Performance Prediction

A Case Study Using a Scalable Shared-Virtual-Memory Machine

Xian-He Sun
Louisiana State University
Jianping Zhu
Mississippi State University

A simple formula shows the relationship between scalability, single-processor computing power, and degradation of parallelism. Starting with this formula, the authors investigate the prediction and application of scalability.

Recent trends in parallel processing suggest that the issue of performance prediction is becoming more complex and difficult. Scientists have adopted massively parallel computing as a cost-effective way to achieve high computing power. They have introduced various architectures and algorithms to deliver performance scalability with many processors. Shared virtual memory and other kinds of system support, which hide the communication and other implementation details from the user, are becoming more prevalent. At the same time, with various architectures and algorithms available, performance prediction is becoming critical in choosing an appropriate algorithm-machine pair for an application, especially when the machine has a sophisticated hierarchical architecture.

In this article, we combine simple formulas with runtime information to predict performance in modern parallel computers. After presenting a simple prediction formula, we discuss a case study involving a virtual memory machine to illustrate how to use the formula in practice. We discuss four different aspects:

- We propose a method to measure the needed runtime parameters.
- We propose an adjustment to catch the influence of architecture variation when the system size is scaled up from one level of architecture hierarchy to another.
- We demonstrate, through the case study, that it is possible to predict the influence of architecture hierarchy on scalability by simply using hardware specifications.
- Finally, we discuss the issue of choosing an appropriate algorithm for a given application when the computing system is scaled up from one level of hierarchy to another.

The challenge of performance prediction

There are two commonly used synchronization and communication models: message passing and shared memory. Processes communicate through explicit message passing in the message-passing model and through shared variables in the shared-memory model. Traditionally, message passing has been the natural choice of distributed-memory machines. With shared virtual address space, shared virtual memory can be supported on distributed-memory machines, but requires sophisticated hardware and system support. Shared-virtual-memory machines are scalable and provide a sequential-like programming environment. However, performance prediction for shared-virtual-memory machines is more difficult than for traditional message-passing machines, because communication is implicit and memory access time is nonuniform.

Parallel machines can yield very high raw computation power. However, the high computation power might not be realized in solving a given application, because the achievable efficiency of an application can drop quickly as system size increases. To evaluate the ability of maintaining performance, researchers have proposed several metrics to measure the scalability of algorithm-machine combinations.[1-4] Isospeed scalability, which measures the ability of an algorithm-machine combination to maintain unit processor speed, is one of the proposed metrics. In parallel computing, the sustained performance is typically only a small portion of the hardware peak performance and can vary significantly with the problem size and system size. Performance tuning, architectural improvement, algorithm and architecture selection, and compiler optimization are critical steps in improving the efficiency of parallel computers. Performance prediction is an especially essential component of the last two steps.[5,6] Currently, the challenge of performance prediction is to predict the performance variation of scalable computing, where the machine ensemble size and problem size vary over a wide range.[7] (The "Definitions and applications" sidebar describes the motivation and application of the scalability study in more detail.)

Performance models are developed in terms of execution time and scalability. Experimental results on a 64-node Kendall Square KSR-1 show that when performance information about a small-scale system is available, the performance of a larger-scale system can be predicted. Thus, we can compare machine architectures and algorithms in terms of scalability, without runtime information. Our proposed formula and methodology are not bound to any particular algorithm or architecture. Because a 64-node KSR-1 is a shared-virtual-memory machine with variable memory access times, the implementation experience learned in this study is reasonably general and should be applicable to a class of applications on similar architectures.

Definition and analysis

One of the main motivations of parallel processing is to solve large problems fast. Considering both execution time and problem size, what we seek from parallel processing is speed, which is defined as work divided by time. In general, how work should be defined is controversial. For scientific applications, it is commonly agreed that the floating-point operation (flop) count is a good estimate of work problem size. Some authors refer to problem size as the parameter that determines the workload—for instance, the order of matrices. In this article, problem size refers to the work to be performed, and we will use "problem size" and "work" interchangeably. The average unit speed is a good measure of parallel processing speed.

Definition 1: The *average unit speed* (or average speed, for short) is the achieved speed of the given computing system divided by p, the number of processors.

Ideally, average speed remains constant when system size increases. The hardware peak performance specified by vendors is usually based on this ideal assumption. However, if problem size is fixed, the ideal situation is unlikely. The reason is that for a problem with a fixed size, the communication/computation ratio is likely to increase with the number of processors, so the average speed decreases with increased system size. On the other hand, if the system size is fixed, the communication/computation ratio is likely to decrease with increased problem size for most practical algorithms. For these algorithms, increasing the problem size with the system size can keep the average speed constant. On the basis of this observation, the *isospeed scalability* has been formally defined as the ability to maintain the average speed.[3]

Definition 2: An *algorithm-machine combination* is scalable if the achieved average speed of the algorithm on a given machine can remain constant with increasing numbers of processors, provided the problem size can be increased with the system size.

Definitions and applications

As parallel machines with more and more processors become available, the performance metric *scalability* becomes increasingly important. Scalability measures how an algorithm performs when the problem size is scaled up linearly with the number of processors. Let $T(p, W)$ be the execution time for solving a problem with work W (problem size) on p processors. In the ideal situation, both the number of processors and the amount of work are scaled up N times, while the execution time remains unchanged:

$$T(N\times p, N\times W) = T(p, W) \qquad \text{(A)}$$

Equation A is true if and only if the average unit speed of the given computing system is constant, where average unit speed is defined as the quotient of the achieved speed of the given computing system and the number of processors. Scalability is formally defined as the ability to maintain a given average unit speed.[1] Let W be the amount of work of an algorithm when p processors are employed on a machine, and let W' be the amount of work of the algorithm when p' processors are employed to maintain the average speed. Then the scalability from system size p to system size p' of the algorithm-machine combination is defined as

$$\psi(p,p') = \frac{p'\cdot W}{p\cdot W'} \qquad \text{(B)}$$

Because the average speed is fixed, the isospeed scalability (Equation B) can be equivalently defined in terms of execution time:[1]

$$\psi(p,p') = \frac{T(p,W)}{T(p',W')} \qquad \text{(C)}$$

where $T(p', W')$ is the corresponding execution time of solving W' on p' processors. When $T(N\times p, N\times W) = T(p, W)$, the scalability is 1 (by Equation C). However, this is the *ideal* situation. In general, $T(N\times p, N\times W) > T(p, W)$, and the scalability is less than 1.

Speed is defined as work divided by time. The definition of problem size is still under debate. However, it is commonly agreed that the floating-point operation (flop) count is a good estimate for scientific computations. To eliminate the effect of numerical inefficiencies in parallel algorithms, the flop count is, in practice, based on some optimal sequential algorithms.

Examples

Periodic tridiagonal systems arising in many applications are multiple right-side systems. They are usually kernels in much larger codes. It is often more efficient to use a parallel solver for these systems than to remap data among processors, so that different processors can solve different right sides concurrently—especially for distributed-memory machines, where communication cost is high.

