J. Parallel Distrib. Comput. 68 (2008) 1403–1412

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

Algorithm-system scalability of heterogeneous computing☆

Yong Chen, Xian-He Sun*, Ming Wu

Department of Computer Science, Illinois Institute of Technology, Chicago, IL 60616, USA

ARTICLE INFO

Article history:
Received 30 September 2005
Received in revised form
7 December 2007
Accepted 3 June 2008
Available online 20 June 2008

Keywords:
Scalability
Heterogeneous computing
Performance evaluation
Parallel computing

ABSTRACT

Scalability is a key factor of the design of distributed systems and parallel algorithms and machines. However, conventional scalabilities are designed for homogeneous parallel processing. There is no suitable and commonly accepted definition of scalability metric for heterogeneous systems. Isospeed scalability is a well-defined metric for homogeneous computing. This study extends the isospeed scalability metric to general heterogeneous computing systems. The proposed isospeed-efficiency model is suitable for both homogeneous and heterogeneous computing. Through theoretical analyses, we derive methodologies of scalability measurement and prediction for heterogeneous systems. Experimental results have verified the analytical results and confirmed that the proposed isospeed-efficiency scalability works well in both homogeneous and heterogeneous environments.

1. Introduction

Scalability is an essential factor for performance evaluation and optimization of parallel and distributed systems. It has been used widely for describing how the system size and the problem size will influence the performance of parallel computers and algorithms. It measures the ability of parallel architectures to support parallel processing at different machine ensemble sizes, and measures the inherent parallelism of parallel algorithms. Scalability can also be used to predict the performance of parallel systems at large sizes based on their performance at small sizes. It suggests which parallel computer could be built with more processors and which algorithm might be suitable for a larger computer system.

Although scalability is important for parallel and distributed systems, most of current research focuses on homogeneous environments, once the mainstream of parallel computing. Homogeneous parallel computers such as MPP architecture machines have led parallel computer architecture for decades. Heterogeneous systems, however, have emerged as a major high-performance computing platform in recent years. Typical examples include CLUMPs [4] (CLUster of SMPs) and Grid [6]. CLUMPs leverage both the usability of SMP and the scalability of cluster. With a high-speed network, such as Myrinet or InfiniBand, clusters of SMPs become more promising for high-performance parallel machines. Grid computing environment is another emerging computing platform. Grid connects and coordinates computing resources of different virtual organizations. It makes possible to share and aggregate millions of heterogeneous computing resources distributed geographically across organizations and domains. As computing environments evolve, understanding scalability of heterogeneous environments becomes timely important and necessary. In this study, we propose an algorithm-system approach for general heterogeneous computing, based on the isospeed metric proposed in [22]. Analytical and experimental studies are conducted to confirm the feasibility of the newly proposed scalability model, and the results have shown that the new model is practical and effective. The preliminary study and result were published in [20].

The rest of this paper is organized as follows: Section 2 reviews the related work and Section 3 introduces the proposed general scalability model. The theoretical studies of the proposed scalability metric are then given. Section 4 presents experimental results and analyses, and finally, we summarize our current work and discuss future work in Section 5.

2. Related work

Scalability is important for parallel and distributed systems; however, there is still no widely accepted scalability metric for general heterogeneous systems. Several metrics were proposed [9,11,12,17,19,21–24], but most of these metrics were designed for homogeneous environments. In [22], Sun and Rover proposed an isospeed scalability metric to describe the scalability of an algorithm-machine (or code-machine) combination in homogeneous environments. This metric is based on an average unit speed concept, defined as the system's achieved speed divided by the

☆ This research was supported in part by the National Science Foundation under NSF grant CNS-0509118, CNS-0406328, and EIA-0224377.

* Corresponding author.
E-mail addresses: chenyon1@iit.edu (Y. Chen), sun@iit.edu (X.-H. Sun), wuming@iit.edu (M. Wu).

0743-7315/$ – see front matter © 2008 Elsevier Inc. All rights reserved.
doi:10.1016/j.jpdc.2008.06.007

number of processors. The achieved speed is defined as the work divided by the execution time. An algorithm-machine combination is defined as scalable if the achieved average unit speed of the algorithm on the given machine can remain constant while increasing the number of processors, provided the problem size can be increased with the system size. By this definition, the scalability function $\psi(p, p')$ is defined as

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

where p and p' are the initial and scaled number of processors, and W and W' are the initial and scaled work (problem size) respectively. Requiring a larger problem size incremental to maintain the average unit speed results in a lower scalability. The isospeed scalability model works well in homogeneous environments and is well cited in scholarly publications [2,5,8,12,15,24], including several widely used textbooks.

There is another well-known scalability metric, isoefficiency scalability [11]. The isoefficiency scalability is defined as the ability of a parallel machine to keep the parallel efficiency constant when the system and problem size increase, where the parallel efficiency is defined as the speedup over the number of processors. Speedup, in turn, is defined as the ratio of sequential execution time and parallel execution time. In theoretical analysis, the requirement of sequential execution time does not appear to be a problem. In practice, the performance of sequential processing varies with the problem size on advanced multi-hierarchy memory architectures. Running a large application on a single node of a parallel system is problematic, if not impossible. Scalability measures the ability of parallel systems at different system and problem size and does not need to refer single-node sequential execution time of large scale computing. Isoefficiency scalability may have some difficulty to be extended to general heterogeneous environments. The relation between the isoefficiency and isospeed scalability was studied in [12,23].

Isospeed scalability uses the average unit speed as efficiency and does not refer sequential execution time. It is more appropriate for high-speed computing practice. However, similar to the isoefficiency scalability, it is based on the assumption that the underlying parallel machine is homogeneous. This assumption does not stand for many modern computing systems. It is necessary to extend the isospeed scalability metric to general parallel computing environments. In this study, we combine the merits of both isospeed and isoefficiency scalability to propose a new generalized scalability model.

There are some recent attempts to generalized scalability metrics. Pastor and Bosque proposed a heterogeneous efficiency function to define the heterogeneous scalability [2,15]. Their work extended the homogeneous isoefficiency model to heterogeneous computing. Jogalekar and Woodside proposed a strategy-based scalability metric for general distributed systems [9]. The metric is based on the productivity which is defined as the value delivered by the system divided by its cost (money charge) per unit time. A system is scalable if the productivity keeps pace with the cost. Their metric measures the worthiness of renting a service. However, commercial charge varies from customer to customer based on business considerations and does not necessarily reflect the inherent scalability of the underlying computing system. These recent studies demonstrated the importance of the scalability analysis for heterogeneous computing but have their own limitations.

