5-1-1993

COMPUTING MULTIPLE QUADRATIC FORMS FOR A MINIMUM VARIANCE DISTORTIONLESS RESPONSE ADAPTIVE BEAMFORMER USING PARALLELISM: ANALYSES AND EXPERIMENTS

Mu-Cheng Wang
Purdue University School of Electrical Engineering

Wayne G. Nation
Purdue University School of Electrical Engineering

James B. Armstrong
Purdue University School of Electrical Engineering

Howard Jay Siegel
Purdue University School of Electrical Engineering

Shin-Dug Kim
Purdue University School of Electrical Engineering

Computing Multiple Quadratic Forms for a Minimum Variance Distortionless Response Adaptive Beamformer Using Parallelism: Analyses and Experiments

Mu-Cheng Wang
Wayne G. Nation
James B. Armstrong
Howard Jay Siegel
Shin-Dug Kim
Mark A. Nichols
Michael Gherrity

TR-EE 93-20
May 1993

School of Electrical Engineering
Purdue University
West Lafayette, Indiana 47907-1285

COMPUTING MULTIPLE QUADRATIC FORMS FOR A MINIMUM VARIANCE DISTORTIONLESS RESPONSE ADAPTIVE BEAMFORMER USING PARALLELISM: ANALYSES AND EXPERIMENTS

Mu-Cheng Wang[†*]
Wayne G. Nation[†*]
James B. Armstrong[†*]
Howard Jay Siegel[†*]
Shin-Dug Kim[†*]
Mark A. Nichols[†*]
Michael Gherrity[‡]

[†] Parallel Processing Laboratory
School of Electrical Engineering
Purdue University
West Lafayette, IN 47907-1285 USA

[‡] Superconcurrency Research Team
Naval Ocean Systems Center, Code 421
San Diego, CA 92152-5000 USA

May 1993

Purdue University
School of Electrical Engineering
Technical Report TR-EE 93-20

[*] This research was supported by Rome Laboratory under contract number 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 number N00014-90-J-1937, by the National Science Foundation under grant number CDA-9015696, and by National Aeronautical and Space Administration under grant number NGT-50961.

Wayne G. Nation. Shin-Dug Kim, and Mark A. Nichols are currently with IBM Corporation. Rochester. MN, KwangWoon University, Seoul. Korea, and NCR Corporation, San Diego, CA, respectively.

Abstract

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 proposed 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 from experiments on the MasPar MP-1 (SIMD), nCUBE 2 (MIMD), and PASM (mixed-mode) prototype.

Keywords: data-parallel algorithms, MasPar MP-1, multiple quadratic forms, nCUBE 2, PASM, scalable algorithms

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 [SiS82]. 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 [ChD92]. 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 (MQFs) that are part of an adaptive beamformer calculation with minimum variance distortionless response (MVDR) [Hay86]. 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 non-interacting subtasks is studied. The analytical results show that none of the algorithms presented in this paper is best for all data-sizelmachine-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 proposed 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., [Pan91]). 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.

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 s-vector, or steering vector, be an $n \times 1$ vector of complex numbers and $\underline{v}$ be the total number of s-vectors. Define $\underline{M}$ 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 q-th s-vector is denoted by $\underline{s_q}$, for $0 \le q < v$. Element m of the s-vector q is denoted $\underline{s_q(m)}$, 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:

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

For easy reference, Table 1 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, if M is a positive definite matrix, the quadratic $P(x) = (1/2)x^T Mx - x^T b$ is minimized at the point where $Mx = b$ and the minimum value is $P(M^{-1}b) = -(1/2)b^T M^{-1} b$. The parallel algorithms proposed here can be easily generalized to compute the equation $x^T My$ that appears in the fundamental variational principles of physics [Str86].

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.

Notation	Meaning
$M(i,j)$	the element of matrix M in row i and column j
n	the number of rows and columns of mamx M
v	total number of s-vectors
P	number of PEs in a parallel system
$s_q(m)$	the element m of the s-vector q
r_q	$s_q^H M$
w_q	$s_q^H M s_q$ or $r_q s_q$

Table 1: Summary of notation used throughout the paper.

3. Parallel System Model

The model of a parallel system that is assumed here 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 [Bla90] and nCUBE 2 [HaM89], as well as in the PASM prototype [FiC91, SiS87]. The proposed data-parallel algorithms can be implemented on a SIMD, MIMD, or mixed-mode distributed memory machine. These three models of parallel machines are overviewed below.

An SIMD [Fly66] machine is typically composed of a control unit (CU), a set of PEs, and an interconnection network, as shown in Figure 1. In an SIMD machine, the enabled PEs receive and synchronously execute common instructions that are broadcast from the CU; thus, there is a single instruction stream. The PEs fetch data from their memory modules; thus, there are multiple data streams. The interconnection network allows PEs to communicate among themselves and exchange data. Examples of SIMD machines that have been constructed include CLIP4 [Fou81], CM-2 [TuR88], DAP [Hun89], Illiac IV [BoD72], MasPar [Bla90], MPP [Bat80, Bat82], and STARAN [Bat74, Bat77].

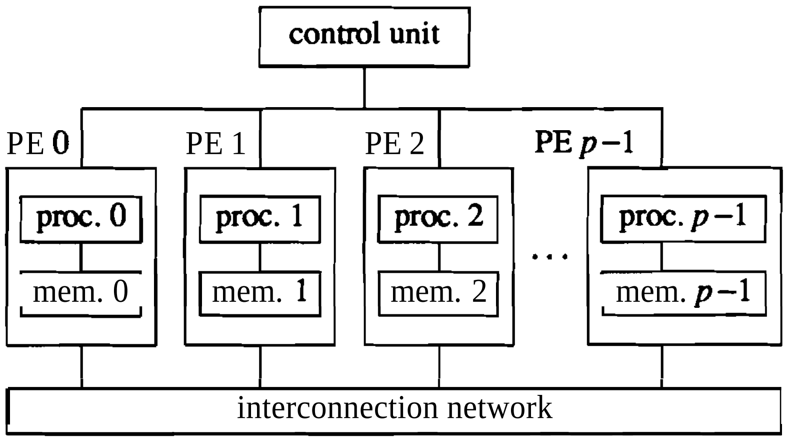


Figure 1: SIMD machine model with p PEs.

An MIMD machine consists of a set of PEs and an interconnection network, as shown in Figure 2. 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 [Fly66]. 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 [CrG85], CM-5 [PaS92], IBM RP3 [PfB85], Intel iPSC Cube [Arl88], and nCUBE [HaM89].

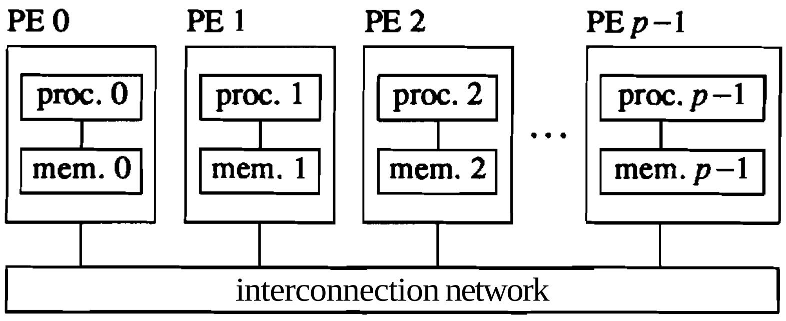


Figure 2: MIMD machine model with p PEs.

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 [LiM87], 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 [DuB88], 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 [ArW93, SiN90, SiS81, SiS87] can dynamically switch between SIMD and full MIMD modes of parallelism at instruction-level granularity with negligible overhead. Triton/1 [PhW93] is capable of mixed-mode parallelism at the instruction-level. Unlike PASM and TRAC, Triton/1 is not partitionable. The design can support up to 4096 processors.

PASM is reconfigurable along three dimensions: modes of parallelism (SIMD/MIMD) as described above, partitionability, and inter-processor connections. It can be dynamically partitioned into independent or communicating submachines of various sizes, each having the same characteristics as the original machine. PASM uses a flexible multistage cube type network [Sie90], 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 [BrC90].

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 [BeK91, BeS91, FiC91, SiA92]. Computational characteristics of algorithms affect the choice of the best mode for executing each section of code [Jam87]. A general discussion of these trade-offs is beyond the scope of this paper.

Furthermore, no particular interconnection network is assumed for the proposed algorithms, but the multistage cube and hypercube (single-stage cube) networks [Sie90] are shown to be flexible interconnection networks that can perform the **PE-to-PE** permutations required by the MQF implementations without any conflicts (**i.e.**, without two 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 [Bat82]**, BBN Butterfly **[CrG85]**, Cedar **[KuD86]**, IBM RP3 **[PfB85]**, **Goodyear** Aerospace Corporation **STARAN [Bat77]**, PASM **[SiN90, SiS87]**, and NYU Ultracomputer **[GoG83]**. The class of multistage cube topologies includes: baseline **[WuF80]**, flip **[Bat76]**, generalized cube **[Sie90]**, indirect binary n-cube **[Pea77]** multistage shuffle-exchange **[ThN81]**, omega **[Law75]**, and SW-banyan ($S=F=2$, $L=n$) **[LiM87]** networks **[Sie90, WuF80]**. The hypercube interconnection topology has been implemented in the CM-2 **[TuR88]**, Intel **iPSC** cube **[Arl88]**, and **nCUBE [HaM89]**.

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 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.**, $v \neq c*p$, $\lceil v/p \rceil$ s-vectors will be assigned to each of v **mod** p **PEs**, and $\lfloor v/p \rfloor$ 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 wp.

Solving a quadratic form can be decomposed into two phases. Letting $*$ denote scalar **multiplication**, the phase (1) calculation is $r_q(y) = \sum_{x=0}^{n-1} s_q^H(x)*M(x,y)$, 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 $\lceil v/p \rceil$ 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 (synchronous) or MIMD (asynchronous) mode. If all PEs are operating synchronously, the computations being performed by the PEs can be overlapped with the CU's execution of control-flow instructions and/or any instruction common to all PEs. This is called CU/PE overlap [KiN91]. Also, during phase (1), each element of mamx M can be embedded as an immediate operand of an SIMD instruction broadcast from the control unit 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) [BeS91, FiC88].

If all PEs are operating asynchronously, the data-dependent variable-time instructions are better performed in MIMD mode than in SIMD mode [FiC88]. 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 &-tails and the characteristics of the data being processed.

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 two. Furthermore, the impact of partitioning the problem is discussed. The special cases of $p = n$ $(a = 1, b = n)$ and $p = n^2$ $(a = b = n)$ are studied first to help explain the general case.

There are two common data layouts of the M matrix and s vectors in memory, referred to in [FoJ88] as the block and scattered decompositions. For the MQF problem, the block decomposition method has contiguous enmes of the M matrix and s-vectors assigned to PEs in blocks. The scattered decomposition method has consecutive entries in the M mamx 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 two initially. Based on the description of the coupled implementation, the proposed method can be readily extended to cover the case when n is not a power of two. This case will be briefly discussed in Subsection 5.6.

5.2. When $p = n$

Consider the case when the number of PEs is equal to the number of elements in a s-vector, i.e., $p = n$. As shown in Figure 3, PE j stores the j-th column of the M matrix and the Hermitians of v s-vectors (s_q^H's, $0 \le q < v$). Given $v = 6$ and $p = n = 4$, Figure 4 illustrates how M is distributed across the **PEs**.

PE 0 memory	PE 1 memory		PE p−1 memory
$M(i, 0)$ $0 \le i < n$	$M(i, 1)$ $0 \le i < n$	...	$M(i,p-1)$ $0 \le i < n$
$s_q^H(j)$ $0 \le j < n$ $0 \le q < v$	$s_q^H(j)$ $0 \le j < n$ $0 \le q < v$		$s_q^H(j)$ $0 \le j < n$ $0 \le q < v$

Figure 3: Distribution of data onto p = n **PEs** for v steering vectors.

As in the uncoupled data-parallel method, the calculation of the MQF problem can be **decom**posed 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) **cal**culation is $w_q = \sum_{y=0}^{n-1} \sigma_q(y)$, where $\sigma_q(y) = r_q(y) * s_q(y)$. To form the intermediate result $r_q(j)$, for $0 \le q < v$, PE j will execute vn complex multiplications and v (n − 1) complex additions. All **PEs** involved can perform these computations in parallel (with PE j calculating $r_q(j)$). As shown in Figure 4, the **resulting** element in PE j is $r_q(j)$ for $0 \le q < v$.

In the second phase, PE j first computes $r_q(j) * s_q(j)$ for all q. Thus, v complex multiplications are performed, as shown in Figure 5. Similar to phase (1), all **PEs** involved can perform these multiplications at the same time. Next, the summation **is performed, i.e.,** $w_q = \sum_{j=0}^{n-1} \sigma_q(j)$. However, in this case, the elements to be totaled are stored in distinct **PEs**. Recursive doubling can be used to compute the sums [Sto80]. Performing interleaved **recursive** doubling procedures [SiA92, SiS81] to find simultaneously the sum of the elements of p distinct vectors is referred to here as a <u>p-**recursive** doubling</u>.

