STAS: A Scalability Testing and Analysis System

Yong Chen, Xian-He Sun
Illinois Institute of Technology
Chicago, Illinois, 60616, USA
{chenyon1, sun}@iit.edu

Abstract

Scalability is a crucial factor in performance evaluation and analysis of parallel and distributed systems. Much effort has been devoted to scalability research and several metrics are proposed. However, the lacking of an effective scalability analysis toolkit is still a major barrier for researchers to measure and analyze scalabilities. Isospeed scalability is a known metric and has been extended for general computing systems recently. This paper proposes an effective Scalability Testing and Analysis System, called STAS, and presents its implementation with isospeed-e scalability metric. STAS provides the facility to conduct automated isospeed-e scalability measure and analysis. It reduces the burden for users to evaluate the performance of algorithms and systems. Experiments have been conducted to verify the design and implementation.

1. Introduction

Scalability plays an important role in the design of parallel and distributed system machines and algorithms. Generally, the scalability concept can be defined as the ability of a system to keep its performance when the system ensemble size is scaled up. Scalability metrics have been extensively studied in parallel and distributed computing domain. It has been used widely for describing how the system and problem size influence the performance of parallel computers and algorithms. Scalability can also be used to predict the performance of parallel systems at large system size based on their performance at small size. It suggests which parallel computer could be built with more processors and which algorithm might be suitable for a larger computer system.

Although scalability of parallel and distributed systems has been studied intensively, there still no an effective and efficient scalability analysis tool exists for algorithm designers and system researchers. The software infrastructure for evaluating the scalability of algorithms and parallel processing systems has not kept in pace with theoretical research. Lack of such an effective analysis tool is a major barrier for applying scalability metrics in practice, thus hampering the broader use of high performance computing. A system to deliver an integrated performance modeling, measurement and analysis, called Scalability Testing and Analysis System (STAS) is proposed in this study. In contrast to existing performance tools, STAS provides detailed scalability analysis of algorithms and systems based on the novel *isospeed-e scalability* [9]. STAS adopts extendable modular design. Each component is designed and developed separately but coupled tightly together to deliver an effective performance analysis system. STAS performs characterizing the computing system and constructing testing machine set automatically. It supports compiler or users' hints as algorithm workload analysis, and conducts runtime measurement and output scalability analysis.

The rest of this paper is organized as following. Section 2 discusses the isospeed metric and its recent extension, isospeed-e metric. The details of design and implementation of STAS are introduced in Section 3. Section 4 demonstrates how STAS could be applied to conduct scalability measure and analysis for algorithms and systems. Section 5 discusses related work. Finally, we summarize our study and discuss the future work in Section 6.

2. Isospeed-e scalability

Several metrics are proposed to measure the scalability of algorithms and parallel machines [2][4] [7] [9][11], but most of these metrics are designed for homogeneous environments only. In [11], Sun and Rover proposed the *isospeed scalability* metric. An algorithm-machine combination is defined to be scalable if the achieved average unit speed of the algorithm on the given machine can remain constant with increasing number of processors, provided the problem size can be increased with the system size, where the average unit speed is defined as the system's achieved speed divided by the number of processors. The scalability function is $\psi(p,p')=\dfrac{p'W}{pW'}$, 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. The isospeed scalability works well in homogeneous environment and is well cited in scholarly publications, including several widely used textbooks [1][3][4][7][9][10]. However, it is based on the assumption that the underlying parallel machine is homogeneous. This assumption does not stand for many modern computing systems. Our recent study has successfully extended it and proposed a novel metric, *isospeed-efficiency scalability* (isospeed-e in short), for both homogeneous and heterogeneous computing [9].

In *isospeed-e scalability* metric [9], a new concept of *marked speed* is introduced to describe the combined computing power of a general parallel computing system. The marked speed of a computing node is the (benchmarked) sustained speed of that node. It represents the computational capability of that node. It can be calculated based on hardware peak performance, which in general is higher than actually delivered performance. In practice, computation intensive benchmarks can be used to measure the marked speed of each node. Once the marked speed of a computing node is measured, it is used as a constant parameter, like CPU frequency or memory access latency, in all experimental analyses. The marked speed of a computing system is defined as the sum of the marked speed of each node that composes the computing system. Let C_i denote the marked speed of node i. In a heterogeneous environment, C_i may be different from each other due to the heterogeneity. In homogeneous environment, all C_i are the same. Let C stand for the marked speed of a computing system. According to definition, we have $C=\sum_{i=1}^{p} C_i$ in a general parallel 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.