3. Isospeed-efficiency (Isospeed-e) scalability

In a general heterogeneous environment, a code runs on a tightly coupled or distributed system. We often refer to the code as the algorithm behind it to emphasize the importance of the scalability analysis of the algorithm. Thus, we choose the term algorithm-system combination, instead of code-machine combination, for the scalability study. To completely describe the attributes of a given algorithm-system combination, we need to characterize all computing features of the system including the CPU frequency, memory capacity and speed, network bandwidth, I/O latency and etc. In engineering practice, however, we cannot get into all the details; otherwise, the scalability model will be too complex to use. It is desired to balance the simplicity and the effectiveness. The model should be capable of catching the key features of an algorithm-system combination and hiding the details at the same time. For this reason, we introduce a new concept, *marked-speed*, to describe the aggregate computing power of a general heterogeneous system.

3.1. Definition of marked-speed

Definition 1. The *marked-speed* of a general computing system is defined as the combined *marked-speed* of all nodes in the system, where the *marked-speed* of each node is defined as the (benchmarked) sustained speed, and *speed* is defined as work divided by execution time.

As defined, a general system's marked-speed is the numeric summation of the quantitative marked-speed of all nodes that compose the system. It captures the essential of the computing power and represents the cumulative computational capability of a general parallel/distributed system, but does not represent other non-computation features like the network communication capability. The marked-speed can be calculated based on the hardware peak performance, which in general is much higher than an actual delivered performance. In practice, we can use standard benchmarks, such as Linpack [13], NPB [14] or an appropriate benchmark from the Perfect benchmarks suite [16], to measure each node's sustained speed and calculate the whole system's marked-speed. To guarantee the comparability, we should use the same benchmark for measurement. We will demonstrate the usage of marked-speed in the analytical study section. The marked-speed is a quantitative measurement of computational power [25,2,15].

Let C denote the marked-speed of the computing system and C_i denote the marked-speed of node i. In a heterogeneous environment, C_i might be different from each other due to the heterogeneity of the nodes. In a homogeneous environment, all C_i are the same. According to Definition 1, we have $C = \sum_{i=1}^{p} C_i$ in a general parallel/distributed computing environment with p nodes. In a homogeneous environment, we have $C = \sum_{i=1}^{p} C_i = p.C_1$ because all C_i are the same.

3.2. Definition of isospeed-efficiency scalability

While the marked-speed for a given benchmark is a constant, the actual achieved speed of an application may vary with the system and problem size and may not be the same as the marked-speed. This is especially true for parallel and distributed processing where communication overhead is a major factor of actual achieved speed. We introduce another concept, *speed-efficiency*, to characterize the performance gain of an algorithm-system combination.

Definition 2. The *Speed-efficiency* of an algorithm-system combination is defined as the achieved speed of the algorithm on the system divided by the marked-speed of the system.

Table 1
Comparison of Isospeed and Isospeed-e Scalability

Isospeed metric (Applied to homogeneous systems only)		Isospeed-e metric (Applied to both homogeneous and heterogeneous systems)	
Speed	$\frac{W}{T}$	Achieved speed	$\frac{W}{T}$
Average speed	$\frac{W}{pT}$	Speed-efficiency	$\frac{W}{TC}$
Isospeed condition	$\frac{W}{pT} = \frac{W'}{T'p'}$	Isospeed-efficiency condition	$\frac{W}{TC} = \frac{W'}{T'C'}$
Scalability function	$\psi(p, p') = \frac{p'W}{pW'}$	Scalability function	$\psi(C, C') = \frac{C'W}{CW'}$

Let S denote the achieved speed, W denote the work and T denote the execution time, we have $S = \frac{W}{T}$. Let E_s stand for the speed-efficiency. Thus, we have $E_s = \frac{S}{C} = \frac{W}{TC}$. In homogeneous environments, the speed-efficiency becomes the same as the definition presented in [22] because each node has the same marked-speed and the marked-speed of the system can be expressed by using the system size p.

Based on the previous definitions and discussion, we propose the following *isospeed-efficiency scalability (isospeed-e scalability in short)* for any algorithm-system combination on a general parallel/distributed computing system.

Definition 3 (*Isospeed-e Scalability*). An algorithm-system combination is *scalable* if the achieved speed-efficiency of the combination can *remain constant* with increasing the system ensemble size, provided the problem size can be increased with the system size.

The proposed isospeed-e scalability model does not restrict the underlying system and is applicable to both homogeneous and heterogeneous systems. The method for increasing the system ensemble size includes increasing nodes or the number of processors within nodes, or upgrading to more powerful nodes. The approach to increasing the problem size depends on the algorithm.

3.3. Isospeed-e scalability function

For a scalable algorithm or application, its communication requirement should not increase faster than its computation requirement. Therefore, we can increase the problem size to keep the speed-efficiency constant when the system size is increased. The increment of the problem size depends on the underlying computing system and the algorithm itself. This variation provides a quantitative measurement of the scalability. The marked-speed introduced previously is an appropriate representation of the computational capability, thus we use it to represent a general system and call a system with marked-speed C as a system with system size C in the rest of this study.

Let C be the initial system size of a specified computing system, W and T be the initial problem size and the execution time. Let C' be the scaled system size, W' be the increased problem size and T' be the new execution time for the scaled problem size. We define the *isospeed-e scalability function* as:

$$\psi(C, C') = \frac{C'W}{CW'}$$

where W' is constrained by the isospeed-efficiency condition:

$$\frac{W}{TC} = \frac{W'}{T'C'}.$$

In the ideal situation, there is no communication necessary, which means that $W' = C'W/C$ and thus $\psi(C, C') = 1$. Generally, $W' > C'W/C$ and $\psi(C, C') < 1$.

If we apply the isospeed-e scalability to a homogeneous environment, we have $C = pC_1$, and $C' = p'C_1$ because all C_i are the same. The scalability function becomes:

$$\psi(C, C') = \frac{C'W}{CW'} = \frac{p'W}{pW'}.$$

This shows that the original homogeneous isospeed scalability model is a special case of the isospeed-e scalability model. Table 1 compares the isospeed and the isospeed-e scalability in detail.

3.4. Theoretical studies

We have analyzed the proposed isospeed-e scalability model in theory for further understandings of scalability studies, and this section presents the analysis results.

Theorem 1. *Suppose that an algorithm has a balanced workload on each node and the sequential portion (which cannot be parallelized) of the algorithm is α. If we can find a problem size to keep the speed-efficiency constant when the system size is increased, then the system is scalable and the scalability is:*

$$\psi(C, C') = \frac{t_0 + T_o}{t_0' + T_o'}$$

where t_0 and t_0' are the execution time of the sequential portion, T_o and T_o' are the communication overhead of system C and C' separately.

Analysis: The condition that "the sequential portion of the algorithm is α" means that only $(1 - \alpha)W$ of work can be parallelized. The αW of work must be computed sequentially on one node. The assumption that "the algorithm has a balanced workload on each node" means that the workload is distributed evenly among all computing nodes. Since C_i and C are the marked-speed of the ith node and the whole computing system, this assumption is translated to that the assigned workload on node i is $W_i = (1 - \alpha)W\frac{C_i}{C}$. The parallel execution time can be divided into two parts, $T = T_c + T_o$, where T_c is the computation time, and T_o is the total communication overhead spent on message passing, data transmission, synchronization and etc.

