Performance Range Comparison for Restructuring Compilation

Xian-He Sun*

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

Mario Pantano, Thomas Fahringer

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

Abstract

A major difficulty in restructuring compilation is how to compare parallel performance over a range of system and problem sizes. This study introduces the concept of range comparison for data-parallel programming. Unlike conventional execution time comparison in which performance is compared for a particular system and problem size, range comparison compares the performance of programs over a range of ensemble and problem sizes via scalability and performance crossing point analysis. An algorithm is developed to predict the crossing point automatically. The correctness of the algorithm is proved and a methodology is developed to integrate range comparison into restructuring compilations. A preliminary prototype of the methodology is implemented and tested under Vienna Fortran Compilation System. Experimental results demonstrate that range comparison is feasible and effective.

1 Introduction

Traditionally, Highly parallel scalable MultiProcessing Systems (HMPs) have been programmed using message passing where the user is responsible for explicitly inserting communication statements into a sequential program. The development of parallel languages such as Vienna Fortran, Fortran D and High Performance Fortran (HPF) [1, 2] improved the situation by providing high-level features for the specification of data distributions. However, current technology of code restructuring systems inherently lacks the power to fully exploit the performance offered by HMPs. The primary motivation of parallel processing is high performance. Effectiveness and efficiency of restructuring compilation are the current barriers for the success of a simple, high-level programming model approach.

Restructuring a program can be seen as an iterative process in which a parallel program is transformed at each iteration. The performance of the current parallel program is analyzed and predicted at each iteration. Then, based on the performance result, the next restructuring transformation is selected for improving the performance of the current parallel program. This iterative process terminates when certain predefined performance criteria are met or as a result of explicit user intervention. Integrating performance analysis with a restructuring system is critical to support automatic performance tuning in the iterative restructuring process. The development of a fully compiler integrated performance system for scalable parallel machines is especially challenging. In a scalable environment, the performance of a program may vary with data distribution, number of processors (system ensemble size), and problem size. Predicting the performance variations and integrating the performance indices automatically into a restructuring compiler are two major challenges facing researchers in the field [3].

A key question of restructuring is how to predict and compare the scaled relative performances of a small number of data distributions and transformations automatically, so that appropriate optimization decisions can be made. In order to compare relative performance over a range of problem and ensemble sizes, scalability prediction is proposed as a solution in this study. A restructured code with a smaller initial execution time and a larger scalability will be a clear

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

winner over the scalable range. Otherwise, the superior range of the implementation will end at the first fast/slow crossing point. We introduce a technique called range comparison, which is concerned with locating the crossing point. An iterative algorithm is first derived to predict the scalability and crossing point on a given parallel platform. Then, the connection between the iterative algorithm and an existing static performance estimator, P^3T, [4] is discussed. A preliminary prototype of automatic range comparison is implemented under the Vienna Fortran Compilation System (VFCS). Finally, two applications are tested with two different data distributions to verify the correctness and feasibility of the range comparison approach. While current experimental results are primitive, they clearly demonstrate the feasibility and effectiveness of the range comparison approach for program restructuring.

2 VFCS and P^3T

Vienna Fortran Compilation System (VFCS) consists of a parallelizing compiler for Vienna Fortran/HPF and tools for program analysis and transformation. The parallelization techniques of VFCS are based upon the *Single-Program-Multiple-Data (SPMD)* paradigm. Currently two parallelization techniques, the *overlap* strategy and the *inspector-executor* strategy [5] are implemented. The overlap strategy is targeted towards regular computations, like stencil computations, and relies heavily on compile-time analysis and compile-time optimization. The second parallelization strategy, a run-time technique based on the inspector-executor strategy, is targeted towards irregular computations, which are characterized by loops exhibiting irregular access patterns (using indirection arrays) that are dependent on run-time data.

P^3T [4] is a static, interactive performance estimator that assists users in performance tuning of regular programs. P^3T is based on a single profile run to obtain characteristic data for branching probabilities, statement and loop execution counts. The shortcoming of P^3T is the lack of information on the influence of data distribution, especially for scalable computing. Communication overhead is an important metric in choosing an appropriate data distribution. It consists of two parts: number of data transfers and amount of data transferred. For the sake of brevity, only issues of static estimation of communication overhead are discussed in this section. Interested readers may refer to [4] for more information regarding the other performance parameters of P^3T.

