

HETEROGENEOUS COMPUTING BASED ON PVM WITH DYNAMICALLY VARYING LOADS

SHIANG-JIUN CHEN
Dept. of Computer Science & Eng.
Univ. of Texas at Arlington
Arlington, TX 76019-0015
sjchen@uta.edu

HEE YONG YOUN & CHAN SU YU
School of Engineering
Info & Communication Univ.
Taejon, Korea
{youn,cyu}@icu.ac.kr

SEONG-MOO YOO
Dept. of Computer Science
Columbus State University
Columbus, Georgia 31907
yoo_seongmoo@colstate.edu

Abstract

Parallel Virtual Machine (PVM) has been identified as an effective system for heterogeneous computing. In this paper we investigate the performance of a heterogeneous distributed system based on PVM, where the computing nodes are connected each other through several levels of communication hierarchy and have different specifications. The loads of them also dynamically change. Three important numerical algorithms are solved as sample cases; heat equation, simple boundary condition, and Laplace equation. Experiment shows that almost linear speedup is achieved when the loads of the hosts are balanced and thus they can run at the similar speeds. It also reveals that the speedup gets more significant as the size of data is increased due to the reduced communication overhead.

Key words: Heterogeneous computing, load balancing, numerical algorithm, parallel virtual machine.

1 INTRODUCTION

Parallel and distributed processing has merits in performance and cost. Here the two widely employed paradigms are Massively Parallel Processors (MPP) and Heterogeneous Computing (HC) [1,2]. Even though MPP run much faster than HC, it costs more than ten million dollars while the latter costs far less. There exists a clear tradeoff between the two computing paradigms with respect to the performance and cost. Under this condition, Parallel Virtual Machine (PVM) [3] was developed for allowing efficient heterogeneous computing.

Previous works on PVM [4-6] have already identified its effectiveness. It has been used for solving various problems including material design, climate prediction, groundwater modeling, and so forth. It has also been found to be useful for solving large scale computation intensive problems. In these problems, matrix operations such as Cholesky, LU, and QR factorization, etc. are usually required to be solved. Using PVM for implementing these operations not only saves time, but also saves cost.

In earlier studies mostly a network of workstations (NOWs) [7] was used as the main platform. Here the

constituent nodes usually have the same (or similar) architectures and consequently same performances. They are also located closely with direct ethernet connection, and the performances are measured while the system is used for only the problems being solved. The performance of such system will be greatly different from that of the system with the heterogeneous environment. Here the nodes are connected through several levels of communication network hierarchy, and have quite different specifications. The loads of them also dynamically change. The main objective of this paper is to investigate the performance of distributed computing based on PVM with such fully heterogeneous environment.

In this paper, in order to achieve the objective, a distributed system based on PVM is built using the computer systems connected by the campus LAN in The University of Texas at Arlington. For the experiment three important numerical algorithms are taken as sample cases; heat equation, simple boundary condition, and Laplace equation. The PVM system is implemented in six computer systems whose performances are widely varying. Experiment shows that the PVM-based system allows almost linear speedup when the loads of the nodes are balanced such that the completion times of the assigned task to them are similar. We also study the communication performance of PVM by measuring the transmission time of various size data. It reveals that the larger the data, the more regular and efficient the communication time is.

The rest of the paper is organized as follows. Section 2 introduces the problems to be solved as sample cases. The experiment results are presented in Section 3 for various size problems. Section 4 provides a conclusion and the direction for the future research.

2 SAMPLE CASES

Under PVM, individual computer can be connected each other through a variety of networks that appear as one large distributed system. PVM provides functions which start up tasks in the virtual machine and allow the tasks to communicate and synchronize with each other automatically. Applications can be written in C/C++ or

Fortran 77 [8,9], and can be implemented using the PVM message passing libraries. There are two parts in a PVM system, the daemon (pvmd3) residing in all the constituent nodes (computers) constructing a virtual machine, and the libraries of PVM interface routines.

In order to study the effectiveness of heterogeneous computing based on PVM, we solve three important numerical problems; heat equation which is a kind of partial differential equation, simple boundary value problem arising in many applications, and finally Laplace equation. In this section we discuss the problems, methods employed for the solutions, and the applied parallelization approaches.