For most distributed-memory computers, the time for a processor to communicate with its nearest neighbors varies linearly with the problem size. Let S be the number of bytes to be transferred. The transfer time for a processor to communicate with a neighbor can be expressed as $\alpha + S\beta$, where α is a fixed startup time and β is the incremental transmission time per byte. Let τ_{comp} represent the unit cost of a computation operation normalized to the communication time. The time required to solve a periodic tridiagonal system by the Parallel Diagonal Dominant (PDD) algorithm[2] with p processors is

$$T(p,W) = (9\tfrac{n}{p}+1)n_1\cdot\tau_{\text{comp}} + 2(\alpha + 4\cdot n_1\cdot\beta)$$

where n is the order of the system, and n_1 is the number of right sides.

For solving tridiagonal systems, we choose the Thomas algorithm—the LU decomposition method for tridiagonal systems—as the conventional sequential algorithm.[3] It takes $7n\cdot n_1$ flops for a periodic system with n_1 multiple right sides. Let's assume the number of right sides is fixed. Then, when the problem size W increases N times to W', we have

$$W' = (N\times 5n)\cdot n_1 = 5n'\cdot n_1 \qquad \text{(D)}$$

$$n' = N\cdot n \qquad \text{(E)}$$

Thus, for the PDD algorithm, the execution time of the scaled problem is

$$\begin{aligned} T(N\times p, N\times W) &= (9\tfrac{n'}{N\cdot p}+1)n_1\cdot\tau_{\text{comp}} + 2(\alpha+4\cdot n_1\cdot\beta)\\ &= (9\tfrac{N\cdot n}{N\cdot p}+1)n_1\cdot\tau_{\text{comp}} + 2(\alpha+4\cdot n_1\cdot\beta)\\ &= (9\tfrac{n}{p}+1)n_1\cdot\tau_{\text{comp}} + 2(\alpha+4\cdot n_1\cdot\beta)\\ &= T(p,W)\end{aligned}$$

By Equation C, the PDD algorithm is perfectly scalable. Given our assumption, its scalability is 1.

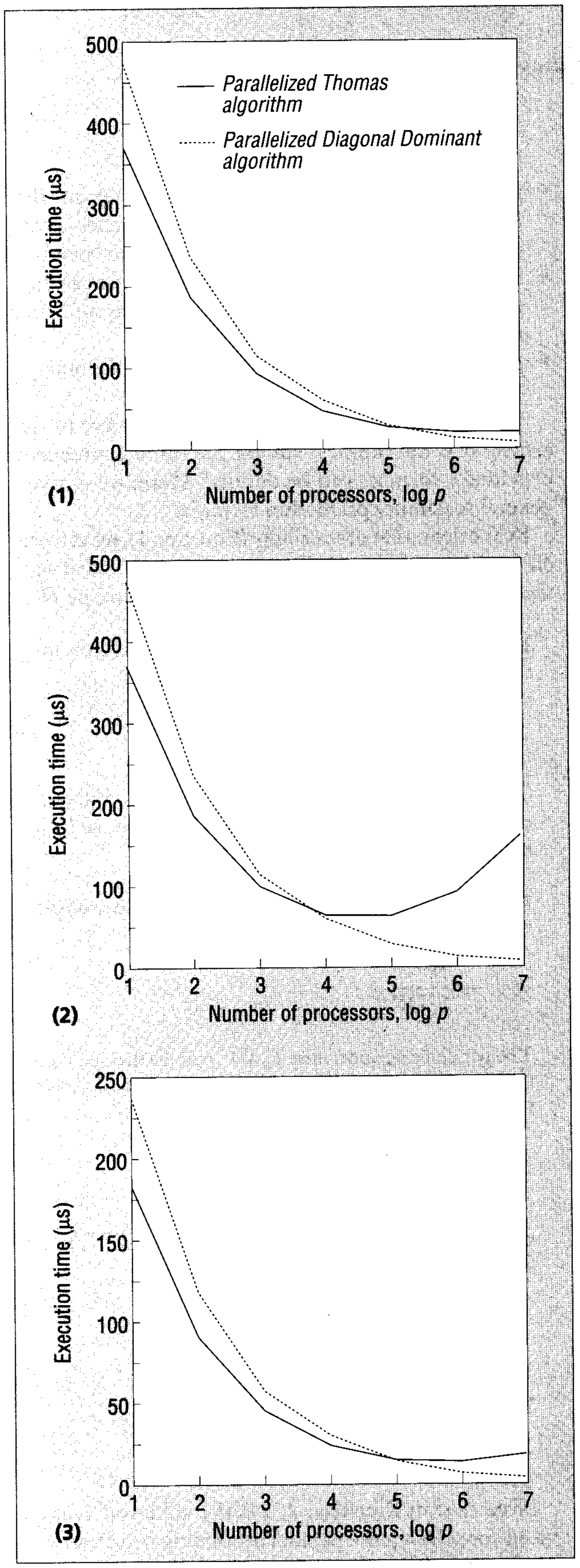
We can also parallelize the Thomas algorithm, for parallel processing. The time required by the parallelized Thomas (PT) algorithm is

$$T(p,W) = (7\tfrac{n}{p})n_1\cdot\tau_{\text{comp}} + 2p(\alpha + 6\cdot n_1\cdot\beta)$$

for solving a periodic system with multiple right sides.[3] The PT algorithm has a smaller operation count than does the PDD algorithm, but its communication cost, $2p(\alpha + 6\cdot n_1\cdot\beta)$, increases with the number of processors. The PT algorithm is not perfectly scalable. Its scalability analysis needs more deliberations. (See the prediction formula, Equation 3, proposed in the main text of this article.)

Applications

The PT algorithm has a smaller operation count and a lower scalability than does the PDD algorithm. The PT algorithm is superior to the PDD algorithm when the



problem size and system size are small and becomes inferior when the problem size and system size scale upward. Finding the crossing point of the change in superiority directly impacts performance optimization and execution-time reduction. Figure A depicts the time variations of the PT algorithm and the PDD algorithm under different machine and problem assumptions. The crossing points are $p = 64$ in Figure A1, $p = 16$ in Figure A2, and $p = 32$ in Figure A3, where p is the number of processors. As shown by these figures, the crossing point varies with the machine parameters, the application parameters, and the algorithms. Therefore, scalability study has found its role in tuning performance, optimizing compiler restructuring, identifying appropriate architectures, investigating alternate algorithms, and selecting the best algorithm-machine pair for an application.

References

1. X.-H. Sun and D. Rover, "Scalability of Parallel Algorithm-Machine Combinations," *IEEE Trans. Parallel and Distributed Systems*, Vol. 5, No. 6, June 1994, pp. 599–613.
2. X.-H. Sun, "Application and Accuracy of the Parallel Diagonal Dominant Algorithm," *Parallel Computing*, Vol. 21, Aug. 1995, pp. 1241–1267.
3. T. Eidson and G. Erlebacher, "Implementation of a Fully-Balanced Periodic Tridiagonal Solver on a Parallel Distributed Memory Architecture," *Concurrency: Practice and Experience*, Vol. 7, No. 4, June 1995.

