Communication Overhead: Prediction and its Influence on Scalability

Michael Noelle, Mario Pantano

Institute for Software Technology and Parallel Systems, University of Vienna
Liechtensteinstr. 22 1090 Vienna, Austria
{noelle,pantano}@par.univie.ac.at

Xian-He Sun*

Department of Computer Science
Louisiana State University
Baton Rouge, LA 70803-4020
sun@bit.csc.lsu.edu

Abstract. ***In a parallel or distributed system, relative performance of different programs varies with system and problem size. A determine factor of this performance variation is communication delay. In this study, the influence of communication on scalability is carefully examined. A model is first introduced to estimate communication overhead. The model then is used in a newly proposal iterative method to predict the scaling property of a program automatically. Experimental testing is performed on a Cray T3E and a QSW machine to confirm the methodology. Experimental and analytical results show that the combination of estimation and prediction is feasible and effective. It has real potential in performance prediction and program optimization.***

Keywords: Performance prediction, Communication cost, Scalable computing

1 Introduction

A major difficulty in parallel programming is how to compare performance over a range of system and problem size. Performance efficience varys when system and problem size scale up, and this variation differs with programs, machines, and algorithm-machine combinations. Several scalability metrics have been proposed to measure the performance variations [1, 2]. While scalability analysis, analogous of complexity analysis, has been used in algorithm design, how to apply scalability in actual program development eludes researchers still. To achieve the goal of automatic scalable software development, during the last few years, we have proposed the *isospeed scalability* [1], studied the relation between execution time and scalability [6], and introduced the crossing-point analysis mechanism so that programs can be compared over a range of system and problem size [4]. More recently, the integration of the crossing-point analysis into a data-parallel compilation system has been proposed in [5]. In this study, combined with an estimation model, we present a methodology for general parallel programming, where MPI application programs are studied.

Scalability is a function of parallel degradation. In many situations, the source of the parallel degradation is mainly due to the time spent in data exchanges between processors. A good estimation of communication time requires a deep insight in the algorithm as well as in the underlying parallel architecture. Therefore, models that incorporate specific knowledge of the algorithm and general assumptions of the hardware are needed for predicting the communication overhead. In this paper we investigate the relation of scalability and communication time and how the commu-

* This author was supported in part by National Science Foundation under NSF grant ASC-9720215 and by LSU 1998 COR award.

nication modeling can be efficiently used for automatic scalability prediction and automatic range comparison for MPI programs.

2 Scalability and Range Comparison

A major driving force behind parallel computing is to solve large problems fast. Speed, defined as work divided by time, has been proposed as an alternative primary metric for scalable computing. Average speed is the achieved speed per processor. For many algorithm-machine combinations, the average speed can be maintained by increasing the problem size. The necessary problem size increase varies with algorithm-machine 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 of the algorithm when $p' > p$ 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:

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

Where the work W' is determined by the isospeed constraint. Finally, let $T_p(W)$ be the time for computing W work on a p processors system, equation (2) shows how scaled execution time can be computed from scalability,

$$T_{p'}(W') = \psi^{-1}(p, p') \cdot T_p(W). \tag{2}$$

The parallel execution time $T_p(W)$ can be divided into two parts: the ideal parallel processing time and parallel processing overhead, $T_o(W)$.

$$T_p(W) = \frac{T_s(W)}{p} + T_o(W) = \frac{W \cdot \Delta}{p} + T_o(W), \tag{3}$$

where T_s is the sequential execution time, Δ is the computing capacity, defined as time per unit of work, of a single processor. The parallel processing overhead T_o contains the load imbalance overhead, communication overhead, and other possible parallelism degradations. By the definition of scalability (see (1)), scalability can be predicted if and only if the scaled work size, W', can be predicted. A prediction formula has been given in [6] to compute W':

$$W' = \frac{a \cdot p' \cdot T_o^{'}(W')}{1 - a\Delta} \tag{4}$$

where a is the achieved average speed and $T_o'(W')$ is the parallel processing overhead on p' processors. When parallel degradation does exist (i.e. $T_o^{'}(W') > 0$), $a \cdot \Delta < 1$ and, therefore, equation (4) is traceable. $T_o^{'} > 0$ is a necessary and sufficient condition of equation 4. When $T_o^{'} = 0$, ideal scalability is achieved with $\psi(p, p^{'}) = 1$.

Equation 2 shows that if a program is faster at the initial state and has a better scalability then it will remain faster over the scalable range. Range comparison becomes more difficult when the initial faster program has a smaller scalability. When the system size scales up, an originally faster code with lower scalability can become slower than another code with a better scalability. Finding the fast/slow crossing point is critical for optimizing performance. Finding the superiority/inferiority crossing point, however, is very difficult [4].

