Performance Considerations of Shared Virtual Memory Machines

Xian-He Sun, *Senior Member, IEEE*, and Jianping Zhu

Abstract—**Generalized speedup is defined as parallel speed over sequential speed. In this paper the generalized speedup and its relation with other existing performance metrics, such as traditional speedup, efficiency, scalability, etc., are carefully studied. In terms of the introduced asymptotic speed, we show that the difference between the generalized speedup and the traditional speedup lies in the definition of the efficiency of uniprocessor processing, which is a very important issue in shared virtual memory machines. A scientific application has been implemented on a KSR-1 parallel computer. Experimental and theoretical results show that the generalized speedup is distinct from the traditional speedup and provides a more reasonable measurement. In the study of different speedups, an interesting relation between fixed-time and memory-bounded speedup is revealed. Various causes of superlinear speedup are also presented.**

Index Terms—**High performance computing, parallel processing, performance evaluation, performance metrics, scalability, speedup, shared virtual memory.**

I. Introduction

In recent years parallel processing has enjoyed unprecedented attention from researchers, government agencies, and industries. This attention is mainly due to the fact that, with the current circuit technology, parallel processing seems to be the only remaining way to achieve higher performance. However, while various parallel computers and algorithms have been developed, their performance evaluation is still elusive. In fact, the more advanced the hardware and software, the more difficult it is to evaluate the parallel performance. In this paper, targeting recent development of shared virtual memory machines, we study the *generalized speedup* [1] performance metric, its relation with other existing performance metrics, and the implementation issues.

Distributed-memory parallel computers dominate today's parallel computing arena. These machines, such as the Kendall Square KSR-1, Intel Paragon, TMC CM-5, and IBM SP2, have successfully delivered high performance computing power for solving some of the so-called "grand-challenge" problems. From the viewpoint of processes, there are two basic process synchronization and communication models. One is the shared-memory model in which processes communicate through shared variables. The other is the message-passing model in which processes communicate through explicit message passing. The shared-memory model provides a sequential-like program paradigm. Virtual address space separates the user logical memory from physical memory. This separation allows an extremely large virtual memory to be provided on a sequential machine when only a small physical memory is available. Shared virtual address combines the private virtual address spaces distributed over the nodes of a parallel computer into a globally shared virtual memory [2]. With shared virtual address space, the shared-memory model supports shared virtual memory, but requires sophisticated hardware and system support. An example of a distributed-memory machine which supports shared virtual address space is the Kendall Square KSR-1.[1] Shared virtual memory simplifies the software development and porting process by enabling even extremely large programs to run on a single processor before being partitioned and distributed across multiple processors. However, the memory access of the shared virtual memory is nonuniform [2]. The access time of local memory and remote memory is different. Running a large program on a small number of processors is possible but could be very inefficient. The inefficient sequential processing will lead to a misleading high performance in terms of speedup or efficiency.

Generalized speedup, defined as parallel speed over sequential speed, is a newly proposed performance metric [1]. In this paper, through both theoretical proofs and experimental results, we show that generalized speedup provides a more reasonable measurement than traditional speedup. In the process of studying generalized speedup, the relation between the generalized speedup and many other metrics, such as efficiency, scaled speedup, and scalability, are also studied. The relation between fixed-time and memory-bounded scaled speedup is analyzed. Various reasons for superlinearity in different speedups are also discussed. Results show that the main difference between the traditional speedup and the generalized speedup is how to evaluate the efficiency of the sequential processing on a single processor.

The paper is organized as follows. In Section II, we study traditional speedup, including the scaled speedup concept, and introduce some terminology. Analysis shows that the traditional speedup, fixed-size or scaled size, may achieve superlinearity on shared virtual memory machines. Furthermore, with the traditional speedup metric, the slower the remote

Manuscript received Mar. 5, 1994; revised Mar. 14, 1995.

X.-H. Sun is with the Department of Computer Science, Louisiana State University, Baton Rouge, LA 70803-4020; e-mail: sun@bit.csc.lsu.edu.

J. Zhu is with the NSF Engineering Research Center, Department of Mathematics and Statistics, Mississippi State University, Mississippi State, MS 39762.

To order reprints of this article, e-mail: transactions@computer.org, and reference IEEECS Log Number D95055.

1. Traditionally, the message-passing model is bounded by the local memory of the processing processors. With recent technology advancement, the message-passing model has extended the ability to support shared virtual memory.

memory access is, the larger the speedup. Generalized speedup is studied in Section III. The term *asymptotic speed* is introduced for the measurement of generalized speedup. Analysis shows the differences and the similarities between the generalized speedup and the traditional speedup. Relations between different performance metrics are also discussed. Experimental results of a production application on a Kendall Square KSR-1 parallel computer are given in Section IV. Section V contains a summary.

II. The Traditional Speedup

One of the most frequently used performance metrics in parallel processing is speedup. It is defined as sequential execution time over parallel execution time. Parallel algorithms often exploit parallelism by sacrificing mathematical efficiency. To measure the true parallel processing gain, the sequential execution time should be based on a commonly used sequential algorithm. To distinguish it from other interpretations of speedup, the speedup measured with a commonly used sequential algorithm has been called *absolute speedup* [3]. Another widely used interpretation is the *relative speedup* [3], which uses the uniprocessor execution time of the parallel algorithm as the sequential time. There are several reasons to use the relative speedup. First, the performance of an algorithm varies with the number of processors. Relative speedup measures the variation. Second, relative speedup avoids the difficulty of choosing the practical sequential algorithm, implementing the sequential algorithm, and matching the implementation/programming skill between the sequential algorithm and the parallel algorithm. Also, when problem size is fixed, the time ratio of the chosen sequential algorithm and the uniprocessor execution of the parallel algorithm is fixed. Therefore, the relative speedup is proportional to the absolute speedup. Relative speedup is the speedup commonly used in performance study. In this study we will focus on relative speedup and reserve the terms *traditional speedup* and *speedup* for relative speedup. The concepts and results of this study can be extended to absolute speedup.

From the problem size point of view, speedup can be divided into the *fixed-size speedup* and the *scaled speedup*. Fixed-size speedup emphasizes how much execution time can be reduced with parallel processing. Amdahl's law [4] is based on the fixed-size speedup. The scaled speedup is concentrated on exploring the computational power of parallel computers for solving otherwise intractable large problems. Depending on the scaling restrictions of the problem size, the scaled speedup can be classified as the *fixed-time speedup* [5] and the *memory-bounded speedup* [6]. As the number of processors increases, fixed-time speedup scales problem size to meet the fixed execution time. Then the scaled problem is also solved on an uniprocessor to get the speedup. As the number of processors increases, memory-bounded speedup scales problem size to utilize the associated memory increase. A detailed study of the memory-bounded speedup can be found in [6].