Figure A. Execution times of PT and PDD algorithms under different machine and problem assumptions: (1) α (fixed startup time) = 10^{-3}, β (incremental transmission time per byte) = 10^{-5}, τ_{comp} (unit cost of a computation) = 10^{-4}, n (order of the system) = 1,024, n_1 (number of right sides) = 1,024; (2) $\alpha = 10^{-2}$, $\beta = 10^{-4}$, $\tau_{\text{comp}} = 10^{-4}$, n = 1,024, n_1 = 1,024; (3) $\alpha = 10^{-3}$, $\beta = 10^{-5}$, $\tau_{\text{comp}} = 10^{-4}$, n = 512, n_1 = 1,024.

For a large class of algorithm-machine combinations, we can maintain the average speed by increasing problem size.[3] The necessary increase of problem size varies with algorithms, machines, and their combinations. This variation provides a quantitative measurement for scalability. Let W be the amount of work of an algorithm when p processors are employed in a machine, and let W' be the amount of work needed to maintain the average speed when $p' > p$ processors are employed. We define the *scalability* from system size p to system size p' of the algorithm-machine combination as follows:

$$\psi(p, p') = \frac{p' \cdot W}{p \cdot W'} \tag{1}$$

The work W' is determined by the isospeed constraint. When

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

(that is, when average speed is maintained with work per processor unchanged) the scalability is 1. However, this is the ideal case. In general, we might have to increase work per processor to achieve the fixed average speed, in which case scalability will be less than 1.

Speedup is a widely used performance metric in parallel processing. It is defined as sequential execution time over parallel execution time and is used to measure the parallel processing gain over sequential processing. Traditionally, parallel efficiency is defined as speedup divided by p (where p, the number of processors, is the ideal speedup). The traditional parallel efficiency is the efficiency in terms of speedup. Contrary to speedup, average speed is an indicator of uniprocessor efficiency, where uniprocessor efficiency is average unit speed divided by sustained uniprocessor speed. Maintaining average speed is equivalent to maintaining the uniprocessor efficiency. Under certain assumptions, maintaining average speed is also equivalent to maintaining the parallel efficiency.[8] However, in practice, these two approaches can lead to totally different results. Scaled speedup is a variation of speedup. It has been proposed in recent years to extend the capability of speedup for scalable computing. The "Scalability versus scaled speedup" sidebar shows the difference between scalability and scaled speedup.

Researchers have proposed three different approaches to finding the scalability values, as defined by Equation 1, of an algorithm-machine combination.[3] The scalability can be

- *measured* using software by a control program that invokes the application program and searches for the run that has the desired fixed average unit speed,
- *computed* by first finding the relation between average unit speed and execution time (or work) and then using Equation 1 (or Equation 4), or
- *predicted* by deriving a general scalability formula.

The third approach, prediction, is the topic of this article. It is the simplest among the three approaches, provided a formula can be defined. Here, we derive a general scalability prediction formula.

By the definition of scalability (Equation 1), we can predict scalability if and only if we can predict the scaled work size W'. Proposition 1 provides a way to determine W'.

Proposition 1: If parallel degradation exists, then for scalability (Equation 1),

$$W' = \frac{a \cdot p' \cdot T_o'}{1 - a \cdot \tau} \tag{2}$$

where a is the fixed average speed, τ is the computing rate (reciprocal of speed) of a single processor, and T_0' is the parallel processing overhead.

Proof: Since W' is the scaled work satisfying the isospeed requirement,

$$a = \frac{W'}{p' \cdot T_{p'}(W')}$$

The parallel execution time $T_{p'}(W')$ can be divided into two parts: ideal parallel processing time, and parallel processing overhead T_o'.

$$T_{p'}(W') = \frac{T_1'}{p'} + T_o' = \frac{W' \cdot \tau}{p'} + T_o'$$

where T_1' is the sequential execution time and T_1'/p' is the ideal parallel execution time. Thus,

$$a = \frac{W'}{W' \cdot \tau + T_o' \cdot p'}$$

and

$$W' = \frac{a \cdot p' \cdot T_o'}{1 - a \cdot \tau}$$

Note that in Equation 2, a is the achieved average speed considering the parallel processing overhead, and τ is the computing rate without considering the overhead. When parallel degradation does exist (when $T_o' >$

Scalability versus scaled speedup

In scalable computing, we scale up the problem size and the parallel-system ensemble size to explore the computational power of parallel computers for solving otherwise intractable, large problems. Both scalability and scaled speedup are performance metrics of scalable computing. They are, however, fundamentally different. The speedup metric was introduced more than three decades ago to justify the value of parallel processing. It measures the execution-time reduction of parallel processing over sequential processing. Scaled speedup, namely the fixed-time speedup and the memory-bounded speedup, are models of speedup for scalable computing.[1] As the ensemble size increases, fixed-time speedup scales problem size to meet the fixed execution time, and memory-bounded speedup scales problem size to use the associated memory increase. We also solve the scaled problem sequentially to get the speedup. The problem size increase provides sufficient work for parallel processing and, therefore, more reasonably compares parallel and sequential processing in terms of execution-time reduction.

With the continuous development of parallel computers, scalability has emerged and has been recognized in recent years as an important property of parallel algorithms and architectures. Scalability measures the effort needed to maintain the current efficiency, if possible, when system ensemble size increases. Researchers have proposed a few scalability metrics, with different qualitative definitions and quantitative measurements of *efficiency* and *effort*. In isospeed scalability, the efficiency is defined in terms of average unit speed, and the effort is measured in terms of problem size increase. Scalability and scaled speedup quantify different properties of parallel processing. These quantitative measurements are distinct.

For instance, in the Householder Transformation algorithm, when the ensemble size scales up from p to $m \cdot p$, the increase of problem size in the memory-bounded scaled speedup will be $n' = n\sqrt{m}$. The reason is that the memory requirement of the Householder Transformation algorithm is a square function of n, the order of the matrix. The corresponding scaled speedup is

Memory-bounded speedup

$$= \frac{\textit{Sequential solving time of the scaled problem}}{\textit{Parallel solving time of the scaled problem}}$$

$$= \frac{\left[2(n\sqrt{m})^3 + 3(n\sqrt{m})^2\right]\tau}{\left[\frac{2(n\sqrt{m})^3}{mp} + 3(n\sqrt{m})^2\right]\tau + (n\sqrt{m})^2\beta}$$

$$= \frac{\left[2n^3\sqrt{m} + 3n^2\right]\tau}{\left[\frac{2n^3}{p\sqrt{m}} + 3n^2\right]\tau + n^2\beta}$$

The memory-bounded scaled speedup is greater than the fixed-size speedup, where the problem size is fixed and $m = 1$.

Although they are distinct, scalability and scaled speedup are related. One noticeable result is that an algorithm-machine combination is ideally scalable if and only if it achieves the ideal scaled (fixed-time or memory-bounded) speedup.[2] Vipin Kumar and Anshul Gupta[3] and Sartaj Sahni and Venkat Thanvantri[4] survey parallel processing performance metrics, including speedup, scaled speedup, and scalability.

References

1. X.-H. Sun and L. Ni, "Scalable Problems and Memory-Bounded Speedup," *J. Parallel and Distributed Computing*, Vol. 19, Sept. 1993, pp. 27–37.