Let program 1 have execution time t, scalability $\Phi(p, p')$, and scaled problem size W'. Let program 2 have execution time T, scalability $\Psi(p, p')$, and scaled problem size W^*. By Definition **??**, p' is the crossing point of program 1 and 2 if and only if

$$\frac{\Phi(p, p')}{\Psi(p, p')} > \alpha. \tag{5}$$

In fact, by equation (2), when $\Phi(p, p') \geq \alpha\Psi(p, p')$ we have $t_{p'}(W') \leq T_{p'}(W^*)$. Notice that since $\alpha > 1$ program 2 has a smaller execution time at the initial state, $t_p(W) > T_p(W)$. This superiority/inferiority changing in execution time gives the meaning of performance crossing point. Figure 1 gives the procedure of range comparison in terms of scalability. Range comparison is given in terms of scalability. Scalability yet need to be predicted

Range Comparison
Begin
Find the Scalability of program 1 $\Phi(p, p')$;
Find the Scalability of program 2 $\Psi(p, p')$;
If $\Phi(p, p') \leq \alpha\Psi(p, p')$ **then**
program 2 is superior over range $< p, p' >$;
Else
p' is a scaled crossing point
End{If}
End{Range Comparison }

Figure 1: Range Comparison Via Performance Crossing point

for performance comparison. Parallel processing overhead $T_o^{'}(W')$ in general is a function of problem size. With unknowns on both sides of the equation, using formula (4) for scalability prediction is not a straightforward task. The iterative method listed in Figure 2 is proposed to compute W' and to predict the scalability automatically [5]. As first assumption, we assume that the application is scalable and its work W is a monotonically increasing function of a scaling parameter n.

The iterative algorithm consists of three parts: the main program and two subroutines for computing the function of $\phi(W)$ and the inverse of $\phi(W)$. Function $\phi(W)$ is implied by equation (4). Mathematically, the iterative algorithm is to find a fixed point of $\phi(W)$ such that $W = \phi(W)$. It is general, efficient, and correct. It is also stable, in the sense that if the input value are close to the actual data, the predicted scalability will be close to the actual scalability. The accuracy and correctness of the algorithm, however, are still based on the accuracy and correctness of its input. An appropriate estimation model of parallel overhead is needed for providing appropriate inputs.

Iterative Method
Begin
Initial Value: $W_0 = \frac{p' \cdot W}{p}$;
Compute $\phi(W_0)$;
If $\phi(W_0) := W_0$ do
$W' = W_0$;
Elseif $\phi(W_0) > W_0$ do
Begin Iteration (k=0; k++)
$W_{k+1} = \phi(W_k)$;
until $\|W_{k+1} - W_k\| < \epsilon$;
$W' = W_{k+1}$;
Else do
Begin Iteration (k=0; k++)
$W_{k+1} = \phi^{-1}(W_k)$
until $\|W_{k+1} - W_k\| < \epsilon$
$W' = W_{k+1}$;
End{If}
End{Iterative Method}

Subroutine $\phi(W)$
Solve $f(n) = W$ for n;
Compute $T_o = g(n)$;
Compute $\phi(W) = \frac{a \cdot p \cdot T_o}{1 - a \cdot \triangle}$;
Subroutine $\phi^{-1}(W)$
Compute $T_o = \frac{1 - a \cdot \triangle}{a \cdot p} W$;
Solve $g(n) = T_o$ for n;
Compute $\phi^{-1}(W) = f(n)$;

Figure 2: The Iterative Method for Predicting Scalability

3 Communication Prediction

Following Section 2 communication overhead has to be estimated in order to predict the scalability. A standard model of the communication behavior in distributed memory machines is to compute the communication time for each message as the sum of the startup and the length dependent transmission time, i.e. $T_{comm} = \rho + \beta \cdot D$. D gives the length of a single message, ρ, β are machine dependent values describing the startup time and the time required for the transmission of a single byte. These values are either supplied by the manufacturer of the machine or determined through a separate measurement. This model is applicable only if there is no interference between messages. In contrast, our assumption is that any interference that may occur from the communication patterns used in the algorithm is statistically reproducible. Based on this assumption we do not consider ρ, β as constants. We parameterized the communication model in terms of characteristics of the algorithm, i.e. communication patterns used and the length of the messages. The reason for the latter distinction is that the hardware in cooperation with the message passing software handles short and long messages differently in many cases. Therefore, according to the size of messages and the communication patterns required different values for ρ, β are estimated. A more detailed discussion on the derivation of the communication model for several communication pattern-machines combinations can be found in [**?**].

