# USC–SIPI REPORT #135

## Kronecker and Array Algebra for Parallel Image Processing

by

**Daniel G. Antzoulatos**
**December 1988**

Signal and Image Processing Institute
**UNIVERSITY OF SOUTHERN CALIFORNIA**
Department of Electrical Engineering-Systems
3740 McClintock Avenue, Room 404
Los Angeles, CA 90089-2564 U.S.A.

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

| REPORT DOCUMENTATION PAGE | | | | Form Approved OMB NO. 0704-0188 |
|---|---|---|---|---|
| 1a. REPORT SECURITY CLASSIFICATION<br>UNCLASSIFIED | | 1b. RESTRICTIVE MARKINGS | | |
| 2a. SECURITY CLASSIFICATION AUTHORITY | | 3. DISTRIBUTION/AVAILABILITY OF REPORT<br>Distribution unlimited as it appears on the report. | | |
| 2b. DECLASSIFICATION/DOWNGRADING SCHEDULE | | | | |
| 4. PERFORMING ORGANIZATION REPORT NUMBER(S)<br>USC-SIPI Report # 135 | | 5. MONITORING ORGANIZATION REPORT NUMBERS(S) | | |
| 6a. NAME OF PERFORMING ORGANIZATION<br>University of Southern California<br>Signal and Image Processing Institute | 6b. OFFICE SYMBOL<br>(If Applicable) | 7a. NAME OF MONITORING ORGANIZATION<br>US Army Research Office<br>Physics Division | | |
| 6c. ADDRESS (City, State, and Zip Code)<br>Department of Electrical Engineering<br>University Park/MC-0272<br>Los Angeles, CA 90089 U.S.A. | | 7b. ADDRESS (City, State, Zip Code)<br>Attn. Dr. Bob D. Guenther<br>P.O. Box 12211<br>Research Triangle Park, NC 27709-2211 | | |
| 8a. NAME OF FUNDING/SPONSORING<br>Defense Sciences Office<br>Defense Advanced Research Projects Agency | 8b. OFFICE SYMBOL<br>(If Applicable) | 9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER<br>P-21739-PH-A<br>Contract # DAAG29-84-K-0066 | | |
| 8c. ADDRESS (City, State, and Zip Code)<br>Attn. Dr. Andrew Yang<br>1401 Wilson Blvd.<br>Arlington, VA 22209 | | 10. SOURCE OF FUNDING NUMBERS | | |
| | | PROGRAM | PROJECT NO.<br>P-21739-PH-A | TASK NO. | WORK UNIT ACCESSION NO. |
| 11. TITLE (Include Security Classification)<br>Kronecker and Array Algebra for Parallel Image Processing<br>(UNCLASSIFIED) | | | | |
| 12. PERSONAL AUTHOR(S)<br>Daniel G. Antzoulatos | | | | |
| 13a. TYPE OF REPORT<br>Technical Report | 13b. TIME COVERED | 14. DATE OF REPORT (Year, Month, Day)<br>89-02-15 | 15. PAGE COUNT<br>142 pages | |
| 16. SUPPLEMENTARY NOTATION | | | | |
| 17 COSATI CODES<br>FIELD / GROUP / SUB-GROUP | | 18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)<br>Interconnection networks, parallel processing, shuffle permutation, 2-D Omega processor, Hadamard transfomations, and matrix rotations & transposition. | | |

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

This report presents a mathematical framework for implementing highly concurrent tasks on three-dimensional multilayered massively parallel synchronous computing structures. The framework is an extension and unification of ideas and concepts found in signal processing, parallel processing, and large scale computing. It takes advantage of the duality between Kronecker and array algebras. The former is matrix algebra augmented with Kronecker products and shuffle permutations. Array algebra - a derivative of tensor analysis - is extended for more flexibility. Useful identities and theorems for both algebras are presented/derived. Multilayered architectures are given new classifications. Closely examined are those architectures that consist of cascaded pairs of a global interplane communication structure and a regular array of processing elements - isoplanar homosyndetic architectures. Of particular interest are global interconnections of the shuffle permutation class. The primary example of such an architecture is an envisioned 2-D Omega processor - a 2-D processing extension of the Omega network. Canonical forms of 2-D shuffle/processing stages are derived. The computational tasks addressed are those which can be cast in the context of matrix algebra. Mappings of linear tranformations are demonstrated including 2-D Hadamard transformations, matrix rotations, and transposition. Potential optical implementation technologies are discussed and the applicability of Kronecker algebra in the implementation of shuffles using strategically located thin lenses is demonstrated.

| 20. DISTRIBUTION/AVAILABILITY OF ABSTRACT | 21. ABSTRACT CLASSIFICATION | |
|---|---|---|
| [X] UNCLASSIFIED [ ] SAME AS REPORT [ ] DTIC USERS | UNCLASSIFIED | |
| 22a. NAME OF RESPONSIBLE INDIVIDUAL<br>Professor Alexander A. Sawchuk | 22. TELEPHONE (Include Area Code)<br>(213) 743-5527 | 22c. OFFICE SYMBOL |

*In memory of my father,*

*Angelos*

# Acknowledgments

Writing a dissertation is like running a marathon on an unmarked course: you madly wander around for the start; once there, you cling to what seems to be the proper course and, after some distance, slowly become paranoid of an elusive end.

All my dissertation committee members are appreciated for coaching me to the finish line. I especially would like to thank my committee chairman, Dr. Alexander (Sandy) Sawchuk, who as director of the Signal and Image Processing Institute, provided the freedom and tools necessary for performing this research and more importantly, shined a spotlight whenever I needed to stay on course. Dr. C.S. Raghavendra, through his teachings, provided the initial impetus and foundation for this work. I am thankful to Dr. William Harris, Jr. for graciously stepping in as the mathematics committee member replacing Dr. K. McCurley. The constructive comments of the remaining two members of my original guidance committee, Drs. Kai Hwang and Keith Jenkins are, of course, equally appreciated.

The suggestions of Drs. John W. Brewer and William J. Vetter were invaluable in steering me in the right direction. A review of the first two chapters by Josef Giglmayr provided encouragement and additional references.

I am grateful to Dr. Allan Weber for unreservedly assisting me every time I had a computer operation problem. The discussions we had on "how things should have been done" were always enjoyable. The administrative support of Linda Varilla, Jerene Brooks, Gloria Bullock and Delsa Tan was indeed of great help.

The trials and tribulations of a marathon are lessened when shared with fellow runners like: Bob Frankot, Ted Broida, Richard Hansen, Gregory Yovanof, Shankar Chatterjee, Kung-Shiuh Huang and Tom Hebert. Especially valued is the friendship and empathy of my long-time office mates Herb Barad and Dimitrios Kalivas.

I could not have completed this course without the emotional support and patience of my family and friends. The constant encouragement of my mother, Barbara, and my

aunt, Sondra Taubenblatt, undoubtedly kept me going particularly during the rough spots. I am indebted to Angelo Karamanis and my brother, Michael, for making sure that I took time out to occasionally revive myself and reflect on the values of life.

I would not have embarked on this course in the first place if it were not for Drs. Demetrios and Helen Boussalis who inspired me to the degree that role models should.

I am certainly grateful to TRW, Inc., for the Master's and Ph.D. fellowships they generously awarded me.

Lastly, I would like to thank the creators of Macintosh, LaTeX, PostScript and various other recent generation hardware and software products for bringing common man closer to drawing and typesetting perfection ... and curse them for keeping it an addictive illusion by leaving in just enough incompatibilities, gaps and bugs.

During this endeavor, there were relationships I neglected to nourish—and that I truly regret.

# Contents

# List of Figures

# List of Tables

# Abstract

This dissertation presents a mathematical framework for implementing highly concurrent tasks on three-dimensional multilayered massively parallel synchronous computing structures. The framework is an extension and unification of ideas and concepts found in signal processing, parallel processing and large scale computing. It takes advantage of the duality between Kronecker and array algebras. The former is matrix algebra augmented with Kronecker products and shuffle permutations. Array algebra—a derivative of tensor analysis—is extended for more flexibility. Useful identities and theorems for both algebras are presented/derived. Multilayered architectures are given new classifications. Closely examined are those architectures that consist of cascaded pairs of a global interplane communication structure and a regular array of processing elements—*isoplanar homosyndetic architectures.* Of particular interest are global interconnections of the shuffle permutation class. The primary example of such an architecture is an envisioned *2-D Omega processor*—a 2-D processing extension of the Omega network. Canonical forms of 2-D shuffle/processing stages are derived. The computational tasks addressed are those which can be cast in the context of matrix algebra. Mappings of linear transformations are demonstrated including 2-D Hadamard transformations, matrix rotations and transposition. Potential optical implementation technologies are discussed and the applicability of Kronecker algebra in the implementation of shuffles using strategically located thin lenses is demonstrated.

# Chapter 1

# Introduction

## 1.1 The Nature of Image Processing

*Image processing* is a widely diverse field concerned with processing pictorial information. Images are usually snapshots of some measurable aspect of the physical universe, such as visible or infrared pictures, tomographic projections, etc. Computer processing of such information requires that it be spatially and temporally sampled as well as quantized in magnitude. However, the digitization process alone renders the captured information only an approximation of the actual analog physical events. In addition, the captured data is inherently distorted either by the basic acquisition methodology and/or by imperfectly operating components.

One major goal of image processing is to reconstruct as much of the original observation as possible using the available data. This also includes enhancing the data in a way that exposes certain interesting events or features to a human or a machine.

Over the past thirty years, the field has built an *ad hoc* collection of procedures to fulfill its goals. Some are elegant mathematical principles while others are heuristic methods with no underlying proof. One may categorize image processing tasks according to the input and output data structures:

- signal processing: *image* $\mapsto$ *image*
- classification: *image* $\mapsto$ *features*

- understanding: *features* $\mapsto$ *perception*

The signal processing tasks are characterized by massive amount of data handling but only a moderate amount of processing per datum. During classification, the data volume is somewhat reduced but the processing per item is greater. Image understanding deals with only a few key features but often requires an extraordinary amount of knowledge processing. In a rough sense, all three categories have about the same *processing-bandwidth* product, i.e., required total raw processing power.

Digital processing of images demands a great deal of computing power. In the digital domain the fundamental data structure, an *image*, is commonly a regularly-spaced two-dimensional array. A typical scientific image is of large scale ($> 10^6$ samples or pixels) and therefore each pixel operation corresponds to a total image count on the order of a million operations. Even simple image processing tasks require on the order of tens or hundreds of operations per sample.

Processing images at realtime rates (order of tens per second) requires computing power on the order of billions of operations per second, $\mathcal{O}$(GOPS). This is two to four orders of magnitude higher than the computing power of currently available single CPU machines. This gap is constantly expanding as image acquisition, storage and display technologies rapidly outpace processing capabilities. Thus, the future only holds the promise of increased data volume and the need for greater and faster image processing capabilities.

### 1.1.1 Parallel Processing Requirements

A logical option is to allocate more processors to the job—all working in parallel. Successfully applying parallel techniques to image processing, however, hinges on two key issues:

- Identifying the tasks that lend themselves to parallel processing.
- Devising the appropriate algorithm and underlying architecture for the parallel task.

Currently, there is no comprehensive and unifying framework in which to resolve these issues. There is no acknowledged architectural model that is well suited to the full spectrum of image processing tasks.

The goal of this research is to address such issues and to present new and novel perspectives to the design of parallel image processing systems. In the process, a particular class of promising architectures is explored. These architectures are characterized by parallel computing on a large scale as well as extensive modularity and scalability. It is envisioned that such architectures will be well suited for the emerging technology of *digital optical computing* where massive interconnections in all three dimensions of space are possible.

## 1.2 Evolution of Image Processing Architectures

It has been obvious from the early days of electronic computing that sequential *von Neumann* types of architectures are inadequate for processing spatial (2-dimensional) data. A large number of multiprocessor architectures for the solution of two-dimensional problems have been proposed over the last three decades. One of the earliest publications was that of Unger [64] in 1958. This was a seminal piece of work that described the use of a *four-connected* cellular array of 1-bit ALUs for solving a class of pattern recognition problems. The basic model is shown in Fig. 1.1. Other design ideas Unger introduced were: multiple processing planes, parallel (electro-optical) input, pipelined shifting, etc. This work set the tone for subsequent hardware implementations of parallel processors (e.g., SOLOMON, ILLIAC III, CLIP4, MPP and others) [46, 42].

However, it has taken over 25 years for technology to reach a level of maturity sufficient for implementing most of Unger's design at a reasonable cost and scale. The *Geometric Arithmetic Parallel Processor (GAPP)* architecture currently produced by the NCR Corp. is such a cellular array of $6 \times 12$ processors on a chip.

Obtaining significant performance improvements in many scientific tasks, e.g., image processing, necessitates the use of parallel processing on a large scale. Because of the massive amount of data to be processed, many processors must work on similar as well

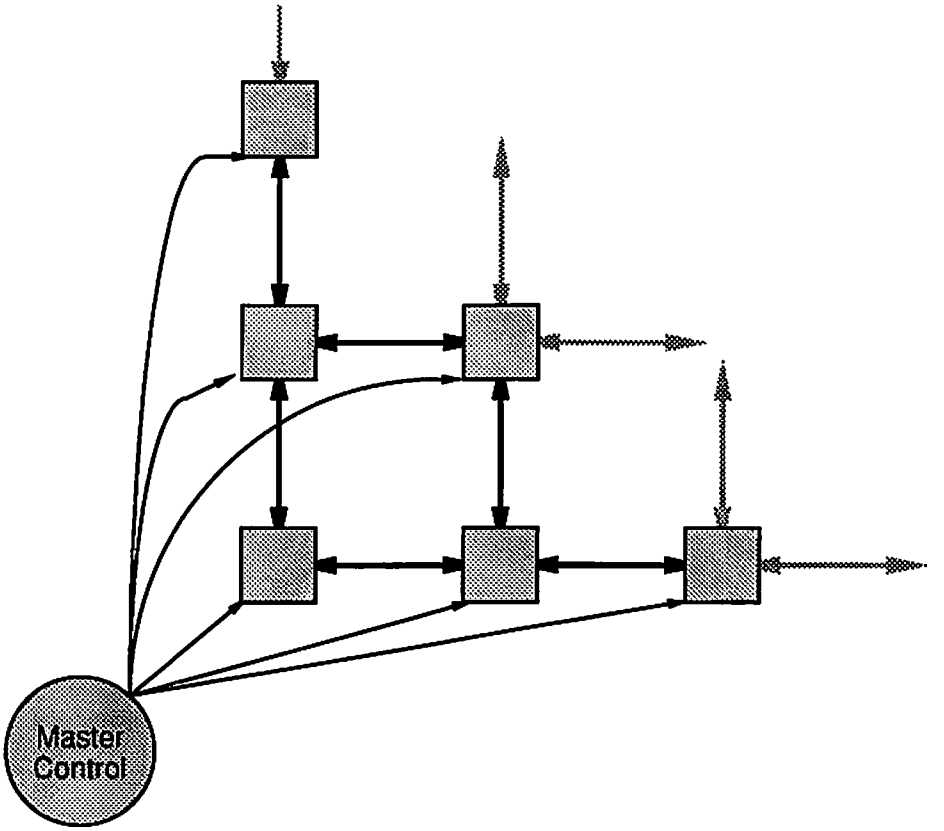



Figure 1.1: Unger's Spatial Computer

as different parts of the problem at once. This, however, introduces an overall design complexity [1] much greater than that experienced in the early years of processor design. A set of formal design tools is a prerequisite for the systematic exploitation of massively parallel structures.

## 1.3 Architectural/Algorithmic Issues in Parallel Computing

The algorithmic and architectural approach taken to solve a problem, the implementation technology and the exact formulation of the problem itself are currently highly interdependent issues (Fig. 1.2). For example, the availability of a new technology

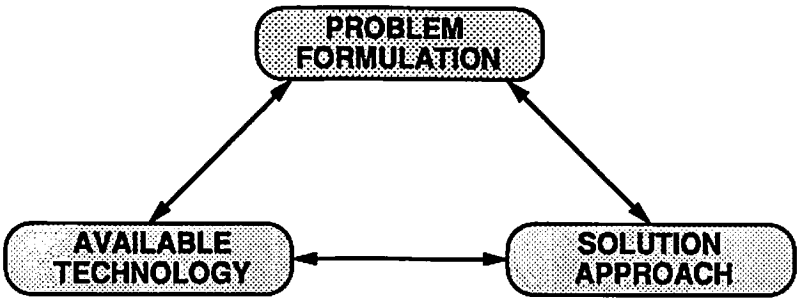



Figure 1.2: Solution Issues

may lead to a solution of a previously baffling problem. If, however, the statement of the problem is implicitly tied to the old technologies, then a new solution may never surface. It is difficult to visualize an efficient parallel implementation of a problem which is stated only as a sequential algorithm. By taking one or more steps back to a higher level of abstractness, the problem may reveal a world of concurrency which was previously unthinkable.

Therefore, it is important to eliminate any explicit or implicit assumptions that may exist among these issues. The statement of the problem should not assume any particular solution approach, and the solution approach should not assume the availability (or lack) of any particular technology. The issues need to be made as abstract as possible.

Ideally, there should be no link between problem formulation and technology. The problem should be stated such that it can be mapped into a near-complete set of solutions which are filtered out according to available technology. In order to achieve this relationship, a more abstract design methodology is needed. This will the subject of the following sections.

### 1.3.1 New Terminology

Design terminologies of the past must be refined in order to convey the abstractness required of new design methods. It is evident that there are a number of terms in use today that are too ambiguous or carry implicit and undesirable restrictions. For example,

the term *algorithm* has been given several definitions over the years. A representative definition is found in [44]:

> "Given both the problem and the device, an *algorithm* is the precise characterization of a method of solving the problem, presented in a language comprehensible to the device."

The important fact that an algorithm cannot be divorced from the environment in which it operates is often forgotten by a great many users of the term. Therefore, one cannot really talk about *algorithm-to-architecture* mapping, because when stating an algorithm one also implies the architecture on which it is to execute. Hopefully, a bit more precision will be added to the design vocabulary by using the following terms:

**Computational Task** The first step, of course, in any design is to identify the problem to be solved. Once the abstract problem has been understood, then it can be decomposed into one or more *computational tasks.* The specificity of these tasks should be general enough to allow a wide choice of implementations. For example, a Discrete Fourier Transform, a 2-D convolution, and an image histogramming operation are all considered computational tasks—they specify *what* needs to be done but not *how.* On the other hand, a Fast Fourier Transform is too specific because, in its present usage, it assumes a particular underlying computational structure.

A computational task is independent of any implementation parameters and therefore may be realized by a number of different algorithms. The form in which the computational task is presented—its *casting*— may be mathematical (e.g., matrix equation) or it may be a set of high level language statements. It is, basically, a *declarative* description of the operation to be performed.

A computational task may also be called the *realized task* or *effected operation* when referring to what has been executed on a processor.

**Processing Flow** Because a single processor was often assumed to be the underlying computing engine, an algorithm also carried with it an implied architecture. In parallel processing, there are numerous architectural possibilities and for each one an equally

vast number of algorithms that can effect a given task. In order to convey a sense of how a given task is carried out, both the algorithm and the architecture must be explicitly described. This has led to the suggestion of new terms, such as "algotecture" and "archirithm". They denote the integrated nature of algorithm and architecture but are awkward terms.

Instead, the expression *processing flow* will be used to denote the processing and communication steps needed to realize a computational task. The communication steps imply that a certain class of architectures are to be used. For clarity, the term "algorithm" will be avoided in the context of parallel processing.

**Computational Task-to-Processing Flow Mapping** In recent literature, the terms "algorithm-to-architecture" mapping and "architecture-to-algorithm" mapping have been used to describe the formal process of deriving an appropriate match between a parallel algorithm and a parallel architecture. The first label stems from the traditional design standpoint where there is a fixed architecture and the computational task is algorithmically manipulated to fit it— the *software solution.* Proponents of the architecture-to-algorithm label consider the algorithm to be fixed or (semi-fixed) while the goal is to find the most efficient architecture which will execute it—the *hardware solution.*

If both the algorithm and the architecture are variable, then there seems to be a dilemma as to what is mapped to what. Again, this is due to the inadequacy of the terms "algorithm" and to a lesser extent "architecture". A much more precise description of this mapping is: "task-to-algorithm/architecture," or using the new terminology, *computational task-to-processing flow* (or simply *task-to-flow*).

### 1.3.2 Parallel Architecture Classification

### (2-D ... or not 2-D)

Describing a parallel architecture by referring to the order of its dimensionality may seem like a reasonable classification approach. For example, a pyramid computer may be referred to as a 2-D processor ... or is it 3-D? The answer depends on one's perspective.

From the point of view of the architect of the pyramid, it is obviously a 3-D processor because of its conceptual multilayered structure. From an implementation point of view, a simple pyramid could be manufactured using conventional VLSI and therefore would be laid out as a 2-D circuit ... or it could be implemented in optics and be a 3-D processor. From a systems engineering ("black box") point of view , the pyramid accepts and processes two-dimensional data arrays and therefore the pyramid should be called a 2-D processor. On the other hand, if one also considers the serial output emanating from the root processor (tip of the pyramid), is the structure then a 2D-to-1D converter?

Of course, all of these viewpoints are valid. The problem lies with the ambiguous context of the dimensionality designation. With multiprocessor architectures, dimensionality may refer to distinct aspects of a computing structure. It may be the conceptual structure or the physical structure or the data processing method or the mathematical task that is being executed. In an effort to minimize any confusion, the following terminology will be used to describe the major dimensionality aspects of multiprocessor architectures:

**physical structure** – the physical layout of the architecture. Usually refers to the lowest implementation level, i.e., planar or 2-D (VLSI); volume or 3-D (optics).

