Computers Elect. Engng Vol. 18, No. 2, pp. 145–152, 1992
Printed in Great Britain.

0045-7906/92 $5.00 + 0.00

PARALLEL MATRIX MULTIPLICATION ON NETWORKED MICROCOMPUTERS

ÇETIN K. KOÇ and SENG C. GAN
Department of Electrical Engineering, University of Houston, Houston, TX 77204, U.S.A.

(Received 30 January 1991; accepted in final revised form 17 June 1991)

Abstract—We present methods for utilizing parallel processing capability of the idle microcomputers on a local area network to perform computationally intensive operations frequently encountered in linear algebra. By making use of the data communication properties of the network, parallel algorithms for multiplication of large matrices have been designed and implemented. The techniques presented in this paper can be used to obtain efficient implementations of basic linear algebra subroutines.

1. INTRODUCTION

A typical local area network consists of hundreds of computers, many of which are unused much of the time. The research presented in this paper is a step toward utilization of these idle computers for parallel processing of some computationally intensive operations encountered in linear algebra. We have programmed a microcomputer network so that it emulates a shared-memory multiprocessor, and explored techniques for efficient implementation of basic linear algebra operations. The techniques and algorithms attempt to exploit the properties of the local area network (LAN) in order to obtain high-performance.

The ultimate aim of this effort is to develop a software package for implementation of basic linear algebra operations on networked microcomputers. To the best of our knowledge, no such implementation exists. However, optimized implementations of linear algebra primitives on several different computing environments, e.g. on vector computers and parallel computers, have been reported. These efforts aim to obtain efficient implementations of Basic Linear Algebra Subprograms (BLAS), which is a set of basic routines to solve linear algebra problems. The interfaces to BLAS are standardized so that applications run on a wide range of computers. This standardization allows programmers to custom-code BLAS for each particular computer, improving the efficiency of the implementation. This effort has been very useful and highly successful; the original BLAS [6] was used in a wide range of software including LINPACK [3].

The original BLAS specified vector–vector operations, in other words, operations that are of $O(n)$ where n is the dimension of the vectors involved. Extended levels of BLAS have also been specified, suitable for implementation on a wide range of advanced computer architectures. Level 2 BLAS specifies matrix–vector operations [5], while Level 3 BLAS specifies matrix–matrix operations [4]. The complexity of operations required in Levels 2 and 3 of BLAS are $O(n^2)$ and $O(n^3)$, respectively.

The reason for the evolution of higher levels of BLAS is the development of computers with increasingly powerful components. In addition to advantages such as portability, clarity, and ease of software maintenance, the original BLAS (Level 1 BLAS) and Level 2 BLAS exploit the vector processing feature of supercomputers. Later on, Level 3 BLAS was created to take advantage of hierarchical memory structures found in recent parallel computers (see [4] and the references therein).

In this paper we present techniques for efficient implementation of Level 3 BLAS primitives on networked microcomputers. This is achieved by programming the local area network to emulate a shared-memory multiprocessor, where data files on the file server take the role of the shared main memory. We note that the data transmission rate of the network depends on several factors, e.g. the size of the data packets and the probability of packet collision. Thus, an efficient implementation requires close inspection of the data transmission properties of local area networks. The following section reviews the fundamentals of local area networks and the factors affecting its

performance. The techniques presented in Section 3 make use of some of the properties of local area networks. Details of the implementation and measurements are given in Section 4. Finally, we conclude the paper in Section 5 by summarizing the techniques and the results of the experiments.

2. REVIEW OF MICROCOMPUTER NETWORKS

The most popular and widely used microcomputer network is *Ethernet* [8], which is designed to handle large and irregular data transfers. In Ethernet networks, a message is assembled into a sequence of bits called a *packet*. The process of forming the bit sequence and extracting data from the sequence is called *packet assembly and disassembly* (PAD). This process takes a considerable amount of computing time. Packets are broadcast on a cable connecting the nodes of the network.

For large networks, *bridges* are used to interconnect multiple segments of the Ethernet network [10,9]. A bridge forwards packets if the destination node is located in the other segment of network. As a result, local packet traffic (packet transfers between nodes in the source network) does not appear on the other networks. Bridges improve reliability, since the failure of one LAN does not corrupt other LANs. Performance of the LAN is also improved since fewer packets appear on each LAN, causing less congestion. Furthermore, the effective path length of the medium is shorter. As a result, collisions can be detected and resolved faster, resulting in higher network throughput.