4 Implementation

We have implemented a prototype version of the iterative algorithm depicted in Figure 2 in the Scala performance analysis System [**?**]. Scala is a post-execution performance analysis tool being developed as a support to the Vienna HPF Compiler (VFC) []. It provides features for the automatic code instrumentation and post-execution performance analysis for F90, C and HPF applications. The code instrumentation is performed by the Scala Instrumentation System (SIS) [7] embedded within the VFC inserting calls to the runtime measurements libraries. At the end of execution tracefiles containing measurements for each code region instrumented and for the various communication activities is produced. These tracefiles are then analyzed by the post-execution performance analysis system and several indices and statistics for each code region are visualized. For the estimation of T_0' in Figure 2, we derived the communication models using Scala and SIS for the measurements.

The following communication patterns are frequently used in the area of digital image processing algorithms [8]: • *1-d torus* • *2-d torus* • *Hypercube* • *De Bruijn*. Using a statistical approach we fitted the inconsistent measurement by means of a least squares approximation.

In the case of the Jacobi algorithm the bidirectional 1-d and 2-d torus is used. The resulting parameters may be found in Table 1. At the current state, the number of communications performed and the number of bytes transferred as well as the specification of the complexity of the algorithm, in terms of floating point operations as function of the problem size, is still left to the user. These limitations will be removed in the release version of the tool.

5 Experiments

In this section we present experiment results to validate the methodology proposed. We chosed a Jacobi relaxition algorithm coded in Fortran 90 and MPI with two different communication patterns, bidirectional 1-d and 2-d torus on two machines, namely, Cray T3E and QSW CS-2[1].

5.1 Cray T3E

The Cray T3E series is a second generation of scalable parallel processing systems from SGI/Cray Research. Its interprocessor communication is a 3D torus which helps to minimize the hops between nodes and provides scalability up to the maximum configuration of 2048 nodes. The Cray T3E/512 used in this validation is installed at the High-Performance Computing Center (HLRS) in Stutgart, Germany with each node is a DEC Alpha 21164 having a clock speed of 300 MHz and 128MByte DRAM.

5.2 QSW CS-2

The Quadrics Supercomputers World (QSW) CS-2 installed at the Vienna Center for Parallel Computing (VCPC) is a 128 processor general purpose, distributed-memory parallel machine. The nodes are interconnected through a multistage switch. Each node is a SuperSPARC

[1] was known as Meiko CS-2

	Cray T3E				QWS CS-2			
	1-d		2-d		1-d		2-d	
	short	long	short	long	short	long	short	long
ρ	35.60	71.25	40.38	77.55	220.36	297.28	150.49	-
β	$2.00 \cdot 10^{-2}$	$12.52 \cdot 10^{-3}$	$2.15 \cdot 10^{-2}$	$14.18 \cdot 10^{-3}$	$5.57 \cdot 10^{-2}$	$3.55 \cdot 10^{-2}$	$6.74 \cdot 10^{-2}$	

Table 1: Communication models (in μs)

running at 50 MHz, with 64 MBytes of memory and capable of 100 MFlops. The architecture allows systems with up to 4096 nodes to be manufactured, with any mixture of scalar and vector nodes. It is modular in design allowing the addition of further modules to a system without any of the existing hardware becoming redundant.

5.3 Prediction results

The Jacobi algorithm has been executed on 4 processors on both Cray T3E and QWS CS-2 and using the iterative method in Figure 2 implemented in Scala we predicted the scalability presented in Tables 2 to 5. The scalability values obtained on the QSW CS-2 are shown in Tables 2 and 3 for the bidirectional 1-d torus and bidirectional 2-d torus, respectively.

The predicted values obtained on Cray T3E are shown in Tables 4 and 5. The difference between the predicted and measured values is given in percentage in the third column of the tables.

The measured values are computed as the ratio between the measured execution times on p' and p processors. As we can see from the tables, the predicted values are quite close to the measured ones verifying the accuracy of the prediction. This analysis gives a first insight on the behavior of the algorithm when the number of processors and the problem sizes are increased. On both QWS and Cray, Jacobi achieves high scalability when a bidirectional 2-d torus communication pattern is used.

