[3] W. G. Alexander and D. B. Wortman, "Static and dynamic characteristics of XPL programs," *Computer*, pp. 41–46, Nov. 1975.
[4] D. E. Knuth, "An Empirical Study of FORTRAN Programs," *Software-Practice Experience*, vol. 1, pp. 105–133, Apr.–June 1971.
[5] H. S. Stone, *et al.*, *Introduction to Computer Architecture*. Chicago, IL: Science Research Ass. Inc., 1975, ch. 9.
[6] R. O. Winder, "A data base for computer performance evaluation," *Computer*, pp. 25–29, Mar. 1973.
[7] D. E. Lang, T. K. Agerwala, and K. M. Chandy, "A modeling approach and design tool for pipelined central processors," *SIGARCH Newsletter*, vol. 7, pp. 122–129, Apr. 1979.
[8] B. L. Peuto and L. J. Shustek, "An instruction timing model of CPU performance," in *Proc. 4th Annu. Symp. Comput. Arch.*, Mar. 23–25, 1977, pp. 165–178.
[9] IBM System/370 Principles of Operation, Rep. GA22-7000-6, Mar. 1980.
[10] A. Lunde, "Empirical evaluation of some features of instruction set processor architectures," *Commun. Ass. Comput. Mach.*, vol. 20, pp. 143–153, Mar. 1977.
[11] Private communication with M. S. Goheen of Amdahl Corp.
[12] M. Kobayashi, "Instruction reference behavior and locality of reference in paging," Ph.D. dissertation, Univ. California, Berkeley, CA, 1980.

Optimality of a Two-Phase Strategy for Routing in Interconnection Networks

L. G. VALIANT

***Abstract*—It is shown that for *d*-way shuffle graphs all oblivious algorithms for realizing permutations in logarithmic time send packets along routes twice as long as the diameter of the graph. This confirms the optimality of the strategy that sends packets to random nodes in a first phase and to the correct destinations in the second. For the shuffle-exchange graph the corresponding route length is shown to be strictly longer than for the 2-way shuffle.**

***Index Terms*—Complexity, interconnection network, parallel computer, routing, shuffle graph.**

I. Introduction

We consider the problem of realizing routing requests in packet switching networks that are sparse in connections and yet have to support heavy parallel traffic. The relevance of this to the design of homogeneous highly parallel computers is well known [2], [5], [9], [10], [15]. The paradigmatic case is that of realizing arbitrary permutations. There is one packet initially at every node, each with a distinct, but otherwise arbitrary, destination address, and the task is to route every packet to its destination simultaneously. If the network has N nodes we consider it to be sparse if the number of connections at each node is a constant, or proportional to $\log N$. The aim is to have every packet arrive within $0(\log N)$ time under the assumptions that it takes unit time for a packet to traverse a connection, and only one packet can traverse any one connection at any time.

Routing algorithms can be distinguished as being either *oblivious* or *nonoblivious* depending on whether the path taken by a packet depends only on its own source and destination, or whether it depends on the rest of the permutation specification also. If the algorithm is *deterministic* then obliviousness means that for each source-destination pair there is a unique route which any packet with that source-destination specification must take. If *randomization* is used by the algorithm then obliviousness means that for each source-destination pair there is a fixed probability distribution (independent of the rest of the permutation) that specifies for each path from the source to the destination the probability that that path will be taken.

An obvious and useful attribute of oblivious algorithms is that they are suited to fully distributed implementations. Classical routing methods in permutation networks are nonoblivious, for example, and no efficient distributed implementation of them is known [3], [5]–[7], [9]. Batcher's sorting networks [2] are exceptional in that while being nonoblivious they are fully distributed. Unfortunately they take $\Omega((\log N)^2)$ steps and are not robust, for example, when the permutation is only partial. Recently, Borodin and Hopcroft [4] have shown that if we insist on obliviousness then we have to pay a heavy price for determinism, namely a runtime of $\Omega(N^{1/2})$, rather than the desired $0(\log N)$, in the constant degree case.