Media access management protocol of the Ethernet is known as the Carrier Sense Multiple Access and Collision Detection (CSMA/CD) method. This protocol controls access to the cable and resolves conflicts [7]. A node executes the following steps for packet transmission:

(1) If the medium is idle, the node transmits the packet.
(2) If the medium is busy, the node waits until the network is idle, and then transmits the packet.
(3) If there is a collision during the transmission, the node ceases the transmission.
(4) When collision is detected, the node signals all nodes in the network to stop the transmission.
(5) After the signal, the node waits a random amount of time before trying to retransmit.

An heuristic called *binary exponential backoff* is used to determine the amount of time delay before trying retransmission [10].

The performance of Ethernet depends on several factors. Current research shows that typical traffic level of Ethernet networks is very low, consisting of short bursts of large file transfers followed by long periods of no traffic [2]. The rate of packet collision is usually very low under normal working conditions. The most significant factors that influence the network throughput are the packet segmentation and the PAD abilities of the nodes in the network.

An Ethernet packet consists of about 100 bytes of header information and up to 1400 bytes of data. The data transmission rate of the LAN degrades significantly if there is a large number of small packets. For example, the maximum packet size for DECNET/DOS is 1411 bytes. The file server sends multiple packets to the client if the data requested has a size larger than 1411 bytes, which results in more PAD time. If the packets are sent through a bridge, the processing time needed there further reduces the transmission rate. Obviously, higher transmission time increases the probability of collision and degrades the performance of the network.

Typically, a microcomputer network consists of two kinds of nodes: *file servers* and *clients*. In general, file servers have faster CPUs, better network interface cards, larger memory, and more advanced operating systems than the clients. As a result, file servers can perform the PAD operations faster, and thus packet transfer between a client and the file server is faster than between any two clients.

3. ALGORITHM DESIGN TECHNIQUES

A high-performance implementation of Level 3 BLAS requires reduction of the communication overhead. This can be achieved by decreasing the amount of data transfer and the probability of packet collision, and also by overlapping computation with communication. The techniques we

have developed are *row partitioning*, *delayed execution* and *alternating computation and communication*. In the following, we elaborate on these techniques.

3.1. Row partitioning

Given $n \times n$ matrices, A, B and C, we perform the operation:

$$D := C + AB$$

on the microcomputer network by partitioning n rows of matrix D among p computers. Let $m = [n/p]$, then each one of the first $p-1$ processors computes m rows of matrix D, while the last processor computes the remaining $n - (p-1)m \leqslant m$ rows. Let A_i and C_i be the ith rows of the matrices A and C, respectively. The first computer reads $C_1, C_2, \ldots, C_m$ and $A_1, A_2, \ldots, A_m$, and the entire matrix B and computes $D_1, D_2, \ldots, D_m$. In general, the kth computer for $1 \leqslant k \leqslant p-1$ performs:

$$D_j := C_j + A_j B$$

for $(k-1)m + 1 \leqslant j \leqslant km$, while the last computer performs the above for $(p-1)m + 1 \leqslant j \leqslant n$.

This partitioning method allows each microcomputer to proceed independently. Matrices A, B, C and D are stored in the file server as data files to take advantage of the fast data transfer speed between the file server and the clients.

We note that A, B, C and D do not have to be square matrices as long as the partitioning produces tasks which are approximately the same size. Also, the parallel processing literature contains a large set of parallel algorithms for other linear algebra computations, e.g. for matrix inversion, singular value decomposition, etc. (See, for example, [1] and references therein.) The partitioning technique outlined above is applicable to these computational methods as well.

3.2. Delayed execution

The file server synchronizes the clients' clocks with its clock as a part of the file sharing procedure. We take advantage of this fact by allowing each one to start at a different time. The smallest amount of time between the starting times of any two nodes is called the *delay factor*. The delay factor has to be long enough so that the next node starts only after the previous node has finished communicating with the file server. If all nodes in the network are the same, the nodes will continue to communicate with the file server at different times. For microcomputer networks with different types of nodes, the delay factors of each node have to be different to achieve the same effect.

The communication speed of each node is highly dependent on the PAD processing capabilities of the nodes and the file server and also on the total amount of packet collisions. Packet assembly and disassembly processing capability of the node relies on the computing power of the node, while the total amount of packet collisions depends on the network traffic level at the moment. These factors are highly dependent on the actual network configuration.