2.1 Number of Data Transfers

The number of data transfers is a critical parameter which reflects the high message startup costs on most distributed memory architectures. Commonly the overhead for communication is decreasing if it can be hoisted outside of a nested loop. Moreover, communication inside of a specific loop body in many cases implies that the loop is sequentialized due to synchronization between the processors involved in the communication. P^3T carefully models the loop nesting level at which a communication is placed, array access patterns, data dependences and distribution, control flow, and compiler communication optimizations (e.g., communication vectorization and fusion) in order to determine the number of data transfers with high accuracy.

For communication that can be hoisted outside a loop nest we assume the loosely synchronous communication model [6] which implies that all involved processors communicate simultaneously. For such a communication statement the number of data transfers is determined by the maximum number of data transfers across all involved processors. For communication that cannot be hoisted outside a loop nest due to a data dependence we assume that it sequentializes the loop at which the communication is placed as well as all data transfers implied by the communication. The number of data transfers for such a communication is given by the sum of data transfers across all processors involved in the communication.

2.2 Amount of Data Transferred

The current generation of distributed memory architectures reduces the impact of the message length on the communication overhead. For applications that transmit small data volumes, the startup cost is the predominate communication cost factor. However, for increasing data volumes transmitted, the message transfer time per byte and in turn the amount of data transferred becomes the first order performance effect. In order to provide a highly accurate estimate for the amount of data transferred (given in bytes) as induced by a parallel program, P^3T estimates the number of non-local data elements accessed and incorporates machine specific data type sizes. For this purpose, P^3T examines the loop nesting level at which a communication is placed, array access patterns, data dependences

and distributions, control flow, and compiler communication optimizations.

As the compiler specifies the communication pattern at the source code level, the target architecture can be for the most part – except for data type sizes – ignored. Consequently, this parameter ports easily to a large class of distributed memory architectures.

3 Performance Range Comparison

While execution time is an important performance metric for restructuring compilations, its comparison bonds to a specific pair of system and problem size. Execution time alone is not sufficient for performance comparison over a range of system and problem sizes. Scalability has been recognized as an important property of parallel algorithms and machines in recent years. Several scalability metrics have been proposed [7, 8, 9]. However, scalability has been traditionally studied separately as an independent property. Only very recently has the relation of scalability and execution time been studied [10]. Based on these relations, the concepts of crossing point analysis and range comparison are introduced [11]. To fully understand the concept of range comparison, some background for scalability and crossing point analysis needs to be introduced.

3.1 Isospeed Scalability

In this paper, problem size refers to the work to be performed. The following definition was given in [7].

Definition 1 (Isospeed Scalability) *An algorithm-machine combination is scalable if the achieved average speed of the algorithm on the given machine can remain constant with the increasing number of processors, provided the problem size can be increased with the system size.*

For a large class of Algorithm-Machine Combinations (AMCs), 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 parallel execution time could be computed from scalability,

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

Three approaches have been proposed to determine scalabilities [7]. They are: *computing* the relation between problem size and speed, directly *measuring* the scalability, and *predicting* scalability with certain predetermined parameters. Among them, scalability prediction seems to be the most useful for data-parallel compilation systems.

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

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

where T_s is the sequential execution time. 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 [12] to compute W':

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

where a is the average speed, Δ is the computing rate of a single processor, and $T_o(W')$ is the parallel processing overhead on p' processors. 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.

3.2 Performance Crossing Point and Range Comparison

Theorem 1 gives a relation between scalability and execution time of two different algorithm-machine combinations [10].

Theorem 1 *If algorithm-machine combinations 1 and 2 have execution time $\alpha \cdot T$ and T, respectively, at the same initial state (the same initial ensemble and problem size), then combination 1 has a higher scalability than combination 2 at a scaled ensemble size if and only if the execution time of combination 1 is smaller than the α multiple of the execution time of*

combination 2 for solving W' at the scaled ensemble size, where W' is the scaled problem size of combination 1.

Theorem 1 shows that if an AMC is faster at the initial state and has a better scalability than that of others then it will remain faster over the scalable range. Range comparison becomes more difficult when the initially faster AMC has a smaller scalability. When the system ensemble 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 choosing efficient program transformations in a data-parallel environment. Finding the superiority/inferiority crossing point, however, is very difficult and is depending on the view of scalable computing [11]. Definition 2 gives a formal definition of crossing point based on the isospeed scalability.