To help clarify the operations involved in the p-recursive doubling technique, Figure 6 illustrates how N = **4 PEs** would sum the elements for an arbitrary set of vectors. In this example, there are p = 6 vectors, $\underline{\alpha_\eta}$ for $0 \le \eta < 6$, each having N elements. If $\underline{\alpha_\eta^j}$ is the j-th 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 Figure 6, 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.

$$\underbrace{\overset{r_q(j)}{\left[\begin{array}{c|c|c|c} r_0(0) & r_0(1) & r_0(2) & r_0(3) \\ r_1(0) & r_1(1) & r_1(2) & r_1(3) \\ r_2(0) & r_2(1) & r_2(2) & r_2(3) \\ r_3(0) & r_3(1) & r_3(2) & r_3(3) \\ r_4(0) & r_4(1) & r_4(2) & r_4(3) \\ r_5(0) & r_5(1) & r_5(2) & r_5(3) \end{array}\right]}}_{\mathrm{PE}(0)\quad \mathrm{PE}(1)\quad \mathrm{PE}(2)\quad \mathrm{PE}(3)} = \underbrace{\overset{s_q^{\mathrm{H}}(j)}{\left[\begin{array}{cccc} s_0^{\mathrm{H}}(0) & s_0^{\mathrm{H}}(1) & s_0^{\mathrm{H}}(2) & s_0^{\mathrm{H}}(3) \\ s_1^{\mathrm{H}}(0) & s_1^{\mathrm{H}}(1) & s_1^{\mathrm{H}}(2) & s_1^{\mathrm{H}}(3) \\ s_2^{\mathrm{H}}(0) & s_2^{\mathrm{H}}(1) & s_2^{\mathrm{H}}(2) & s_2^{\mathrm{H}}(3) \\ s_3^{\mathrm{H}}(0) & s_3^{\mathrm{H}}(1) & s_3^{\mathrm{H}}(2) & s_3^{\mathrm{H}}(3) \\ s_4^{\mathrm{H}}(0) & s_4^{\mathrm{H}}(1) & s_4^{\mathrm{H}}(2) & s_4^{\mathrm{H}}(3) \\ s_5^{\mathrm{H}}(0) & s_5^{\mathrm{H}}(1) & s_5^{\mathrm{H}}(2) & s_5^{\mathrm{H}}(3) \end{array}\right]}}_{\text{in all PEs}} \times \underbrace{\overset{M}{\left[\begin{array}{c|c|c|c} M(0,0) & M(0,1) & M(0,2) & M(0,3) \\ M(1,0) & M(1,1) & M(1,2) & M(1,3) \\ M(2,0) & M(2,1) & M(2,2) & M(2,3) \\ M(3,0) & M(3,1) & M(3,2) & M(3.3) \end{array}\right]}}_{\mathrm{PE}(0)\quad \mathrm{PE}(1)\quad \mathrm{PE}(2)\quad \mathrm{PE}(3)}$$

Figure 4: Distribution of the matrices onto the PEs for $v = 6$ and $p = n = 4$, where $r_q(j) = \sum_{i=0}^{3} s_q^{\mathrm{H}}(i) * M(i,j)$

$$
\begin{array}{|c|c|c|c|}
\hline
r_0(0)*s_0(0) & r_0(1)*s_0(1) & r_0(2)*s_0(2) & r_0(3)*s_0(3) \\
r_1(0)*s_1(0) & r_1(1)*s_1(1) & r_1(2)*s_1(2) & r_1(3)*s_1(3) \\
r_2(0)*s_2(0) & r_2(1)*s_2(1) & r_2(2)*s_2(2) & r_2(3)*s_2(3) \\
r_3(0)*s_3(0) & r_3(1)*s_3(1) & r_3(2)*s_3(2) & r_3(3)*s_3(3) \\
r_4(0)*s_4(0) & r_4(1)*s_4(1) & r_4(2)*s_4(2) & r_4(3)*s_4(3) \\
r_5(0)*s_5(0) & r_5(1)*s_5(1) & r_5(2)*s_5(2) & r_5(3)*s_5(3) \\
\hline
\multicolumn{1}{c}{\text{PE 0}} & \multicolumn{1}{c}{\text{PE 1}} & \multicolumn{1}{c}{\text{PE 2}} & \multicolumn{1}{c}{\text{PE 3}}
\end{array}
=
\begin{array}{|c|c|c|c|}
\hline
r_0(0) & r_0(1) & r_0(2) & r_0(3) \\
r_1(0) & r_1(1) & r_1(2) & r_1(3) \\
r_2(0) & r_2(1) & r_2(2) & r_2(3) \\
r_3(0) & r_3(1) & r_3(2) & r_3(3) \\
r_4(0) & r_4(1) & r_4(2) & r_4(3) \\
r_5(0) & r_5(1) & r_5(2) & r_5(3) \\
\hline
\multicolumn{1}{c}{\text{PEO}} & \multicolumn{1}{c}{\text{PE 1}} & \multicolumn{1}{c}{\text{PE2}} & \multicolumn{1}{c}{\text{PE3}}
\end{array}
*
\begin{array}{|c|c|c|c|}
\hline
s_0(0) & s_0(1) & s_0(2) & s_0(3) \\
s_1(0) & s_1(1) & s_1(2) & s_1(3) \\
s_2(0) & s_2(1) & s_2(2) & s_2(3) \\
s_3(0) & s_3(1) & s_3(2) & s_3(3) \\
s_4(0) & s_4(1) & s_4(2) & s_4(3) \\
s_5(0) & s_5(1) & s_5(2) & s_5(3) \\
\hline
\multicolumn{1}{c}{\text{PEO}} & \multicolumn{1}{c}{\text{PE 1}} & \multicolumn{1}{c}{\text{PE2}} & \multicolumn{1}{c}{\text{PE3}}
\end{array}
$$

Figure 5: Distribution of $r_q(j) * s_q(j)$ onto PEs for v = 6 and $p = n = 4$ ($*$ represents element-wise scalar multiplication).

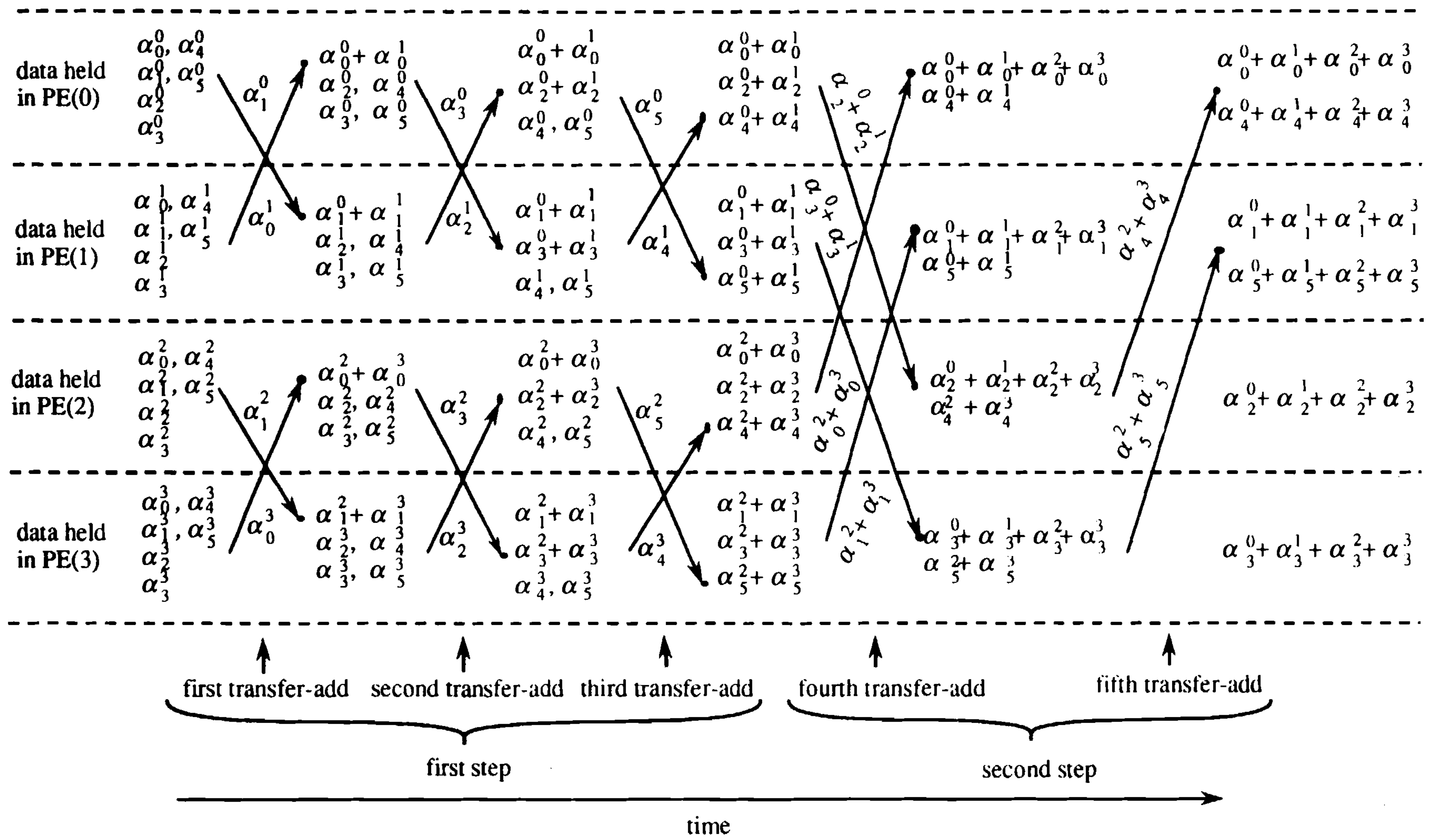


Figure 6: Using recursive doubling to sum simultaneously six vectors, α_η^j, for $0 \leq \eta < 6$ and $0 \leq j < 4$, on N = 4 PEs.

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 p-recursive doubling procedure is $\sum_{i=1}^{\log_2 N} \lceil \rho/2^i \rceil$. Figure 7 gives an algorithm to perform the p-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 [Sie90]. 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 [Sie90].

```
/* The following procedure will be performed on PE(j), 0 ≤ j < N. */
for i = 1 to $\log_2 N$ do (  /* step i */
    $k = j \oplus 2^{(i-1)}$;        /* the destination PE number */
    for $\beta = 0$ to $(\lfloor \rho/2^i \rfloor - 1)$ do (
            $\eta = (k \bmod 2^i) + \beta 2^i$;
            transfer local partial sum of vector $\eta$ to PE($k$);
            perform the complex addition with the received data and the
            corresponding local partial sum in PE($j$);
    }
    if $(\rho/2^i \ne \lfloor \rho/2^i \rfloor)$ {
            $\beta = \lfloor \rho/2^i \rfloor$;
            $\eta = (k \bmod 2^i) + \beta 2^i$;
            if $(j \,\&\, 2^{(i-1)} \ne 0)$ {
                transfer local partial sum of vector $\eta$ to PE($k$);
            ) else {
                perform the complex addition with the received data and the
                corresponding local partial sum in PE($j$);
                }
    }
}
```

Figure 7: An algorithm to perform the p-recursive doubling procedure.

In the coupled data-parallel method for p = n, $\sigma_q(j)$ is equivalent to α^j_η in the example above. Let w_q be the final sum of vector q, i.e., $w_q = \sum_{j=0}^{3} \sigma_q(j)$. Then, as shown in Figure 8, for the example above w_0 and w_4 are placed in PE 0, w_1 and w_5 are placed in PE 1, and w_2 and w_3 are placed in PE

2 and PE 3, respectively.

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

Figure 8: Distribution of w_q's onto p PEs for $v = 6$ and $p = n = 4$.

In summary, there are vn complex multiplications and $v(n-1)$ complex additions during the first phase, and the second phase requires v complex multiplications followed by $\sum_{i=1}^{\log_2 n} \lceil v/2^i \rceil$ transfer-adds. Thus, the computational complexity of the coupled implementation in this case is $v(n+1)$ complex multiplications, $v(n-1)$ complex additions, and $\sum_{i=1}^{\log_2 n} \lceil v/2^i \rceil$ transfer-adds. If $\sum_{i=1}^{\log_2 n} \lceil v/2^i \rceil$ transfer-adds can be decomposed into $\sum_{i=1}^{\log_2 n} \lceil v/2^i \rceil$ inter-PE transfers and $\sum_{i=1}^{\log_2 n} \lceil v/2^i \rceil$ complex additions, the computational complexity becomes $v(n+1)$ complex multiplications, $v(n-1)+\sum_{i=1}^{\log_2 n} \lceil v/2^i \rceil$ complex additions, and $\sum_{i=1}^{\log_2 n} \lceil v/2^i \rceil$ inter-PE transfers.

If n divides v (i.e., $v = cn$ and c is an integer), $\sum_{i=1}^{\log_2 n} \lceil v/2^i \rceil = \sum_{i=1}^{\log_2 n} (v/2^i) = (v/n)(n-1)$. Then, the coupled method requires $v(n+1)$ complex multiplications, $v(n-1)+(v/n)(n-1) = v(n-1/n)$ complex additions, and $(v/n)(n-1)$ transfer-adds. Comparing the parallel complexity to the serial complexity given in Section 2 shows that a speedup of $n = p$ is achieved on the number of complex multiplications and additions. This multiplicative factor of $n = p$ speedup on calculations comes with an overhead cost that is only an additive term of less than v inter-PE transfers.

Because $(v/2^i) \le \lceil v/2^i \rceil \le (v/2^i) + 1$ and $\sum_{i=1}^{\log_2 n} (v/2^i) = (v/n)(n-1)$,

$$(v/n)(n-1) \le \sum_{i=1}^{\log_2 n} \lceil v/2^i \rceil \le (v/n)(n-1) + \log_2 n.$$

Consequently, the number of complex additions and inter-PE transfers are bounded by $v(n-1)+(v/n)(n-1)+\log_2 n = vn-(v/n)+\log_2 n$ and $(v/n)(n-1)+\log_2 n$, respectively. If $v \ge n$, as with the case for the MVDR application, this implies that $\sum_{i=1}^{\log_2 n} (v/2^i) = (v/n)(n-1) \gg \log_2 n$. Hence, $\sum_{i=1}^{\log_2 n} \lceil v/2^i \rceil \simeq (v/n)(n-1)$. Therefore, a speedup of approximately $n = p$ on the number of complex multiplications and additions is achievable even when n does not divide v.

As an example, let $v = c*p = c*n$ and assume that the time to do a complex multiplication is ten times that required by a complex addition, and, on the average, the time to do an inter-PE transfer is the same as a complex multiplication. Then, the total speedup attained is:

$$\textbf{speedup} = \frac{\text{serial-time}}{\text{parallel-time}} = \frac{vn^2 - v + 10(vn^2 + vn)}{(vn - v/n) + 10(vn + v) + 10(v - v/n)}$$

$$= \frac{(11n^2 + 10n - 1)}{(11n + 20 - 11/n)}.$$

If $p = n = 2^8 = 256$, then the speedup is approximately 255. The total speedup in this example is only a function of n and is independent of v.

The memory storage requirement for the coupled implementation is n complex numbers for a column of M, vn complex **numbers** for v s-vector Hermitians s_q^H, and v complex numbers for the v single components of s-vector s_q used in the second phase of the computation. If the v single s-vector components can be obtained by negating the imaginary part of the corresponding s_q^H stored in each PE local memory initially, the memory storage requirement can be reduced to $n + vn$ complex numbers. If an SIMD or mixed-mode machine is used, each element of the v s-vector **Hermi**tians can be stored on the CU and can be embedded as an immediate operand of an SIMD instruction broadcast from the CU to the **PEs.** By doing this, the memory storage local to each PE can be further reduced to just n complex numbers. Because in an MIMD machine each PE executes instructions from its local memory asynchronously, this saving in PE memory storage is not possible.

5.3. When $p = n^2$

This subsection considers the case when $p = n^2$. This analysis assumes that n and p must be a power of two. Let PE(i,j) be the PE in row i and column j in an $n \times n$ logical PE grid, where $0 \le i, j < n$. As shown in **Figure** 9, the data initially stored in PE(i,j) is $M(i,j)$, $s_q^H(i)$ for $0 \le q < v$, and $s_m(j)$ for $m \bmod n = i, 0 \le m < v$, and $0 \le i, j < n$.