A class of $0(\log N)$ time distributed routing algorithms was proposed in [14], [15]. Analytic proofs were given of this runtime for graphs of degree about $\log N$. Recently, using a new technique, Aleliunas [1] and Upfal [13] have obtained similar results for the more difficult case of constant degree graphs. The algorithms are oblivious but randomized. They consist essentially of two consecutive phases. The first phase sends each packet to a random node in the network, while the second then forwards it to the desired destination. This method of randomization is perhaps counter-intuitive because it makes packets travel up to twice the distance necessary. The question arises as to whether this is merely a technical device for obtaining analytic proofs, or an unavoidable price to pay for logarithmic runtime.

In this paper we show that if the network has near optimal diameter (e.g., the d-way shuffle has diameter $\log_d N$ and indegree and outdegree d) then any oblivious routing algorithm *either* takes time $\Omega(N^\epsilon)$ for some $\epsilon > 0$, *or* makes packets travel at least $2 \log_d N$ edges (i.e., twice the diameter). We deduce that the two phase strategy is optimal in this sense for oblivious algorithms.

Conclusions can be drawn also for such graphs as the shuffle-exchange [9], [10] that have nonoptimal diameter. Even if each edge is interpreted as being bidirectional, giving a directed graph of indegree and outdegree three, it can be shown that every oblivious algorithm *either* takes time $\Omega(N^\epsilon)$ *or* makes packets travel at lease $2.8 \log_2 N$ edge lengths. This makes it clearly inferior to the 2-way shuffle.

II. Preliminaries

Let $G = (V, E,)$ be a directed graph where $V = \{1, 2, \cdots, N\}$ is the set of vertices and $E \subseteq V \times V$ the set of edges. Let R_{ij} be the set of directed paths (possibly with cycles) starting at vertex i and finishing at vertex j. If R is one such path then $\lambda(R)$ denotes its length, the number of edge traversals, in it. An oblivious randomized routing algorithm associates with each pair $(i, j) \in V \times V$ a random variable X_{ij} that takes values in R_{ij}. For a path R, $\Pr(X_{ij} = R)$ is the probability that the algorithm sends a packet originating at i and destined for j via the path R. Clearly the sum of $\Pr(X_{ij} = R)$ over all $R \in R_{ij}$ is unity. We define the *maximum route length* of the algorithm to be

$$\max_{\substack{i,j \in V \\ R \in R_{ij}}} \{\lambda(R) \mid \Pr(X_{ij} = R) > 0\}.$$

Similarly, the *mean route length* is M/N^2 where

$$M = \sum_{\substack{i,j \in V \\ R \in R_{ij}}} (\lambda(R) \cdot \Pr(X_{ij} = R)).$$

We suppose that the indegree and outdegree of every vertex in G

Manuscript received April 2, 1982; revised August 3, 1982.
The author is with the Aiken Computation Laboratory, Harvard University, Cambridge, MA 02138.

0018-9340/83/0900-0861$01.00

than each individual programs. Each trace in a program mix will have an equal weight, regardless of the total number of instructions in the programs when they are aggregated to obtain a *normalized* statistic, say sequence length, for the mix. Thus, each program is regarded as a valid sample of programs of its class. The maximum length of a path was limited to 64 instructions for programming reasons.

A. Paths

On the average, each trace in the FX/S mix consists of 1.2 thousand unique paths, each of which was repeated 616 times, totaling 720 thousand paths. An instruction on the System/370 occupies two, four, or six bytes of memory depending on the instruction format. The path length in bytes is defined to be the total number of bytes which instructions in a path occupy. The path length in bytes will be presented in parentheses in the sequel as well as the path length in instructions. 90 percent of all paths were less than or equal to 15 instructions (56 bytes) with a mean of 6.7 instructions (23.8 bytes). Each of the CX mix had about a thousand unique paths, each of which was repeated 185 times amounting to 5.8 million paths in total. Paths of the CX mix were much shorter than those of the FX/S mix; 90 percent of all paths are shorter than or equal to 9 instructions (38 bytes) with the mean of 4.3 instructions (17.3 bytes). Note that these data can be used as a guideline when the size of an instruction buffer is to be selected.