Let p and S_p be the number of processors and the speedup with p processors.

DEFINITION 1.

- *Superlinear speedup*: $S_p > p$
- *Unitary speedup*: $S_p = p$.
- *Linear speedup*: $S_p = a \cdot p$ *for some constant* $a > 0$.

It is debatable if any machine-algorithm pair can achieve "truly" superlinear speedup. Seven possible causes of superlinear speedup are listed in Fig. 1. The first four causes in Fig. 1 are patterned from [7].

1. cache size increased in parallel processing
2. overhead reduced in parallel processing
3. latency hidden in parallel processing
4. randomized algorithms
5. mathematical inefficiency of the serial algorithm
6. high memory access latency in the sequential processing
7. profile shifting

Fig. 1. Causes of superlinear speedup.

Cause 1 is unlikely applicable for scaled speedup, since when problem size scales up, by memory or by time constraint, the cache hit ratio is unlikely to increase. Cause 2 in Fig. 1 can be considered theoretically [8], there is no measured superlinear speedup ever attributed to it. Cause 3 does not exist for relative speedup since both the sequential and parallel execution use the same algorithm. Since parallel algorithms are often mathematically inefficient, cause 5 is a likely source of superlinear speedup of relative speedup. A good example of superlinear speedup based on 5 can be found in [9]. Cause 7 will be explained in the end of Section III, after the generalized speedup is introduced.

With the virtual memory and shared virtual memory architecture, cause 6 can lead to an extremely high speedup, especially for scaled speedup where an extremely large problem has to be run on a single processor. Fig. 5 shows a measured superlinear speedup on a KSR-1 machine. The measured superlinear speedup is due to the inherent deficiency of the traditional speedup metric. To analyze the deficiency of the traditional speedup, we need to introduce the following definition.

DEFINITION 2. *The* `cost` *of parallelism* i *is the ratio of the total number of processor cycles consumed in order to perform one unit operation of work when* i *processors are active to the machine clock rate.*

The sequential execution time can be written in terms of work:

$$\text{Sequential execution time} = \text{Amount of work} \times \frac{\text{Processor cycles per unit of work}}{\text{Machine clock rate}}. \quad (1)$$

The ratio in the right hand side of (1), processor cycles per unit of work over machine clock rate, is the cost of sequential processing.

Work can be defined as arithmetic operations, instructions, transitions, or whatever is needed to complete the application. In scientific computing the number of floating-point operations (FLOPS) is commonly used to measure work. In general, work

may be of different types, and units of different operations may require different numbers of instruction cycles to finish. (For example, the times consumed by one division and one multiplication may be different depending on the underlying machine, and operation and memory reference ratio may be different for different computations.) The influence of work type on the performance is one of the topics studied in [1]. In this paper, we study the influence of inefficient memory access on the performance. We assume that there is only one work type and that any increase in the number of processor cycles is due to inefficient memory access.

In a shared virtual memory environment, the memory available depends on the system size. Let W_i be the amount of work executed when i processors are active (work performed in all steps that use i processors), and let $W = \sum_{i=1}^{p} W_i$ represent the total work. The cost of parallelism i in a p processor system, denoted as $c_p(i, W)$, is the elapsed time for one unit operation of work when i processors are active. Then, $W_i \cdot c_p(i, W)$ gives the accumulated elapsed time where i processors are active. $c_p(i, W)$ contains both computation time and remote memory access time.

The uniprocessor execution time can be represented in terms of uniprocessor cost,

$$t(1) = \sum_{i=1}^{p} W_i \cdot c_p(s, W),$$

where $c_p(s, W)$ is the cost of sequential processing on a parallel system with p processors. It is different from $c_p(1, W)$ which is the cost of the sequential portion of the parallel processing. Parallel execution time can be represented in terms of parallel cost,

$$t(p) = \sum_{i=1}^{p} \frac{W_i}{i} \cdot c_p(i, W).$$

The traditional speedup is defined as

$$S_p = \frac{t(1)}{t(p)} = \frac{\sum_{i=1}^{p} W_i \cdot c_p(s, W)}{\sum_{i=1}^{p} W_i \cdot c_p(i, W)}. \tag{2}$$

Depending on architecture memory hierarchy, in general $c_p(i, W)$ may not equal $c_p(j, W)$ for $i \neq j$ [10]. If $c_p(i, W) = c_p(p, W)$, for $1 \leq i < p$, then

$$S_p = \frac{c_p(s, W)}{c_p(p, W)} \cdot \frac{W}{\sum_{i=1}^{p} \frac{W_i}{i}}. \tag{3}$$

The first ratio of (3) is the *cost ratio*, which gives the influence of memory access delay. The second ratio,

$$\frac{W}{\sum_{i=1}^{p} \frac{W_i}{i}} \tag{4}$$

is the simple analytic model based on degree of parallelism [6]. It assumes that memory access time is constant as problem size and system size vary. The cost ratio distinguishes the different performance analysis methods with or without consideration of the memory influence. In general, cost ratio depends on memory miss ratio, page replacement policy, data reference pattern, etc. Let remote access ratio be the quotient of the number of remote memory accesses and the number of local memory accesses. For a simple case, if we assume there is no remote access in parallel processing and the remote access ratio of the sequential processing is $(p - 1)/p$, then

$$\frac{c_p(s, W)}{c_p(p, W)} = \frac{1}{p} + \frac{p-1}{p} \cdot \frac{\text{time of per remote access}}{\text{time of per local access}} \tag{5}$$

Equation (5) approximately equals the time of per remote access over the time of per local access. Since the remote memory access is much slower than the local memory access under the current technology, the speedup given by (3) could be considerably larger than the simple analytic model (4). In fact, the slower the remote access is, the larger the difference. For the KSR-1, the time ratio of remote and local access is about 7.5 (see Section IV). Therefore, for $p = 32$, the cost ratio is 7.3. For any $W / \sum_{i=1}^{p} \frac{W_i}{i} > 0.14$, under the assumed remote access ratio, we will have a superlinear speedup.

