JOURNAL OF PARALLEL AND DISTRIBUTED COMPUTING 21, 124–139 (1994)

Multiple Quadratic Forms: A Case Study in the Design of Data-Parallel Algorithms

MU-CHENG WANG,*[,1,2] WAYNE G. NATION,†[,1,3] JAMES B. ARMSTRONG,‡[,1,4] HOWARD JAY SIEGEL,‡[,1,5] SHIN DUG KIM,§[,1,6] MARK A. NICHOLS,¶[,1,7] AND MICHAEL GHERRITY**[,1,8]

*Computer Science Department, Queens College, The City University of New York, 65-30 Kissena Boulevard, Flushing, New York 11367; †IBM AS/400 Division, 3605 Highway 52 NW, Rochester, Minnesota 55901; ‡Parallel Processing Laboratory, School of Electrical Engineering, Purdue University, West Lafayette, Indiana 47907-1285; §Computer Engineering, KwangWoon University, Seoul, Korea; ¶NCR Corporation, 16550 W. Bernardo Drive, San Diego, California 92127-1806; and **Superconcurrency Research Team, Naval Ocean Systems Center, Code 421, San Diego, California 92152-5000

Data-parallel implementations of the computationally intensive task of solving multiple quadratic forms (MQFs) have been examined. Coupled and uncoupled parallel methods are investigated, where coupling relates to the degree of interaction among the processors. Also, the impact of partitioning a large MQF problem into smaller non-interacting subtasks is studied. Trade-offs among the implementations for various data-size/machine-size ratios are categorized in terms of complex arithmetic operation counts, communication overhead, and memory storage requirements. Furthermore, the impact on performance of the mode of parallelism used is considered, specifically, SIMD versus MIMD versus SIMD/MIMD mixed-mode. From the complexity analyses, it is shown that none of the algorithms presented in this paper is best for all data-size/machine-size ratios. Thus, to achieve scalability (i.e., good performance as the number of processors available in a machine increases), instead of using a single algorithm, the approach discussed is to have a set of algorithms from which the most appropriate algorithm or combination of algorithms is selected based on the ratio calculated from the scaled machine size. The analytical results have been verified by experiments on the MasPar MP-1 (SIMD), nCUBE 2 (MIMD), and PASM (mixed-mode) prototype. © 1994 Academic Press, Inc.

1. INTRODUCTION

The real-time processing of computationally intensive tasks often requires a parallel implementation. There may be several parallel implementations that can satisfactorily perform the task. As the size of the problem, the memory space constraints, the execution time constraints, and the machine architecture are changed, the parallel programmer must reevaluate which approach is the best [47]. An algorithm is regarded as "scalable" if it continues to perform effectively the task for which it was designed as the number of processors increases [15]. Here, the term *scalable* is applied to a set of algorithms. In this sense, as the number of processors is varied, the goal is to select the algorithm or combination of algorithms in the set that most effectively performs the task.

Several data-parallel algorithms are developed and analyzed in this research for computing the multiple quadratic forms (*MQF*s) that are part of an adaptive beamformer calculation with minimum variance distortionless response (*MVDR*) [29]. The implementations of the MQF problem for various data-size/machine-size ratios are evaluated in terms of the number of complex arithmetic operations, communication overhead, and memory storage requirements. Both coupled and uncoupled parallel methods are investigated, where *coupling* relates to the degree of interaction of the processors. The impact on performance of the mode of parallelism used is considered, specifically, SIMD versus MIMD versus SIMD/MIMD mixed-mode. Then, the effect of partitioning a large MQF problem into smaller noninteracting subtasks is studied. The analytical results show that none of the algorithms presented in this paper is best for all data-size/machine-size ratios. Thus, to achieve scalability (i.e., good performance as the number of processors available in a machine increases), instead of using a single algorithm, the approach discussed is to have a set of algorithms from which the most appropriate algorithm or combination of algorithms is selected based on the ratio calculated from the scaled machine size. The importance of using a set of algorithms also has been recognized by other researchers (e.g., [37]). Experimental results from the MasPar MP-1 (SIMD), nCUBE 2 (MIMD), and PASM (mixed-mode) parallel processing systems are shown to support the theoretical results derived herein. These machines are all part of the Purdue School of Electrical Engineering Parallel Processing Laboratory.

[1] This research was supported by Rome Laboratory under Contract F30602-92-C-0150, by the Naval Ocean Systems Center under the High Performance Computing Block, ONT, by the Office of Naval Research under Grant N00014-90-J-1937, by the National Science Foundation under Grant CDA-9015696, and by National Aeronautical and Space Administration under Grant NGT-50961.

[2] mucheng@qcunix.acc.qc.edu.
[3] nation@vnet.ibm.com.
[4] jarmstro@ecn.purdue.edu.
[5] hj@ecn.purdue.edu.
[6] sdkim@dobong.kwangwoon.ac.kr.
[7] mark.nichols@sandiegoca.ncr.com.
[8] gherrity@nosc.mil.

0743-7315/94 $6.00

In Section 2, the MQF problem is defined. Parallel processing system models for which the application is targeted are overviewed in Section 3. Section 4 describes the uncoupled data-parallel algorithm. Several coupled data-parallel implementations and the impact of problem partitioning are presented in Section 5. A generalized method, of which the uncoupled and coupled methods are special cases, is defined in Section 6. In Section 7, the theoretical results of Sections 4, 5, and 6 that are useful for choosing an optimal algorithm are reviewed. Also in Section 7, a combined coupled/uncoupled approach is presented. Experimental results are discussed in Section 8. Related work is addressed in Section 9.

2. THE MQF PROBLEM

Let the *s-vector*, or steering vector, be an $n \times 1$ vector of complex numbers and v be the total number of s-vectors. Define M to be an $n \times n$ matrix of complex numbers and $M(i, j)$ to be the element of M in row i and column j, for $0 \le i, j < n$. The qth s-vector is denoted by s_q, for $0 \le q < v$. Element m of the s-vector q is denoted $s_q(m)$, for $0 \le m < n$. Let H denote the Hermitian transposition, i.e., the complex conjugate transposition of the s-vector (where the complex conjugate of $a + bi$ is $a - bi$). Then, the MQF calculation can be formally defined as

$$w_q = s_q^{\mathrm{H}} M s_q \quad \text{for } 0 \le q < v. \tag{1}$$

For easy reference, Table I summarizes most of the parameters used in the paper.

In addition to the MVDR problem, this type of computation also appears in other problems. For example, when mathematically modeling various physical phenomena, minimization problems involving positive definite matrices occur [50]. An equation of the quadratic form $P(x) = (1/2)x^{\mathrm{T}}Mx - x^{\mathrm{T}}b$, where M is a positive definite matrix, is minimized at the point where $Mx = b$ and the minimum value is $P(M^{-1}b) = -(1/2)b^{\mathrm{T}}M^{-1}b$. The parallel algorithms proposed here can be easily generalized to compute this equation.

If the computation of v quadratic forms is performed on a serial machine, $v(n^2 + n) = vn^2 + vn$ complex multiplications and $v(n(n - 1) + (n - 1)) = vn^2 - v$ complex additions are required. Also, the serial machine must have enough memory space to store the entire M matrix and v s-vectors, with n^2 and vn complex numbers, respectively. It is assumed that each complex number is represented by two floating point numbers.

TABLE I
Summary of Notation Used throughout the Paper

Notation	Meaning
$M(i, j)$	The element of matrix M in row i and column j
n	The number of rows and columns of matrix M
v	Total number of s-vectors
p	Number of PEs in a parallel system
$s_q(m)$	The element m of the qth s-vector
r_q	$s_q^{\mathrm{H}} M$
w_q	$s_q^{\mathrm{H}} M s_q$ or $r_q s_q$

3. PARALLEL SYSTEM MODEL

There are several parallel machine models that have been proposed (e.g., Parallel-RAM [26], Vector-RAM [12]), and the time complexities of the algorithms described here are different on each. To focus the scope of this research, one particular parallel system model is assumed. The model includes p processing elements (*PEs*) of the same computing power, where each PE is a processor and memory module pair. Such a configuration is often referred to as a physically distributed memory machine, and is used in most current parallel systems with 64 or more processors, e.g., MasPar MP-1 [11] and nCUBE 2 [28], as well as in the PASM prototype [21, 48]. The proposed data-parallel algorithms can be implemented on an SIMD, MIMD, or mixed-mode distributed memory machine. The models used here for these three types of parallel machines are briefly overviewed below.

An *SIMD* [22] machine is typically composed of a control unit (*CU*), a set of PEs, and an interconnection network. In an SIMD machine, the enabled PEs receive and synchronously execute common instructions that are broadcast from the CU. The PEs fetch data from their memory modules. Because all the PEs are operating synchronously, the computations performed by the PEs can be overlapped with the CU's execution of control-flow instructions and/or any instruction operating on data common to all PEs. This is called *CU/PE overlap* [32]. The interconnection network allows PEs to communicate among themselves and exchange data. Examples of SIMD machines that have been constructed include CLIP4 [25], CM-2 [53], DAP [30], Illiac IV [13], MasPar MP-1 [11], MPP [6, 7], and STARAN [5].

An *MIMD* machine consists of a set of PEs and an interconnection network. In contrast to the SIMD machine, where all processors are performing the same instruction in lock-step on their own data, in an MIMD machine, each processor executes instructions from its local memory, asynchronously with respect to each other [22]. As with the SIMD model, the interconnection network provides communication links among the PEs. Examples of large MIMD systems that have been constructed are BBN Butterfly [17], CM-5 [51], IBM RP3 [40], Intel iPSC Cube [2], and nCUBE 2 [28].

In a *mixed-mode* parallel processing system, processors are capable of executing in either SIMD or MIMD mode of parallelism. The ability to switch between the two modes at instruction-level granularity with very little overhead allows the parallelism mode to vary for each portion of an algorithm. There have been at least three

mixed-mode parallel prototypes built. TRAC [35], designed and built at the University of Texas at Austin, implemented mixed-mode switching at the subtask level, rather than at the instruction level. OPSILA [19], from Laboratoire de Signaux et Systems in Nice, France, has the capability of executing instructions in either SIMD or *SPMD* (single program–multiple data stream) mode, a subclass of MIMD. PASM [3, 45, 46, 48] can dynamically switch between SIMD and full MIMD modes of parallelism at instruction-level granularity with negligible overhead. Triton/1 [41], currently under development, is capable of mixed-mode parallelism at the instruction-level.

PASM is reconfigurable along three dimensions: modes of parallelism (SIMD/MIMD) as described above, partitionability, and interprocessor connections. It can be dynamically partitioned into dependent or communicating submachines of various sizes, each having the same characteristics as the original machine. PASM uses a fault-tolerant variation [1] of the flexible multistage cube type network [44], which allows the connection patterns among the processors to be varied. Extrapolation techniques can be used to study the potential performance of larger PASM systems with the small-scale prototype, as done, for example, in [14].

Experimental results for the MQF problem were obtained from SIMD (MasPar MP-1), MIMD (nCUBE 2), and mixed-mode (PASM) machines. There are many trade-offs among these modes of parallelism [8, 10, 21, 43]. Computational characteristics of algorithms affect the choice of the best mode for executing each section of code [31]. A general discussion of these trade-offs is beyond the scope of this paper (see [43]).