**conceptual structure** – the abstract geometrical representation of the architecture. It may be a single processor (point), a series of processors cascaded along a *line* (1-D), a *plane* of processors (2-D), a *volume (or cube)* of processors (3-D), or it may be a higher dimensional geometry, i.e., a *hyper-volume* of processors. The driving factor of the architectural geometry is, usually, the underlying processor connectivity or topology. As a result, interprocessor communication descriptors (e.g., ring, star, mesh, cube, etc.) are frequently used which specify either explicitly or implicitly the conceptual dimensionality.

**processing flow (structure)** – the dimensionality of the data communicated throughout the system. The processing flow may be characterized at several levels with the top most being *serial* and *parallel.* Serial processing may involve concurrency (pipelining). Parallel concurrency may be further qualified according

| Physical Structure | Conceptual Structure | Processing Flow | Effected Operation |
|---|---|---|---|
| | single (point) | serial | scalar |
| planar (2-D) | linear (1-D) | N-parallel (1-D) | vector (1-D) |
| volume (3-D) | planar (2-D) | $N^2$-parallel (2-D) | matrix (2-D) |
| | hyper-volume (M-D) | $N^M$-parallel (M-D) | hyper-array (M-D) |

Table 1.1: Multiprocessor Structural Aspects and Typical Dimensionality Designations

to the structure and amount of data being processed in one time-slice[1]. For example, *N-parallel* processing implies the simultaneous manipulation of an $N$-element data structure[2]. Similarly, *$N^2$-parallel* (or $N \times M$-parallel) processing describes the clocking of data structures of size $N^2$ (or $N \times M$).

**effected operation (structure)** – the dimensionality of the realized computational task. This defines the algebraic dimensionality of the input/output data structures. Ex., 2-D discrete fourier transform, vector (1-D) permutation, etc.

The various structural aspects of an architecture and the typical dimensionality designations used within the context of each aspect, are summarized in Table 1.1.

Although for many architectures, some of these aspects may indeed be tautological, in general, they should be considered independent and distinct. The question becomes,

*Which structural aspect(s) should be used for classifying a multidimensional computer architecture?*

Ideally, a complete classification would require a composite descriptor of all of the above structural aspects. Traditionally however, a single dominant aspect is used; one that describes the most exotic or innovative feature. It may be the processing flow,

[1]Based on the execution time granularity of the processors

[2]The terms *vector-processing* and *array-processing* have also been used to describe parallel processing of data organized as one or two dimensional structures. However, they are used in too many other contexts and are too ambiguous to have with any universal meaning.

e.g., pipelined, systolic[3], $N$-parallel, etc., or the conceptual structure, e.g., pyramid, hypercube, etc. The unspecified features are considered either variable or implicitly fixed.

In most fields, the most appealing aspect has been the method by which data is manipulated or processed—the processing flow. In optical computing, for example, processors are characterized as *$N$-parallel* and *$N^2$-parallel*; it is implicit that the conceptual structure is three-dimensional.

In the context of this study, multiprocessor architectures are examined according to two primary aspects: the *processing flow* and the *conceptual structure*. However, the former is considered the *default dimensionality classifier* when only one dimensionality designation is given.

The conceptual structure is more elaborately classified in the following sections.

### 1.3.3 Conceptual Structure

Processors which have a multidimensional conceptual structure are typically characterized by their underlying interconnection architecture. In most cases, this is appropriate because the interconnection network is fixed and the processing elements are kept general or unspecified.

One popular example is that of the *hypercube* or *Boolean n-cube*. This is a particular network topology in which the interconnections for each processor are fixed—but increase as more processors are added to the system. Typically, the number of processors is a power of 2, i.e. $2^k$, where $k$ refers to the abstract dimensionality of the architecture. Each processor communicates with its $k$ nearest neighbors. An example of a 4-dimensional hypercube is shown in Fig. 1.3. An algorithm that can effectively use all processors in parallel could, in principle, surpass the power of today's supercomputers. It has been suggested that hypercube architectures are well suited for many image processing operations – both local and global operations. For example, global statistical computations for an $N \times N$ image can be done on the order of $N$ steps.

[3]Actually a misnomer. A more appropriate term is *peristaltic*.

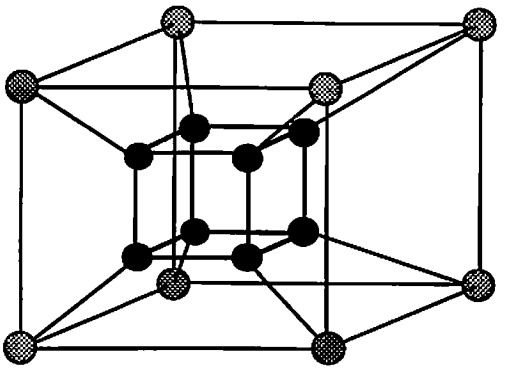


Figure 1.3: Hypercube Example (4-dimensional)

The complexity of a hypercube node is typically a multibit computer which, at present, imposes a limit on the dimensionality of the cube. One exception is the *Connection Machine* [19, 18]: a massively parallel machine with *O(10000)* boolean processors and a hybrid mesh-hypercube network. The prototype contains $2^{16}$ processing cells configured in a Boolean *12-cube* structure of $4 \times 4$ mesh connected cells.

The above are examples of fixed interconnection structures. In the following sections, two other classes of fixed interconnection architectures will be described: *hierarchical* and *isoplanar* (non-hierarchical).

### Hierarchical Architectures

In order to mechanize cerebral functions such as vision, it seems reasonable to use the human brain as a blueprint. Unfortunately, an accurate model of human perception is not available. This does not mean that what is known or has been hypothesized about how biological vision systems work could not contribute to efficient image processing architectures. One aspect that is generally acknowledged is the hierarchical manner in which biological vision systems process information. There seems to be a progression in processing from simple, local information to increasingly more abstract and global information. This has motivated many researchers (Rosenfeld [49], Tanimoto [60] and Uhr [62] among others) to suggest hierarchical architectures, such as pyramids (see Fig. 1.4) or cones, for the broad spectrum of image processing tasks.

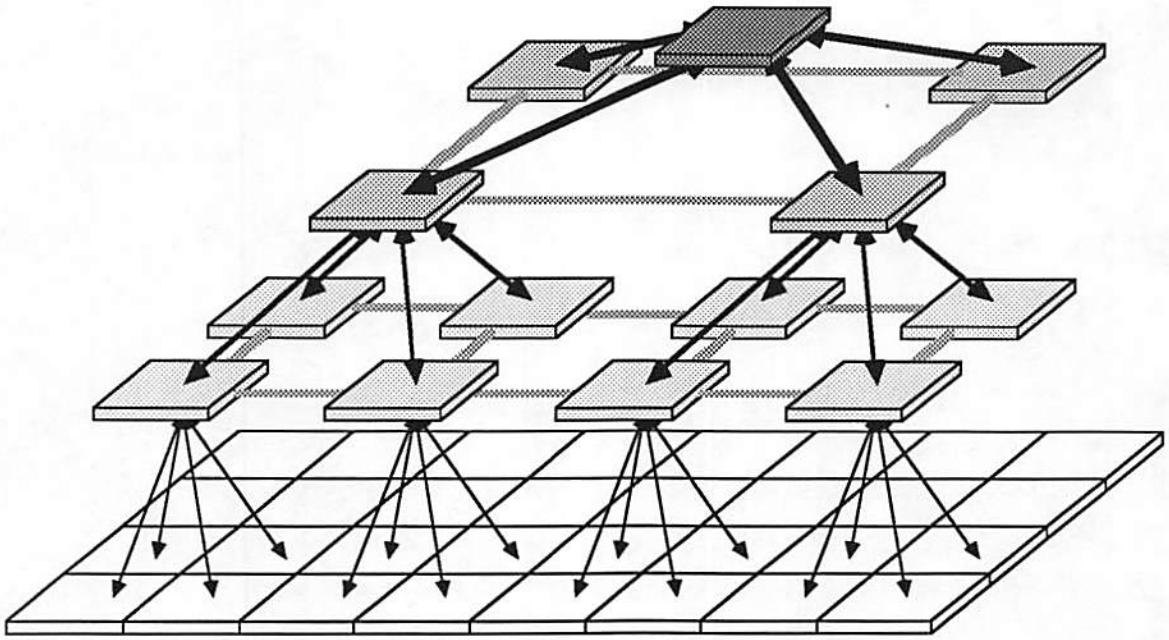


Figure 1.4: Pyramid Architecture

Many variations are possible. In [61] the importance of hierarchical architectures is discussed in terms of three candidate tessellation schemes: square, triangular and hexagonal. Each cell has a hierarchical neighborhood either by edge-adjacency or by vertex adjacency and includes the corresponding lateral neighborhood. Each hierarchical neighborhood can be considered a vector or pattern. A class of cellular logic operations on binary vectors (bit-pyramid) is defined.

As natural and as appealing as hierarchical architectures are, they exhibit certain disadvantages not unlike those found in von Neumann machines. For example, global communication tasks (e.g., histogramming) present a bottleneck at the higher levels. Such problems in general can be dealt with by increasing communication bandwidth and processor complexity at the higher levels. Various other modifications and enhancements have been suggested [4, 62, 63] which address these and other problems.

In addition to the functional difficulties of hierarchical architectures there are the physical problems of implementation. Modularity and uniformity are not outstanding features of the basic hierarchical model and even less so in the enhanced versions.

### Isoplanar Architectures

A different class of conceptual 3-D architectures is a multiplane structure where each plane is of the same size, intraplane communication structure and processing node complexity. Such a structure is conceptually shown in Fig. 1.5 and called *isoplanar*.

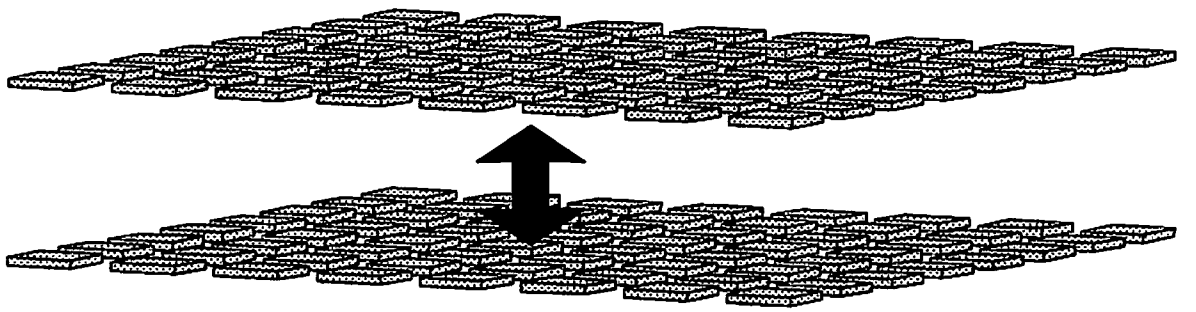


Figure 1.5: 3-D Isoplanar Architecture

If the communication between planes is global in nature, then there might be a possibility of a modular, uniform architecture that can accomplish a broad range of functions with a minimum number of communication steps.

**Prism Machine** One alternative to the pyramid architecture is that of the *Prism Machine* as suggested by Rosenfeld [50]. As shown in Fig. 1.6, each plane of this conceptual 3-D machine is of the same size, $n \times n$. A total of $\log_2 n$ planes comprise the prism machine. On each plane the processors are 4-connected, i.e., north, south, east and west. Between planes there is a butterfly-like 2-D connection. A processor at position $(i,j)$ at level $k$, denoted $(i,j)_k$, has three connections to the plane above:

$$(i,j)_{k+1}\ ,\ (i+2^k,j)_{k+1}\ ,\ (i,j+2^k)_{k+1},$$

and three to the plane below:

$$(i,j)_{k-1}\ ,\ (i-2^{k-1},j)_{k-1}\ ,\ (i,j-2^{k-1})_{k-1},$$

(where the additions/subtractions are *modulo* $n$)

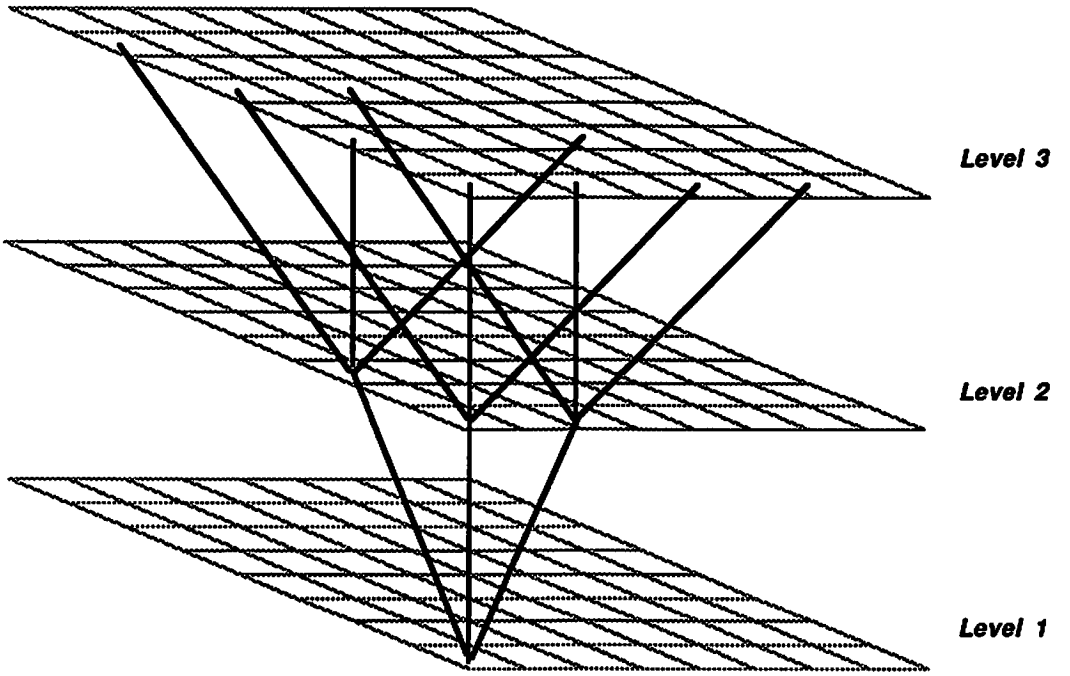



Figure 1.6: Prism Machine: three $8 \times 8$ planes

Note that the number of input/output links is fixed regardless of the size of the machine. Thus, unlike the hypercube, for instance, the number of I/O ports per processor is constant. The number of processors required in this configuration is approximately $\frac{3}{4}\log_2 n$ times more than that of the pyramid. So, why use it? It allows pixel-based operations to take place without the bottle-neck problems associated with the top levels of the pyramid configuration. The hardware of the prism may be simplified because each processing plane and the interconnections within it are identical.

### Homosyndetic Architectures

Extending the concept of interchangeability to the connection space between planes defines a new class of multidimensional architectures. This class will be referred to as *homosyndetic*[4]: having identical interconnection stages. (This characteristic has, generally, been described in literature as *uniform* connectivity.) The main advantage of such

[4] *homo* (= same) + *syndetic* (= connective)

an architecture is the great flexibility in trading time for space, e.g., only one computation plane and one communication "plane" need be used if time is not a critical factor. Otherwise, pairs of identical processing and communication stages may be pipelined in order to effect a speed-up.

The conceptual 2-D versions of such interconnection schemes have been studied by many researchers [20]. The *omega* ($\Omega$) network is one such implementation which uses identical *shuffle-exchange* stages [38, 56] as shown in Fig. 1.7.

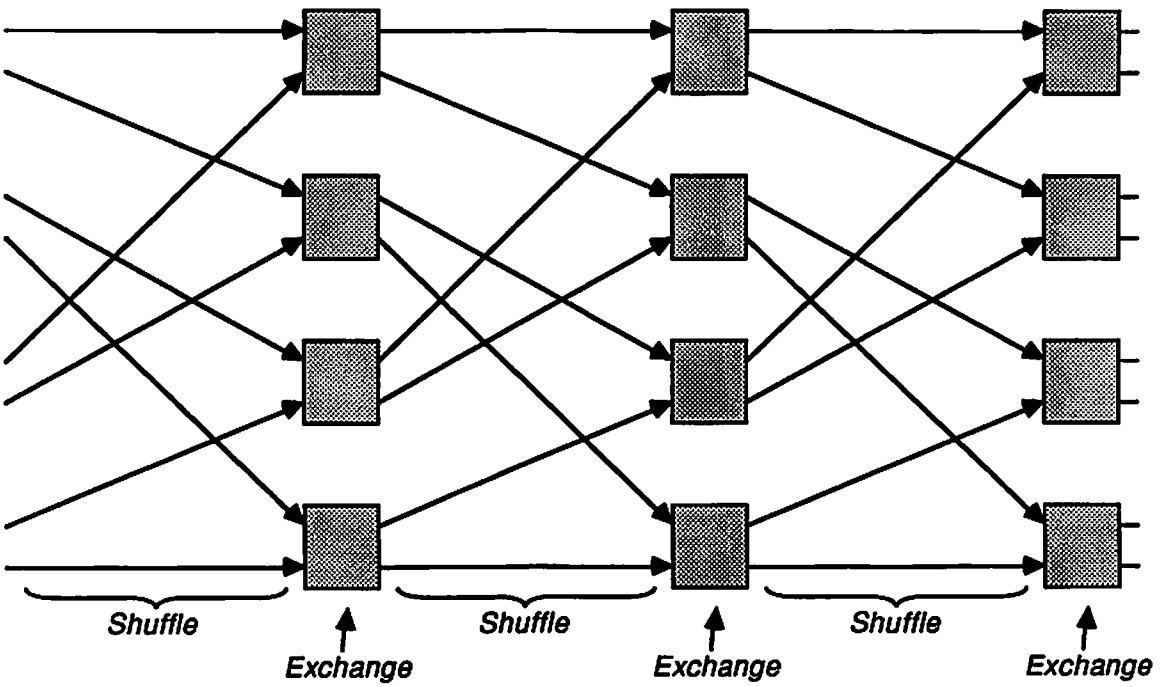



Figure 1.7: Omega Network

(Again, one must be careful about dimensionality descriptions. The pictured omega network is a 2-D conceptual structure that operates on a vector (1-D) of data. In other words, the processing flow is $N$-parallel. Using the processing flow dimensionality as the prime descriptor, the omega network is implied to be one-dimensional. A 2-D Omega structure (3-D conceptual structure) is presented in Chapter 3. )

### 1.3.4 Processing Flow Issues

The issues or primary importance in parallel processing are: *interprocessor communication complexity* and *processor complexity.* These subjects are discussed briefly in the following two sections and are the main considerations for defining candidate architectures.

#### Interconnection Complexity

Parallel processing implies the use of distinct processing units which share data and/or instructions to some extent. The key to speed and efficiency is to get the pertinent data to arrive at the right processor at the right time. The manner in which this communication is implemented will significantly affect the performance of the system. The degree of locality is a major issue especially when attempting to optimize for a broad spectrum of tasks as in image processing. Even restricting the machine to a certain level of operations, e.g., signal processing, we find a mixture of local (e.g., convolution) and global (e.g., histogramming) operations. Other associated interconnection considerations are:

- static (fixed) vs. dynamic (reconfigurable)
- in dynamic networks: blocking vs. non-blocking
  i.e., does ordering a new connection preclude the use of an existing connection
- expansion independent vs. expansion dependent
  e.g., each processor is connected only to a fixed number of other processors regardless of the total number in the system *vs.* local connections grow as the number of processors in the system grows
- single stage vs. multistage networks, i.e., must the data be routed through more than one stage which would mean additional communication delays

Certain interconnection structures (shuffle-exchange, banyan, etc) are optimal for correspondingly different communication requirements. The feasibility of these interconnections depends heavily on the implementation technology. In VLSI/VHSIC it is generally not possible to cross conductors so the interconnections are restricted in

number, distance and direction. Digital optics on the other hand promises *free interconnections* – from anywhere to anywhere.

### Processor Complexity

Realities of hardware implementation capabilities also restrict the complexity of each processing element in the processing plane. Relatively massive parallel designs invariably limit processing elements to binary operations and a few bits of memory. Smaller scale designs allow more sophistication either in the form of off-the-shelf multibit processors or customized designs. The latter introduces yet further complexities as a decision must be made about the assignment of functionality to each architectural level. This includes decisions about the hardwired logic, microcode, register sets, co-processors, available memory etc.

Also the overall control of the processing structure must be taken into account, i.e., synchronous *vs.* asynchronous, *single-instruction multiple-data (SIMD), multiple-instruction multiple-data (MIMD),* etc.

## 1.3.5 Design Approach

Being able to formally map a given algorithm to a multiprocessor architecture has received a great deal of attention in recent years. Almost exclusively, such mappings apply to algorithms and architectures characterized by a fair amount of regularity and iterativeness as exhibited, for example, in systolic arrays. The role of these methods within a generalized design framework is discussed in chapter 4. There are a number of promising analytical methods in this area.

*Space-time Representations:* These approaches abstract a computation in terms of its computational sequencing. A mathematical projection is then made onto a number of feasible processor space-time domains. The mapping is done under specified conditions of timing and allowable interconnection geometries. If the problem size is too large for the processor space, then it can be partitioned into multiple passes in time (space vs. time trade-off).

