University of Alberta

An $O(N^{1.695})$ Permutation Routing Algorithm on Augmented Data Manipulators

by

E.S. Elmallah and Chin-Hung Lam

Technical Report TR 97–07
September 1997

DEPARTMENT OF COMPUTING SCIENCE
The University of Alberta
Edmonton, Alberta, Canada

An $O(N^{1.695})$ Permutation Routing Algorithm on Augmented Data Manipulators

Ehab S. Elmallah and Chin-Hung Lam
Department of Computing Science
University of Alberta, Edmonton, Alberta

Abstract

This paper shows an $O(N^{1.695})$ time algorithm for routing any arbitrary admissible permutation on Augmented Data Manipulator networks.

1 Introduction

The class of *Augmented Data Manipulators* (ADMs) is defined as follows: for $n \geq 1$, an ADM of size $N = 2^n$, has N inputs and outputs. The switch fabric has $n+1$ stages of switches, labeled from left to right, V_i for $i = n, n-1, \cdots, 0$ (e.g., see Figure 1.a). Each stage has N switches, and each switch has a unique address between 0 and $N-1$. Switches in the input stage V_n are all 1-to-3 multiplexers. Similarly, switches in the output stage are all 3-to-1 demultiplexers. Switches in all possible intermediate stages (i.e., stages with indices $i = n-1, n-2, \cdots, 1$) are 3×3 switches, in which at most one input-output connection can be active at any time. Links are organized into n stages, labeled from left to right L_i for $i = n-1, \ldots, 0$. Stage L_i connects switch $x \in V_{i+1}$ to switches $\{x, x+2^i, x-2^i\} \in V_i$ (mod N). A special case arises in the leftmost stage L_{n-1}, where $x + 2^{n-1} = x - 2^{n-1}$ (mod N), and hence each input switch has access to exactly two distinct switches in V_{n-1}.

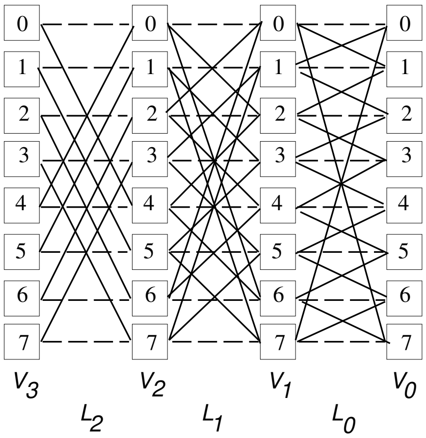
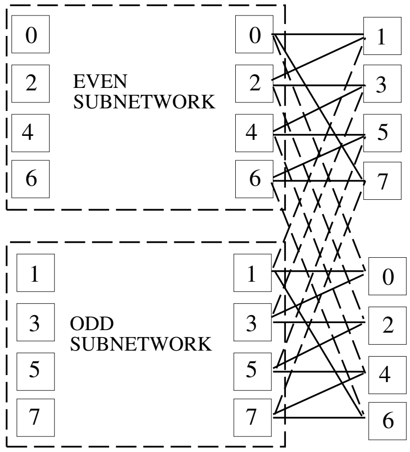


Figure 1.a An ADM with $N = 8$ inputs



1.b The even/odd recursive decomposition

The topology of the ADM networks has been originally introduced in [3], and subsequently explored in [14]. Several combinatorial and enumeration results on routing permutations in ADM networks appear in the literature. In one direction, for any $N = 2^n$, it is known that the set of ADM admissible permutations is a proper superset of its counterpart in the *generalized cube network* [13] (or, equivalently, the *Omega* network [4] under a suitable relabeling of the terminals) (see, for example, [15] and the references therein). The related problem of counting the number of ADM admissible permutations has been initiated in [10] and [1]. Later, Leland [7] (see also [8]) has shown a set of recurrence relations for the exact number $P(N)$ of ADM admissible permutations. The results shown in [6] give another distinct set of exact recurrence relations for $P(N)$. The following lower and upper bounds on $P(N)$ have also been shown in [7]: $P(4) = 24$, $P(8) = 26496$, and for $N \geq 16$,

$$\phi^{N(\lg N-2)} 23.7^{\frac{N}{4}} \leq P(N) \leq \phi^{N(\lg N-3+\frac{1}{192})} P(8)^{\frac{N}{8}},$$

where $\phi = \frac{1+\sqrt{5}}{2}$ is the golden ratio.