A multistage cube or hypercube (single-stage cube) network [44] is assumed for the proposed algorithms. These networks are flexible interconnection networks that can perform the PE-to-PE permutations required by the MQF implementations without any conflicts (i.e., without two simultaneous communication paths needing a common network link). The multistage cube network has been used or proposed for use in systems, such as the Goodyear Aerospace Corporation ASPRO [7], BBN Butterfly [17], Cedar [33], IBM RP3 [40], Goodyear Aerospace Corporation STARAN [5], PASM [45, 48], and NYU Ultracomputer [27]. The class of multistage cube topologies includes: baseline [55], delta [38], flip [4], generalized cube [44], indirect binary n-cube [39], multistage shuffle-exchange [52], omega [34], and SW-banyan ($S = F = 2$, $L = n$) [35] networks [44, 55]. The hypercube interconnection topology has been implemented in the CM-2 [53], Intel iPSC cube [2], and nCUBE 2 [28]. These algorithms can be adapted for use on systems with other types of interconnection networks.

The following sections describe the data-parallel implementations of the MQF algorithms. Included is a discussion of the impact of the mode of parallelism and the network topology used.

4. THE UNCOUPLED DATA-PARALLEL METHOD

The uncoupled method uses the obvious approach to parallelizing the MQF problem. Assume that p divides v, i.e., $v = c * p$ where c is an integer. By distributing v different s-vectors evenly among p PEs, each PE can compute c quadratic forms in parallel without having to communicate with any other PE. If p does not divide v, i.e., $v \neq c * p$, $\lceil v/p \rceil$ different s-vectors will be assigned to each of $v \bmod p$ PEs, and $\lfloor v/p \rfloor$ different s-vectors will be assigned to each of the remaining PEs. However, in this case some PEs will do more work than others. If $v < p$, only v PEs will actually be utilized. Thus, for the uncoupled method, utilization is good when either p divides v or $v \gg p$.

Solving a quadratic form can be decomposed into two phases. Letting $*$ denote scalar multiplication, the phase (1) calculation is $r_q(y) = \sum_{x=0}^{n-1} s_q^{\mathrm{H}}(x) * M(x, y)$, and the phase (2) calculation is $w_q = \sum_{y=0}^{n-1} r_q(y) * s_q(y)$. The number of complex multiplications and additions required for each s-vector during phase (1) is n^2 and $n(n-1)$, respectively. For phase (2), n complex multiplications and $(n-1)$ complex additions are performed for each s-vector. Thus, the total number of complex multiplications and additions performed for each s-vector is $n^2 + n$ and $n^2 - 1$, respectively. Because some PEs are assigned $\lfloor v/p \rfloor$ different s-vectors, the maximum number of parallel complex multiplications and additions is $\lceil v/p \rceil (n^2 + n)$ and $\lceil v/p \rceil (n^2 - 1)$, respectively.

For the uncoupled method, the maximum storage requirement per PE is n^2 complex elements for the M matrix and $\lceil v/p \rceil n$ complex elements for the s-vectors. This assumes that as each element of r_q is computed, it is used to calculate w_q and is not stored (i.e., phases (1) and (2) are temporally interleaved).

The uncoupled data-parallel method described above can be implemented in either SIMD or MIMD mode. In SIMD mode, CU/PE overlap can occur. Also, during phase (1), each element of matrix M can be embedded as an immediate operand of an SIMD instruction broadcast from the CU to the PEs. Thus, each PE need not store the M matrix. One disadvantage of the implicit synchronization of the SIMD mode of parallelism is that it can cause some PEs to remain idle because of data-dependent variable-time instructions (e.g., a floating point addition instruction) [10, 20].

The data-dependent variable-time instructions are better performed in MIMD mode than in SIMD mode [20]. However, CU/PE overlap is not possible in MIMD mode. Because both the SIMD and MIMD modes of parallelism have their advantages and disadvantages, the right choice of execution mode depends on the machine design details and the characteristics of the data being processed. In this case, mixed-mode would not be used because the advantages and disadvantages of each mode of parallelism are the same throughout the program; i.e., the program characteristics do not change. Hence, the

program will be performed entirely in one of the two modes of parallelism.

5. THE COUPLED DATA-PARALLEL METHOD

5.1. Overview

This section describes how to map the MQF problem onto p PEs configured as an $a \times b$ ($=p$) logical grid (not necessarily a physical grid or mesh), where $1 \le a, b \le n$, and a, b, and p are powers of 2. Furthermore, the impact of partitioning the problem is discussed. The special cases of $p = n$ ($a = 1, b = n$) and $p = n^2$ ($a = b = n$) were studied first and used to develop this general case (see [54]).

There are two common data layouts of the M matrix and s-vectors in memory, referred to in [23] as the block and scattered decompositions. For the MQF problem, the block decomposition method has contiguous entries of the M matrix and s-vectors assigned to PEs in blocks. The scattered decomposition method has consecutive entries in the M matrix and s-vectors assigned to different PEs. For this case study, the block decomposition method is used.

For ease of presentation, n is assumed to be a power of 2. The proposed method can be readily extended to cover the case when n is not a power of 2 (see [54]).

5.2. Coupled Data-Parallel Algorithm: General Description and Evaluation

Let $PE(i, j)$ denote the PE in row i and column j in an $a \times b$ logical PE grid, where $0 \le i < a$ and $0 \le j < b$. Figure 1 illustrates how M, s_q^H, and s_q are distributed across the PEs of the $a \times b$ grid. The s-vectors are loaded into the PE memories such that an (n/a)-element part of the Hermitian of each s-vector, s_q^H, and an (n/b)-element part of at most $\lceil v/a \rceil$ different s-vectors, s_q, are stored in each PE memory. Each PE also holds an $(n/a) \times (n/b)$ portion of M. The exact elements stored in each PE are defined for general PE(i, j) in Fig. 1. For example, if $v = 6$, $n = 4$, $a = 2$, and $b = 4$, PE(1,2) contains $M(2,2)$, $M(3,2)$, $s_q^H(2)$ and $s_q^H(3)$ for $0 \le q < 6$, $s_1(2)$, $s_3(2)$, and $s_5(2)$, as shown in the boxes in Fig. 2.

The calculation can be decomposed into two phases: the phase (1) calculation is $r_q(y) = \sum_{x=0}^{n-1} s_q^H(x) * M(x, y)$, and the phase (2) calculation is $w_q = \sum_{y=0}^{n-1} r_q(y) * s_q(y)$. Assuming $v = 6$, $n = 4$, $a = 2$, and $b = 4$, Fig. 3 illustrates the complex multiplications and subsequent complex additions performed by PE(1,2). For each s-vector s_q^H, each

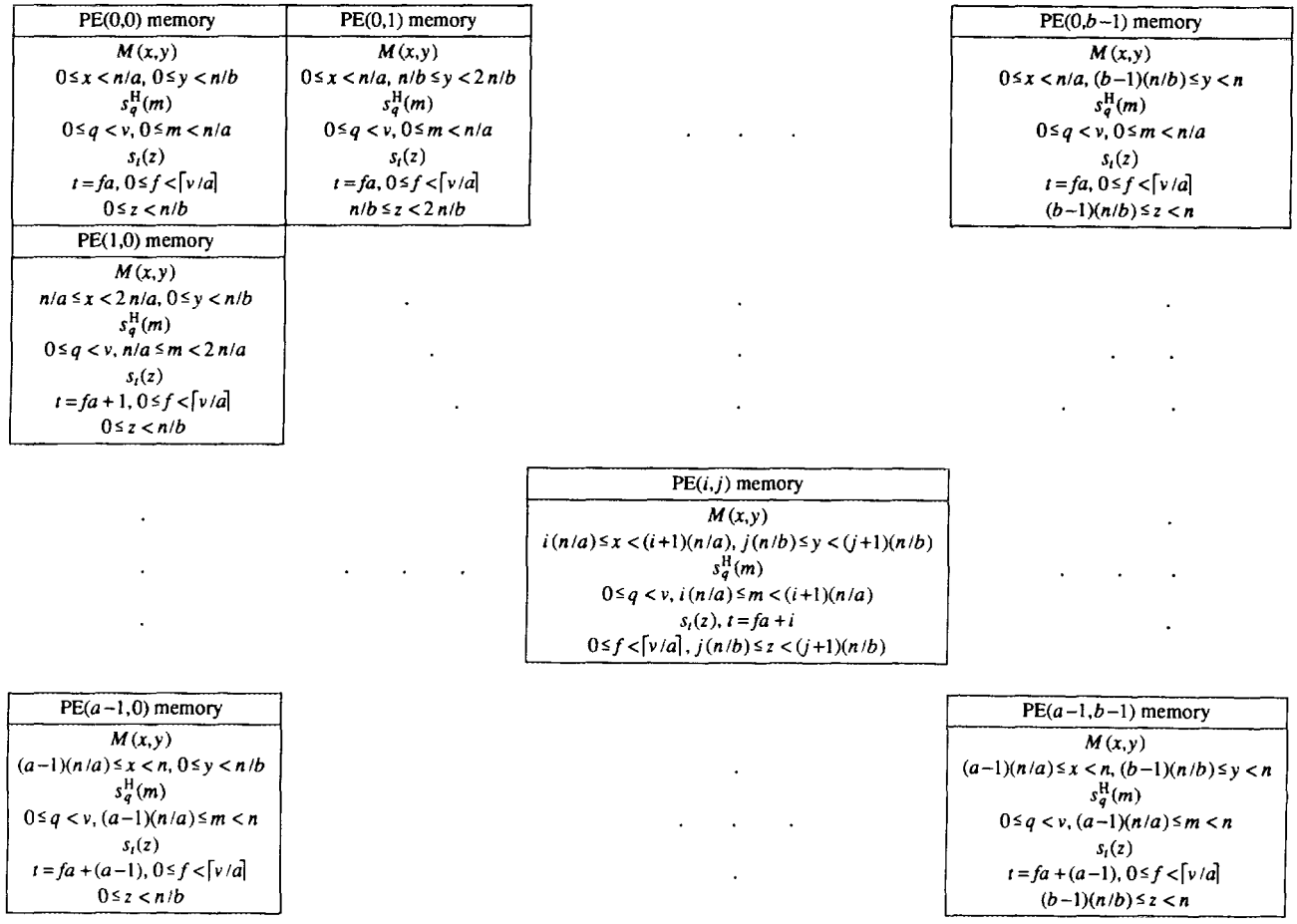

FIG. 1. Distribution of M and v steering vectors onto $a \times b$ PEs. (The maximum value t can have is $v - 1$.)

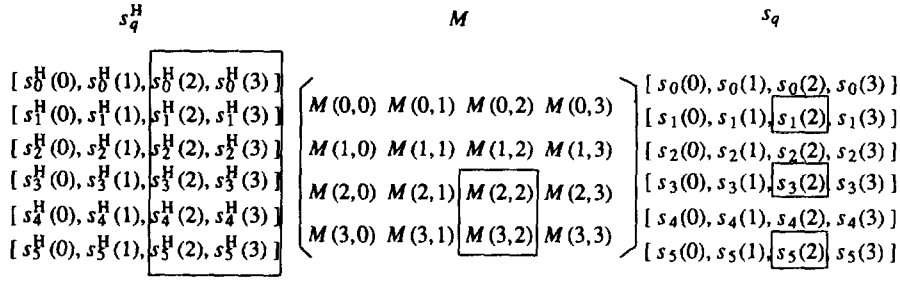

FIG. 2. Data elements in boxes are stored in PE(1,2)'s local memory.