Similar to Subsection 5.2, the calculations of the quadratic foxms can be decomposed into two phases: i.e., to compute (1) $r_q(j) = \sum_{i=0}^{n-1} s_q^H(i) * M(i,j)$, and (2) $w_q = \sum_{j=0}^{n-1} \sigma_q(j)$, where $\sigma_q(j) = r_q(j)^* \, s_q(j)$. In phase (1), PE($i,j$) first **performs** the v complex multiplications to compute $s_q^H(i)^* \, M(i,j)$, for all q. All **PEs** can perform these multiplications simultaneously. Then, because the result of $s_q^H(i) * M(i,j)$ is stored on PE(i,j), the v-recursive doubling procedure is used to foxm $r_q(j)$ for $0 \le q < v$. In this case, all the **PEs** in column j (i.e., PE(i,j) for $0 \le i < n$ and j fixed) are participating in the same v-recursive doubling procedure to foxm $r_q(j)$. Therefore, there are n independent v-recursive doubling procedures executing at the same time, as shown in Figure 10.

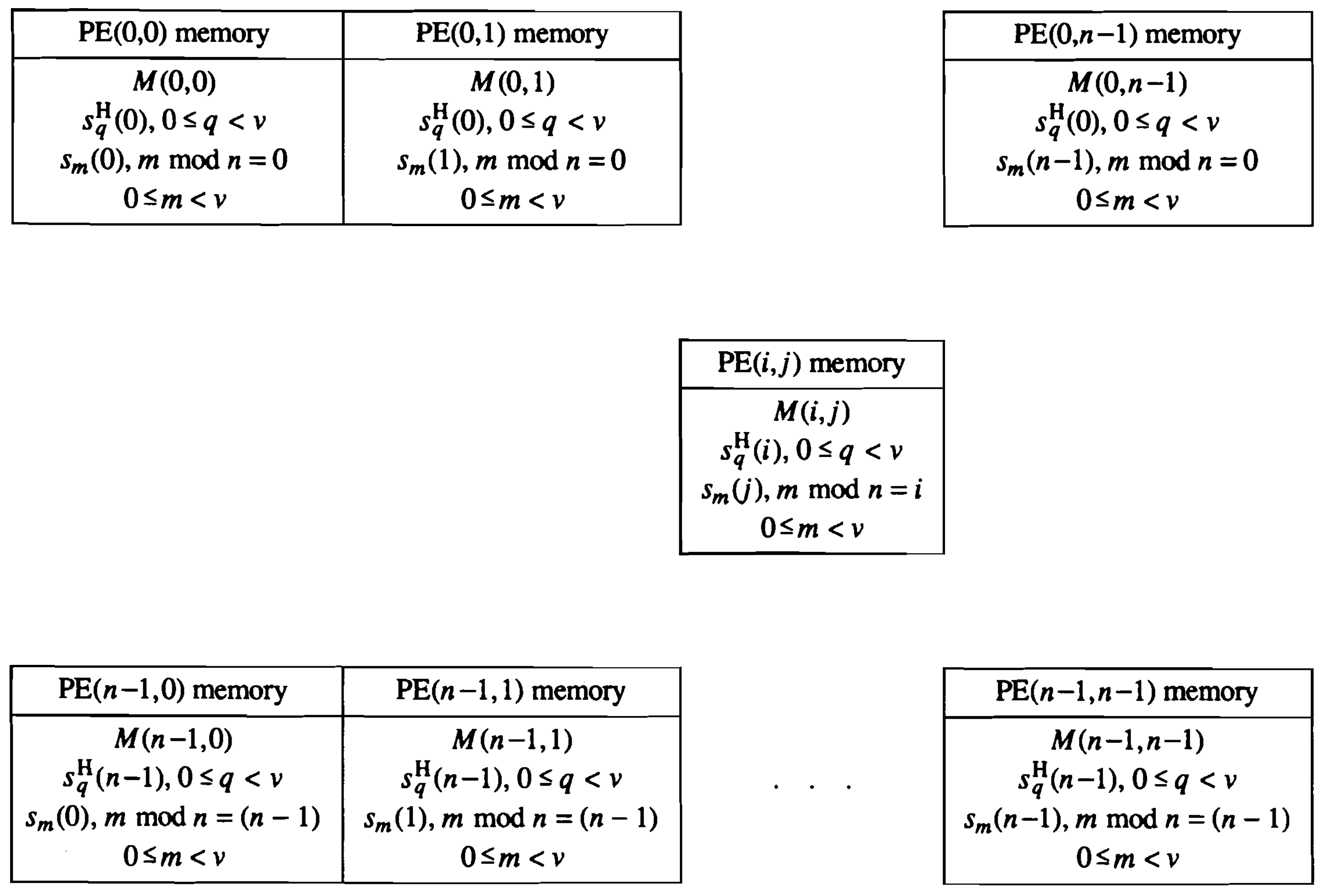


Figure 9: Distribution of M and v steering vectors onto n^2 PEs.

logical PE(i,j)	$s_q^{\mathrm{H}}(i) * M(i,j)$ within that PE	ν-recursive doubling in column j PEs	$r_q(j) = \sum_{i=0}^{3} s_q^{\mathrm{H}}(i) * M(i,j)$ within that PE
PE(0,0)	$s_0^{\mathrm{H}}(0) * M(0,0)$, $s_1^{\mathrm{H}}(0) * M(0,0)$, $\ldots$, $s_9^{\mathrm{H}}(0) * M(0,0)$		$r_0(0)$, $r_4(0)$, $r_8(0)$
PE(1,0)	$s_0^{\mathrm{H}}(1) * M(1,0)$, $s_1^{\mathrm{H}}(1) * M(1,0)$, $\ldots$, $s_9^{\mathrm{H}}(1) * M(1,0)$	$\Longrightarrow$	$r_1(0)$, $r_5(0)$, $r_9(0)$
PE(2,0)	$s_0^{\mathrm{H}}(2) * M(2,0)$, $s_1^{\mathrm{H}}(2) * M(2,0)$, $\ldots$, $s_9^{\mathrm{H}}(2) * M(2,0)$		$r_2(0)$, $r_6(0)$,
PE(3,0)	$s_0^{\mathrm{H}}(3) * M(3,0)$, $s_1^{\mathrm{H}}(3) * M(3,0)$, $\ldots$, $s_9^{\mathrm{H}}(3) * M(3,0)$		$r_3(0)$, $r_7(0)$,

Figure 10: Example of the $r_q(j)$ calculation on the column $j = 0$ PEs, for $n = 4, p = 16, \nu = 10, 0 \le i < 4$.

The number of transfer-add operations required to perform the v-recursive doubling is $\sum_{i=1}^{\log_2 n} \lceil v/2^i \rceil$. The final sums $r_q(j)$ are placed in PE(i,j), where q mod n = i for $0 \le q < v$. For v = 10 and n = 4, Figure 11 shows how $r_q(j)$, for all q and j, is calculated and distributed across the logical grid of PEs.

Assume logical PE(i,j) corresponds to physical PE $(i*n)+j$, for $0 \le i,j < n$. Because all n logical columns of PEs perform the n v-recursive doublings simultaneously, each transfer may involve all n^2 PEs sending data. Both the multistage cube and hypercube networks can support the required inter-PE communications for all n^2 PEs without any conflicts. That is because, each of the required parallel transfers can be done in a single pass through the multistage cube network without any conflicts [Sie90]. With the hypercube network, no conflicts will occur during each inter-PE transfer [Sie90] because each of these transfers involves pairs of PEs directly connected.

The second phase computes $w_q = \sum_{j=0}^{n-1} \sigma_q(j)$, where $\sigma_q(j) = r_q(j) * s_q(j)$. After the phase (1) computation, PE(i,j) holds $\lceil v/n \rceil$ distinct $r_q(j)$ elements and the corresponding components of $s_q(j)$ initially resident in its local memory, where q mod n = i for $0 \le q < v$. Thus, to form $\sigma_q(j)$, the number of complex multiplications required is $\lceil v/n \rceil$ and all PEs can perform these operations independently (see Figure 12). To total v vectors, each with n elements (i.e., $w_q = \sum_{j=0}^{n-1} \sigma_q(j)$ for $0 \le q < v$), all PEs in the same row will participate in a $\lceil v/n \rceil$-recursive doubling procedure, as shown in Figure 13. Because there are n rows of PEs, n independent $\lceil v/n \rceil$-recursive doubling procedures will be performed in parallel. The number of transfer-add operations required to total the components of w_q is $\sum_{i=1}^{\log_2 n} \lceil \lceil v/n \rceil / 2^i \rceil = \sum_{i=1}^{\log_2 n} \lceil v/(n2^i) \rceil$. As a result, each PE will get at most $\lceil v/p \rceil$ components of w_q. More precisely, PE(i, j) gets w_q, for $q = fp + jn + i$, $0 \le f < \lceil v/p \rceil$, and $0 \le q < v$. Figure 14 shows the distribution of w_q across the n^2 PEs for v = 10 and n = 4 at the end of computation. Analogous to the case discussed for the first phase, both the multistage cube and hypercube networks can support the necessary inter-PE transfers without any conflicts.

In summary, there are v complex multiplications followed by $\sum_{i=1}^{\log_2 n} \lceil v/2^i \rceil$ transfer-adds performed during the first phase of computation and the second phase requires $\lceil v/n \rceil$ complex multiplications followed by $\sum_{i=1}^{\log_2 n} \lceil v/(n2^i) \rceil$ transfer-adds. Thus, the computational complexity of the coupled implementation in this case is $v + \lceil v/n \rceil$ complex multiplications and $\sum_{i=1}^{\log_2 n} \lceil v/2^i \rceil + \sum_{i=1}^{\log_2 n} \lceil v/(n2^i) \rceil$ transfer-adds. By regarding each transfer-add as an inter-PE transfer followed by a

$r_q(j)$

$i\backslash j$	0	1	2	3
0	$r_0(0)$ $r_4(0)$ $r_8(0)$	$r_0(1)$ $r_4(1)$ $r_8(1)$	$r_0(2)$ $r_4(2)$ $r_8(2)$	$r_0(3)$ $r_4(3)$ $r_8(3)$
	$r_1(0)$ $r_5(0)$ $r_9(0)$	$r_1(1)$ $r_5(1)$ $r_9(1)$	$r_1(2)$ $r_5(2)$ $r_9(2)$	$r_1(3)$ $r_5(3)$ $r_9(3)$
2	$r_2(0)$ $r_6(0)$	$r_2(1)$ $r_8(1)$	$r_2(2)$ $r_6(2)$	$r_2(3)$ $r_6(3)$
3	$r_3(0)$ $r_7(0)$	$r_3(1)$ $r_7(1)$	$r_3(2)$ $r_7(2)$	$r_3(3)$ $r_7(3)$

cell(i,j) represents contents of PE(i,j) memory

=

$s_q^{\mathrm{H}}(i)$

$s_0^{\mathrm{H}}(0)$ $s_1^{\mathrm{H}}(0)$ • • • $s_8^{\mathrm{H}}(0)$ $s_9^{\mathrm{H}}(0)$	$s_0^{\mathrm{H}}(1)$ $s_1^{\mathrm{H}}(1)$ • • • $s_8^{\mathrm{H}}(1)$ $s_9^{\mathrm{H}}(1)$	$s_0^{\mathrm{H}}(2)$ $s_1^{\mathrm{H}}(2)$ • • • $s_8^{\mathrm{H}}(2)$ $s_9^{\mathrm{H}}(2)$	$s_0^{\mathrm{H}}(3)$ $s_1^{\mathrm{H}}(3)$ • • • $s_8^{\mathrm{H}}(3)$ $s_9^{\mathrm{H}}(3)$

PE(0,0) PE(1,0) PE(2,0) PE(3,0)

PE(0,1) PE(1,1) PE(2,1) PE(3,1)

PE(0,2) PE(1,2) PE(2,2) PE(3,2)

PE(0,3) PE(1,3) PE(2,3) PE(3,3)

$s_q^{\mathrm{H}}(i)$ is in PE(i,j)

$0 \le j < 4, 0 \le q < 10$

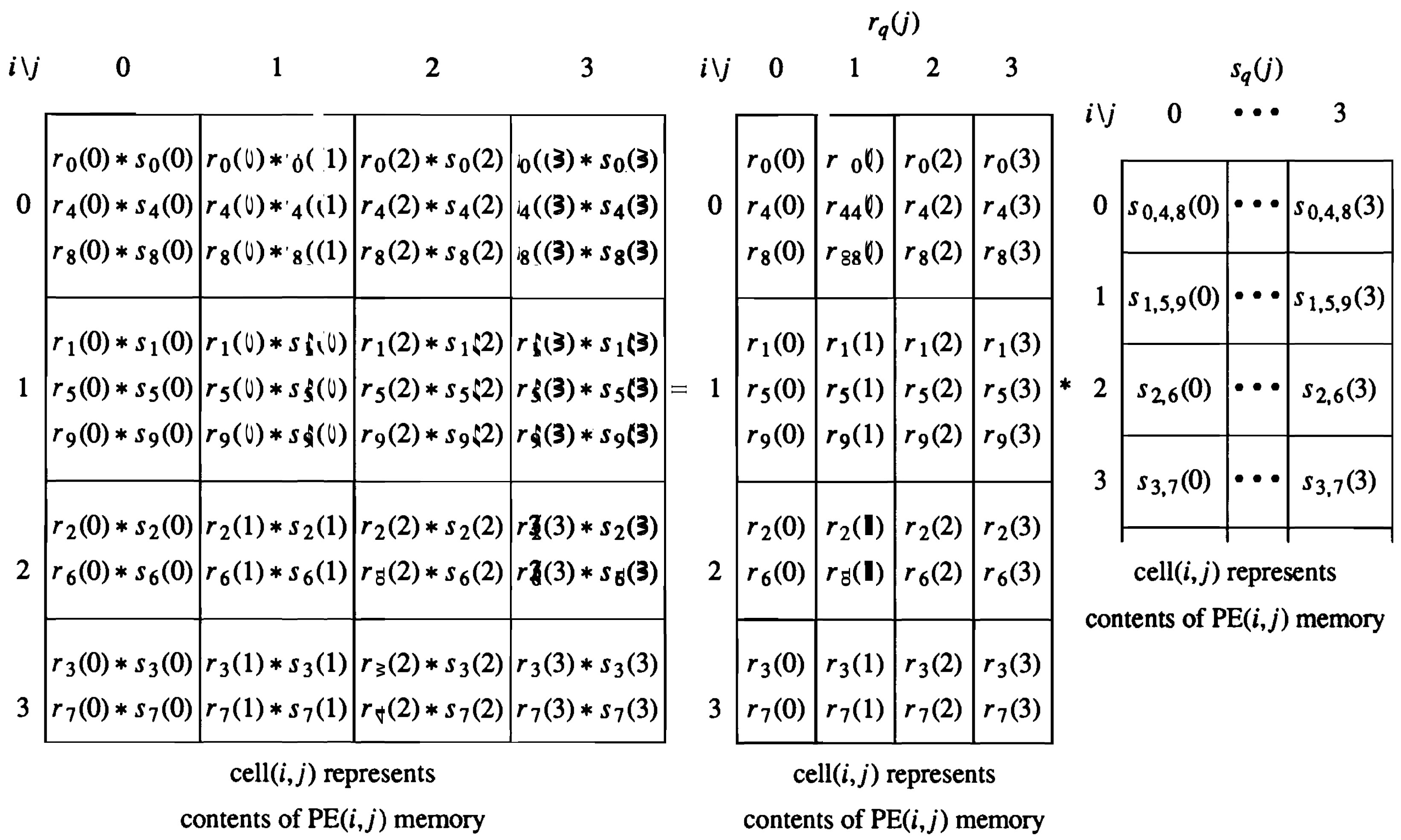
×

$M(i,j)$

$i\backslash j$	0	1	2	3
0	$M(0,0)$	$M(0,1)$	$M(0,2)$	$M(0,3)$
1	$M(1,0)$	$M(1,1)$	$M(1,2)$	$M(1,3)$
2	$M(2,0)$	$M(2,1)$	$M(2,2)$	$M(2,3)$
3	$M(3,0)$	$M(3,1)$	$M(3,2)$	$M(3,3)$

cell(i,j) representes contents of PE(i,j) memory

Figure 11: Distribution of the matrices onto the PEs for $v = 10$ and $n = 4$, where $r_q(j) = \sum_{i=0}^{3} s_q^{\mathrm{H}}(i) * M(i,j)$.