III. The Generalized Speedup

While parallel computers are designed for solving large problems, a single processor of a parallel computer is not designed to solve a very large problem. A uniprocessor does not have the computing power that the parallel system has. While solving a small problem is inappropriate on a parallel system, solving a large problem on a single processor is not appropriate either. To create a useful comparison, we need a metric that can vary problem sizes for uniprocessor and multiple processors. *Generalized speedup* [1] is one such metric.

$$\text{Generalized Speedup} = \frac{\text{Parallel Speed}}{\text{Sequential Speed}}. \tag{6}$$

Speed is defined as the quotient of work and elapsed time. Parallel speed might be based on scaled parallel work. Sequential speed might be based on the unscaled uniprocessor work. By definition, generalized speedup measures the speed improvement of parallel processing over sequential processing. In contrast, the traditional speedup (2) measures time reduction of parallel processing. If the problem size (work) for both parallel and sequential processing are the same, the generalized speedup is the same as the traditional speedup. From this point of view, the traditional speedup is a special case of the generalized speedup. For this and for historical reasons, we call the traditional speedup the speedup, and call the speedup given in (6) the generalized speedup.

Like the traditional speedup, the generalized speedup can also be further divided into fixed-size, fixed-time, and memory-bounded speedup. Unlike the traditional speedup, for the generalized speedup, the scaled problem is solved only on multiple processors. The fixed-time generalized speedup is *sizeup* [1]. The fixed-time benchmark SLALOM [11] is based on sizeup.

If memory access time is fixed, one might always assume that the uniprocessor cost $c_p(s)$ will be stablized after some initial decrease (due to initialization, loop overhead, etc.), assuming the memory is large enough. When cache and remote memory access are considered, cost will increase when a

slower memory has to be accessed. Fig. 2 depicts the typical cost changing pattern.

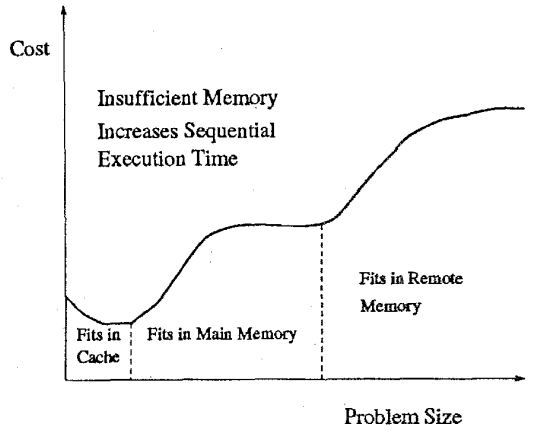


Fig. 2. Cost variation pattern.

From (1), we can see that uniprocessor speed is the reciprocal of uniprocessor cost. When the cost reaches its lowest value, the speed reaches its highest value. The uniprocessor speed corresponding to the stablized main memory cost is called the *asymptotic speed* (of uniprocessor). Asymptotic speed represents the performance of the sequential processing with efficient memory access. The asymptotic speed is the appropriate sequential speed for (6). For memory-bounded speedup, the appropriate memory bound is the largest problem size which can maintain the asymptotic speed. After choosing the asymptotic speed as the sequential speed, the corresponding asymptotic cost has only local access and is independent of the problem size. We use $c(s, W_0)$ to denote the corresponding asymptotic cost, where W_0 is a problem size which achieves the asymptotic speed. If there is no remote access in parallel processing, as assumed in Section II, then $c(s, W_0)/c_p(p, W_0) = 1$. By (3), the corresponding speedup equals the simple speedup which does not consider the influence of memory access time. In general, parallel work W is not the same as W_0, and $c_p(i, W)$ may not equal $c_p(p, W)$ for $1 \le i \le p$. So, in general, we have

$$\textit{Generalized Speedup} = \frac{\dfrac{W}{\sum_{i=1}^{p} \dfrac{W_i}{i} \cdot c_p(i, W)}}{\dfrac{1}{c(s, W_0)}} = \frac{W \cdot c(s, W_0)}{\sum_{i=1}^{p} \dfrac{W_i}{i} \cdot c_p(i, W)}. \quad (7)$$

Equation (7) is another form of the generalized speedup. It is a quotient of sequential and parallel time as is traditional speedup (2). The difference is that, in (7), the sequential time is based on the asymptotic speed. When remote memory is needed for sequential processing, $c(s, W_0)$ is smaller than $c_p(s, W)$. Therefore, the generalized speedup gives a smaller speedup than traditional speedup.

Parallel efficiency is defined as

$$\textit{Efficiency} = \frac{\text{speedup}}{\text{number of processors}}. \quad (8)$$

The *Generalized Efficiency* can be defined similarly as

$$\textit{Generalized Efficiency} = \frac{\text{generalized speedup}}{\text{number of processors}}. \quad (9)$$

By definition,

$$\text{Efficiency} = \frac{W \cdot c(s, W)}{p \cdot \sum_{i=1}^{p} \dfrac{W_i}{i} \cdot c_p(i, W)} \quad (10)$$

and

$$\text{Generalized Efficiency} = \frac{W \cdot c(s, W_0)}{p \cdot \sum_{i=1}^{p} \dfrac{W_i}{i} \cdot c_p(i, W)}. \quad (11)$$

Equations (10) and (11) show the difference between the two efficiencies. Traditional speedup compares parallel processing with the measured sequential processing. Generalized speedup compares parallel processing with the sequential processing based on the asymptotic cost. From this point of view, generalized speedup is a reform of traditional speedup. The following lemmas are direct results of (7).

LEMMA 1. *If $c_p(s, W)$ is independent of problem size, traditional speedup is the same as generalized speedup.*

LEMMA 2. *If the parallel work, W, achieves the asymptotic speed, that is $W = W_0$, then the fixed-size traditional speedup is the same as the fixed-size generalized speedup.*

By Lemma 1, if the simple analytic model (4) is used to analyze performance, there is no difference between the traditional and the generalized speedup. If the problem size W is larger than the suggested initial problem size W_0, then the single processor speedup S_1 may not equal to one. S_1 measures the sequential inefficiency due to the difference in memory access.

The generalized speedup is also closely related to the scalability study. *Isospeed* scalability has been proposed recently in [12]. The isospeed scalability measures the ability of an algorithm-machine combination maintaining the average (unit) speed, where the average speed is defined as the speed over the number of processors. When the system size is increased, the problem size is scaled up accordingly to maintain the average speed. If the average speed can be maintained, we say the algorithm-machine combination is scalable and the scalability is

$$\psi(p, p') = \frac{p'W}{pW'}, \quad (12)$$

where W' is the amount of work needed to maintain the average speed when the system size has been changed from p to p', and W is the problem size solved when p processors were used. By definition

$$\textit{Average speed} = \frac{W}{p \cdot \sum_{i=1}^{p} \dfrac{W_i}{i} \cdot c_p(i, W)}$$

Since the sequential cost is fixed in (11), fixing average speed is equivalent to fixing generalized efficiency. Therefore the isospeed scalability can be seen as the iso-generalized-efficiency scalability. When the memory influence is not consedered, i.e., $c_p(s, W)$ is independent of the problem size, the iso-generalized-efficiency will be the same as the iso-traditional-efficiency. In this case, the isospeed scalability is the same as the isoefficiency scalability proposed in [13], [2].