PE calculates (n/a) products for each of (n/b) $r_q(y)$'s. PE(i, j) calculates a partial $r_q(y)$ sum $\sum_{x=i(n/a)}^{(i+1)(n/a)-1} s_q^H(x) * M(x, y)$, for $j(n/b) \le y < (j+1)(n/b)$ and $0 \le q < v$. Thus, each PE performs $v(n/b)(n/a) = vn^2/p$ complex multiplications followed by $v(n/b)(n/a - 1) = (vn/p)(n - a)$ complex additions. All p PEs do these calculations simultaneously.

Next, the partial sums indicated above must be combined among PEs to calculate the $r_q(y)$ values. The elements to be totaled to calculate a given $r_q(y)$ are resident in the a PEs of a single column. Recursive doubling can be used to compute the sums [49]. Performing interleaved recursive doubling procedures [43, 46] to find simultaneously the sum of the elements of ρ distinct vectors is referred to here as a *ρ-recursive doubling*.

To help clarify the operations involved in the ρ-recursive doubling technique, Fig. 4 illustrates how $N = 4$ PEs would sum the elements for an arbitrary set of vectors. In this example, there are $\rho = 6$ vectors, α_η for $0 \le \eta < 6$, each having N elements. If α_η^j is the jth element of vector η with α_η^j initially stored in PE(j), then the sum that is sought is $\sum_{j=0}^{3} \alpha_\eta^j$ for each η, $0 \le \eta < 6$. In Fig. 4, the arrows denote the transfer of data from one PE to another followed by an addition; together these form a transfer-add operation. Five transfer-adds are required: three for the first step and two for the second step.

PE(1,2) memory contents when forming

partial sums $\sum_{x=2}^{3} s_q^H(x) * M(x,y),\ 2 \le y < 3,\ 0 \le q < 6$

$$
\begin{aligned}
&s_0^H(2) * M(2,2) + s_0^H(3) * M(3,2)\\
&s_1^H(2) * M(2,2) + s_1^H(3) * M(3,2)\\
&s_2^H(2) * M(2,2) + s_2^H(3) * M(3,2)\\
&s_3^H(2) * M(2,2) + s_3^H(3) * M(3,2)\\
&s_4^H(2) * M(2,2) + s_4^H(3) * M(3,2)\\
&s_5^H(2) * M(2,2) + s_5^H(3) * M(3,2)
\end{aligned}
$$

FIG. 3. Contents of PE(i, j), $i = 1, j = 2$, memory after forming the partial sums $\sum_{x=i(n/a)}^{(i+1)(n/a)-1} s_q^H(x) * M(x, y)$, $0 \le q < v$, $j(n/b) \le y < (j+1)(n/b)$, for $v = 6$, $n = 4$, $a = 2$, and $b = 4$.

In general, the final sums for all ρ vectors, each with N elements, are computed in $\log_2 N$ steps, where $\lceil \rho/2^i \rceil$ transfer-add operations occur at step i for $1 \le i \le \log_2 N$. Thus, the total number of transfer-adds required in the ρ-recursive doubling procedure is $\sum_{i=1}^{\log_2 N} \lceil \rho/2^i \rceil$. Figure 5 gives an algorithm to perform the ρ-recursive doubling procedure. With the multistage cube network, each of these parallel transfers can be done in a single pass through the network without any conflicts [44]. With the hypercube network, because each of these parallel transfers involves pairs of PEs that are directly connected, no conflict will occur for any inter-PE transfer [44].

In the coupled algorithm, because each of the a PEs in a given column contributes to n/b different $r_q(y)$'s for each of the v different s-vectors, a (vn/b)-recursive doubling procedure, utilizing $\sum_{i=1}^{\log_2 a} \lceil (vn/b)/2^i \rceil$ transfer-add operations, on the a PEs of each column can be used. (Recall that all the partial sums needed to compute a given $r_q(y)$ are stored in PEs in the same column.) All b logical columns of PEs simultaneously perform the b (vn/b)-recursive doublings (one per column). Both the multistage cube and hypercube networks can perform the needed inter-PE communications for all PEs with no conflicts. After the recursive doubling, PE(i, j) will hold $r_q(y)$, where $j(n/b) \le y < (j+1)(n/b)$ and for all q, $0 \le q < v$, where $q \bmod a = i$. As an example for $v = 6$, $n = 4$, $a = 2$, and $b = 4$, the calculation of $r_q(j)$ (for all q, j) is illustrated in Fig. 6.

Phase (2) of the coupled algorithm involves the formation of $w_q = \sum_{y=0}^{n-1} r_q(y) * s_q(y)$. After the first phase of computation, each PE holds at most $\lceil v/a \rceil (n/b)$ distinct $r_q(y)$ components and the corresponding elements of the s-vectors, $s_q(y)$, required to form $r_q(y) * s_q(y)$. PE(i, j) forms the local partial sums of the products $\sum_{y=j(n/b)}^{(j+1)(n/b)-1} r_q(y) * s_q(y)$, for all q, $0 \le q < v$, where $q \bmod a = i$. This is shown in Fig. 7 for $v = 6$, $n = 4$, $a = 2$, and $b = 4$. To do this, $\lceil v/a \rceil (n/b)$ complex multiplications and $\lceil v/a \rceil (n/b - 1)$ complex additions are needed. Then, similar to that of the first phase, a $\lceil v/a \rceil$-recursive doubling procedure is employed in each row of b PEs to combine the partial sums above to form w_q for all q. All PEs in row i, i.e., PE(i, j) for $0 \le j < b$ and i fixed, participate in the same $\lceil v/a \rceil$-recursive doubling procedure to form at most $\lceil v/a \rceil$ different w_q values, for all q, $0 \le q < v$, where $q \bmod a = i$. To compute all v values of w_q in parallel, a independent

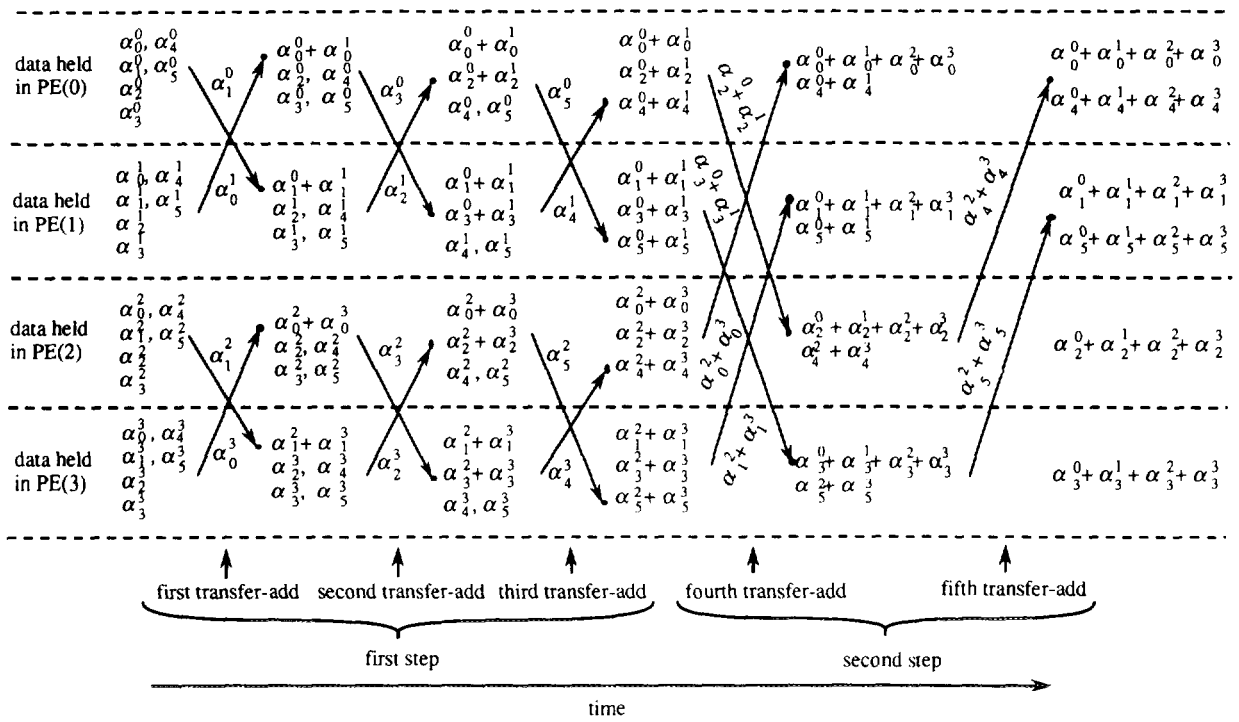

FIG. 4. Using recursive doubling to sum simultaneously six vectors, α_η^j, for $0 \le \eta < 6$ and $0 \le j < 4$, on $N = 4$ PEs.

$\lceil v/a \rceil$-recursive doubling procedures are performed simultaneously (one per row). Both the multistage cube and hypercube networks can perform the required inter-PE transfers with no conflicts.

```
/* The following procedure will be performed on PE(j), 0 ≤ j < N. */
for i = 1 to log2N do { /* step i */
    k = j ⊕ 2^(i-1);      /* the destination PE number */
    for β = 0 to (⌊ρ/2^i⌋ - 1) do {
        η = (k mod 2^i) + β2^i;
        transfer local partial sum of vector η to PE(k);
        perform the complex addition with the received data and the
        corresponding local partial sum in PE(j);
    }
    if (ρ/2^i ≠ ⌊ρ/2^i⌋) {
        β = ⌊ρ/2^i⌋;
        η = (k mod 2^i) + β2^i;
        if (j & 2^(i-1) ≠ 0) {
            transfer local partial sum of vector η to PE(k);
        } else {
            perform the complex addition with the received data and the
            corresponding local partial sum in PE(j);
        }
    }
}
```

FIG. 5. An algorithm to perform the ρ-recursive doubling procedure.

After the $\lceil v/a \rceil$-recursive doubling procedures, each PE stores at most $\lceil v/p \rceil$ elements of w_q. Specifically, the results placed in PE(i, j) are w_q, for all q, $0 \le q < v$, where $q = ja + i + fab$, for $0 \le f < \lceil v/p \rceil$. This is illustrated in Fig. 8 for $v = 6$, $n = 4$, $a = 2$, and $b = 4$. To form w_q, $\sum_{i=1}^{\log_2 b} \lceil v/(2^i a) \rceil$ transfer-add operations are performed in this phase.

In summary, the first phase requires vn^2/p complex multiplications followed by $(vn/p)(n - a)$ complex additions and $\sum_{i=1}^{\log_2 a} \lceil vn/(2^i b) \rceil$ transfer-adds. The second phase requires $\lceil v/a \rceil (n/b)$ complex multiplications followed by $\lceil v/a \rceil (n/b - 1)$ complex additions and $\sum_{i=1}^{\log_2 b} \lceil v/(2^i a) \rceil$ transfer-adds. Table II summarizes the computational complexity of the uncoupled algorithm from Section 4 and the coupled algorithm of this subsection.