2.1 Heat Equation (Diffusion Equation)

Here we consider a one-dimensional heat equation. A function $u(x,y)$ satisfying the following equation is defined as heat equation (or diffusion equation).

$$\frac{\partial^2 u}{\partial x^2} = \frac{\partial u}{\partial y}$$

In order to solve the equation we need to get a discretization form of it, which is

$$\frac{u_{i+1,j} - u_{i,j}}{\Delta y} = \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{\Delta x^2}.$$

Then we obtain the explicit formula

$$u_{i+1,j} = u_{i,j} + \frac{\Delta y}{\Delta x^2} (u_{i,j+1} - 2u_{i,j} + u_{i,j-1})$$

The initial and boundary conditions are

$$u(y,0) = 0, \quad u(y,1) = 0, \quad \forall t,$$

$$u(0,x) = \sin(\pi x), \quad \forall 0 \leq x \leq 1.$$

To solve a problem in parallel using PVM, the input data need to be partitioned and packed into several blocks, and then sent to slave processes using PVM routines. When each child node running a slave process receives a message, it unpacks the message and does what the parent node running the master process wants it to do. After that, each child node packs the result and sends it back to the parent node. When the parent node sends messages to children nodes, it waits until messages are received from all the children nodes.

The computation involved in the heat equation is applying a recursive function to a collection of points in the complex plane until either the function values reach a specific value or begin to diverge. The heat equation is solved in parallel by the following procedure.

BEGIN:
PARENT NODE
Step 1: While(TRUE) DO Step 2-10

Step 2: Get tid and child tid
Step 3: For I=0,1,2, ..., NPROC DO Steps 4-5
Step 4: pack the data
Step 5: send the data to children
WAIT FOR RESPONSE

CHILD NODE

Step 6: get mytid and my parent tid
Step 7: While (true) do step 8-10
Step 8: receive the message from parent
Step 9: exchange the boundary condition with other children
Step 10: finish computation and send the result to parent

PARENT NODE

Step 11: Receive the solution from children and update the solution
END

Matrix multiplication is one of the basic operations required in many important algorithms. It can be accomplished by

$$c[i][j] = \sum a[i][k] \times b[k][j] \quad 0 \leq i, j, k \leq m-1,$$

where A, B, and C are $m \times m$ matrices. The worst case running time of matrix multiplication is $O(m^3)$. For parallel matrix computations, matrices are partitioned and then distributed to several processors. There are two ways to do it; column-wise blocking and row-wise blocking [10]. With the row-wise (column-wise) blocking, each row-wise (column-wise) block is sent to a different processor. For example, if we want to calculate $A \times B$ using n processors, we partition Matrix A into n row blocks, and assign each block to each processor. Matrix B is sent to all the processors.

2.2 Simple Boundary Value Problem

As an example, consider the following second-order differential equation

$$-u''(x) = 0$$

where the boundary value is equal to 100. To solve this problem, the centered difference method is used to reduce the order. After that, Jacobi Algorithm is applied to calculate the value.

2.2.1 Centered Difference Method

We use centered difference method to calculate the linear second-order boundary-value problem, $y''(x_i) = 0$, $x_i = 0$ on D, $x_i = c$ on Boundary(D), where D is a domain in $[0,1] \times [0,1]$. Then expanding y in second-degree Taylor polynomial about x_i and evaluating y at x_{i-1} and x_{i+1} ,

$$y(x_{i+1}) = y(x_i + 1) = y(x_i) + y'(x_i) + \frac{1}{2}y''(x_i) + c_1,$$

$$y(x_{i-1}) = y(x_i - 1) = y(x_i) - y'(x_i) + \frac{1}{2}y''(x_i) + c_2,$$

Then we obtain

$$y''(x_i) = y(x_{i+1}) - 2y(x_i) + y(x_{i-1}) + c.$$

From $D = 0$ and $x = 0$, $Ax = b$ is derived using centered differencing. Then Jacobi algorithm is used to solve $Ax = b$.

2.2.2 Jacobi Algorithm

The input here are an initial approximation $x^{(0)}$, the number of equations and unknowns, the entries $1 \leq i, j \leq n$ of the matrix A , the entries b_i , the entries XO_i , $1 \leq i \leq n$ of $XO = x^{(0)}$, tolerance(TOL), and maximum number of iterations N . The outputs are the approximate solution x_1, \dots, x_n . If the number of iterations exceeds the bound while the solution is beyond the tolerance, the execution is stopped with an error message. The following is the pseudo code for Jacobi algorithm.