Proof. If a system C is used to compute a problem with size W, and W' is the increased problem size to satisfy the isospeed-efficiency condition when the computing system is scaled to C', we have

$$\frac{W}{(T_c + T_o)C} = \frac{W'}{(T_c' + T_o')C'}.$$

Since the workload is distributed evenly and the sequential portion of the algorithm is α, we have

$$W_i = (1 - \alpha)W\frac{C_i}{C}.$$

Thus,

$$T_c = \frac{W_i}{C_i} + t_0 = \frac{(1 - \alpha)W}{C} + t_0$$

where $t_0 = \frac{\alpha W}{C_j}$, which represents the execution time of the sequential portion of the algorithm, and C_j is the marked-speed of node j where the sequential computation happens on.

Hence,

$$\frac{W}{\left[(\frac{(1-\alpha)W}{C} + t_0) + T_o\right] \cdot C} = \frac{W'}{\left[(\frac{(1-\alpha)W'}{C'} + t_0') + T_o'\right] \cdot C'}$$

where $t'_0 = \frac{\alpha W'}{C_{j'}}$ and the sequential portion is computed on node j' when the system size is scaled.

Then,

$$\frac{W}{(1-\alpha)W + Ct_0 + CT_o} = \frac{W'}{(1-\alpha)W' + C't'_0 + C'T'_o}.$$

Thus,

$$(1-\alpha)WW' + (C't'_0 + C'T'_o)W = (1-\alpha)WW' + (Ct_0 + CT_o)W'.$$

The scaled problem size W', therefore, is

$$W' = \frac{C't'_0 + C'T'_o}{Ct_0 + CT_o} \cdot W = \frac{C'(t'_0 + T'_o)}{C(t_0 + T_o)} \cdot W.$$

Thus, the computing system is scalable and the scalability is

$$\psi(C, C') = \frac{C'W}{CW'} = \frac{C'W}{C \cdot \frac{C'(t'_0+T'_o)}{C(t_0+T_o)} \cdot W} = \frac{t_0 + T_o}{t'_0 + T'_o}. \quad \blacksquare$$

Theorem 1 provides a method to calculate the scalability of an algorithm-system combination, and also shows an insightful understanding of the scalability. It reflects that the scalability is decided by both the sequential portion of the work and the communication overhead. When the problem size is scaled to keep the speed-efficiency constant, the sequential portion of the work is increased, as well as the communication overhead due to scaled system size. Therefore, the scalability is likely to be smaller than 1 in practice.

Corollary 1. *If an algorithm can be parallelized perfectly and has a balanced workload on each node, and if the communication overhead is constant for any problem size and system size, then the algorithm-system combination is scalable and the scalability is perfect with a constant value 1.*

Proof. According to Theorem 1, we have

$$\psi(C, C') = \frac{t_0 + T_o}{t'_0 + T'_o}.$$

In the ideal case, the algorithm can be parallelized perfectly, which means that $\alpha = 0$. Thus, $t_0 = t'_0 = 0$.

If the communication overhead is constant at any problem size and system size, we have $T_o = T'_o$.

Therefore, the scalability is

$$\psi(C, C') = \frac{t_0 + T_o}{t'_0 + T'_o} = 1. \quad \blacksquare$$

Corollary 1 analyzes the scalability of an ideal case. According to the previous discussion, the scalability of an ideal case is 1. Corollary 1 also reveals all the conditions that a perfectly scalable algorithm-system combination requires.

Corollary 2. *If an algorithm can be parallelized perfectly and has a balanced workload on each node, and if we can find a problem size to keep the speed-efficiency constant when the system size is increased, then the algorithm-system combination is scalable and the scalability is*

$$\psi(C, C') = \frac{T_o}{T'_o}.$$

Proof. Similar to the proof in Corollary 1, if the algorithm can be parallelized perfectly, we have $\alpha = 0$ and $t_0 = t'_0 = 0$. According to Theorem 1, the scalability is

$$\psi(C, C') = \frac{t_0 + T_o}{t'_0 + T'_o} = \frac{T_o}{T'_o}. \quad \blacksquare$$

Corollary 2 shows another meaningful understanding of the scalability and is useful in analyzing the scalability of an algorithm-system combination. It demonstrates that if an algorithm can be parallelized perfectly and has a balanced workload on each node, then the scalability will only be decided by the communication overhead at different system sizes.

In practice, we usually compute the sequential portion of the algorithm on the same node before and after the system is scaled. The following theorem analyzes the scalability in this situation.

Theorem 2. *Let an algorithm have a balanced workload on each node and the sequential portion of the algorithm be α. Suppose that the sequential portion of the algorithm is computed on the same node before and after the system is scaled. If we can find a problem size to keep the speed-efficiency constant for the initial system C and the scaled system C', then the system is scalable and the scalability is*

$$\psi(C, C') = \frac{C\beta W - C'\beta W + CT_o}{CT'_o}$$

where $\beta = \alpha/C_i$, C_i is the marked-speed of the node where the sequential portion of the algorithm is computed, W is the initial problem size, T_o and T'_o are the communication overhead for system C and C', respectively.

Proof. The proof of Theorem 2 is similar to that of Theorem 1. We only need to notice that $t_0 = \alpha W/C_i$, and $t'_0 = \alpha W'/C'_i$. Since the sequential portion of the algorithm is computed on the same node, we have $C_i = C'_i$. Let $\beta = \alpha/C_i$, thus $t_0 = \beta W$ and $t'_0 = \beta W'$. According to the isospeed-efficiency condition, we have:

$$\frac{W}{(T_c + T_o)C} = \frac{W'}{(T'_c + T'_o)C'}.$$

Then

$$\frac{W}{\left[(\frac{(1-\alpha)W}{C} + t_0) + T_o\right] \cdot C} = \frac{W'}{\left[(\frac{(1-\alpha)W'}{C'} + t'_0) + T'_o\right] \cdot C'}.$$

So,

$$(1-\alpha)WW' + (C't'_0 + C'T'_o)W = (1-\alpha)WW' + (Ct_0 + CT_o)W'.$$

Thus,

$$C'T'_oW = (C\beta W + CT_o - C'\beta W)W'.$$

Therefore, the increased problem size W' is

$$W' = \frac{C'T'_oW}{C\beta W - C'\beta W + CT_o}.$$

Thus, the computing system is scalable and the scalability is

$$\psi = \frac{C'W}{CW'} = \frac{C'W}{C} \cdot \frac{C\beta W - C'\beta W + CT_o}{C'T'_oW} = \frac{C\beta W - C'\beta W + CT_o}{CT'_o}. \quad \blacksquare$$

These theorems and corollaries reveal that if we are able to analyze the communication overhead at system size C and C', the workload and the sequential ratio of the algorithm, we can calculate and predict the scalability of a system with size C' based on the system with size C. We will show these methods in experimental analysis.

