..., n$. The local computation and global computation of the matrix multiplication are shown in Figure (7) and (8), respectively.

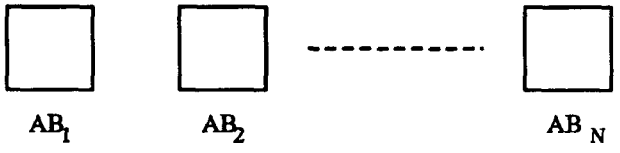

Figure 7. Matrix Multiplication with Local Computation

We have studied two cases of memory-bounded

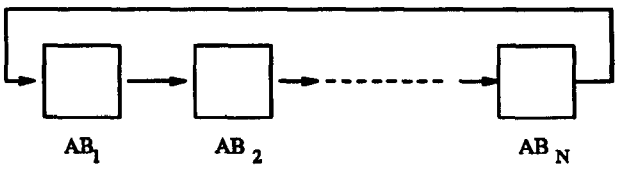

Figure 8. Matrix Multiplication with Global Computation

scaled speedup, global computation and local computation. Most of the applications are some combination of these two computations. Data is distributed in some part and duplicated in the other part. The duplication may be required by inherent properties of the given application, or may be added in deliberately to reduce the communication. Speedup formulation for these applications depends on the ratio of the global and the local computation. Deriving speedup formulation for these combined applications is difficult. This is not only because we are facing more complicated situation, but also because of the uncertainty of the ratio. The duplicated part might not increase with system size. It might increase but with a speed which is different from the increasing speed of the global part. And, an application may start as global computation, but, when the computation power increases, duplication may be added in as a part of the effort for better solution. In general, $G(N)$ is application dependent. We derive $G(N)$ for a special case as an example. The structure of this derivation can be used as a guideline for general applications.

Lemma 1 *If function g is a homomorphism function, $g(cx) = \bar{g}(c)g(x)$; g^{-1} exists and also homomorphism, $g^{-1}(cx) = h(c)g^{-1}(x)$ for some function h, then $\bar{g}$ has inverse and $\bar{g}^{-1} = h$.*

Proof: Since

$$\begin{aligned} cy &= g[g^{-1}(cy)] = g[h(c)g^{-1}(y)] \\ &= \bar{g}[h(c)]g[g^{-1}(y)] = \bar{g}[h(c)]y, \end{aligned}$$

we have

$$\bar{g}[h(c)] = c \qquad \text{for any real number } c \qquad (18)$$

Also, since

$$cy = g^{-1}[g(cy)] = g^{-1}[\bar{g}(c)g(y)] = h[\bar{g}(c)]y,$$

we have

$$h[\bar{g}(c)] = c \qquad \text{for any real number } c \qquad (19)$$

By Eq.(18) and Eq.(19), the function $\bar{g}$ has inverse

and $\bar{g}^{-1} = h$.

□

Theorem 2 *Assume $W = g(M)$ for some homomorphism function g, where $g(cM) = \bar{g}(c)g(M)$, g inverse exists and is a homomorphism. If the workload is scaled up to meet the time limitation with global computation first and the rest of the unused memory space is then used to increase the problem size further with local computation, we have*

$$G(N) = (1 + \bar{g}[1 - \frac{\bar{g}^{-1}(N)}{N}])N. \qquad (20)$$

Proof: By the fixed-time speedup, after the number of nodes changes from 1 to N, the parallel portion of work will increase from W_N to NW_N (see Figure 5). The storage requirement is given by the function g^{-1}. For operation requirement NW_N, the memory requirement is $g^{-1}(NW_N) = \bar{g}^{-1}(N)g^{-1}(W_N)$.