By regarding a transfer-add as an inter-PE transfer and a complex addition, the complex additions required in the coupled scheme becomes $(vn/p)(n - a) + \lceil v/a \rceil (n/b - 1) + \sum_{i=1}^{\log_2 a} \lceil vn/(2^i b) \rceil + \sum_{i=1}^{\log_2 b} \lceil v/(2^i a) \rceil$. The lower bounds for the number of complex operations can be derived as follows:

$$\text{Multiplications: } vn^2/p + \lceil v/a \rceil (n/b) \ge vn^2/p + (v/a)(n/b) = (v/p)(n^2 + n); \quad (2)$$

$$\text{Additions: } (vn/p)(n - a) + \lceil v/a \rceil (n/b - 1) + \sum_{i=1}^{\log_2 a} \lceil vn/(2^i b) \rceil + \sum_{i=1}^{\log_2 b} \lceil v/(2^i a) \rceil \ge (vn/p)(n - a) + (v/a)(n/b - 1)$$

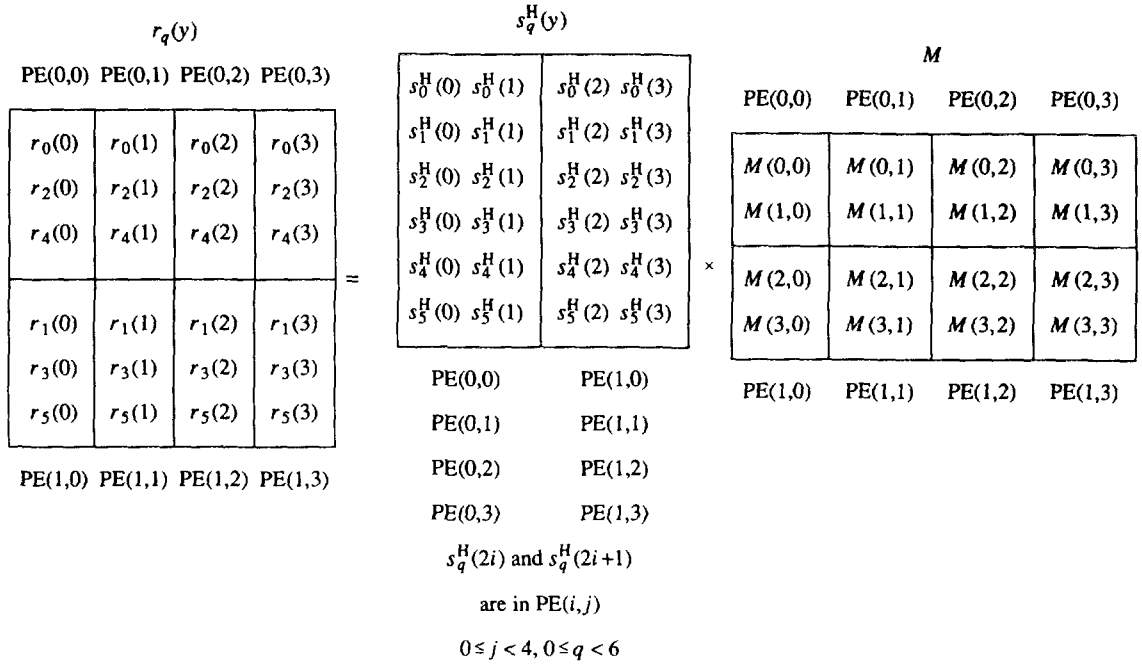

FIG. 6. Distribution of data over the PEs for $v = 6$, $n = 4$, $a = 2$, and $b = 4$, where $r_q(y) = \sum_{x=0}^{3} s_q^H(x) * M(x, y)$.

$$+ \sum_{i=1}^{\log_2 a} vn/(2^i b) + \sum_{i=1}^{\log_2 b} v/(2^i a)$$

$$= (v/p)(n^2 - 1); \quad (3)$$

Inter-PE transfers: $\sum_{i=1}^{\log_2 a} \lceil vn/(2^i b) \rceil + \sum_{i=1}^{\log_2 b} \lceil v/2^i a \rceil$

$$\geq \sum_{i=1}^{\log_2 a} vn/(2^i b) + \sum_{i=1}^{\log_2 b} v/(2^i a)$$

$$= (vn/p)(a - 1) + (v/p)(b - 1). \quad (4)$$

The lower bounds are attainable when v/p is an integer.

Recall that with the coupled method, $p = a * b$, for $0 < a, b \leq n$, and a, b, and n are assumed to be powers of 2. As shown in Eqs. (2) and (3), the lower bounds for the number of complex multiplications and additions are independent of a and b, and there is a factor of p speedup on the required number of serial arithmetic operations. However, if v, p, and n are fixed, to minimize the number of inter-PE transfers, a should be set as small as possible while remaining a power of two (and b should not be greater than n due to the structure of the algorithm). Thus, if $p \leq n$, a $1 \times p$ logical grid of PEs should be

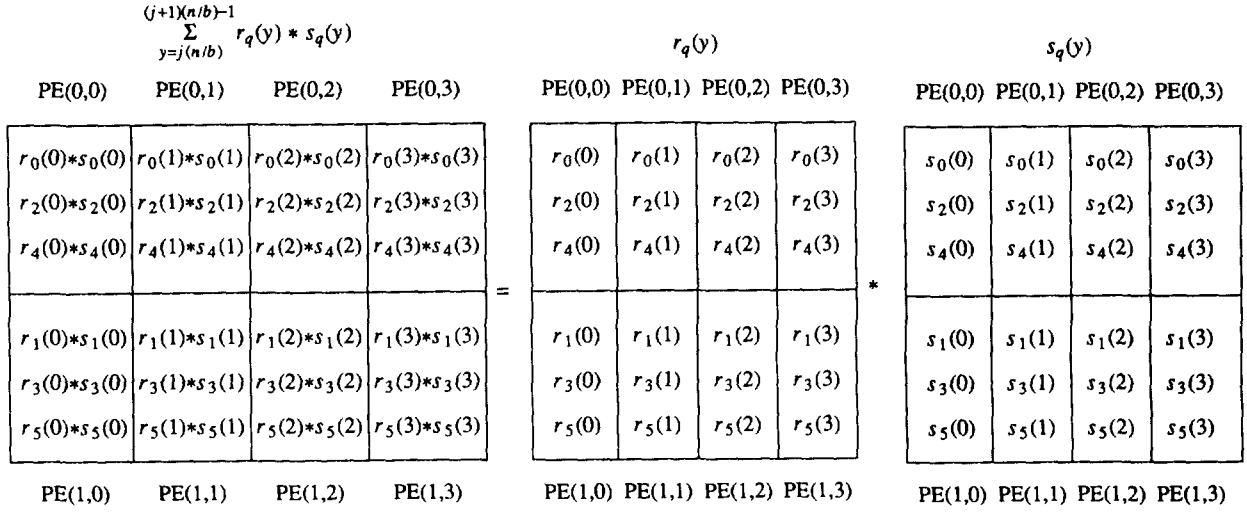

FIG. 7. Distribution of data over the PEs for $v = 6$, $n = 4$, $a = 2$, and $b = 4$ (* represents element-wise scalar multiplication).

PE(0,0)	PE(0,1)	PE(0,2)	PE(0,3)
w_0	w_2	w_4	-
w_1	w_3	w_5	-
PE(1,0)	PE(1,1)	PE(1,2)	PE(1,3)

FIG. 8. Distribution of w_q's at conclusion of procedure for $v = 6$, $n = 4$, $a = 2$, and $b = 4$.

chosen to compute the problem and consequently, the recursive doubling procedure is involved only in the second phase of computation. If $n < p \leq n^2$, a logical $(p/n) \times n$ grid of PEs is the optimal choice and the recursive doubling procedure is required to total the partial sums stored on different PEs during both the first and second phases of computation. If $p > n^2$, according to the computational characteristics of the coupled method, only n^2 PEs can be utilized and the remaining PEs are idle. An alternative using machine partitioning is discussed in Section 6.

The data memory storage requirements for this method are $n^2/p + vn/a + \lceil v/a \rceil (n/b)$ complex numbers. The n^2/p term is for the $(n/a) \times (n/b)$ subarray of M, the vn/a term is for the (n/a)-element part of each of the v different s-vector Hermitians, s_q^H, and the $\lceil v/a \rceil (n/b)$ term is for the (n/b)-element part of $\lceil v/a \rceil$ different s-vectors, s_q, stored in each PE memory.

In terms of computational effectiveness, consider the trade-offs between the SIMD and MIMD modes of parallelism for the coupled method. The trade-offs for the local computations, i.e., the sequences of complex multiplications and additions executed within each PE, are similar to those discussed in Section 4. For the transfer-add operations, in general, they are executed more effectively in SIMD mode [10]. Typically, in MIMD mode, explicit synchronization primitives and identification protocols are required for inter-PE communications. However, due to the implicit synchronization in SIMD mode, these are not necessary. The choice between a pure SIMD and a pure MIMD mode implementation will depend upon characteristics of the data being processed, machine architecture, and the resulting relative impact of effects discussed above. For PASM, if MIMD mode is better for the local computations, then the optimal implementation of the coupled data-parallel method would be mixed-mode, i.e., MIMD for local computations and SIMD for the transfer-add operations; otherwise, the optimal implementation would be SIMD mode.

5.3. Comparison of the Single and Multiple Groups Coupled Data-Parallel Methods

When employing the coupled data-parallel method, it is assumed that all active PEs are working together as a single group at execution time. However, some machines (e.g., nCUBE, PASM) allow another possibility. For the $n < p \leq n^2$ case, if $p = a * n$ PEs are partitioned into $a = p/n$ independent groups of $1 \times n$ PEs, $\lceil v/a \rceil$ different s-vectors are assigned to each of v mod a groups, and $\lceil v/a \rceil$ different s-vectors are assigned to each of the remaining groups. Each independent group of PEs will have a copy of the M matrix. Then the recursive doubling becomes unnecessary during the first phase of computation. This may reduce overall execution time. Assuming $n < p \leq n^2$ and $a = p/n$, Table III shows the computational complexity of the *single group coupled method* (i.e., one group of $a \times n$ PEs) and the *multiple groups coupled method* (i.e., "a" groups of $1 \times n$ PEs). More storage is required for the multiple groups coupled method; however, computational complexity is the performance metric stressed here.

As shown in Table III, if a divides v (i.e., $v = ca$ and c is an integer), the multiple groups coupled method is always better than the single group coupled method. When $v \neq ca$, although the number of inter-PE transfers is reduced by partitioning all PEs into "a" groups of $1 \times n$ PEs (consequently, the communication overhead is reduced), the number of complex additions and multiplications is increased. Thus, if the reduction in communication time is greater than the increase in computation time, the multiple groups coupled method will outperform the single group coupled method (as shown in Section 8); otherwise, the single group coupled method will be at least as good. This trade-off between computation time and communication time depends upon the speed at which a particular machine can perform a multiplication, addition, and inter-PE transfer.