3.5. Calculation of isospeed-e scalability

The isospeed-e scalability can be obtained in many ways. The most straightforward way is to compute the scalability. This

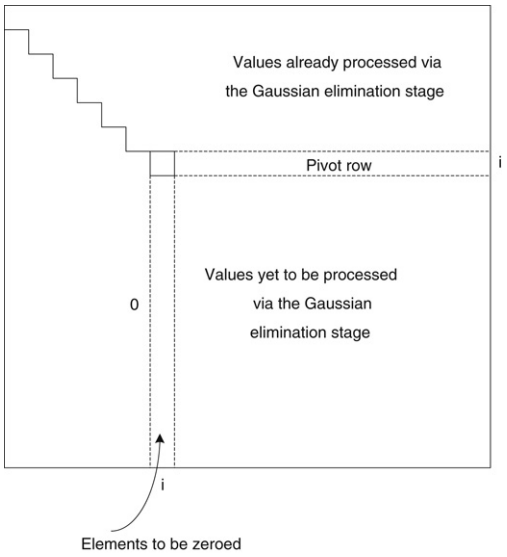

Fig. 1. Gaussian Elimination algorithm.

method measures the execution time at different system and problem sizes and computes the scalability according to the isospeed-e scalability definition.

Another approach is to analyze and predict the scalability. This method examines the computational and communicational ratio of the algorithm, as well as the communication latency of the machine, and then utilizes derived theoretical analysis results to predict the scalability based on the measurements of base cases. This method can also be used to verify the computed scalability. The third approach is to measure the scalability directly when scaling the problem size to maintain the isospeed-efficiency [22]. The experiments given in Section 4 illustrate the computation and prediction of the isospeed-e scalability.

4. Experimental results and analyses

We have carried out experimental testing to verify the proposed isospeed-e scalability model and the theoretical analysis results, and to demonstrate the isospeed-e scalability is practically applicable.

4.1. Algorithms and implementations

Two classical algorithms, Gaussian Elimination (GE) and Matrix Multiplication (MM) algorithms, and one application, 2D Convolution, were selected for testing. Gaussian Elimination and Matrix Multiplication algorithms are widely used in scientific computing. 2D Convolution is a real image processing application, which conducts 2D convolution on two images.

4.1.1. Gaussian Elimination algorithm

Gaussian Elimination algorithm solves dense linear equations $Ax = b$ where A is a known matrix of size $N \times N$, x is the required solution vector, and b is a known vector of size N. The algorithm has two stages:

(1) Gaussian elimination stage. In this stage, the original equations are reduced to an upper triangular form $Ux = y$, where U is a matrix of size $N \times N$ in which all elements below the diagonal are zeros and the diagonal elements have the value 1. The vector y is the modified version of vector b. This stage is composed of $N - 1$ steps. The ith step eliminates nonzero sub-diagonal elements in column i by scaling the ith row by the factor A_{ji}/A_{ii} and subtracting it from row j in the range $[i + 1, N]$ to make the element A_{ji} zero in each case. Fig. 1 demonstrates the ith step of the Gaussian elimination stage.

(2) Back substitution stage. In this stage, the new equations are solved to obtain the value of x.

The parallel Gaussian Elimination algorithm used in the experiment is described as following.

(1) Process 0 distributes the data of matrix A and vector b to other nodes proportionally according to their marked-speeds following the row-based heterogeneous cyclic distribution [10]
(2) All processes compute concurrently:
  (2.1) For $(i = 0; i < N - 1; i++)$
    (2.1.1) The process which owns the pivot row broadcasts the pivot row to all processes
    (2.1.2) For $(j = i + 1; j < N; j++)$
      (a) Each process judges if row j belongs to itself or not
      (b) If yes, then it conducts Gaussian elimination on this row.
  (2.2) Synchronize all processes due to data dependence
(3) Process 0 collects intermediate results from other processes and conducts the back substitution stage.

We can analyze the sequential time complexity of the Gaussian Elimination algorithm. The Gaussian elimination stage has three levels of loop and each loop has N iterations. Thus, the time complexity of this stage is $O(N^3)$, where N is the rank of matrix A. Similarly, the back substitution stage has $O(N^2)$ time complexity. Therefore, the total time complexity of the Gaussian Elimination algorithm is $O(N^3)$. After analyzing the algorithm precisely and counting the factor of each term, the total workload of this algorithm is:

$$W(N) = \frac{2}{3}N^3 - \frac{1}{2}N^2 - 3\frac{1}{6}N + 3.$$

This formula was used to calculate the workload of Gaussian Elimination algorithm in the experiments. The implementation of this algorithm followed the same technique adopted in the Matrix Multiplication experiment, which is explained in detail in the following section.

4.1.2. Matrix Multiplication algorithm

Matrix Multiplication algorithm calculates the product of two matrices, $C = A \times B$. For simplicity, we restrict matrix A and B to be square $N \times N$ matrices. There are many classical parallel algorithms for matrix multiplication [7], such as Cannon's algorithm and the outer product algorithm used in ScaLAPACK [18]. But most of these algorithms are based on homogeneous environments. Literature [1] conducted a thorough research for matrix multiplication optimization on heterogeneous platforms. It stated that the matrix multiplication optimization problem on heterogeneous platforms is a problem to balance the workload with different speed resources and minimize the communication overhead. Unfortunately, this problem has been proven to be an NP-Complete problem. A polynomial heuristic algorithm called Optimal Column-based Tiling was thus proposed and proven to be a good solution for heterogeneous platforms.

Our experiments were designed to verify the proposed scalability model. There is no intent to introduce new algorithms for heterogeneous platforms. We have implemented a straightforward row-based heuristic algorithm. This algorithm adopts the HoHe strategy [10] which distributes homogeneous processes over different speed processors with each process running on a separate processor, while the distribution of matrices over the processes following the heterogeneous block cyclic pattern. In our algorithm, process 0 distributes matrix A following a row-based heterogeneous block distribution initially, which means that A is distributed

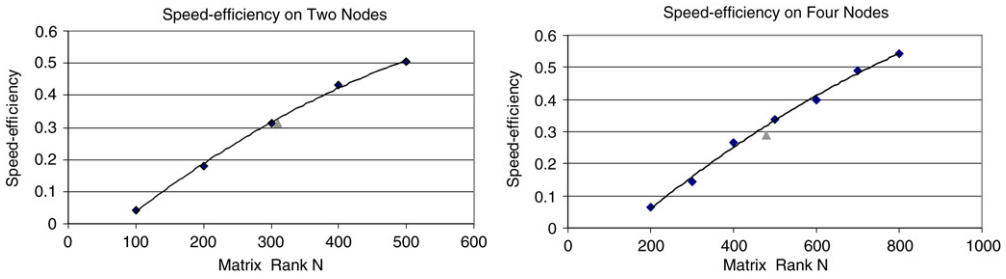

Fig. 2. Speed-efficiency on two and four nodes of Sunwulf.