$i\backslash j$	0	1	2	3
0	$r_0(0)*s_0(0)$ $r_4(0)*s_4(0)$ $r_8(0)*s_8(0)$	$r_0(0)*s_0(1)$ $r_4(0)*s_4(1)$ $r_8(0)*s_8(1)$	$r_0(2)*s_0(2)$ $r_4(2)*s_4(2)$ $r_8(2)*s_8(2)$	$r_0(3)*s_0(3)$ $r_4(3)*s_4(3)$ $r_8(3)*s_8(3)$
1	$r_1(0)*s_1(0)$ $r_5(0)*s_5(0)$ $r_9(0)*s_9(0)$	$r_1(0)*s_1(0)$ $r_5(0)*s_5(0)$ $r_9(0)*s_9(0)$	$r_1(2)*s_1(2)$ $r_5(2)*s_5(2)$ $r_9(2)*s_9(2)$	$r_1(3)*s_1(3)$ $r_5(3)*s_5(3)$ $r_9(3)*s_9(3)$
2	$r_2(0)*s_2(0)$ $r_6(0)*s_6(0)$	$r_2(1)*s_2(1)$ $r_6(1)*s_6(1)$	$r_2(2)*s_2(2)$ $r_6(2)*s_6(2)$	$r_2(3)*s_2(3)$ $r_6(3)*s_6(3)$
3	$r_3(0)*s_3(0)$ $r_7(0)*s_7(0)$	$r_3(1)*s_3(1)$ $r_7(1)*s_7(1)$	$r_3(2)*s_3(2)$ $r_7(2)*s_7(2)$	$r_3(3)*s_3(3)$ $r_7(3)*s_7(3)$

cell(i,j) represents contents of PE(i,j) memory

=

$r_q(j)$

$i\backslash j$	0	1	2	3
0	$r_0(0)$ $r_4(0)$ $r_8(0)$	$r_0(1)$ $r_4(1)$ $r_8(1)$	$r_0(2)$ $r_4(2)$ $r_8(2)$	$r_0(3)$ $r_4(3)$ $r_8(3)$
1	$r_1(0)$ $r_5(0)$ $r_9(0)$	$r_1(1)$ $r_5(1)$ $r_9(1)$	$r_1(2)$ $r_5(2)$ $r_9(2)$	$r_1(3)$ $r_5(3)$ $r_9(3)$
2	$r_2(0)$ $r_6(0)$	$r_2(1)$ $r_6(1)$	$r_2(2)$ $r_6(2)$	$r_2(3)$ $r_6(3)$
3	$r_3(0)$ $r_7(0)$	$r_3(1)$ $r_7(1)$	$r_3(2)$ $r_7(2)$	$r_3(3)$ $r_7(3)$

cell(i,j) represents contents of PE(i,j) memory

*

$s_q(j)$

$i\backslash j$	0	• • •	3
0	$s_{0,4,8}(0)$	• • •	$s_{0,4,8}(3)$
1	$s_{1,5,9}(0)$	• • •	$s_{1,5,9}(3)$
2	$s_{2,6}(0)$	• • •	$s_{2,6}(3)$
3	$s_{3,7}(0)$	• • •	$s_{3,7}(3)$

cell(i,j) represents contents of PE(i,j) memory

Figure 12: Complex multiplications in second phase for $v = 10$ and $n = 4$ (* represents element-wise scalar multiplication).

logical PE(i,j)	$r_q(j) * s_q(j)$ within that PE	(v/n)-recursive doubling in row i PEs	$w_q = \sum_{j=0}^{3} r_q(j) * s_q(j)$
PE(0,0)	$r_0(0) * s_0(0)$, $r_4(0) * s_4(0)$, $r_8(0) * s_8(0)$		w_0
PE(0,1)	$r_0(1) * s_0(1)$, $r_4(1) * s_4(1)$, $r_8(1) * s_8(1)$	$\Rightarrow$	w_4
PE(0,2)	$r_0(2) * s_0(2)$, $r_4(2) * s_4(2)$, $r_8(2) * s_8(2)$		w_8
PE(0,3)	$r_0(3) * s_0(3)$, $r_4(3) * s_4(3)$, $r_8(3) * s_8(3)$		

igure 13: Example of the w_q calculation on the row $i = 0$ PEs, for $n = 4$, $p = 16$, $v = 10$, and $0 \le j < 4$.

$i \backslash j$	0	1	2	3
0	w_0	w_4	w_8	–
1	w_1	w_5	w_9	–
2	w_2	w_6	–	–
3	w_3	w_7	–	–

cell(i,j) represents
contents of PE(i,j) memory

Figure 14: Final distribution of scalar w_q for $v=10$ and $n=4$.

complex addition, the computational complexity becomes $v+\lceil v/n\rceil$ complex multiplications, $\sum_{i=1}^{\log_2 n}\lceil v/2^i\rceil+\sum_{i=1}^{\log_2 n}\lceil v/(n2^i)\rceil$ complex additions, and $\sum_{i=1}^{\log_2 n}\lceil v/2^i\rceil+\sum_{i=1}^{\log_2 n}\lceil v/(n2^i)\rceil$ inter-PE transfers.

Because $\sum_{i=1}^{\log_2 n} v/2^i+\sum_{i=1}^{\log_2 n} v/(n2^i)=v(1-1/n)+v(1/n-1/n^2)=v-(v/n^2)$,

$$v-(v/n^2)\le\sum_{i=1}^{\log_2 n}\lceil v/2^i\rceil+\sum_{i=1}^{\log_2 n}\lceil v/(n2^i)\rceil\le v-(v/n^2)+2\log_2 n.$$

Consequently, both the number of complex additions and inter-PE transfers are bounded by $v-(v/n^2)+2\log_2 n$. If the time to do a complex multiplication and the time to do an inter-PE transfer are ten times that required by a complex addition, as used in the previous subsection, then the total speedup is serial-time/parallel-time, or,

$$\text{speedup}=\frac{(vn^2-v)+10(vn^2+vn)}{\left(\sum_{i=1}^{\log_2 n}\lceil v/2^i\rceil+\sum_{i=1}^{\log_2 n}\lceil v/(n2^i)\rceil\right)+10(v+\lceil v/n\rceil)+10\left(\sum_{i=1}^{\log_2 n}\lceil v/2^i\rceil+\sum_{i=1}^{\log_2 n}\lceil v/(n2^i)\rceil\right)}$$

$$\ge\frac{v(11n^2+10n-1)}{(v-(v/n^2)+2\log_2 n)+10(v+(v/n)+1)+10(v-(v/n^2)+2\log_2 n)}$$

$$=\frac{v(11n^2+10n-1)}{v(21+10/n-11/n^2)+22\log_2 n+10}.$$

If $n=128$, $v=4n=512$, and $p=n^2=16{,}384$, the speedup is 8,482.

If $v \geq n$, then $v - (v/n^2) w\, 2\log_2 n$ and, consequently, $\sum_{i=1}^{\log_2 n} \lceil v/2^i \rceil + \sum_{i=1}^{\log_2 n} \lceil v/(n2^i) \rceil \simeq v - (v/n^2)$. Comparing the number of complex multiplications and additions to the corresponding operations required in the serial algorithm shows that an approximate speedup of $n^2 = p$ is achieved. This multiplicative factor of $n^2 = p$ speedup on calculations incurs an overhead cost that is an additive term of approximately v inter-PE transfers.

The memory storage **requirements** per PE for this method is $1 + v + \lceil v/n \rceil$; **i.e.,** one location for $M(i,j)$, v locations for the s-vector components of $s_q^H(i)$ for $0 \leq q < v$, and $\lceil v/n \rceil$ locations for the s-vector components $s_m(j)$ for $m = fn$, $0 \leq f < \lceil v/n \rceil$, and $0 \leq m < v$. Unlike the $p = n$ case discussed in Subsection 5.2, because there is no common data stored on all **PEs,** the possibility of broadcasting common data to the **PEs** as immediate operands in SIMD mode does not exist. Consequently, the memory storage **requirements** per PE are the same for both SIMD and MIMD machines.

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

The previous two subsections introduced the coupled method via two special case examples, $p = n$ (subsection 5.2) and $p = n^2$ (subsection 5.3). This subsection **formalizes** the presentation of the coupled method: **i.e.,** the mapping of the MQF problem onto an $a \times b$ $(= p)$ logical grid of p **PEs,** where $1 \leq a,b \leq n$, and a, b, and p are powers of two.

Let PE(i, j) denote the PE in row i and column j in an $a \times b$ logical PE grid, where $0 \leq i < a$ and $0 \leq j < b$. Figure 15 illustrates how M, s_q^H, and s_q are distributed across the **PEs** of the $a \times b$ grid. Initially, 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 $\lceil v/a \rceil$ 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 Figure 15. 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 $0 \leq q < 6$, $s_1(2)$, $s_3(2)$, and $s_5(2)$, as shown in the boxes in Figure 16.

As in the previous cases, 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 $w_q = \sum_{y=0}^{n-1} \sigma_q(y)$, where $\sigma_q(y) = r_q(y) * s_q(y)$. Assuming $v = 6$, $n = 4$, $a = 2$, and $b = 4$, Figure 17 illustrates the complex multiplications and **subsequent** complex additions performed by **PE(1,2).** For each s-vector s_q^H, each PE calculates (n/a) products for each of (n/b) $r_q(y)$'s. **PE(i, j)** calculates $\sum_{x=i(n/a)}^{(i+1)(n/a)-1} s_q^H(x) * M(x,y)$, for $j(n/b) \leq y < (j+1)(n/b)$ and $0 \leq q < v$. 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.