B. Sequences

There are about 150 nonprivileged instructions which a program in problem state can issue. Of those only about ten instructions (branch instructions, supervisor call instruction, and execute instruction) can terminate a sequence when it does not reach the maximum length of 64 instructions. Even so, there still exist 1,400 ($=140 \times 10$), 196,000 ($=140 \times 140 \times 10$), and 27 440 000 ($=140 \times 140 \times 140 \times 10$) possible distinct sequences of two, three, and four instructions, respectively. However, only a small subset of possible distinct sequences are actually used. Fig. 1 shows the total number of distinct sequences in a mix as a function of the sequence length SL. It reaches a maximum of 272 when SL = 4 for the FX/S mix, 257 when SL = 4 for the CX mix. Then, it decreases gradually as sequences become longer. More than half of all sequences are shorter than 8 and 6 instructions for the FX/S and CX mixes, respectively.

An interesting and practically useful question is how many distinct instructions in the distinct sequences or how many distinct sequences themselves are needed to cover, say 95 percent of all instructions. To answer this question, distinct sequences were sorted in descending order of the number of instructions they cover. The cumulative distributions are shown in Fig. 2. Cumulative distributions of sequences sorted by the frequency of occurrences are also shown in the same figure. We immediately notice that the distributions are quite skewed and that a small subset of distinct sequences are responsible for most instruction execution. More than 95 percent of all instructions and sequences are respectively covered by 728 (24 percent) and 602 (20 percent) distinct sequences for the FX/S mix, and 348 (19 percent) and 233 (13 percent) for the CX mix. Since the impact of pipeline delays on performance depends on sequences and their execution frequencies, simulation costs can be significantly reduced by using (as input) relatively few sequences; these may adequately represent the entire instruction execution.

Paths and sequences have been studied for each program mix as a whole, assuming that programs in a mix are similar to each other. Here, we will investigate the similarity within a mix quantitatively. Let R be the ratio of the total number of distinct sequences in a program mix to the sum of the total numbers of distinct sequences in a program over the mix, n be the number of programs in a program mix, and r be the ratio of the distinct sequences common to all programs in the mix to the total number of distinct sequences in the mix. If we assume that r is the same for all programs in a program mix, and that each program has the same number of distinct sequences, then the equation $[n(1-r)+r]/n = R$ should hold, which yields

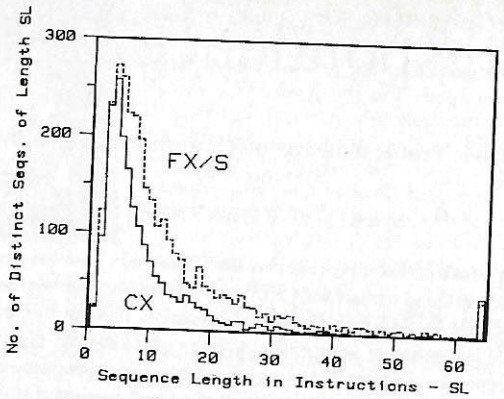

Fig. 1. Distribution of distinct sequences.

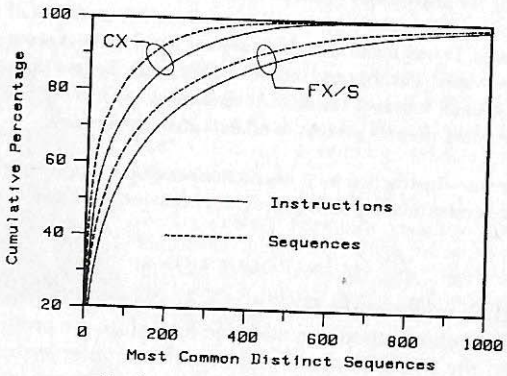

Fig. 2. Most common sequences.