proportionally into other nodes according to these nodes' marked-speeds. Then process 0 distributes matrix B to other nodes. After data distribution, each node computes part of the matrix multiplication on its own data. Finally, process 0 collects all results from other processes. Since there is no communication during the computation, the communication only occurs in data distribution and collection stages. This algorithm is not a perfect algorithm, but this algorithm does balance the workload between different speed resources, since each node works on $N \times \frac{C_i}{C}$ rows of data and the workload of each node is $2 \times N^3 \times \frac{C_i}{C}$. The total workload of our algorithm is $W(N) = 2 \times N^3$. Considering the problem to balance the workload and minimize the communication is an NP-Complete problem, this algorithm is a minimum-communication-times approximation solution. This approximation is practical because the communication time is composed of a startup time and data transmission time, and the startup time is much greater than the transmission time for transferring a small amount of data. Several parallel matrix multiplication algorithms partition matrix to submatrices and shift to horizontal or vertical neighbors to pipeline the computation and communication. However, some of these algorithms are based on special architectures, such as mesh or torus architecture. They might not be as efficient as expected on general parallel platforms in practice because they require too many communication operations.

The implementation of our algorithm followed the HoHe strategy, which generated the same number of processes as the number of processors and distributed each process on a separate processor. As stated in [10], traditional high-level parallel programming tools lack the facilities to support programmers to operate with quantitative characteristics of heterogeneous platforms, such as the speed of processors. Low-level tools, such as PVM and MPI, allow programmers to write parallel applications adaptable to the performance of processors. This is not supported directly by the tools, however, and is extremely complicated. In our experiments, we created a static configuration file to store quantitative characteristics of heterogeneous processors, which are the marked-speed in our study. These marked-speeds of all nodes were pre-measured and stored as a ⟨hostname, marked-speed⟩ pair in the configuration file. After the program was started, process 0 opened this file and read in the marked-speed of each node. When process 0 distributing the data, it partitioned the data of matrices proportionally following the row-based heterogeneous cyclic distribution and sent the partitioned data to each process.

4.1.3. 2D Convolution application

The 2D Convolution application performs 2D convolution on two $N \times N$ images, where each element is a complex number. It was implemented by taking a 2D Fast Fourier Transform (FFT) of each input image first, then performing a point-wise multiplication of the intermediate results, followed by an inverse 2D FFT. The 2D-FFT was implemented by performing N times of N-point 1D-FFT along rows first, followed by N times of N-point 1D-FFT along columns of the intermediate results of the row FFT. A typical 1D-FFT algorithm was developed by Cooley and Tukey in 1965. The procedure of the 2D convolution is shown as following:

$A = \text{2D-FFT(image1)}$
$B = \text{2D-FFT(image2)}$
$C = \text{MM\_Point}(A, B)$
$D = \text{Inverse-2DFFT}(C)$

where A, B, C, and D are $N \times N$ matrices of complex numbers, and D is the final output.

We have implemented the 2D Convolution on heterogeneous platforms and adopted the HoHe strategy for load balancing. The algorithm is as following.

(1) Process 0 reads in image data (matrices) from input files, and all other processes create the sub-image for the part of data they will work on.
(2) Process 0 distributes the data of matrix A and B to other nodes proportionally according to their marked-speeds following the row-based heterogeneous block distribution.
(3) Each process computes forward 2D-FFT on its two sub-images concurrently.
(4) Each process computes point-wise multiplication on its two sub-images and obtains the intermediate sub-image.
(5) Each process computes inverse 2D-FFT on its intermediate sub-image.
(6) Process 0 gathers the results from all other processes and outputs the final result.

The total workload of the 2D Convolution algorithm is:

$$W(N) = 66N^2 \lg N + 21N^2 + 84N \lg N.$$

Table 2
Marked-speed of Sunwulf nodes(Unit: Mflops)

Node	Server node with 1 CPU	SunBlade compute node	SunFire V210 compute node with 1 CPU
Marked-speed	20.88	20.29	36.45

4.2. Experimental setup

Our experiments were conducted on the heterogeneous Sunwulf compute farm in the Scalable Computing Software (SCS) laboratory at Illinois Institute of Technology. Sunwulf compute farm is composed of one SunFire server node (Sunwulf node), 64 SunBlade compute nodes (hpc-1 to hpc-64) and 20 SunFire V210 compute nodes (hpc-65 to hpc-84). The server node has four CPUs with 480 MHz and 4 GB memory. The SunBlade compute node has one 500-MHz CPU and 128 MB memory. The SunFire V210 compute node has two 1 GHz CPUs and 2 GB memory. The network connecting all these nodes is 100 Mbps Ethernet. The software platform included SunOS 5.8 and MPICH 1.2.5 release.

4.3. Measuring the marked-speed

The NASA Parallel Benchmark [14] was adopted to measure the marked-speed. We run every benchmark on each node and took the average speed on each node as its marked-speed. Table 2 gives

Table 3
Workload, execution time, achieved speed and speed-efficiency of Gaussian Elimination algorithm at different problem sizes on two nodes of Sunwulf

	Matrix rank N	Workload W	Execution time T (ms)	Achieved speed (Mflops)	Speed-efficiency
Case 1.1	100	661 353	260.770	2.536	0.041
Case 1.2	200	5 312 703	473.786	11.213	0.181
Case 1.3	300	17 954 053	925.242	19.405	0.313
Case 1.4	400	42 585 403	1587.725	26.822	0.432
Case 1.5	500	83 206 753	2657.918	31.305	0.505

Table 4
Workload, execution time, and speed-efficiency of Gaussian Elimination algorithm at different problem sizes on four nodes of Sunwulf

	Matrix rank N	Workload W	Execution time T (ms)	Achieved speed (Mflops)	Speed-efficiency
Case 2.1	200	5 312 703	787.315	6.748	0.066
Case 2.2	300	17 954 053	1227.864	14.622	0.142
Case 2.3	400	42 585 403	1555.409	27.379	0.267
Case 2.4	500	83 206 753	2398.865	34.686	0.338
Case 2.5	600	1.44E + 08	3503.558	41.049	0.399
Case 2.6	700	2.28E + 08	4542.754	50.282	0.490
Case 2.7	800	3.41E + 08	6137.099	55.565	0.541

Table 5
Required matrix size to maintain speed-efficiency of Gaussian Elimination algorithm on Sunwulf