LEMMA 3. *If the sequential cost $c_p(s, W)$ is independent of problem size or if the simple analysis model (4) is used for speedup, the isoefficiency and isospeed scalability are equivalent to each other.*

The following theorem gives the relation between the scalability and the fixed-time speedup.

THEOREM 1. *Scalability (12) equals one if and only if the fixed-time generalized speedup is unitary.*

PROOF. Let $c(s, W_0)$, $c_p(i, W)$, W, W_i be as defined in (7). If scalability (12) equals 1, let W', p' be as defined in (12) and define W_i' similarly as W_i, we have

$$\frac{p'}{W'} = \frac{p}{W}, \tag{13}$$

for any number of processors p and p'. By definition, generalized speedup

$$G\_S_{p'} = \frac{W' \cdot c(s, W_0)}{\sum_{i=1}^{p'} \frac{W'}{i} \cdot c_{p'}(i, W')}.$$

With some arithmetic manipulation, we have

$$\frac{W'}{p'} = \frac{G\_S_{p'}}{p'} \cdot \frac{\sum_{i=1}^{p'} \frac{W'}{i} \cdot c_{p'}(i, W')}{c(s, W_0)}.$$

Similarly, we have

$$\frac{W}{p} = \frac{G\_S_p}{p} \cdot = \frac{\sum_{i=1}^{p} \frac{W}{i} \cdot c_p(i, W)}{c(s, W_0)}.$$

By (13) and the above two equations,

$$\frac{G\_S_{p'}}{p'} \cdot \frac{\sum_{i=1}^{p'} \frac{W'}{i} \cdot c_{p'}(i, W')}{c(s, W_0)} = \frac{G\_S_p}{p} \cdot \frac{\sum_{i=1}^{p} \frac{W}{i} \cdot c_p(i, W)}{c(s, W_0)}. \tag{14}$$

For fixed speed,

$$\frac{W'}{p' \cdot \sum_{i=1}^{p'} \frac{W'}{i} \cdot c_{p'}(i, W')} = \frac{W}{p \cdot \sum_{i=1}^{p} \frac{W}{i} \cdot c_p(i, W)}.$$

By equation (13),

$$\sum_{i=1}^{p'} \frac{W'}{i} \cdot c_{p'}(i, W') = \sum_{i=1}^{p} \frac{W}{i} \cdot c_p(i, W). \tag{15}$$

Substituting (15) into (14), we have

$$\frac{G\_S_{p'}}{p'} = \frac{G\_S_p}{p}.$$

For $p = 1$,

$$G\_S_{p'} = p' \cdot G\_S_p. \tag{16}$$

Equation (16) is the corresponding unitary speedup when G_S_1 is not equal to one. If the work W equals W_0, then $G\_S_1 = 1$ and (16) becomes

$$G\_S_{p'} = p',$$

which is the unitary speedup defined in Definition 1.

If the fixed-time generalized speedup is unitary, then for any number of processors, p and p', and the corresponding problem sizes, W and W', where W' is the scaled problem size under the fixed-time constraint, we have

$$\frac{W \cdot c(s, W_0)}{\sum_{i=1}^{p} \frac{W}{i} \cdot c_p(i, W)} = p,$$

and

$$\frac{W' \cdot c(s, W_0)}{\sum_{i=1}^{p'} \frac{W'}{i} \cdot c_{p'}(i, W')} = p'.$$

Therefore,

$$\frac{W}{p \cdot \sum_{i=1}^{p} \frac{W}{i} \cdot c_p(i, W)} = \frac{W'}{p' \cdot \sum_{i=1}^{p'} \frac{W'}{i} \cdot c_{p'}(i, W')}.$$

The average speed is maintained. Also since

$$\sum_{i=1}^{p} \frac{W}{i} \cdot c_p(i, W) = \sum_{i=1}^{p'} \frac{W'}{i} \cdot c_{p'}(i, W'),$$

we have the equality

$$\frac{W}{p} = \frac{W'}{p'}.$$

The scalability (12) equals one. □

The following theorem gives the relation between memory-bounded speedup and fixed-time speedup. The theorem is for generalized speedup. However, based on Lemma 1, the result is true for traditional speedup when uniprocessor cost is fixed or the simple analysis model is used.

THEOREM 2. *If problem size increases proportionally to the number of processors in memory-bounded scaleup, then memory-bounded generalized speedup is linear if and only if fixed-time generalized speedup is linear.*

PROOF. Let $c(s, W_0)$, $c_p(i, W)$, W, and W_i be as defined in Theorem 1. Let W', W^* be the scaled problem size of fixed-time and memory-bounded scaleup, respectively, and W_i' and W_i^* be defined accordingly.

If memory-bounded speedup is linear, we have

$$\frac{W \cdot c(s, W_0)}{\sum_{i=1}^{p} \frac{W}{i} \cdot c_p(i, W)} = a \cdot p,$$

and

$$\frac{W^* \cdot c(s, W_0)}{\sum_{i=1}^{p'} \frac{W^*}{i} \cdot c_{p'}(i, W^*)} = a \cdot p',$$

for some constant $a > 0$. Combine the two equations, we have the equation

$$\frac{W}{p \cdot \sum_{i=1}^{p} \frac{W}{i} \cdot c_p(i, W)} = \frac{W^*}{p' \cdot \sum_{i=1}^{p'} \frac{W^*}{i} \cdot c_{p'}(i, W^*)}. \quad (17)$$

By assumption, W^* is proportional to the number of processors available,

$$W^* = \frac{p'}{p} \cdot W. \quad (18)$$

Substituting (18) into (17), we get the fixed-time equality:

$$\sum_{i=1}^{p'} \frac{W^*}{i} \cdot c_{p'}(i, W^*) = \sum_{i=1}^{p} \frac{W}{i} \cdot c_p(i, W). \quad (19)$$

That is $W' = W^*$, and the fixed-time generalized speedup is linear.