TABLE II
Computational Complexity for the Uncoupled and Coupled Parallel Algorithms

	Uncoupled complexity	Coupled complexity
Complex multiplications	$\lceil v/p \rceil (n^2 + n)$	$vn^2/p + \lceil v/a \rceil (n/b)$
Complex additions	$\lceil v/p \rceil (n^2 - 1)$	$(vn/p)(n - a) + \lceil v/a \rceil (n/b - 1)$
Transfer-adds	—	$\sum_{i=1}^{\log_2 a} \lceil vn/(2^i b) \rceil + \sum_{i=1}^{\log_2 b} \lceil v/2^i a \rceil$

Note. Recall that p is the total number of PEs in the logical PE grid ($p = a * b$).

TABLE III

Computational Complexity for the One Group $a \times n$ PEs and "a" Groups of $1 \times n$ PEs Coupled Methods when $n < p \leq n^2$

	L: one group of $a \times n$ PEs, where $a = p/n$	R: "a" groups of $(1 \times n)$ PEs	$\Delta = \lvert L - R \rvert$	
			$v = ca$	$v \neq ca$
M	$vn/a + \lceil v/a \rceil$	$\lceil v/a \rceil(n + 1)$	L = R	L < R ($0 < \Delta < n$)
A	$(v/a)(n - a) + \sum_{i=1}^{\log_2 a} \lceil v/2^i \rceil + \sum_{i=1}^{\log_2 n} \lceil v/2^i a \rceil$	$\lceil v/a \rceil(n - 1) + \sum_{i=1}^{\log_2 n} \lceil v/(2^i a) \rceil$	L = R	L ≤ R ($0 \leq \Delta < n - 1$)
T	$\sum_{i=1}^{\log_2 a} \lceil v/2^i \rceil + \sum_{i=1}^{\log_2 n} \lceil v/(2^i a) \rceil$	$\sum_{i=1}^{\log_2 n} \lceil v/(2^i a) \rceil$	L > R ($\Delta = v - (v/a)$)	L > R ($\log_2 a + 2^{\lfloor \log_2 v \rfloor}(1 - 1/a) \geq \Delta > v - \lceil v/a \rceil$)

Note. M, A, and T represent multiplications, additions, and inter-PE transfers, respectively; c is an integer constant.

6. GENERALIZING THE MQF ALGORITHM

Given p PEs where $p = a * n \leq n^2$, the multiple groups coupled method described in the previous subsection partitions all p PEs into a independent groups of $1 \times n$ PEs. This idea can be generalized to allow different methods of solving the MQF problem, three of which are the uncoupled method (Section 4), the single group coupled method (Subsection 5.2), and the multiple group coupled method (Subsection 5.3). Let $p = \gamma * \alpha * \beta$, where α, β, and γ are powers of 2 and $1 \leq \gamma \leq p$ and $1 \leq \alpha, \beta \leq n$. Given p PEs where $p \leq n^2$, if all p PEs can be partitioned into γ groups of $\alpha \times \beta$ PEs, then the problem is to compute the MQF for $\lceil v/\gamma \rceil$ different s-vectors on $\alpha \times \beta$ PEs. It can be seen that the single group coupled and uncoupled methods represent two extreme cases of the generalized method, i.e., for $\gamma = 1$, $\alpha = a$, $\beta = b$, and $\gamma = p$, $\alpha = \beta = 1$, respectively.

Let v, a, and b in Table II be replaced by $\lceil v/\gamma \rceil$, α, and β, respectively. Then, the computational complexity of the generalized method is $\lceil v/\gamma \rceil(n^2/(\alpha\beta)) + \lceil v/(\gamma\alpha) \rceil (n/\beta)$ complex multiplications, $\lceil v/\gamma \rceil(n/(\alpha\beta))(n - \alpha) + \lceil v/(\gamma\alpha) \rceil(n/\beta - 1)$ complex additions, and $\sum_{i=1}^{\log_2 \alpha} \lceil vn/(\gamma\beta 2^i) \rceil + \sum_{i=1}^{\log_2 \beta} \lceil v/(\gamma\alpha 2^i) \rceil$ transfer-adds. By regarding a transfer-add as an inter-PE transfer and a complex addition, the number of complex additions required in the generalized method becomes $\lceil v/\gamma \rceil(n/(\alpha\beta))(n - \alpha) + \lceil v/(\gamma\alpha) \rceil(n/\beta - 1) + \sum_{i=1}^{\log_2 \alpha} \lceil vn/(\gamma\beta 2^i) \rceil + \sum_{i=1}^{\log_2 \beta} \lceil v/(\gamma\alpha 2^i) \rceil$.

Similar to the single group coupled method, given v, p, and n values, the actual number of complex operations and inter-PE transfers are dependent on the values of γ, α, and β. The lower bounds for those operations are derived as follows:

$$\text{Inter-PE transfers: } \sum_{i=1}^{\log_2 \alpha} \lceil vn/(\gamma\beta 2^i) \rceil + \sum_{i=1}^{\log_2 \beta} \lceil v/(\gamma\alpha 2^i) \rceil$$

$$\geq (vn/(\gamma\beta))(1 - 1/\alpha) + (v/(\gamma\alpha))(1 - 1/\beta) = (vn/p)(\alpha - 1) + (v/p)(\beta - 1); \tag{5}$$

$$\text{Multiplications: } \lceil v/\gamma \rceil(n^2/(\alpha\beta)) + \lceil v/(\gamma\alpha) \rceil(n/\beta)$$

$$\geq (v/\gamma)(n^2/(\alpha\beta)) + (v/(\gamma\alpha))(n/\beta) = (v/p)(n^2 + n); \tag{6}$$

$$\text{Additions: } \lceil v/\gamma \rceil(n/(\alpha\beta))(n - \alpha) + \lceil v/(\gamma\alpha) \rceil(n/\beta - 1)$$

$$+ \sum_{i=1}^{\log_2 \alpha} \lceil vn/(\gamma\beta 2^i) \rceil + \sum_{i=1}^{\log_2 \beta} \lceil v/(\gamma\alpha 2^i) \rceil$$

$$\geq (vn/(\gamma\alpha\beta))(n - \alpha) + (v/(\gamma\alpha\beta))(n - \beta) + (vn/p)(\alpha - 1) + (v/p)(\beta - 1)$$

$$= (v/p)(n^2 - 1). \tag{7}$$

The lower bounds are attainable when v/p is an integer.

The lower bounds for Eqs. (6) and (7) are independent of α and β. When the number of PE groups (γ) is decided and keeping v, p, and n fixed, to minimize Eq. (5), α should be set as small as possible while remaining a power of two and β should not be greater than n, which is identical to the results obtained for the single group coupled method.

When $p > n^2$, not all of the PEs can be utilized by the single group coupled method. However, by partitioning p PEs into γ groups, where γ is a power of 2 and $1 \le p/\gamma \le n^2$, the resources can be fully utilized. This is done by assigning $\lceil v/\gamma \rceil$ different s-vectors to each of $v \bmod \gamma$ groups, and $\lfloor v/\gamma \rfloor$ different s-vectors to each of the remaining groups. Within each group, whether the single group or multiple groups optimal coupled method should be used can be determined based on their relative performance. Thus, the computational complexity required for this problem is equal to the complexity of computing the MQF problem for $\lceil v/\gamma \rceil$ different s-vectors on p/γ PEs. The optimal value of γ is dependent on v, n, and the machine dependent time for floating-point operations and inter-PE transfers. Table IV summarizes the computational complexity of the uncoupled, single group, and generalized methods.

Table V shows the comparison of the computational complexity between the single group coupled and uncoupled methods (the two extreme cases of γ in the generalize method). If p divides v, $\lceil v/p \rceil = v/p$,

$$vn^2/p + \lceil v/a \rceil (n/b) = (v/p)(n^2 + n), \text{ and}$$

$$(vn/p)(n - a) + \lceil v/a \rceil ((n/b) - 1) + \sum_{i=1}^{\log_2 a} \lceil vn/(2^i b) \rceil + \sum_{i=1}^{\log_2 b} \lceil v/(2^i a) \rceil$$

$$= (vn/p)(n - a) + (v/p)(n - b) + (vn/b)(1 - (1/a)) + (v/a)(1 - (1/b)) = (v/p)(n^2 - 1).$$

Thus, both uncoupled and single group coupled methods require the same number of complex additions and multiplications. However, because there are inter-PE transfers associated with the single group coupled method, the uncoupled method will outperform the single group coupled method when p divides v.

If p does not divide v,

$$vn^2/p + \lceil v/a \rceil (n/b) < \lceil v/p \rceil n^2 + \lceil v/a \rceil (n/b) < \lceil v/p \rceil (n^2 + n).$$

If it is assumed that $b = p$ for $0 < p < n$ and $b = n$ for $n \le p \le n^2$, it can be shown that

$$(vn/p)(n - a) + \lceil v/a \rceil ((n/b) - 1) + \sum_{i=1}^{\log_2 a} \lceil vn/(2^i b) \rceil + \sum_{i=1}^{\log_2 b} \lceil v/(2^i a) \rceil < \lceil v/p \rceil (n^2 - 1).$$

Although the single group coupled method takes less time to perform the needed complex additions and multiplications than the uncoupled method, the single group coupled method has the overhead of inter-PE transfers. Thus, if this compared communication overhead time is greater than the increase in computation time for the uncoupled method, the uncoupled method will outperform the single group coupled method; otherwise, the single group coupled method will be at least as good.

It can be seen from Table IV that, in general, if β and p

TABLE IV
Computational Complexity for the Coupled and Uncoupled Methods when $p \le n^2$

	Coupled method where $p = a * b = \gamma * \alpha * \beta$		Uncoupled method
	One group of $(a \times b)$ PEs	γ groups of $(\alpha \times \beta)$ PEs	
M	$\frac{vn^2}{p} + \lceil \frac{v}{a} \rceil \frac{n}{b}$	$\lceil \frac{v}{\gamma} \rceil \frac{n^2}{\alpha\beta} + \lceil \frac{v}{\gamma\alpha} \rceil \frac{n}{\beta}$	$\lceil \frac{v}{p} \rceil (n^2 + n)$
A	$\frac{vn}{p}(n - a) + \lceil \frac{v}{a} \rceil (\frac{n}{b} - 1) + \sum_{i=1}^{\log_2 a} \lceil \frac{vn}{2^i b} \rceil + \sum_{i=1}^{\log_2 b} \lceil \frac{v}{2^i a} \rceil$	$\lceil \frac{v}{\gamma} \rceil (\frac{n}{\alpha\beta})(n - \alpha) + \lceil \frac{v}{\gamma\alpha} \rceil (\frac{n}{\beta} - 1) + \sum_{i=1}^{\log_2 \alpha} \lceil \frac{vn}{\gamma\beta 2^i} \rceil + \sum_{i=1}^{\log_2 \beta} \lceil \frac{v}{\gamma\alpha 2^i} \rceil$	$\lceil \frac{v}{p} \rceil (n^2 - 1)$
T	$\sum_{i=1}^{\log_2 a} \lceil \frac{vn}{2^i b} \rceil + \sum_{i=1}^{\log_2 b} \lceil \frac{v}{2^i a} \rceil$	$\sum_{i=1}^{\log_2 \alpha} \lceil \frac{vn}{\gamma\beta 2^i} \rceil + \sum_{i=1}^{\log_2 \beta} \lceil \frac{v}{\gamma\alpha 2^i} \rceil$	

Note. M, A, and T represent the number of multiplications, additions, and inter-PE transfers, respectively.