2. X.-H. Sun, "The Relation of Scalability and Execution Time," *Proc. Int'l Parallel Processing Symp. '96*, IEEE Computer Society Press, Los Alamitos, Calif., 1996, pp. 457–462.

3. V. Kumar and A. Gupta, "Analyzing Scalability of Parallel Algorithms and Architectures," *J. Parallel and Distributed Computing*, Vol. 22, No. 3, Sept. 1994, pp. 379–391.

4. S. Sahni and V. Thanvantri, "Performance Metrics: Keeping the Focus on Runtime," *IEEE Parallel & Distributed Technology*, Vol. 4, No. 1, Spring 1996, pp. 43–56.

0), then $a \cdot \tau < 1$, and, therefore, Equation 2 is traceable. $T_o' > 0$ is a necessary and sufficient condition of Proposition 1. When $T_o' = 0$, based on the definition of scalability (Equation 1), ideal scalability is achieved with $\psi(p', W') = 1$. Parallel processing overhead is generally a function of system size and problem size. We use the notation T_o', instead of T_o, in the derivation to reflect the fact that the overhead is for the system/work pair (p', W').

Combining Equations 1 and 2, we have

$$\psi(p, p') = \frac{W(1 - a \cdot \tau)}{p \cdot a \cdot T_o'} \qquad (3)$$

Equation 3 is very useful. It not only gives a way to predict scalability, but more importantly, it shows the following properties of isospeed scalability:

(1) When the computing rate τ is fixed, scalability (Equation 1) increases with the decrease of the average speed a.
(2) τ, the computing rate of a single processor, is the inverse of single-processor speed. Equation 3 shows that, when the average speed a is fixed, scalability increases with single-processor speed.
(3) Scalability increases as the degradation of parallelism T_o' decreases.

Property 1 shows that less effort is needed to maintain lower efficiency, where we consider $a \cdot \tau$ as the

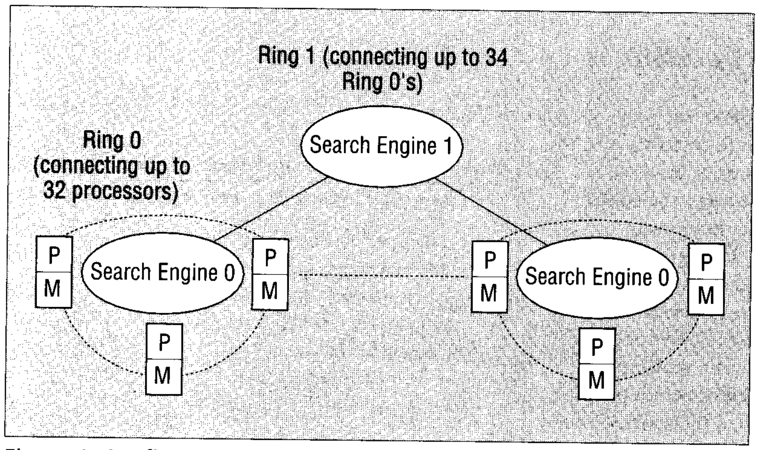

Ring 1 (connecting up to 34 Ring 0's)
Ring 0 (connecting up to 32 processors)
Search Engine 1
Search Engine 0
Search Engine 0
P
M

Figure 1. Configuration of a KSR-1 parallel computer (where P is a processor, and M represents 32 Mbytes of local memory).

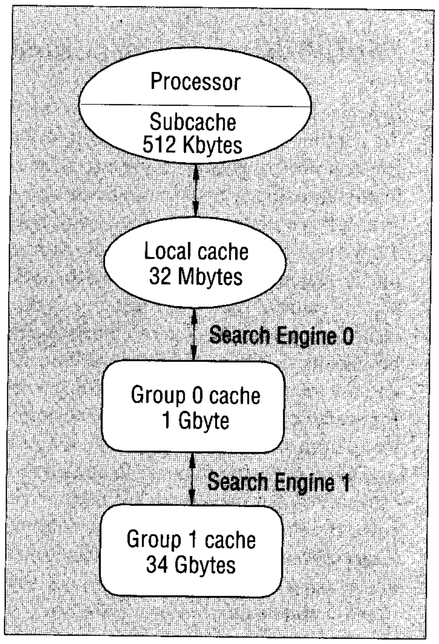


Figure 2. Memory hierarchy of the KSR-1.

```
Initializing the matrix
for $i$ = 1, n
 1. $\alpha_i = -\,\mathrm{sign}(\mathbf{a}^i_{i1})(\mathbf{b}^{i\,T}_i \mathbf{b}^i_i)^{1/2}$
 2. $\mathbf{w}_i = \mathbf{b}^i_i - \alpha_i \mathbf{e}_1$
 3. $\beta_j = \mathbf{w}^T_i \mathbf{b}^i_j\,(\alpha^2_i - \alpha_i \alpha^i_{i1})$, $j = i + 1,$
      ..., n
 4. $\mathbf{b}^i_j = \mathbf{b}^i_j - \beta_j \mathbf{w}_i$, $j = i + 1, \ldots, n$
end for
```

Figure 3. The Householder Transformation, where n and m are the numbers of columns and rows in the matrix $\boldsymbol{A}$ of Equation 6, respectively, and all bold-face lower-case letters represent vectors of length $(m + 1)$. We can calculate each β_j and update each $\mathbf{b}^i_j$ in parallel for different values of index j.

uniprocessor efficiency. Equation 3 gives the relation between the effort (scalability) and the performance (fixed average speed) of an algorithm-machine combination. Property 1 also shows that, by adjusting the average speed a, we can apply isospeed scalability to a large class of algorithm-machine combinations, from massively parallel systems with relatively weak processing elements to supercomputers with a few powerful processors. Equation 3 also gives the relation between isospeed scalability, computing power of a single processor, and degradation of parallelism. Properties 2 and 3 show that isospeed scalability does not give credit to slow computing and fast communication. These two properties are very important in the evaluation of computing systems.

Although Equation 3 is very useful, using it in performance prediction might not be as simple as it looks. The degradation of parallelism T_o, which contains both communication-delay and workload-imbalance degradation, can be difficult to compute. Also, the single-processor rate can vary with algorithm and with problem size, especially for shared-virtual-memory machines.[8,9] The detailed case study in the next section illustrates how the prediction formula can be used in practice, and how the predicted scalability can be used to evaluate machine architectures.

Finally, Equation 4 shows how we can compute parallel execution time from scalability:

$$T_{p'}(W') = \psi^{-1}(p, p') \cdot T_p(W) \tag{4}$$

where $T_p(W)$ and $T_{p'}(W')$ are the parallel execution times of solving the problem with the work of W and W' on a system of p and p' processors, respectively. The computing rate τ of a single processor is machine-dependent. The degradation of parallelism T_o is both architecture- and algorithm-dependent. Equation 3 helps us find a good algorithm-machine combination in terms of scalability. Equation 4 shows that larger scalability leads to smaller execution time.

The case study

Here, we discuss the case study for solving an application problem on a KSR-1 parallel computer.

