Shared Virtual Memory and Generalized Speedup [*]

Xian-He Sun

ICASE
Mail Stop 132C
NASA Langley Research Center
Hampton, VA 23681-0001

Jianping Zhu

NSF Engineering Research Center
Dept. of Math. and Stat.
Mississippi State University
Mississippi State, MS 39762

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, various causes of superlinear speedup are also presented.

[*]This research was supported by the National Aeronautics and Space Administration under NASA contract NAS1-19480 while the first author was in residence at the Institute for Computer Applications in Science and Engineering (ICASE), NASA Langley Research Center, Hampton, VA 23681-0001.

1 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 we target recent development of shared virtual memory machines and revisit the *generalized speedup* [17] performance metric.

Distributed-memory parallel computers dominate today's parallel computing arena. These machines, such as the Kendall Square KSR-1, Intel Paragon, and TMC CM-5, have successfully delivered high performance computing power for solving certain 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 program paradigm. 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. 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. 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 non-uniform [8]. 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 new performance metric proposed in [17]. In this paper, we revisit generalized speedup and address the measurement issues. 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, scalability, are also studied. 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 2 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 memory access is, the larger the speedup. Generalized speedup is studied in Section 3. 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. Efficiency and scalability issues are also discussed. Experimental results of a production application on a Kendall Square KSR-1 parallel computer are given in Section 4. Section 5 contains a summary.

2 The Traditional Speedup

One of the best accepted and the most frequently used performance metrics in parallel processing is speedup. It measures the parallel processing gain over sequential processing and 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* [14]. Absolute speedup is an important metric, especially when new parallel algorithms are introduced. Another widely used interpretation is the *relative speedup* [14], 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. The well known Amdahl's law [1] and Gustafson's scaled speedup [4] are both based on relative speedup. 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.

The absolute speedup and the relative speedup are distinguished by the sequential algorithm. After a sequential algorithm is chosen, from the problem size point of view, speedup can be further 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 is based on the fixed-size speedup. With one parameter, the sequential processing ratio, Amdahl's law gives the limitation of 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* and the *memory-bounded speedup* [18]. When p processors are used, 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. Corresponding to Amdahl's law, Gustafson has given a simple fixed-time speedup formula [5]. The memory-bounded speedup [18] is another practically used scaled speedup. It is defined in a similar way to the fixed-time speedup. The difference is that in memory-bounded speedup the problem size is scaled based on the available memory, while in fixed-time speedup the problem size is scaled up to meet the fixed execution time. A detailed study of the memory-bounded speedup can be found in [18].

Speedup can also be classified based on the achieved performance. Let p and S_p be the number of processors and the speedup with p processors. The following terms were used in [7].

Definition 1

- *Super-linear speedup:* $\lim_{p \to \infty} \frac{S_p}{p} = \infty$
- *Linear super-unitary speedup:* $p < S_p < c \cdot p$ *for some constant* $c > 1$.
- *Unitary speedup:* $S_p = p$.
- *Linear sub-unitary speedup:* $\epsilon \cdot p < S_p < p$ *for some positive constant* $\epsilon < 1$.
- *Sub-linear speedup:* $\lim_{p \to \infty} \frac{S_p}{p} = 0$.

We say a speedup is a superlinear speedup if it is either super-linear or linear super-unitary. It is debatable if any machine-algorithm pair can achieve "truly" superlinear speedup. Four possible causes of superlinear speedup given in [7] are listed in Fig. 1.

1. cache size increased in parallel processing
2. overhead reduced in parallel processing
3. latency hidden in parallel processing
4. Randomized algorithms

Figure 1. Causes of Superlinear Speedup: part 1

Cause 2 in Fig. 1 can be considered theoretically [15], 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. 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. Two other causes of superlinear relative speedup and scaled speedup are listed in Fig. 2.

5. mathematical inefficiency of the serial algorithm
6. higher memory access latency in the sequential processing

Figure 2. Causes of Superlinear Speedup: part 2

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 [13].

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. Figure 7 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:

$$\textit{Sequential execution time} = \textit{Amount of work} \times \frac{\textit{Processor cycles per unit of work}}{\textit{Machine clock rate}}. \tag{1}$$

The ratio in the right hand side of Eq. (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 [17]. 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, 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

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

If $c_p(i, W) = c_p(p, W)$, for $1 \leq i < p$, then

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

The first ratio of Eq. (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 [18]. 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. 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

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

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 Eq. (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 4). 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.