BEGIN

Step 1. Set $k = 1$.

Step 2. While ($k \leq N$) do Steps 3-6

Step 3. For $i = 1, \dots, n$

$$\text{set } x_i = \frac{-\sum_{j=1, j \neq i}^n (a_{ij}XO_j) + b_j}{a_{ii}}$$

Step 4. If $\|x - XO\| < TOL$ then OUTPUT (x_1, \dots, x_n);

Step 5. Set $k = k + 1$.

Step 6. For $i = 1, \dots, n$

set $XO_i = x_i$

Step 7. OUTPUT ('Maximum number of iterations exceeded');

(Procedure completed unsuccessfully.)

STOP

2.3 Laplace Equation

Physical phenomenon involving more than one variable is expressed using the equations with partial derivatives. Here, we present a brief introduction to some of the techniques available for approximating the solution of partial-differential equations involving two variables. The definition of the Laplace Equation is

$$\frac{\partial^2 u(x, y)}{\partial x^2} + \frac{\partial^2 u(x, y)}{\partial y^2} = 0,$$

$$u(x, y) = g(x, y)$$

This problem is solved by transforming it to linear system equations. The method we use is central-difference method. Following that we use Jacobi algorithm to solve it.

2.3.1 Central-Difference Method

The Laplace equation is

$$\Delta u = 0 \text{ for } (x, y) \text{ in } D$$

$$u(x, y) = g(x, y), \text{ for } (x, y) \text{ in } S$$

Here D is the domain as $D = \{(x, y) | a < x < b, c < y < d\}$, and S denotes the boundary of D . The finite-difference method is used to transform the equation to a linear system problem. The first step is to choose interval numbers n and m . Then define the step sizes h and k by $h = (b-a)/n$ and $k = (d-c)/m$. Partitioning the interval $[a, b]$ into n equal parts of width h and the interval $[c, d]$ into m equal parts of width k provides a grid on the rectangle D by drawing vertical and horizontal lines through the points with coordinates (x_i, y_j) , where $x_i = a + ih$, for each $i = 0, 1, \dots, n$ and $y_j = c + jk$, for each $j = 0, 1, \dots, m$. Here $u[i][j]$ is obtained by averaging the four surrounding values, and Figure 1 depicts it.

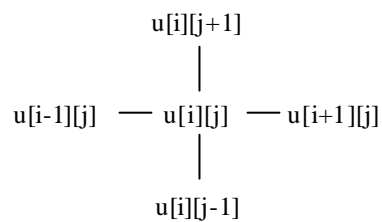


Figure 1. Computation of $u[i][j]$.

Then we use Taylor series for the variable x on x_i to generate the central-difference formula

$$\frac{\partial^2 u}{\partial x^2}(x_i, y_j) = \frac{u(x_{i+1}, y_j) - 2u(x_i, y_j) + u(x_{i-1}, y_j))}{h^2} + c_1$$

This method is similar to the central-difference formula for variable y on y_j which is generated by

$$\frac{\partial^2 u}{\partial y^2}(x_i, y_j) = \frac{u(x_i, y_{j+1}) - 2u(x_i, y_j) + u(x_i, y_{j-1}))}{k^2} + c_2$$

Using these two formulas, we get the Laplace equation at points $f(x_i, y_j)$ as

$$\frac{u(x_{i+1}, y_j) - 2u(x_i, y_j) + u(x_{i-1}, y_j)}{h^2} + \frac{u(x_i, y_{j+1}) - 2u(x_i, y_j) + u(x_i, y_{j-1})}{k^2}$$

for each $i = 1, 2, \dots, (n-1)$ and $j = 1, 2, \dots, (m-1)$, and the boundary condition

$$u(x_0, y_j) = g(x_0, y_j) \quad \text{for each } j = 0, 1, \dots, m,$$

$$u(x_n, y_j) = g(x_n, y_j) \quad \text{for each } j = 0, 1, \dots, m,$$

$$u(x_i, y_0) = g(x_i, y_0) \quad \text{for each } i = 0, 1, \dots, n-1,$$

$$u(x_i, y_m) = g(x_i, y_m) \quad \text{for each } i = 0, 1, \dots, n-1.$$

Therefore, we can transform the partial differential equation to the system of equations $Ax = b$,

$$\text{where } A = \begin{bmatrix} 4 & -1 & 0 & 0 & \dots & \dots & \dots \\ -1 & 4 & -1 & 0 & \dots & \dots & \dots \\ & & -1 & 4 & -1 & & \dots \\ & & & -1 & 4 & -1 & \dots \\ & & & & & \dots & \dots \\ & & & & & & \dots \\ 0 & 0 & \dots & \dots & \dots & -1 & 4 \end{bmatrix}.$$