- *Transformation of Index Sets*, is one such approach which works well with simple computations embedded within multiple loops [33, 32]. Each loop is characterized by a symbolic index. The numerical range of the index is predetermined. The execution step within multiple loops is characterized by the current value of each index – collectively expressed as the *index vector*. All computations associated with a certain index vector comprise a *global computation module*. The transformation maps a global module to a corresponding hardware processing element (PE). Thus, the total number of PEs assumed is equal to the total number of loop cycles. Fortunately, the transformation also indicates an appropriate partitioning of the algorithm so that it can be executed on a PE array of arbitrary size and shape.

  By this technique, one can derive a number of different mappings and interconnection geometries for a given algorithm and a given PE plane. The final choice is a subjective matter and is not the absolute "optimum" solution. Thus, its usefulness lies in verifying that a particular multiprocessor architecture can support a given algorithm.

- A slightly different class of algorithms is that presented by Quinton [43]. It is a loosely defined method for the design of systolic arrays which implement a specific class of algorithms defined as *uniform recurrent equations*. It is a two-step method: a) find a timing function that is compatible with the data dependencies, b) map from the dependence graph to the processor space. UREs are a set of recursive equations; the first one is stated as a function of all the other variables. Each variable is recursively equated to a past value (indexed by a smaller vector). The particular mapping depends on: a) chosen form of the URE set, b)specific timing parameters and c) other variables.

*Signal-Flow Graphs:* A graphical analytic tool for constructing systolic as well as wavefront arrays [24]. It is a modification of the signal flow representation found in signal processing. It models data dependencies of a recursive algorithm. Linear algorithms are best suited for this method which includes formal design verification by means of *z-transforms*. The notation is unwieldy for complex computations.

*Recursive Iterative Algorithms:* This is one of the most recent attempts to classify algorithms which lend themselves to implementation on processor arrays. Rao [45] presents formal mathematical tools which allow analysis and synthesis of a broad class of functions including systolic-type algorithms.

### Summary

Unfortunately, there does not seem to be one methodology that can serve as a universal analysis and synthesis tool. The best one could hope for is a handful of tools that span as much of the parallel processing design set as possible.

## 1.4 Research Contributions

This thesis presents a mathematical framework believed to be an important component of a parallel processing design toolbox. It serves the class of tasks that can be cast in the context of matrix algebra. It is especially useful for those tasks which exhibit a sparcity and regularity in their computations. Such tasks can be implemented on computing structures which feature a large degree of modularity and flexibility in terms of time-space trade-offs.

The framework is an extension and unification of ideas and concepts scattered throughout the literature on large scale computing, parallel processing and signal processing. In the process of conceiving and understanding the role of this framework, a number of contributions were made, including:

- new terms and definitions in the field of multidimensional architectures that allow greater design precision;
- consolidation of a mathematical outer-product algebra based on Kronecker products, shuffles and *array algebra* concepts;
- derivation of useful identities;
- expanded interpretation of array algebra;

- classification of isoplanar homosyndetic computing structures and derivation of their mathematical canonical forms;
- demonstration of the applicability of the methodology to not only task-to-flow mapping, but also design of optical interconnections using classical components (simple lenses).

## 1.5 Thesis Organization

The mathematical foundation is presented in Chapter 2. The Kronecker product of matrices and associated identities are stated. A matrix representation of shuffles and the perfect shuffle are defined in both one- and two-dimensional domains. Matrix restructuring is addressed in terms of Kronecker products and shuffles. A new outer product similar to the Kronecker product, called the star product is defined. The basic concept of array algebra is introduced and shown to ease some of the mathematical difficulties of manipulating Kronecker products and shuffles.

In Chapter 3, the class of isoplanar and homosyndetic processors is examined. Kronecker algebra canonical forms of single stage and multistage structures are presented including those of 1-D and 2-D Omega processors. Analogous canonical forms in array algebra are examined.

The process of mapping a task to an isoplanar homosyndetic structure is presented in Chapter 4. Various decomposition theorems are examined. Mapping methods for representative applications are illustrated.

Chapter 5 discusses the potential implementation technologies and demonstrates the applicability of Kronecker algebra in the implementation of shuffles using properly spaced thin lenses.

In Chapter 6, general conclusions are drawn along with suggestions for further exploration of this work.

# Chapter 2

# Mathematical Framework

In order to fully exploit the parallelism inherent in large scale scientific tasks —including image processing—it is necessary to develop tools which will unable us to detect any concurrency and exercise it to the greatest possible extent.

The choice of tools is dependent on the choice of data structure by which to represent the input and output elements of an operation. The most natural representation of an image is a two-dimensional data structure—a matrix. It is not uncommon, though, to treat an image as a one-dimensional structure—a vector—purely for mathematical convenience. The conversion of the two-dimensional representation to a one-dimensional representation is called *lexicographical ordering* or *vectorization.* In this chapter, an additional data structure is suggested that may be used to represent any multielement quantity including images. It is a dimensionless structure, called an *array.*

In general, data elements and their associated operations may be presented in any of the following domains:

- *Vector Domain*
- *Matrix Domain*
- *Array Domain*

The tools presented in the following sections are based on the premise that the tasks can be cast in a concise mathematical form, namely, that of matrix algebra. This

*declarative* method of programming can be applied to a substantially large class of tasks. By means of various operations and transformations, the original expression is converted into a domain that exposes the inherent parallelism and allows the designer to map to a number of possible parallel architectures.

This new mathematical framework is based on the extensive use of two operations: (a) an outer product, called the *Kronecker Product*, and (b) a class of permutations, called *shuffles*.

These operations are incorporated in two basic algebras:

- *Kronecker Algebra:* an extension of ordinary matrix (linear) algebra that makes extensive use of Kronecker products and matrix representations of shuffles. Data elements are treated as vector (1-D) or matrix (2-D) structures.
- *Array Algebra:* a distant derivative of tensor algebra for treating data elements as arrays – data structures of arbitrary dimension.

In order to ease the algebraic manipulations involved, a number of identities will be introduced.

## 2.1 Kronecker Algebra

Kronecker algebra is simply matrix algebra augmented by a notation of Kronecker operations as well as matrix representations of shuffles. This provides a rich set of identities which allow the expression of sparcity and regularity of data as a function of matrix operations. These Kronecker algebra expressions are well suited for representing parallel processing flows.

The matrix notation used in this study is summarized in the following section.

### 2.1.1 Matrix Notation

The following matrix notation will be used:

- *Vectors* are assumed to be *column vectors* and are denoted by boldface lower-case letters, e.g., $\boldsymbol{v}$.

- The size of a vector may be explicitly denoted by an underscript: $\underset{n}{v}$.

- *Matrices* are denoted by a boldface upper-case letter, e.g., $\boldsymbol{A}$.

- The size of a matrix may be explicitly denoted by an underscript expression, $r \times c$, ($r$ rows by $c$ columns); e.g., $\underset{n\times m}{\boldsymbol{A}}$. Square matrices may be denoted by a single underscript variable, e.g., $\underset{n}{\boldsymbol{I}} = \underset{n\times n}{\boldsymbol{I}}$.

- A specific element or submatrix of a matrix may be labelled by a subscripted pair of indices, e.g., $a_{00}$, $\boldsymbol{A}_{q(r-1)}$. For indexed submatrices, the size may be explicitly shown as an underscript, e.g., $\underset{n\times m}{\boldsymbol{A}_{ij}}$.

- A matrix may be alternatively denoted by bracketing an indexed element or submatrix, e.g., $[a_{ij}]$, $[\boldsymbol{A}_{q(r-1)}]$.

- Indices are assumed to start count at *zero* and increment by one for a total count denoted by $\langle\cdot\rangle$, i.e., $0 \leq i \leq \langle i \rangle - 1$ unless stated otherwise.

- The $k^{th}$ *column vector* of a matrix $\boldsymbol{A}$ will be denoted as: $\boldsymbol{A}_{*k}$
  Similarly, the $k^{th}$ *row vector* of $\boldsymbol{A}$ will be denoted as: $\boldsymbol{A}_{k*}$.

- A matrix, $\boldsymbol{A}$, is *lexicographically vectorized* by the function $\mathbf{vec}(\boldsymbol{A})$. *Column-scanning* is denoted by $\mathbf{vec_c}(\boldsymbol{A})$ and *row-scanning* by $\mathbf{vec_r}(\boldsymbol{A})$:

$$\mathbf{vec_c}\left(\underset{p\times q}{\boldsymbol{A}}\right) \triangleq \begin{bmatrix} \boldsymbol{A}_{*0} \\ \hline \boldsymbol{A}_{*1} \\ \hline \boldsymbol{A}_{*2} \\ \hline \boldsymbol{A}_{*(q-1)} \end{bmatrix}_{pq\times 1} \quad \textit{and} \quad \mathbf{vec_r}\left(\underset{p\times q}{\boldsymbol{A}}\right) \triangleq \begin{bmatrix} \boldsymbol{A}_{0*}^T \\ \hline \boldsymbol{A}_{1*}^T \\ \hline \boldsymbol{A}_{2*}^T \\ \hline \boldsymbol{A}_{(p-1)*}^T \end{bmatrix}_{pq\times 1}$$

Note that the vectorization functions, $\mathbf{vec}(\cdot)$, $\mathbf{vec_r}(\cdot)$ and $\mathbf{vec_c}(\cdot)$ produce column vectors.

- *Unit vectors* are $q$-element vectors with all zero entries except for the $k^{th}$ element which is equal to *one*. They are denoted by:

$$\underset{q}{u}^{(k)} \triangleq [\underbrace{0 \cdots 0 \overbrace{1}^{k^{th}position} 0 \cdots 0}_{q\ elements}]^T.$$

- An *elementary matrix*, $\underset{p\times q}{U}^{(i,k)}$, of size $p \times q$ has all zero entries except for the $(i,k)^{th}$ element which has the value of *one*. Such a matrix is generated by an outer product of unit vectors:

$$\underset{p\times q}{U}^{(i,k)} \triangleq \underset{p}{u}^{(i)} \cdot \underset{q}{u}^{(k)T}. \tag{2-1}$$

*Example 1:*

$$\underset{2\times 3}{U}^{(0,1)} \triangleq \underset{2}{u}^{(0)} \cdot \underset{3}{u}^{(1)T}$$

$$= \begin{bmatrix} 1 \\ 0 \end{bmatrix} [010] = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}.$$

- The *reflection matrix* (also called *counter-identity matrix*), $J$, is a square matrix with ones along the counter (NE-SW) diagonal and zeroes elsewhere. E.g.,

$$\underset{8}{J} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

Its effect is to reverse the order of the rows (or columns) of the matrix with which it is pre-multiplied (or post-multiplied). Note that $\underset{n}{J} = \underset{n}{J}^{-1} = \underset{n}{J}^{T}$.

### 2.1.2 Kronecker Products

The foundation of Kronecker algebra is a matrix outer product called the Kronecker product [5, 17].

DEFINITION 1: *The (right)* Kronecker product *(or direct product) of* $\underset{p\times q}{A}$ *with* $\underset{m\times n}{B}$ *is defined as:*

$$\underset{p\times q}{A} \otimes \underset{m\times n}{B} \triangleq \begin{bmatrix} a_{00}B & a_{01}B & \cdots & a_{0(q-1)}B \\ a_{10}B & a_{11}B & & a_{1(q-1)}B \\ \vdots & & \ddots & \vdots \\ a_{(p-1)0}B & a_{(p-1)1}B & \cdots & a_{(p-1)(q-1)}B \end{bmatrix}_{pm\times qn}.$$

Occasionally, in literature, a *left* Kronecker product is used, where:

$$A \overset{\text{left}}{\otimes} B = B \overset{\text{right}}{\otimes} A.$$

The $k^{th}$ Kronecker power of a matrix $A$ will be denoted as:

$$A^{\otimes k} \triangleq \underbrace{A \otimes A \otimes \cdots \otimes A}_{k \text{ factors}} = A \otimes A^{\otimes k-1} = A^{\otimes k-1} \otimes A.$$

By definition, $A^{\otimes 0} \triangleq 1$ and $A^{\otimes 1} \triangleq A$.

DEFINITION 2: *The* Kronecker sum *of* $\underset{k\times k}{A}$ *and* $\underset{m\times m}{B}$ *is defined as:*

$$\underset{k\times k}{A} \oplus \underset{m\times m}{B} \triangleq \underset{k\times k}{A} \otimes \underset{m}{I} + \underset{k}{I} \otimes \underset{m\times m}{B}.$$

#### Properties

- Note that, in general,

$$\underset{p\times q}{A} \otimes \underset{m\times n}{B} \neq \underset{m\times n}{B} \otimes \underset{p\times q}{A}.$$

The exact relationship of these two products is explained by means of shuffle matrices as will be shown in Section 2.1.3.

| | | |
|---|---|---|
| *(In general)* | $\underset{p\times q}{A} \otimes \underset{m\times n}{B} \neq \underset{m\times n}{B} \otimes \underset{p\times q}{A}$ | (2–2) |
| *Associativity:* | $\underset{p\times q}{A} \otimes (\underset{m\times n}{B} \otimes \underset{k\times l}{C}) = (\underset{p\times q}{A} \otimes \underset{m\times n}{B}) \otimes \underset{k\times l}{C}$ | (2–3) |
| *Distributivity:* | $(\underset{p\times q}{A} + \underset{p\times q}{H}) \otimes (\underset{m\times n}{B} + \underset{m\times n}{R}) =$ $\underset{p\times q}{A} \otimes \underset{m\times n}{B} + \underset{p\times q}{A} \otimes \underset{m\times n}{R} + \underset{p\times q}{H} \otimes \underset{m\times n}{B} + \underset{p\times q}{H} \otimes \underset{m\times n}{R}$ | (2–4) |
| *Mixed Products:* | $(\underset{p\times q}{A} \otimes \underset{m\times n}{B})(\underset{q\times r}{D} \otimes \underset{n\times k}{G}) = \underset{p\times q}{A}\,\underset{q\times r}{D} \otimes \underset{m\times n}{B}\,\underset{n\times k}{G}$ | (2–5) |
| *Transposition:* | $(\underset{p\times q}{A} \otimes \underset{m\times n}{B})^T = \underset{q\times p}{A}^T \otimes \underset{n\times m}{B}^T$ | (2–6) |
| *Inversion:* | $(\underset{n}{N} \otimes \underset{m}{M})^{-1} = \underset{n}{N}^{-1} \otimes \underset{m}{M}^{-1}$ | (2–7) |
| *Traces:* | $trace(\underset{n}{N} \otimes \underset{m}{M}) = trace(\underset{n}{N}) \cdot trace(\underset{m}{M})$ | (2–8) |
| *Determinants:* | $\det(\underset{n}{N} \otimes \underset{m}{M}) = (\det \underset{n}{N})^n \cdot (\det \underset{m}{M})^m$ | (2–9) |
| *Exponentiation:* | $\exp(\underset{n}{N} \oplus \underset{m}{M}) = \exp(\underset{n}{N}) \otimes \exp(\underset{m}{M})$ | (2–10) |
| | $\exp(\underset{n}{I} \otimes \underset{m}{M}) = \underset{n}{I} \otimes \exp(\underset{m}{M})$ | (2–11) |
| | $\exp(\underset{m}{M} \otimes \underset{n}{I}) = \exp(\underset{m}{M}) \otimes \underset{n}{I}$ | (2–12) |

Table 2.1: Basic Kronecker Algebra Properties

Kronecker algebra properties are summarized in Table 2.1. Note that they are valid only for dimensionally compatible, i.e., *conformable* matrices. Distributivity (2–4), may alternatively be expressed as,

$$\left(\sum_i \underset{p\times q}{\boldsymbol{A}_i}\right) \otimes \underset{p\times q}{\boldsymbol{H}} = \sum_i \left(\underset{p\times q}{\boldsymbol{A}_i} \otimes \underset{p\times q}{\boldsymbol{H}}\right)$$

and more generally as,

$$\left(\sum_i \underset{p\times q}{\boldsymbol{A}_i}\right) \otimes \left(\sum_i \underset{p\times q}{\boldsymbol{H}_j}\right) = \sum_i \sum_j \left(\underset{p\times q}{\boldsymbol{A}_i} \otimes \underset{p\times q}{\boldsymbol{H}_j}\right) \tag{2–13}$$

The mixed product rule (2–5) is a particularly powerful property. It is used to derive many other useful identities, some of which are listed in Table 2.2. Associativity extends the mixed product rule into the general form

$$\prod_{i=0}^{n-1} \left(\bigotimes_{j=0}^{m-1} \underset{p_i\times q_i}{\boldsymbol{A}_{ij}}\right) = \bigotimes_{j=0}^{m-1} \left(\prod_{i=0}^{n-1} \underset{p_i\times q_i}{\boldsymbol{A}_{ij}}\right) \tag{2–14}$$

where $q_k = p_{k+1} \;\; \forall \;\; 0 \leq k \leq n-2$ to retain conformity.

$$(\underset{p\times q}{\boldsymbol{A}} \otimes \underset{n}{\boldsymbol{I}})(\underset{q}{\boldsymbol{I}} \otimes \underset{n\times k}{\boldsymbol{G}}) = \underset{p\times q}{\boldsymbol{A}} \otimes \underset{n\times k}{\boldsymbol{G}} \tag{2–15}$$

$$(\underset{q}{\boldsymbol{I}} \otimes \underset{m\times n}{\boldsymbol{B}})(\underset{q\times r}{\boldsymbol{D}} \otimes \underset{n}{\boldsymbol{I}}) = \underset{q\times r}{\boldsymbol{D}} \otimes \underset{m\times n}{\boldsymbol{B}} \tag{2–16}$$

$$(\underset{m}{\boldsymbol{M}} \otimes \underset{n}{\boldsymbol{I}})(\underset{m}{\boldsymbol{I}} \otimes \underset{n}{\boldsymbol{N}}) = \underset{m}{\boldsymbol{M}} \otimes \underset{n}{\boldsymbol{N}} = (\underset{m}{\boldsymbol{I}} \otimes \underset{n}{\boldsymbol{N}})(\underset{m}{\boldsymbol{M}} \otimes \underset{n}{\boldsymbol{I}}) \tag{2–17}$$

$$\underset{n}{\boldsymbol{v}} \otimes (\underset{p\times q}{\boldsymbol{A}} \cdot \underset{q\times r}{\boldsymbol{D}}) = (\underset{n}{\boldsymbol{v}} \otimes \underset{p\times q}{\boldsymbol{A}}) \cdot \underset{q\times r}{\boldsymbol{D}} \tag{2–18}$$

$$\underset{n}{\boldsymbol{u}^{(i)}} \otimes \underset{m}{\boldsymbol{u}^{(j)^T}} = \underset{n}{\boldsymbol{u}^{(i)}} \cdot \underset{m}{\boldsymbol{u}^{(j)^T}} \tag{2–19}$$

Table 2.2: Useful Kronecker Algebra Identities

### Block Kronecker Products

In addition to the traditional Kronecker product, other manifestations of a Kronecker-like outer-product have been proposed, such as that by [15]:

DEFINITION 3 *(Block Kronecker Product): Let matrices* $\underset{pm\times qn}{\boldsymbol{A}}$ *and* $\underset{rn\times tk}{\boldsymbol{B}}$ *be partitioned into submatrices,* $\boldsymbol{A}_{ij}$ *and* $\boldsymbol{B}_{ij}$ *of size* $m\times n$ *and* $n\times k$*, respectively. The* block Kronecker product *of* $\boldsymbol{A}$ *with* $\boldsymbol{B}$ *is defined as,*

$$\underset{pm\times qn}{\boldsymbol{A}} \boxtimes \underset{rn\times tk}{\boldsymbol{B}} = \begin{bmatrix} \boldsymbol{A}_{00}\boldsymbol{B} & \boldsymbol{A}_{01}\boldsymbol{B} & \cdots & \boldsymbol{A}_{0(q-1)}\boldsymbol{B} \\ \boldsymbol{A}_{10}\boldsymbol{B} & \boldsymbol{A}_{11}\boldsymbol{B} & & \boldsymbol{A}_{1(q-1)}\boldsymbol{B} \\ \vdots & & \ddots & \vdots \\ \boldsymbol{A}_{(p-1)0}\boldsymbol{B} & \boldsymbol{A}_{(p-1)1}\boldsymbol{B} & \cdots & \boldsymbol{A}_{(p-1)(q-1)}\boldsymbol{B} \end{bmatrix}_{prm\times qtk}$$

*where*

$$\boldsymbol{A}_{ij}\boldsymbol{B} = \begin{bmatrix} \boldsymbol{A}_{ij}\boldsymbol{B}_{00} & \boldsymbol{A}_{ij}\boldsymbol{B}_{01} & \cdots & \boldsymbol{A}_{ij}\boldsymbol{B}_{0(t-1)} \\ \boldsymbol{A}_{ij}\boldsymbol{B}_{10} & \boldsymbol{A}_{ij}\boldsymbol{B}_{11} & & \boldsymbol{A}_{ij}\boldsymbol{B}_{1(t-1)} \\ \vdots & & \ddots & \vdots \\ \boldsymbol{A}_{ij}\boldsymbol{B}_{(t-1)0} & \boldsymbol{A}_{ij}\boldsymbol{B}_{(r-1)1} & \cdots & \boldsymbol{A}_{ij}\boldsymbol{B}_{(r-1)(t-1)} \end{bmatrix}_{rm\times tk}$$

### 2.1.3 Shuffle Permutations

DEFINITION 4: *A* permutation matrix, $\boldsymbol{P}$*, is a square matrix representing a given permutation,* $\pi$*.*

$$\underset{q\times q}{\boldsymbol{P}}\Big|_{\pi} \triangleq \left[ \begin{array}{c|c|c|c} \underset{q}{\boldsymbol{u}}^{(\pi(0))} & \underset{q}{\boldsymbol{u}}^{(\pi(1))} & \cdots & \underset{q}{\boldsymbol{u}}^{(\pi(q-1))} \end{array} \right]$$

*where* $\underset{q}{\boldsymbol{u}}^{(k)}$ *is a* $q$*-element unit vector (*$k^{th}$ *element is one; others zero).*

When used as a pre-multiplier of a matrix, $\boldsymbol{A}$, it rearranges the order of the rows of $\boldsymbol{A}$ according to $\pi$. When used as a post-multiplier, the effect is to permute the ordering of the columns of $\boldsymbol{A}$ according to $\pi^{-1}$.