Definition 2 (scaled crossing point) *For any $\alpha > 1$, if algorithm-machine combinations 1 and 2 have execution time $\alpha\dot{T}$ and T respectively at the same initial state, then we say a scaled ensemble size p' is a crossing point of combinations 1 and 2 if the ratio of the isospeed scalability of combination 1 and combination 2 is greater than α at p'.*

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

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

In fact, as proven by Theorem 2, when $\Phi(p,p') \geq \alpha\Psi(p,p')$ we have $t_{p'}(W') \leq T_{p'}(W^*)$. Notice that since $\alpha > 1$ combination 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. The correctness of Theorems 2 is proved in [11].

Theorem 2 *If algorithm-machine combination 1 has a larger execution time than algorithm-machine combination 2 at the same initial state, then, for any scaled ensemble size p', p' is a scaled crossing point if and only if combination 1 has a smaller scaled execution time than that of combination 2.*

Based on the above theoretical findings Figure 1 gives the range comparison algorithm in terms of scalability.

`Assumption:` Assume algorithm-machine combinations 1 and 2 have execution time αT and T respectively at the same initial state, where $\alpha > 1$.
`Objective:` Find the superior range of combination 2

Range Comparison
Begin
$p' = p$;
Repeat
  $p' = p' + 1$;
  Find the Scalability of comb. 1 $\Phi(p,p')$;
  Find the Scalability of comb. 2 $\Psi(p,p')$;
Until($\Phi(p,p') > \alpha\Psi(p,p')$ **or** p' = the up-limit)
If $\Phi(p,p') > \alpha\Psi(p,p')$ **then**
  p' is the smallest scaled crossing point
  Comb. 2 is superior over range $< p, p'-1 >$;
Else
  Comb. 2 is superior over range $< p, p' >$
End{If}
End{Range Comparison }

Figure 1: Range Comparison Via Performance Crossing point

3.3 Automatic Crossing-Point Prediction

The range comparison algorithm listed in Figure 1 is in terms of scalability. Scalabilities of different code implementations, or different algorithm-machine combinations in general, still need to be determined for range comparison. In this paper we propose an iterative method listed in Figure 2 to compute W' and to predict the scalability automatically. We assume that the underlying application is scalable and its problem size (computation work) is a monotonically increasing function of a scaling parameter n (input data size). We also assume that parallel overhead T_o is either independent of parameter n or is monotonically increasing with n. Function $\phi(W)$ is implied by equation (4). Mathematically, the iterative algorithm is to find a fixed point of $\phi(W)$ such that $W = \phi(W)$. A proof of correctness of the algorithm is provided in [13].

Like most iterative methods, the convergence rate of the algorithm is application dependent. It depends on the properties of function $f(n)$. For most scientific computations, $f(n)$ is a low degree polynomial function and the algorithm converges very fast. Our experimental results show that the algorithm only requires three to five iterations to converge to a solution with an error bound of 10^{-2}.

`Assumption:` Assume work W and overhead T_o are increasing functions of the scaling parameter n, $W = f(n)$ and $T_o = g(n)$, or $T_o = g(n)$ is a constant, and assume the parallel code under study has been executed on the target machine with W work and p processors.
`Objective:` Compute the scalability from machine ensemble size p to machine ensemble size p', where $p' > p$, with an error of $\epsilon > 0$.

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: An Iterative Method for Predicting Scalability

4 Integrated Range Comparison Under VFCS

Scientific applications have been tested under VFCS to confirm the correctness and efficiency of the proposed range comparison. The experiments have been carried out on an iPSC/860 hypercube with 16 processors. The parallel processing overhead T_o used in the scalability iteration algorithm contains communication overhead and load imbalance. Limited by the current functionality of P^3T, we choose two codes, Jacobi and Redblack, which have good load balance. T_o, therefore, contains only the communication time that can be obtained by the formula

$$T_o = Z(\alpha + (\beta \cdot D) + \gamma \cdot h), \tag{6}$$

where Z - the number of data transfers - and D - the amount of data transferred - can be predicted at compile time for any problem size W using P^3T. α and β are the startup time and the transfer time per message byte, respectively. γ represents the additional overhead for each network hop and h is the number of hops.