$R)/(n-1)$. Since R was computed to be about 0.5 and $n = 10$ for the FX/S and CX mixes, we estimate $r = 0.56$. Hence, we may say that roughly half of the distinct sequences in a program are common in a program mix.

IV. Concluding Remarks

The measurement results of traces of the scientific and commercial programs have shown that most paths are very short and that there are relatively few distinct paths in a program. There are far fewer distinct sequences than distinct paths. It was found that only a small subset of all distinct sequences are responsible for most instruction execution.

This shows that it is possible to obtain a very accurate simulation results of, say, pipelining by using a small subset of most common sequences as an input to a simulator. Since there are relatively few distinct sequences, a simulation program can still run considerably faster even when all distinct sequences are fed into the simulator than when the original trace is used. For example, a detailed pipeline simulator can run about five hundred times faster without losing much accuracy in such a case [11].

Acknowledgment

The author is very grateful to M. H. MacDougall for many helpful discussions. Thanks are also due to J. K. Howell and W. N. Yamaguchi for the instruction traces. Finally, the author would like to express his thanks to W. J. Harding, M. H. MacDougall, and R. Shuey whose comments made this paper clearer.

References

[1] A. B. Barak and M. Aharoni, "A study of machine-level software profile," *Software-Practice Experience*, vol. 8, pp. 131–136, Mar.–Apr. 1978.

[2] J. L. Elshoff, "An analysis of some commercial PL/I ..."

is bounded above by a constant d. For vertices i, j we define

$$d(i,j) = \min_{R \in R_{ij}} \{\lambda(R)\}.$$

Finally, we define the *diameter* function D for $q \geq 0$ to be $D(q) = \max\{D^+(q), D^-(q)\}$ where

$$D^+(q) = \max_i \{|\{j | d(i,j) \leq q\}|\}, \text{ and}$$

$$D^-(q) = \max_j \{|\{i | d(i,j) \leq q\}|\}.$$

For our results we need just one elementary fact about matrices.

Lemma: If A is an $m \times m$ matrix of real numbers such that

$$\sum_i \sum_j A_{ij} = X$$

then for some permutation σ on $\{1, \cdots, m\}$

$$\sum_i A_{i,\sigma(i)} \geq X/(2m-1).$$

Proof: Suppose that the largest element of A has indexes i_1, j_1. Let $\sigma(i_1) = j_1$ and let

$$X_1 = A_{i_1 j_1} + \sum_{j \neq j_1} A_{i_1 j} + \sum_{i \neq i_1} A_{i j_1} \leq (2m-1) A_{i_1 j_1}.$$

Then, the matrix obtained by removing row i_1 and column j_1 from A has elements that sum to $X - X_1 \geq X - (2m-1)A_{i_1 j_1}$.

We repeat this process $m-1$ times for the submatrices so constructed successively until σ is defined everywhere. We obtain that

$$0 = X - X_1 - X_2 - \cdots - X_m \geq X - (2m-1)\left(\sum_i A_{i,\sigma(i)}\right)$$

from which the result follows. □

III. Results

The lower bound result depends on the diameter function D of the graph and on no other property of it.

Theorem: If G is an N vertex directed graph with diameter function D then for any oblivious randomized routing algorithm that realizes arbitrary permutations the following hold.

a) If the maximum route length is $2t$ then for some permutation and some vertex k the expected number of packets passing through k is at least $(N - D(t))/2D(t)$.

b) If the mean route length is $2t$, then for some permutation and some vertex k the expected number of packets passing through k is at least

$$\max_{0<\epsilon<1} \left\{ \frac{(\epsilon/2)N - D(t(1+\epsilon))}{2D(t(1+\epsilon))} \right\}.$$