3 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* [17] is one such metric.

$$\text{Generalized Speedup} = \frac{\text{Parallel Speed}}{\text{Sequential Speed}}. \qquad (6)$$

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 sometimes call the traditional speedup the speedup, and call the speedup given in Eq. (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* [17]. The fixed-time benchmark SLALOM [6] 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. Figure 3 depicts the typical cost changing pattern.

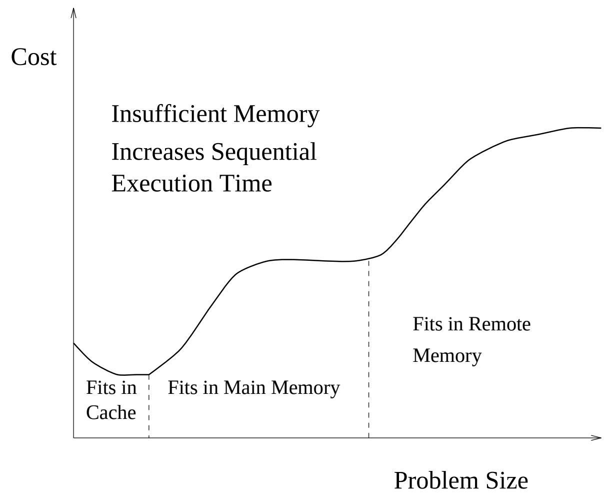

Figure 3. Cost Variation Pattern.

From Eq. (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 Eq. (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 2, then $c(s, W_0)/c_p(p, W_0) = 1$. By Eq. (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. So we have

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

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 Eq. (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

$$\textit{Efficiency} = \frac{\text{speedup}}{\text{number of processors}}. \tag{8}$$

The *Generalized efficiency* can be defined similarly as

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

By definition,

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

and

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

Equations (10) and (11) show the difference between the two efficiencies. The traditional efficiency assumes that the measured sequential processing achieves hundred percent efficiency. The generalized efficiency assumes that the sequential processing based on the asymptotic cost achieves hundred percent efficiency. 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 propositions are direct results of Eq.(7).

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

Proposition 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 Proposition 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 [19]. 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

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

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

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

Since the sequential cost is fixed in Eq. (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 influency 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 by Kumar [11, 8].

Proposition 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 Eq. (7).

If scalability (12) equals 1, let W', p' be as defined in Eq. (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 the definition of generalized speedup, generalized speedup

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

With some arithmetic manipulation, we have

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

Similarly, we have

$$\frac{W}{p} = \frac{G\_S_p}{p} \cdot \frac{\sum_i^{p} \frac{W_i}{i} \cdot c_{p}(i, W)}{c(s, W_0)}.$$

By Eq. (13) and the above two equations,

$$\frac{G\_S_{p'}}{p'} \cdot \frac{\sum_i^{p'} \frac{W'_i}{i} \cdot c_{p'}(i, W')}{c(s, W_0)} = \frac{G\_S_p}{p} \cdot \frac{\sum_i^{p} \frac{W_i}{i} \cdot c_{p}(i, W)}{c(s, W_0)}.$$

For fixed-time speedup

$$\sum_i^{p'} \frac{W'_i}{i} \cdot c_{p'}(i, W') = \sum_i^{p} \frac{W_i}{i} \cdot c_p(i, W).$$

Thus,

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

For $p = 1$,

$$G\_S_{p'} = p' \cdot G\_S_p. \tag{14}$$

Equation (14) 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 Eq. (14) 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

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

and

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

Therefore,

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

The average speed is maintained. Also since

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

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 Proposition 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 unitary if and only if fixed-time generalized speedup is unitary.*

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 unitary, we have

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

and

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

Combine the two equations, we have the equation

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

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

$$W^* = \frac{p'}{p} \cdot W. \tag{16}$$

Substituting Eq. (16) into Eq. (15), we get the fixed-time equality:

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

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

If fixed-time speedup is unitary, then, following similar deductions as used for Eq. (15), we have

$$\frac{W}{p \cdot \sum_i^p \frac{W}{i} \cdot c_p(i, W)} = \frac{W'}{p' \cdot \sum_i^{p'} \frac{W'}{i} \cdot c_{p'}(i, W')}. \tag{18}$$

Applying the fixed-time equality Eq. (17) to Eq. (18), we have the reduced equation

$$W' = \frac{p'}{p} \cdot W. \tag{19}$$

With the assumption Eq. (16), Eq. (19) leads to

$$W^* = W',$$

and memory-bounded generalized speedup is unitary. □

The following corollary is a direct result of Theorem 1 and Theorem 2.