Let S denote the actual achieved speed, which is defined as work divided by execution time, of a computing system. Let W denote work and T denote execution time, we have $S=W/T$. Marked speed describes the computational capability of a computing system, which is a constant for a study. The achieved speed of an application may not be the same as the benchmarked marked speed, especially for distributed/parallel computing where the marked speed does not consider the communication cost. Achieved speed describes the actual computational performance when the system tries to solve users' applications. It varies with the system and problem size. The *speed-efficiency* of a computing system is defined as the achieved speed divided by the marked speed of the computing system. The speed-efficiency reflects the performance gain of an algorithm-system combination. Let E_s denote the speed-efficiency. Then we have $E_s = \frac{S}{C} = \frac{W}{TC}$.

With the support of the new concept of marked speed and the new definition of speed-efficiency, the *isospeed-efficiency scalability* metric is defined. An algorithm-system combination is scalable if the achieved speed-efficiency of the combination can remain constant with increasing system ensemble size, provided the problem size can be increased with the system size. The isospeed-e scalability function is

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

where C and C' are the marked speed of initial and scaled system respectively, W is the initial problem size of a specified algorithm, W' is the increased problem size which is constrained by isospeed-efficiency condition $\frac{W}{TC} = \frac{W'}{T'C'}$. When we apply the isospeed-e scalability to a homogeneous environment, because all C_i are equal, we have $C = pC_i$, and $C' = p'C_i$. Thus, the scalability function is

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

This shows that the original homogeneous isospeed scalability metric is a special case of isospeed-efficiency scalability metric.

3. STAS: Scalability Testing and Analysis System

Scalability metrics have been extensively studied in homogeneous and heterogeneous parallel systems, however, there no a scalability analysis tool exists that performs system and code characterization and scalability measure and analysis automatically and effectively. Such a tool reduces the burden to analyze system scalability for machine designers and provides guidelines to build a scalable machine. It also provides quantitative analysis of algorithm scalability and guides users and the compiler in selecting transformation and optimization strategies. For instance, system designers can use this tool to analyze if a newly released architecture is more scalable than existing ones. Algorithm researchers can use this tool to evaluate if the inherent scalability of code matches their theoretical analysis, to get a better understanding of the algorithm or find the performance bottleneck. These motivate the design of Scalability Testing and Analysis System (STAS). STAS is designed and developed by using the novel isospeed-e scalability metric. It is capable to characterize and describe the scaling features of algorithms and underlying machines. It is suitable for general parallel/distributed system, and can adapt to measure other scalability or performance metrics with minimal effort. STAS provides a realistic scalability analysis tool for designers and researchers.

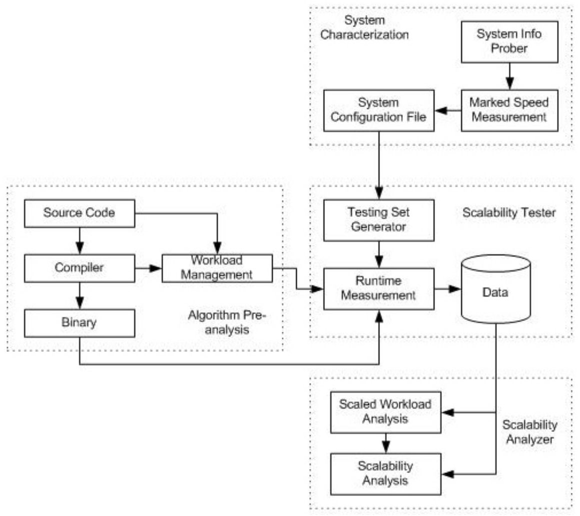

Fig. 1 STAS structure

As shown in Fig. 1, STAS consists of four components: system characterization component, algorithm pre-analysis component, scalability tester and scalability analyzer component. These four components are integrated tightly with each other. The details of design and implementation of STAS are discussed in the following.