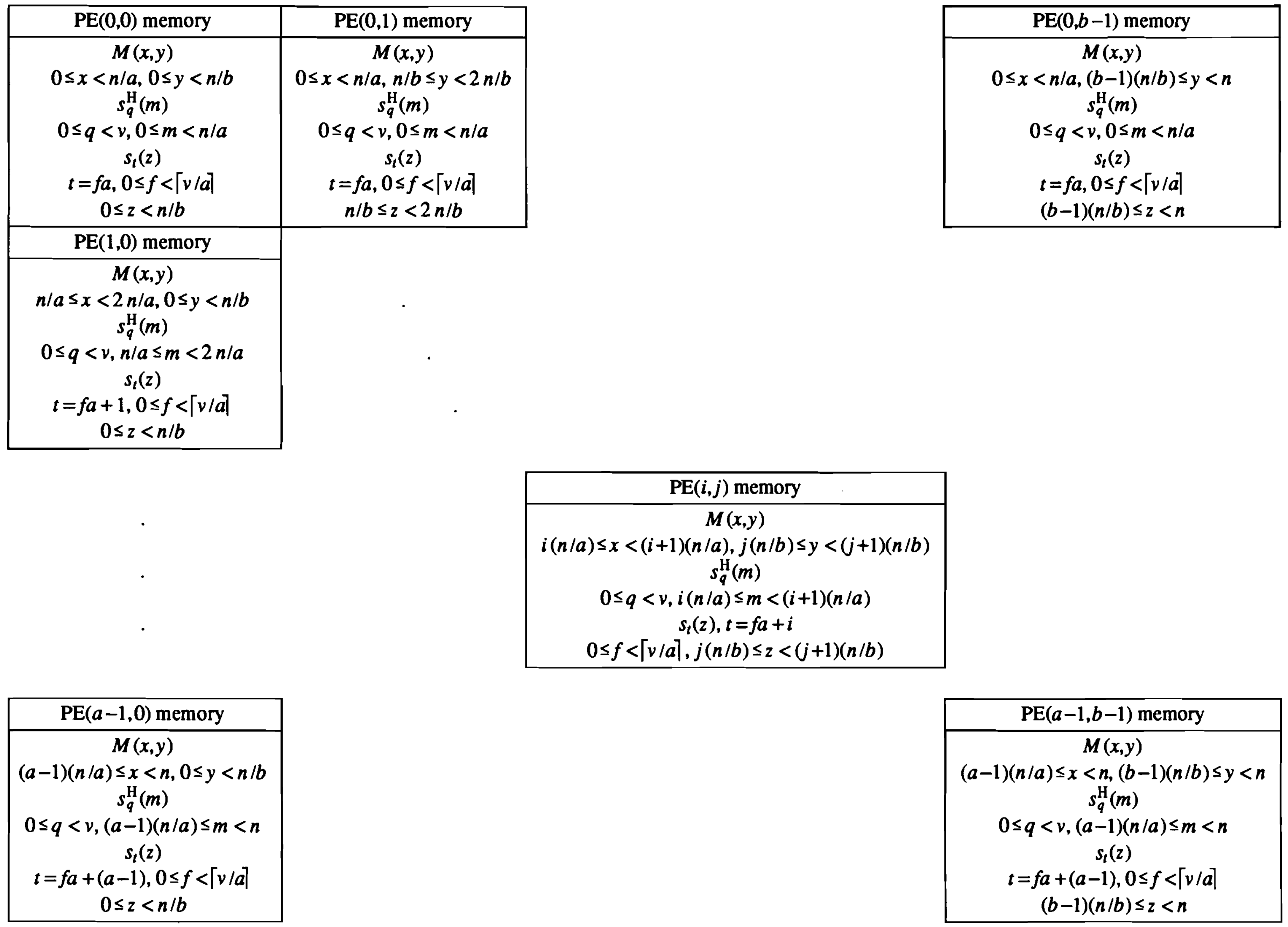

PE(0,0) memory
$M(x,y)$
$0 \le x < n/a,\ 0 \le y < n/b$
$s_q^{\mathrm{H}}(m)$
$0 \le q < v,\ 0 \le m < n/a$
$s_t(z)$
$t = fa,\ 0 \le f < \lceil v/a \rceil$
$0 \le z < n/b$
PE(0,1) memory
$M(x,y)$
$0 \le x < n/a,\ n/b \le y < 2n/b$
$s_q^{\mathrm{H}}(m)$
$0 \le q < v,\ 0 \le m < n/a$
$s_t(z)$
$t = fa,\ 0 \le f < \lceil v/a \rceil$
$n/b \le z < 2n/b$
PE(0,$b-1$) memory
$M(x,y)$
$0 \le x < n/a,\ (b-1)(n/b) \le y < n$
$s_q^{\mathrm{H}}(m)$
$0 \le q < v,\ 0 \le m < n/a$
$s_t(z)$
$t = fa,\ 0 \le f < \lceil v/a \rceil$
$(b-1)(n/b) \le z < n$
PE(1,0) memory
$M(x,y)$
$n/a \le x < 2n/a,\ 0 \le y < n/b$
$s_q^{\mathrm{H}}(m)$
$0 \le q < v,\ n/a \le m < 2n/a$
$s_t(z)$
$t = fa + 1,\ 0 \le f < \lceil v/a \rceil$
$0 \le z < n/b$
PE(i,j) memory
$M(x,y)$
$i(n/a) \le x < (i+1)(n/a),\ j(n/b) \le y < (j+1)(n/b)$
$s_q^{\mathrm{H}}(m)$
$0 \le q < v,\ i(n/a) \le m < (i+1)(n/a)$
$s_t(z),\ t = fa + i$
$0 \le f < \lceil v/a \rceil,\ j(n/b) \le z < (j+1)(n/b)$
PE($a-1$,0) memory
$M(x,y)$
$(a-1)(n/a) \le x < n,\ 0 \le y < n/b$
$s_q^{\mathrm{H}}(m)$
$0 \le q < v,\ (a-1)(n/a) \le m < n$
$s_t(z)$
$t = fa + (a-1),\ 0 \le f < \lceil v/a \rceil$
$0 \le z < n/b$
PE($a-1,b-1$) memory
$M(x,y)$
$(a-1)(n/a) \le x < n,\ (b-1)(n/b) \le y < n$
$s_q^{\mathrm{H}}(m)$
$0 \le q < v,\ (a-1)(n/a) \le m < n$
$s_t(z)$
$t = fa + (a-1),\ 0 \le f < \lceil v/a \rceil$
$(b-1)(n/b) \le z < n$

Figure 15: Distribution of M and v steering vectors onto $a \times b$ PEs. (The maximum value t can have is $v - 1$.)

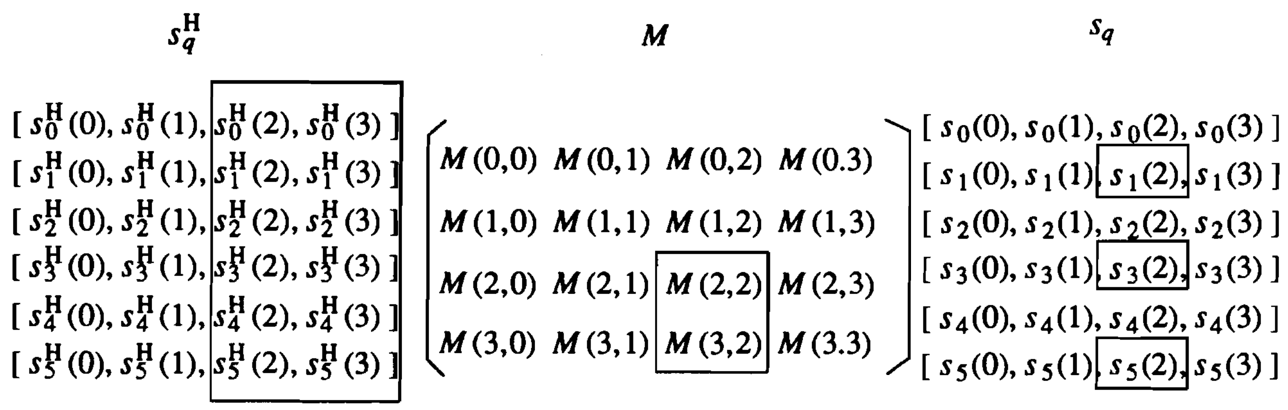

s_q^H
M
s_q
$[s_0^H(0), s_0^H(1), s_0^H(2), s_0^H(3)]$
$[s_1^H(0), s_1^H(1), s_1^H(2), s_1^H(3)]$
$[s_2^H(0), s_2^H(1), s_2^H(2), s_2^H(3)]$
$[s_3^H(0), s_3^H(1), s_3^H(2), s_3^H(3)]$
$[s_4^H(0), s_4^H(1), s_4^H(2), s_4^H(3)]$
$[s_5^H(0), s_5^H(1), s_5^H(2), s_5^H(3)]$
$M(0,0)$ $M(0,1)$ $M(0,2)$ $M(0.3)$
$M(1,0)$ $M(1,1)$ $M(1,2)$ $M(1,3)$
$M(2,0)$ $M(2,1)$ $M(2,2)$ $M(2,3)$
$M(3,0)$ $M(3,1)$ $M(3,2)$ $M(3.3)$
$[s_0(0), s_0(1), s_0(2), s_0(3)]$
$[s_1(0), s_1(1), s_1(2), s_1(3)]$
$[s_2(0), s_2(1), s_2(2), s_2(3)]$
$[s_3(0), s_3(1), s_3(2), s_3(3)]$
$[s_4(0), s_4(1), s_4(2), s_4(3)]$
$[s_5(0), s_5(1), s_5(2), s_5(3)]$

Figure 16: Data elements in boxes are stored in **PE(1,2)** local memory.

Next, the summation in phase (1) is performed. The elements to be totaled to calculate $r_q(y)$ are resident in the a **PEs** of a given column. In the coupled algorithm, because each of the a **PEs** in a given column contributes ton lb different $r_q(y)$'s for each of the v s-vectors, a (vn/b)-**recursive doubling procedure**, utilizing $\sum_{i=l}^{\log_2 a} \lceil (vn/b)/2^i \rceil$ transfer-add operations, on the a **PEs** of each column can be used. 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. As a result, **PE**(i,j) will hold $r_q(y)$, where $j(n/b) \le y < (j+1)(n/b)$ and for all q, $0 \le q$ cv, where q moda =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 Figure 18.

Phase (2) of the coupled algorithm involves the formation of $w_q = \sum_{y=0}^{n-1} \sigma_q(y)$, where $\sigma_q(y) = r_q(y) * s_q(y)$. 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 $\sigma_q(y)$. PE(i,j) forms the local partial sums of the products $\sum_{y=j(n/b)}^{(j+1)(n/b)-1} \sigma_q(y)$, for all q, $0 \le q <$ v, where q mad a =i. This is shown in Figure 19 for v =6, n =4, a =2, and b =4. To do this, $\lceil v/a \rceil (n$lb) complex multiplications and $\lceil v/a \rceil$(nlb − 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 $\lceil v/a \rceil$ w_q values, for all q, $0 \le$ q < v, where q **mod** a = i. To compute all v values of w_q in parallel, a independent $\lceil v/a \rceil$-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

PE(1,2) memory contents when forming

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

$$\boxed{\begin{array}{l}
s_0^{\mathrm{H}}(2) * M(2,2) + s_0^{\mathrm{H}}(3) * M(3,2) \\
s_1^{\mathrm{H}}(2) * M(2,2) + s_1^{\mathrm{H}}(3) * M(3,2) \\
s_2^{\mathrm{H}}(2) * M(2,2) + s_2^{\mathrm{H}}(3) * M(3,2) \\
s_3^{\mathrm{H}}(2) * M(2,2) + s_3^{\mathrm{H}}(3) * M(3,2) \\
s_4^{\mathrm{H}}(2) * M(2,2) + s_4^{\mathrm{H}}(3) * M(3,2) \\
s_5^{\mathrm{H}}(2) * M(2,2) + s_5^{\mathrm{H}}(3) * M(3,2)
\end{array}}$$

'igure 17: Contents of PE(i,j), $i = 1$, $j = 2$, memory after forming the partial sums $\sum_{x=i(n/a)}^{(i+1)(n/a)-1} s_q^{\mathrm{H}}(x) * M(x,y)$, $0 \le q < v$, $j(n/b) \le y < (j+1)(n/b)$, for $v = 6$, $n = 4$, $a = 2$, and $b = 4$.

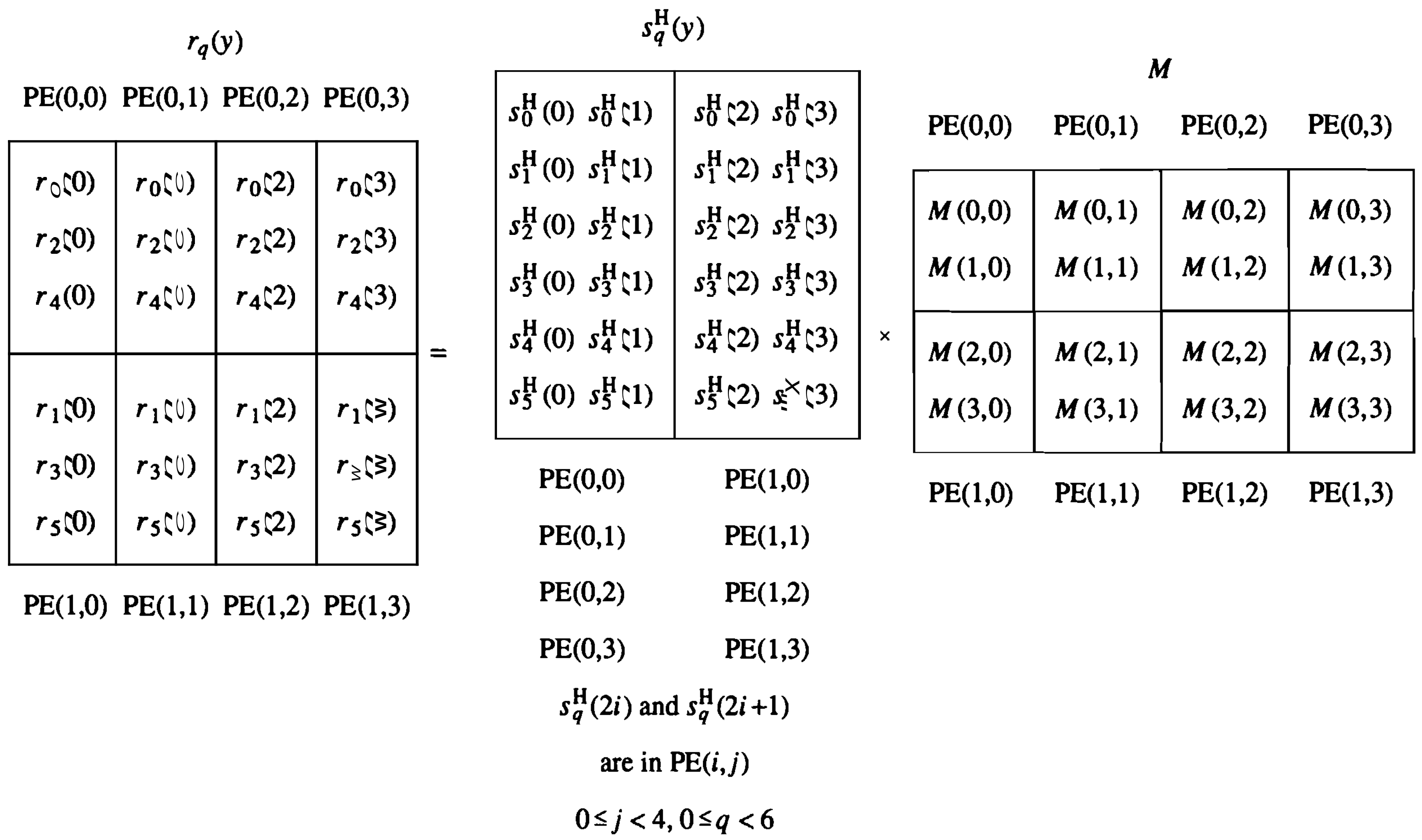
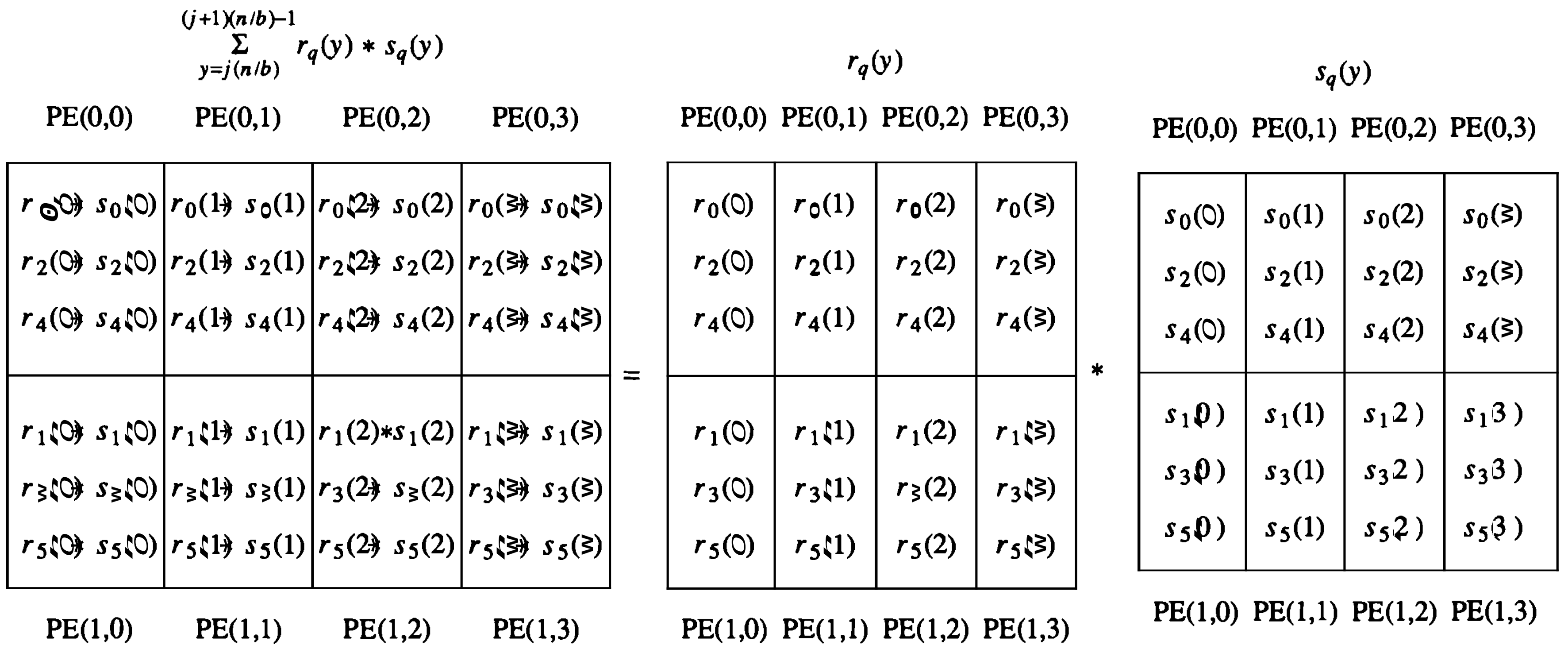

$r_q(y)$
PE(0,0) PE(0,1) PE(0,2) PE(0,3)
PE(1,0) PE(1,1) PE(1,2) PE(1,3)
$s_q^{\mathrm{H}}(y)$
$s_q^{\mathrm{H}}(2i)$ and $s_q^{\mathrm{H}}(2i+1)$
are in PE(i,j)
$0 \le j < 4, 0 \le q < 6$
M

Figure 18: Distribution of data over the PEs for $v = 6$, $n = 4$, $a = 2$, and $b = 4$, where $r_q(y) = \sum_{x=0}^{3} s_q^{\mathrm{H}}(x) * M(x,y)$.