The machine

We performed our case study on a KSR-1 parallel computer. It has a distributed physical memory, which makes a large ensemble size possible, and a shared address space, which lets users develop programs in a shared-memory-like environment.

Figure 1 shows the architecture of the KSR-1 parallel computer. Each processor on the KSR-1 has 32 Mbytes of local memory. The CPU is a super-scalar processor with a hardware 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.

Access to nonlocal data on the KSR-1 is provided by a hierarchy of search engines. The search engine SE 0 locates data in the local ring, while SE 1 provides data access between local rings. Figure 2 shows the KSR-1's memory hierarchy.

Each processor has 512 Kbytes of fast *subcache*, which is similar to the normal cache in other parallel computers. The 32 Mbytes of local memory in each processor is called a *local cache*. A local ring (Ring 0) with up to 32 processors can have 1 Gbyte of local cache, which is called the Group 0 cache. Access to the Group 0 cache is provided by SE 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 SE 1. Kendall Square Research calls the entire memory hierarchy *Allcache*. A processor can access the Allcache memory system via different search engines, as shown in Figure 2. The latencies for different memory locations are 2 cycles for subcache, 20 cycles for local cache, 150 cycles for Group 0 cache, and 570 cycles for Group 1 cache.[10]

The application

The numerical algorithm used in this case study is the Householder Transformation algorithm for the QR factorization of matrices. It is used for solving the normal equation

$$A^T A x = A^T \mathbf{b} \tag{5}$$

without explicitly forming $A^T A$.

In many cases—for instance, the inverse problem of partial differential equations—the normal equation system resulting from the discretization is too ill-conditioned to be solved directly. Tikhnov's regularization method is frequently used in this case to increase numerical stability.[11] The key step in solving the regularized least-squares problem (RLSP) is to introduce a regularization factor $\alpha > 0$. Instead of solving Equation 5 directly, we solve the system

$$(A^T A + \alpha I)\mathbf{x} = A^T \mathbf{b} \tag{6}$$

for **x**. Figure 3 describes the Householder Transformation.

Scalability analysis

Based on the definition of isospeed scalability, the work W' at processor number p' should keep the system ensemble running at the same average speed a as with p processors, so that

$$a = \frac{W}{p \cdot T_p(W)} = \frac{W'}{p' \cdot T_{p'}(W')} \tag{7}$$

where $T_p(W)$ and $T_{p'}(W')$ are the execution times using p and p' processors, respectively.

For the particular problem discussed here, let us assume $m = n$. On the basis of the above algorithm, the operation count is $3n^2$ for Steps 1 and 2, and $2n^3$ for Steps 3 and 4. The total work is

$$W(n) = 2n^3 + 3n^2 \tag{8}$$

for a matrix size of $n \times n$ in Equation 6. Because we can parallelize the computation for Steps 3 and 4 in Figure 3 for different values of index j, whereas we can compute Steps 1 and 2 in Figure 3 only on a single processor (we can actually distribute this computation, too, but the gain wouldn't be enough to cover the loss due to interprocessor communications), the runtime model is

$$T_p(n) = \left[\tfrac{2n^3}{p} + 3n^2\right]\tau + n^2\beta \tag{9}$$

where p is the number of processors, τ is the rate of computing without communication overhead, and β is the latency for access of remote data. The last term in Equation 9 represents the communication cost, because a vector of length $(n + 1)$ must be communicated n times in the transformation process. The total communication cost is $(n + 1) * n\beta < n^2\beta$.

On the basis of the discussion given earlier, we can represent the runtime $T_p(n)$ in Equation 9 as

$$T_p(n) = T_C(n, p) + T_o(n, p) \tag{10}$$

where $T_C(n, p)$ is the computing time with ideal parallelism, and $T_o(n, p)$ represents the degradation of parallelism. We then have

$$T_C(n, p) = \frac{2n^3 + 3n^2}{p}\tau$$

$$T_o(n, p) = (3n^2 - \tfrac{3n^2}{p})\tau + n^2\beta$$

The first term of T_o is due to the workload imbalance.

The second term is due to the communication (remote memory access) delay. Using Equation 2, we get

$$W' = \frac{a \cdot p'(-\frac{3n'^2}{p'}\tau + 3n'^2\tau + n'^2\beta)}{1 - a \cdot \tau} \tag{11}$$

The matrix size n is the parameter used to adjust the problem size. Substituting

$$W' = 2n'^3 + 3n'^2$$

into Equation 11, we have

$$2n'^3 + 3n'^2 = \frac{a \cdot p'(-\frac{3n'^2}{p'}\tau + 3n'^2\tau + n'^2\beta)}{1 - a \cdot \tau}$$

which eventually leads to

$$n' = \frac{3a \cdot \tau \cdot p' + a \cdot \beta \cdot p'}{2(1 - a \cdot \tau)} - \frac{3}{2(1 - a \cdot \tau)} \tag{12}$$

Equation 12 is true for any work-processor pair that maintains the fixed average speed, assuming that τ and β are unchanged. In particular,

$$n = \frac{3a \cdot \tau \cdot p + a \cdot \beta \cdot p}{2(1 - a \cdot \tau)} - \frac{3}{2(1 - a \cdot \tau)} \tag{13}$$

Combining Equations 12 and 13, we have

$$(n' - n) = \frac{3a \cdot \tau + a \cdot \beta}{2(1 - a \cdot \tau)}(p' - p) \tag{14}$$

which shows that the variation of n is directly proportional to the variation of ensemble size, provided that τ and β are independent of the number of processors.

Equation 14 indicates that the matrix size n' must increase at the same rate as the number of processors p' to maintain the prespecified average speed a. If $p' = m \cdot p$, then $n' = m \cdot n$. Assuming n is large so that the cubical term in Equation 8 is dominant, we have the relation

$$W'(n') = W'(m \cdot n) \approx m^3 W(n)$$

Therefore, we can estimate the scalability of this algorithm-machine combination as

$$\psi(p, p') = \psi(p, m \cdot p) \approx \frac{m \cdot p \cdot W}{p \cdot m^3 \cdot W} = \frac{1}{m^2} \tag{15}$$

In particular, if $m = 2$, which means the number of processors is doubled for each case, the scalability will be approximately ¼.

It is clear from Equation 14 that the parameters τ and β must be determined before we can predict the execution time and scalability. With the runtime model given by Equation 9, we can estimate τ and β in the model to fit the measured runtimes using the method of least squares for regression.[8] Assuming the executions times $T_{p_1}(n_1), \ldots, T_{p_k}(n_k)$ are available on $p_1, p_2, \ldots, p_k$ processors, with problem sizes $n_1, n_2, \ldots, n_k$, we have

$$\tau = \frac{\sum_{i=1}^{k} b_i T_{p_i} \sum_{i=1}^{k} c_i^2 - \sum_{i=1}^{k} c_i T_{p_i} \sum_{i=1}^{k} b_i c_i}{\sum_{i=1}^{k} b_i^2 \sum_{i=1}^{k} c_i^2 - (\sum_{i=1}^{k} b_i c_i)^2}$$

$$\beta = \frac{\sum_{i=1}^{k} b_i^2 \sum_{i=1}^{k} c_i T_{p_i} - \sum_{i=1}^{k} b_i c_i \sum_{i=1}^{k} b_i T_{p_i}}{\sum_{i=1}^{k} b_i^2 \sum_{i=1}^{k} c_i^2 - (\sum_{i=1}^{k} b_i c_i)^2} \tag{16}$$

where

$$b_i = \frac{2n_i^3}{p_i} + 3n_i^2, \quad c_i = n_i^2$$

Current progress shows that certain software tools can automatically determine parameters τ and β.[7,12] Therefore, we can incorporate our prediction formula in these tools to predict performance variations.