*Example 2:* For $\pi(i) = (i-1) \bmod 4$, the representative (column) permutation matrix is,

$$\boldsymbol{P} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

Reordering the rows of an arbitrary matrix, $\underset{4\times4}{A}$, according to $\pi(i)$ is expressed as

$$PA = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} A_{0*} \\ \hline A_{1*} \\ \hline A_{2*} \\ \hline A_{3*} \end{bmatrix} = \begin{bmatrix} A_{1*} \\ \hline A_{2*} \\ \hline A_{3*} \\ \hline A_{0*} \end{bmatrix}$$

and permutation of the columns of $A$ by $\pi^{-1}$ is expressed as

$$\begin{aligned} AP &= \left[\, A_{*0} \,\middle|\, A_{*1} \,\middle|\, A_{*2} \,\middle|\, A_{*3} \,\right] \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix} \\ &= \left[\, A_{*3} \,\middle|\, A_{*0} \,\middle|\, A_{*1} \,\middle|\, A_{*2} \,\right]. \end{aligned}$$

Note that permutation matrices of one-to-one permutations are orthogonal ($P^T = P^{-1}$) and form a group with respect to matrix multiplication. Therefore permuting the columns of a matrix by $\pi(\cdot)$ is expressed as postmultiplication by $P^{-1}$,

$$\begin{aligned} AP^T &= \left[\, A_{*0} \,\middle|\, A_{*1} \,\middle|\, A_{*2} \,\middle|\, A_{*3} \,\right] \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \\ &= \left[\, A_{*1} \,\middle|\, A_{*2} \,\middle|\, A_{*3} \,\middle|\, A_{*0} \,\right]. \end{aligned}$$

*Shuffles* are an important class of permutations which have been used in various applications of data routing. The most well known are shuffles of one-dimensional quantities.

## 1-D Shuffles

Shuffling an $N$-element vector, $\boldsymbol{x}$, results in an $N$-element vector, $\boldsymbol{y}$, containing the same elements as $\boldsymbol{x}$ but in locations determined by a shuffling permutation, $\sigma(\cdot)$, i.e.,

$$y_{\sigma(i)} = x_i \qquad \forall \ 0 \leq i \leq N-1 \tag{2-20}$$

An intuitive interpretation of a shuffle of $N$ elements is as follows:

1. Factor $N$ into a product of two integers, $p \cdot q$, such that:
   - the *N-element* input vector is partitioned into $p$ consecutive groups of size $q$
   - the *N-element* output vector is partitioned into $q$ consecutive groups of size $p$.
2. Sequentially fill each of the $q$ output groups with the respectively ordered element from each of the $p$ input groups, e.g., the first element of each input group comprises the first output group.

A shuffle may be classified according to the input vector partitioning factor alone, e.g., *p-shuffle*. In order, however, to determine the exact permuted location of an element, a shuffle requires the specification of two of the three parameters $(N, p, q)$. The *p|q-shuffle* of $N = p \cdot q$ elements, is formally characterized by the permutation:

$$\boxed{\sigma_{p|q}(i) = \left(p \cdot i + \left\lfloor \frac{i}{q} \right\rfloor\right) \bmod N \qquad \text{for} \quad 0 \leq i \leq N-1}$$

where $i$ is the input position of an element and $\sigma(i)$ is the corresponding output position[1] e.g.,

$$y_{\sigma_{p|q}(i)} = x_i \qquad \text{for} \quad 0 \leq i \leq N-1$$

[1]Permutations of sequences where position $i$ contains the number $i$ may lead to misinterpreting the permutation of the position as the permutation or transformation of the contained number. Shuffling such a sequence by $\sigma_{p|q}(i)$ results in the number $i$ being transferred to position $\sigma_{p|q}(i)$. This shuffling may, incorrectly, be interpreted as transforming the input number $i$ (at location $i$) by $\sigma_{p|q}(i)$. In this case, coincidently, such an interpretation is correct if the transformation $\sigma_{q|p}(i)$ is used, i.e.,

$$y_{\sigma_{p|q}(i)} = x_i \iff y_i = \sigma_{q|p}(x_i) \qquad \forall \ x_i = i, \quad 0 \leq i \leq pq-1$$

(Note the duality relationship.)

*Shuffling matrices* are permutation matrices representing the permutation $\sigma_{p|q}(\cdot)$,

$$S_{[p|q]} \triangleq \left[ \underset{pq}{u}^{(\sigma_{p|q}(0))} \;\vdots\; \underset{pq}{u}^{(\sigma_{p|q}(1))} \;\vdots\; \cdots \;\vdots\; \underset{pq}{u}^{(\sigma_{p|q}(pq-1))} \right]$$

In other words, the elements $s_{ij}$ of $S_{[p|q]}$ are by definition,

$$s_{ij} = \begin{cases} 1 & \text{if } i = \left(pj + \left\lfloor \frac{j}{q} \right\rfloor\right) \bmod pq \\ 0 & \text{otherwise} \end{cases}$$

Note that the size of $S_{[p|q]}$ is $pq \times pq$.

*Example 3:* The shuffling of a 16-element vector may be done in any one of three ways depending on the underlying permutations (see Fig. 2.1):

$$\sigma_{2|8} \quad or \quad \sigma_{8|2} \quad or \quad \sigma_{4|4}$$

(excluding the trivial identities $\sigma_{1|16}$ and $\sigma_{16|1}$)

The matrix representing a given shuffle may be generated in several ways. A very useful decomposition is the following [7]:

DEFINITION 5 *(Sum of Kronecker Products Decomposition)*:

$$S^{-1}_{[p|q]} \triangleq \sum_{i=0}^{p-1} \sum_{k=0}^{q-1} \underset{p\times q}{U}^{(i,k)} \otimes \left(\underset{p\times q}{U}^{(i,k)}\right)^T \tag{2-21}$$

*and, conversely,*

$$S_{[p|q]} \triangleq \sum_{i=0}^{p-1} \sum_{k=0}^{q-1} \left(\underset{p\times q}{U}^{(i,k)}\right)^T \otimes \underset{p\times q}{U}^{(i,k)} \tag{2-22}$$

*Example 4:* The smallest non-trivial shuffle is the $S^{-1}_{[2|2]}$ which is basically a *butterfly* interconnection.

$$S^{-1}_{[2|2]} \triangleq \sum_{i=0}^{1} \sum_{k=0}^{1} \underset{2\times 2}{U}^{(i,k)} \otimes \left(\underset{2\times 2}{U}^{(i,k)}\right)^T$$

$$= (U^{(00)} \otimes U^{(00)}) + (U^{(01)} \otimes U^{(10)}) + (U^{(10)} \otimes U^{(01)}) + (U^{(11)} \otimes U^{(11)})$$

$$= \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \otimes \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix} \otimes \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \otimes \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}$$

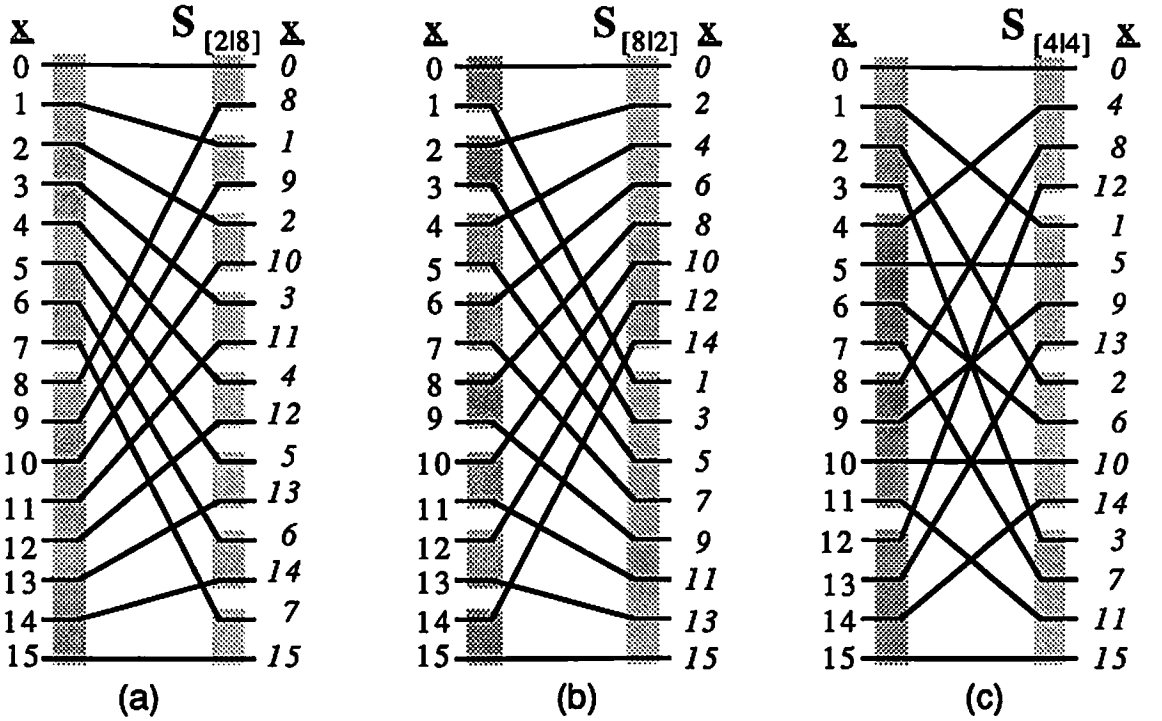



Figure 2.1: Shuffles of 16 elements: (a) $\sigma_{2|8}$, (b) $\sigma_{8|2}$, and (c) $\sigma_{4|4}$

$$= \begin{bmatrix} 1 & 0 & . & . \\ 0 & 0 & . & . \\ . & . & . & . \\ . & . & . & . \end{bmatrix} + \begin{bmatrix} . & . & 0 & 0 \\ . & . & 1 & 0 \\ . & . & . & . \\ . & . & . & . \end{bmatrix} + \begin{bmatrix} . & . & . & . \\ . & . & . & . \\ 0 & 1 & . & . \\ 0 & 0 & . & . \end{bmatrix} + \begin{bmatrix} . & . & . & . \\ . & . & . & . \\ . & . & 0 & 0 \\ . & . & 0 & 1 \end{bmatrix}$$

$$\Longrightarrow S^{-1}_{[2|2]} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where the "." elements represent zero values.

## General Properties/Identities

From the intuitive description of a shuffle, it is obvious that,

COROLLARY 1 *(Shuffle Identity)*: $S_{[1|N]} = S_{[N|1]} = \underset{N}{I}$,

*Proof:* Using (2–21):

$$\begin{aligned} S_{[1|N]} &\triangleq \sum_{k=0}^{N-1} (\underset{1\times N}{U}^{(0,k)})^T \otimes \underset{1\times N}{U}^{(0,k)} \\ &= \sum_{k=0}^{N-1} \underset{N}{u}^{(k)} \otimes \underset{N}{u}^{(k)^T} \\ [\text{by (2–19)}] \quad &= \sum_{k=0}^{N-1} \underset{N}{u}^{(k)} \cdot \underset{N}{u}^{(k)^T} \\ &= \sum_{k=0}^{N-1} \underset{N\times N}{U}^{(k,k)} \\ &= \underset{N}{I}. \end{aligned}$$

And similarly for $S_{[N|1]}$. □

COROLLARY 2: *The inverse of a p|q-shuffle is the q|p-shuffle:*

$$S_{[p|q]}^{-1} = S_{[q|p]}.$$

*Proof:* This is an immediate consequence of the definition of a shuffle matrix, i.e.,

$$\begin{aligned} S_{[q|p]} &\triangleq \sum_{i=0}^{q-1} \sum_{k=0}^{p-1} (\underset{q\times p}{U}^{(i,k)})^T \otimes \underset{q\times p}{U}^{(i,k)} \\ &= \sum_{k=0}^{p-1} \sum_{i=0}^{q-1} \underset{p\times q}{U}^{(k,i)} \otimes (\underset{p\times q}{U}^{(k,i)})^T \\ &\triangleq S_{[p|q]}^{-1}. \end{aligned}$$

□

In the same manner, it can be shown that

COROLLARY 3: $S_{[p|q]}^T = S_{[q|p]}$.

COROLLARY 4 *(Shuffle Orthogonality )*: *Shuffling matrices are orthogonal.*

$$S^{-1}_{[p|q]} = S^{T}_{[p|q]}.$$

*Proof:* A consequence of the previous two corollaries. □

COROLLARY 5 *(Symmetric Shuffle)*: *For any positive integer, p,* $S_{[p|p]}$ *is symmetric,*

$$S^{T}_{[p|p]} = S_{[p|p]}$$

*and*

$$\Longrightarrow S^{-1}_{[p|p]} = S_{[p|p]}.$$

*Proof:* Easily shown by using the definitions of the shuffle and its inverse and Corollary 4. □

Shuffles play an important role in the world of Kronecker algebra. This is exemplified by the commutativity properties of the Kronecker product:

THEOREM 6 *(Pseudo-Commutativity)*: *For any two matrices* $\underset{p\times q}{A}$ *and* $\underset{r\times t}{B}$*:*

$$\underset{p\times q}{A} \otimes \underset{r\times t}{B} = S^{-1}_{[p|r]}(\underset{r\times t}{B} \otimes \underset{p\times q}{A})S_{[q|t]}. \tag{2-23}$$

*Proof:* Two different approaches are presented in Appendix A. □

A given shuffle may also be expressed as a product of other shuffles. This is shown in the following theorems and corollaries which are based on two shuffle factorizations presented in [9]:

THEOREM 7 *(Shuffle Products Decomposition)*:

$$S_{[r|p\cdot q]} = S_{[r\cdot q|p]}S_{[r\cdot p|q]} = S_{[r\cdot p|q]}S_{[r\cdot q|p]}. \tag{2-24}$$

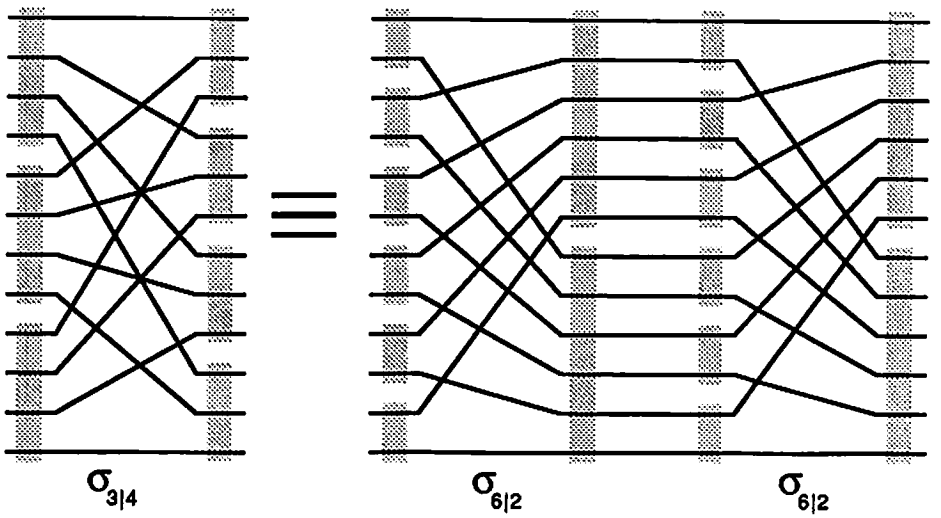



Figure 2.2: Equivalence of $S_{[3|4]} = S_{12}^{-1} S_{12}^{-1}$

*Proof:* Stated as Equation (2) of Theorem 5 in [9] where $r = b_2, p = b_1, q = b_0$. □

*Example 5:* The equivalence of $S_{[3|4]} = S_{[3|2\cdot 2]} \equiv S_{[3\cdot 2|2]} S_{[3\cdot 2|2]}$ is illustrated in Fig. 2.2.

THEOREM 8 *(Products of Kronecker Products Decomposition)*:

$$S_{[r\cdot p|q]} = (S_{[r|q]} \otimes \underset{p}{I})(\underset{r}{I} \otimes S_{[p|q]}) \tag{2–25}$$

*Proof:* Stated as Equation (1) of Theorem 5 in [9] where $r = b_2, p = b_1, q = b_0$. □

COROLLARY 9:

$$S_{[r\cdot p|q]} = S_{[rq|p]}^{-1} (\underset{p}{I} \otimes S_{[r|q]}) S_{[rq|p]} (\underset{r}{I} \otimes S_{[p|q]}) \tag{2–26}$$

*Proof:* Apply the pseudo-commutativity property (Theorem 6) to (2–25). □

As is shown in Chapter 4, the above corollary may be implemented by two shuffle/exchange stages, where the exchanges are actually shuffles of lower order.

**Summary** The basic properties of matrices representing shuffles are summarized in Table 2.3.

| | |
|---|---|
| *Identity:* | $S_{[1\|N]} = S_{[N\|1]} = \underset{N}{I}$ |
| *Orthogonality:* | $S^{-1}_{[p\|q]} = S^{T}_{[p\|q]} = S_{[q\|p]}$ |
| *Symmetry:* | $S^{T}_{[p\|p]} = S_{[p\|p]}$ |
| *Pseudo-Commutativity:* | $\underset{p\times q}{A} \otimes \underset{r\times t}{B} = S^{-1}_{[p\|r]}(\underset{r\times t}{B} \otimes \underset{p\times q}{A})S_{[q\|t]}$ |

Table 2.3: Basic Shuffle Matrix Properties

## Perfect Shuffle

If the elements to be permuted are initially divided into two groups ($p = 2$), then the shuffle permutation $\sigma_{2|\frac{N}{2}}$ takes on a special name: the *perfect shuffle.*

Since $N$ is the only variable, the perfect $N$-shuffle permutation and its representative matrix will be denoted, respectively, as

$$\sigma_N \quad \text{and} \quad S_N.$$

A perfect shuffle is more commonly known as the permutation that effects a circular left shift of the binary representation of the ordering index, when $N = 2^n$. For example, the perfect shuffle of eight objects is performed by the mapping shown in Table 2.4.

An equivalent formulation of the perfect shuffle is:

$$\sigma_N(i) = \begin{cases} 2i & \text{if} \quad 0 \le i \le N-1 \\ 2(i-N)+1 & \text{if} \quad N \le i \le 2N-1 \end{cases}$$

The inverse perfect shuffle, $\sigma_N^{-1}$, is represented by,

$$S_N^{-1} = S^{-1}_{[2|\frac{N}{2}]} = S_{[\frac{N}{2}|2]}$$

| Initial Position | | Permuted Position |
|---|---|---|
| $0 = 000_2$ | $\longrightarrow$ | $000_2 = 0$ |
| $1 = 001_2$ | $\longrightarrow$ | $010_2 = 2$ |
| $2 = 010_2$ | $\longrightarrow$ | $100_2 = 4$ |
| $3 = 011_2$ | $\longrightarrow$ | $110_2 = 6$ |
| $4 = 100_2$ | $\longrightarrow$ | $001_2 = 1$ |
| $5 = 101_2$ | $\longrightarrow$ | $011_2 = 3$ |
| $6 = 110_2$ | $\longrightarrow$ | $101_2 = 5$ |
| $7 = 111_2$ | $\longrightarrow$ | $111_2 = 7$ |

Table 2.4: Labelling of an 8-element perfect shuffle by circular left-shift

and may be viewed as the operation that separates the odd and even numbered elements of a sequence (see Fig. 2.1(b)). The inverse perfect shuffle is equivalent to a circular right shift of the binary representation of the index, when $N = 2^n$.