As already pointed out, the prediction of the scalability and execution time is strongly dependent on the estimation of the parallel overhead T_o'. Since the Jacobi algorithm in general implies good load balance, in our experiments we consider T_o as communication overhead which is obtained by using the standard communication model

$$T_0 = Z \cdot (\rho + (\beta \cdot D))$$

where Z is the maximum number of communications per processor. The startup time ρ and the transfer time per message byte, β, have been obtained using the methodology described in Section 3 and the values are given in Table 1. D is the length (in Bytes) of a single message. Tables 6 and 7 show the values obtained on the QSW CS-2 and Tables 8 and 9 contain the values collected on Cray T3E for bidirectional 1-d torus and bidirectional 2-d torus, respectively. The tables present the predicted versus the measured execution times together with the scaled work W' and the problem size N obtained from the complexity of the algorithm.

The estimation of ρ and β, in some case, is difficult due to the hardware characteristics of the machines. The inaccuracy in estimating the communication implies inaccuracy in the execution time and in the scalability prediction. This effect can be seen on Cray T3E for Jacobi with bidirectional 1-d torus (Table 8). In this case, each processor transmit messages greater than 4096 Bytes and the estimation of the communication for long messages (> 4096) becomes inaccurate. The model determined by the least squares fit has a high standard deviation and likewise the parameters describe the communication behavior with lesser accuracy. The reasons for this inaccuracy require further investigations. In the same way, is effected the scalability in Table 4.

Table 2: Bidirectional 1-d torus on QSW CS-2: Predicted (P) and measured (M) scalability

	Jacobi 1-d										
$\psi(p,p')$	$p'=4,\ n=1024$		$p'=8,\ n=1750$			$p'=16,\ n=3157$			$p'=32,\ n=5912$		
	P	M	P	M	%	P	M	%	P	M	%
p=4	1.000	1.000	0.684	0.688	0.5	0.420	0.421	0.2	0.239	0.238	0.4
p=8			1.000	1.000	-	0.614	0.611	0.4	0.350	0.346	1.1
p=16						1.000	1.000	-	0.569	0.566	0.5
p=32									1.000	1.000	-

Table 3: Bidirectional 2-d torus on QSW CS-2: Predicted (P) and measured (M) scalability

	Jacobi 2-d										
$\psi(p,p')$	$p'=4,\ n=1024$		$p'=8,\ n=1619$			$p'=16,\ n=2222$			$p'=32,\ n=3210$		
	P	M	P	M	%	P	M	%	P	M	%
p=4	1.000	1.000	0.799	0.801	0.2	0.849	0.851	0.2	0.818	0.815	0.3
p=8			1.000	1.000	-	1.062	1.061	0.9	1.023	1.018	0.4
p=16						1.000	1.000	-	0.964	0.958	0.6
p=32									1.000	1.000	-

From the predicted execution times obtained on Cray T3E, we note that the execution time (T_{1-d}) for Jacobi with bidirectional 1-d torus pattern (Table 8) is smaller at the initial state $p=4$ than its counterpart (T_{2-d}) with bidirectional 2-d torus implementation (Table 9). In particular, $T_{2-d} = \alpha T_{1-d}$ where $\alpha = 1.030$.
The ratio between the scalability values obtained from $p = 4$ to $p' = 8$ (see Tables 4 and 5), $\frac{0.908}{0.723} > \alpha = 1.030$. By Equation 5 we can conclude that between $p = 4$ and $p' = 8$ there is a crossing point. This result is verified in Figure 3.

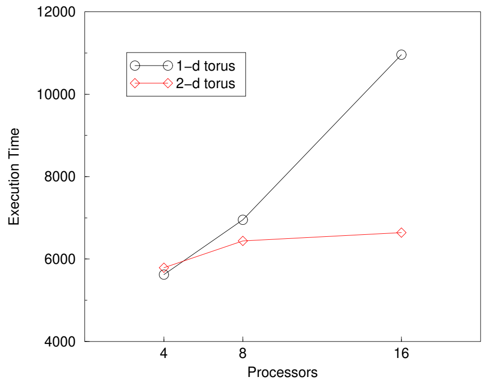

Figure 3: Crossing Point on Cray T3E

6 Conclusions

References

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

Table 4: Bidirectional 1-d torus on CRAY T3E: Predicted (P) and measured (M) scalability

	Jacobi 1-d											
$\psi(p,p')$	$p'=8$, $n=1702$			$p'=16$, $n=2976$			$p'=32$, $n=5434$			$p'=64$, $n=10275$		
	P	M	%	P	M	%	P	M	%	P	M	%
p=4	0.723	0.808	11.7	0.473	0.512	8.2	0.284	0.313	10.2	0.158	0.170	7.5
p=8	1.000	1.000	-	0.654	0.634	3.0	0.392	0.387	1.2	0.219	0.210	4.1
p=16	-	-	-	1.000	1.000	-	0.600	0.610	1.6	0.335	0.332	0.8
p=32	-	-	-	-	-	-	1.000	1.000	-	0.559	0.543	2.8
p=64	-	-	-	-	-	-	-	-	-	1.000	1.000	-