3.3. Alternating computation and communication

If we have p microcomputers and the delay factor is equal to d, then the last node of the network has to wait $(p-1)d$ clock ticks before getting started. In order to reduce the latency, we allow each node to alternate computation and communication. First, each node reads a block of the matrices A, B and C. It then performs multiplications to compute a portion of D using the operands in its memory and writes the result to the file server.

The nodes continue to alternate between reading the remaining portions of the matrices, performing multiplications and writing the computed values (the entries of matrix D) to the file server.

3.4. The parallel algorithm

The above techniques can be used to design parallel algorithms of Level 3 BLAS primitives. The parallel algorithm for the matrix multiplication problem is illustrated in Fig. 1. According to the delayed execution technique, the second node starts d clock ticks after the first node, the third node starts d clock ticks after the second, and so on. The nodes start reading the portions of A, B and

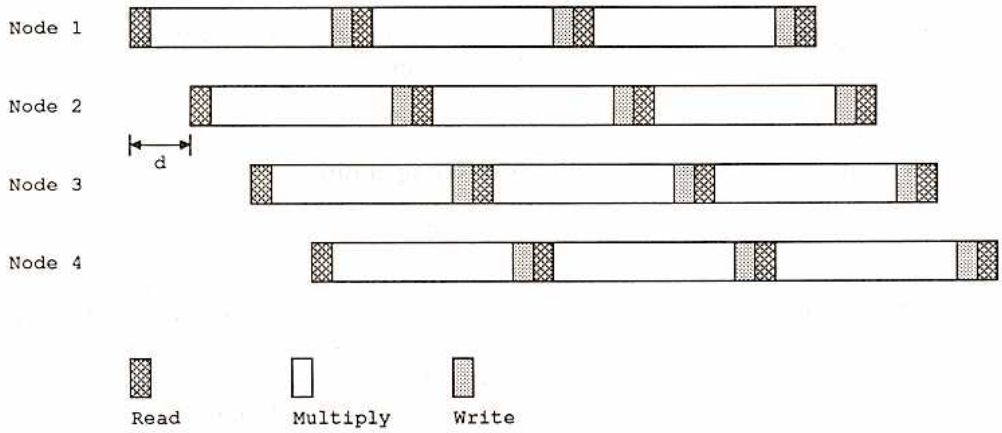

Fig. 1. Timing diagram of the parallel algorithm.

C, and perform multiplications. The partial result is written to a file in the file server. The nodes then alternate between reading, computing and writing until all entries of D have been computed.

An inner-product operation involving vectors of length n requires $2n - 1$ arithmetic operations (floating-point multiplications and additions). Let M be the number of floating-point arithmetic operations which a single node can perform in one second and N be the number of bytes a node can transfer in one second (without the presence of packet collisions). The delay factor needs to be long enough to allow the first node to read $3n$ operands ($=24n$ bytes, since a double-precision floating-point requires 8 bytes of storage) from the matrices A, B and C, before the second node starts communicating with the file server. It also needs to be small enough so that when the first node finishes performing the multiplications, the last node (pth node) just finishes reading the operands. This implies:

$$d \geqslant \frac{24n}{N} \quad \text{and} \quad pd \leqslant \frac{24n}{N} + \frac{2n-1}{M},$$

where p is the number of computers and d is the delay factor. A large number of packets needs to be transmitted across the network if the block size is large. Long transmission time increases the probability of packet collision. When packet collision happens, the corrupted packets will be retransmitted after a random amount of time. More packet collisions may still happen during the retransmission. However, if no packet collision occurs, then larger block size results in more efficiency. As a general rule, large block size is used only when the network has very little traffic.

The delay factor and block size are highly dependent on the computing performance of the node and the traffic level of the network. The computing performance of the nodes can be measured, however the traffic level of the network is highly unpredictable. A feasible way to pick the optimum parameters is by mesuring the performance of the algorithms using various block size and delay factor combinations.

4. IMPLEMENTATION AND MEASUREMENTS

In order to test the developed techniques, we have implemented and tested four algorithms on a microcomputer network of 12 computers, each of which is an IBM-PC/AT with an Intel 80287 numeric coprocessor. Each computer has about 400 kilobytes of free space after loading the networking software. The network operating system used in this cluster is DECNET/DOS Version 2.0. The file server a VAX 8650 minicomputer, which is also used for general computing work. The experiments have been performed to multiply 120×120 matrices with double-precision floating-point entries. With delay factor $d = 2$, the first node starts computation at the 0th clock tick, the second node starts at the 2nd clock tick, and so on. The Ethernet packet sizes are 1411 bytes for READ operations and 1459 bytes for WRITE operation.