In a second direction, several tag-based control algorithms have been devised for routing useful permutations in ADM networks (see, for example, [9], [8], [6], and [12]). Some algorithms in this class are designed to exploit the existing plurality of input-output paths to handle some possible fault patterns in the network. Viewed as universal routing algorithms, however, tag-based routing is typically very efficient but limited in power: any such algorithm realizes only a strict subset of the set of all ADM admissible permutations. To remedy the situation, several backtrack search algorithms that can route any arbitrary admissible permutation has been discussed in [7], [6], and [12]. None of the above formulations, however, provide a time complexity analysis that gives a worst case bound on the time required to decide whether a given permutation is an ADM admissible permutation. In fact, [7] hints that such a universal backtrack search algorithm is expected to be extremely inefficient.

The main result of this paper (Theorem 1 in Section 4) fills the above apparent gap in the literature by showing an $O(N^{1.695})$ deterministic routing algorithm to solve the problem. The algorithm is similar to that given in [6] and [12] in that all three algorithms exploit the basic combinatorial properties summarized in the next section.

2 Background Results

The correctness and timing of the routing algorithm hinges on the fundamental properties of ADM networks summarized in lemmas 1 and 2 below. The result in Lemma 1 has been first mentioned in [3], and used subsequently in [1, 6, 7]. Properties equivalent to Lemma 2 have also been used in [1, 6, 7]. For our purpose, however, we use graph-theoretic concepts to present and prove the properties. Our objective is to expose the combinatorial properties of the network while minimizing the use of bit-level notation that is typically associated with tag-based control algorithms.

To start, we need the following definition: given a permutation π on a set of n-bit long integers, we say that π is *all-pairs parity switching* with respect to the ith bit $(0 \leq i \leq n-1)$, written $\pi \in \mathrm{PS}(i)$, if for any possible pair $(x, \pi(x))$ the ith bit of x is the complement of its counterpart in $\pi(x)$. Otherwise (i.e., if for some pair $(x, \pi(x))$ the above property fails), we write $x \notin \mathrm{PS}(i)$.

Lemma 1:(the even/odd decomposition) Let G be an ADM of size N then $G \setminus V_0$ (the subgraph obtained by deleting the right most stage V_0 from G) is a disjoint union of two ADM networks, denoted G_{even} and G_{odd}, each of size $N/2$, where G_{even} (G_{odd}) has all the even (respectively, odd) numbered switches of $G \setminus V_0$.

Proof:

To show that $G \setminus V_0$ is a disjoint union of at least two components, note that only links in L_0 (more specifically, the $\pm 2^0$ connections) join two switches with different even/odd parities. To see that G_{even} (or G_{odd}) induces an ADM, note that stripping the least significant bit from each address in the subnetwork gives a properly labeled ADM of size $N/2$. ■

Call the two subnetworks (G_{even} and G_{odd}) defined in Lemma 1 the *rank*-1 subnetworks of G (or, the (1)-subnetworks of G, for short). For $i \geq 2$, applying the recursive decomposition to each of the available $(i-1)$-subnetworks, gives rise to 2^i ADM subnetworks, each of size $\frac{N}{2^i}$; call such subnetworks the rank-i (or, the (i)-subnetworks) of G. It is also convenient, to view G itself as a rank-0 subnetwork. By inspection, each (i)-subnetwork is composed of switches in $V_n \bigcup \cdots \bigcup V_i$, whose addresses (written in binary) share a unique pattern in their least significant i bits.