Table 5: Bidirectional 2-d torus on CRAY T3E: Predicted (P) and measured (M) scalability

	Jacobi 2-d											
$\psi(p,p')$	$p'=8$, $n=1519$			$p'=16$, $n=2186$			$p'=32$, $n=3124$			$p'=64$, $n=4372$		
	P	M	%	P	M	%	P	M	%	P	M	%
p=4	0.908	0.899	1.0	0.877	0.871	0.6	0.859	0.856	0.3	0.877	0.871	0.6
p=8	1.000	1.000	-	0.966	0.969	0.3	0.946	0.952	0.6	0.966	0.969	0.3
p=16	-	-	-	1.000	1.000	-	0.979	0.982	0.3	1.000	1.000	-
p=32	-	-	-	-	-	-	1.000	1.000	-	1.021	1.018	0.2
p=64	-	-	-	-	-	-	-	-	-	1.000	1.000	-

[2] S. Sahni and V. Thanvantri, "Performance metrics: Keeping the focus on runtime," *IEEE Parallel & Distributed Technology*, pp. 43–56, Spring 1996.

[3] X.-H. Sun, "The relation of scalability and execution time," in *Proc. of the International Parallel Processing Symposium'96*, April 1996.

[4] X.-H. Sun, "Performance range comparison via crossing point analysis," in *Lecture Notes in Computer Science, No 1388* (J. Rolim, ed.), Springer, march 1998.

[5] X.-H. Sun, M. Pantano, and T. Fahringer, "Performance range comparison for restructuring compilation," in *Proc. of the International Conference on Parallel Processing*, August 1998.

[6] X.-H. Sun and J. Zhu, "Performance prediction: A case study using a scalable shared-virtual-memory machine," *IEEE Parallel & Distributed Technology*, pp. 36–49, Winter 1996.

[7] M. Calzarossa, L. Massari, A. Merlo, M. Pantano, and D. Tessera, "Integration of a compilation system and a performance tool: the HPF+ approach ," in *Proceedings HPCN Europe 1998*, Lecture Notes in Computer Science 1401, pp. 809–815, Springer, April 1998.

[8] T. Andreae, M. Noelle, and G. Schreiber, "Embedding cartesian products of graphs into de Bruijn graphs.," *J. of Parallel and Distributed Computing*, vol. 46, pp. 194–200, November 1997.

Table 6: Bidirectional 1-d torus on QSW CS-2: execution times (in μs)

	Jacobi 1-d								
p'	N	W'	T_o'			$T_{p'}(W')$		
			P	M	%	P	M	%
4	1024	2097152	-	1370	-	-	202510	-
8	1750	3064682	2002	1800	10	296067	294080	0.7
16	3157	4986422	3256	4160	27	482166	480990	0.2
32	5912	8740604	5712	6480	13	847322	848620	0.1

Table 7: Bidirectional 2-d torus on QSW CS-2: execution times (in μs)

	Jacobi 2-d								
p'	N	W'	T_o'			$T_{p'}(W')$		
			P	M	%	P	M	%
4	1024	2097152	-	1530	-	-	202340	-
8	1619	2622967	1913	1740	9	253241	252450	0.3
16	2222	2469847	1802	2100	16	238327	237720	0.2
32	3210	2563045	1869	1840	1.5	247359	247980	0.2

Table 8: Bidirectional 1-d torus on CRAY T3E: execution times (in μs)

	Jacobi 1-d								
p'	N	W'	T_o'			$T_{p'}(W')$		
			P	M	%	P	M	%
4	1024	2097152	-	350	-	-	5620	-
8	1702	2899273	484	400	17.3	7773	6950	10.5
16	2976	4429752	739	1000	35	11881	10960	7.7
32	5434	7382118	1232	1500	21	19788	17940	9.3
64	10275	13196979	2202	3600	63	35569	32980	7.2

Table 9: Bidirectional 2-d torus on CRAY T3E: execution times (in μs)

	Jacobi 2-d								
p'	N	W'	T_o'			$T_{p'}(W')$		
			P	M	%	P	M	%
4	1024	2097152	-	490	-	-	5790	-
8	1519	2309987	540	670	24	6376	6440	1
16	2186	2390245	558	660	18	6602	6640	0.5
32	3124	2441112	570	520	8.7	6740	6760	0.2
64	4372	2390059	558	510	8.6	6602	6490	1.7