*Example 6:*

$$S_4 = S_{[2|2]} = S^{-1}_{[2|2]} = S_4^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

COROLLARY 10 *(Small Shuffle Factorization): For even $N > 2$:*

$$(a) \quad S^{-1}_{2\cdot N} = (S_N^{-1} \otimes \underset{2}{I})(\underset{N/2}{I} \otimes S_4^{-1}) \tag{2-27}$$

$$(b) \quad S_{2\cdot N} = S^{-1}_{2\cdot N}(S_N \otimes \underset{2}{I})(\underset{2}{I} \otimes S_N) \tag{2-28}$$

*Proof:*

(a) Rewrite *Theorem 8* using perfect shuffle notation.

$$S^{-1}_{2\cdot N} = S^{-1}_{[2|2\cdot\frac{N}{2}]} = S_{[2\cdot\frac{N}{2}|2]} = (S_{[N/2|2]} \otimes \underset{2}{I})(\underset{N/2}{I} \otimes S^{-1}_{[2|2]})$$

$$\Longrightarrow \boxed{S^{-1}_{2\cdot N} = (S_N^{-1} \otimes \underset{2}{I})(\underset{N/2}{I} \otimes S_4^{-1})}$$

(b) Rewrite *Theorem 7* using perfect shuffle notation.

$$S_{2\cdot N} = S_{[2|2\cdot\frac{N}{2}]} = S_{[2\cdot\frac{N}{2}|2]} \cdot S_{[2\cdot 2|\frac{N}{2}]} = S_{2\cdot N}^{-1} \cdot S_{[2\cdot 2|N/2]}$$

But $S_{[2\cdot 2|N/2]} = (S_{[2|N/2]} \otimes \underset{2}{I})(\underset{2}{I} \otimes S_{[2|N/2]})$

$$\Longrightarrow \boxed{S_{2\cdot N} = S_{2\cdot N}^{-1} \cdot (S_N \otimes \underset{2}{I})(\underset{2}{I} \otimes S_N)}$$

□

*Example 7:*

$$S_8^{-1} = S_{2\cdot 4}^{-1} = (S_4^{-1} \otimes \underset{2}{I})(\underset{2}{I} \otimes S_4)$$

$$= \left( \begin{bmatrix} 1 & . & . & . \\ . & . & 1 & . \\ . & 1 & . & . \\ . & . & . & 1 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \right) \left( \begin{bmatrix} 1 & . \\ . & 1 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \right)$$

$$= \begin{bmatrix} 1 & 0 & . & . & . & . & . & . \\ 0 & 1 & . & . & . & . & . & . \\ . & . & . & . & 1 & 0 & . & . \\ . & . & . & . & 0 & 1 & . & . \\ . & . & 1 & 0 & . & . & . & . \\ . & . & 0 & 1 & . & . & . & . \\ . & . & . & . & . & . & 1 & 0 \\ . & . & . & . & . & . & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 & . & . & . & . \\ 0 & 0 & 1 & 0 & . & . & . & . \\ 0 & 1 & 0 & 0 & . & . & . & . \\ 0 & 0 & 0 & 1 & . & . & . & . \\ . & . & . & . & 1 & 0 & 0 & 0 \\ . & . & . & . & 0 & 0 & 1 & 0 \\ . & . & . & . & 0 & 1 & 0 & 0 \\ . & . & . & . & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\Longrightarrow S_8^{-1} = \begin{bmatrix} 1 & . & . & . & . & . & . & . \\ . & . & 1 & . & . & . & . & . \\ . & . & . & . & 1 & . & . & . \\ . & . & . & . & . & . & 1 & . \\ . & 1 & . & . & . & . & . & . \\ . & . & . & 1 & . & . & . & . \\ . & . & . & . & . & 1 & . & . \\ . & . & . & . & . & . & . & 1 \end{bmatrix}.$$

THEOREM 11 *(Cyclic Group): The $N = 2^n$ perfect shuffle generates a cyclic group of order n, i.e.,*

$$S_N^n = \underset{N}{I}$$

*and*

$$S_N = \left(S_N^{-1}\right)^{n-1}, \qquad S_N^{-1} = (S_N)^{n-1}.$$

This fact is immediate when considering the binary representation of $N = 2^n$ numbers, i.e., *a circular left-shift* $= (n-1)$ *circular right-shifts.*

*Example 8:* $S_8 = S_8^{-1} \cdot S_8^{-1}$ is supported also by corollary 10:

$$S_8 = S_8^{-1} \cdot (S_4 \otimes \underset{2}{I})(\underset{2}{I} \otimes S_4) = S_8^{-1} \cdot (S_4^{-1} \otimes \underset{2}{I})(\underset{2}{I} \otimes S_4^{-1}) = S_8^{-1} \cdot S_8^{-1}$$

$$\Longrightarrow S_8 = \begin{bmatrix} 1 & . & . & . & . & . & . & . \\ . & . & 1 & . & . & . & . & . \\ . & . & . & . & 1 & . & . & . \\ . & . & . & . & . & . & 1 & . \\ . & 1 & . & . & . & . & . & . \\ . & . & . & 1 & . & . & . & . \\ . & . & . & . & . & 1 & . & . \\ . & . & . & . & . & . & . & 1 \end{bmatrix} \begin{bmatrix} 1 & . & . & . & . & . & . & . \\ . & . & 1 & . & . & . & . & . \\ . & . & . & . & 1 & . & . & . \\ . & . & . & . & . & . & 1 & . \\ . & 1 & . & . & . & . & . & . \\ . & . & . & 1 & . & . & . & . \\ . & . & . & . & . & 1 & . & . \\ . & . & . & . & . & . & . & 1 \end{bmatrix}$$

$$\Longrightarrow S_8 = \begin{bmatrix} 1 & . & . & . & . & . & . & . \\ . & . & . & . & 1 & . & . & . \\ . & 1 & . & . & . & . & . & . \\ . & . & . & . & . & 1 & . & . \\ . & . & 1 & . & . & . & . & . \\ . & . & . & . & . & . & 1 & . \\ . & . & . & 1 & . & . & . & . \\ . & . & . & . & . & . & . & 1 \end{bmatrix}.$$

COROLLARY 12: *Any shuffle of the form , $S_{[g|2^k]}$ , may be decomposed into a power of perfect inverse shuffles,*

$$S_{[g|2^k]} = \left(S_{g \cdot 2^k}^{-1}\right)^k \tag{2-29}$$

*Proof:* By Theorem 7,

$$\begin{aligned}
S_{[g|2^k]} &= S_{[g|2^{k-1}\cdot 2]} \\
&= S_{[g\cdot 2^{k-1}|2]} \cdot S_{[g\cdot 2|2^{k-1}]} \\
&= S_{[g\cdot 2^{k-1}|2]} \cdot S_{[2g|2^{k-2}\cdot 2]} \\
&= S_{[g\cdot 2^{k-1}|2]} \cdot S_{[g\cdot 2^{k-1}|2]} \cdot S_{[4g|2^{k-3}\cdot 2]} \\
&= S_{[g\cdot 2^{k-1}|2]} \cdot S_{[g\cdot 2^{k-1}|2]} \cdot S_{[g\cdot 2^{k-1}|2]} \cdot S_{[8g|2^{k-3}]} \\
&= \vdots \\
&= (S_{[g\cdot 2^{k-1}|2]})^r \cdot S_{[g\cdot 2^r|2^{k-r}]} \\
&= (S^{-1}_{g\cdot 2^k})^r \cdot S_{[g\cdot 2^r|2^{k-r}]}
\end{aligned}$$

where $r$ is any integer $0 \leq r \leq k$.

For $r = k$,

$$\boxed{S_{[g|2^k]} = \left(S^{-1}_{g\cdot 2^k}\right)^k}$$

□

*Example 9:* For $g = 2$ and $k = n - 1$,

$$S_{[2|2^{n-1}]} = S_{2^n} \equiv \left(S^{-1}_{2^n}\right)^{n-1}$$

which verifies one of the cyclic properties.

*Example 10:* The equivalence of

$$S_{[3|4]} = S_{[3|2^2]} \equiv (S^{-1}_{3\cdot 4})^2 = (S^{-1}_{12})^2 = S^{-1}_{12}\ S^{-1}_{12}$$

was illustrated in Fig. 2.2.

### 2-D Shuffles

The mathematical representation of shuffles of vector quantities is a fairly straightforward process. If one is dealing, however, with two-dimensional quantities, what is the corresponding mathematical representation? How does one shuffle a matrix? Simply multiplying the data matrix by a shuffling matrix would only effect a shuffle along

the rows or columns. More complicated permutation schemes cannot be expressed in the matrix domain. Instead, it is advantageous to convert the image, $\boldsymbol{X}$, into a one-dimensional object,

$$\mathbf{vec}(\boldsymbol{X}),$$

permute it,

$$\boldsymbol{P} \cdot \mathbf{vec}(\boldsymbol{X}),$$

and then re-convert the result into a matrix,

$$\boldsymbol{Y} = \mathbf{vec}^{-1}(\boldsymbol{P} \cdot \mathbf{vec}(\boldsymbol{X})).$$

This technique of transferring to the vector domain in order to operate on the two-dimensional data, e.g.,

| **Original Domain** | | **Processing Domain** | | **Result Domain** |
|---|---|---|---|---|
| *matrix* | $\Longrightarrow$ | *vector* | $\Longrightarrow$ | *matrix* |

is the only way in Kronecker algebra to represent all possible linear operations of an image.

A formal treatment of two-dimensional shuffles by means of Kronecker algebra follows.

DEFINITION 6 *(Two-dimensional Shuffle): A two-dimensional shuffle is a positional permutation of the elements of a two-dimensional array according to a particular shuffling scheme. This permutation will be denoted by the double value mapping* $\boldsymbol{\sigma}$*, i.e.,*

| *Initial Position* | | *Final Position* |
|---|---|---|
| $(i,j)$ | $\longmapsto$ | $\boldsymbol{\sigma}(i,j)$. |

A general matrix representation, $\boldsymbol{S}$, of a two-dimensional shuffle, is meaningful only in the vector domain, where

$$\mathbf{vec}(\boldsymbol{Y}) = \boldsymbol{S} \cdot \mathbf{vec}(\boldsymbol{X}).$$

*Example 11:* The *folded* or *raster-scan* two-dimensional perfect shuffle is obtained for $\boldsymbol{S} = \boldsymbol{S}_{N^2}$, i.e., a perfect shuffle of the vectorized matrix. This operation cannot be represented mathematically in the matrix domain, i.e., the folded 2-D shuffle

cannot be expressed as a simple multiplication of an input matrix representing the image. Implementation of such an interconnection in the two-dimensional domain is illustrated in Fig. 2.3 for $N = 4$.

DEFINITION 7 *(Separable Two-dimensional Shuffle): A two-dimensional shuffle is separable if the two indices can be shuffled independently:*

$$\begin{array}{cc} \underline{\textit{Initial Position}} & \underline{\textit{Final Position}} \\ (i,j) \quad \longmapsto & \quad \sigma_{p|q,r|t}(i,j) \triangleq \left(\sigma_{p|q}(i), \sigma_{r|t}(j)\right). \end{array}$$

A separable shuffle is readily expressed in the matrix domain by premultiplying by the row shuffling matrix and postmultiplying by the inverse of the column shuffling matrix,

$$\boldsymbol{Y} = \boldsymbol{S}_{[r|t]} \cdot \boldsymbol{X} \cdot \boldsymbol{S}^{-1}_{[p|q]}.$$

In the vector domain, the two-dimensional separable shuffle is represented by the matrix:

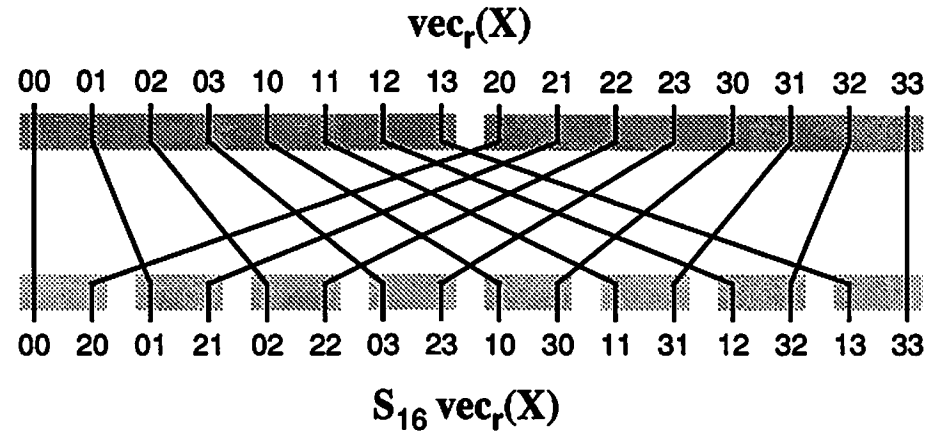
$${}^{2}\boldsymbol{S}_{[p|q][r|t]} \triangleq \begin{cases} \boldsymbol{S}_{[p|q]} \otimes \boldsymbol{S}_{[r|t]} & \textit{(In column-scanned vector-space)} \\ \\ \boldsymbol{S}_{[r|t]} \otimes \boldsymbol{S}_{[p|q]} & \textit{(In row-scanned vector-space)} \end{cases}$$

DEFINITION 8 *(Uniform Two-dimensional Shuffle): A uniform two-dimensional shuffle is defined as a separable two-dimensional permutation where both the row and the column indices undergo the same one-dimensional shuffle:*

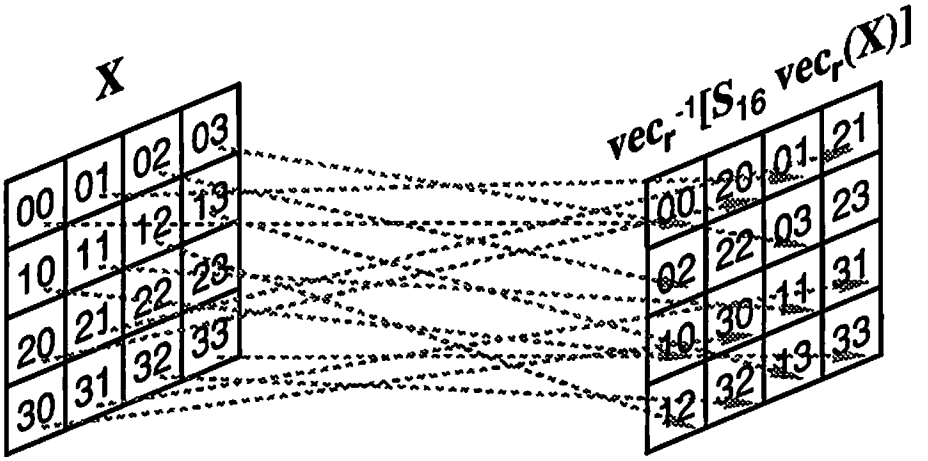
$$\begin{array}{cc} \underline{\textit{Initial Position}} & \underline{\textit{Final Position}} \\ (i,j) \quad \longmapsto & \quad \sigma_{p||q}(i,j) \triangleq \left(\sigma_{p|q}(i), \sigma_{p|q}(j)\right). \end{array}$$

Analogously, in the vector domain,

$${}^{2}\boldsymbol{S}_{[p||q]} \triangleq {}^{2}\boldsymbol{S}_{[p|q][p|q]}.$$




(a) Shuffle of row-vectorized matrix.




(b) Direct image-to-image interconnection.

Figure 2.3: Two-dimensional *folded* shuffle of a $4 \times 4$ image.

DEFINITION 9 *(Two-dimensional Separable Perfect Shuffle): A two-dimensional perfect shuffle is defined as a uniform two-dimensional shuffle where both the row and the column indices undergo a one-dimensional perfect shuffle:*

$$\begin{array}{cc} \underline{\textit{Initial Position}} & \underline{\textit{Final Position}} \\ (i,j) \quad \longmapsto & \mathcal{P}_N(i,j) \triangleq \Big(\sigma_N(i), \sigma_N(j)\Big). \end{array}$$

Similarly,

$$\mathcal{S}_N \triangleq \mathcal{S}_{[2||\frac{N}{2}]}$$

Note that $\mathcal{S}_N$ is of size $N^2 \times N^2$.

*Example 12:* Consider a $4 \times 4$ matrix, $\boldsymbol{X}$. A two-dimensional perfect shuffle of $\boldsymbol{X}$ can be expressed as follows:

$$\mathbf{vec}\,(\boldsymbol{X}') = \mathcal{S}_4 \cdot \mathbf{vec}\,(\boldsymbol{X})$$

or

$$\boldsymbol{X}' = \boldsymbol{S}_4 \cdot \boldsymbol{X} \cdot \boldsymbol{S}_4^{-1}$$

From the last expression, an implementation is directly derived as shown in Fig. 2.4.

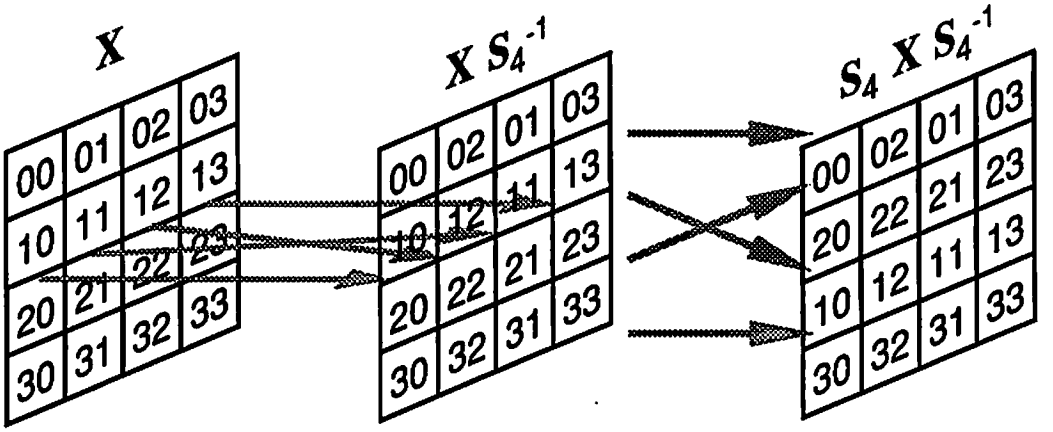
It should be noted that, in general, a separable two-dimensional perfect shuffle of a matrix is not the same as a one-dimensional perfect shuffle of a vectorized version of the given matrix.

### 2.1.4 Data Restructuring

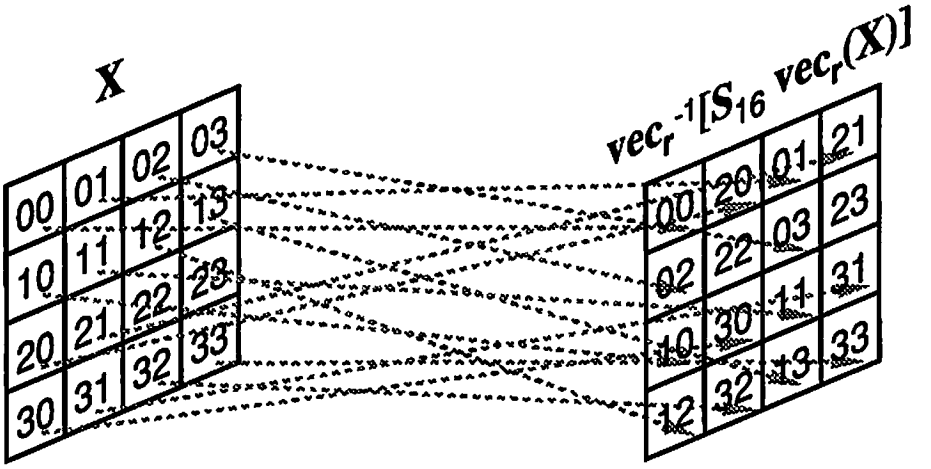
Within Kronecker algebra, it may be desirable to restructure the representation of data, e.g., convert a matrix to a vector, etc. Following are convenient identities for this purpose.

DEFINITION 10: *Let* $\underset{m\times n}{\boldsymbol{A}}$ *represent a matrix which can be uniformly partitioned into submatrices of size* $m' \times n'$. *Extraction of the* $(i,j)^{th}$ *submatrix may be specified as follows:*

$$\underset{m'\times n'}{\boldsymbol{A}_{ij}} = \left(\underset{m/m'}{\boldsymbol{u}}^{(i)^T} \otimes \underset{m'}{\boldsymbol{I}}\right) \cdot \underset{m\times n}{\boldsymbol{A}} \cdot \left(\underset{n/n'}{\boldsymbol{u}}^{(j)} \otimes \underset{n'}{\boldsymbol{I}}\right) \qquad (2\text{–}30)$$




(a) Consecutive column and row shuffles.




(b) Direct interconnection.

Figure 2.4: Two-dimensional separable shuffle of a $4 \times 4$ image.

*Example 13:* Extract the second column from the matrix $A = \begin{bmatrix} 00 & 01 & 02 \\ 10 & 11 & 12 \end{bmatrix}_{2\times 3}$

$$\begin{aligned} \underset{2\times 1}{A_{01}} &= (\underset{2/2}{u}^{(0)^T} \otimes \underset{2}{I}) \cdot \underset{2\times 3}{A} \cdot (\underset{3}{u^{(1)}} \otimes \underset{1}{I}) \\ &= \underset{2\times 3}{A} \cdot \underset{3}{u^{(1)}} \\ &= \begin{bmatrix} 00 & 01 & 02 \\ 10 & 11 & 12 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \\ &= \begin{bmatrix} 01 \\ 11 \end{bmatrix} \end{aligned}$$

*Example 14:* Extract the upper-right $2 \times 2$ submatrix from

$$B = \begin{bmatrix} 00 & 01 & 02 & 03 \\ 10 & 11 & 12 & 13 \\ 20 & 21 & 22 & 23 \\ 30 & 31 & 32 & 33 \end{bmatrix}_{4\times 4}$$

The desired submatrix is

$$\begin{aligned} \underset{2\times 2}{B_{01}} &= (\underset{4/2}{u}^{(0)^T} \otimes \underset{2}{I}) \cdot \underset{4\times 4}{B} \cdot (\underset{4/2}{u}^{(1)} \otimes \underset{2}{I}) \\ &= (\underset{2}{u^{(0)^T}} \otimes \underset{2}{I}) \cdot \underset{4\times 4}{B} \cdot (\underset{2}{u^{(1)}} \otimes \underset{2}{I}) \\ &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 00 & 01 & 02 & 03 \\ 10 & 11 & 12 & 13 \\ 20 & 21 & 22 & 23 \\ 30 & 31 & 32 & 33 \end{bmatrix} \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} 00 & 01 & 02 & 03 \\ 10 & 11 & 12 & 13 \end{bmatrix} \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} 02 & 03 \\ 12 & 13 \end{bmatrix} \end{aligned}$$

## Matrix Vectorization

The general concept of transforming a matrix, $\boldsymbol{A}$, into a vector will be denoted by the function $\mathbf{vec}(\boldsymbol{A})$ . For example, a matrix may be *lexicographically* ordered by either row or column. That is, a vector is formed by sequentially scanning the rows or columns of the matrix. The resulting column vectors are formally defined as follows:

*Column Scanning Vectorization*

$$\mathbf{vec_c}\left(\underset{p\times q}{\boldsymbol{A}}\right) \triangleq \sum_{j=0}^{q-1} \underset{q}{\boldsymbol{u}^{(j)}} \otimes \left(\underset{p\times q}{\boldsymbol{A}} \cdot \underset{q}{\boldsymbol{u}^{(j)}}\right) \triangleq \begin{bmatrix} \boldsymbol{A}_{*0} \\ ---- \\ \boldsymbol{A}_{*1} \\ ---- \\ \boldsymbol{A}_{*2} \\ ---- \\ \boldsymbol{A}_{*(q-1)} \end{bmatrix}_{pq\times 1} \tag{2-31}$$

where $\boldsymbol{A}_{*k}$ denotes the $k^{th}$ *column vector* of $\boldsymbol{A}$.