$\sum_{y=j(n/b)}^{(j+1)(n/b)-1} r_q(y) * s_q(y)$
PE(0,0) PE(0,1) PE(0,2) PE(0,3)
$r_0(0)*s_0(0)$ $r_0(1)*s_0(1)$ $r_0(2)*s_0(2)$ $r_0(3)*s_0(3)$
$r_2(0)*s_2(0)$ $r_2(1)*s_2(1)$ $r_2(2)*s_2(2)$ $r_2(3)*s_2(3)$
$r_4(0)*s_4(0)$ $r_4(1)*s_4(1)$ $r_4(2)*s_4(2)$ $r_4(3)*s_4(3)$
$r_1(0)*s_1(0)$ $r_1(1)*s_1(1)$ $r_1(2)*s_1(2)$ $r_1(3)*s_1(3)$
$r_3(0)*s_3(0)$ $r_3(1)*s_3(1)$ $r_3(2)*s_3(2)$ $r_3(3)*s_3(3)$
$r_5(0)*s_5(0)$ $r_5(1)*s_5(1)$ $r_5(2)*s_5(2)$ $r_5(3)*s_5(3)$
PE(1,0) PE(1,1) PE(1,2) PE(1,3)
=
$r_q(y)$
PE(0,0) PE(0,1) PE(0,2) PE(0,3)
$r_0(0)$ $r_0(1)$ $r_0(2)$ $r_0(3)$
$r_2(0)$ $r_2(1)$ $r_2(2)$ $r_2(3)$
$r_4(0)$ $r_4(1)$ $r_4(2)$ $r_4(3)$
$r_1(0)$ $r_1(1)$ $r_1(2)$ $r_1(3)$
$r_3(0)$ $r_3(1)$ $r_3(2)$ $r_3(3)$
$r_5(0)$ $r_5(1)$ $r_5(2)$ $r_5(3)$
PE(1,0) PE(1,1) PE(1,2) PE(1,3)
*
$s_q(y)$
PE(0,0) PE(0,1) PE(0,2) PE(0,3)
$s_0(0)$ $s_0(1)$ $s_0(2)$ $s_0(3)$
$s_2(0)$ $s_2(1)$ $s_2(2)$ $s_2(3)$
$s_4(0)$ $s_4(1)$ $s_4(2)$ $s_4(3)$
$s_1(0)$ $s_1(1)$ $s_1(2)$ $s_1(3)$
$s_3(0)$ $s_3(1)$ $s_3(2)$ $s_3(3)$
$s_5(0)$ $s_5(1)$ $s_5(2)$ $s_5(3)$
PE(1,0) PE(1,1) PE(1,2) PE(1,3)

Figure 19: Distribution of data over the PEs for $v = 6$, $n = 4$, $a = 2$, and $b = 4$ (* represents element-wise scalar multiplication).

conflicts.

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 Figure *20* 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.

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)

Figure 20: Distribution of w_q**'s** at conclusion of procedure for $v = 6$, $n = 4$, $a = 2$, and $b = 4$.

In summary, the **first** phase requires $vn^2 1p$ 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 $\lceil v/a \rceil (nlb)$ **com**plex 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 *2* summarizes the computational complexity of the uncoupled algorithm from Section 4 and the coupled algorithm of this subsection.

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

Table 2: Computational complexity for the uncoupled and coupled parallel algorithms. Recall that p is the total number of **PEs** in the logical PE grid ($p = a * b$).

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

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

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

$$\geq (vn/p)(n-a) + (v/a)(n/b - 1) + \sum_{i=1}^{\log_2 a} vn/(2^i b) + \sum_{i=1}^{\log_2 b} v/(2^i a) = (v/p)(n^2 - 1) \quad (3)$$

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

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

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 two. As shown in Equations (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, 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 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.

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 s-vector Hermitians, s_q^H, and the $\lceil v/a \rceil (n/b)$ **term** is for the (n/b)-element part of $\lceil v/a \rceil$ 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 **[BeS91]**. 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.

55. 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 $n < p \le n^2$ case, if p PEs are partitioned into $a = p/n$ independent groups of $1 \times n$ PEs, $\lceil v/a \rceil$ s-vectors are assigned to each of v mod a groups, and $\lfloor v/a \rfloor$ s-vectors are assigned to each of the remaining groups. Then the recursive doubling becomes unnecessary during the first phase of computation. This may reduce overall execution time. Assuming $n < p \le n^2$ and $a = p/n$, Table 3 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.

	L: one group of $a \times n$ PEs, where $a = p/n$	R: "a" groups of $(1 \times n)$ PEs	$\Delta = L - R$	
			$v = ca$	$v \ne ca$
M	$vn/a + \lceil v/a \rceil$	$\lceil v/a \rceil (n+1)$	L=R	$L \le R$ $(0 \le \Delta \le n)$
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	$L \le R$ $(0 \le \Delta \le n-1)$
T	$\sum_{i=1}^{\log_2 a} \lceil v/2^i \rceil + \sum_{i=1}^{\log_2 n} \lceil v/(2^i a) \rceil$	$\sum_{i=1}^{\log_2 n} \lceil v/(2^i a) \rceil$	$L > R$ $(\Delta > v - (v/a))$	$L > R$ $(\Delta > v - (v/a))$

Table 3: Computational complexity for the one group $a \times n$ PEs and "a" groups of $1 \times n$ PEs coupled methods, where $n < p \le n^2$ and M, A, and T represent multiplications, additions, and inter-PE transfers, respectively.

As shown in Table 3, 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; otherwise the single group coupled method will be at least as good.

5.6. Coupled Data-Parallel Method When n is Not a Power of 2

This subsection extends the discussion of the coupled method to include the case when n is not a power of 2. The data allocation among the PEs is slightly modified, but the algorithm for the coupled method described earlier is still valid. Consequently, it will not be discussed in great detail.

The dismbution of the M matrix and the Hermitians of v s-vectors across $a \times b$ PEs are described below. Initially, a $\lceil v/a \rceil$-element part of the Hermitian of each s-vector, s_q^H, a $(\lceil n/b \rceil)$-element part of $\lceil v/a \rceil$ s-vectors, s_q, and a $(\lceil n/a \rceil) \times (\lceil n/b \rceil)$ portion of M are stored in each PE's local memory. Stated precisely, the memory of PE(i,j) is loaded with $s_q^H(m)$ for $i(\lceil n/a \rceil) \leq m < (i+1)(\lceil n/a \rceil)$, $0 \leq m < n$, and $0 \leq q < v$ and $s_t(z)$, where $t = fa + i$ for $j(\lceil n/b \rceil) \leq z < (j+1)(\lceil n/b \rceil)$, $0 \leq z < n$, and $0 \leq f < \lceil v/a \rceil$. PE$(i,j)$ holds $M(x,y)$ for $i(\lceil n/a \rceil) \leq x < (i+1)(\lceil n/a \rceil)$, $j(\lceil n/b \rceil) \leq y < (j+1)(\lceil n/b \rceil)$, and $0 \leq x,y < n$.

Similar to the previous case, the calculation of MQF is decomposed into two phases, i.e., to compute (1) $r_q(y) = \sum_{x=0}^{n-1} s_q^H(x) * M(x,y)$, and (2) $w_q = \sum_{y=0}^{n-1} \sigma_q(y)$, where $\sigma_q(y) = r_q(y) * s_q(y)$. Because of the data dismbution described above, phase (1) requires at most: $v\lceil n/b \rceil \lceil n/a \rceil$ complex multiplications followed by $v\lceil n/b \rceil(\lceil n/a \rceil - 1)$ complex additions and $\sum_{i=1}^{\log_2 a} \lceil (v\lceil n/b \rceil)/2^i \rceil$ transfer-adds. After the first phase, each PE holds at most $\lceil v/a \rceil \lceil n/b \rceil$ distinct $r_q(y)$ components and the corresponding elements of the s-vectors $s_q(y)$ required to form $\sigma_q(y)$. Thus, in phase (2), $\lceil v/a \rceil \lceil n/b \rceil$ complex multiplications followed by $\lceil v/a \rceil(\lceil n/b \rceil - 1)$ complex additions and $\sum_{i=1}^{\log_2 b} \lceil v/(2^i a) \rceil$ transfer-adds are performed. Table 4 compares the computational complexity of the coupled algorithm when n is and is not a power of 2.

The data memory storage requirements of the coupled method for this case is $\lceil n/a \rceil \lceil n/b \rceil + v\lceil n/a \rceil + \lceil v/a \rceil \lceil n/b \rceil$ complex numbers, i.e., $\lceil n/a \rceil \lceil n/b \rceil$ for the $\lceil n/a \rceil \times \lceil n/b \rceil$ subarray of M, $v\lceil n/a \rceil$ for the $\lceil n/a \rceil$-element part of each of the v s_q^H vectors, and $\lceil v/a \rceil \lceil n/b \rceil$ for the $\lceil n/b \rceil$-element part of $\lceil v/a \rceil$ s_q vectors stored in each PE memory. The analyses in Subsections 5.4 and 5.5 can be used here. Depending on the trade-off between the increase in communication

	n is not a power of 2	n is a power of 2
complex multiplications	$v\lceil \frac{n}{b}\rceil\lceil \frac{n}{a}\rceil + \lceil \frac{v}{a}\rceil\lceil \frac{n}{b}\rceil$	$\frac{vn^2}{P} + \lceil \frac{v}{a}\rceil \frac{n}{b}$
complex additions	$v\lceil \frac{n}{b}\rceil(\lceil \frac{n}{a}\rceil - 1) + \lceil \frac{v}{a}\rceil(\lceil \frac{n}{b}\rceil - 1)$	$(\frac{vn}{b})(n-a) + \lceil \frac{v}{a}\rceil(\frac{n}{b} - 1)$
transfer-adds	$\sum_{i=1}^{\log_2 a} \lceil \frac{v\lceil n/b\rceil}{2^i}\rceil + \sum_{i=1}^{\log_2 b} \lceil \frac{v}{2^i a}\rceil$	$\sum_{i=1}^{\log_2 a} \lceil \frac{vn}{2^i b}\rceil + \sum_{i=1}^{\log_2 b} \lceil \frac{v}{2^i a}\rceil$

Table 4: Comparison of computational complexity of the coupled algorithm when n is and is not a power of 2. Recall that p is the total number of **PEs** in the logical PE grid ($p = a * b$).

time and the decrease in computation time, the optimal implementation can be either the (single or multiple groups) coupled or uncoupled method.

6. Generalizing the MQF Algorithm

Given p **PEs** where $p = a * n \le n^2$, 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 5.4), and the multiple group coupled method (Subsection 5.5). Let $p = \gamma * a * p$, where a, p, and γ are powers of two and $1 \le \gamma \le p$ and $1 \le a, \beta \le n$. Given p **PEs** where $p \le n^2$, if all p **PEs** can be partitioned into γ groups of $\alpha \times \beta$ **PEs**, then the problem is to compute the MQF for $\lceil v/\gamma \rceil$ 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 $\gamma = 1$, $a = a$, $\beta = b$, and $\gamma = p$, $a = \beta = 1$, respectively.

Let v, a, and b in Table 2 be replaced by $\lceil v/\gamma \rceil$, a, 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, $\lceil v/\gamma \rceil (n/(\alpha\beta))(n - a) + \lceil v/(\gamma\alpha) \rceil (n/\beta - 1)$ complex additions, and $\sum_{i=1} \lceil vn/(\gamma\beta 2^i) \rceil + \sum_{i=1}^{\log_2 \beta} \lceil v/(\gamma\alpha 2^i) \rceil$ 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 $\lceil v/\gamma \rceil (n/(\alpha\beta))(n - a) + \lceil v/(\gamma\alpha) \rceil (n/\beta - 1) + \sum_{i=1}^{\log_2 \alpha} \lceil vn/(\gamma\beta 2^i) \rceil + \sum_{i=1}^{\log_2 \beta} \lceil v/(\gamma\alpha 2^i) \rceil$.

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, a, and β. By fixing $v, p,$

and n, the lower bounds for those operations are derived as follows:

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

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

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

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

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

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

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

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

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

The lower bounds for Equations (6) and (7) are independent of a and β. When the number of PE groups (y) is decided, to minimize Equation (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 two and $1 \leq p/\gamma \leq n^2$, the resources can be fully utilized. This is done by assigning $\lceil v/\gamma\rceil$ s-vectors to each of v mod γ groups, and $\lfloor v/\gamma\rfloor$ 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 $\lceil v/\gamma\rceil$ s-vectors on p/γ PEs. The optimal value of γ is dependent on v, n, and the communication overhead. Table 5 summarizes the computational complexity of the uncoupled, single group, and generalized methods.