Table 1. The *measured* minimum (min), maximum (max), average (avg) times of the parallel algorithms as a function of the delay factor (in clock ticks). The table also contains the standard deviations (SD) and the number of test runs (ntr)

	MAXBS					MINBS				
Delay factor	Min	Max	Avg	SD	Ntr	Min	Max	Avg	SD	Ntr
0	5424	6420	5696	299	9	6896	7601	7203	249	6
1	5696	5816	5736	69	3	7035	7422	7194	202	3
2	1050	5659	1581	1179	40	991	7395	1485	1380	39
	MAXBSA					MINBSA				
Delay factor	Min	Max	Avg	SD	Ntr	Min	Max	Avg	SD	Ntr
0	4889	5293	5036	177	4	5013	5843	5343	349	5
1	4986	5143	5042	88	3	4984	5127	5046	73	3
2	1000	1907	1265	222	36	813	5311	1202	696	39

The test programs are designed to explore the effect of the delay factor, the block size and the alternation technique on the performance. The number of test runs, the minimum, maximum, average completion times and the standard deviations are summarized in Table 1.

MAXBS (maximum block size without alternation). This algorithm reads all operands, performs the computation and writes the results to the file server (Fig. 2). In IBM-compatible computers, the maximum block size is 64 kilobytes (65,536 bytes). To avoid unnecessary packet segmentation, the program reads 64,906 bytes at once, which is an integer multiple of Ethernet packet READ size ($64{,}906 = 46 \times 1411$). Similarly, WRITE size is 64,196 bytes ($=44 \times 1459$).

MINBS (minimum block size without alternation). This program is similar to MAXBS except it reads 1411 bytes and writes 1459 bytes at once (Fig. 2).

MAXBSA (maximum block size with alternation). This program alternates between computation and communication. It reads and writes using maximum block sizes. It reads a block and performs some computation before reading and writing the next block (Fig. 3).

MINBSA (minimum block size with alternation). This program is similar to MAXBSA except that it reads and writes at the minimum block sizes (Fig. 3).

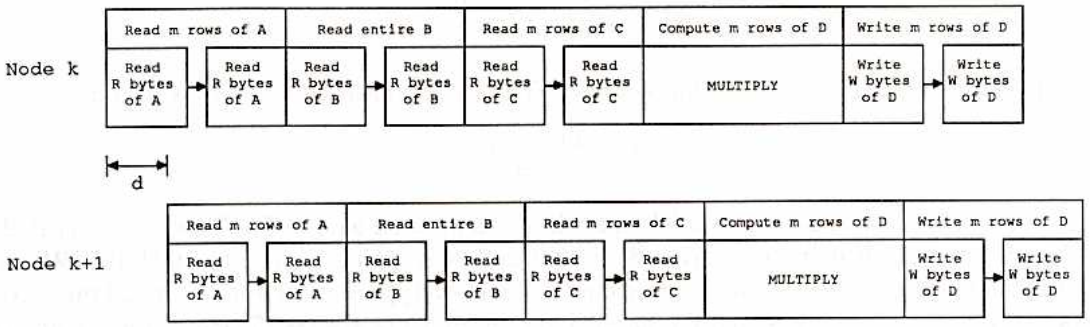

Fig. 2. Timing diagram of algorithm MAXBS ($R = 64{,}906$ and $W = 64{,}196$) and algorithm MINBS ($R = 1411$ and $W = 1459$).

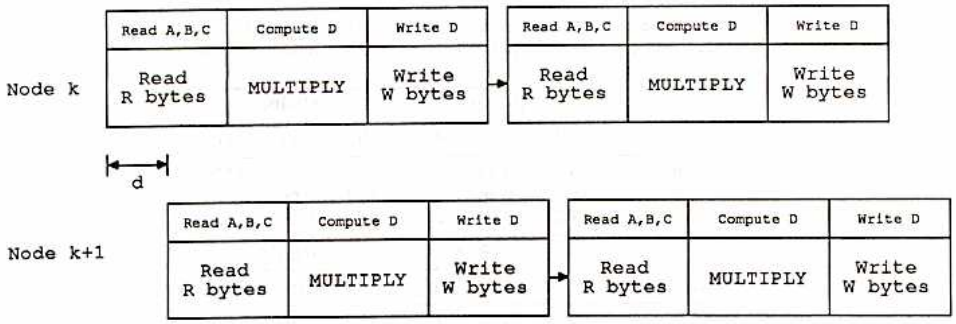

Fig. 3. Timing diagram of algorithm MAXBSA ($R = 64{,}906$ and $W = 64{,}196$) and algorithm MINBSA ($R = 1411$ and $W = 1459$).