Jacobi and Redblack, have been parallelized with VFCS and their performance measured on 4 processors. The performance indices obtained, and needed for computing the initial state of the scalability prediction, are W, T_p, T_c, Z, D, T_o, where T_c is the computation time. Based on equation (3), the execution models of Jacobi and Redblack are:

$$T_p = \frac{W}{p} * \Delta + T_o = 11 * \frac{(n-2)^2}{p} \Delta + T_o$$

and

$$T_p = \frac{W}{p} * \Delta + T_o = 6 * \frac{(n-1)^2}{p} \Delta + T_o$$

respectively. Computation is uniformly distributed across all processors. $T_c = \frac{W}{p} * \Delta$. The computing rate $\Delta = \frac{T_c * p}{W}$ and the average speed $a = \frac{W}{p * T_p(W)}$ can be determined by the measured computing time and total execution time. The initial value of the prediction algorithm, $W_0 = \frac{p' * W}{p}$, is computed based on the work, W, performed on 4 processors. Starting with iteration $k = 0$, a new input data size $n_k = f^{-1}(W)$ is computed for $k > 0$. The original source code is then modified and automatically parallelized by using VFCS. After parallelization, P^3T automatically estimates the number of transfers Z and the amount of data transferred D. The communication overhead T_o' and the scaled work W_k' are predicted using (6) and

(4), respectively. Scalability from processors p to processors p' is determined when the terminating condition $||W_k - W_{k-1}|| < \epsilon$ is satisfied for a fixed $\epsilon > 0$ ($\epsilon = 0.01$ is used in our experiments). Otherwise the method iterates with the new parameter n_{k+1}. Figure 3 depicts the path for predicting the scalability by using VFCS and P^3T.

Tables 1 and 2 show the measured and predicted scalability of Jacobi algorithm with two different data distribution strategies: two-dimensional block distribution (Jacobi_2D) and one-dimensional (column) distribution (Jacobi_C) of all program arrays to a two-dimensional and one-dimensional processors array, respectively. The experimental results confirm that

$\psi(p,p')$	$p'=8$			$p'=16$		
	Pred	Meas	diff	Pred	Meas	diff
p=4	0.718	0.738	2.7%	0.605	0.617	1.9 %
p=8	1.000	1.000	0%	0.842	0.819	2.7%
p=16				1.000	1.000	0%

Table 1: Jacobi: 2D distribution, predicted and measured scalability

$\psi(p,p')$	$p'=8$			$p'=16$		
	Pred	Meas	diff	Pred	Meas	diff
p=4	0.721	0.739	2.4%	0.576	0.581	0.8 %
p=8	1.000	1.000	0%	0.796	0.808	1.5%
p=16				1.000	1.000	0%

Table 2: Jacobi: column distribution, predicted and measured scalability

our predicted scalabilities are very accurate and the variations of scaled performance for various data distributions are also captured.

Table 3 shows the predicted and measured scalability values of the Redblack algorithm with 2D distribution. Indeed, scalability can be used to predict execution time by using (2). Table 4 presents the predicted execution times versus the measured ones.

The execution time of Redblack can be written as $\alpha * T_4(64) = \alpha * 1869 = 5560\ \mu sec$ and for Jacobi $T_4(64) = 1869\ \mu sec$. α is 2.975. According to Tables 1, 2, and 3, the scalability of Jacobi algorithm is higher than that of Redblack algorithm. Therefore, by Theorem 1, the smaller initial execution time and larger scalability show that Jacobi scales better than