The solution of this formula involves approximations of $u(x,y)$ using the points $(x_i, y_j), (x_{i-1}, y_j), (x_{i+1}, y_j), (x_i, y_{j-1}),$ and (x_i, y_{j+1}) . For the Laplace equation, we use the same way employed for solving the simple boundary condition problem. The difference between the two is that the simple boundary condition problem is one-dimensional equation, while the Laplace equation is two-dimensional equation. Therefore only the Step 5 of the simple boundary condition problem is changed into two loops for two-dimensional computation.

3 EXPERIMENTS

In this section we investigate the performance of a heterogeneous computing system based on PVM using the problems introduced in the previous section. We first show the machines included, and then study the communication speed of PVM system since it is a very important factor deciding the overall performance.

3.1 The Host Machines

Six different hosts are used to run the test programs, and they are listed in Table I.

Table I. The host machines.

Host	Arch	CPU	OS	Clock	Mem
csr	SUN4	SUN4	OS 4.2	400M	98M
banach	Alpha	Alpha	OSF-1	400M	2G
math	SGI64	IP28	IRIX	195M	2G
omega	Alpha	400	OSF-1	400M	1G
ranger	SUNMP	Sparc	Solaris	50M	32M
gamma	SUNMP	Ultrasparc	Solaris	250M	2G

We check the communication speed between different hosts. For this, we choose 'csr' as the console. A program measuring the communication time is run on the console. When the console gets the 'tid' from all the hosts, it packs a message and sends it to the hosts. As soon as a host receives a message, it echos the message back to the console. The message send (actually roundtrip) time is listed in Table II(a) - (d). Table II(a) is the communication time to itself (hence to 'csr'), and II(b) - (d) are the time between 'csr' and others. The load of each host when the data are collected are 'csr' = 2.01%, 'math' = 2.01%, 'gamma' = 4.02%, 'ranger' = 2.04%, 'banach' = 0.02%, and 'omega' = 0.2%, respectively. We also measure the time to pack the message. We run the program many times for the message sizes of 100, 1000, 10000, 100000, and 1000000 bytes. Here two representative data for each operation are listed, while the unit is byte/second. Note from the tables that the communication time varies a lot when the size of data is relativity small. It becomes stable as the size of data increases. This shows that, for efficient communication, the data size needs to be as big as possible. The frequency of communication also needs to be as small as possible, especially for the small size data.

Table II(a). Sending / Packing speed on host 'csr'.

Size	10 ²	10 ³	10 ⁴	10 ⁵	10 ⁶
Send	0.20	0.33	2.22	2.98	3.21
Send	0.03	0.33	1.25	3.12	3.23
Pack	100	1000	20	9.53	8.73
Pack	0.20	1000	10000	9.53	8.93

Table II(b). Time between 'csr' and 'math'.

Size	10 ²	10 ³	10 ⁴	10 ⁵	10 ⁶
Send	0.02	0.13	0.10	0.14	0.17
Send	0.02	0.12	0.07	0.18	0.19
Pack	100	1000	10	8.70	8.89
Pack	0.20	1000	20	9.50	8.80

Table II(c). Time between 'csr' and 'omega'.

Size	10 ²	10 ³	10 ⁴	10 ⁵	10 ⁶
Send	0.02	0.13	0.06	0.20	0.16
Send	0.02	0.20	0.20	0.22	0.19
Pack	100	1000	20	9.10	8.70
Pack	100	1000	20	9.10	8.97

Table II(d). Time between 'csr' and 'gamma'.