We observe that the programs run faster when the delay factor is larger. The delayed execution technique decreases packet collisions and increases efficiency. Also programs employing the alternation technique run much faster than their counterparts and are also more *stable*, i.e. they have smaller deviations in the execution times. The effect of block size is highly dependent on the network traffic level. In the test runs, MAXBSA produced smaller deviations in the execution times than MINBSA. Furthermore, for this cluster of computers, large block size is more efficient than small block size. This implies that this particular network has low packet traffic for most of the time. The same matrix multiplication problem ($n = 120$, double-precision floating-point entries) was also run on a single computer in the network. It takes 4440 clock ticks for a single computer to perform the same computation. We give the measured maximum, minimum and average speedup of the parallel algorithms in Table 2.

The theoretical speedup of the parallel algorithms can also be computed however an accurate estimate of the speedup requires that the bandwidth of the network be accurately measured. Here we give a rough analysis of the speedup as a function of the network bandwidth N. Given $n \times n$ matrices A, B and C, the computation of $D = C + AB$ requires a total of $2n^3$ floating-point arithmetic operations, which takes

$$T_1 = \frac{2n^3}{M} \text{ s}$$

on a single computer. Our parallel algorithm equally partitions this work among p computers. Assuming the computers on the network have equal computing power, we conclude that p processors will require

$$T_p = \frac{T_1}{p} + T_c \text{s}$$

to perform the same computation, where T_c is the communication time. The host computer sends n/p rows of matrix A and the entire matrices B and C to each one of p processors and, after the computations are completed, the host receives n/p rows of matrix D from each of these processors. Thus, a total of

$$8\left[\frac{n^2}{p} + n^2 + n^2 + \frac{n^2}{p}\right]p$$

bytes of data will be broadcast on the network. The communication time T_c is found to be

$$T_c = \frac{2n^2 + 2n^2 p}{N}.$$

The *theoretical speedup* S of the parallel algorithm is computed as $S = T_1/T_p$, which is plotted in Fig. 4 as a function of n (ranging from 8 to 128) and N (ranging from 5 to 100 kb/s). We take $M = 16{,}000$ floating-point operations per second, which is a typical value for 287-based microcomputers. We observe from Fig. 4 that the parallel algorithm is not always efficient for every possible combination of matrix dimension n and network bandwidth N. For example, when $N = 60$ kb/s the parallel algorithm is efficient (i.e. the speedup is more than 1) for $n \geqslant 32$.

Table 2. The *measured* minimum (min), maximum (max), and average (avg) speedup of the parallel algorithms as a function of the delay factor

	MAXBS			MINBS		
Delay factor	Min	Max	Avg	Min	Max	Avg
0	0.82	0.69	0.78	0.64	0.58	0.62
1	0.78	0.76	0.77	0.63	0.60	0.62
2	4.23	0.78	2.81	4.48	0.60	0.62
	MAXBSA			MINBSA		
Delay factor	Min	Max	Avg	Min	Max	Avg
0	0.91	0.84	0.88	0.89	0.76	0.83
1	0.89	0.86	0.88	0.89	0.87	0.88
2	4.44	2.33	3.51	5.46	0.84	3.69