3.1. System characterization component

The system characterization component is responsible for obtaining the underlying system information and collecting the marked speed of each node. It has two modules: system information probe module and marked speed measurement module. The output of this component is the system configuration file, which contains pairs of <hostname, marked speed> for each node. The probe module obtains hostname information from system files, and performs checkup to remove redundant hostnames, switches and broken nodes to guarantee only distinct active nodes are kept in system configuration file. The marked speed measurement module measures the marked speed of each node according to the output of system information probe agent. Different standard benchmark suites might be selected as the marked speed benchmark based on the underlying application, as long as the same benchmark is used for all nodes to guarantee the comparability. A common choice of the benchmark is a computational intensive benchmark that shows the best performance a node can deliver. In our current implementation, NPB (NASA Parallel Benchmark) 3.1 suite is chosen because it is one of the widely used scientific application benchmark. The marked speed measurement agent remote login each node and performs benchmark testing. The testing result is filtered to acquire the “Mop/s total” value. The final marked speed of each node is calculated as the average of the collected value of all benchmarks. The marked speed results are written into system configuration file *ms.conf*, which will be fed into scalability tester component.

3.2. Algorithm pre-analysis component

The algorithm pre-analysis component takes the algorithm source code as the input and outputs the binary and workload analysis. In the current implementation, only MPI programs are supported, but this component could be configured to support PVM or HPF programs. The workload analysis is either supported by compiler hints or user hints. We have done the compiler analysis work in [10], but it is not integrated in the current implementation of STAS yet. The modular design allows us to integrate compiler analysis effortlessly in the future. However, the complier analysis has some difficulty in estimating the amount of work of an algorithm automatically. The current implementation of the STAS workload analysis component is supported by users’ hints, with the workload formula and parameters. For instance, the workload formula for the Matrix Multiplication application is $2 \times N^3$, and the workload parameter is the matrix rank *N*. We believe the user’s hints are the most straightforward and effective way to estimate the workload, since algorithm developers know their algorithms well and system designers are likely to use well-studied benchmarks to evaluate their systems.

3.3. Scalability tester component

The scalability tester component has three modules: testing set generator module, runtime measurement module and database module. The testing set generator constructs machine set with different system size, and calculates the marked speed of each machine set. It starts generating a set with size 2, and double the set size every time to generate the next machine set. Each previous machine set is fully included in the next machine set. The generator continues constructing machine set until it reaches the limit. In practice, the heterogeneous system is usually composed of several groups of nodes. The computing capability of nodes in one group is similar to each other, but different nodes from different groups may have large variations in computing capability. In order to fairly represent the system size scaling, the group concept is also introduced in constructing machine set. When nodes are selected to add to a machine set, the group information is considered to balance the nodes from different groups in machine sets. After one machine set is constructed, the marked speed of this machine set is calculated according to the definition. For instance, a machine set with two nodes, where one node has marked speed of 20 Mflops and the other node has marked speed of 30 Mflops, has a marked speed of 50 Mflops.

The machine set, along with workload analysis results and the binary, are fed into runtime measurement module. This module first generates the MPI procgroup file with the machine set and binary file name. Such a procgroup file provides full control to start MPI jobs, which is required to perform scalability measurement on a heterogeneous environment. Fig. 2 shows an example of generated procgroup file in our experiments. In this example, there are three groups of nodes, sunwulf, hpc-1 to hpc-64 and hpc-65 to hpc-84. The machine set size is 8 and is composed of the sunwulf node, three arbitrary nodes from hpc-1 to hpc-64 and four arbitrary nodes from hpc-65 and hpc-84. The binary file is /ufs1/home/scs/yongchen/scal/mm/hetero/hmm. The second functionality of the runtime measurement module is to perform a series of runtime testing with workload parameters for each machine set. The testing command is “mpirun -p4pg <generated procgroup file> <binary file name> <workload parameters>” and is invoked from the runtime measurement module. The execution time is collected and stored in database along with workload formula, tuned workload parameters and the marked speed of the machine set. To reach a target speed-efficiency, an estimated workload is first calculated based on the workload formula, then the workload parameter(s) is tuned with 2% variation (plus or minus 2%) in each testing toward the target speed-efficiency. The speed-efficiency is calculated according to the definition, $E_s=W/TC$. MySQL version 14.7 with distribution 4.1.10a for sun-solaris2.9 is selected as the database support.

```
sunwulf    0    /ufs1/home/scs/yongchen/scal/mm/hetero/hmm
hpc-53     1    /ufs1/home/scs/yongchen/scal/mm/hetero/hmm
hpc-41     1    /ufs1/home/scs/yongchen/scal/mm/hetero/hmm
hpc-22     1    /ufs1/home/scs/yongchen/scal/mm/hetero/hmm
hpc-72     1    /ufs1/home/scs/yongchen/scal/mm/hetero/hmm
hpc-66     1    /ufs1/home/scs/yongchen/scal/mm/hetero/hmm
hpc-81     1    /ufs1/home/scs/yongchen/scal/mm/hetero/hmm
hpc-69     1    /ufs1/home/scs/yongchen/scal/mm/hetero/hmm
```