Size	10 ²	10 ³	10 ⁴	10 ⁵	10 ⁶
Send	0.07	0.50	2.00	3.12	3.21
Send	0.04	0.33	1.67	3.23	3.28
Pack	0.20	1000	10	8.70	8.40
Pack	100	1000	10	8.70	8.97

Comparing Table II(a) and Table II(b), the speed for send in Table II(a) is higher than that of Table II(b). It is reasonable because Table II(a) is for the communication to itself. The speed for packing is almost the same regardless of the hosts. Also notice that the send time to 'gamma' is much faster than to the other hosts, which is mainly due to the relatively higher performance of host 'gamma' than the others.

3.2 Computation Performance for Heat Equation

Here the heat equation is solved using three hosts; 'csr', 'gamma', and 'ranger'. Their relative speeds are first compared by running both the C code of the problem and the PVM version in each machine separately. The result is summarized in Table III, which compares the computation time.

Table III. Heat equation.

		size		
		500	1000	5000
csr	C	5.3	5.98	10.8
	PVM	5.8	6.75	12.7
gamma	C	5.1	5.79	10.9
	PVM	18.25	25.63	44.5
ranger	C	6.7	7.9	14.1
	PVM	-	-	-

Note that the PVM version is slower than the c code version. This is due to the PVM function call overheads. Also notice that 'csr' and 'gamma' are slightly faster than 'ranger'. The heat equation is then executed using one to three hosts for different size problems of 500, 1000, and 5000. Table IV lists the data, and Figure 2 plots the speedup.

Table IV. Comparison of computation speed.

		size		
		500	1000	5000
gamma		18.25	25.63	44.5
csr,gamma		10.34	11.25	20.01
csr,gamma,ranger		6.43	6.75	11.21

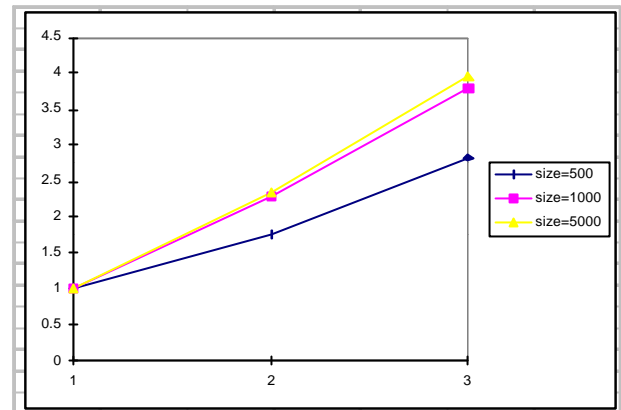


Figure 2. Speedup with different number of hosts.

When the data size is 500 and the number of nodes is 2, the resulting speed up is 1.76(18.25/10.34). For the same size problem but with 3 nodes, the speed up is 2.84. When the problem size increases to 1000 and the number of nodes is 2, the speed up is 2.28. We get a speed up of 3.80 when three nodes are used for the same size problem. For the last case of 5000, the speed up is 2.34 and 3.97 for two and three nodes, respectively. When the participating nodes have similar computing power, more number of nodes in the system allows the higher speed up as shown in the experiment above. Also, as the size of the problem increases, the speedup becomes more significant.

We run the program again, using 'csr' as console. When the experiment is done, the load of the console 'csr' is very low as 0.01% while the load of 'gamma' is very high as 16.5%. That of 'ranger' is 0.02%. This is for testing the effect of load conditions of the hosts on the overall performance. Table V is the result.

Table V. Performance with unbalanced loads.

		size		
		500	1000	5000
csr		5.8	6.75	12.7
csr,gamma		36.8	43.27	59.7
csr,gamma,ranger		26.4	31.02	36.6

In the case of two unbalanced hosts where 'gamma' has much higher load than 'csr', if the load is evenly distributed between them, the speed is even much lower than the single machine computation using 'csr'. In the case of three hosts ('csr,' 'gamma,' and 'ranger'), the result is slightly improved compared to the two host case since the load of the host 'ranger' is reduced. We observe the similar results with other combinations of hosts.

3.3 Performance for Simple Boundary Equation

The result is summarized in Table VI and VII. Here the respective loads are; 'gamma' = 4.02%, 'math' = 2.01%,

'banach' = 1.02%. For the PVM implementation we try two different languages; Fortran 77 and C.