Proof: For the first bound let p_{ijk} be the probability that "a packet originating at i and destined for j passes through vertex k immediately after having traveled along exactly t edges." Clearly, for any fixed i,j if $d(i,j) \geq t$ then

$$\sum_k p_{ijk} = 1.$$

Hence,

$$\sum_i \sum_j \sum_k p_{ijk} \geq |\{(i,j) | d(i,j) \geq t\}| \geq N^2 - ND(t).$$

We can therefore pick a k such that

$$\sum_i \sum_j p_{ijk} \geq N - D(t). \quad (*)$$

Since at most $D(t)$ vertices are within distance t of k in either the

ments. Applying the lemma to P and using (*) gives that for some permutation σ

$$\sum p_{i,\sigma(i)} \geq (N - D(t))/(2D(t)).$$

Hence, for any permutation $\bar{\sigma}$ on $\{1, \cdots, N\}$ that extends σ

$$\sum p_{i,\sigma(i),k} \geq (N - D(t))/(2D(t)).$$

We conclude that when the routing algorithm realizes the permutation $\bar{\sigma}$ the expected number of packets travelling through k exceeds the above quantity.

To obtain the bound for the mean route length $r = 2t$ we note that by definition,

$$\sum_{i,j,R} \Pr(X_{ij} = R) = N^2, \quad \text{and}$$

$$\sum_{i,j,R} \lambda(R) \cdot \Pr(X_{ij} = R) = N^2 r. \quad (**)$$

Hence, for any ϵ such that $0 < \epsilon < 1$,

$$\sum_{i,j,R} \Pr(X_{ij} = R \text{ and } \lambda(R) < r(1+\epsilon)) \geq (\epsilon/2)N^2 \quad (***)$$

for otherwise

$$\sum_{i,j,R} \Pr(X_{ij} = R \text{ and } \lambda(R) \geq r(1+\epsilon)) \geq (1 - \epsilon/2)N^2$$

and then

$$\sum_{i,j,R} \lambda(R) \cdot \Pr(X_{ij} = R \text{ and } \lambda(R) \geq r(1+\epsilon)) \geq (1+\epsilon)(1-\epsilon/2)N^2 r > N^2 r$$

which would contradict (**).

Now, let p_{ijk} be the probability that "a packet originating at i and destined for j gets a route of length no more than $r(1+\epsilon)$ and passes through k immediately after having travelled along exactly $t(1+\epsilon)$ edges of this route." Since there are at most $ND(t(1+\epsilon))$ pairs (i,j) such that $d(i,j) < t(1+\epsilon)$, if we exclude these from the summation (***) we get

$$\sum_{i,j,R} \lambda(R) \cdot \Pr(X_{ij} = R \text{ and } \lambda(R) < r(1+\epsilon) \text{ and } d(i,j) \geq t(1+\epsilon)) \geq (\epsilon/2)N^2 - ND(t(1+\epsilon)).$$

By the definition of p_{ijk}, the left hand side of the above inequality equals

$$\sum_i \sum_j \sum_k p_{ijk}.$$

Hence, for at least one of the N possible choices of k

$$\sum_i \sum_j p_{ijk} \geq (\epsilon/2)N - D(t(1+\epsilon)).$$

Now it follows exactly as in the first half of the theorem that we can pick a permutation σ such that

$$\sum_i p_{i,\sigma(i),k} \geq [(\epsilon/2)N - D(t(1+\epsilon))]/2D(t(1+\epsilon)).$$

We conclude that when a routing algorithm with mean route length $r = 2t$ realizes this permutation then the expected number of packets passing through k is at least the above quantity. □

IV. Applications

The *d-way shuffle* [15] has indegree and outdegree d, $N = d^$ nodes, diameter $r = \log$ N and $D(q) \leq (dq+1 - 1)/(d-1) \leq$

$$\frac{\epsilon d^n/2 - d^{(1-\mu)n+1}}{2d^{(1-\mu)n+1}} = (d^{\mu n-1} - 1)/2$$

$$= \begin{cases} \Omega(N^{\mu}) \text{ if } \mu \text{ is a constant} \\ \Omega((\log N)^k) \text{ if } \mu = (k \log_d n)/n \end{cases}$$

Since at most d packets can leave a node simultaneously, these quantities are lower bounds on the expected runtime of the algorithm for the bad permutation.