Table 6 shows the comparison of the computational complexity between the single group coupled and uncoupled methods (the two extreme cases of y in the generalize method). If p divides v, $\lceil v/p\rceil = v/p$ and

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

	Coupled Method where $p = a*b = \gamma * a*\beta$		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	$\sum_{i=1}^{\log_2 a} \lceil \frac{vn}{2^i b} \rceil + \sum_{i=1}^{\log_2 b} \lceil \frac{v}{2^i a} \rceil$	$\sum_{i=1}^{\log_2 \alpha} \lceil \frac{vn}{\gamma\beta 2^i} \rceil + \sum_{i=1}^{\log_2 \beta} \lceil \frac{v}{\gamma\alpha 2^i} \rceil$	–

Table 5: Computational complexity for the coupled and uncoupled methods when $p \le n^2$ and M, A, and T represent the number of multiplications, additions, and inter-PE transfers, respectively.

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

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

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 $v = cp$.

If p does not divide v, $\lceil v/p \rceil = (v/p) + 1$ and

$$vn^2/p + \lceil v/a \rceil (n/b) < (v/p)(n^2+n) + (n/b) < ((v/p)+1)(n^2+n)$$

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

$$< (v/p)(n^2-1) + ((n/b)-1) + \log_2 a + \log_2 b < ((v/p)+1)(n^2-1).$$

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 the reduction in communication time is greater than the increase in computation time, the uncoupled method will outperform the single group coupled

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

Table 6: Computational complexity for the uncoupled and single group coupled methods, where $0 < p \le n^2$ and M, A, and T represent the number of multiplications, additions, and inter-PE transfers, respectively.

method; otherwise the single group coupled method will be at least as good.

It can be seen from Table **5** that, in general, if a and p are kept constant and γ is doubled (β 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, *a*, and β. The parameters v and *n* are input data parameters and γ, a , and β are logical system configuration parameters used by the generalized method. Section **4** and Subsection **5.4** describe the structure of the algorithm when certain restrictions are placed on γ, a , and β. The uncoupled method applies when $\alpha\beta = 1$. When $\gamma = 1$, $\alpha\beta = p$ and $\beta \le n$, the coupled method is used. Subsection **5.4** showed that, when using the coupled method and γ is fixed, it is best to make a as small as possible, but keep β less than or equal to n. Subsection **5.5** made a strong argument for partitioning the problem in certain situations. It demonstrated that if $\gamma = a$, $a = 1$, and $\beta = n$, when $v = ca$, it is better to use a groups of $1 \times n$ **PEs** then 1 group of a $\times n$ **PEs**. Finally, Section **6 concludes** that the logical configuration $2\gamma \times \alpha \times \beta/2$ (when $\beta/2$ is an integer) will outperform the logical configuration $\gamma \times \alpha \times \beta$ when 2γ divides v: another argument for partitioning the problem.

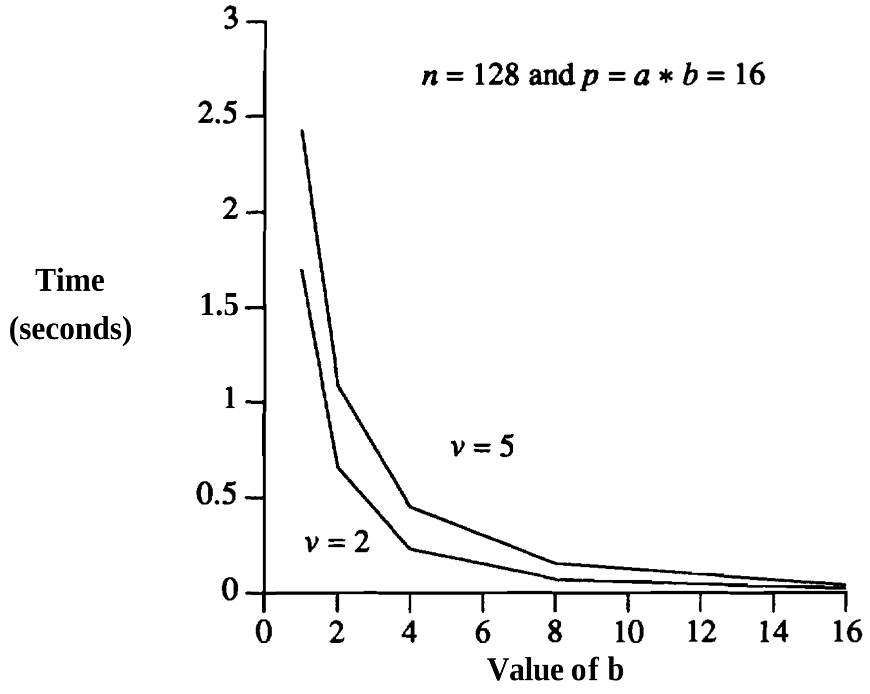
Now consider the many possible ways to solve the MQF problem by the algorithms presented in this paper, given v, n, and p. One can use any variant of the generalized method (e.g., the uncoupled method, the single group coupled method, the multiple group method), or a combined approach. A combined approach 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 5. 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

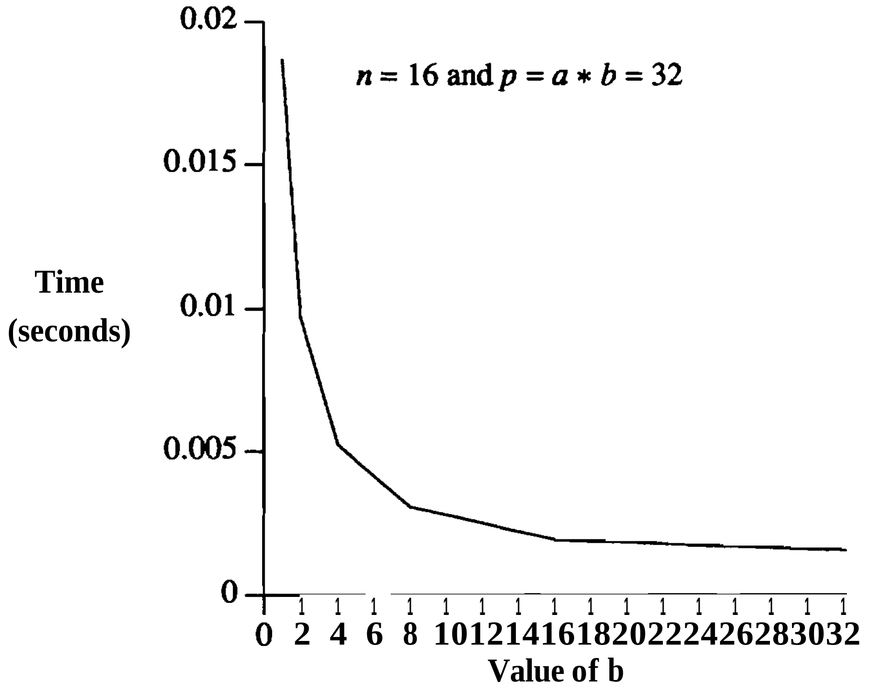
The goal of this section is to validate experimentally many of the theoretical results found earlier. Both the uncoupled and coupled methods for solving MQFs have been implemented on the 16-PE small-scale PASM prototype, the 64-PE nCUBE 2, and the 16K-PE MasPar MP-1. The implementations assume that a,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

Figure 21 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 (Figure 21a) and the nCUBE (Figure 21b) 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 presented in Subsection 5.4 that communication overhead is minimized when a is chosen as small as possible (i.e., a and b being a power 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 Figure (22a). 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. As one can see, 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

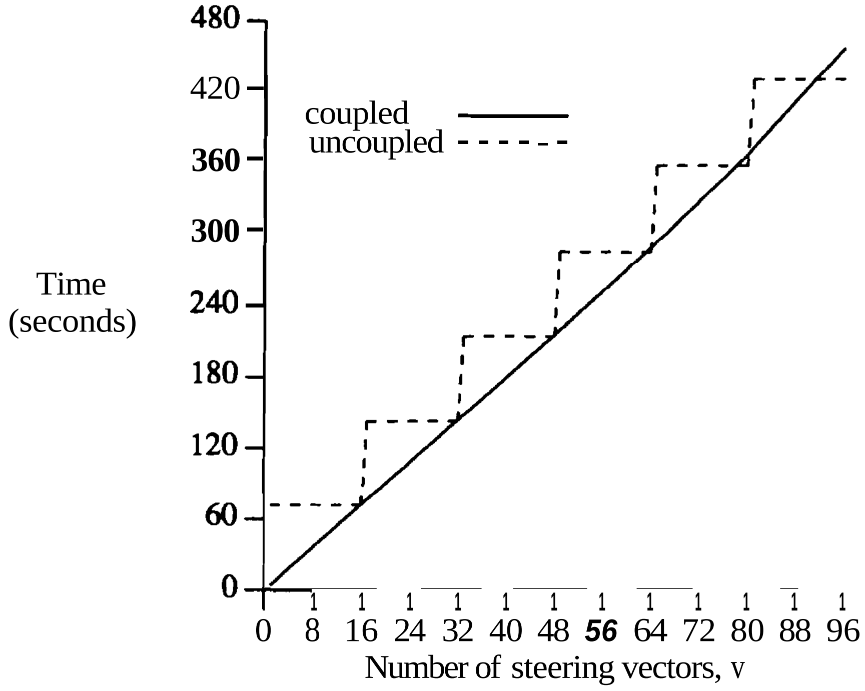


(a)

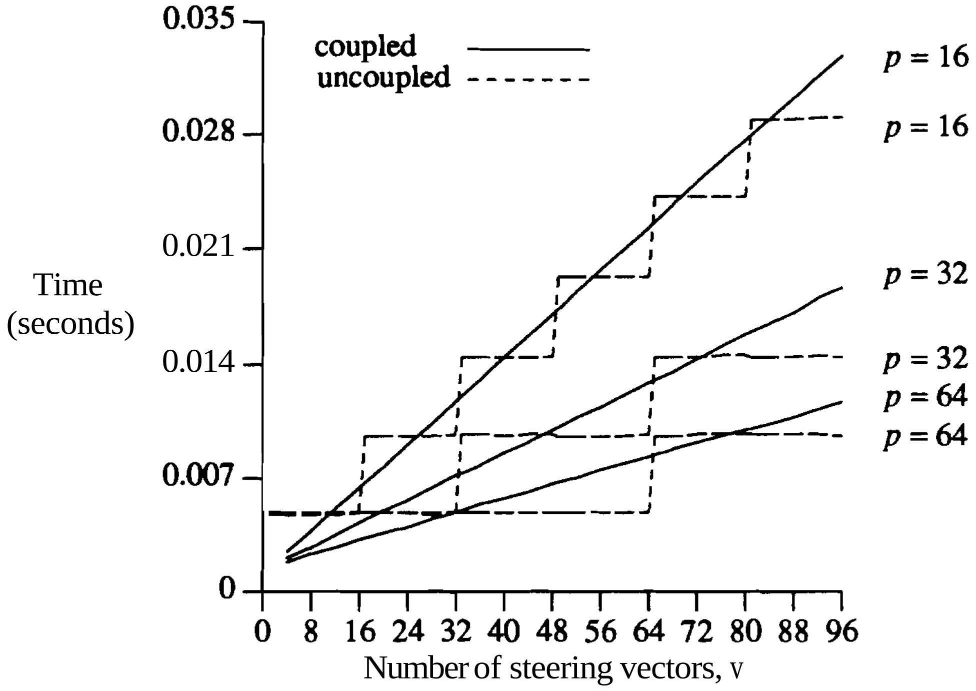


(b)

Figure 21: Communication cost for various p = a * b logical configurations for (a) the PASM prototype for $n = 128$, $p = 16$, and v = 2, 5, and (b) the nCUBE 2 for n = 16, $p = 32$, and v = 32.



(a)



(b)

Figure 22: 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 $p=16, 32, 64$.

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 (0.1% for v = 96, n = 128 on 16 **PEs**) of the entire execution time. These results corroborate the mathematical derivations in Table 5 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 1x8 **PEs**, 4 groups of **1×4 PEs**) is relatively insignificant when compared to the complex operations computation time. Consequently, only results from the single group coupled methods **are** reported for PASM for the performance of the coupled method.

Figure **22(b)** 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 × *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 ton).

The experimental results validate the theoretical conclusions summarized in Table 3 and Table 5. Table 5 says that whenever v = cp the uncoupled method outperforms the single group coupled method. Figure **22(b)** verifies this forp = **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 Figure **22(b),** when p = 32, the uncoupled method outperforms the single group coupled method for v = 24, 32, 48, 56, 64, 80, 88, 96. The exact points of intersection depend on the communication and computation cost relationship.

Now consider the computational complexities shown in Table 3. When v = ca, the table shows that "a" groups of (1 × n) **PEs** will outperform one group of p = a * *n* **PEs**. Let v = 80, a = 4, and n = 16. From Figure **22(b)**, one group of 64 = 4 * 16 **PEs** takes .O1 seconds. 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** seconds for v = 80, which is less than .O1 seconds. Thus, in this case, the multiple **group** method outperforms the single group method, as predicted by Table 3.

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** seconds), and step 2 would compute 16 vectors by the coupled method (**.0032** seconds). By this combined method, the MQF problem for v = 80 takes **.008**1 seconds on the **nCUBE** 2. The coupled and uncoupled methods take .O1 seconds and **.0097** seconds, 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, andp values, different combined approaches may need to be considered.