If fixed-time speedup is linear, then, following similar deductions as used for (17), we have

$$\frac{W}{p \cdot \sum_{i=1}^{p} \frac{W}{i} \cdot c_p(i, W)} = \frac{W'}{p' \cdot \sum_{i=1}^{p'} \frac{W'}{i} \cdot c_{p'}(i, W')}. \quad (20)$$

Applying the fixed-time equality (19) to (20), we have the reduced equation

$$W' = \frac{p'}{p} \cdot W. \quad (21)$$

With the assumption (18), (21) leads to

$$W^* = W',$$

and memory-bounded generalized speedup is linear. □

The assumption of Theorem 2 is problem size (work) increases proportionally to the number of processors. The assumption is true for many applications. However, it is not true for dense matrix computation where the memory requirement is a square function of the order of the matrix and the computation is a cubic function of the order of the matrix. For this kind of computational intensive applications, in general, memory-bounded speedup will lead to a large speedup. The following corollaries are direct results of Theorem 1 and Theorem 2.

COROLLARY 1. *If problem size increases proportionally to the number of processors in memory-bounded scaleup, then memory-bounded generalized speedup is unitary if and only if fixed-time generalized speedup is unitary.*

COROLLARY 2. *If work increases proportionally with the number of processors, then scalability* (12) *equals one if and only if the memory-bounded generalized speedup is unitary.*

Since uniprocessor cost varies on shared virtual memory machines, the above theoretical results are not applicable to traditional speedup on shared virtual memory machines.

Finally, to complete our discussion on the superlinear speedup, there is a new cause of superlinearity for generalized speedup. The new source of superlinear speedup is called *profile shifting* [11], and is due to the problem size difference between sequential and parallel processing (see Fig. 1). An application may contain different work types. While problem size increases, some work types may increase faster than the others. When the work types with lower costs increase faster, superlinear speedup may occur. A superlinear speedup due to profile shifting was studied in [11].

IV. Experimental Results

In this section, we discuss the timing results for solving a scientific application on KSR-1 parallel computers. We first give a brief description of the architecture and the application, and then present the timing results and analyses.

A. The Machine

The KSR-1 computer discussed here is a representative of parallel computers with shared virtual memory. Fig. 3 shows the architecture of the KSR-1 parallel computer [14]. Each processor on the KSR-1 has 32 Mbytes of local memory. The CPU is a super-scalar processor with a peak performance of 40 Mflops in double precision. Processors are organized into different rings. The local ring (ring:0) can connect up to 32 processors, and a higher level ring of rings (ring:1) can contain up to 34 local rings with a maximum of 1,088 processors.

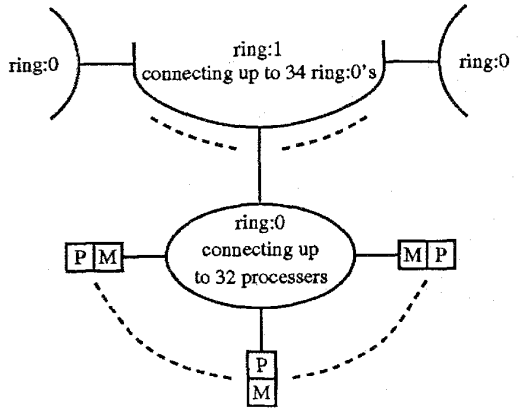


Fig. 3. Configuration of KSR-1 parallel computers (*P*: processor *M*: 32 Mbytes of local memory).

If a nonlocal data element is needed, the local search engine (SE:0) will search the processors in the local ring (ring:0). If the search engine SE:0 can not locate the data element within the local ring, the request will be passed to the search engine at the next level (SE:1) to locate the data. This is done automatically by a hierarchy of search engines connected in a fat-tree-like structure [14], [15]. The memory hierarchy of KSR-1 is shown in Fig. 4.

Each processor has 512 Kbytes of fast *subcache* which is similar to the normal cache on other parallel computers. This subcache is divided into two equal parts: an instruction subcache and a data subcache. The 32 Mbytes of local memory on

each processor is called a *local cache*. A local ring (ring:0) with up to 32 processors can have 1 Gbytes total of local cache which is called *Group:0 cache*. Access to the Group:0 cache is provided by Search Engine:0. Finally, a higher level ring of rings (ring:1) connects up to 34 local rings with 34 Gbytes of total local cache which is called *Group:1 cache*. Access to the Group:1 cache is provided by Search Engine:1. The entire memory hierarchy is called ALLCACHE memory by the Kendall Square Research. Access by a processor to the ALLCACHE memory system is accomplished by going through different Search Engines as shown in Fig. 4. The latencies for different memory locations [16] are:

- 2 cycles for *subcache*,
- 20 cycles for *local cache*,
- 150 cycles for *Group:0 cache*, and
- 570 cycles for *Group:1 cache*.

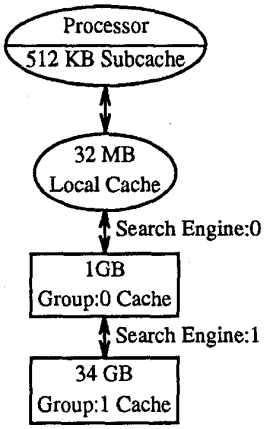


Fig. 4. Memory hierarchy of KSR-1.

B. The Application

Regularized least squares problems (RLSP) [17] are frequently encountered in scientific and engineering applications [18]. The major work is to solve the equation

$$(A^TA + \alpha \mathbf{I})\mathbf{x} = A^T\mathbf{b} \tag{22}$$

by orthogonal factorization schemes (Householder Transformations and Givens rotations). Efficient Householder algorithms have been discussed in [19] for shared memory supercomputers, and in [20] for distributed memory parallel computers.

Note that (22) can also be written as

$$\left(A^T, \sqrt{\alpha} I\right)\begin{pmatrix} A \\ \sqrt{\alpha} I \end{pmatrix} \mathrm{x} = \left(A^T, \sqrt{\alpha} I\right)\begin{pmatrix} b \\ 0 \end{pmatrix} \tag{23}$$

or

$$B^T B \mathrm{x} = B^T \begin{pmatrix} b \\ 0 \end{pmatrix}, \tag{24}$$

so that the major task is to carry out the QR factorization for matrix B which is neither a complete full matrix nor a sparse matrix. The upper part is full and the lower part is sparse (in diagonal form). Because of the special structure in B, not all elements in the matrix are affected in a particular step. Only a submatrix of $\mathbf{B}$ will be transformed in each step. If the columns of the submatrix $\mathbf{B}_i$ at step i are denoted by $\mathbf{B}_i = \left[\mathbf{b}_i^i \mathbf{b}_{i+1}^i \cdots \mathbf{b}_n^i\right]$, then the Householder Transformation can be described as:

Householder Transformation

Initialize matrix B
for $i = 1, n$

1. $\alpha_i = -sign\left(a_{ii}^{(i)}\right)\left(\mathbf{b}_i^{iT}\mathbf{b}_i^i\right)^{1/2}$
2. $\mathbf{w}_i = \mathbf{b}_i^i - \alpha_i \mathbf{e}_1$
3. $\beta_j = \mathbf{w}_i^T \mathbf{b}_j^i \left(\alpha_i^2 - \alpha_i a_{ii}^{(i)}\right), j = i+1, \cdots, n$
4. $\mathbf{b}_j^i = \mathbf{b}_j^i - \beta_j \mathbf{w}_i, \quad j = i+1, \cdots n$

end for

The calculation of β_js and updating of $\mathbf{b}_j^i$s can be done in parallel for different index j.