Lemma 2: Let π be a permutation on $\{0, \cdots, N-1\}$ that is passable through an ADM network G of size N then

1. if $\pi \notin \mathrm{PS}(0)$ then there exists a unique setting of the switches in V_1 (equivalently, a perfect matching in L_0) to accept π.

2. Else, if $\pi \in \mathrm{PS}(0)$ then there exist at most 4 possible settings of the switches in V_1 to accept π.

Proof:

To prove the lemma, it suffices to analyze the possible settings of the output switches in either G_{even} or G_{odd}; the lemma follows by applying the argument to both subnetworks. Without loss of generality, we focus on G_{even}. For our purpose, let $X \subset V_1$ be the subset of the even numbered switches of V_1, and let $Y \subset V_0$ be the subset of the odd numbered switches (V_0 is G's output stage). Note that the subnetwork induced by $X \bigcup Y$ is a cycle on N switches whose links are the union of two perfect matchings corresponding to connections with the $+2^0$ and the -2^0 offsets, respectively.

Routing π in G entails setting the switches in V_1 so as to induce a perfect matching in L_0. For any such collection of valid settings (as may be specified in the context below), denote by $X^=$, and $Y^=$ the possible subsets of X and Y, respectively, that use the identity connections. Note that the equality $|X^=| = |Y^=|$ holds true since π is of the same size as G. We are now ready to prove parts 1 and 2 of the lemma.

1. If $\pi \notin \mathrm{PS}(0)$ then, by definition, $|X^=| = |Y^=| \geq 1$ in any configuration that admits π. In fact, the sets $X^=$ and $Y^=$ are uniquely defined by the pairs $(x, \pi(x))$ that agrees on the least significant bit of their binary representations. Now, none of the switches in $X^=$ and $Y^=$ uses the links in the cycle induced by $X \bigcup Y$. Consequently, deleting $X^=$ and $Y^=$ from $X \bigcup Y$ splits the cycle into one (or more) path(s). By assumption π is admissible in G, and hence any such path has an even number of switches, and a unique possible perfect matching, as required.

2. If $\pi \in \mathrm{PS}(0)$ then, by definition, $|X^=| = |Y^=| = 0$ in any configuration that admits π. Consequently, there are two possible settings of X corresponding to the two possible perfect matchings in the cycle induced by $X \bigcup Y$. Applying a similar argument to G_{odd} gives a total of 4 possible settings of switches in V_1.

■

3 The Algorithm

In this section we show a polynomial time algorithm for solving the decision problem mentioned above. The routing algorithm (function `dfsADM` below) is a simple divide-and-conquer algorithm implemented as depth-first search. The algorithm uses a global matrix ADM in which element $ADM[i,j]$ stores a setting of switch j in the ith stage of switches. Function `dfsADM`(i,π) takes as input two parameters: an index i, $0 \le i \le n$, of some stage V_i of switches, and a permutation π on a subset of $\{0,\cdots,N-1\}$. For $i=0$, π is the input permutation specified in an instance of the decision problem. For $i \ge 1$, an important invariant of the algorithm is that parameter π corresponds to a proper subset of $\{0,\cdots,N-1\}$ of size $\frac{N}{2^i}$ in which all integers (written in binary) share a unique suffix of the i least significant bits. In any of the above cases, if $\pi(x)$ is defined for some integer x, then we interpret that $x \in V_{n-1}$ is an input switch and $\pi(x) \in V_i$ is some switch in the ith stage of switches. The recursive decomposition of ADM networks ensures that π corresponds to a permutation over the set of inputs of exactly one of the 2^i possible (i)-subnetworks, denoted G_i in function `dfsADM`.