*Row Scanning Vectorization*

$$\mathbf{vec_r}\left(\underset{p\times q}{\boldsymbol{A}}\right) \triangleq \sum_{j=0}^{p-1} \underset{p}{\boldsymbol{u}^{(j)}} \otimes \left(\underset{q\times p}{\boldsymbol{A}^{T}} \cdot \underset{p}{\boldsymbol{u}^{(j)}}\right) \triangleq \begin{bmatrix} \boldsymbol{A}_{0*}^{T} \\ ---- \\ \boldsymbol{A}_{1*}^{T} \\ ---- \\ \boldsymbol{A}_{2*}^{T} \\ ---- \\ \boldsymbol{A}_{(p-1)*}^{T} \end{bmatrix}_{pq\times 1} \tag{2-32}$$

where $\boldsymbol{A}_{k*}$ denotes the $k^{th}$ *row vector* of $\boldsymbol{A}$. Note, of course, that $\mathbf{vec_r}(\boldsymbol{A}) = \mathbf{vec_c}(\boldsymbol{A}^T)$.

Corresponding inverse transformations reshape a vector $\boldsymbol{a}_n$ into a matrix of size $p \times q$ (provided $n = p \cdot q$).

$$\textit{(column-major)} \qquad \mathbf{vec_c^{-1}}(\boldsymbol{a})\Big|_{p\times q} \triangleq \sum_{k=0}^{q-1} \left(\underset{q}{\boldsymbol{u}^{(k)^T}} \otimes \underset{p}{\boldsymbol{I}}\right) \cdot \boldsymbol{a} \cdot \underset{q}{\boldsymbol{u}^{(k)^T}}$$

$$\textit{(row-major)} \qquad \mathbf{vec_r^{-1}}(\boldsymbol{a})\Big|_{p\times q} \triangleq \sum_{j=0}^{p-1} \underset{p}{\boldsymbol{u}^{(j)}} \cdot \boldsymbol{a}^T \cdot \left(\underset{p}{\boldsymbol{u}^{(j)}} \otimes \underset{q}{\boldsymbol{I}}\right)$$

Of course, vectorization is a completely reversible process (given the dimensions of the originating matrix).

$$\mathbf{vec}_{\mathbf{c}}^{-1}\left(\mathbf{vec}_{\mathbf{c}}\left(\underset{p\times q}{\boldsymbol{A}}\right)\right) \triangleq \sum_{k=0}^{q-1}\left(\underset{q}{\boldsymbol{u}^{(k)^T}}\otimes\underset{p}{\boldsymbol{I}}\right)\cdot\mathbf{vec}_{\mathbf{c}}\left(\underset{p\times q}{\boldsymbol{A}}\right)\cdot\underset{q}{\boldsymbol{u}^{(k)^T}} = \ldots = \underset{p\times q}{\boldsymbol{A}}$$

$$\mathbf{vec}_{\mathbf{r}}^{-1}\left(\mathbf{vec}_{\mathbf{r}}\left(\underset{p\times q}{\boldsymbol{A}}\right)\right) \triangleq \sum_{j=0}^{p-1}\underset{p}{\boldsymbol{u}^{(j)}}\cdot\left(\mathbf{vec}_{\mathbf{r}}\left(\underset{p\times q}{\boldsymbol{A}}\right)\right)^T\cdot\left(\underset{p}{\boldsymbol{u}^{(j)}}\otimes\underset{q}{\boldsymbol{I}}\right) = \ldots = \underset{p\times q}{\boldsymbol{A}}$$

THEOREM 13 *(Linearized Transposition): In the vector-space domain, the transposition of a matrix,* $\underset{p\times q}{\boldsymbol{A}}$*, is a linear operation, namely a shuffle:*

$$\mathbf{vec}_{\mathbf{c}}\left(\underset{q\times p}{\boldsymbol{A}^T}\right) = \boldsymbol{S}_{[p|q]}^{-1}\mathbf{vec}_{\mathbf{c}}\left(\underset{p\times q}{\boldsymbol{A}}\right) \tag{2–33}$$

$$\mathbf{vec}_{\mathbf{r}}\left(\underset{q\times p}{\boldsymbol{A}^T}\right) = \boldsymbol{S}_{[p|q]}\mathbf{vec}_{\mathbf{r}}\left(\underset{p\times q}{\boldsymbol{A}}\right) \tag{2–34}$$

*Proof:* [7]. □

COROLLARY 14 *(Shuffling of vectorized matrices): The row and column scanned vectors of a matrix have a simple shuffled relationship:*

$$\mathbf{vec}_{\mathbf{c}}\left(\underset{p\times q}{\boldsymbol{A}}\right) = \boldsymbol{S}_{[p|q]}\mathbf{vec}_{\mathbf{r}}\left(\underset{p\times q}{\boldsymbol{A}}\right) \tag{2–35}$$

*and*

$$\mathbf{vec}_{\mathbf{r}}\left(\underset{p\times q}{\boldsymbol{A}}\right) = \boldsymbol{S}_{[p|q]}^{-1}\mathbf{vec}_{\mathbf{c}}\left(\underset{p\times q}{\boldsymbol{A}}\right) \tag{2–36}$$

Other useful identities are listed in Table 2.5.

*Proof:*

(2–41): Proof is given in [65].

(2–42): Since $\mathbf{vec}_{\mathbf{r}}(\boldsymbol{A}) = \mathbf{vec}_{\mathbf{c}}(\boldsymbol{A}^T)$, it follows from (2–41) that:

$$\begin{aligned}\mathbf{vec}_{\mathbf{r}}(\boldsymbol{RDC}) &= \mathbf{vec}_{\mathbf{c}}((\boldsymbol{RDC})^T)\\ &= \mathbf{vec}_{\mathbf{c}}(\boldsymbol{C}^T\boldsymbol{D}^T\boldsymbol{R}^T)\\ &= (\boldsymbol{R}\otimes\boldsymbol{C}^T)\mathbf{vec}_{\mathbf{c}}(\boldsymbol{D}^T)\\ &= (\boldsymbol{R}\otimes\boldsymbol{C}^T)\mathbf{vec}_{\mathbf{r}}(\boldsymbol{D})\end{aligned}$$

□

$$\mathbf{vec_c}\left(\underset{q\times p}{\boldsymbol{A}^T}\right) = \boldsymbol{S}_{[p|q]}^{-1}\mathbf{vec_c}\left(\underset{p\times q}{\boldsymbol{A}}\right) \tag{2-37}$$

$$\mathbf{vec_r}\left(\underset{q\times p}{\boldsymbol{A}^T}\right) = \boldsymbol{S}_{[p|q]}\mathbf{vec_r}\left(\underset{p\times q}{\boldsymbol{A}}\right) \tag{2-38}$$

$$\mathbf{vec_c}\left(\underset{p\times q}{\boldsymbol{A}}\right) = \boldsymbol{S}_{[p|q]}\mathbf{vec_r}\left(\underset{p\times q}{\boldsymbol{A}}\right) \tag{2-39}$$

$$\mathbf{vec_r}\left(\underset{p\times q}{\boldsymbol{A}}\right) = \boldsymbol{S}_{[p|q]}^{-1}\mathbf{vec_c}\left(\underset{p\times q}{\boldsymbol{A}}\right) \tag{2-40}$$

$$\mathbf{vec_c}\left(\underset{s\times r}{\boldsymbol{R}}\ \underset{r\times q}{\boldsymbol{D}}\ \underset{q\times t}{\boldsymbol{C}}\right) = \left(\underset{t\times q}{\boldsymbol{C}^T} \otimes \underset{s\times r}{\boldsymbol{R}}\right)\mathbf{vec_c}\left(\underset{r\times q}{\boldsymbol{D}}\right) \tag{2-41}$$

$$\mathbf{vec_r}\left(\underset{s\times r}{\boldsymbol{R}}\ \underset{r\times q}{\boldsymbol{D}}\ \underset{q\times t}{\boldsymbol{C}}\right) = \left(\underset{s\times r}{\boldsymbol{R}} \otimes \underset{t\times q}{\boldsymbol{C}^T}\right)\mathbf{vec_r}\left(\underset{r\times q}{\boldsymbol{D}}\right) \tag{2-42}$$

Table 2.5: Vectorization Identities

### 2.1.5 Other Outer Products

The left and right Kronecker products represent two members of a family of four basic outer product operations as described by Eisele and Mason [12]. A third type of outer product is, what has been called by Tait[59], the *star product.*

DEFINITION 11 *(Star Product): The star product of two matrices* $\underset{p\times m}{\boldsymbol{A}}$ *and* $\underset{q\times n}{\boldsymbol{B}}$ *is an outer product defined as:*

$$\underset{p\times m}{\boldsymbol{A}} \star \underset{q\times n}{\boldsymbol{B}} = \begin{bmatrix} A_{*1}B_{1*} & A_{*2}B_{1*} & \cdots & A_{*m}B_{1*} \\ A_{*1}B_{2*} & A_{*2}B_{2*} & \cdots & A_{*m}B_{2*} \\ \vdots & \vdots & \ddots & \vdots \\ A_{*1}B_{q*} & A_{*2}B_{q*} & \cdots & A_{*m}B_{q*} \end{bmatrix}_{pq\times mn}$$

*where* $A_{*i}$ *is the* $i^{th}$ *column of* $A$ *and* $B_{i*}$ *the* $i^{th}$ *row of* $B$.

The star product follows a column-by-row multiplication rule as opposed to a row-by-column rule of ordinary matrix multiplication.

A set of basic properties analogous to those of the Kronecker product are presented in Table 2.6.

| | | |
|---|---|---|
| *Associativity:* | $\underset{p\times q}{A} \star (\underset{m\times n}{B} \star \underset{k\times l}{C}) = (\underset{p\times q}{A} \star \underset{m\times n}{B}) \star \underset{k\times l}{C}$ | (2–43) |
| *Distributivity:* | $(\underset{p\times q}{A} + \underset{p\times q}{H}) \star (\underset{m\times n}{B} + \underset{m\times n}{R}) =$ $\underset{p\times q}{A} \star \underset{m\times n}{B} + \underset{p\times q}{A} \star \underset{m\times n}{R} + \underset{p\times q}{H} \star \underset{m\times n}{B} + \underset{p\times q}{H} \star \underset{m\times n}{R}$ | (2–44) |
| *Mixed Products:* | $(\underset{p\times q}{A} \star \underset{m\times n}{B})(\underset{n\times k}{C} \star \underset{q\times t}{D}) = \underset{m\times n}{B}\,\underset{n\times k}{C} \otimes \underset{p\times q}{A}\,\underset{q\times t}{D}$ | (2–45) |
| | $(\underset{p\times q}{A}\,\underset{q\times t}{D}) \star (\underset{m\times n}{B}\,\underset{n\times k}{C}) = (\underset{m\times n}{B} \otimes \underset{p\times q}{A})(\underset{q\times t}{D} \star \underset{n\times k}{C})$ $= (\underset{p\times q}{A} \star \underset{m\times n}{B})(\underset{q\times t}{D} \otimes \underset{n\times k}{C})$ | (2–46) |
| *Transposition:* | $(\underset{p\times q}{A} \star \underset{m\times n}{B})^T = \underset{n\times m}{B}^T \star \underset{q\times p}{A}^T$ | (2–47) |
| *Inversion:* | $(\underset{n}{N} \star \underset{m}{M})^{-1} = \underset{m}{M}^{-1} \star \underset{n}{N}^{-1}$ | (2–48) |
| *Traces:* | $trace(\underset{n}{N} \star \underset{m}{M}) = trace(\underset{n}{N}\,\underset{m}{M})$ | (2–49) |
| *Determinants:* | $\det(\underset{n}{N} \otimes \underset{m}{M}) = \det(S_{[n\|m]})\,(\det \underset{n}{N})^n\,(\det \underset{m}{M})^m$ | (2–50) |

Table 2.6: Star Product Properties

There are two important observations to be made:

*Observation 1:* The star product is nothing more than a shuffled Kronecker product,

$$\underset{p\times m}{A} \star \underset{q\times n}{B} = S_{[p|q]}(\underset{p\times m}{A} \otimes \underset{q\times n}{B}). \tag{2–51}$$

*Observation 2:* The star product of two identity matrices is a shuffle,

$$\underset{m}{I} \star \underset{n}{I} = S_{[m|n]}. \tag{2–52}$$

Although Tait[59] did recognize the significance of $\underset{m}{I} \star \underset{n}{I}$ in relating the two outer products, the connection to shuffle permutations and their fundamental role in array algebra was not made explicit. The star product, being a shuffled Kronecker product,

is not really a fundamental operation, as stated by Tait, but rather a mathematical convenience.

The fourth outer product is the vector outer product of vectorizations of the two matrices

$$\mathbf{vec}(\boldsymbol{A}) \cdot \mathbf{vec}(\boldsymbol{B})$$

It is presented only to complete the description of the four basic outer product operations.

## 2.2 Array Algebra

The mathematical manipulation of Kronecker products can get quite involved and cumbersome. Typically, there are a number of summation signs to carry along and the application of identities usually requires a decomposition into several indexed terms.

A more compact way to carry out the manipulations is afforded by the use of a special algebra first introduced by Tait [59] in 1971 as a means of performing multivariate analysis—in particular, matrix differentiation. It has been re-examined since then by Vetter and others [58, 65, 66, 67]. The basic concept is to go beyond the two-dimensional world of matrix algebra, and into the higher dimensional domain of *array algebra*, where arrays are treated as multidimensional scalars. Operations performed in the array domain have a corresponding effect in the matrix domain and vice versa. The strategy is to transform expressions from the matrix domain into the array domain where the data structures are easily manipulated and then transform back into the matrix domain. A simple analogy can be drawn to the Fourier domain where, for example, a simple point-by-point multiplication is equivalent to a time domain convolution. The most common duality relations are catalogued for easy reference (much like those of the Fourier transform) and others can be derived as needed. What follows is an alternative interpretation and extension of array algebra.

### 2.2.1 Array Notation

Matrix algebra and Kronecker algebra deal with vectors and matrices in the *vector domain* or *matrix domain*, respectively. These quantities are at most two-dimensional, and

are operated on in a specific structured manner. Array algebra manipulates quantities of arbitrary dimension, in an *array domain.*

DEFINITION 12: *An* array, $\mathcal{A}$, *is defined as a collection or set of unlabelled elements.*

For example, $\mathcal{A} = \{x : x$ is a pixel value of a $512 \times 512$ digitized picture$\}$.

DEFINITION 13: *A* dimensioned *or* indexed array *is an array whose elements are referenced by a set of indices or coordinates, e.g.,* $(i, j, k, \ldots)$. *An indexed array may be denoted by a variably indexed element, e.g.,* $\{\mathcal{A}(i, j, k, \ldots)\}$.

The number of variables in the index set define the *dimensionality* of the array. The exact indexing of the elements may be arbitrary, but each different labelling defines a unique indexed array. All indexed arrays derived from the same simple array are *informationally equivalent.*

DEFINITION 14: *Simple and indexed arrays comprise the* array domain.

The array domain may be thought of as the conceptual domain of acquired data. Thus, an array containing an image may be naturally envisioned as two-dimensional with a certain orientation and pixel labelling. Of course, it may also be considered as a composite pattern of submatrices (or *tiles*) which may be more naturally organized as a stack, (i.e., a three-dimensional array) or as a rectangular tessellation of tiles (i.e., a four-dimensional array).

DEFINITION 15: *An* array expression *is a scalar mathematical expression of multiple array elements, e.g.,* $\mathcal{A}(i, j) + \mathcal{B}(m, n)$.

Note that all scalar properties hold, for example,

$$\textit{(Commutativity):} \qquad \mathcal{A}(i, j)\mathcal{B}(k, m, n) = \mathcal{B}(k, m, n)\mathcal{A}(i, j)$$

DEFINITION 16: *An* array kernel expression *is an array expression used to generate elements of a new array over the range of the indices.*

For example, the kernel expression, $\{\mathcal{A}(i,j) + \mathcal{B}(m,n)\}$, generates an array of new elements over the range of the indices $i, j, m$, and $n$, such that

$$\mathcal{C} = \{x : \; x = \mathcal{A}(i,j) + \mathcal{B}(m,n) \qquad \forall \;\; 1 \leq i,m \leq M \text{ and } 1 \leq j,n \leq N\}.$$

Note that the generated elements belong to a simple array, not an indexed one. The terms of an array expression are treated as scalars.

DEFINITION 17: *An* **array equation** *is the assignment of an array kernel expression to a specific element of an indexed array.*

For example, $\mathcal{C}(i,j,m,n) := \mathcal{A}(i,j) + \mathcal{B}(m,n)$, generates an indexed array of new elements over the range of the indices $i, j, m$, and $n$, such that

$$\mathcal{C} = \Big\{\mathcal{C}(i,j,m,n) : \;\; \mathcal{C}(i,j,m,n) \;\; = \;\; \mathcal{A}(i,j) + \mathcal{B}(m,n) \qquad \forall \;\; 1 \leq i,m \leq M \text{ and } 1 \leq j,n \leq N\Big\}.$$

In order to mathematically manipulate an array in matrix or Kronecker algebra, a mapping must be performed from the array domain into the vector or matrix domain. This step is, often, superfluous since the array may already be organized as the desired data structure. In array algebra, however, the mapping is always considered explicitly.

It is apparent that there are several ways to map from a dimensioned array to a vector or a matrix. Consider, for example, the set (or array) of upper and lower case letters of the english alphabet,

$$\mathcal{A} = \{\text{lower-case english letter}\} \cup \{\text{upper-case english letter}\}$$

There are at least two possible ways to index $\mathcal{A}$. A one-dimensional approach may have the lower case letters followed by the upper case letters; both subsets sequenced in

the same fashion—the contemporary standard. Alternatively, a two-dimensional array seems a bit more natural in order to separate small and capital characters. The two-dimensional array may be mapped into at least two different matrices; one the transpose of the other.

Transforming an indexed array $\{\mathcal{A}(x_1, x_2, ..., x_n)\}$ into a matrix $[A(i,j)]$, is based on a mapping of the n-tuple array index set into a pair of matrix indices, i.e.,

$$(x_1, x_2, ..., x_n) \mapsto (i, j).$$

Such a mapping is not unique and therefore an array may be represented by several different matrices.

In order to preserve the ability to map back and forth between dimensioned arrays and the vector/matrix domain, a new type of dimensioned array will be introduced; one that preserves the positional mapping information.

DEFINITION 18: *A* **matrix-array** *(m-array),* $\{\mathcal{A}|_{i_{k+1}\cdots i_n}^{i_1\cdots i_k}\}$, *is an indexed array represented by a unique matrix,* $[a_{ij}]$, *generated by applying an implicit element mapping,*

$$\mathcal{A}|_{i_{k+1}\cdots i_n}^{i_1\cdots i_k} : \quad \mathcal{A}(i_1\cdots i_n) \mapsto a_{f(i_1\cdots i_k), f(i_{k+1}\cdots i_n)} \quad .$$

*with the matrix index function,*

$$f(i_1, i_2, i_3, \cdots, i_q) = i_1 + i_2 \cdot \langle i_1 \rangle + i_3 \cdot \langle i_1 i_2 \rangle + \cdots + i_q \cdot \langle i_1 i_2 \cdots i_{q-1} \rangle$$

*where each index,* $i_m$, *starts count at zero and is incremented to span a total of* $\langle i_m \rangle$ *counts.*

The sequences $i_1 \cdots i_k$ and $i_{k+1} \cdots i_n$ are referred to as row and column index sequences, respectively. The correspondence between a matrix-array, $\{\mathcal{A}|_{i_{k+1}\cdots i_n}^{i_1\cdots i_k}\}$ (or its generating element $\mathcal{A}|_{i_{k+1}\cdots i_n}^{i_1\cdots i_k}$) and a matrix, $\boldsymbol{A}$, is denoted by a $\Longleftrightarrow$ sign, i.e.,

$$\{\mathcal{A}|_{i_{k+1}\cdots i_n}^{i_1\cdots i_k}\} \iff \boldsymbol{A}$$

or

$$\mathcal{A}|_{i_{k+1}\cdots i_n}^{i_1\cdots i_k} \iff \boldsymbol{A}$$

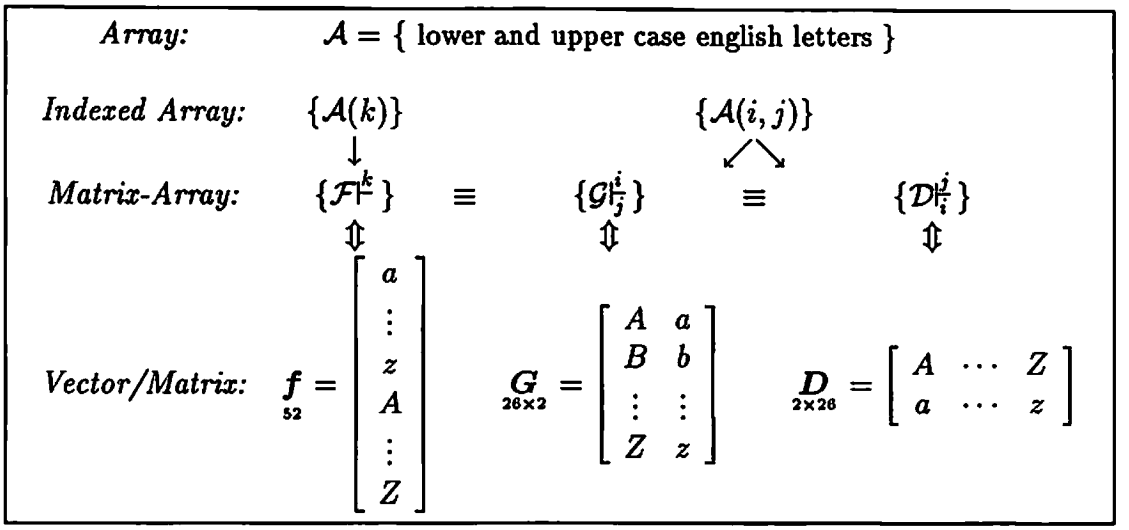



Figure 2.5: Array and Matrix Domain Example

In Fig. 2.5, the alphabet example is shown in terms of the array and vector/matrix domains.