Part b) of the Theorem implies that if the mean route length of an algorithm is $2(1-\mu)n$ then the expected traffic through some node will be as high as

$$\frac{\epsilon d^n/2 - d^{(1+\epsilon)(1-\mu)n+1}}{2d^{(1+\epsilon)(1-\mu)n+1}}$$

Choosing $\epsilon = \mu/2$ gives a lower bound of

$$(\mu/8)d^{(\mu/2+\mu^2/2)n-1} - 1/2 = \begin{cases} \Omega(N^{\mu/2}) \text{ if } \mu \text{ is a constant} \\ \Omega((\log N)^k) \text{ if } \mu = (2k \log_d n)/n \end{cases}$$

We conclude that the two-phase routing algorithm suggested in [5] is optimal in that it has maximum route length $2n$ and runtime provably $O(\log N)$ [1].

The *shuffle-exchange* graph was suggested by Stone [11] as being suitable for several natural algorithms. For realizing permutations, however, it is inferior to the 2-way shuffle. To make the comparison we regard the 2-way shuffle, in the standard way, as a directed graph with indegree and outdegree 2. To be as generous as possible to the shuffle exchange, which is normally defined as an undirected graph of degree three, we interpret all the edges to be bidirectional so that every node has indegree and outdegree three. The naive observation is that for $N = 2^n$ nodes the 2-way shuffle has diameter n, while the shuffle exchange has diameter $2n$. We have seen that a good algorithm for the former needs to have route length $2n$. What we can show here is that any good algorithm for the shuffle-exchange must have strictly longer route length, at least $2.8n$.

If the nodes of the shuffle-exchange are labeled with the binary representations of the numbers $0, 1, \cdots, N-1$ respectively then we can regard the three edges emanating from a node as a) "left shift on the binary representation," b) "right shift on the binary representation," and c) "exchange the least significant bit." Hence any, directed path can be characterized by a sequence of symbols from the alphabet $\{L, s, e\}$ where s denotes a shift, e denotes an exchange, R denotes that the shifts to follow, before the next L, are all right shifts, and L denotes that the shifts to follow, before the next R, are all left shifts. Typical such string is *RssesLsseseRse*. For a minimal path we can assume that e's never occur consecutively. Also, the reader can prove easily the following.

Lemma: For any pair of nodes in the shuffle-exchange graph there is a shortest path in which the shifts reverse directions at most once.

Hence, to compute the function $D(q)$ for the shuffle exchange we start by counting the number of sequences of S s's and E e's where $S + E = q$ and no two e's are consecutive. This is just $\binom{S+1}{E}$. Allowing for the shift indicators and summing gives

$$D(q) \leq \sum_{S=q/2}^{S=q} (2q^2) \binom{S+1}{q-S}.$$

To find an upper bound let $S = (1+\alpha)q/2$. Then, $q - S > (1-\alpha)q/2$. Using Stirling's approximation we get that the largest term grows as

$$\frac{(1+\alpha)^{(1+\alpha)q/2}}{(1-\alpha)^{(1-\alpha)q/2}(2\alpha)^{\alpha q}}$$

where α equals $1/\sqrt{5}$. This gives an upper bound on $D(q)$ of $2^{\beta q}$ for sufficiently large n, where $\beta = 0.71$. Hence, we can deduce from the Theorem a lower bound of $2n/\beta \simeq 2.8n$ on the maximal and mean route length of any oblivious $O(\log N)$ time algorithm.

ameter is so far from optimal that our technique yields no lower bounds. It is reasonable to conjecture that the bound does not hold and that efficient algorithms with route length about n rather than the $2n$ suggested in [14], [15] exist. The obvious candidate is the following. Fix a small $\epsilon > 0$. For each pair (i, j) of nodes guess a random k under the constraint that $d(i,k), d(k,j) \leq (n+\epsilon)/2$. The first phase now routes the packet from i to k, and the second from k to j.