Fig. 2 Generated Procgroup File Example

3.4. Scalability analyzer component

The scalability analyzer component takes the metadata stored in database as input and analyzes it. The output is the scalability of algorithm-system combination. This component consists of scaled workload analysis module and scalability analysis module. The scaled workload analysis module scans the metadata in database and obtains the required workload size to achieve the user-specified speed-efficiency for each machine set. The scalability analysis module then calculates the scalability by using the isospeed-e scalability function, $\psi(C,C')=\dfrac{C'W}{CW'}$.

4. Experimental results and analyses

In this section, we demonstrate how STAS can be applied to measure and analyze the scalability of algorithm-system combinations. The Sunwulf cluster at the Scalable Computing Software (SCS) laboratory at Illinois Institute of Technology is the underlying testing system. Two classical scientific computing algorithms, Gaussian Elimination and Matrix Multiplication, and one real application, 2-D convolution, are selected as the testing algorithms.

4.1. System characterization

The experimental platform, Sunwulf cluster, 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 and 4GB memory. Each CPU is 480 MHz. The SunBlade compute node has one 500-MHz CPU and 128M memory. The SunFire V210 compute node has two 1GHz CPUs and 2GB memory. The network connecting all these nodes is 100M Ethernet. The software platform includes SunOS 5.8 and MPICH 1.2.5 release version. Fig. 3 shows the measured marked speed of hpc-1 to hpc-84 node through STAS system characterization component (note: hpc-17, hpc-25, hpc-47 and hpc-58 do not work when tested). Though the system configuration of node hpc-1 to hpc-64 or node hpc-65 to hpc-84 is the same with each other, the measured marked speed has slight variation due to varied load of each node when tested. These variations will not affect the scalability analysis. Fig. 4 illustrates a piece of generated system marked speed configuration file, *ms.conf*, which is fed into scalability tester component to perform runtime measurement.

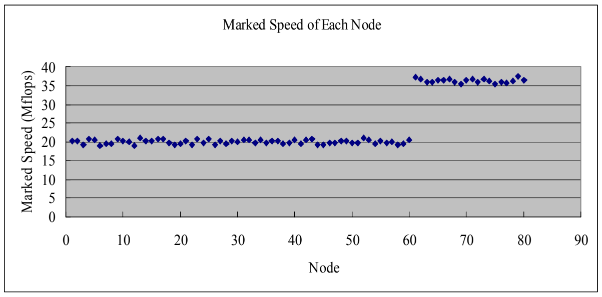

Fig. 3 Measured Marked Speed of Each Node at Sunwulf Cluster

```
      ...
hpc-56   20.56
hpc-57   19.48
hpc-59   20.29
hpc-60   19.84
hpc-61   20.04
hpc-62   19.36
hpc-63   19.57
hpc-64   20.43
hpc-65   37.21
hpc-66   36.78
hpc-67   35.84
      ...
```

Fig. 4 A Slice of ms.conf File

4.2. Tested algorithms

4.2.1. Gaussian Elimination (GE) 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 tested parallel Gaussian Elimination algorithm is described as following.

```
(1) Process 0 distributes the data of matrix $A$ and vector $b$ proportionally to other nodes according to their marked speeds by using row-based heterogeneous cyclic distribution[5]
(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 conducts Gaussian elimination on this row
  (2.2) Synchronize all processes due to data dependence
(3) Process 0 collects temporary results from other processes and conducts the back substitution stage
```

The total workload of this parallel algorithm (sum of the workload in each node) is $W(N) = \frac{2}{3}N^3 - \frac{1}{2}N^2 - 3\frac{1}{6}N + 3$. The algorithm implementation follows HoHe strategy [5], which generates the same number of processes as the number of processors and distributes each process on a separate processor. The system marked speed configuration file generated from STAS system characterization component is used for guiding balanced data distribution. The configuration file is provided to tested program through command line parameters. Once the program is started, process 0 will open this file and read in the marked speed of each node. When process 0 distributes data, it will partition matrices proportionally to the marked speed and send to each process.