Scalability prediction and its application

The peak performance specified by vendors gives the hardware performance limit but can hardly be used to accurately predict execution time. For most application problems, the sustained speed is only a small percentage of the peak performance. The same argument applies to communication latency. The observed latency can be significantly different from the machine specifications. The architecture specification[10] for KSR-1 gives

$$\tau = 0.025\ \mu\text{s}, \quad \beta_1 = 7.5\ \mu\text{s} \tag{17}$$

Note that in Equation 9, β represents the latency for access of remote data. Because the KSR-1 parallel computer used in our case study has two levels of remote cache, we use β_1 and β_2 to represent the hardware specifications of latencies for data access in Group 0 and Group 1 cache. The estimated latencies from our numerical experiment for the two groups of cache are represented by β' and β''.

To determine the value of τ and β for this particular algorithm-machine pair, we run the code on $p = 2$ and 4 processors and measured the total execution time $T_p(n)$ with $n = 362$ and 512, respectively. We then calculate τ and β using the model in Equation 16. The parameters obtained this way are

$$\tau' = 0.18 \mu s, \quad \beta' = 3.37 \mu s \qquad (18)$$

Comparing Equations 17 and 18, we see that τ' is significantly larger than τ. The sustained computational speed is

$$\frac{1}{\tau'} = 5.56 \text{ Mflops}$$

which is about 14% of the peak performance of 40 Mflops. This speed includes all the effects of subcache misses and other overheads. On the other hand, the value of β' in Equation 18 is significantly smaller than the value of β in Equation 17, which means the actual observed communication speed is faster. This can be attributed to two factors:

- *Overlapping of communications with computations.* In the Householder Transformation, one processor calculates the pivoting column and then broadcasts it to all other processors. This broadcasting process can be partly overlapped with the other computations.
- *Automatic prefetch.* The KSR-1 Fortran compiler analyzes loops and, whenever possible, generates instructions to prefetch remote data needed for subsequent loops, thus saving data access time.

Figure 4a shows both the measured execution time and the predicted execution time in seconds. The predicted execution time is based on Equations 9 and 18. The problem size is scaled up using the memory-bounded scaled-up model.[3] For the RLSP application, the memory requirement is a square function of the parameter n, and the operation count is a cubical function of n. That explains why the runtime goes up with more processors.

Figure 4a makes it clear that the predicted execution time matches the measured execution time well until $p = 22$. After that, the error increases significantly. This is due to the multi-ring structure of the KSR-1. Each ring has 32 processors.

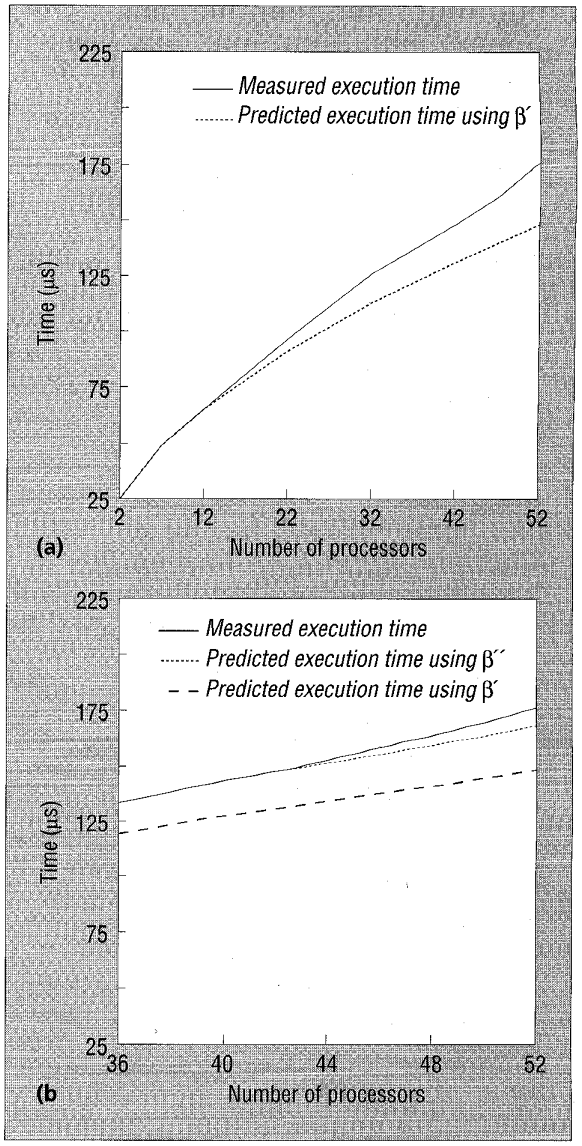


Figure 4. Measured and predicted execution time, where problem size is scaled up with available memory: (a) without using adjusted parameters; (b) using adjusted parameters.

Because several of the 32 processors are committed to I/O and control processes and are usually not used in computation, multi-ring communication is involved even for p fewer than (but close to) 32. This multi-ring communication requires data access to the Group 1 cache, which slows the computations significantly. The listed access time for the Group 1 cache on KSR-1 is[10]

$$\beta_2 = 28.5 \ \mu s \qquad (19)$$

Again, the measured access time for our application is

Table 1. Predicted and measured matrix sizes.

Size	1	2	4	8	16	32	56
Predicted	–	54	115	238	484	976	2,889
Measured	29	57	109	230	461	1,006	2,773

significantly different from the listed value, especially when most communications are within a single ring. To determine the communication delay for multiple rings, we ran the code on 36 processors and measured the execution time. Then we calculated the value of β from Equation 9 by setting $\tau = 0.18\ \mu s$, as given in Equation 18. The new β value is

$$\beta'' = 6.27\ \mu s \qquad (20)$$

which is about twice as large as in Equation 18.

Figure 4b shows the execution time for $p > 32$. With the new value of β'', the predicted runtime matches the measured execution time nicely.

Based on Equation 12 and the test runs on $p = 2, 4$, and 36 processors, we can predict the matrix size n'. Table 1 shows the predicted and measured matrix sizes. The average speed a maintained in this test is 3.25 Mflops, which is about 58% of the sustained speed in Equation 18. From Table 1, we see that the predicted matrix size is very close to the actual matrix size measured by running the code on 8, 16, 32, and 56 processors.

The last column in Table 1 shows the predicted size n' using β'', the inter-ring data access time. If the β' given in Equation 18, the intra-ring data access time, is used to predict the matrix size, then n' will be 1,715 at $p = 56$, which is significantly smaller than the measured n'. The difference shows the influence of slower remote memory access of the Group 1 cache on scalability.