Table VI. Simple boundary problem based on C.

	size		
	100	1000	5000
gamma	2.65	2.7	5.3
gamma,math	1.68	1.73	3.2
gamma,math,banach	1.39	1.36	2.5

Table VII. Simple boundary problem based on Fortran.

	size	
	100	1000
gamma	2.67	2.75
gamma,math	1.67	1.73
gamma,math,banach	1.34	1.35

Notice that the results are almost the same for both languages in terms of the speedup and absolute computation time.

3.4 Computation Performance for Laplace Equation

For Laplace equation we use 'csr', 'gamma', and 'ranger'. Table VIII summarizes the results. Here the loads during the experiment are almost balanced as 'csr' = 2.01%, 'gamma' = 4.02%, 'ranger' = 1.02%.

Table VIII. Computation speed for Laplace equation.

	size	
	1000	5000
csr	44.5	104.7
csr, gamma	20	43.8
csr, gamma, ranger	6.75	17.3

When the problem size is 1000 and the number of hosts is 2, the resulting speedup is 2.23. For the same size but with 3 nodes, the speedup is 6.60. When the problem size increases to 5000 and the number of hosts is 2, the speedup is 2.39. We get a speedup of 6.05 with three hosts for the same size. We here confirm that the speedup increases as more nodes are added as far as the loads of them are balanced.

4 CONCLUSION

We have studied the performance of heterogeneous distributed computing based on PVM using three different numerical problems. PVM can efficiently partition a job and distribute them to several hosts so that they can run in parallel. We have found that the speedup of the computation can be linearly increased as more number of hosts are used if their loads are balanced. PVM can be said to be an efficient system for solving computation intensive

problems in distributed platform. One difficulty is that if some hosts stall, the master process halts without any notice. A corrective measure needs to be implemented.

As identified from this study, the performance of heterogeneous computing is strongly dependent on the load condition of the hosts. In order to minimize the variance of the loads during the experiments, thus, all the experiment were done during midnight. In addition to the load factor, the network structure and communication speed between the hosts and the application problems solved are also important factors influencing the performance. A deeper analysis of various experiment results is required to quantify the interrelationship between these factors. This will allow us the way for dynamically reflecting the conditions in distributing the jobs, and consequently maximizing the performance.

5 REFERENCES

- [1] Kumar, Vipin, Grama, Ananth, Gupta, Anshul, Karypis, George. *Introduction to Parallel Computing: Design and Analysis of Algorithms*. Addison Wesley, 1994.
- [2] A. Clematis, A. Corana, *Performance Analysis of Task-Based Algorithms on Heterogeneous Systems*, Lecture Notes in Computer Science, Vol. 1497, Springer-Verlag, pp. 11-18, 1998.
- [3] Al Geist, Adam Beguelin, Jack Dongarra, Weicheng Jiang, Robert Manchek, and Vaidy Sunderam. *PVM: Parallel Virtual Machine*. MIT press, 1994.
- [4] M. Šoch, J. Trdlí cda, P.Turidí k, *PVM, Computational Geometry, and Parallel Computing Course*, Lecture Notes in Computer Science, Vol. 1156, Springer-Verlag, 1996.
- [5] J. Touriño, R. Doallo, *A PVM-Based Library for Sparse Matrix Factorizations*, Lecture Notes in Computer Science, Vol. 1497, Springer-Verlag, pp.304-311, 1998.
- [6] G.A. Geist, J.A. Kohl, P.M. Dapadopouloos, S.L. Scott, *Beyond PVM 3.4: What we have learned, What Next, and Why*, Oak Ridge National Laboratory, Computer Science and Mathematics Division, Oak Ridge.
- [7] Peter Carlin. *Distributed Linear Algebra on Networks pf Workstations. Thesis*. California Institue of Technology, 1994.
- [8] M. Kemelmakher, O. Kremien, *Scalable and Adaptive Resource Sharing in PVM*, Lecture Notes in Computer Science, Vol. 1497, Springer-Verlag, 1998.
- [9] Paul Gray, Alan Krantz, Soeren Olesen, Vaidy Sunderam, *Advances in Heterogeneous Network Computing*, Lecture Notes in Computer Science, Vol.1497, Springer-Verlag, 1998.
- [10] H.Y. Youn, E.N. Huh, and S.M. Yoo, "Maximum Load Balancing and Processor Utilization for Solving Linear Systems," *ISCA Parallel and Distributed Computing Systems (PDCS '99)*, pp. 351-356, Aug. 1999.