Corollary 1 *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.*

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* [6], and is due to the problem size difference between sequential and parallel processing. 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 [6].

7. profile shifting

Figure 4. Causes of Superlinear Speedup: part 3

4 Experimental Results

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

4.1 The Machine

The machine to be discussed here can be viewed as a combination of (or a compromise between) the distributed and shared memory parallel architectures. Their hybrid is called the *Shared Virtual Memory* architecture. A representative of this category is the new KSR-1 parallel computer from Kendall Square Research. It has distributed physical memory which makes the system scalable to a large number of processors, and a shared address space which provides users a shared-memory-like programming environment.

Figure 5 shows the architecture of the KSR-1 parallel computer [9]. 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 1088 processors.

If a non-local 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.

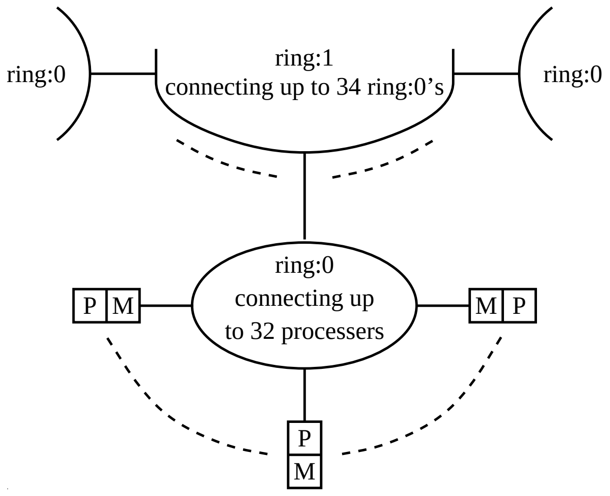

Figure 5. Configuration of KSR-1 parallel computers.

P: processor M: 32 Mbytes of local memory

This is done automatically by a hierarchy of search engines connected in a fat-tree-like structure [9, 12]. The memory hierarchy of KSR-1 is shown in Fig. 6.

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. 6. The latencies for different memory locations [10] are: 2 cycles for *subcache*, 20 cycles for *local cache*, 150 cycles for *Group:0 cache*, and 570 cycles for *Group:1 cache*.

4.2 The Application

Least squares problems are frequently encountered in scientific and engineering applications. The major work of solving least squares problems is to solve the normal equation

$$\mathbf{A}^{\mathbf{T}}\mathbf{A}\mathbf{x} = \mathbf{A}^{\mathbf{T}}\mathbf{b} \tag{20}$$

by orthogonal factorization schemes (Householder Transformations and Givens rotations). Efficient Householder algorithms have been discussed in [3] for shared memory supercomputers, and in [16]

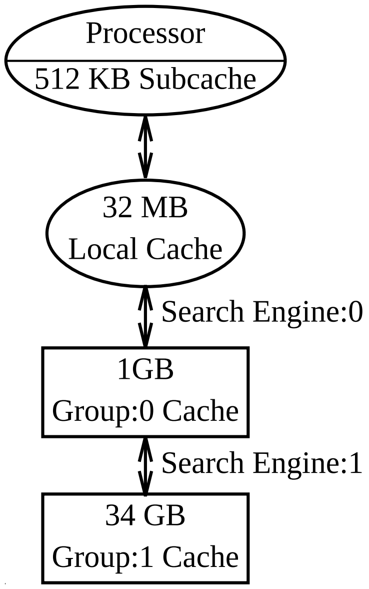

Figure 6. Memory hierarchy of KSR-1.

for distributed memory parallel computers.