With the matrix sizes given in Table 1 and the parameters given in Equations 18 and 19, we can compute the scalability $\psi(p, p')$. Tables 2 and 3 give the predicted and measured scalabilities, respectively. We can see that the predicted and measured scalabilities are fairly close. The prediction for an ensemble size of 56 is based on the inter-ring data access time β''. Figure 5 depicts the difference between the measured scalability and the predicted scalability obtained using β'. The curves in the figure represent measured and predicted $\psi(p, 56)$, with p varying from 1 to 32. To clearly see the difference between the two curves in Figure 5, we have to plot them using a log scale. To avoid plotting the curves in the negative region, which would be rather confusing, we use a negative log scale. This allows the separation of curves and keeps them in the positive region. However, the tradeoff is that the curve

Table 2. Predicted scalability of RLSP/KSR-1 combination.

Scalability	Number of processors						
$\psi(p, p')$	1	2	4	8	16	32	56
1	1.00000	0.33238	0.07183	0.01652	0.00397	0.00097	0.00007
2		1.00000	0.21611	0.04971	0.01193	0.00292	0.00020
4			1.00000	0.23003	0.05520	0.01352	0.00092
8				1.00000	0.23999	0.05879	0.00398
16					1.00000	0.24499	0.01658
32						1.00000	0.06767
56							1.00000

Table 3 . Measured Scalability of RLSP/KSR-1 combination.

Scalability	Number of processors						
$\psi(p, p')$	1	2	4	8	16	32	56
1	1.00000	0.28382	0.08418	0.01830	0.00459	0.00089	0.00007
2		1.00000	0.29660	0.06446	0.01616	0.00313	0.00026
4			1.00000	0.21734	0.05449	0.01054	0.00088
8				1.00000	0.25070	0.04849	0.00406
16					1.00000	0.19343	0.01621
32						1.00000	0.08378
56							1.00000

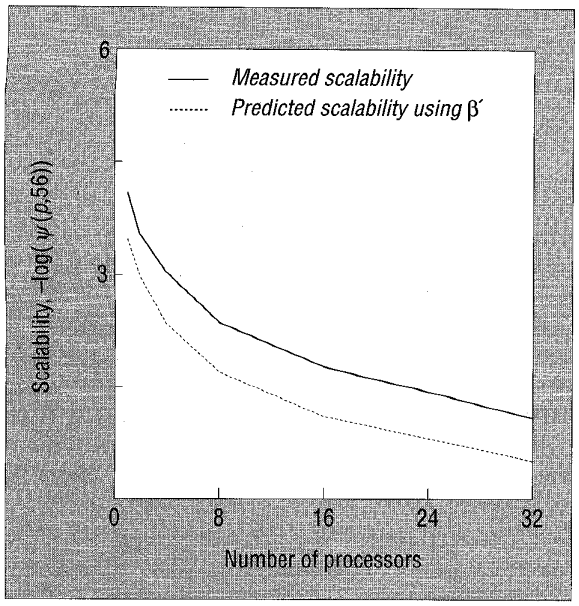


Figure 5. Measured and predicted scalability. Equation 18 is used in this prediction.

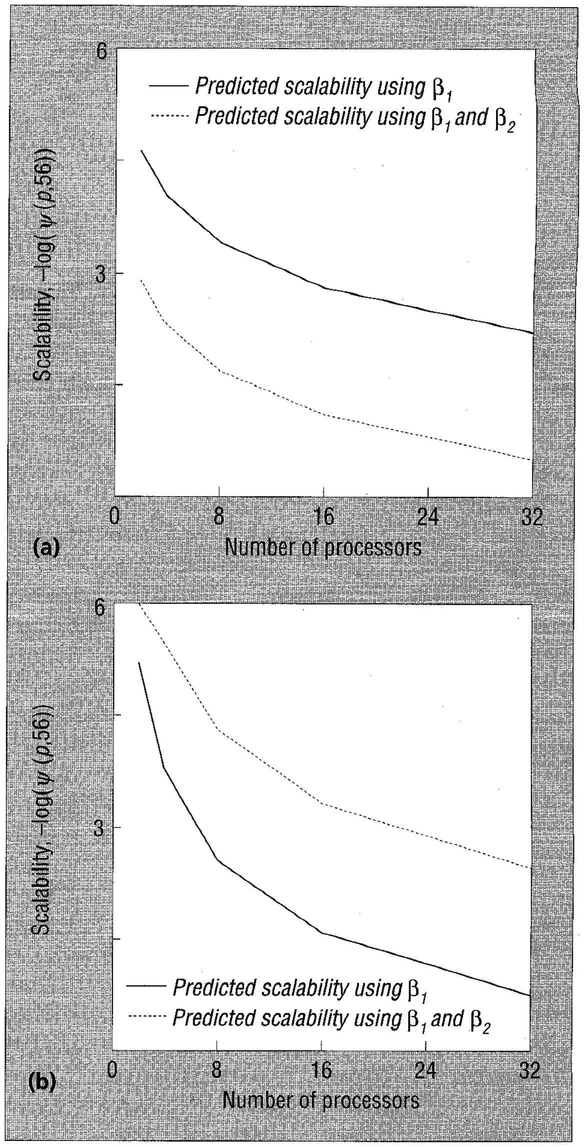


Figure 6. Predicted scalability: (a) using machine specifications for the Householder Transformation; (b) using machine specifications for the Givens Rotation.

with lower $-\log(\psi(p, 56))$ value actually represents higher scalability than the curve with higher $-\log(\psi(p, 56))$ value. For the same reasons, we use the negative log scale in plotting other figures, as well.

A single bus is an efficient architecture for supporting the shared-memory communication model and has been used successfully in several commercial shared-memory machines. Because of network contention, the single-bus architecture has trouble supporting many processors efficiently.

To function as a scalable shared-virtual-memory machine, the architecture of KSR-1 combines buses and a multi-ring structure. Each local ring has 32 processors connected to a single bus. Then, the local rings connect to form a multi-ring structure. Theoretically, we can scale up the computing system to any number of processors by increasing the number of levels of the connection. Figure 5 shows the limitation of the multi-ring approach. The scalability is severely reduced when inter-ring remote access is required. It shows that, unless inter-ring communication can be improved, uniprocessor efficiency will reduce quickly with the increase of ensemble size. Increasing the number of levels in the multi-ring hierarchy does not necessarily ensure high computing power.

The scalability difference given in Figure 5 derives from the measured scalability and the measured τ and β'. Figure 6a shows the scalability difference using the theoretical performance data τ, β_1, and β_2, where the average speed is fixed at 58% of the hardware peak performance. This figure gives the theoretical difference of the RLSP application when Group 1 communication is required. Comparing the curves in Figures 5 and 6a, we can clearly see the similarity. Both figures show that scalability with remote cache access is much lower than scalability without remote data access. The general trends in both figures are very similar. Because we plotted the curves in Figure 6a on the basis of machine specification, we know that, although machine specification does not adequately estimate execution time or speed, it does help predict how architecture variation influences performance. Equation 3 is very useful; it shows performance variation based only on the hardware specification. Architecture variation influences different algorithms in different ways. When architecture scales up from one level of hierarchy to another, an inferior algorithm might become superior. The scalability formula

(Equation 3) provides a guideline for choosing algorithms for optimal performance.