Let M represent the size of the memory associated with each node which can be used for parallel processing. Then, when the number of nodes equals 1, the total memory available is M, which is equal to $g^{-1}(W_N)$. When the number of nodes equals N, the total memory available changes to NM. We first fix the execution time and increase the problem size to meet the time limitation. After the timed-bounded scaled up, the unused memory space is the difference between current available memory and current memory requirement, which equals

$$\begin{aligned} & NM - g^{-1}(NW_N) = NM - \bar{g}^{-1}(N)g^{-1}(W_N) \\ = \quad & NM - g^{-1}(N)M = (N - \bar{g}^{-1}(N))M. \end{aligned}$$

The unused space at each node is

$$\frac{[N - \bar{g}^{-1}(N)]M}{N} = [1 - \frac{\bar{g}^{-1}(N)}{N}]M.$$

The problem size can be further scaled by using these unused memory space. The further scaled computation on each node is given by the function g, and it is equal to

$$g(\left[1 - \frac{g^{-1}(N)}{N}\right] M) = \bar{g}(1 - \frac{\bar{g}^{-1}(N)}{N})g(M) \qquad (21)$$

$$= \bar{g}(1 - \frac{\bar{g}^{-1}(N)}{N})W_N \qquad (22)$$

Therefore, the computation on each node becomes

original operation on each node
\+ the operation increase on each node

$$= W_N + \bar{g}(1 - \frac{\bar{g}^{-1}(N)}{N})W_N \qquad (23)$$

$$= \left[1 + \bar{g}\left(1 - \frac{\bar{g}^{-1}(N)}{N}\right)\right] W_N. \qquad (24)$$

and, for the scaled parallel computation W_N^*,

$$W_N^* = N\left[1 + \bar{g}\left(1 - \frac{\bar{g}^{-1}(N)}{N}\right)\right] W_N.$$

This concludes

$$G(N) = N[1 + \bar{g}(1 - \frac{\bar{g}^{-1}(N)}{N})]. \qquad (25)$$

□

Figure 9 depicts the speedup difference among the fixed-sized model, the timed-bounded model and the memory-bounded model. The associate function, $\bar{g}$, used in Figure 9 is the associate function of matrix multiplication, $\bar{g}(N) = N^{\frac{3}{2}}$. As most matrix computations have the same associate function $\bar{g}(N) = N^{\frac{3}{2}}$, the speedup relation depicted by Figure 9 is in general true for a large class of applications.

5 Conclusion

It is known that the performance of parallel processing is influenced by the inherent parallelism of the application, by the computation power and by the memory capacity of the parallel computing system. However, how are these three factors related to each other and, how do they influence the performance of parallel processing generally is unknown. Discovering the answers for these unknowns is very important for designing efficient parallel algorithms and for constructing high performance parallel systems. In this paper one model of speedup, *memory-bounded speedup*, is careful studied. This model is simple, and it contains all of these three factors as its parameters. It shows the degradations and the possible performance gain of parallel computation. Incidently, the importance of memory capacity to the speedup measurement was also realized by Gustafson as indicated in his recent paper [14].