If π is admissible then `dfsADM` returns zero and sets the appropriate elements in matrix ADM to correspond to a non-conflicting set of routes; otherwise an error value of -1 is returned. More specifically, for $i=n$, V_i is the leftmost stage, and step 1 of the function terminates the recursion successfully. For $i \le n-1$, the algorithm tries to route π through G_i by first splitting π into two subpermutations: π^{even} and π^{odd} to be routed through

1. the last stage of links in G_i, and

2. the even and the odd subnetworks of G_i, respectively;

the splitting is done in step 2. Step 3 iterates two times, once for $\pi^{hlaf} = \pi^{even}$, and another for $\pi^{half} = \pi^{odd}$. In each iteration we denote by $G_{i+1} \subset G_i$ (read G_{i+1} is a proper subgraph of G_i) the unique $(i+1)$-subnetwork through which π^{half} should be routed. Based on Lemma 2, the algorithm subsequently tries to compute two objects: (1) a valid setting of the output switches of G_{i+1}, and (2) an induced permutation (denoted, $\pi^{=}_{i+1}$, π^{+}_{i+1}, or $\pi^{-}_{i=1}$), that together suffice to pass π^{half} in G_i. Recursion is then used to route each of the induced permutations.

```
function dfsADM (i, π) {
#    input:   an index i, 0 ≤ i ≤ n, of a stage of switches, and a permutation
#             π on a subset of {0,···,N−1}, as defined above.  Throughout the
#             presentation, let G_i be the unique (i)-subnetwork for which
#             π is an input-output mapping (cf., the above discussion).
#    output:  an error return:  error = 0 only if G_i admits π.
```

1. `if` $(i == n)$ `return` 0

2. `for (`x `in` π`) {`

    `set` $\pi^{even}(x) = \pi(x)$ `only if the` i`th bit in` x`'s binary`
    `representation is zero; else, set` $\pi^{odd}(x) = \pi(x)$

  `}`

3. `for (`π^{half} `in` $\{\pi^{even}, \pi^{odd}\}$`) {`

    `# Let` $G_{i+1} \subset G_i$ `be the unique` $(i+1)$`-subnetwork for which`
    `#` π^{half} `is an input-output mapping;`
    `# denote by` $X \subset V_{i+1}$ `the set of` G_{i+1}`'s output switches;`
    `# denote by` $Y \subset V_i$ `the set of` G_i`'s output switches;`
    `# denote by` $Y^{idle} \subseteq Y$ `the subset of switches in` Y `that are`
    `#` ***available*** `to connections from` X`;`

    `3.1 if (`$\pi \notin \mathrm{PS}(i)$`) {`

        `# By Lemma 2.1, there is a unique subset` $X^{=} \subseteq X$ `of switches`
        `# that must be set to the identity connection:`
        `3.1.1 find the unique subset` $X^{=}$`;`
        `3.1.2 find whether there exists a conflict-free setting of the`
            `switches in` $X \backslash X^{=}$ `to access switches in` Y`;`
        `3.1.2 if no such setting exists then return` (-1)`;`
        `3.1.3 else, save the obtained setting in the global matrix` ADM`;`
        `3.1.4 mark the` Y `switches involved in the obtained settings`
            `as` ***unavailable*** `for subsequent allocations;`
        `3.1.5 given the settings in 3.1.3, let` $\pi^{=}_{i+1}$ `be the permutation that`
            `must be passed through` G_{i+1} `so as to admit` π^{half}`;`
        `3.1.6` $error =$ `dfsADM(`$i+1$`,` $\pi^{=}_{i+1}$`)`

      `}`

    `3.2 else if (`$\pi^{half} \in \mathrm{PS}(i)$`) {`

        `# By Lemma 2.2, if` π^{half} `is admissible in` G_i `then all switches`
        `# in` X `are set to either the` $+2^i$ `or the` -2^i `connections;`
        `3.2.1 set` X `to use the` $+2^i$ `connections; save the settings`
            `in matrix` ADM`; let` π^{+}_{i+1} `be the permutation that should`
            `be passed through` G_{i+1} `in order to admit` π^{half}`;`
        `3.2.2` $error =$ `dfsADM(`$i+1$`,` π^{+}_{i+1}`)`
        `3.2.3 if (`$error \neq 0$`) {`

            - `repeat step 3.2.1 for the` -2^i `connections;`
            - $error =$ `dfsADM(`$i+1$`,` π^{-}_{i+1}`)`

          `}`

```
            }
        }
    4.  return error
}
```

Correctness of `dfsADM` follows by Lemma 2 since it exhausts all possible ways of passing an admissible permutation π. Assuming a uniform cost model (see, for example, [2]), the following timing result is then a direct consequence of the structure of the algorithm.