In many cases, for instance the inverse problem of partial differential equations [2], the normal equation system resulting from the discretization is too ill-conditioned to be solved directly. Tikhnov's regularization method [20] is frequently used in this case to increase numerical stability. The key step in this process is to introduce a regularization factor $\alpha > 0$. Instead of solving (20) directly, we solve the following system

$$(\mathbf{A}^{\mathbf{T}}\mathbf{A} + \alpha\mathbf{I})\mathbf{x} = \mathbf{A}^{\mathbf{T}}\mathbf{b} \tag{21}$$

for $\mathbf{x}$. Eq. (21) can also be written as

$$(A^T, \sqrt{\alpha}I)\begin{pmatrix} A \\ \sqrt{\alpha}I \end{pmatrix}\mathbf{x} = (A^T, \sqrt{\alpha}I)\begin{pmatrix} b \\ 0 \end{pmatrix} \tag{22}$$

or

$$B^T B\mathbf{x} = B^T \begin{pmatrix} b \\ 0 \end{pmatrix}, \tag{23}$$

so that the major task is to carry out the QR factorization for matrix B which has the structure

$$B = \left[\begin{array}{cccc} \multicolumn{4}{c}{\boxed{\begin{array}{cccc} a^{(1)}_{11} & a^{(1)}_{12} & \cdots & a^{(1)}_{1n} \\ \vdots & \vdots & \vdots & \vdots \\ a^{(1)}_{m1} & a^{(1)}_{m2} & \cdots & a^{(1)}_{mn} \\ \sqrt{\alpha} & & & \end{array}}} \\ & \sqrt{\alpha} & & \\ & & \ddots & \\ & & & \sqrt{\alpha} \end{array}\right], \tag{24}$$

where we usually have $m \geq n$ with m of the same order as n. Matrix B 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 (24), not all elements in the matrix are affected in a particular transformation step. In the first step, all elements within the frame in matrix (24) will be affected. In each new step, the frame in (24) will shift downwards one row with the left most column out of the game. Therefore, at the ith step, the submatrix $\mathbf{B}_i$ affected in the transformation has the form:

$$B_i = \left[\begin{array}{cccc} a^{(i)}_{ii} & \cdots & \cdots & a^{(i)}_{in} \\ \vdots & \vdots & \vdots & \vdots \\ a^{(i)}_{m+i-1,i} & \cdots & \cdots & a^{(i)}_{m+i-1,n} \\ \sqrt{\alpha} & 0 & \cdots & 0 \end{array}\right]. \tag{25}$$

If the columns of matrix $\mathbf{B}_i$ of (25) are denoted by $\mathbf{b}^i_j$, i.e.

$$\mathbf{B}_i = [\mathbf{b}^i_i \; \mathbf{b}^i_{i+1} \cdots \mathbf{b}^i_n], \tag{26}$$