As a 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 memory-

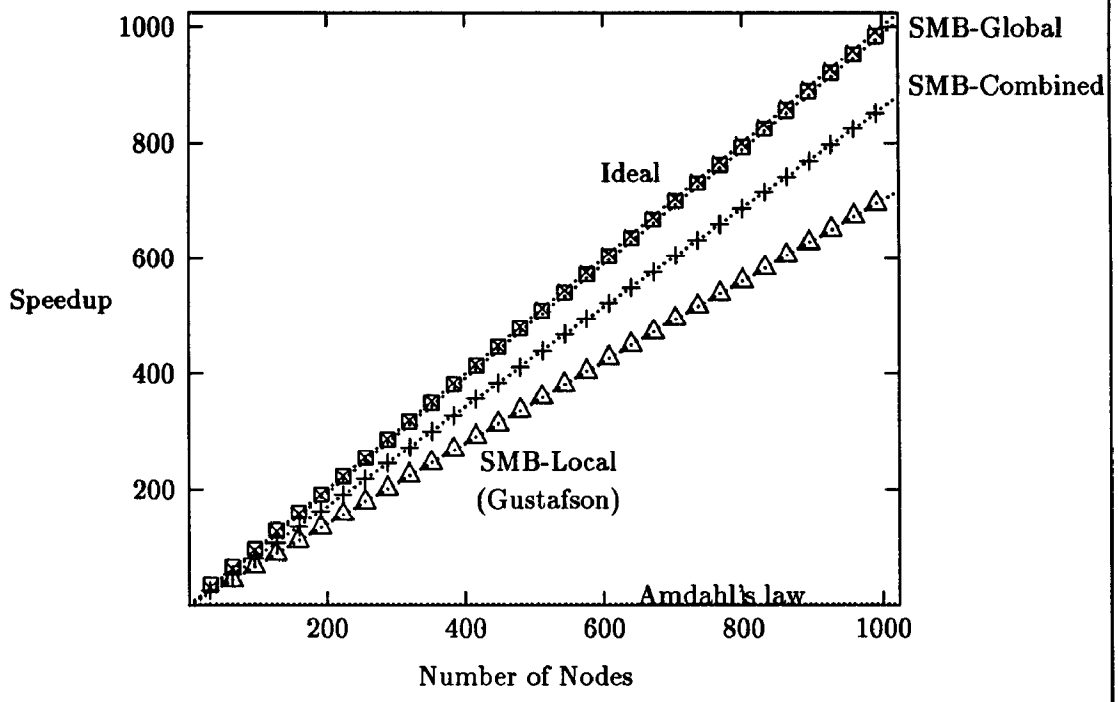

Figure 9. Amdahl's law, Gustafson's speedup and SMB speedup, where $W_1 = 0.3$ and $\bar{g}(N) = N^{\frac{3}{2}}$

bounded speedup. One set of formulations are general speedup formulas. These formulas contain more parameters and provide more accurate information. The second set of formulations only consider a special, simplified case. These formulations give the performance in principle and lead to a better understanding of parallel processing. 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 of the Amdahl's law and Gustafson's speedup as its special cases. Amdhal's law suggests that the sequential portion of the workload will dominate the performance when the number of processors is large. Gustafson' scaled speedup claims that the influence of the sequential portion is independent of the system size. Simplified memory-bounded Scaled speedup declares that the sequential fraction will change with the system size. Since the computation requirement increases faster than the memory requirement for most applications, the sequential fraction could be reduced when the number of processors increase.

The three models of speedup, *fixed-size speedup*, *fixed-time speedup* and *memory-bounded speedup*, are based on different viewpoints and suitable for different classes of applications. Applications do exist which do not fit any of the models of speedup, but satisfy some combination of the models. We plan to study more on the performance issues and arrive at a better understanding of parallel processing.

References

1. A. H. Karp and H. P. Flatt, "Measuring parallel processor performance," *CACM*, vol. 33, pp. 539–543, May 1990.

2. J. Ortega and R. Voigt, "Solution of partial differential equations on vector and parallel computers," *SIAM Review*, June 1985.

3. M. Barton and G. Withers, "Computing performance as a function of the speed, quantity, and cost of the processors," in *Proc. Supercomputing'89*, pp. 759–764, 1989.

4. G. Amdahl, "Validity of the single-processor approch to achieving large scale computing capabilities," in *Proc. AFIPS Conf.*, pp. 483–485, 1967.

5. J. Gustafson, "Reevaluating amdahl's law," *CACM*, vol. 31, pp. 523–533, May 1988.

6. J. Gustafson, G. Montry, and R. Benner, "Development of parallel methods for a 1024-processor hypercube," *SIAM J. on SSTC*, vol. 9, July 1988.

7. A. Veen, "Dataflow machine architectures," *ACM Comp. Surv.*, pp. 365–396, Dec. 1986.

8. E. Coffman, "Introduction to deterministic scheduling theory." Computer and Job/Shop Scheduling Theory, 1976. ed. E.G. Coffman, John Wiley and Sons.

9. M. A. Marsan, G. Bailbo, and G. Conte, "A class of generalized stochastic petri nets for the performance analysis of multiprocessor systems," *ACM TOCS*, pp. 93–122, May 1984.

10. D. Eager, J. Zahorjan, and E. Lazowska, "Speedup versus efficiency in parallel system," *IEEE TC*, pp. 403–423, March 1989.

11. M. Kumar, "Measuring parallelism in computation intensive scientific/engineering applications," *IEEE-TC*, vol. 37, pp. 1088–1098, Sep. 1988.

12. X.-H. Sun, "Parallel computation models for scientific computing on multicomputers." Ph.D. Dissertation, Computer Science Department, Michigan State University, 1990. In preparation.

13. K. Sevcik, "Characterizations of parallelism in applications and their use in scheduling," in *Proc. of ACM SIGMETRICS and Performance'89*, May 1989.

14. J. Gustafson, "Fixed time, tiered memory, and superlinear speedup," in *Proc. of the Fifth Conf. on Distributed Memory Computers*, 1990. to appear.