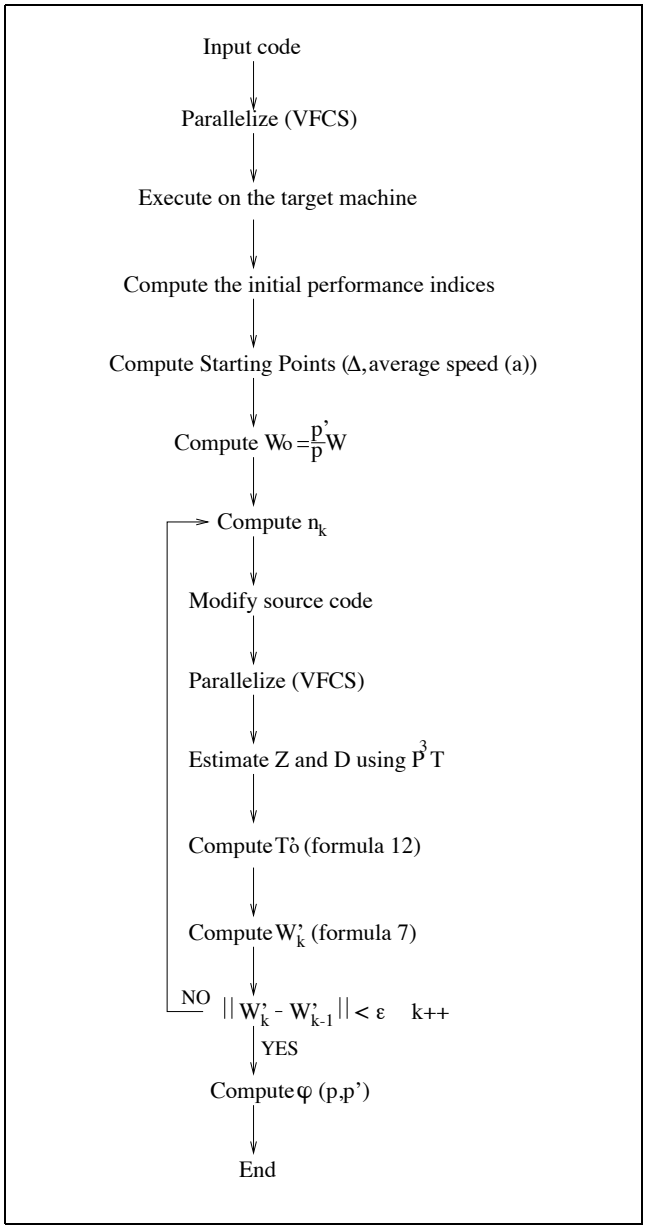

Figure 3: Predicting scalability under VFCS

$\psi(p,p')$	$p'=8$			$p'=16$		
	Pred	Meas	diff	Pred	Meas	diff
p=4	0.524	0.565	7.8 %	0.445	0.477	7.1 %
p=8	1.000	1.000	0%	0.851	0.846	0.5%
p=16				1.000	1.000	0%

Table 3: Redblack: 2D distribution, Predicted and Measured Scalability

Alg.	p=8			p=16		
	Pred	Meas	diff	Pred	Meas	diff
J_2D	2603	2532	2.7%	3089	3066	0.7%
J_C	2373	2313	2.5%	2971	2944	0.9%
R_2D	10611	9840	7.2%	11324	11641	2.7%

Table 4: Predicted and Measured Execution times (in $\mu secs$)

Redblack, which is confirmed by measured results as given in Table 4.

A more interesting result is given by the two different versions of the Jacobi algorithm. From Tables 1 and 2, we can see that the 2D distribution implementation has a larger initial execution time and a better scalability, on $p = 16$, than that of column distribution. According to Theorem 2, there will be a crossing point at some scaled ensemble size p'. However, in this case the crossing point is greater than 16 and cannot be confirmed by our testing environment. Table 5 shows the scaled input data sizes (parameter n) for Jacobi with the two data distribution strategies used. As we see, at $p = 8$ the isospeed scalability

Algorithm	p=4	p=8	p=16
Jacobi_2D	64	105	161
Jacobi_C	64	105	165

Table 5: Scaled input data sizes

is maintained at the same data size ($n = 105$). The initial problem size used in Tables 1, 2, and 3 is determined by the asymptotic speed [12] for best performance, where $n = 64$ is chosen. As pointed out in [12], the difference of isospeed scalability between "good" and "bad" algorithms increases with the communication/computation ratio. For Jacobi, the communication/computation ratio increases with the decrease in problem size. At the initial state $p = 4$, and $n = 20$, the execution time for Jacobi with column distribution strategy is $T_4(20) = 594\ \mu sec$ and for Jacobi with 2D distribution it is $\alpha * T_4(20) = 753\ \mu sec$, where $\alpha = 1.267$. Considering the scalability results of Table 6, we see that for $p' = 8$, the 2D distribution (Table 6a) scales better than that of column distribution (Table 6b). The ratio between the two predicted scalabilities, $\frac{0.652}{0.373} = 1.747$, is greater than α. Therefore, by Definition 2, $p' = 8$ is a crossing point where the execution time of 2D distribution becomes less than that of column distribution implementation. This performance crossing is due to the communication behavior involved on iPSC/860 for $p' = 8$ and is confirmed by measured performance as shown in Figure 4. Table 7

(a)

$\psi(p,p')$	$p'=4$	$p'=8$	$p'=16$
p=4	1.000	0.652	0.548
p=8		1.000	0.840
p=16			1.000

(b)

$\psi(p,p')$	$p'=4$	$p'=8$	$p'=16$
p=4	1.000	0.373	0.333
p=8		1.000	0.893
p=16			1.000

Table 6: Predicted scalability for Jacobi with (a) 2D distribution and (b) column distribution

presents the scaled input data sizes for Jacobi starting the scalability prediction with $n = 20$.