C. Timing Results

The numerical experiments reported here were conducted on the KSR-1 parallel computer installed at the Cornell Theory Center. There are 128 processors altogether on the machine. During the period when our experiments were performed; however, the computer was configured as two stand-alone machines with 64 processors each. Therefore, the numerical results were obtained using less than 64 processors.

Fig. 5 shows the traditional fixed-size speedup curves obtained by solving the regularized least squares problem with different matrix sizes n. The matrix is of dimensions $2n \times n$. We can see clearly that as the matrix size n increases, the speedup is getting better and better. For the case when n = 2,048, the speedup is 76 on 56 processors. Although it is well known that on most parallel computers, the speedup improves as the problem size increases, what is shown in Fig. 5 is certainly too good to be a reasonable measurement of the real performance of the KSR-1.

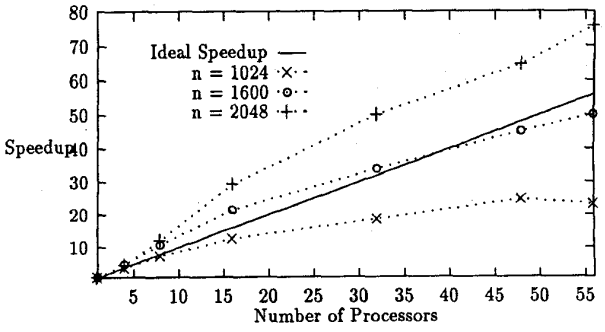


Fig. 5. Fixed-size (traditional) speedup on KSR-1.

The problem with the traditional speedup is that it is defined as the ratio of the sequential time to the parallel time used for solving the same fixed-size problem. The complex memory hierarchy on the KSR-1 makes the computational speed of a

single processor highly dependent on the problem size. When the problem is so big that not all data of the matrix can be put in the local memory (32 Mbytes) of the single computing processor, part of the data must be put in the local memory of other processors on the system. These data are accessed by the computing processor through Search Engine:0. As a result, the computational speed on a single processor slows down significantly due to the high latency of Group:0 cache. The sustained computational speed on a single processor is 5.5 Mflops, 4.5 Mflops and 2.7 Mflops for problem sizes 1,024, 1,600, and 2,048, respectively. On the other hand, with multiple processors, most of the data needed are in the local memory of each processor, so the computational speed suffers less from the high Group:0 cache latency. Therefore, the excellent speedups shown in Fig. 5 are the results of significant uniprocessor performance degradation when a large problem is solved on a single processor.

Fig. 6 shows the measured single processor speed as a function of problem size n. The Householder Transformation algorithm given before was implemented in KSR Fortran. The algorithm has a numerical complexity of $W = 2n^3 + 8.5n^2 + 26.5n$, and the speed is calculated using $s = W/t$ where t is the CPU time used to finish the computation.

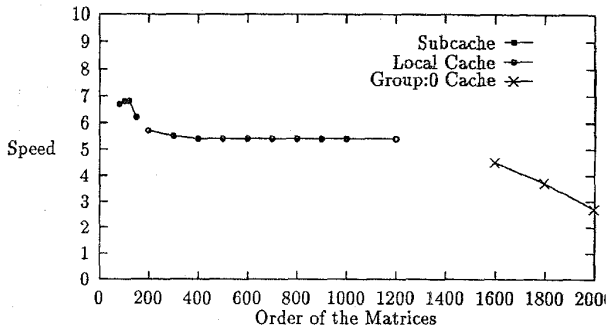


Fig. 6. Speed variation of uniprocessor processing on KSR-1.

As can be seen from Fig. 6, the three segments represent significantly different speeds for different matrix sizes. When the whole matrix can be fit into the subcache, the performance is close to 7 Mflops. The speed decreases to around 5.5 Mflops when the matrix can not be fit into the subcache, but still can be accommodated in the local cache. Note, however, when the matrix is so big that access to Group:0 cache through Search Engine:0 is needed, the performance degrades significantly and there is no clear stable performance level as can be observed in the other two segments. This is largely due to the high Group:0 cache latency and the contention for the Search Engine which is used by all processors on the machine. Therefore, the access time of Group:0 cache is less uniform as compared to that of the subcache and local cache.

To take the difference of single processing speeds for different problem sizes into consideration, we have to use the generalized speedup to measure the performance of multiple processors on the KSR-1. As can be seen from the definition of (6), the generalized speedup is defined as the ratio of the parallel speed to the asymptotic sequential speed, where the parallel speed is based on a scaled problem. In our numerical tests, the parallel problem was scaled in a memory-bounded fashion as the number of processors increases. The initial problem was selected based on the asymptotic speed (5.5 Mflops from Fig. 6) and then scaled proportionally according to the number of processors, i.e., with p processors, the problem is scaled to a size that will fill $M \times p$ Mbytes of memory, where M is the memory required by the unscaled problem. Fig. 7 shows the comparisons of the traditional scaled speedup and the generalized speedup. For the traditional scaled speedup, the scaled problem is solved on both one and p processors, and the value of the speedup is calculated as the ratio of the time of one processor to that of p processors. While for the generalized speedup, the scaled problem is solved only on multiple processors, not on a single processor. The value of the speedup is calculated using (6), where the asymptotic speed is used for the sequential speed. It is clear that Fig. 7 shows that the generalized speedup gives much more reasonable performance measurement on KSR-1 than does the traditional scaled speedup. With the traditional scaled speedup, the speedup is above 20 with only 10 processors. This excellent superlinear speedup is a result of the severely degraded single processors speed, rather than the perfect scalability of the machine and the algorithm.