As further illustration, note that a three-dimensional array, $\mathcal{A}(i,j,k)$, may be mapped into a matrix representation by any of the following matrix-arrays:

$$\mathcal{A}\vdash^{i}_{jk} : \mathcal{A}(i,j,k) \mapsto a_{f(i)\ f(jk)}$$

or

$$\mathcal{A}\vdash^{ij}_{k} : \mathcal{A}(i,j,k) \mapsto a_{f(ij)\ f(k)}$$

or

$$\mathcal{A}\vdash^{ik}_{j} : \mathcal{A}(i,j,k) \mapsto a_{f(ik)\ f(j)}$$

The effect of the function $f(\cdot)$ is to unravel the index sequence in a left-to-right order. The leftmost pair of row and column indices are the first to be expanded throughout their ranges. The next pair is then expanded, and so on, until all the indices have been unraveled.

As an illustration, consider the m-array representation, $\mathcal{B}\vdash^{im}_{jn}$, of a four-dimensional array $\mathcal{B}(i,m,j,n)$. If the ranges of the indices are

$$1 \le i,j,m \le 2 \ \text{ and } \ 1 \le n \le 3,$$

then, the first expansion $\vdash_j^i$ yields

$$\mathcal{B}\vdash_{jn}^{im} \iff \underset{4\times 6}{\boldsymbol{B}} = \begin{bmatrix} \mathcal{B}\vdash_{1n}^{1m} & \mathcal{B}\vdash_{2n}^{1m} \\ \mathcal{B}\vdash_{1n}^{2m} & \mathcal{B}\vdash_{2n}^{2m} \end{bmatrix}$$

and with the second expansion,

$$\underset{4\times 6}{\boldsymbol{B}} = \begin{bmatrix} \mathcal{B}\vdash_{11}^{11} & \mathcal{B}\vdash_{12}^{11} & \mathcal{B}\vdash_{13}^{11} & \mathcal{B}\vdash_{21}^{11} & \mathcal{B}\vdash_{22}^{11} & \mathcal{B}\vdash_{23}^{11} \\ \mathcal{B}\vdash_{11}^{12} & \mathcal{B}\vdash_{12}^{12} & \mathcal{B}\vdash_{13}^{12} & \mathcal{B}\vdash_{21}^{12} & \mathcal{B}\vdash_{22}^{12} & \mathcal{B}\vdash_{23}^{12} \\ \mathcal{B}\vdash_{11}^{21} & \mathcal{B}\vdash_{12}^{21} & \mathcal{B}\vdash_{13}^{21} & \mathcal{B}\vdash_{21}^{21} & \mathcal{B}\vdash_{22}^{21} & \mathcal{B}\vdash_{23}^{21} \\ \mathcal{B}\vdash_{11}^{22} & \mathcal{B}\vdash_{12}^{22} & \mathcal{B}\vdash_{13}^{22} & \mathcal{B}\vdash_{21}^{22} & \mathcal{B}\vdash_{22}^{22} & \mathcal{B}\vdash_{23}^{22} \end{bmatrix}$$

$$\Longrightarrow \mathcal{B}\vdash_{jn}^{im} \iff \underset{4\times 6}{\boldsymbol{B}} = \begin{bmatrix} b_{11} & b_{12} & b_{13} & b_{14} & b_{15} & b_{16} \\ b_{21} & b_{22} & b_{23} & b_{24} & b_{25} & b_{26} \\ b_{31} & b_{32} & b_{33} & b_{34} & b_{35} & b_{36} \\ b_{41} & b_{42} & b_{43} & b_{44} & b_{45} & b_{46} \end{bmatrix}$$

The mapping of the elements $\mathcal{B}\vdash_{jn}^{im}$ to the elements, $b_{st}$, of $\underset{4\times 6}{\boldsymbol{B}}$ is, obviously,

$$\begin{aligned} s &= f(i,m) &&= i + (m-1)\langle i \rangle \\ t &= f(j,n) &&= j + (n-1)\langle j \rangle. \end{aligned}$$

(Note that the indices in this case start at *one*.)

The size of a m-array is implied by the ranges of its individual index variables. In the above expansion, $\{\mathcal{B}\vdash_{jn}^{im}\}$ has $\langle i \rangle \cdot \langle m \rangle = 2 \cdot 2 = 4$ elements in its row index and $\langle j \rangle \cdot \langle n \rangle = 2 \cdot 3 = 6$ in its column which is the size of its corresponding matrix, $\underset{4\times 6}{\boldsymbol{B}}$. The size of a missing index sequence is *one* by definition, e.g., $\mathcal{C}\vdash^{im} \iff \underset{4\times 1}{\boldsymbol{C}}$.

DEFINITION 19: *A* matrix-array expression *is an array expression that is defined in terms of matrix-arrays which are treated as scalars.*

The equivalence of two matrix array expressions is denoted by the equal sign, e.g.,

$$\mathcal{A}\vdash_j^i \mathcal{B}\vdash_k^n = \mathcal{B}\vdash_k^n \mathcal{A}\vdash_j^i$$

DEFINITION 20: *A* **matrix-array equation** *(or simply* m-array equation*) is an array equation employing matrix-array expressions. The symbol* := *is used to emphasize the assignment aspect of this equation.*

For example, the outer product of two vectors corresponds to the array equation,

$$\mathcal{C}\vdash_{\vec{j}}^{\vec{i}} := \mathcal{A}\vdash^{\vec{i}}\mathcal{B}\vdash_{\vec{j}}$$

which assigns values to the matrix-array elements $\mathcal{C}\vdash_{\vec{j}}^{\vec{i}}$.

Tait has used the tensor summation (or Einstein) notation with respect to common index variables: **Elements or products of elements are summed over the range of any replicated index variable.**

For example,

$$\mathcal{A}\vdash_{\vec{j}}^{\vec{i}}\mathcal{B}\vdash_{\vec{k}}^{\vec{j}} \triangleq \sum_{j=0}^{\langle j\rangle-1} \mathcal{A}(i,j)\mathcal{B}(j,k)$$

$$\mathcal{A}\vdash_{\vec{i}}^{\vec{i}} \triangleq \sum_{i=0}^{\langle i\rangle-1} \mathcal{A}(i,i)$$

However, such a notation prevents the expression of point-by-point operations, e.g., the element-by-element multiplication of two vectors should be denoted as

$$\mathcal{C}\vdash^{\vec{i}} := \mathcal{A}\vdash^{\vec{i}}\mathcal{B}\vdash^{\vec{i}}.$$

Therefore, the summation notation will only be used in the case of array contraction, i.e., in the context of a matrix-array equation where the assigned m-array does not contain the index variable in question. For example,

$$\mathcal{C}\vdash_{\vec{k}}^{\vec{i}} := \mathcal{A}\vdash_{\vec{j}}^{\vec{i}}\mathcal{B}\vdash_{\vec{k}}^{\vec{j}} \quad \text{matrix multiplication}$$

$$\mathcal{C}\vdash_{\vec{j}}^{\vec{i}} := \mathcal{A}\vdash_{\vec{j}}^{\vec{i}}\mathcal{B}\vdash_{\vec{j}}^{\vec{i}} \quad \text{point-by-point multiplication}$$

### 2.2.2 General Properties/Identities

The correspondence between matrix equations and array equations leads to the use of *array algebra* in place of more complicated matrix or Kronecker algebra operations.

Some basic duality relations are:

$$\text{Transposition:} \quad C = A^T \iff C\|_i^j := A\|_j^i \tag{2-53}$$

$$\text{Addition:} \quad C = A \pm B \iff C\|_j^i := A\|_j^i \pm B\|_j^i \tag{2-54}$$

$$\text{Inner Product:} \quad C = AB \iff C\|_y^i := A\|_j^i B\|_y^j \tag{2-55}$$

$$\text{Kronecker Product:} \quad C = A \otimes B \iff C\|_{jy}^{ix} := A\|_j^i B\|_y^x \tag{2-56}$$

$$\text{Star Product:} \quad C = A \star B \iff C\|_{jy}^{xi} := A\|_j^i B\|_y^x \tag{2-57}$$

$$\text{Vectorization:} \quad c = \mathrm{vec_c}(A) \iff C\|^{ji} := A\|_j^i \tag{2-58}$$

$$c = \mathrm{vec_r}(A) \iff C\|^{ij} := A\|_j^i \tag{2-59}$$

One may question whether there is a corresponding matrix equation for every array-matrix equation. According to Tait[59], such is the case.

THEOREM 15: *Let $i_a = i\cdots$ and $j_a = j\cdots$ be disjoint sequences*[2] *and let $i_b$ and $j_b$ be disjoint and exhaustive sequences arbitrarily drawn from $\{i_a\} \cup \{j_a\}$. If $B\|_{j_b}^{i_b} := A\|_{j_a}^{i_a}$, then their corresponding matrices are related by a binary transformation, $T(\cdot)$,*

$$B\|_{j_b}^{i_b} := A\|_{j_a}^{i_a} \iff B = T(A).$$

A proof is given in [59], based on the premise that any permutation of the elements of $i_a$ and $j_a$ amounts to a repeated application of index exchanges within a sequence and/or transferring the first index of one sequence into the first position of the other. There exists a matrix transformation for each of these two index manipulations.

### 2.2.3 Data Restructuring

Just as matrix restructuring was defined in terms of Kronecker algebra in Section 2.1.4, it can also be defined in terms of Array algebra. Any two data structures may be related by means of their corresponding matrices by applying a combination of the following two rules.

[2]Sequences of disjoint index sets

**(I) Exchanging indices within a sequence** The rules for exchanging any two indices within a row or column sequence are are defined below:

$$\mathcal{B}\Big|_{x\cdots}^{ij\widehat{kl}m\cdots} := \mathcal{A}\Big|_{x\cdots}^{ik\widehat{jl}m\cdots} \iff \boldsymbol{B} = \Big[\underset{(i)}{\boldsymbol{I}} \otimes (\underset{(k)}{\boldsymbol{I}} \star \underset{(j)}{\boldsymbol{I}}) \otimes \underset{(l)}{\boldsymbol{I}} \otimes \underset{(m)}{\boldsymbol{I}} \otimes \cdots\Big]\boldsymbol{A} \qquad (2\text{–}60)$$

$$\mathcal{B}\Big|_{x\cdots}^{ij\widehat{kl}m\cdots} := \mathcal{A}\Big|_{x\cdots}^{il\widehat{kj}m\cdots} \iff \boldsymbol{B} = \Big[\underset{(i)}{\boldsymbol{I}} \otimes (\underset{(l)}{\boldsymbol{I}} \star \underset{(k)}{\boldsymbol{I}} \star \underset{(j)}{\boldsymbol{I}}) \otimes \underset{(m)}{\boldsymbol{I}} \otimes \cdots\Big]\boldsymbol{A} \qquad (2\text{–}61)$$

Column index exchanges are similar with identical permutation matrices post-multiplying $\boldsymbol{A}$ instead of pre-multiplying it. These rules are easily derived by application of the star and Kronecker relationships.

*Example 1:* If $\mathcal{C}\Big|_{xy}^{ji} := \mathcal{R}\Big|_{xy}^{ij}$, then

$$\boldsymbol{C} = (\underset{(j)}{\boldsymbol{I}} \star \underset{(i)}{\boldsymbol{I}})\boldsymbol{R} = \boldsymbol{S}_{[(j)|(i)]}\boldsymbol{R}.$$

This correspondence may be used to derive two important relationships previously stated.

(a) For

$$\mathcal{C}\Big|_{xy}^{ji} = \mathcal{A}\Big|_{x}^{i}\mathcal{B}\Big|_{y}^{j} \iff \boldsymbol{C} = \underset{(i)\times(x)}{\boldsymbol{A}} \star \underset{(j)\times(y)}{\boldsymbol{B}}$$

and

$$\mathcal{R}\Big|_{xy}^{ij} = \mathcal{A}\Big|_{x}^{i}\mathcal{B}\Big|_{y}^{j} \iff \boldsymbol{R} = \underset{(i)\times(x)}{\boldsymbol{A}} \otimes \underset{(j)\times(y)}{\boldsymbol{B}},$$

the relationship between Kronecker and star products is verified, i.e.,

$$\boldsymbol{A} \star \boldsymbol{B} = \boldsymbol{S}_{[(j)|(i)]}\big(\underset{(i)\times(x)}{\boldsymbol{A}} \otimes \underset{(j)\times(y)}{\boldsymbol{B}}\big).$$

(b) Similarly for

$$\mathcal{C}\Big|^{ji} = \mathcal{A}\Big|_{j}^{i} \iff \boldsymbol{C} = \mathbf{vec_c}(\boldsymbol{A})$$

and

$$\mathcal{R}\Big|^{ij} = \mathcal{A}\Big|_{j}^{i} \iff \boldsymbol{R} = \mathbf{vec_r}(\boldsymbol{A}),$$

the relationship between column and and row vectorization is verified,

$$\mathbf{vec_c}(\boldsymbol{A}) = \boldsymbol{S}_{[(j)|(i)]}\underset{(i)\times(j)}{\boldsymbol{A}}$$

**(II) Transferring the first index between sequences** The rules for raising and lowering the first index of a sequence are stated as:

$$\mathcal{A}|_{xy\cdots}^{\dot{i}j\cdots} := \mathcal{B}|_{\dot{i}xy\cdots}^{j\cdots} \quad \Longleftrightarrow \quad \boldsymbol{A} = (\underset{(i)}{\boldsymbol{I}} \otimes \boldsymbol{B}) \left[ \underset{(i)^2}{\bar{\boldsymbol{I}}} \otimes \underset{(xy\cdots)}{\boldsymbol{I}} \right] \tag{2-62}$$

$$\mathcal{A}|_{\dot{x}y\cdots}^{ij\cdots} := \mathcal{B}|_{y\cdots}^{\dot{x}ij\cdots} \quad \Longleftrightarrow \quad \boldsymbol{A} = \left[ \underset{(i)^2}{\bar{\boldsymbol{I}}}^{T} \otimes \underset{(ij\cdots)}{\boldsymbol{I}} \right] (\underset{(x)}{\boldsymbol{I}} \otimes \boldsymbol{B}) \tag{2-63}$$

where $\underset{(i)^2}{\bar{\boldsymbol{I}}} = \mathbf{vec}\left(\underset{(i)}{\boldsymbol{I}}\right)$.

This rule may be used to derive an alternative expression of vectorization:

COROLLARY 16:

$$\mathbf{vec_c}\left(\underset{p\times q}{\boldsymbol{A}}\right) = (\underset{q}{\boldsymbol{I}} \otimes \underset{p\times q}{\boldsymbol{A}})\, \underset{q^2}{\bar{\boldsymbol{I}}} \tag{2-64}$$

$$\mathbf{vec_r}\left(\underset{p\times q}{\boldsymbol{A}}\right) = \boldsymbol{S}_{[p|q]}(\underset{q}{\boldsymbol{I}} \otimes \underset{p\times q}{\boldsymbol{A}})\, \underset{q^2}{\bar{\boldsymbol{I}}} \tag{2-65}$$

*Proof:*

(a) By definition, $\boldsymbol{c} = \mathbf{vec_c}\left(\underset{p\times q}{\boldsymbol{A}}\right) \Longleftrightarrow \mathcal{C}|^{\dot{j}i} = \mathcal{A}|_{\dot{j}}^{i}$ where the index $j$ was raised to achieve column vectorization. Using (2–62), it is obvious that

$$\boldsymbol{c} = (\underset{q}{\boldsymbol{I}} \otimes \boldsymbol{A})\, \underset{q^2}{\bar{\boldsymbol{I}}}$$

(b) By definition, $\boldsymbol{r} = \mathbf{vec_r}\left(\underset{p\times q}{\boldsymbol{A}}\right) \Longleftrightarrow \mathcal{R}|^{\dot{i}\dot{j}} = \mathcal{A}|_{\dot{j}}^{i}$ where the index $j$ was raised and then exchanged into the second position. Since, the raising operation was just shown, the exchange operation, yields

$$\boldsymbol{r} = \boldsymbol{S}_{[q|p]}\boldsymbol{c} = \boldsymbol{S}_{[q|p]}(\underset{q}{\boldsymbol{I}} \otimes \boldsymbol{A})\, \underset{q^2}{\bar{\boldsymbol{I}}}$$

□

### 2.2.4 Programming Perspective

One may gain better insight into array algebra by giving it a programming interpretation. An array equation can be considered simply a short-hand notation for implementing programmatically the corresponding matrix operation. For example, the Kronecker product of two matrices,

$$\underset{pw \times qz}{C} = \underset{p \times q}{A} \otimes \underset{w \times z}{B}$$

is implemented by the array equation:

$$C\vdots_{jn}^{im} := A\vdots_{j}^{i} B\vdots_{n}^{m}$$

which is short-hand for the program segment:

**for** $i = 1, \ldots, p$
  **for** $j = 1, \ldots, q$
    **for** $m = 1, \ldots, w$
      **for** $n = 1, \ldots, z$
        $C(\ \mathrm{map}(i,m),\ \mathrm{map}(j,n)) := A(i,j) * B(m,n)$

where

**function** $\mathrm{map}(x, y)$
  $\mathrm{map} = x + (y-1) * count(x)$

The above is a concise implementation of a Kronecker product in a sequential language. Note that there are no *data-dependencies* in the assignment statement and therefore the elements of the resulting array, $C$, may be generated independently of each other, i.e., in parallel. In this programmatic setting, the characteristics of array equations become immediately apparent. Arrays are treated as scalars because they represent the kernel or characteristic operation of the resulting elements. Whether one writes, $A(i,j) * B(m,n)$ or $B(m,n) * A(i,j)$ is of no consequence and therefore,

$$A\vdots_{j}^{i} B\vdots_{n}^{m} = B\vdots_{n}^{m} A\vdots_{j}^{i}.$$

Using array-matrix expressions imposes a specific coordinate mapping between a four-dimensional result $C'(i,j,m,n)$ and a desired two-dimensional matrix $C(s,t)$. It is this

mapping function, *map()*, that determines if the resulting matrix is a Kronecker product or the *informational equivalent* star product.

$$\underset{pw \times qz}{D} = \underset{p \times q}{A} * \underset{w \times z}{B} \iff \mathcal{D}\|_{jn}^{mi} := \mathcal{A}\|_{j}^{i}\,\mathcal{B}\|_{n}^{m}$$

which is implemented by:

```
for i = 1,...,p
    for j = 1,...,q
        for m = 1,...,w
            for n = 1,...,z
                D( map(m,i), map(j,n)) := A(i,j) * B(m,n)
```

The characteristic expression of both products is the same. It is structuring of $C$ and $D$ that is different.

Actually, there are 24(= 4!) different ways of mapping the variables $i, j, m, n$ to a two-dimensional matrix. Such informationally equivalent matrices can be expressed in matrix algebra by one of the four basic outer product operations and the transpose operation [12].

### 2.2.5 General Application

Array algebra is a useful tool in proving vector space equations as well as synthesizing new relationships. Given a matrix expression, the following methodology will produce one or more alternative expressions.

1. Start with a high-level matrix expression
2. Equate the expression to a substitution matrix in order to form a matrix equation.
3. Derive an equivalent m-array equation. The sets of row and column indices of the substitution m-array will be called the *base* sets.
   (This step may require successive applications of duality identities until a pure m-array expression on the right-hand side is produced.)

4. "Judiciously" manipulate the terms of the m-array expression. This includes rearrangement of the m-array terms and/or introduction of identity terms (e.g., $\mathcal{A}\big|^{i}_{j} = \mathcal{I}\big|^{i}_{x}\,\mathcal{A}\big|^{x}_{z}\,\mathcal{I}\big|^{z}_{j}$).

5. Apply appropriate identities to convert the new m-array expression into a m-array expression of a purely matrix expression.

6. If the row and column index sets match the *base* sets, then the matrix expressions of the two m-arrays are equivalent. (Alternatively, index exchange and transfer identities may be applied.)

As an example, identity (2–45) will be proved by means of array algebra.