4.2.2. Matrix Multiplication (MM) 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, such as the Cannon's algorithm and the outer product algorithm used in ScaLAPACK. But these algorithms are based on homogeneous environment. The tested Matrix Multiplication algorithm in our experiments is a row-based heuristic algorithm for heterogeneous environment [9]. This algorithm adopts the HoHe strategy too. In our algorithm, first, process 0 distributes matrix A by using a row-based heterogeneous block distribution, which means A is distributed 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. The total workload of this parallel algorithm is $W(N) = 2 \times N^3$. The algorithm implementation follows the same technique as in Gaussian Elimination algorithm.

4.2.3. 2-D Convolution. 2-D convolution conducts two-dimensional convolution on two $N \times N$ images, where each element is a complex number. Fast Fourier Transform (FFT) is the kernel in the algorithm. 2-D convolution is implemented by first taking 2-D FFT of each input image, then performing point-wise multiplication of the intermediate results from 2-D FFT, followed by an inverse 2-D FFT. 2D-FFT can be obtained by first performing N times of N-point 1D-FFT along rows followed by N times of N-point 1D-FFT along columns of the intermediate result of the row FFT. The procedure of 2-D convolution can be described as following:

```
A = 2D-FFT(image1)
B = 2D-FFT(image2)
C = MM_Point(A,B)
D = Inverse-2DFFT(C)
```

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

The tested parallel 2-D convolution algorithm is described as following. The total workload is: $W(N) = 66N^2 \lg N + 21N^2 + 84N \lg N$.

(1) Process 0 read 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 proportionally to other nodes according to their marked speeds by using 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 final results from all other processes and output the final result

4.3. Scalability testing and analysis

4.3.1. Machine set. Table 1 shows the constructed machine set from testing set generator. All nodes in Sunwulf cluster are classified into three groups, Sunwulf node, hpc-1 to hpc-64, and hpc-65 to hpc-84. Initially, sunwulf node and one SunFire compute node are selected as Set1. These two nodes are kept in constructing next set. One SunBlade compute node and one additional SunFire compute node are added to Set1 to form Set2. Due to the SunFire node group has only 20 available nodes, the machine set construction stops at Set5.

Table 1 Machine Set Constructed From STAS

Machine Set	# of Nodes	Server Node	SunBlade Node	SunFire Node	Marked Speed
Set1	2	1	0	1	57.72
Set2	4	1	1	2	113.57
Set3	8	1	3	4	226.64
Set4	16	1	7	8	455.71
Set5	32	1	15	16	911.45

4.3.2. Scalability results. We have performed scalability testing and analysis by using STAS system for GE-Sunwulf, MM-Sunwulf and 2DConv-Sunwulf combination. The target speed-efficiency is set as 0.2. The runtime measurement module tunes parameters with 2% variation in each testing and stops when the achieved speed-efficiency reaches the target. All measurement metadata are stored in *scal* database. If we plot the metadata with plotting tools, the speed-efficiency diagram of each combination is obtained. The MM-Sunwulf and 2DConv-Sunwulf results are showing in Fig. 5 and Fig. 6.