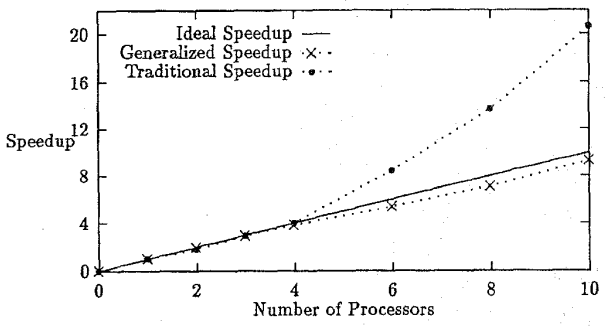


Fig 7. Comparison of generalized and traditional speedup on KSR-1.

Finally, Table I gives the measured isospeed scalability (see (12)) of solving the regularized least squares problem on a KSR-1 computer. The speed to be maintained on different number of processors is 3.25 Mflops, which is 60% of the asymptotic speed of 5.5 Mflops. The size of the $2n \times n$ matrix is increased as the number of processors increases. It starts as $n = 27$ on one processor and increases to $n = 2{,}773$ on 56 processors. One may notice that $\psi(2, 4) > \psi(1, 2)$ in Table I, which means that the machine-algorithm pair scales better from 2 processors to 4 processors than it does from one processor to two processors. This can be explained by the fact that on one processor, the matrix is small enough that all data can be accommodated in the subcache. Once all the data is loaded into the subcache, the whole computation process does not need data from local cache and Group:0 cache. Therefore, the data access time on one processor is significantly shorter than that on two processors which involves subcache, local cache

TABLE I
MEASURED SCALABILITY OF RLSP-KSR1 COMBINATION

$\psi(N, N')$	1	2	4	8	16	32	56
1	1.00000	0.12170	0.06871	0.01705	0.00374	0.00072	0.00006
2		1.00000	0.56464	0.14011	0.03077	0.00595	0.00050
4			1.00000	0.24814	0.05449	0.01054	0.00088
8				1.00000	0.21959	0.04247	0.00356
16					1.00000	0.19343	0.01621
32						1.00000	0.08378
56							1.00000

and Group:0 cache to pass messages. As a result, significant increase in the work W is necessary in the case of two processors to offset the extra data access time involving different memory hierarchies. This is the major reason for the low $\psi(1, 2)$ value. When the number of processors increases from 2 to 4, the data access pattern is the same for both cases with subcache, local cache and Group:0 cache all involved, so that the work W does not need to be increased significantly to offset the extra communication cost when going from 2 processors to 4 processors. It is interesting to notice, while the scalability of the RLSP-KSR1 combination is relatively low, the data in Table I has a similar decreasing pattern as the measured and computed scalability of Burg-nCUBE, SLALOM-nCUBE, Burg-MasPar, and SLALOM-MasPar combinations [12]. The scalabilities are all decreasing along columns and have some irregular behavior at $\psi(1, 2)$ and $\psi(2, 4)$.

Interested readers may wonder how the measured scalability is related to the measured generalized speedup given in Fig. 7. While Fig. 7 demonstrates a nearly linear generalized speedup, the corresponding scalability given in Table I is far from ideal (the ideal scalability would be unity). The low scalability is expected. Recall that the scaled speedup given in Fig. 7 is memory-bounded speedup [6]. That is when the number of processors is doubled, the usage of memory is also doubled. As a result, the number of elements in the matrix is increased by a factor of 2. Corollary 2 shows that if work W increases linearly with the number of processors, then unitary memory-bounded speedup will lead to ideal scalability. For the regularized least squares application, however, the work W is a cubic function of the matrix size n. When the memory usage is doubled, the number of floating point operations is increased by a factor of eight. If a perfect generalized speedup is achieved from p to $p' = 2p$, the average speed at p and p' should be the same. By (12) we have

$$\psi(p, p') = \frac{2pW}{8pW} = \frac{1}{4}.$$

With the measured speedup being a little lower than unitary as shown in Fig. 7, a less than 0.25 scalability is expected. Table I confirms this relation, except at $\psi(2, 4)$ for the reason pointed out earlier. The scalability in the last column is noticeably lower than other columns. It is because when 56 nodes are involved in computations, communication has to pass through ring:1, which slows down the communication significantly.

Computation intensive applications have often been used to achieve high flops. The RLSP application is a computation intensive application. Table I shows that isospeed scalability does not give credits for computation intensive applications. The computation intensive applications may achieve a high speed on multiple processors, but the initial speed is also high. The isospeed scalability measures the ability to maintain the speed, rather than to achieve a particular speed.

The implementation is conducted on a KSR-1 shared virtual memory machine. The theoretical and analytical results given in Section II and Section III, however, are general and can be applied on different parallel platforms. For instance, for Intel Paragon parallel computers, where virtual memory is supported to swap data in and out from memory to disk, we expect that inefficient sequential processing will cause similar superlinear (traditional) speedup as demonstrated on KSR-1. For distributed-memory machines which do not support virtual memory, such as CM-5, traditional speedup has another draw back. Due to memory constraint, scaled problems often cannot be solved on a single processor. Therefore, scaled speedup is unmeasurable. Defining asymptotic speed similarly as given in Section III, the generalized speedup can be applied to this kind of distributed-memory machines to measure scalable computations. Generalized speedup is defined as parallel speed over sequential speed. Given a reasonable initial sequential speed, it can be used on any parallel platforms to measure the performance of scalable computations.

V. CONCLUSION

Since the scaled up principle was proposed in 1988 by Gustafson and other researchers at Sandia National Laboratory [21], the principle has been widely used in performance measurement of parallel algorithms and architectures. One difficulty of measuring scaled speedup is that very large problems have to be solved on uniprocessor, which is very inefficient if virtual memory is supported, or is impossible otherwise. To overcome this shortcoming, generalized speedup was proposed [1]. Generalized speedup is defined as parallel speed over sequential speed and does not require solving large problems on uniprocessor. The study [1] emphasized the fixed-time generalized speedup, *sizeup*. To meet the need of the emerging shared virtual memory machines, the generalized speedup, particularly implementation issues, has been carefully studied in the current research. It has shown that traditional speedup is a special case of generalized speedup, and, on the other hand, generalized

speedup is a reform of traditional speedup. The main difference between generalized speedup and traditional speedup is how to define the uniprocessor efficiency. When uniprocessor speed is fixed these two speedups are the same. Extending these results to scalability study, we have found that the difference between isospeed scalability [12] and isoefficiency scalability [13] is also due to the uniprocessor efficiency. When the uniprocessor speed is independent of the problem size, these two proposed scalabilities are the same. As part of the performance study, we have shown that an algorithm-machine combination achieves a perfect scalability if and only if it achieves a perfect speedup. An interesting relation between fixed-time and memory-bounded speedups is revealed. Seven causes of superlinear speedup are also listed.