TABLE V
Computational Complexity for the Uncoupled and Single Group Coupled Methods when $0 < p \leq n^2$

	L: single group coupled method with $p = a * b$ PEs	R: uncoupled method	$v = cp$	$v \neq cp$
M	$vn^2/p + \lceil v / a \rceil (n/b)$	$\lceil v/p \rceil (n^2 + n)$	L = R	L < R
A	$(vn/p)(n - a) + \lceil v/a \rceil ((n/b) - 1) + \sum_{i=1}^{\log_2 a} \lceil vn/(2^i b) \rceil + \sum_{i=1}^{\log_2 b} \lceil v/(2^i a) \rceil$	$\lceil v/p \rceil (n^2 - 1)$	L = R	L < R
T	$\sum_{i=1}^{\log_2 a} \lceil vn/(2^i b) \rceil + \sum_{i=1}^{\log_2 b} \lceil v/(2^i a) \rceil$	—	L > R	L > R

Note. M, A, and T represent the number of multiplications, additions, and inter-PE transfers, respectively; c is an integer constant.

are kept constant and γ is doubled (α is halved), the number of transfers decreases. However, if the number of vectors, v, is not a multiple of 2γ, then more computation per PE is required when γ is doubled; otherwise, the same amount of computation takes place. In the former case, the γ value that can provide the best performance is dependent on the trade-off between the increase in computation time and the reduction in communication time. In the latter case, it would be preferable to double the number of groups (i.e., double γ).

7. CHOOSING AN OPTIMAL ALGORITHM

Results from the previous sections that relate different algorithm data-parallel approaches include the following. Section 6 presents the complexity of the algorithm proposed in this paper in terms of v, n, γ, α, and β. The parameters v and n are input data parameters and γ, α, and β are logical system configuration parameters used by the generalized method. Section 4 and Subsection 5.2 describe the structure of the algorithm when certain restrictions are placed on γ, α, and β. The uncoupled method applies when $\alpha * \beta = 1$. When $\gamma = 1$, $\alpha * \beta = p$, and $\beta \leq n$, the coupled method is used. Subsection 5.2 showed that, when using the coupled method and γ is fixed, it is best to make α as small as possible, but keep β less than or equal to n. Subsection 5.3 made a strong argument for partitioning the problem in certain situations. It demonstrated that if $\gamma = a$, $\alpha = 1$, and $\beta = n$, when $v = ca$, it is better to use a groups of $1 \times n$ PEs than 1 group of $a \times n$ PEs. Finally, Section 6 concludes that the logical configuration $2\gamma \times \alpha/2 \times \beta$ (when $\alpha/2$ is an integer) will outperform the logical configuration $\gamma \times \alpha \times \beta$ when 2γ divides v: another argument for partitioning the problem.

Now consider the many possible ways to solve the MQF problem by the algorithms presented in this paper, given v, n, and p. One can use any variant of the generalized method (e.g., the uncoupled method, the single group coupled method, the multiple group method) or a combined approach. A combined approach may use a different method to compute the MQF problem for each different subset of steering vectors. For example, if $v > p$, then the best algorithm *may* use the uncoupled method for $v - (v \bmod p)$ vectors and the single group coupled method for $v \bmod p$ vectors. The combined approach permits any combination of methods, each being used to compute the MQF problem for a different subset of steering vectors. The total number of complex multiplications, complex additions, and inter-PE transfers for any algorithm can be computed from Table IV. The relative cost of a complex multiplication, complex addition, and inter-PE transfer for a given system can be obtained by experimentation and used to determine the optimal algorithm. Trade-offs involved in changing the logical system configuration, such as those mentioned earlier in this section, can be used to limit the number of possible *algorithm* choices to solve the MQF problem for a given v, n, and p.

8. EXPERIMENTAL STUDIES

The goal of this section is to validate experimentally many of the theoretical results found earlier. Both the uncoupled and coupled methods for solving MQFs have been implemented on the 16-PE small-scale PASM prototype, the 64-PE nCUBE 2, and the 16K-PE MasPar MP-1. The implementations assume that a, b, and p are powers of 2.

Figure 9 shows that for a logical $a \times b$ machine configuration when using the single group coupled method, the communication overhead for both the PASM prototype (Fig. 9a) and the nCUBE (Fig. 9b) decreases as b increases. The same is true for the MasPar (not shown), where communication time nearly doubled when the logical configuration was changed from 64×128 to 128×64. These experimental results confirm the conclusion pre-

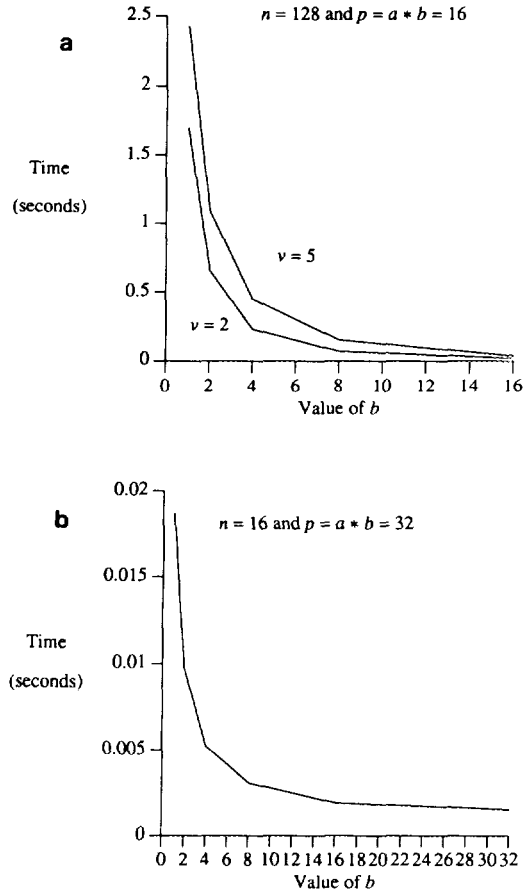

FIG. 9. Communication cost for various $p = a * b$ logical configurations for (a) the PASM prototype for $n = 128$, $p = 16$, and $v = 2$ and 5, and (b) the nCUBE 2 for $n = 16$, $p = 32$, and $v = 32$.

sented in Subsection 5.2 that communication overhead is minimized when a is chosen as small as possible (i.e., a and b being powers of 2 and $p = a * b$). Consequently, all the following experiments using the coupled method chose a to be as small as possible.

The results of executing the uncoupled and coupled methods on the PASM prototype when transfer costs are minimal compared to computation costs are shown in Fig. 10a. The algorithms were executed in mixed-mode: the floating-point computation was executed in MIMD mode and all other computation was done in SIMD mode. The difference between the performance of the uncoupled and coupled methods on the PASM prototype when p divides v is not significant. This is because the current implementation of the PASM prototype does floating-point operations by software emulation, so the total execution time is dominated by the complex operation computation time. The execution time for transfer-add operations represents only a very small portion of the entire execution time. These results corroborate the mathematical derivations in Table V showing that the uncoupled method outperforms the coupled method whenever $v = cp$, even when communication costs are low. The reduction in communication time by partitioning 16 PEs into several independent groups (e.g., two groups of 1×8 PEs, four groups of 1×4 PEs) is relatively insignificant when compared to the computation time of the complex operations. Consequently, only results from the single group coupled methods are reported for PASM for the performance of the coupled method.

Figure 10b shows the results of executing the uncoupled and single group coupled methods on the nCUBE 2, using 16, 32, and 64 PEs for a varying number (v) of steering vectors of size $n = 16$. The logical system configuration ($a \times b$) used to generate the graph for 16 PEs was 1×16, for 32 PEs was 2×16, and for 64 PEs was 4×16 (recall that b should be less than or equal to n).

The experimental results validate the theoretical conclusions summarized in Tables III and V. Table V says that whenever $v = cp$ the uncoupled method outperforms the single group coupled method. Figure 10b verifies this for $p = 16$, 32, and 64. Furthermore, when p does not divide v, the choice of method depends on the relationship between communication and computation costs. For example, as shown in Fig. 10b, when $p = 32$, the uncoupled method outperforms the single group coupled method for $v = 24$, 32, 48, 56, 64, 80, 88, and 96. The

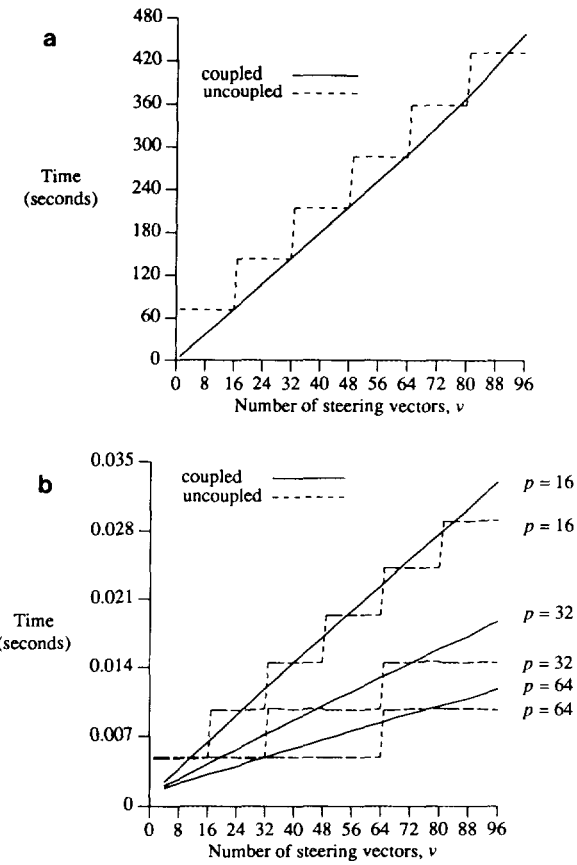

FIG. 10. Execution time of the uncoupled and coupled data-parallel methods for (a) the PASM prototype for $n = 128$ and $p = 16$, and (b) the nCUBE 2 for $n = 16$ and $p = 16$, 32, and 64.

exact points of intersection depend on the communication and computation cost relationship.

Now consider the computational complexities shown in Table III. When $v = ca$, the table shows that "a" groups of $1 \times n$ PEs will outperform one group of $p = a * n$ PEs. Let $v = 80$, $a = 4$, and $n = 16$. From Fig. 10b, one group of $64 = 4 * 16$ PEs takes .01 s. To compute $v = 80$, it would take four groups of 1×16 PEs as long as it takes one group of 1×16 PEs to compute 20 steering vectors. By looking at the graph for the single group coupled method using $p = 16$ PEs for $v = 20$ vectors, one can deduce that four groups of 1×16 PEs take .0077 s for $v = 80$, which is less than .01 s. Thus, in this case, the multiple group method outperforms the single group method, as predicted by Table III.