Lemma 3: Let $T(N)$ be the time required to test whether a given N-element permutation is admissible through an ADM of equal size, then for some constant c, $T(N)$ satisfies

$$T(N) \leq cN + 4T(\frac{N}{2}), \qquad T(1) \leq c,$$

That is, $T(N) \in O(N^2)$.

The above algorithm has been implemented. For $N = 8$, all of the ADM admissible permutations (26,496) are verified. For $N = 16$ and 32, there are approximately 1.549×10^{12} and 1.169×10^{31} ADM admissible permutations to verify, respectively. In these two latter cases, an exhaustive verification is prohibitive; we choose to measure the percentage of the admissible permutations on a random sample of 10^6 permutations; the results are 7.689% for $N = 16$ (the recurrence relations in [7] give $\approx$ 7.404%), and .004562% for $N = 32$ (versus $\approx$ .004443% from [7]). For $N \geq 64$, the percentage of the admissible permutations in a random sample of size 10^6 is so small that no reliable figure can be given.

4 A Tighter Bound

An upper bound on T(n) is given by:

$$T(n) \leq \sum_{i=1}^{n} 2^i \left(\frac{cN}{2^i}\right) F_i = cN \sum_{i=1}^{n} F_i$$

where, the summation is of the computations done in a single invocation of function `dfsADM` (i.e., not involving the recursive calls) over all (i)-subnetworks, for $i = 1, 2, \cdots, n$. Specifically, 2^i is the number of (i)-subnetworks in the recursive decomposition, $\left(\frac{cN}{2^i}\right)$ is the cost incurred by the nonrecursive part of `dfsADM` when the formal parameter π has $\frac{N}{2^i}$ integers, and F_i is an upper bound on the total number of induced permutations that the algorithm tries to route through any particular (i)-subnetwork.

For example, for stage $i = 1$, there are two (1)-subnetworks, each of size $\frac{N}{2}$. The maximum number of induced permutations generated and tested through each (1)-subnetwork results when $\pi \in \mathrm{PS}(0)$; the corresponding upper bound is $F_1 = 2$. In general, for any $i \geq 1$, setting $F_i = 2^i$ gives the $T(N) \in O(N^2)$ bound mentioned above. The main result of this section is Theorem 1 below; the result is obtained by using the improved bound on F_i shown in part 3 of Lemma 5 (as can be readily verified).

Theorem 1: $T(N) \in O(N^{1+\lg\phi})$, where $\phi = \frac{1+\sqrt{5}}{2}$ is the golden ratio ($\lg\phi < .695$).

The proof of Lemma 5 has two basic ingredients: Lemma 4 and the recurrence relations in Lemma 5. To introduce the basic ideas we need the following notation. For any possible stage $i \geq 2$, let $G_i \subset G_{i-1} \subset G_{i-2}$ be three subnetworks of ranks i, $(i-1)$, and $(i-2)$, respectively. If $i = 1$ is implied by the context, then only G_i and G_{i-1} are defined.

At some point, the algorithm tries to route a given permutation, say π_{i-1} through G_{i-1}. If $\pi_{i-1} \notin \mathrm{PS}(i-1)$ then the algorithm sets at least one of G_i's output switches to the identity connection, and computes an induced permutation, denoted $\pi_i^=$ thereafter. Permutation $\pi_i^=$ has $\frac{N}{2^i}$ elements, and it is tested for admissibility in G_i. We henceforth refer to permutations obtained in this way as $(^=_i)$-induced permutations.