	System configuration	Matrix size N	Workload W	Marked-speed (Mflops)
Case 1	2 nodes, C_2	310	19 811 638	62.05
Case 2	4 nodes, C_4	480	73 611 283	102.63
Case 3	8 nodes, C_8	1000	6.66E + 08	183.79
Case 4	16 nodes, C_{16}	1700	3.27E + 09	346.11
Case 5	32 nodes, C_{32}	3200	2.18E + 10	670.75

Table 6
Computed scalability of Gaussian Elimination algorithm on Sunwulf

$\psi(C_2, C_4)$	$\psi(C_4, C_8)$	$\psi(C_8, C_{16})$	$\psi(C_{16}, C_{32})$
0.445	0.198	0.383	0.290

the measured marked-speed of the server node with one CPU, the SunBlade compute node and the SunFire V210 compute node. The marked-speed of a specific computing system can be calculated according to Definition 1. For example, if we choose the following nodes to participate computation: server node with one CPU, one SunBlade compute node and two SunFire compute nodes with one CPU, the marked-speed of this computing system is:

$$20.88 + 20.29 + 2 \times 36.45 = 114.07 \text{ (Mflops)}.$$

4.4. Experimental results and analyses

4.4.1. Gaussian Elimination experimental results

The first set of experiments aimed to apply the proposed isospeed-e scalability to analyze the Gaussian Elimination algorithm with different system configurations on the Sunwulf compute farm. We started with two nodes, one SunBlade node and the server node. In this case, server node utilized two CPUs. According to Table 2, the marked-speed of this environment is:

$$C_2 = 20.88 \times 2 + 20.29 = 62.05 \text{ (Mflops)}.$$

Table 3 shows the workload, execution time, achieved speed and speed-efficiency of Gaussian Elimination at different matrix sizes on two nodes. The speed-efficiency was calculated according to Definition 2.

Based on the experimental results, we plot the relationship between the speed-efficiency and the matrix size in the left figure of Fig. 2. According to the definition of the speed-efficiency and the workload formula, the function between the speed-efficiency and the matrix size is polynomial, thus we applied a polynomial trend line to approach the sample results. From the polynomial trend line, we can read the approximate value of the speed-efficiency at any matrix size or read the approximate required matrix size to obtain a specified speed-efficiency. For example, if we want to obtain a speed-efficiency of 0.3, the required matrix size should be around 310. We measured the speed-efficiency when matrix size is 310 and the result is 0.312, which is shown with a gray triangle in Fig. 2. This verifies the method that we read the required matrix size for a specified speed-efficiency from trend line works.

Next we scaled the system size to four nodes and the configuration of these four nodes was also changed. The new computing system was composed of hpc-40, hpc-41, hpc-42 and the server node with two CPUs. Similar to the analysis in the case of two nodes, we calculated the workload and marked-speed, then computed the speed-efficiency. In this case, the marked-speed was $C_4 = 102.63$ Mflops. Table 4 shows the test results.

The speed-efficiency with four nodes is also shown in the right figure of Fig. 2. Similar to the analysis in the case of two nodes, the required matrix size to obtain a 0.3 speed-efficiency was around 480. The measured speed-efficiency with matrix size 480 was 0.288. It also verifies the observation method to obtain the required matrix size for a specified speed-efficiency works well.

Based on these results, we are able to follow the first method in Section 3.5 to calculate the scalability when the system scaled from two nodes to four nodes. Since the required matrix size was around 310 in the case of two nodes and 480 in the case of four nodes to maintain the speed-efficiency, and recall the marked-speed was 62.05 and 102.63 respectively, according to the scalability function definition, we have

$$\psi(C_2, C_4) = \frac{C_4 \cdot W(N)}{C_2 \cdot W(N')} = 0.445.$$

Similar to the previous analysis, we obtained the results in the case of 8 nodes, 16 nodes and 32 nodes. In each case, one node was the server node and the rest nodes were SunBlade compute nodes. The marked-speed of the computing system in each case is denoted as C_8, C_{16}, C_{32} respectively. The required matrix size to maintain the speed-efficiency, along with the workload and the marked-speed are shown in Table 5.

The computed scalability of Gaussian Elimination algorithm on Sunwulf is shown in Table 6.

Table 7
System configuration of each case in matrix multiplication experiment

	System configuration	Server node	SunBlade compute node	SunFire compute node	Marked-speed (Mflops)
Case 1	2 nodes, C'_2	1	0	1	57.33
Case 2	4 nodes, C'_4	1	1	2	114.07
Case 3	8 nodes, C'_8	1	3	4	227.55
Case 4	16 nodes, C'_{16}	1	7	8	454.51
Case 5	32 nodes, C'_{32}	1	15	16	908.43

Table 8
Matrix multiplication experimental results

	System configuration	Matrix size N	Workload W	Marked-speed (Mflops)
Case 1	C'_2	165	8 984 250	57.33
Case 2	C'_4	255	33 162 750	114.07
Case 3	C'_8	430	1.59E + 08	227.55
Case 4	C'_{16}	710	7.16E + 08	454.51
Case 5	C'_{32}	1150	3.04E + 09	908.43

Table 9
Computed scalability of Matrix Multiplication algorithm on Sunwulf

$\psi(C'_2, C'_4)$	$\psi(C'_4, C'_8)$	$\psi(C'_8, C'_{16})$	$\psi(C'_{16}, C'_{32})$
0.539	0.416	0.443	0.470

4.4.2. Matrix Multiplication experimental results

Another set of experiments we carried out to study the isospeed-e scalability was Matrix Multiplication algorithm on Sunwulf compute farm. We tested the algorithm on 2, 4, 8, 16 and 32 nodes respectively. The system configuration was heterogeneous in each case, where half nodes were SunBlade compute nodes and the other half nodes were SunFire V210 nodes except one node was the server node. For instance, in the case of 8 nodes, the computing system was composed of one server node, three SunBlade compute nodes and four SunFire V210 compute nodes. The marked-speed of the system in each case was different with previous experiment due to different configurations. For example, the marked-speed is: $C'_8 = 20.88 + 3 \times 20.29 + 4 \times 36.45 = 227.55$ (Mflops) in the case of 8 nodes. The detailed system configuration and the marked-speed in each case is shown in Table 7.

The testing procedure was similar to the previous experiment. The details are omitted here. The speed-efficiency of Matrix Multiplication algorithm at different system configurations is given in Fig. 3.

Similar to the analysis in the Gaussian Elimination experiment, we computed the scalability following the first method in Section 3.5 when system size changed. We read the required matrix size to maintain the speed-efficiency at different system configurations from Fig. 3. Table 8 shows the results.

According to Table 8, we calculated the isospeed-e scalability for the Matrix Multiplication algorithm on Sunwulf, and the results are shown in Table 9.