Figure 6b shows the scalability curves for the Givens Rotation algorithm,[13] which can be used to solve the least-squares problem and has a different operation and communication requirement than does the Householder Transformation algorithm. (We obtained the data used to plot Figure 6b in exactly the same way as for Figure 6a, except we used a different performance model.) We can see that the scalability of the Givens rotation algorithm is less than that of the Householder Transformation algorithm. However, the difference decreases when the system scales up. This demonstrates that the scalability of the Givens algorithm is less affected by the hierarchical remote cache access than is the Householder Transformation algorithm. The Givens algorithm can provide better scalability—and, therefore, a better execution time—when the system size is large enough to require multilevel ring communication. Figure 6b shows how to compare algorithms using the notion of scalability.

The average speed a maintained in this study is about 58% of the sustained speed. The efficiency maintained is reasonably high. The scalability given in Tables 2 and 3 could be higher if a were lower, as shown in Equation 3. Also, the computing rate τ generally varies with the number of processors and problem size on any machine that has a memory hierarchy. For our implementation, where the initial problem size is large and increases with the number of processors, the computing rate is quite stable. The scalability prediction will be more involved if the computing rate varies with system size.

This article addresses two basic problems: predicting the execution time and predicting the scalability. Like most existing models, the prediction of execution time relies on runtime information (such as τ and β), which can vary with problem size and ensemble size. Our experiments show, however, that although hardware does not realize the advertised performance in solving actual applications, the relative performance of architectures and algorithms can be predicted and compared in terms of scalability if hardware specifications are given. As we've discussed, when the system size scales up, an originally faster algorithm with lower scalability can become slower than another algorithm with better scalability. Finding the fast/slow crossing point is critical to optimizing the performance. The scalability prediction formula provided in this study is the first step to finding that crossing point.

Although we conducted the numerical experiment on a KSR-1 machine, the prediction formula and methodology proposed in this study are not bound to any algorithm or architecture. They can be applied to any algorithm-machine combination when needed runtime information is available. Recent progress in software development shows that the needed information can be determined automatically during runtime or can be approximated during compile time. The methodology introduced in this article, therefore, is feasible and can be incorporated into existing performance-prediction tools in developing an integrated parallel programming environment.

Future work will include fully exploring the use of the prediction formulas in different machines and applications. We will also focus on integrating performance prediction into a software system to support automatic parallel program compilation.

Acknowledgments

This research was supported in part by the National Aeronautics and Space Administration under NASA Contracts NAS1-19480 and NAS1-1672. We are also grateful to the Cornell Theory Center for providing access to the KSR-1 parallel computer, and to the referees for their helpful comments regarding the revision of this article.

References

1. A.Y. Grama, A. Gupta, and V. Kumar, "Isoefficiency: Measuring the Scalability of Parallel Algorithms and Architectures," *IEEE Parallel & Distributed Technology*, Vol. 1, No. 3, Aug. 1993, pp. 12–21.

2. J. Gustafson, G. Montry, and R. Benner, "Development of Parallel Methods for a 1024-Processor Hypercube," *SIAM J. Science and Statistics Computing*, Vol. 9, No. 4, July 1988, pp. 609–638.

3. X.-H. Sun and D. Rover, "Scalability of Parallel Algorithm-Machine Combinations," *IEEE Trans. Parallel and Distributed Systems*, Vol. 5, No. 6, June 1994, pp. 599–613.

4. X. Zhang, Y. Yan, and K. He, "Latency Metric: An Experimental Method for Measuring and Evaluating Parallel Program and Architecture Scalability," *J. Parallel and Distributed Computing*, Vol. 22, No. 3, Sept. 1994, pp. 392–410.

5. M.J. Clement and M.J. Quinn, "Analytical Performance Prediction on Multicomputers," *Proc. Supercomputing*, IEEE Computer Society Press, Los Alamitos, Calif., 1993, pp. 886–894.

6. T. Fahringer and H.A. Zima, "Static Parameter Based Performance Prediction Tool for Parallel Programs," *Proc. ACM Int'l Conf. Supercomputing*, ACM Press, New York, 1993, pp. 207–219.

7. S.R. Sarukkai, P. Mehra, and R.J. Block, "Automated Scalability Analysis of Message-Passing Parallel Programs," *IEEE Parallel & Distributed Technology*, Vol. 3, No. 4, Winter 1995, pp. 21–32.

8. X-H. Sun and J. Zhu, "Performance Considerations of Shared Virtual Memory Machines," *IEEE Trans. Parallel and Distributed Systems*, Vol. 6, No. 11, Nov. 1995, pp. 1185–1194.

9. K. Hwang, *Advanced Computer Architecture: Parallelism, Scalability, Programmability*, McGraw-Hill, New York, 1993.

10. *KSR Technical Summary*, Kendall Square Research, Waltham, Mass., 1991.

11. Y.M. Chen et al., " GPST Inversion Algorithm for History Matching in 3-D 2-Phase Simulators," *IMACS Trans. Scientific Computing*, Vol. 1, 1989, pp. 369–374.

12. M. Calzarossa et al., "A Tool for Workload Characterization of Parallel Systems," *IEEE Parallel & Distributed Technology*, Vol. 3, No. 4, Winter 1995, pp. 72–80.

13. A. Pothen and P. Raghavan, "Distributed Orthogonal Factorization: Givens and Householder Algorithms," *SIAM J. Science and Statistics Computing*, Vol. 10, 1989, pp. 1113–1135.

Xian-He Sun is an assistant professor in the Department of Computer Science at Louisiana State University. His research interests include parallel and distributed processing, parallel numerical algorithms, performance evaluation, and software systems. He received a BS degree in mathematics from Beijing Normal University, and an MS in mathematics and an MS and a PhD in computer science from Michigan State University. He is a guest editor for a special issue of the *Journal of Parallel and Distributed Computing* on Analyzing Scalability of Parallel Algorithms and Architectures. He is a senior member of the IEEE, and a member of the ACM and Phi Kappa Phi. Readers can contact Sun at the Dept. of Computer Science, Louisiana State Univ., Baton Rouge, LA 70803-4020; sun@bit.csc.lsu.edu; http://bit.csc.lsu.edu/~sun/sun.html.

Jianping Zhu is an associate professor in the Department of Mathematics and Statistics at Mississippi State University. He also works in the NSF Engineering Research Center for Computational Field Simulations. His major research interests include numerical methods for solving partial differential equations, parallel computing, and large-scale simulations. He received a BS in engineering mechanics in 1982 from Zhejiang University, China; an MS in computational mechanics in 1984 from Dalian Institute of Technology, China; and a PhD in applied mathematics in 1990 from State University of New York, Stony Brook. He is a member of the AMS, the SIAM, and the AIAA. Readers can contact Zhu at the Dept. of Math and Statistics, NSF Engineering Research Center, Mississippi State Univ., Mississippi State, MS 39762; jzhu@math.msstate.edu; http://www2.msstate.edu/~jz1/index.html.