Algorithm	p=4	p=8	p=16
Jacobi_2D	20	33	50
Jacobi_C	20	43	64

Table 7: Scaled input data sizes for crossing-point testing

5 Conclusion

There are many ways to parallelize an application, and the relative performance of different parallelizations vary with problem size and system ensemble size. Comparing the performance of different implementations over a range of system and problem sizes is crucial in developing effective restructuring compilation

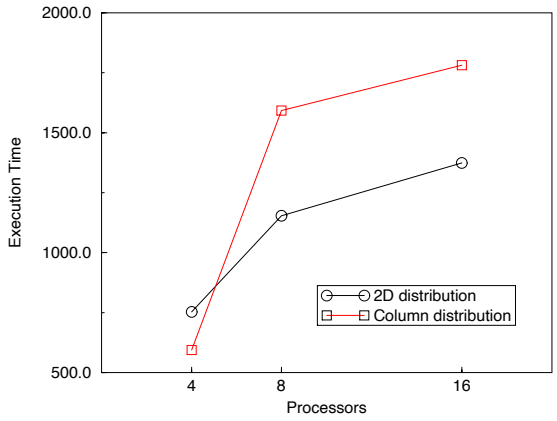

Figure 4: Scaled crossing point of the Jacobi with n=20

systems and ultimately in reducing the burden of parallel programming. In this study a practical methodology is developed and tested for automatic range comparison in a data-parallel compilation system. The proposed methodology is built on rigorous analytical models. Experimental results confirm its effectiveness in a restructuring system.

This work is a part of our current effort in development of the SCALA (SCALability Analyzer), an integrated performance analysis system. SCALA is an advanced system that is comprised of performance prediction techniques, advanced post-execution, and scalability analysis to compute performance indices that reflect the behavior of scalable parallel programs. It is integrated into restructuring systems in order to exploit knowledge from the compiler and guide the user as well as the compiler in the selection of restructuring transformations and optimization strategies.

References

[1] B. Chapman, P. Mehrotra, and H. Zima, "Programming in Vienna Fortran," *Scientific Programming*, vol. 1, pp. 31–50, 1992.

[2] H. P. Fortran Forum, "High performance Fortran language specification version 1.0." Technical Report, Department of Computer Science, Rice University, May 1993.

[3] V. S. Adve, J. M. Crummey, M. Anderson, K. Kennedy, J.-C. Wang, and D. A. Reed, "An integrated compilation performance analysis environment for data parallel programs," in *Proc. of Supercomputing*, (San Diego, CA), Dec. 1995.

[4] T. Fahringer, *Automatic Performance Prediction of Parallel Programs* . Kluwer Academic Publishers, Boston, USA, ISBN 0-7923-9708-8, March 1996.

[5] S. Benkner, S. Andel, R. Blasko, P. Brezany, A. Celic, B. Chapman, M. Egg, T. Fahringer, J. Hulman, Y. Hou, E. Kelc, E. Mehofer, H. Moritsch, M. Paul, K. Sanjari, V. Sipkova, B. Velkov, B. Wender, and H. Zima, *Vienna Fortran Compilation System - Version 2.0 - User's Guide*, October 1995.

[6] G. Fox, M. Johnson, G. Lyzenga, S. Otto, J. Salmon, and D. Walker, *Solving Problems on Concurrent Processors*, vol. 1. and 2. Englewood Cliffs, NY: Prentice Hall, 1988.

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

[8] V. Kumar, A. Grama, A. Gupta, and G. Karypis, *Introduction to Parallel Computing, Design and Analysis of Algorithms.* The Benjamin/Cummings Publishing Company, Inc., 1994.

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

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

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

[12] 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.

[13] X.-H. Sun, M. Pantano, and T. Fahringer, "Integrated range comparison for data-parallel compilation systems." Technical Report #97-004, Department of Computer Science, Apr. 1997.