4.4.3. Comparison of Gaussian Elimination and Matrix Multiplication experiments

Gaussian Elimination and Matrix Multiplication experiments are actually two different algorithm-system combinations. Both experiments have shown that the isospeed-e scalability model works well for algorithm-system combinations and it provides a practical approach to quantify and compare the scalability of different combinations

Compared with the scalability of GE–Sunwulf combination given in Table 6, the scalability of MM–Sunwulf combination, as shown in Table 9, is higher. This indicates that the MM–Sunwulf combination is more scalable than the GE–Sunwulf combination. The Gaussian Elimination algorithm has a certain part of sequential processing and requires much more communication for data transmission during concurrent computation than that of the Matrix Multiplication algorithm. Its scalability should be lower than the scalability of the latter. Our experimental results verified this fact and presented an actual quantified scalability advantage of the Matrix Multiplication algorithm on Sunwulf.

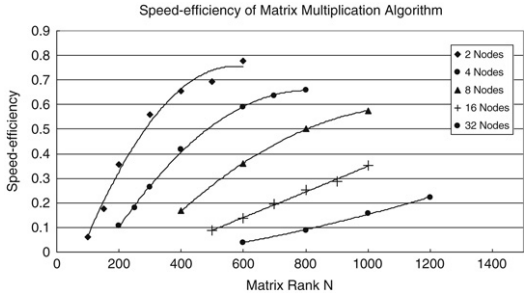

Fig. 3. Speed-efficiency of Matrix Multiplication algorithm at different system configurations.

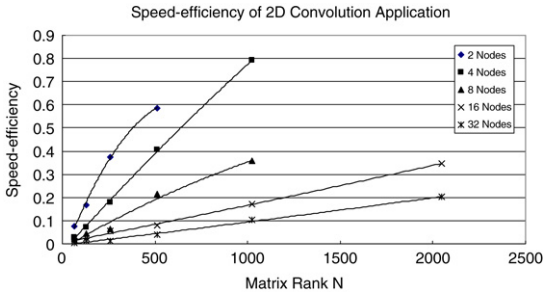

Fig. 4. Speed-efficiency of 2D Convolution application at different system configurations.

4.4.4. 2D Convolution experimental results

We have tested the 2D Convolution application on the same series of system configurations as in the Matrix Multiplication experiment. The testing process was similar to the Matrix Multiplication experiment as well. We first obtained the speed-efficiency diagram in each case, and then read the required matrix size to keep the speed-efficiency and calculated the scalability when system size was changed. The speed-efficiency results of 2D Convolution are shown in Fig. 4.

Table 10 shows the required matrix size to keep the speed-efficiency at different system configurations, and Table 11 shows the scalability results of 2D Convolution application on Sunwulf compute farm.

Table 10
2D Convolution experimental results

	System configuration	Matrix size N	Workload W	Marked-speed (Mflops)
Case 1	C'_2	145	10 492 177	57.33
Case 2	C'_4	270	40 574 873	114.07
Case 3	C'_8	545	1.8E + 08	227.55
Case 4	C'_{16}	1160	9.3E + 08	454.51
Case 5	C'_{32}	2030	3.1E + 09	908.43

Table 11
Computed scalability of 2D Convolution application on Sunwulf

$\psi(C'_2, C'_4)$	$\psi(C'_4, C'_8)$	$\psi(C'_8, C'_{16})$	$\psi(C'_{16}, C'_{32})$
0.515	0.438	0.396	0.606

This set of experiments has validated the proposed isospeed-e scalability via applying it to a real application. Essentially the isospeed-e scalability provides a quantitative evaluation of how scalable an algorithm or an application is on a general computing platform including both homogeneous and heterogeneous environments.

4.5. Scalability prediction

As discussed in Section 3, if we are able to analyze the algorithm and measure the communication latency of the machine, we can predict the scalability of the algorithm-system combination based on the measurements of base cases. We take the GE–Sunwulf combination as an example and illustrate how its scalability can be predicted.

As presented in Section 4.1.1, the sequential part of the parallel Gaussian Elimination algorithm is the back substitution stage, thus, the sequential ratio is $\alpha = O(1/N)$. When N is large enough, we treat $\alpha \approx 0$. The total communication overhead of the algorithm is:

$$T_o = T_{broadcast} + 2 \times (p-1) \times (T_{send} + T_{recv}) + N \times (2 \times T_{broadcast} + T_{barrier})$$

where p is the number of processes. We have measured the parameters in the above equation on Sunwulf and the results are:

$T_{broadcast} \approx 0.12 + p \times 0.23$ (ms)
$T_{send} = T_{recv} \approx 0.08 + (0.00003 \times N)$ (ms)
$T_{barrier} \approx p \times 0.39$ (ms).

The computation time can be written as

$$T_c = \frac{W(N) \times t_c}{p}$$

where t_c is the time of one unit computation. We have measured its value and the result is:

$t_c \approx 3.1 \times 10^{-5}$ (ms).

Based on these analyses and actual parameters, and according to Corollary 2, we predict the scalability as:

$$\psi(C, C') = \frac{T_o}{T'_o}$$

where N appearing within T_o and T'_o is constrained by the isospeed-efficiency condition:

$$\frac{W}{(T_c + T_o)C} = \frac{W'}{(T'_c + T'_o)C'}.$$

The prediction results of the required matrix size N to keep the speed-efficiency constant based on the case of two nodes are shown in Table 12. Compared with the measured required matrix sizes listed in Table 5, the difference between the predicted value and the measured value is trivial, such as $N' = 1683$ and $N = 1700$, $N' = 3226$ and $N = 3200$ in the predicted case and in the measured case respectively. The average prediction error is only about 2.8%. The predicted scalability is shown in Table 13.

Table 12
Predicted required matrix size N' to maintain the speed-efficiency

Nodes	4	8	16	32
N' (prediction)	492	928	1683	3226

Table 13
Predicted scalability of Gaussian Elimination algorithm on Sunwulf

$\psi'(C_2, C_4)$	$\psi'(C_4, C_8)$	$\psi'(C_8, C_{16})$	$\psi'(C_{16}, C_{32})$
0.413	0.267	0.316	0.275

The predicted scalability is a close-match of the computed scalability shown in Table 6, which also verifies the isospeed-e scalability works as expected and the theoretical analysis matches the experimental results. There are some variations between the predicted scalability values and the calculated values. This is because it is particularly hard to obtain the exact machine latency when performing various communication operations, such as $T_{broadcast}$, T_{send}, $T_{receive}$ and $T_{barrier}$ in the analysis. However, the predicted scalability is a good approximation and is practically useful in analyzing the scalability of a large-scale system where the experimental measurement is exceedingly hard or even impossible. The scalability prediction is also helpful in choosing the best algorithm-system combination for a given application.