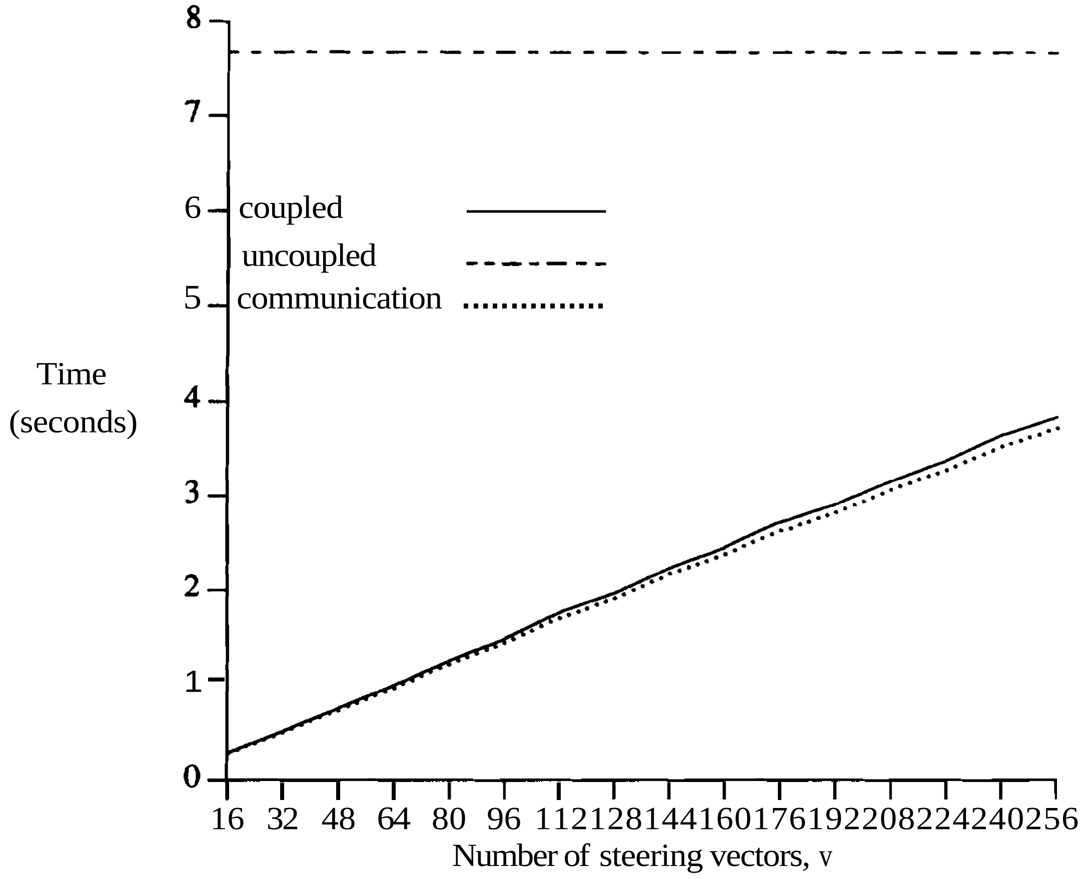


Figure 23: Execution time of the uncoupled and coupled data-parallel methods for the **MasPar** MP-1 for $n=128$ and $p=16{,}381$.

The **MasPar** MP-1 results in Figure 23 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. 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 *w* v), the more sophisticated coupled method is needed to exploit the available parallelism, &spite 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 **[Hay86]** provides a theoretical background for implementing an adaptive beamformer with MVDR and a serial algorithm for implementing this problem can be found in **[Sch86].** To date, no related work has been found that examines the problem of determining parallel implementations for computing multiple quadratic forms for the same mamx (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., [Ber89, ChS88, DeN81, FoO87, Mod88]).**

In **[Mod88],** different parallel algorithms for mamx 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 **[DeN81].** 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 **algorithm is** $O(\log_2 n)$ **when** $p = n^3$ **and** $O(n)$ **when** $p = n^2$.

Several papers **(e.g., [Ber89, ChS88, FoO87])** have been published that discuss performing matrix multiplication on hypercube machines by using a logical mesh of processors. These papers differ **from [DeN81], [Mod88],** and the discussion in this paper in that the analyses are specifically for hypercube machines. **[FoO87]** shows that the optimal performance of matrix multiplication on the Cosmic Cube **[Sei85]** is achieved (in terms of communication overhead and load balancing) by decomposing the problem into square sub-blocks. Similarly, **[ChS88]** discusses partitioning the matrix and the impact of communication overhead when performing mamx multiplication problem on an **nCUBE [HaM89].** The work present**ed in [Ber89]** is an extension of **[FoO87]** 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 mamces. 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 **applica-**

ble to the MQF problem discussed here.

10. Summary

Several parallel data-parallel implementations of the computationally intensive task of computing the MQF problem have been examined. Trade-offs among the implementations for various data-sizelmachine-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.

Both analytical and experimental results demonstrated that a combined approach may be better than any one method. 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.

References

[Arl88] R. Arlauskas, "iPSC/2 system: a second generation hypercube," *Third Conference on Hypercube Computers and Applications*, January 1988, pp. 38-42.

[ArW93] J. B. Armstrong, D. W. Watson, and H. J. Siegel, "Software issues for the PASM parallel processing system," in *Software for Parallel Computation*, Janusz S. Kowalik, Springer-Verlag, Berlin, 1993, to appear.

[Bat74] K. E. Batcher, "STARAN parallel processor system hardware," *AFIPS 1974 National Computer Conference*, May 1974, pp. 405-410.

[Bat76] K. E. Batcher, "The flip network in STARAN," *1976 International Conference on Parallel Processing*, August 1976, pp. 65-71.

[Bat77] K. E. Batcher, "STARAN series E," *1977 International Conference on Parallel Processing*, August 1977, pp. 140-143.

[Bat80] K. E. Batcher, "Design of a massively parallel processor," *IEEE Transactions on Computers*, Vol. C-29, No. 9, September 1980, pp. 836-844.

[Bat82] K. E. Batcher, "Bit serial parallel processing systems," *IEEE Transactions on Computers*, Vol. C-31, No. 5, May 1982, pp. 377-384.

[BeK91] T. B. Berg, S. D. Kim, and H. J, Siegel, "Limitations imposed on mixed-mode performance of optimized phases due to temporal juxtaposition," *Journal of Parallel and Distributed Computing*, Vol. 13, No. 2, October 1991, pp. 154-169.

[Ber89] J. Berntsen, "Communication efficient matrix multiplication on hypercubes," *Parallel Computing*, Vol. 12, No. 3,1989, pp. 335-342.

[BeS91] T. B. Berg and H. J. Siegel, "Instruction execution trade-offs for SIMD vs. MIMD vs. mixed-mode parallelism," *Fifth International Parallel Processing Symposium*, May 1991, pp. 301-308.

[Bla90] T. Blank, "The MasPar MP-1 architecture," *IEEE Compcon*, February 1990, pp. 20-24.

[BoD72] W. J. Bouknight, S. A. Denenberg, D. E. McIntyre, J. M. Randall, A. H. Sameh, and D. L. Slotnick, "The Illiac IV system," *Proceedings of the IEEE*, Vol. *60*, No. 4, April 1972, pp. 369-388.

[BrC90] E. C. Bronson, T. L. Casavant, and L. H. Jamieson, "Experimental application-driven architecture analysis of an SIMD/MIMD parallel processing system," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 1, No. 2, April 1990, pp. 195-205.

[ChD92] J. Choi, J. J. Dongarra, R. Pozo, and D. W. Walker, "ScaLAPACK: a scalable linear' algebra library for distributed memory concurrent computers," *Fourth Symposium on the Frontiers of Massively Parallel Computation*, October 1992, pp. 120-127.

[ChS88] V. Cherkassky and R. Smith, "Efficient mapping and implementation of matrix algorithms on a hypercube," *Journal of Supercomputing*, Vol. 2,1988, pp. 7-27.

[CrG85] W. Crowther, J. Goodhue, R. Thomas, W. Milliken, and T. Blackadar, "Performance measurements on a 128-node butterfly parallel processor," *1985 International Conference on Parallel Processing*, August 1985, pp. 531-540.

[DeN81] E. Dekel, E. Nassimi, and S. Sahni, "Parallel matrix and graph algorithms," *SIAM Journal of Computing,* Vol. 10, No. 4,1981, pp. 657-675.

[DuB88] P. Duclos, F. Boeri, M. Auguin, and G. Giraudon, "Image processing on a SIMD/SPMD architecture: OPSILA," *International Conference on Pattern Recognition,* November 1988, pp. 430-433.

[FiC88] S. A. Fineberg, T. L. Casavant, T. Schwederski, and H. J. Siegel, "Non-deterministic instruction time experiments on the PASM system prototype," *1988 International Conference on Parallel Processing,* August 1988, pp. 444-451.

[FiC91] S. A. Fineberg, T. L. Casavant, and H. J. Siegel, "Experimental analysis of a mixed-mode parallel architecture using bitonic sequence sorting," *Journal of Parallel and Distributed Computing,* Vol. 11, No. 3, March 1991, pp. 239-251.

[Fly66] M. J. Flynn, "Very high-speed computing systems," *Proceedings of the IEEE,* Vol. 54, No. 12, December 1966, pp. 1901-1909.

[FoJ88] G. C. Fox, M. A. Johnson, G. A. Lyzenga, S. W. Otto, J. K. Salmon, and D. W. Walker, *Solving Problems on Concurrent Processors, Volume 1,* Prentice Hall, Englewood Cliffs, NJ, 1988.

[FoO87] G. C. Fox, S. W. Otto, and A. J. G. Hey, "Matrix algorithms on a hypercube I: matrix multiplication," *Parallel Computing,* Vol. 4, No. 1,1987, pp. 17-31.

[Fou81] T. J. Fountain, "CLIP4: progress report," in *Languages and Architectures for Image Processing, M. J. B. Duff and* S. Levialdi, eds., Academic Press, London, England, 1981, pp. 281-291.

[GoG83] A. Gottlieb, R. Grishman, C. P. Kruskal, K. P. McAuliffe, L. Rudolph, and M. Snir, "The NYU Ultracomputer — designing an MIMD shared-memory parallel computer," *IEEE Transactions on Computers,* Vol. C-32, No. 2, February 1983, pp. 175-189.

[HaM89] J. P. Hayes and T. Mudge, "Hypercube supercomputers," *Proceedings of the IEEE,* Vol. 77, No. 12, December 1989, pp. 1829-1841.

[Hay86] S. Haykin, *Adaptive Filter Theory,* Prentice-Hall, Englewood Cliffs, NJ, 1986.

[Hun89] D. J. Hunt, "AMT DAP - a processor array in a workstation environment," *Computer Systems Science and Engineering,* Vol. 4, No. 2, April 1989, pp. 107-114.

[Jam87] L. H. Jamieson, "Characterizing parallel algorithms," in *The Characteristics of Parallel Algorithms,* L. H. Jamieson, D. B. Gannon, and R. J. Douglass, MIT Press, Cambridge, MA, pp. 65-100, 1987.

[KiN91] S. D. Kim, M. A, Nichols, and H. J, Siegel, "Modeling overlapped operation between the control unit and processing elements in an SIMD machine," *Journal of Parallel and Distributed Computing,* Special Issue on Modeling of Parallel Computers, Vol. 12, No. 4, August 1991, pp. 329-342.

[KuD86] D. J. Kuck, E. S. Davidson, D. H. Lawrie, and E. H. Sameh, "Parallel supercomputing today and the Cedar approach," *Science,* Vol. 231, February 1986, pp. 967-974.

[Law75] D. H. Lawrie, "Access and alignment of data in an array processor," *IEEE Transactions on Computers,* Vol. C-24, No, 12, December 1975, pp. 1145-1155.

[LiM87] G. J. Lipovski and M. Malek, ***Parallel Computing: Theory and Comparisons,*** John Wiley and Sons, Inc., New York, NY, 1987.

[Mod88] J. J. Modi, ***Parallel Algorithms and Matrix Computations,*** Oxford University Press, New York, NY, 1988.

[Pan91] C. M. Pancake, "Software support for parallel computing: where are we headed?," ***Communications of the ACM,*** Vol. 34, No. 11, November 1986, pp. 53-64.

[Pea77] M. C. Pease III, "The indirect binary n-cube microprocessor array," ***IEEE Transactions on Computers, Vol. C-26, No. 5, May 1977, pp. 458-473.***

[PfB85] G. F. Pfister, W. C. Brantley, D. A. George, S. L. Harvey, W. J. Kleinfelder, K. P. McAuliffe, E. A. Melton, V. A. Nurton, and J. Weiss, "The IBM Research Parallel Processor Prototype (RP3): introduction and architecture," ***1985 International Conference on Parallel Processing,*** August 1985, pp. 764-771.

[PhW93] M. Philippsen, T. Warschko, W. Tichy, and C. Herter, "Project Triton: towards improved programmability of parallel machines," ***26th Hawaii International Conference on System Sciences,*** January 1993, pp. 192-201.

[Sch86] R. Schreiber, "Implementation of adaptive array algorithms," ***IEEE Transactions on Acoustics, Speech, and Signal Processing,*** Vol. ASSP-34, No. 5, 1986, pp. 1038-1045.

[Sei85] C. L. Seitz, "The Cosmic Cube," ***Communications of the ACM,*** Vol. 28, No. 1, January 1985, pp. 22-33.

[SiA92] H. J. Siegel, J. B. Armstrong, and D. W. Watson, "Mapping computer-vision-related tasks onto reconfigurable parallel-processing systems," ***Computer,*** Vol. 25, No. 2, February 1992, pp. 54-63.

[Sie90] H. J. Siegel, ***Interconnection Networks for Large-Scale Parallel Processing: Theory and Case Studies, Second Edition,*** McGraw-Hill, New York, NY, 1990.

[SiN90] H. J. Siegel, W. G. Nation, and M. D. Allemang, "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.

[SiS81] H. J. Siegel, L. J. Siegel, F. C. Kemmerer, P. T. Mueller, Jr., H. E. Smalley, Jr., and S. D. Smith, "PASM: a partitionable SIMD/MIMD system for image processing and pattern recognition," ***IEEE Transactions on Computers,*** Vol. C-30, No. 12, December 1981, pp. 934-947.

[SiS82] L. J. Siegel, H. J. Siegel, and P. H. Swain, "Performance measures for evaluating algorithms for SIMD machines," ***IEEE Transactions on Software Engineering,*** Vol. SE-8, No. 4, July 1982, pp. 319-331.

[SiS87] H. J. Siegel, T. Schwederski, J. T. Kuehn, and N. J. Davis IV, "An overview of the PASM parallel processing system," in ***Computer Architecture,*** D. D. Gajski, V. M. Milutinovic, H. J. Siegel, and B. P. Furht, eds., IEEE Computer Society Press, Washington, DC, 1987, pp. 387-407.

[Sto80] H. S. Stone, "Parallel computers," in ***Introduction to Computer Architecture, Second Edition,*** H. S. Stone, ed., Science Research Associates, Inc., Chicago, IL, 1980, pp. 363-425.

[Str86] G. Strang, *Introductionto Applied Mathematics,* Wellesley-Cambridge Press, Welles-ley, MA, 1986.

[ThN81] S. Thanawastien and V. P. Nelson, "Interference analysis of shuffle/exchange net-works," ***IEEE Transactions on Computers,*** Vol. C-30, No. 8, August 1981, pp. 545-556.

[TuR88] L. W. Tucker and G. G. Robertson, "Architecture and applications of the Connection Machine," ***Computer,*** Vol. 21, No. 8, August 1988, pp. 26-38.

[WuF80] C.-L. Wu and T. Y. Feng, "On a class of multistage interconnection networks," ***IEEE Transactionson Computers,*** Vol. C-29, No. 8, August 1980, pp. 694-702.