If, on the other hand, $\pi_{i-1} \in \mathrm{PS}(i-1)$ then the algorithm tries out two possible settings of the output switches of G_i (the $+2^{i-1}$ and the -2^{i-1} connections), and computes the two induced permutations π_i^+ and π_i^-, respectively. We call the resulting permutations (^+_i)- and (^-_i)-induced permutations. We are now ready to present the first ingredient:

Lemma 4: For any all-pairs parity-switching permutation $\pi_{i-1} \in \mathrm{PS}(i-1)$, let π_i^+ and π_i^- be as defined above, then at least one of π_i^+ and π_i^- is not in $\mathrm{PS}(i)$

Proof:

For any input x in G_i, $\pi_i^+(x) = \pi_{i-1}(x) - 2^{i-1}$, and $\pi_i^-(x) = \pi_{i-1}(x) + 2^{i-1}$. Hence, the ith bit in the binary representation of $\pi_i^+(x)$ is the complement of its counterpart in $\pi_i^-(x)$. Consequently, the ith bit in the binary representation of x agrees with its counterpart in either $\pi_i^+(x)$ or $\pi_i^-(x)$. That is, either $\pi_i^+(x) \notin \mathrm{PS}(i)$, or $\pi_i^+(x) \notin \mathrm{PS}(i)$, as required. ■

We next incorporate the above observation in a counting argument. First, write F_i as $F_i^= + F_i^+ + F_i^-$, where $F_i^=$, F_i^+, and F_i^- are upper bounds on the number of $(^=_i)$-, (^+_i)-, and (^-_i)-induced permutations generated for routing through G_i. To simplify the counting argument, we use the following charging rules:

1. If $\pi_{i-1} \notin \mathrm{PS}(i-1)$ then the induced permutation $\pi_i^=$ is always assumed to be in $\mathrm{PS}(i)$ (a worst case scenario).

2. Else, if $\pi_{i-1} \in \mathrm{PS}(i-1)$, then, by Lemma 4, at least one of the induced permutations

π_i^+ and π_i^- is not in PS(i). Without loss of generality, we may assume that π_i^- is always not in PS(i). The other permutation, π_i^+ will always be assumed in PS(i) (the worst case).

Lemma 5: The following hold in any ADM network:

1. $F_0^+ = F_0^- = 0$, and $F_1^+ = F_1^- = 1$.
2. For any stage $i \geq 2$: $F_i^- = F_i^+$, $F_i^= = F_{i-1}^-$, and $F_i^+ = F_{i-1}^+ + F_{i-1}^=$.
3. For any stage $i \geq 1$: $F_i < 3(\frac{\phi^i}{\sqrt{5}} + 1)$, where $\phi = \frac{1+\sqrt{5}}{2}$ is the golden ratio.

Proof:

1. For stage $i = 0$ the initial conditions $F_0^+ = F_0^- = 0$ are trivially set. For stage $i = 1$, the worst case occurs if $\pi \in$ PS(0), and hence $F_1^+ = F_1^- = 1$.
2. For any stage $i \geq 1$, the equality $F_i^- = F_i^+$ holds by the structure of the algorithm.

   For $i \geq 2$, to see that $F_i^= = F_{i-1}^-$, note that if $\pi_{i-2} \in$ PS($i-2$), then applying rule (2) to π_{i-2} shows that the induced permutation π_{i-1}^- (at level $i-1$) is not in PS($i-1$). Consequently π_{i-1}^- induces a unique permutation $\pi_i^=$ that contributes to the count $F_i^=$. Moreover, given the charging rules (1) and (2) above, this is the only scenario under which an $(^=_i)$−induced permutation is generated.