then the Householder Transformation can be described as:

```
Householder Transformation

Initialize matrix $B$
for $i = 1, n$
    1. $\alpha_i = -sign(a_{ii}^{(i)})(\mathbf{b}_i^{i^T}\mathbf{b}_i^i)^{1/2}$
    2. $\mathbf{w}_i = \mathbf{b}_i^i - \alpha_i \mathbf{e}_1$
    3. $\beta_j = \mathbf{w}_i^T \mathbf{b}_j^i(\alpha_i^2 - \alpha_i a_{ii}^{(i)}), \quad j = i+1, \cdots, n$
    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_j's and updating of $\mathbf{b}_j^i$'s can be done in parallel for different index j.

4.3 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.

Figure 7 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 $n = 2048$, 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. 7 is certainly too good to be a reasonable measurement of the real performance of the 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 1024, 1600 and 2048 respectively. On the other hand, with multiple processors, most of the data needed are in the local

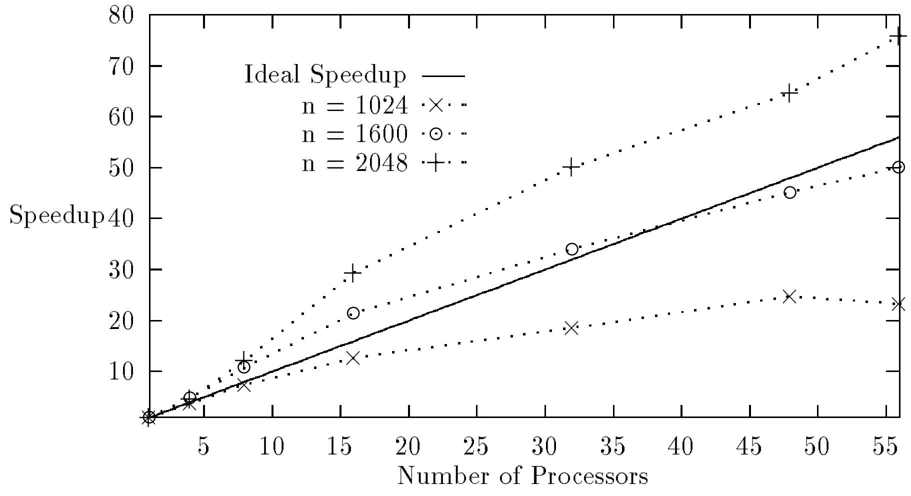

Figure 7. Fixed-size (Traditional) Speedup on KSR-1

memory of each processor, so the computational speed suffers less from the high Group:0 cache latency. Therefore, the excellent speedups shown in Fig. 7 are the results of significant uniprocessor performance degradation when a large problem is solved on a single processor.

Figure 8 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.

As can be seen from Fig. 8, 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 Eq. (6), the generalized speedup is defined as the ratio of the parallel speed to the asymptotic sequential speed, where the parallel speed is

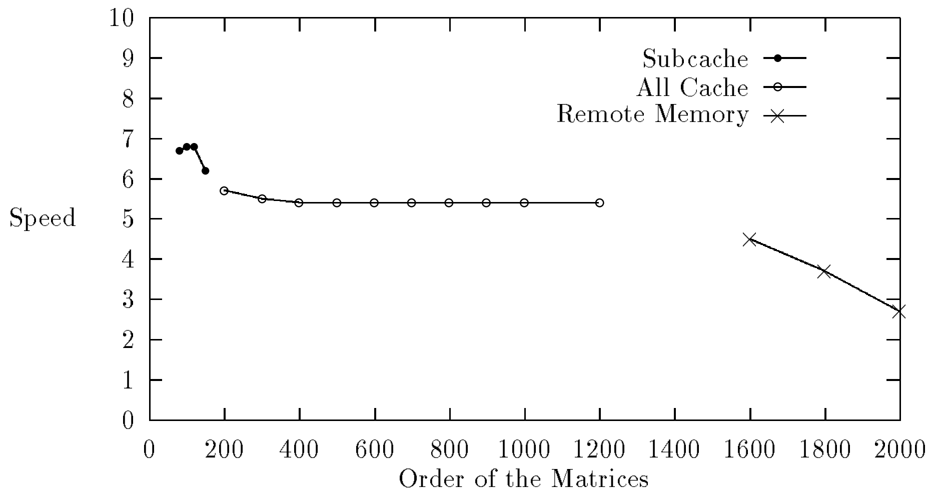

Figure 8. Speed Variation of Uniprocessor Processing on KSR-1

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. 8) 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. Figure 9 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 Eq. (6), where the asymptotic speed is used for the sequential speed. It is clear that Fig. 9 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.

5 Conclusion

Since the scaled up principle was proposed in 1988 by Gustafson and other researchers at Sandia National Laboratory [5], the principle has been widely used in performance measurement of parallel algorithms and architectures. One difficulty of measuring scaled speedup is that vary large problems

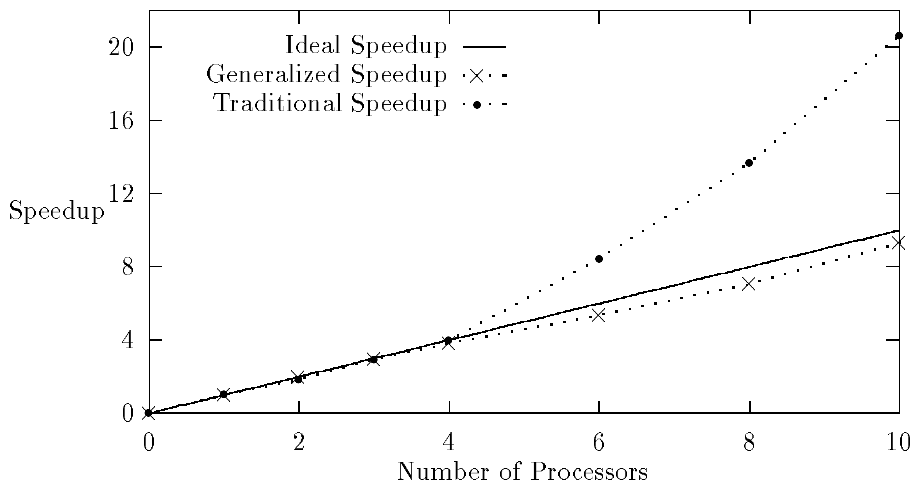

Figure 9. Comparison of Generalized and Traditional Speedup on KSR-1

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 and studied by Gustafson and Sun [17]. Generalized speedup is defined as parallel speed over sequential speed and does not require solving large problems on uniprocessor. The study [17] 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 [19] and isoefficiency scalability [11] 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. 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.