References

[1] R. Aleliunas, "Randomized parallel communication," in *Proc. ACM Symp. Principles Distribut. Computing*, Ottawa, Canada, 1982, pp. 60–72.

[2] K. Batcher, "Sorting networks and their applications," in *Proc. AFIPS Spring Joint Comput. Conf.*, 1968, vol. 32, pp. 307–314.

[3] V. E. Benes, *Mathematical Theory of Connecting Networks and Telephone Traffic*. New York: Academic, 1965.

[4] A. Borodin and J. E. Hopcroft, "Routing, merging and sorting on parallel models of computation," in *Proc. 14th ACM Symp. Theory Comput.*, 1982.

[5] Z. Galil, and W. J. Paul, "An efficient general purpose parallel computer," in *Proc. 13th ACM Symp. Theory Comput.*, 1981 pp. 247–256.

[6] G. Lev, N. Pippenger, and L. G. Valiant, "A fast parallel algorithm for routing in permutation networks," *IEEE Trans. Comput.*, vol. C-30, pp. 93–100, Feb. 1981.

[7] D. Nassimi and S. Sahni, "Parallel algorithms to set up the Benes permutation networks," *IEEE Trans. Comput.*, vol. C-31, pp. 148–154, Feb. 1982.

[8] F. P. Preparata and J. Vuillemin, "The cube connected cycles," *Commun. Ass. Comput. Mach.*, vol. 24, pp. 300–310, 1981.

[9] J. T. Schwartz, "Ultracomputers," *ACM TOPLAS*, vol. 2, pp. 484–521, 1980.

[10] H. J. Siegel, "Interconnection networks for SIMD machines," *Computer*, pp. 57–65, June 1979.

[11] H. Stone, "Parallel processing with the perfect shuffle," *IEEE Trans. Comput.*, vol. C-20, pp. 263–271, Feb. 1971.

[12] H. Sullivan, T. R. Bashkow, and D. Klappholz, "A large scale homogeneous fully distributed parallel machine," in *Proc. 4th ACM Symp. Comput. Arch.*, 1977, pp. 118–127.

[13] E. Upfal, "Efficient schemes for parallel communication," in *Proc. ACM Symp. Principles Distribut. Computing*, Ottawa, Canada, 1982, pp. 55–59.

[14] L. G. Valiant, "A scheme for fast parallel communication," *SIAM J. Comput.*, 1981.

[15] L. G. Valiant and G. J. Brebner, "Universal schemes for parallel communication," in *Proc. 13th ACM Symp. Theory Comput.*, 1981, pp. 263–277.

On the Minimization of Wordwidth in the Control Memory of a Microprogrammed Digital Computer

CHAMARTY D.V.P. RAO AND NRIPENDRA N. BISWAS

***Abstract*—This paper describes a new method for reducing the wordwidth in the control memory of a microprogrammed digital computer. The method requires the computation of maximal compatibility classes (MCC's) of only a subset of subcommands unlike the existing methods in which MCCs of the entire set of subcommands are used. An algorithm has been given to obtain a near minimal solution by appropriate grouping of the MCC's. The necessary modification of the algorithm has also been presented to deal with large size microprograms.**

Manuscript received April 5, 1982; revised October 25, 1982 and December 29, 1982.

C. D.V.P. Rao is with the Department of Electronics and Communication Engineering, N.S. Engineering College, Jawaharlal Nehru Technological University, Hyderabad, India, currently on deputation at the Department of Electrical Communication Engineering, Indian Institute of Science, Bangalore, India.

N. N. Biswas is with the Department of Electrical Communication Engi-