   To see the third identity: $F_i^+ = F_{i-1}^+ + F_{i-1}^=$, for $i \geq 2$, note that every $(^+_{i-1})$-induced permutation π_{i-1}^+ that contributes to F_{i-1}^+ is assumed to be in PS($i-1$) by rule (2); hence, π_{i-1}^+ gives rise to an (^+_i)-induced permutation. Likewise, each $(^=_{i-1})$-induced permutation contributing to $F_{i-1}^=$ is assumed to be in PS($i-1$), and hence, it gives rise to an (^+_i)-induced permutation.
3. From (1) and (2) above, it follows that $F_i^+ = F_i^- =$ the ith Fibonacci number. In addition, $F_i^=$ is the $(i-1)$−Fibonacci number. Hence, $F_i^+ = \frac{\phi^i}{\sqrt{5}}$ (rounded to the nearest integer). The statement $F_i < 3(\frac{\phi^i}{\sqrt{5}}+1)$ then follows since $F_i = F_i^= + F_i^+ + F_i^-$.

■

5 Concluding Remarks

A fundamental problem in switching networks is to devise an efficient routing algorithm that captures the full power of a given network. This paper contributes to the above direction be showing an $O(N^{1.695})$ (worst case) time algorithm for routing admissible permutations on ADM networks. Asymptotically, the running time grows at a rate that is comparable to the growth rate of the number of switches ($N \lg N$) that must be set to route any such permutation. Hence, the algorithm is asymptotically near optimal. Currently, the complexity of the above problem on the closely related class of Gamma networks [11, 16, 5], appears to be open.

Acknowledgment

This research is supported by NSERC Canada under grant number OGP 36899.

References

[1] G. Adams III and H. Siegel, *On the number of permutations performable by the augmented data manipulator network*, IEEE Transactions on Computers, C-31 (1982), pp. 270–277.

[2] A. Aho, J. Hopcroft, and J. Ullman, *The Design and analysis of Computer Algorithms*, Addison-Wesley, 1974.

[3] T. Feng, *Data manipulating functions in parallel processors and their implementations*, IEEE Transactions on Computers, C-23 (1974), pp. 309–318.

[4] D. Lawrie, *Access and alignment of data in an array processor*, IEEE Transactions on Computers, C-24 (1975), pp. 1145–1155.

[5] K. Lee and W. Hegazy, *The extra stage gamma network*, IEEE Transactions on Computers, C-37 (1988), pp. 1445–1450.

[6] K. Lee and D. Lee, *On the augmented data manipulator network in SIMD environments*, IEEE Transactions on Computers, C-37 (1988), pp. 574–584.

[7] M.-D. P. Leland, *Properties and Comparison of Multistage Interconnection Networks for SIMD Machines*, PhD thesis, Department of Computer Sciences, University of Wisconsin - Madison, 1983.

[8] ——, *On the power of the augmented data manipulator network*, in Proceedings of the International Conference on Parallel Processing, August 1985, pp. 74–78.

[9] R. McMillen and H. J. Siegel, *Routing schemes for the augmented data manipulator network in an MIMD system*, IEEE Transactions on Computers, C-31 (1982), pp. 1202–1214.

[10] M. O'Donnell and C. Smith, *A combinatorial problem concerning processor networks*, IEEE Transactions on Computers, C-31 (1982), pp. 163–164.

[11] D. Parker and C. S. Raghavendra, *The gamma network*, IEEE Transactions on Computers, C-33 (1984), pp. 367–373.

[12] D. Rau, J. Fortes, and H. J. Siegel, *Destination tag routing techniques based on a state model for the IADM network*, IEEE Transactions on Computers, C-41 (1992), pp. 274–285.

[13] H. Siegel, W. Nation, C. Kruskal, and L. Napolitano, *Using the multistage cube network topology in parallel supercomputers*, Proceedings of the IEEE, 77 (1989), pp. 1932–1953.

[14] H. Siegel and S. Smith, *Study of multistage SIMD interconnection networks*, in Proceedings of the Fifth Annual Symposium on Computer Architecture, April 1978, pp. 223–229.

[15] H. J. Siegel, *Interconnection Networks for Large-Scale Parallel Processing*, McGraw-Hill Series in Computer Organization and Architecture, 1990.

[16] A. Varma and C. Raghavendra, *On permutations passable by the gamma network*, Journal of Parallel and Distributed Computing, 3 (1986), pp. 72–91.