5. Conclusions and future work

Scalability is an important measurement for analyzing the performance of parallel/distributed computing systems. In this study, we proposed a novel model, named isospeed-efficiency scalability, or isospeed-e scalability in short, for a general computing system. It contains the homogeneous isospeed scalability [22] as a special case. Our main contributions in this study include: (1) introducing a new concept of marked-speed to describe the computational capability of computing systems; (2) presenting an isospeed-e scalability for both homogeneous and heterogeneous computing based on the marked-speed concept; (3) analyzing the new scalability model in theory and deriving useful formulas for calculating and predicting the scalability; and (4) having conducted experiments to validate the proposed isospeed-e scalability. Analytical and experimental results show that the proposed isospeed-e scalability models the scaling characteristics of computing systems well.

The proposed isospeed-e scalability is practical and effective. It works not only on conventional cluster systems, but also on general distributed heterogeneous environments, such as Grid platform. Due to the recent rapid changes in the Grid technology, our current experimental testing was limited on clusters only. We intend to extend the tests of isospeed-e scalability to a Grid platform in future research.

We have demonstrated that the scalability can be predicted based on derived formulas in this study, and we have developed

a scalability testing and analysis toolkit recently [3]. We plan to explore the possibility of extending the prediction of scalability into the system support so that the scalability can be predicted automatically or semi-automatically.

References

[1] O. Beaumont, V. Boudet, F. Rastello, Y. Robert, Matrix multiplication on heterogeneous platforms, IEEE Transaction on Parallel and Distributed Systems 12 (10) (2001) 1033–1051.
[2] J.L. Bosque, L. Pastor, Theoretical scalability analysis for heterogeneous clusters, in: Proceedings of the 4th IEEE/ACM International Symposium on Cluster Computing and the Grid, 2004.
[3] Y. Chen, X.H. Sun, STAS: A Scalability testing and analysis system, in: Proceedings of 2006 IEEE International Conference on Cluster Computing, 2006.
[4] CLUMPs, http://now.cs.berkeley.edu/clumps/.
[5] D. Culler, J. Singh, A. Gupta, Parallel Computer Architecture: A Hardware/Software Approach, Morgan Kaufmann Publishers, 1999.
[6] I. Foster, C. Kesselman, The Grid: Blueprint for a New Computing Infrastructure, 2nd Edition, 2004.
[7] A. Gupta, V. Kumar, Scalability of parallel algorithms for matrix multiplication, in: Proceedings of the 1993 International Conference on Parallel Processing, 1993.
[8] K. Hwang, Z. Xu, Scalable Parallel Computing, McGraw-Hill, 1998.
[9] P.P. Jogalekar, C.M. Woodside, Evaluating the scalability of distributed systems, IEEE Transaction on Parallel and Distributed Systems 11 (6) (2000) 589–603.
[10] A. Kalinov, A. Lastovetsky, Heterogeneous distribution of computations while solving linear algebra problems on networks of heterogeneous computers, Journal of Parallel and Distributed Computing 61 (4) (2001) 520–535.
[11] V. Kumar, A. Grama, A. Gupta, G. Karypis, Introduction to Parallel Computing: Design and Analysis of Parallel Algorithms, 1994.
[12] K. Li, Average-case scalability analysis of parallel computations, Annual Review of Scalable Computing, World Scientific 4 (2002) 51–95.
[13] LINPACK http://www.netlib.org/linpack/.
[14] NAS Parallel Benchmarks http://www.nas.nasa.gov/Software/NPB/.
[15] L. Pastor, J.L. Bosque, An efficiency and scalability model for heterogeneous clusters, in: Proceedings of IEEE International Conference on Cluster Computing, 2001.
[16] Prefect Bechmarks http://www.csrd.uiuc.edu/benchmark/benchmark.html.
[17] S.R. Sarukkai, P. Mehta, R.J. Block, Automated scalability analysis of message-passing parallel programs, IEEE Parallel and Distributed Technology 3 (4) (1995) 21–32.
[18] ScaLAPACK, http://www.netlib.org/scalapack/scalapack_home.html.
[19] X.H. Sun, Scalability versus execution time in scalable systems, Journal of Parallel and Distributed Computing 62 (2) (2002) 173–192.
[20] X.H. Sun, Y. Chen, M. Wu, Scalability of heterogeneous computing, in: Proceedings of the 34th International Conference on Parallel Processing, 2005.
[21] X.H. Sun, L.M. Ni, Scalable problems and memory-bounded speedup, Journal of Parallel and Distributed Computing 19 (1993) 27–37.
[22] X.H. Sun, D. Rover, Scalability of parallel algorithm–machine combinations, IEEE Transaction on Parallel Distributed Systems 5 (1994) 599–613.
[23] X.H. Sun, J. Zhu, Performance considerations: A case study using a scalable shared-virtual-memory machines, IEEE Parallel and Distributed Technology 4 (1996) 36–49.
[24] X. Zhang, Y. Yan, K. He, Latency metric: An experimental method for measuring and evaluating parallel program and architecture scalability, Journal of Parallel and Distributed Computing 22 (1994) 392–410.
[25] X. Zhang, Y. Yan, Modeling and characterizing parallel computing performance on heterogeneous networks of workstations, in: Proceedings of the 7th IEEE Symposium on Parallel and Distributed Processing, 1995.

Xian-He Sun received his B.S. in Mathematics in 1982 from Beijing Normal University, PR China, and completed his M.S. in Mathematics, M.S. and Ph.D. in Computer Science in 1985, 1987, and 1990, respectively, all from Michigan State University, USA. He was a post-doctoral researcher at the Ames National Laboratory, USA, a staff scientist at the ICASE, NASA Langley Research Center, an ASEE fellow at the US Navy Research Laboratories, and was an associate professor in the Department of Computer Science, Louisiana State University before he joined the Computer Science Department, Illinois Institute of Technology (IIT) in August 1999. Currently he is a professor of Computer Science at IIT, a guest faculty in the Mathematics and Computer Science Division at the Argonne National Laboratory, a visiting scientist in the Computing Division at the Fermi National Accelerator Laboratory, and the director of the Scalable Computing Software laboratory at IIT. Dr. Sun's research interests include high-performance computing, performance evaluation, and distributed systems. More information about Prof. Sun can be found at www.cs.iit.edu/~scs/sun.

Yong Chen received his B.E. degree in Computer Engineering in 2000 and M.S. degree in Computer Science in 2003, both from University of Science and Technology of China. He is currently pursuing his Ph.D. degree in Computer Science at Illinois Institute of Technology. His research focuses on parallel and distributed computing in general, and on performance evaluation, optimization, data access performance, and parallel I/O in particular.

Ming Wu received his B.E. degree from Xidian University, China, in 1994, the M.S. degree from University of Science and Technology of China, in 1997, and Ph.D. degree in Computer Science from Illinois Institute of Technology, in 2006. His research interests include Quality of Service support in network computing, performance evaluation and task scheduling systems in distributed computing, failure analysis and modeling in high-performance computing.