Acknowledgement

The authors are grateful to the Cornell Theory Center for providing access to its KSR-1 parallel computer.

References

[1] AMDAHL, G. Validity of the single-processor approach to achieving large scale computing capabilities. In *Proc. AFIPS Conf.* (1967), pp. 483–485.

[2] CHEN, Y. M., ZHU, J. P., CHEN, W. H., AND WASSERMAN, M. L. GPST inversion algorithm for history matching in 3-d 2-phase simulators. In *IMACS Trans. on Scientific Computing I* (1989), pp. 369–374.

[3] DONGARRA, J., DUFF, I. S., SORENSEN, D. C., AND VAN DER VORST, H. A. *Solving Linear Systems on Vector and Shared Memory Computers.* SIAM, Philadelphia, 1991.

[4] GUSTAFSON, J. Reevaluating Amdahl's law. *Communications of the ACM 31* (May 1988), 532–533.

[5] GUSTAFSON, J., MONTRY, G., AND BENNER, R. Development of parallel methods for a 1024-processor hypercube. *SIAM J. of Sci. and Stat. Computing 9*, 4 (July 1988), 609–638.

[6] GUSTAFSON, J., ROVER, D., ELBERT, S., AND CARTER, M. The design of a scalable, fixed-time computer benchmark. *J. of Parallel and Distributed Computing 12*, 4 (1991), 388–401.

[7] HELMBOLD, D., AND MCDOWELL, C. Modeling speedup(n) greater than n. In *Proc. of the 1989 Int'l Conf. on Parallel Processing, Vol. III* (1989), pp. 219–225.

[8] HWANG, K. *Advanced Computer Architecture: Parallelism, Scalability, Programmability.* McGraw-Hill Book Co., 1993.

[9] KENDALL SQUARE RESEARCH. KSR parallel programming. Waltham, USA, 1991.

[10] KENDALL SQUARE RESEARCH. KSR technical summary. Waltham, USA, 1991.

[11] KUMAR, V., AND GUPTA, A. Analysis of scalability of parallel algorithms and architectures: A survey. In *Proc. of 1991 Int'l Conf. on Supercomputing* (June 1991), pp. 396–405.

[12] LEISERSON, C. Fat-trees: Universal networks for hardware-efficient supercomputing. *IEEE Transactions on Computing 34*, 10 (1985), 892–901.

[13] NICOL, D. Inflated speedups in parallel simulations via `malloc()`. *International Journal on Simulation 2* (Dec. 1992), 413–426.

[14] ORTEGA, J., AND VOIGT, R. Solution of partial differential equations on vector and parallel computers. *SIAM Review* (June 1985), 149–240.

[15] Parkinson, D. Parallel efficiency can be greater than unity. *Parallel Computing 3* (1986), 261–262.

[16] Pothen, A., and Raghavan, P. Distributed orthogonal factorization: Givens and Householder algorithms. *SIAM J. of Sci. and Stat. Computing 10* (1989), 1113–1135.

[17] Sun, X.-H., and Gustafson, J. Toward a better parallel performance metric. *Parallel Computing 17* (Dec 1991), 1093–1109.

[18] Sun, X.-H., and Ni, L. Scalable problems and memory-bounded speedup. *J. of Parallel and Distributed Computing 19* (Sept. 1993), 27–37.

[19] Sun, X.-H., and Rover, D. Scalability of parallel algorithm-machine combinations. *IEEE Transactions on Parallel and Distributed Systems* (1994). to appear.

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