The combined approach, discussed in Section 7, can be used to compute the MQF problem for v steering vectors. For instance, the combined approach may consist of the following two steps: (1) use the uncoupled method for the first $(\lfloor v/p \rfloor * p)$ vectors and (2) compute the remaining $(v - \lfloor v/p \rfloor * p)$ vectors by the single group coupled method. For example, if $v = 80$ and $p = 64$, then step 1 would compute 64 vectors by the uncoupled method (.0049 s), and step 2 would compute 16 vectors by the coupled method (.0032 s). By this combined method, the MQF problem for $v = 80$ takes .0081 s on the nCUBE 2. The coupled and uncoupled methods take .01 s and .0097 s, respectively. Therefore, the combined method outperforms the single group coupled and uncoupled methods. Consequently, when choosing an optimal algorithm for a particular set of v, n, and p values, different combined approaches may need to be considered.

The MasPar MP-1 results in Fig. 11 show the number of steering vectors versus execution time for the single group coupled and uncoupled methods. It also graphs the time spent doing communication for the coupled method.

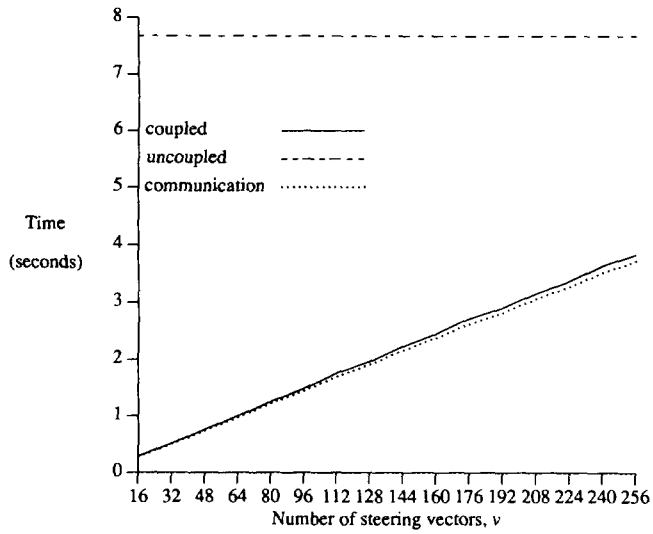

FIG. 11. Execution time of the uncoupled and coupled data-parallel methods for the MasPar MP-1 for $n = 128$ and $p = 16{,}384$.

Because the data is distributed across a large number of PEs ($p = n^2 = 128 * 128$), the computation per PE is negligible compared to the communication time. Despite this fact, the coupled method greatly outperforms the uncoupled method for the number of vectors tested. If the coupled method line (solid line) was extrapolated until it intersected the uncoupled method line (dashed line), the value of v would be approximately 516 at the point of intersection. For $v < 516$, the added computation cost of the uncoupled method outweighs the communication cost of the coupled method. This shows that when massive parallelism is available to implement the MQF problem ($p \gg v$), the more sophisticated coupled method is needed to exploit the available parallelism, despite incurring high communication overhead. In contrast, when $516 < v \leq 16{,}384$, the uncoupled method using only v PEs will outperform the single group coupled method using 16,384 PEs. This is the communication overhead price that is paid for the "fine grain parallelism" that characterizes the coupled method.

9. RELATED WORK

Adaptive filter theory [29] provides a theoretical background for implementing an adaptive beamformer with MVDR and a serial algorithm for implementing this problem can be found in [42]. To date, no related work has been found that examines the problem of determining parallel implementations for computing multiple quadratic forms for the same matrix (the type of computation in the MVDR problem). However, many publications exist that examine the more general problem of parallel implementations of matrix multiplication (e.g., [9, 16, 18, 24, 36]).

In [36], different parallel algorithms for matrix multiplication are discussed. Its treatment covers data layout on a logical mesh of processors (including the $p = n$ and $p = a * b$ general case). However, time complexities and arithmetic operation counts are not provided.

An overview of parallel matrix multiplication algorithms based on different processor interconnection topologies, such as the hypercube, mesh, and perfect shuffle networks, is given in [18]. Concerning the hypercube topology, that paper presents some parallel matrix multiplication algorithms for $p = n^3$ and $p = n^2$ cases. The computational complexity of the parallel algorithms is $O(\log_2 n)$ when $p = n^3$ and $O(n)$ when $p = n^2$.

Several papers (e.g., [9, 16, 24]) have been published that discuss performing matrix multiplication on hypercube machines by using a logical mesh of processors. In [24] it is shown that the optimal performance of matrix multiplication on the Cosmic Cube is achieved (in terms of communication overhead and load balancing) by decomposing the problem into square subblocks. Similarly, [16] discusses partitioning the matrix and the impact of communication overhead when performing matrix multi-

plication problem on an nCUBE [28]. The work presented in [9] is an extension of [24] in that it considers the restrictive condition of only having nearest-neighbor one-to-one communication.

The publications described above address the problem of multiplying together two- or three-dimensional matrices. Unlike previous work, this paper deals with the computation of *multiple* quadratic forms and discusses the impact of the two primary aspects of system configuration on the computation: interprocessor communication and partitionability. Trade-offs among various parallel implementations are also addressed in this paper. Furthermore, it is presented that, for a given problem, a combined uncoupled/coupled approach could achieve the optimal scalability for this computation. Consequently, the results set forth in the above publications are not directly applicable to the MQF problem discussed here.

10. SUMMARY

Several data-parallel implementations of the computationally intensive task of computing the MQF problem have been examined. Trade-offs among the implementations for various data-size/machine-size ratios are categorized in terms of complex arithmetic operation counts, communication overhead, and memory storage requirements. The results showed that when p (the number of PEs) divides v (the number of steering vectors), the uncoupled data-parallel method is the optimal parallel implementation and a speedup of p on the number of complex multiplications and complex additions can be achieved. However, when v is not a multiple of p, a combined approach using both the uncoupled and coupled data-parallel methods should be considered. For both the uncoupled and coupled data-parallel methods, the advantages and disadvantages of executing the different sections of the algorithms in SIMD, MIMD, and mixed-mode were discussed. In addition, trade-offs between the "single-group" and "multiple-group" decomposition for the coupled method were presented.

This research can be directly applied to SIMD, MIMD, and mixed-mode parallel machines interconnected with either the multistage cube or the hypercube interconnection network topology. This work can be extended to other topologies (e.g., mesh-connected). The experiments performed on the MasPar MP-1 (SIMD), nCUBE 2 (MIMD), and PASM (mixed-mode) prototype were used to validate some of the analytically derived relationships among input data and logical system parameters.

Choosing an optimal algorithm for an MQF problem with a given n (the size of a steering vector) and v is machine and p dependent. Therefore, by having a set of algorithms that perform the MQF problem efficiently for various input data parameters (e.g., n, v) and system parameters (e.g., p, communication time, complex addition time, complex multiplication time, mode of parallelism supported), an automatic algorithm selection methodology that may combine several approaches can be implemented using the analysis established. This analysis of the MQF problem that has been presented and experimentally explored supports the viability of such an automatic method that can be developed for a variety of applications.

ACKNOWLEDGMENT

The authors thank Professor Michael Zoltowski of Purdue University for his assistance. A preliminary version of portions of this material was presented at the *1993 International Conference on Parallel Processing*.

REFERENCES

1. Adams, G. B., III, and Siegel, H. J. The extra stage cube: A fault-tolerant interconnection network for supersystems. *IEEE Trans. Comput.* **C-31,** 5 (May 1982), 443–454.
2. Arlauskas, R. iPSC/2 system: A second generation hypercube. *Third Conference on Hypercube Computers and Applications*, Jan. 1988, pp. 38–42.
3. Armstrong, J. B., Watson, D. W., and Siegel, H. J. Software issues for the PASM parallel processing system. In Kowalik, J. S. (Ed.). *Software for Parallel Computation*. Springer-Verlag, Berlin, 1993, pp. 134–148.
4. Batcher, K. E. The flip network in STARAN. *1976 International Conference on Parallel Processing*, Aug. 1976, pp. 65–71.
5. Batcher, K. E. STARAN series E. *1977 International Conference on Parallel Processing*, Aug. 1977, pp. 140–143.
6. Batcher, K. E. Design of a massively parallel processor. *IEEE Trans. Comput.* **C-29,** 9 (Sept. 1980), 836–844.
7. Batcher, K. E. Bit serial parallel processing systems. *IEEE Trans. Comput.* **C-31,** 5 (May 1982), 377–384.
8. Berg, T. B., Kim, S. D., and Siegel, H. J. Limitations imposed on mixed-mode performance of optimized phases due to temporal juxtaposition. *J. Parallel Distribut. Comput.* **13** (Oct. 1991), 154–169.
9. Berntsen, J. Communication efficient matrix multiplication on hypercubes. *Parallel Comput.* **12,** 3 (1989), 335–342.
10. Berg, T. B., and Siegel, H. J. Instruction execution trade-offs for SIMD vs. MIMD vs. mixed-mode parallelism. *Fifth International Parallel Processing Symposium*, May 1991, pp. 301–308.
11. Blank, T. The MasPar MP-1 architecture. *IEEE Compcon*, Feb. 1990, pp. 20–24.
12. Blelloch, G. E. *Vector Models for Data-Parallel Computing*. MIT Press, Cambridge, MA, 1990.
13. Bouknight, W. J., Denenberg, S. A., McIntyre, D. E., Randall, J. M., Sameh, A. H., and Slotnick, D. L. The Illiac IV system. *Proc. IEEE* **60,** 4 (Apr. 1972), 369–388.
14. Bronson, E. C., Casavant, T. L., and Jamieson, L. H. Experimental application-driven architecture analysis of an SIMD/MIMD parallel processing system. *IEEE Trans. Parallel Distribut. Systems* **1,** 2 (Apr. 1990), 195–205.
15. Choi, J., Dongarra, J. J., Pozo, R., and Walker, D. W. ScaLAPACK: A scalable linear algebra library for distributed memory concurrent computers. *Fourth Symposium on the Frontiers of Massively Parallel Computation*, Oct. 1992, pp. 120–127.
16. Cherkassky, V., and Smith, R. Efficient mapping and implementation of matrix algorithms on a hypercube. *J. Supercomput.* **2** (1988), 7–27.
17. Crowther, W., Goodhue, J., Thomas, R., Milliken, W., and Blackadar, T. Performance measurements on a 128-node butterfly paral-

lel processor. *1985 International Conference on Parallel Processing*, Aug. 1985, pp. 531–540.