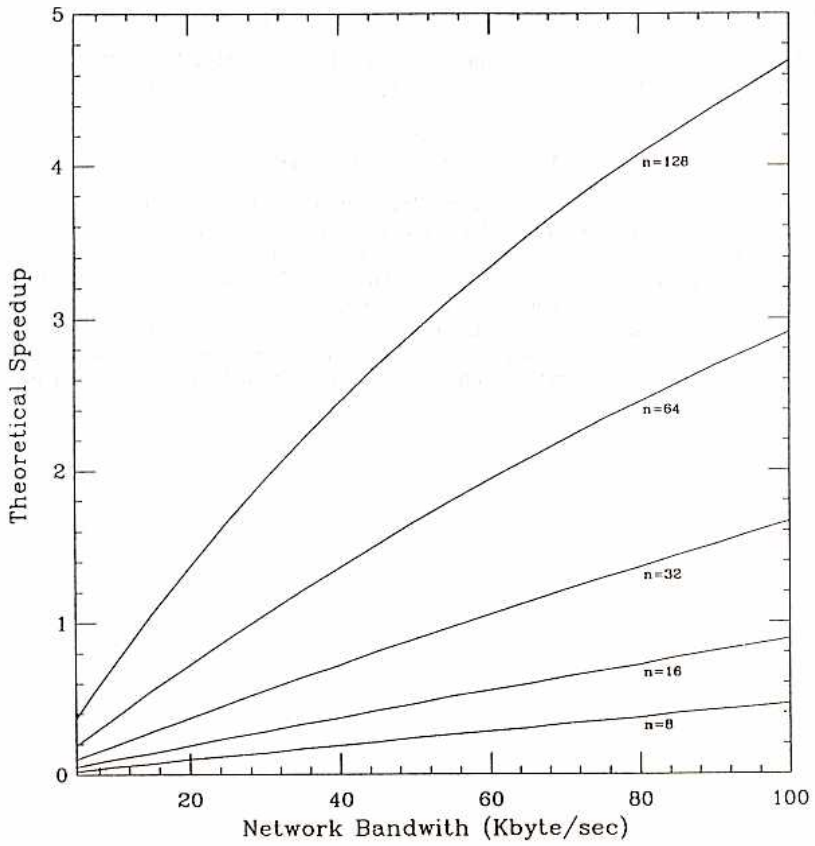

Fig. 4. The *theoretical* speedup of the parallel algorithm with $p = 12$ processors.

5. CONCLUSION

We have developed techniques to efficiently implement Level 3 BLAS primitives on networked microcomputers. Algorithms have been implemented in order to test the effectiveness of the techniques. We summarize the conclusions of this study below:

- The delayed execution method forces each microcomputer to communicate with the file server at a different time, reducing the probability of packet collision.
- The reduction of the number of packets for each data transfer can be achieved by transferring data to and from the file server with a block size producing a minimum number of packets. This technique reduces the overhead for the data transfer and increases the throughput.
- Alternating computation and communication decreases the latency.
- The experiments have indicated that the efficiency of the parallel algorithms is highly dependent on the delay factor and the block size.

REFERENCES

1. D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and Distributed Computation, Numerical Methods*. Prentice-Hall, Englewood Cliffs, NJ (1989).
2. D. R. Boggs, J. C. Mogul and C. A. Kent, Measured capacity of an Ethernet: myths and reality, *Comput. Commun. Rev.* **18,** 222–234 (1988).
3. J. J. Dongarra, J. Bunch, C. Moler and G. Stewart, *LINPACK Users' Guide*. SIAM (1979).
4. J. J. Dongarra, J. DuCroz, S. Hammarling and I. Duff, A set of level 3 basic linear algebra subprograms. *ACM Trans. math. Software* **16,** 1–17 (1990).
5. J. J. Dongarra, J. DuCroz, S. Hammarling and R. Hanson, An extended set of Fortran basic linear algebra subprograms. *ACM Trans. math. Software* **14,** 1–17 (1988).
6. C. Lawson, R. Hanson, D. Kincaid and F. Krogh, Basic linear algebra subprograms for Fortran usage. *ACM Trans. math. Software* **5,** 308–323 (1979).
7. J. Martin and K. K. Chapman, *Local Area Networks: Architecture and Implementations*. Prentice-Hall, Englewood Cliffs, NJ (1989).

8. R. M. Metcalfe and D. R. Boggs, Ethernet: distributed packet switching of local computer networks. *Communs. ACM* **19,** 395–404 (1976).
9. W. Stallings, *Data and Computer Communications*, 2nd edn. Macmillan, New York (1988).
10. A. S. Tanenbaum, *Computer Networks*. Prentice-Hall, Englewood Cliffs, NJ (1981).

AUTHORS' BIOGRAPHIES

Çetin K. Koç—Çetin Kaya Koç has been an Assistant Professor in the Department of Electrical Engineering at the University of Houston since June 1988. He received his B.S. (1980, summa cum laude) and M.S. (1982) degrees in electrical engineering from Istanbul Technical University and his M.S. (1985) and Ph.D. (1988) degrees in electrical and computer engineering from the University of California, Santa Barbara. His research interests include parallel computation, scientific computing, computer arithmetic, computer algebra and cryptography.

Seng C. Gan—Seng C. Gan received his B.S. and M.S. degrees in electrical engineering from the University of Houston in May 1987, and August 1990, respectively. Currently, he is working as a network administrator for the Star Enterprise, a division of Texaco. His research interests include computer networks, parallel processing, and artificial neural networks.