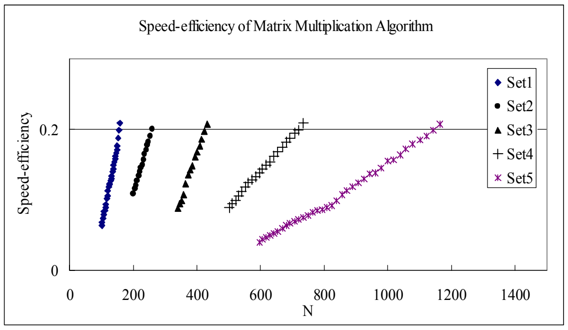

Fig. 5 Speed-efficiency of MM-Sunwulf Combination

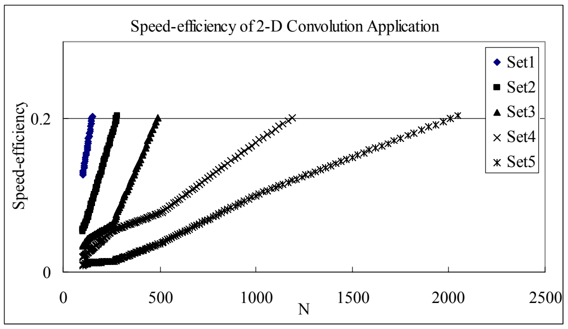

Fig. 6 Speed-efficiency of 2DConv-Sunwulf Combination

The scalability analyzer takes the measured metadata as input, and analyzes the scalability of each combination. Table 2 is the summarized output of three experiments.

Table 2 Scalability Analysis Results

	Ψ*(Set1, Set2)*	Ψ*(Set2, Set3)*	Ψ*(Set3, Set4)*	Ψ*(Set4, Set5)*
GE-Sunwulf	0.35	0.22	0.27	0.32
MM-Sunwulf	0.51	0.42	0.39	0.43
2DConv-Sunwulf	0.54	0.45	0.42	0.61

As we can see from the scalability analysis results from STAS, the GE-Sunwulf is less scalable than MM-Sunwulf, and MM-Sunwulf is less scalable than 2DConv-Sunwulf. The GE algorithm has a sequential portion and has to globally synchronize in each iteration due to data dependency constraints. It involves more communication overhead than Matrix Multiplication algorithm. Intuitively, it is less scalable than MM algorithm. We have used the proposed STAS system to perform the isospeed-e scalability testing of these two algorithms on Sunwulf cluster. The STAS provides a quantitative analysis and confirms the observation. Comparing the Matrix Multiplication and 2D-Convolution algorithm, they have the same amount of communication overhead when the matrix rank is the same, but 2D-Convolution performs much more computation work. When the system size is scaled, the 2D-Convolution should need less problem size increment to keep the achieved speed-efficiency fixed, therefore is more scalable than Matrix Multiplication. This fact is verified through the STAS system as well.

The STAS has successfully performed the testing and analysis of GE-Sunwulf, MM-Sunwulf and 2DConv-Sunwulf combinations. It is an effective and realistic system for evaluating the performance of algorithms and systems. The analysis results provided by STAS can guide users in designing scalable algorithms or systems.

5. Related work

Scalability of parallel and distributed system has been a research issue for many years, but most of the work is targeted on homogeneous environment. isospeed[11] and isoefficiency[2] are two representative scalability metrics. Isoefficiency keeps parallel efficiency constant, where the efficiency is defined as speedup over the number of processors. Here speedup, in turn, is defined as the ratio of sequential execution and parallel execution time. Isoefficiency involves measuring the sequential execution time of algorithms. The requirement of sequential execution time does not appear to be a problem in theoretical analysis, but running a large application on a single node of a parallel system might be problematic in practice. Scalability is the ability to maintain performance when parallel systems scale up. It does not need to and should not refer the sequential execution time. Isospeed and its recent extension, the isospeed-e metric [9], compare the performance of the original and the scaled parallel system directly, and do not require the measurement of sequential processing time.

There are some recent works in scalability analysis. Jogalekar and Woodside proposed a strategy-based scalability metric for general distributed systems[4]. Their scalability metric measures the worthiness of renting a service. Pastor and Bosque proposed a heterogeneous efficiency function to define the heterogeneous scalability[7]. Their work tries to extend the homogeneous isoefficiency scalability model to heterogeneous computing and, therefore, inherits the limitation of parallel speedup, requiring the measurement of solving large-scale problem on single node as we analyzed for homogeneous isoefficiency scalability.