18. Dekel, E., Nassimi, D., and Sahni, S. Parallel matrix and graph algorithms. *SIAM J. Comput.* **10,** 4 (1981), 657–675.
19. Duclos, P., Boeri, F., Auguin, M., and Giraudon, G. Image processing on a SIMD/SPMD architecture: OPSILA. *International Conference on Pattern Recognition*, Nov. 1988, pp. 430–433.
20. Fineberg, S. A., Casavant, T. L., Schwederski, T., and Siegel, H. J. Non-deterministic instruction time experiments on the PASM system prototype. *1988 International Conference on Parallel Processing*, Aug. 1988, pp. 444–451.
21. Fineberg, S. A., Casavant, T. L., and Siegel, H. J. Experimental analysis of a mixed-mode parallel architecture using bitonic sequence sorting. *J. Parallel Distribut. Comput.* **11** (Mar. 1991), 239–251.
22. Flynn, M. J. Very high-speed computing systems. *Proc. IEEE* **54,** 12 (Dec. 1966), 1901–1909.
23. Fox, G. C., Johnson, M. A., Lyzenga, G. A., Otto, S. W., Salmon, J. K., and Walker, D. W. *Solving Problems on Concurrent Processors, Volume 1*. Prentice-Hall, Englewood Cliffs, NJ, 1988.
24. Fox, G. C., Otto, S. W., and Hey, A. J. G. Matrix algorithms on a hypercube I: Matrix multiplication. *Parallel Comput.* **4,** 1 (1987), 17–31.
25. Fountain, T. J. CLIP4: Progress report. In Duff, M. J. B., and Levialdi, S. (Eds.). *Languages and Architectures for Image Processing*. Academic Press, London, 1981, pp. 281–291.
26. Fortune, S., and Wyllie, J. Parallelism in random access machines. *Tenth ACM Symposium on Theory of Computing*, May 1978, pp. 114–118.
27. Gottlieb, A., Grishman, R., Kruskal, C. P., McAuliffe, K. P., Rudolph, L., and Snir, M. The NYU Ultracomputer—Designing an MIMD shared-memory parallel computer. *IEEE Trans. Comput.* **C-32,** 2 (Feb. 1983), 175–189.
28. Hayes, J. P., and Mudge, T. Hypercube supercomputers. *Proc. IEEE* **77,** 12 (Dec. 1989), 1829–1841.
29. Haykin, S. *Adaptive Filter Theory*. Prentice-Hall, Englewood Cliffs, NJ, 1986.
30. Hunt, D. J. AMT DAP—A processor array in a workstation environment. *Comput. Systems Sci. Eng.* **4,** 2 (Apr. 1989), 107–114.
31. Jamieson, L. H. Characterizing parallel algorithms. In Jamieson, L. H., Gannon, D. B., and Douglass, R. J. (Eds.). *The Characteristics of Parallel Algorithms*. MIT Press, Cambridge, MA, 1987, pp. 65–100.
32. Kim, S. D., Nichols, M. A., and Siegel, H. J. Modeling overlapped operation between the control unit and processing elements in an SIMD machine. *J. Parallel Distribut. Comput. (Special Issue on Modeling of Parallel Computers)* **12** (Aug. 1991), 329–342.
33. Kuck, D. J., Davidson, E. S., Lawrie, D. H., and Sameh, E. H. Parallel supercomputing today and the Cedar approach. *Science* **231** (Feb. 1986), 967–974.
34. Lawrie, D. H. Access and alignment of data in an array processor. *IEEE Trans. Comput.* **C-24,** 12 (Dec. 1975), 1145–1155.
35. Lipovski, G. J., and Malek, M. *Parallel Computing: Theory and Comparisons*. Wiley, New York, 1987.
36. Modi, J. J. *Parallel Algorithms and Matrix Computations*. Oxford Univ. Press, New York, 1988.
37. Pancake, C. M. Software support for parallel computing: Where are we headed? *Comm. ACM* **34,** 11 (Nov. 1991), 53–64.
38. Patel, J. H. Performance of processor–memory interconnections for multiprocessors. *IEEE Trans. Comput.* **C-30,** 10 (Oct. 1981), 771–780.
39. Pease III, M. C. The indirect binary n-cube microprocessor array. *IEEE Trans. Comput.* **C-26,** 5 (May 1977), 458–473.
40. Pfister, G. F., Brantley, W. C., George, D. A., Harvey, S. L., Kleinfelder, W. J., McAuliffe, K. P., Melton, E. A., Norton, V. A., and Weiss, J. The IBM Research Parallel Processor Prototype (RP3): Introduction and architecture. *1985 International Conference on Parallel Processing*, Aug. 1985, pp. 764–771.
41. Philippsen, M., Warschko, T., Tichy, W., and Herter, C. Project Triton: Towards improved programmability of parallel machines. *26th Hawaii International Conference on System Sciences*, Jan. 1993, pp. 192–201.
42. Schreiber, R. Implementation of adaptive array algorithms. *IEEE Trans. Acoust. Speech Signal Process.* **ASSP-34,** 5 (1986), 1038–1045.
43. Siegel, H. J., Armstrong, J. B., and Watson, D. W. Mapping computer-vision-related tasks onto reconfigurable parallel-processing systems. *Computer* **25,** 2 (Feb. 1992), 54–63.
44. Siegel, H. J. *Interconnection Networks for Large-Scale Parallel Processing: Theory and Case Studies, Second Edition*. McGraw-Hill, New York, 1990.
45. Siegel, H. J., Nation, W. G., and Allemang, M. D. The organization of the PASM parallel processing system. *1990 Parallel Computing Workshop*, sponsored by the Department of Computer and Information Science at The Ohio State University, 1990, pp. 1–12.
46. Siegel, H. J., Siegel, L. J., Kemmerer, F. C., Mueller, P. T., Jr., Smalley, H. E., Jr., and Smith, S. D. PASM: A partitionable SIMD/MIMD system for image processing and pattern recognition. *IEEE Trans. Comput.* **C-30,** 12 (Dec. 1981), 934–947.
47. Siegel, L. J., Siegel, H. J., and Swain, P. H. Performance measures for evaluating algorithms for SIMD machines. *IEEE Trans. Software Engrg.* **SE-8,** 4 (July 1982), 319–331.
48. Siegel, H. J., Schwederski, T., Kuehn, J. T., and Davis, N. J., IV. An overview of the PASM parallel processing system. In Gajski, D. D., Milutinovic, V. M., Siegel, H. J., and Furht, B. P. (Eds.). *Computer Architecture*. IEEE Computer Society Press, Washington, DC, 1987, pp. 387–407.
49. Stone, H. S. Parallel computers. In Stone, H. S. (Ed.). *Introduction to Computer Architecture, Second Edition*. Science Research Associates, Inc., Chicago, IL, 1980, pp. 363–425.
50. Strang, G. *Introduction to Applied Mathematics*. Wellesley-Cambridge Press, Wellesley, MA, 1986.
51. Thinking Machines Corporation. *CM-5: Technical Summary*. Thinking Machines Corporation, Cambridge, MA, Jan. 1992.
52. Thanawastien, S., and Nelson, V. P. Interference analysis of shuffle/exchange networks. *IEEE Trans. Comput.* **C-30,** 8 (Aug. 1981), 545–556.
53. Tucker, L. W., and Robertson, G. G. Architecture and applications of the Connection Machine. *Computer* **21,** 8 (Aug. 1988), 26–38.
54. Wang, M. C., Nation, W. G., Armstrong, J. B., Siegel, H. J., Kim, S. D., Nichols, M. A., and Gherrity, M. Computing multiple quadratic forms for a minimum variance distortionless response adaptive beamformer using parallelism: Analyses and experiments. Purdue University Tech. Rep. TR-EE 93-20, School of Electrical Engineering, Purdue University, W. Lafayette, IN, May 1993.
55. Wu, C.-L., and Feng, T. Y. On a class of multistage interconnection networks. *IEEE Trans. Comput.* **C-29,** 8 (Aug. 1980), 694–702.

MU-CHENG WANG received the B.S. degree in electrical engineering from the Cheng-Kung University, Taiwan, Republic of China, in 1982, the M.S. degree in computer science from the University of Iowa in 1988, and the Ph.D. degree in electrical engineering from Purdue University in 1992. He is currently an adjunct assistant professor in the Computer Science Department at Queens College of C.U.N.Y. His research interests include parallel and distributed systems, performance evaluation, heterogeneous computing, and computer architecture. He is a member of the IEEE Computer Society and Tau Beta Pi.

WAYNE G. NATION received the M.S.E.E. and Ph.D. degrees from Purdue University, West Lafayette, IN, in 1986 and 1991. Since 1991, he has been with the IBM Corporation and is currently with the AS/4000 Division in Rochester, MN. In Spring 1992, he was on the adjunct faculty at the State University of New York–Binghamton. He has coauthored over 20 technical papers related to his research interests in interconnection networks, scalable and partitionable parallel processing systems, parallel algorithms, and the design of the PASM prototype.

JAMES B. ARMSTRONG is a Ph.D. student in the School of Electrical Engineering at Purdue University. He received the B.S. degree in electrical engineering and computer science, and a certificate in management systems, from Princeton University in 1988. He has received the M.S.E.E. degree from Purdue University in 1989. He has coauthored several publications on parallel processing, is the student manager of the EE School Parallel Processing Laboratory, and is a NASA Graduate Student Researchers' Fellow. His current research is in the area of fault-tolerant computing and software issues for heterogeneous computing systems. He is a member of the Sigma Xi, Eta Kappa Nu, and Tau Beta Pi honorary societies. He is also a student member of the IEEE Computer Society and the ACM.

H. J. SIEGEL is a Professor and Coordinator of the Parallel Processing Laboratory in the School of Electrical Engineering at Purdue. He received two B.S. degrees from MIT (1972), and the M.A. (1974), M.S.E. (1974), and Ph.D. (1977) degrees from Princeton. He has coauthored over 150 technical papers and authored one book. His current research interests include interconnection networks, heterogeneous computing, and the use and design of the PASM reconfigurable mixed-mode parallel system. He is a Fellow of the IEEE and was Coeditor-in-Chief of the *Journal of Parallel and Distributed Computing* (1989–1991). He is currently on the Editorial Board of the *IEEE Transactions on Parallel and Distributed Systems* and is Program Chair of the 8th International Parallel Processing Symposium (1994).

Received April 1, 1993; revised September 26, 1993; accepted October 26, 1993

SHIN-DUG KIM received the B.S. degree in electronic engineering from Yonsei University, Seoul, Korea, in 1982, and the M.S. degree in electrical engineering from the University of Oklahoma in 1987. In 1991, he received the Ph.D. degree at Purdue University in electrical engineering. He is currently an assistant professor of computer engineering at the KwangWoon University, Seoul, Korea. His research interests include parallel system architectures, parallel algorithm design, scalable throughput/speedup systems, and virtual reality. He is a member of the IEEE Computer Society.

MARK A. NICHOLS received the B.S. degree in 1985 with a triple major of electrical engineering, computer engineering, and mathematics (computer science) from Carnegie Mellon University. In 1986 he received the M.S.E.E. degree from the Georgia Institute of Technology and in 1991 completed the Ph.D. degree at Purdue University in electrical engineering. He is currently with NCR, San Diego, CA, where he examines architectural issues regarding parallel processing systems targeted for data-intensive commercial applications such as parallel relational database management. His research interests include parallel architecture modeling, parallel language/compiler design, and interconnection networks. Dr. Nichols is a member of the IEEE Computer Society and the ACM.

MICHAEL GHERRITY is a computer scientist at the Naval Command, Control, and Ocean Surveillance Center, RTD&E Division, and a graduate student at the University of California, San Diego (UCSD). He received the B.S. degree at MIT (1981) and the M.S. degree at UCSD (1988). His current research interests include parallel processing and artificial intelligence.