A scientific application has been implemented on a Kendall Square KSR-1 shared virtual memory machine. Experimental results show that uniprocessor efficiency is an important issue for virtual memory machines, and that the asymptotic speed provides a reasonable way to define the uniprocessor efficiency.

The results in this paper on shared virtual memory can be extended to general parallel computers. Since uniprocessor efficiency is directly related to parallel execution time, scalability, and benchmark evaluations, the range of applicability of the uniprocessor efficiency study is wider than speedups. The uniprocessor efficiency might be explored further in a number of contexts.

ACKNOWLEDGMENTS

This research was supported in part by the U.S. National Aeronautics and Space Administration under NASA contract NAS1-19480 and NAS1-1672.

The authors are grateful to the Cornell Theory Center for providing access to its KSR-1 parallel computer, and to the referees for their helpful comments on the revision of this paper.

REFERENCES

[1] X.-H. Sun and J. Gustafson, "Toward a better parallel performance metric," *Parallel Computing*, vol. 17, pp. 1,093-1,109, Dec. 1991.

[2] K. Hwang, *Advanced Computer Architecture: Parallelism, Scalability, Programmability*, McGraw-Hill, 1993.

[3] J. Ortega and R. Voigt, "Solution of partial differential equations on vector and parallel computers," *SIAM Rev.*, pp. 149-240, June 1985.

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

[5] J. Gustafson, "Reevaluating Amdahl's law," *Comm. ACM*, vol. 31, pp. 532-533, May 1988.

[6] X.-H. Sun and L. Ni, "Scalable problems and memory-bounded speedup," *J. Parallel and Distributed Computing*, vol. 19, pp. 27-37, Sept. 1993.

[7] D. Helmbold and C. McDowell, "Modeling speedup(n) greater than n," *IEEE Trans. Parallel and Distributed Systems*, pp. 250-256, Apr. 1990.

[8] D. Parkinson, "Parallel efficiency can be greater than unity," *Parallel Computing*, vol. 3, pp. 261-262, 1986.

[9] D. Nicol, "Inflated speedups in parallel simulations via malloc()," *Int'l J. Simulation*, vol. 2, pp. 413-426, Dec. 1992.

[10] X.-H. Sun and J. Zhu, "Performance prediction of scalable computing: A case study," *Proc. 28th Hawaii Int'l Conf. of Systems Sciences*, pp. 456-465, Jan. 1995.

[11] J. Gustafson, D. Rover, S. Elbert, and M. Carter, "The design of a scalable, fixed-time computer benchmark," *J. Parallel and Distributed Computing*, vol. 12, no. 4, pp. 388-401, 1991.

[12] X.-H. Sun and D. Rover, "Scalability of parallel algorithm-machine combinations," *IEEE Trans. Parallel and Distributed Systems*, pp. 599-613, June 1994.

[13] A.Y. Grama, A. Gupta, and V. Kumar, "Isoefficiency: Measuring the scalability of parallel algorithms and architectures," *IEEE Parallel and Distributed Technology*, vol. 1, pp. 12-21, Aug. 1993.

[14] Kendall Square Research, "KSR parallel programming," Waltham, Mass., 1991.

[15] C. Leiserson, "Fat-trees: Universal networks for hardware-efficient super-computing," *IEEE Trans. Computers*, vol. 34, no. 10, pp. 892-901, 1985.

[16] Kendall Square Research, "KSR technical summary," Waltham, Mass., 1991.

[17] A.N. Tikhnov and V. Arsenin, *Solution of Ill-Posed Problems*. John Wiley and Sons, 1977.

[18] Y.M. Chen, J.P. Zhu, W.H. Chen, and M.L. Wasserman, "GPST inversion algorithms for history matching in 3D 2-phase simulators," *IMACS Trans. Scientific Computing I*, pp. 369-374, 1989.

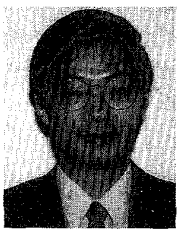
[19] J. Dongarra, I.S. Duff, C.D. Sorensen, and H.A. van der Vorst, *Solving Linear Systems on Vector and Shared Memory Computers*. Philadelphia: SIAM, 1991.

[20] A. Pothen and P. Raghavan, "Distributed orthogonal factorization: Givens and Householder algorithms," *SIAM J. Science and Statistical Computing*, vol. 10, pp. 1,113-1,135, 1989.

[21] J. Gustafson, G. Montry, and R. Benner, "Development of parallel methods for a 1,024-processor hypercube," *SIAM J. Science and Statistical Computing*, vol. 9, pp. 609-638, July 1988.

Xian-He Sun (S'88-M'90-SM'95) received the BS degree in mathematics from Beijing Normal University, Beijing, China, and the MS degree in mathematics and MS and PhD degrees in computer science, the latter three from Michigan State University. He joined the Ames Laboratory, operated for the U.S. Department of Energy by Iowa State University; was a visiting faculty member at Clemson University; and was a staff scientist at ICASE, NASA Langley Research Center. He has been with the Department of Computer Science at Louisiana State University since January 1994. His research interests include parallel processing, parallel numerical algorithms, performance evaluation, and database systems.

Dr. Sun was a guest editor for the special issue of the *Journal of Parallel and Distributed Computing* on Analyzing Scalability of Parallel Algorithms and Architectures. He is a member of the IEEE, ACM, and Phi Kappa Phi.



Jianping Zhu received the BS degree in engineering mechanics in 1982 from Zhejiang University, China; the MS degree in computational mechanics in 1984 from Dalian Institute of Technology, China; and the PhD degree in applied mathematics in 1990 from the State University of New York, Stony Brook. He joined the Department of Mathematics and Statistics and the U.S. National Science Foundation Engineering Research Center at Mississippi State University in 1990 as an assistant professor and has been an associate professor there since 1993. He received the second-place award in the 1990 IBM supercomputing competition and the Intel University Research Partners Fellowship Award in 1992.

Dr. Zhu has written more than 30 refereed publications in scientific computing and a book *Solving Partial Differential Equations on Parallel Computers*, published in 1994 by World Scientific Publishing. His major research interests include numerical methods for solving PDEs, parallel computing, and large-scale simulations. He is a member of the AMS, SIAM and AIAA.