Performance modeling and analysis tool is also a well studied topic. There are numerous well-known performance evaluation toolkits, such as Paradyn, TAU, SCALEA and etc. Paradyn[6] is a performance measurement tool for parallel and distributed programs. It provides precise performance data and automatically searches performance bottlenecks. It gathers and presents performance information in terms of high-level parallel languages and supports measuring programs on massively parallel computers, workstation clusters and heterogeneous systems. TAU[8] is a portable profiling and tracking toolkit for performance analysis of parallel programs. It is capable of gathering performance information through instrumentation of functions, methods, and statements. It also provides the functionality to present performance analysis results with visualization displays. SCALEA[12] is a performance instrumentation, measurement, analysis and visualization tool for parallel programs. It targets to analyze performance of MPI, HPF, OpenMP or hybrid programs and computes a variety of performance metrics. These systems have different strengths and focus on different applications. However, none of them provides the quantitative scalability analysis for algorithms and systems. Our study presents a practical system to conduct inherent scalability analysis of algorithm-system combination for guiding users in designing algorithms and machines.

6. Conclusions and future work

In this study, we have presented a Scalability Testing and Analysis System, STAS. STAS is designed and implemented to assist users conducting scalability analysis of algorithms and systems. STAS adopts isospeed-e as its metric and provides a realistic toolkit for automatic performance evaluation. STAS analyzes the inherent parallelism of algorithms and the scaling property of underlying systems, as well as provides guidance to design scalable algorithms and machines for users.

In the near future, we plan to integrate compiler hints for workload analysis into STAS system. Performance prediction with the support of scalability analysis is another future work. We have demonstrated the methodology of scalability prediction in [9], but more studies need to be done to integrate performance prediction into STAS system. We are also planning to visualize the analysis results in order to better assist users in analyzing the performance of algorithms and systems.

7. Acknowledgments

This research was supported in part by national science foundation under NSF grant CNS-0509118, CNS-0406328, EIA-0224377, and EIA-0130673.

8. References

[1] D. Culler, J. Singh and A. Gupta, *Parallel Computer Architecture: A Hardware/Software Approach*, Morgan Kaufmann Publishers, 1999.

[2] A. Gupta and V. Kumar, “Scalability of Parallel Algorithms for Matrix Multiplication”, *Proceedings of the 1993 International Conference on Parallel Processing*, Vol. 3, pp.115–123, 1993.

[3] K. Hwang and Z. Xu, *Scalable Parallel Computing*, McGraw–Hill, 1998.

[4] P.P. Jogalekar and C.M. Woodside, “Evaluating the Scalability of Distributed Systems”, *IEEE Transaction on Parallel and Distributed Systems*, Vol. 11, No. 6, pp.589-603, 2000.

[5] A. Kalinov and A. Lastovetsky, “Heterogeneous Distribution of Computations While Solving Linear Algebra Problems on Networks of Heterogeneous Computers”, *Journal of Parallel and Distributed Computing*, Vol. 61, No. 4, pp.520-535, 2001.

[6] B.P. Miller, M.D. Callaghan, J.M. Cargille, J.K. Hollingsworth, R.B. Irvin, K.L. Karavanic, K. Kunchithapadam and T. Newhall, “The Paradyn Parallel Performance Measurement Tool”, *IEEE Computer*, Vol. 28, No. 11, pp.37-46, 1995.

[7] L. Pastor and J.L. Bosque, “An Efficiency and Scalability Model for Heterogeneous Clusters”. *IEEE International Conference on Cluster Computing*, pp.427-434, 2001.

[8] S. Shende and A. D. Malony. “The TAU Parallel Performance System”, Submitted to *International Journal of High Performance Computing Applications*, ACTS Collection Special Issue, 2005.

[9] X.H. Sun, Y. Chen and M. Wu, “Scalability of Heterogeneous Computing”, *Proceedings of 34th International Conference on Parallel Processing*, pp.557-564, 2005.

[10] X.H. Sun, T. Fahringer and M. Pantano, “SCALA: A Performance System for Scalable Computing”, *International Journal of High Performance Computing Applications (IJHPCA)*, Vol. 16, No. 4, 2002.

[11] X.H. Sun and D. Rover, “Scalability of Parallel Algorithm–Machine Combinations”, *IEEE Transaction on Parallel Distributed Systems*, Vol. 5, pp.599–613, 1994.

[12] H.L. Truong and T. Fahringer, “SCALEA: A Performance Analysis Tool for Distributed and Parallel Programs”, *Proceedings of 8th International Euro-Par Conference*, 2002.