*Proof:* Let $\boldsymbol{G} = (\boldsymbol{A} \star \boldsymbol{B})(\boldsymbol{C} \star \boldsymbol{D})$. Then,

$$
\begin{aligned}
\underset{pm\times kt}{\boldsymbol{G}} &= (\underset{p\times q}{\boldsymbol{A}} \star \underset{m\times n}{\boldsymbol{B}})(\underset{n\times k}{\boldsymbol{C}} \star \underset{q\times t}{\boldsymbol{D}}) \\
&\Updownarrow \\
\mathcal{G}\big|^{ij}_{gh} &:= (\boldsymbol{A}\star\boldsymbol{B})\big|^{ij}_{xy}\,(\boldsymbol{C}\star\boldsymbol{D})\big|^{xy}_{gh} \\
&:= \mathcal{A}\big|^{j}_{x}\,\mathcal{B}\big|^{i}_{y}\,\mathcal{C}\big|^{y}_{g}\,\mathcal{D}\big|^{x}_{h} \\
&= \mathcal{B}\big|^{i}_{y}\,\mathcal{C}\big|^{y}_{g}\,\mathcal{A}\big|^{j}_{x}\,\mathcal{D}\big|^{x}_{h} \\
&:= (\boldsymbol{BC})\big|^{i}_{g}\,(\boldsymbol{AD})\big|^{j}_{h} \\
&:= \Big[(\boldsymbol{BC})\otimes(\boldsymbol{AD})\Big]\Big|^{ij}_{gh} \\
&\Updownarrow \\
\underset{pm\times kt}{\boldsymbol{G}} &= (\underset{m\times n}{\boldsymbol{B}}\ \underset{n\times k}{\boldsymbol{C}})\otimes(\underset{p\times q}{\boldsymbol{A}}\ \underset{q\times t}{\boldsymbol{D}})
\end{aligned}
$$

$$
\Longrightarrow \boxed{(\underset{p\times q}{\boldsymbol{A}} \star \underset{m\times n}{\boldsymbol{B}})(\underset{n\times k}{\boldsymbol{C}} \star \underset{q\times t}{\boldsymbol{D}}) = (\underset{m\times n}{\boldsymbol{B}}\ \underset{n\times k}{\boldsymbol{C}})\otimes(\underset{p\times q}{\boldsymbol{A}}\ \underset{q\times t}{\boldsymbol{D}})}
$$

□

Note that the scalar nature of of arrays in the array domain allows the rearrangement of terms in the fourth line of the proof.

The advantage of using array algebra is more dramatically illustrated in Appendix A where a comparison is made between an array algebra proof and a matrix algebra proof.

# Chapter 3

# Processing Flows of Isoplanar Homosyndetic Processors

When speed and minimal form factor are of the essence, then customized hardware is most likely the only solution. In order to ease the design, fabrication, maintenance and expandability of specialized architectures, a certain degree of modularity and function replication is often desirable. With these goals in mind, a special class of multistage architectures is examined in this chapter—one that is characterized by:

- parallel data input and output (either one- or two-dimensional),
- identical processing/interconnection stages,
- synchronous operation.

The conceptual structure of this class, is noted as *isoplanar* and *homosyndetic*. This type of multistage structure affords the flexibility of trading off speed for hardware savings. Since all the processing and communication stages are identical, a task that may require, for example, $n$ pipelined stages to complete, may instead be recycled through the same stage $n$ times.

Two-dimensional isoplanar homosyndetic structures have been examined extensively in the past, mainly, from the perspective of interconnection networks (e.g., *shuffle/exchange networks*)[38, 55, 56]. Fixed interconnection stages route data to exchange

boxes which merely switch between two or more transmission paths. Adding computational sophistication to the exchange boxes has not been considered beyond that required to effect a Fast Fourier Transform (FFT).

This chapter presents a mathematical characterization of 2-D as well as 3-D isoplanar homosyndetic architectures. The mathematical expression from which a structure is directly derived is referred to as its *canonical form.*

## 3.1 Shuffled 1-D Processing Flow

### 3.1.1 Characterization of an $N$-Parallel Processing Stage

The simultaneous processing of $N$ units of information can be represented from a matrix algebra point of view as a transformation of an input vector $x$ into an output vector $y$—each of arbitrary length. One of the most basic vector operations is a linear transformation, where each element of the output vector is a weighted sum of all the input elements,

$$y_i = \sum_j h_{ij}\, x_j$$

or in matrix notation,

$$\underset{m}{\boldsymbol{y}} = \underset{m \times n}{\boldsymbol{H}} \cdot \underset{n}{\boldsymbol{x}}.$$

A parallel hardware implementation of this equation is presented in Fig. 3.1. It requires $mn$ multipliers and $m$ devices which can perform an $n$-component sum or equivalently a total of $m(n-1)$ adders. The problem with such an implementation is that it is a *brute-force* approach which does not take advantage of any regularity or sparcity that may be exhibited by the transformation matrix $\boldsymbol{H}$ . Judiciously using such information can reduce the hardware requirements considerably.

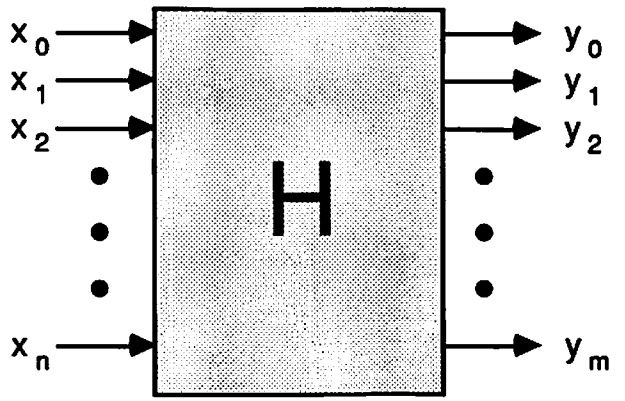



Figure 3.1: Implementation of a general linear operation, $\underset{m}{\boldsymbol{y}} = \underset{mxn}{\boldsymbol{H}}\ \underset{n}{\boldsymbol{x}}$.

### Block Diagonal Transformations

Consider a special case of sparcity of the transformation matrix such that it is block diagonal. The simplest form involves identical diagonal blocks, such that,

$$\underset{mxn}{\boldsymbol{H}} = \begin{bmatrix} \underset{kxq}{\boldsymbol{K}} & & & \\ & \underset{kxq}{\boldsymbol{K}} & & \\ & & \ddots & \\ & & & \underset{kxq}{\boldsymbol{K}} \end{bmatrix} \tag{3-1}$$

where $m/k = n/q = p$. Such a replication of a *kernel block* $\boldsymbol{K}$ along the diagonal can be concisely expressed by means of Kronecker algebra,

$$\underset{mxn}{\boldsymbol{H}} = \underset{p}{\boldsymbol{I}} \otimes \underset{kxq}{\boldsymbol{K}} \tag{3-2}$$

Implementation of $\underset{m}{\boldsymbol{y}} = (\underset{n/q}{\boldsymbol{I}} \otimes \underset{kxq}{\boldsymbol{K}})\underset{n}{\boldsymbol{x}}$ is equivalent to replicating a kernel box so that it operates on disjoint segments of the input vector as shown in Fig. 3.2. This results in a *parallel processing stage of p identical q-input/k-output kernels.* The simple Kronecker expression of such a stage, as given by (3–2) , is called a *normal factor.*

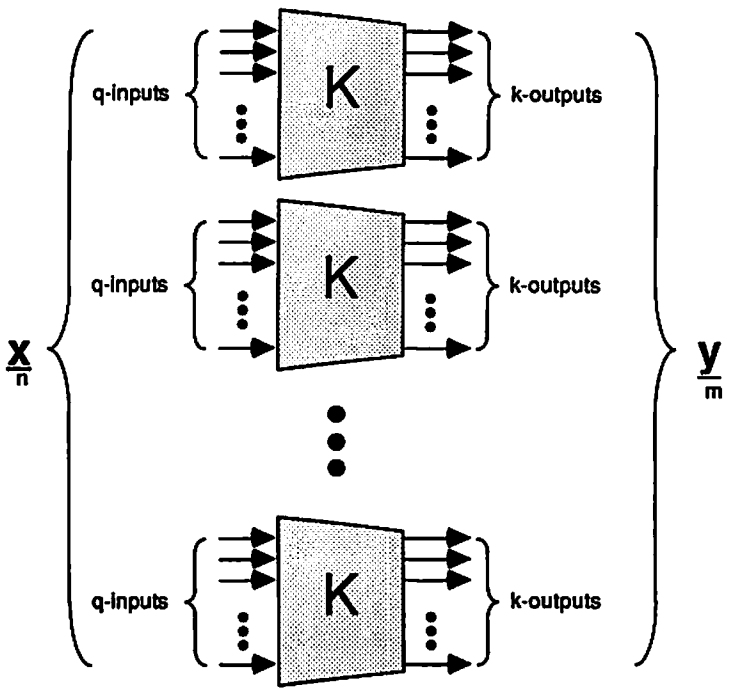



Figure 3.2: Implementation of $\underset{m}{y} = (\underset{n/q}{I} \otimes \underset{k \times q}{K})\underset{n}{x}$

Of course, the kernels of a processing stage need not have identical weights, e.g.,

$$\underset{m \times n}{H'} = \begin{bmatrix} \underset{k \times q}{K_0} & & & \\ & \underset{k \times q}{K_1} & & \\ & & \ddots & \\ & & & \underset{k \times q}{K_{p-1}} \end{bmatrix} \tag{3-3}$$

where $m/k = n/q$. The Kronecker representation of an isoplanar stage of arbitrarily weighted kernels is a bit more complicated

$$\underset{m \times n}{H'} = \sum_{j=0}^{p-1} \underset{p \times p}{U^{(j,j)}} \otimes \underset{k \times q}{K_j}. \tag{3-4}$$

DEFINITION 21: *A processing stage which consists of blocks of identical computational structure—but arbitrary weights—is called* isonode.

DEFINITION 22: *A processing stage which consists of blocks of identical computational structure and kernel weights is called* isokernel.

Obviously, *isokernel* $\subset$ *isonode.*

The implementation of a processing stage of $p$ *q-input/k-output* kernels requires a total of $mk$ multipliers and $m(k-1)$ adders. The greatest hardware reduction over the general implementation is achieved when $k$ and $q$ equal the smallest prime component of $m$ and $n$ respectively. For example, an implementation with $m = 6, n = 9, k = 2$ and $q = 3$ is shown in Fig. 3.3.

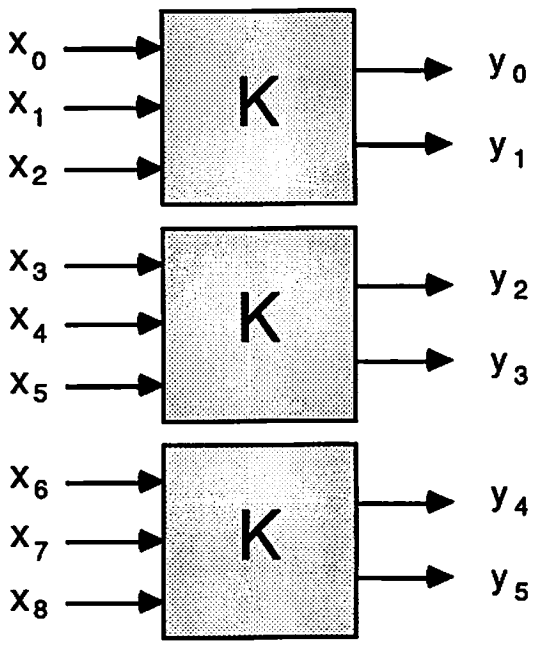



Figure 3.3: Implementation of $\underset{6}{\boldsymbol{y}} = (\underset{3}{\boldsymbol{I}} \otimes \underset{2\times 3}{\boldsymbol{K}})\underset{9}{\boldsymbol{x}}$

### 3.1.2 1-D Omega Processor

There are many mathematical tasks that exhibit a fast $N$-parallel implementation. The typical amount of computation required is on the order of $\log_2 N$ vector processing

stages. Consequently, $N$-parallel networks which consist of $\log_2 N$ stages have been given special names. The structure formed by cascading $\log_2 N$ shuffle/exchange stages is called an *Omega Network* and has been depicted in Fig. 1.7. The Omega network is one of the fundamental isoplanar homosyndetic communication structures as its inverse is isomorphic [38] to other heterosyndetic networks such as the *indirect binary n-cube* (hypercube) and the R-network.

The functionality of the inverse Omega network may be extended by replacing the exchange boxes with processing elements (PEs), resulting in an inverse *2-D Omega Processor* as shown in Fig. 3.4. The sophistication of the processing elements

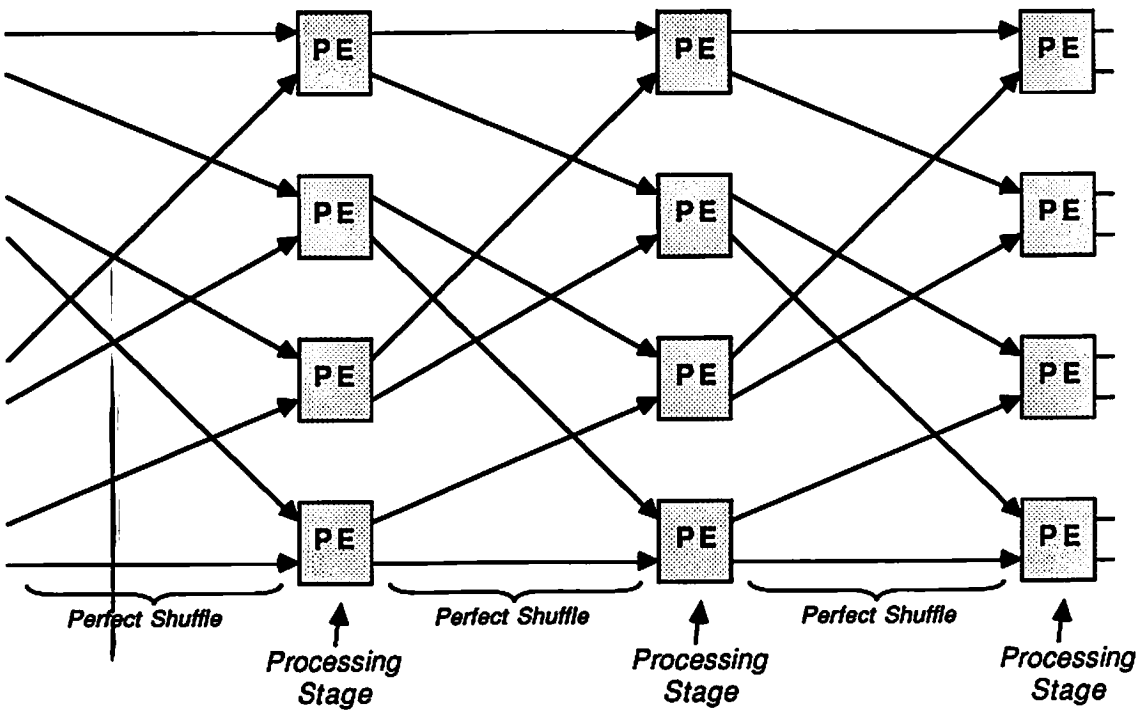



Figure 3.4: Example of a one-dimensional Omega Processor (N=8)

may range from a very fine granularity (e.g., hardwired multipliers and summers) to a microprocessor-level coarseness (e.g., as is found in present day hypercube implementations).

In terms of the mathematical framework presented in Chapter 2, the granularity of interest will be one that implements at least a general small size matrix-vector multiplication. In the case of a 2-input/2-output PE, this can be accomplished with just two multipliers and two adders. More computational capability could be added by making the computing elements general purpose programmable Arithmetic Logic Units (ALUs) as illustrated in Fig. 3.5. A small amount of local memory (RAM) would be required for storing kernel coefficients which may vary from PE to PE. Of course, speed could

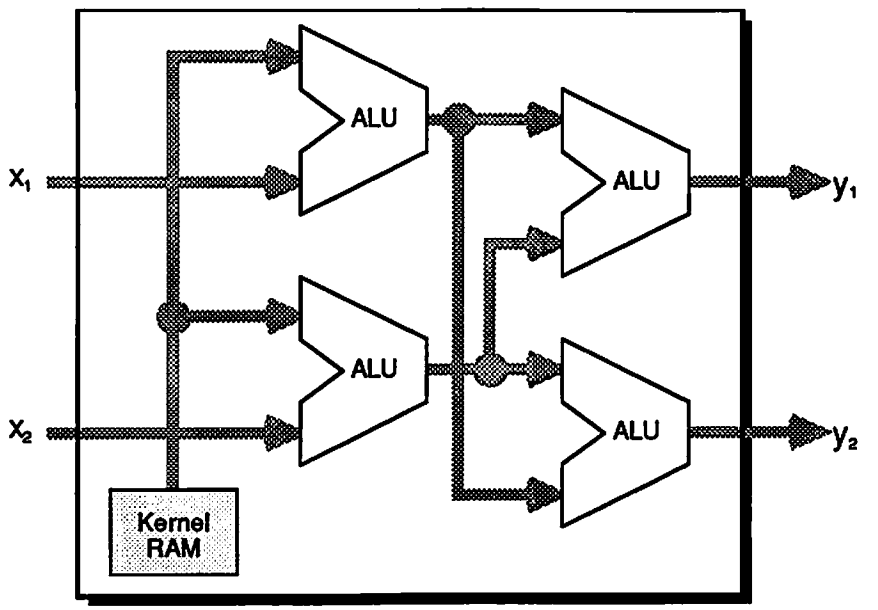



Figure 3.5: 2-input/2-output PE

be traded off for computing hardware by using only one ALU and enough *control store* memory and logic to sequence through a small program that would virtually effect the required PE task. This latter course would also increase the local memory requirement as intermediate results would need to be stored in RAM.

The mathematical representation of a Omega processor is easily derived from the previously examined representations of shuffles and isoplanar processing stages. Namely, a *canonical form* can be derived which can be used as a mathematical *target* when mapping a given task to the 1-D Omega processing flow. The values of the kernel

matrices, $\underset{2\times2}{\boldsymbol{K}}$ , determine the complexity of the canonical form as demonstrated in the following corollaries.

COROLLARY 17 *(1-D General Omega Canonical Form)*: *The canonical form of an N-element (1-D) Omega processing flow using arbitrary 2-input/2-output kernels is*

$$\boldsymbol{\Omega} = \prod_{i=0}^{\log_2 N} \left[ \left( \sum_{j=0}^{\frac{N}{2}-1} \underset{\frac{N}{2}\times\frac{N}{2}}{\boldsymbol{U}}^{(j,j)} \otimes \underset{2\times2}{\boldsymbol{K}_{ij}} \right) \boldsymbol{S}_N \right] \tag{3-5}$$

*where* $\underset{2\times2}{\boldsymbol{K}_{ij}}$ *is the* $j^{th}$ *kernel of the* $i^{th}$ *stage.*

COROLLARY 18 *(1-D Isonode Omega Canonical Form)*: *The canonical form of an N-element Omega processing flow using identically valued 2-input/2-output kernels within each stage is*

$$\boldsymbol{\Omega} = \prod_{i=0}^{\log_2 N} \left[ \left( \underset{N/2}{\boldsymbol{I}} \otimes \underset{2\times2}{\boldsymbol{K}_i} \right) \boldsymbol{S}_N \right] \tag{3-6}$$

COROLLARY 19 *(1-D Isokernel Omega Canonical Form)*: *The canonical form of an N-element Omega processing flow using globally identically valued 2-input/2-output kernels is*

$$\boldsymbol{\Omega} = \left[ \left( \underset{N/2}{\boldsymbol{I}} \otimes \underset{2\times2}{\boldsymbol{K}} \right) \boldsymbol{S}_N \right]^{\log_2 N} \tag{3-7}$$

**Inverse Omega Processor** Reversing the the omega processing flow results in the *inverse Omega processor.* Inverse Omega canonical forms are summarized as follows.

COROLLARY 20 *(1-D General Inverse Omega Canonical Form)*:

$$\boldsymbol{\Omega}^{-1} = \prod_{i=0}^{\log_2 N} \left[ \boldsymbol{S}_N^{-1} \left( \sum_{j=0}^{\frac{N}{2}-1} \underset{\frac{N}{2}\times\frac{N}{2}}{\boldsymbol{U}}^{(j,j)} \otimes \underset{2\times2}{\boldsymbol{K}_{ij}} \right) \right] \tag{3-8}$$

*where* $\underset{2\times2}{\boldsymbol{K}_{ij}}$ *is the* $j^{th}$ *kernel of the* $i^{th}$ *stage.*

COROLLARY 21 *(1-D Isonode 1-D Inverse Omega Canonical Form)*:

$$\Omega^{-1} = \prod_{i=0}^{\log_2 N} \left[ S_N^{-1} \left( \underset{N/2}{I} \otimes \underset{2\times 2}{K_i} \right) \right] \tag{3-9}$$

COROLLARY 22 *(1-D Isokernel Inverse Omega Canonical Form)*:

$$\Omega^{-1} = \left[ S_N^{-1} \left( \underset{N/2}{I} \otimes \underset{2\times 2}{K} \right) \right]^{\log_2 N} \tag{3-10}$$

### 3.1.3 General Isoplanar Homosyndetic Processors

Canonical forms can be generated for any type of general isoplanar homosyndetic architecture. The identical interconnection stages are represented by some permutation matrix $P$ and isonode processing stages represented by the Kronecker product $\underset{p}{I} \otimes \underset{k\times q}{K}$. Thus, the canonical form of an $r$-stage isoplanar homosyndetic architecture may be defined as

$$\prod_{i=0}^{r-1} \underset{pk\times pq}{C_i} \tag{3-11}$$

where either

$$\underset{pk\times pq}{C_i} = P \left( \underset{p}{I} \otimes \underset{k\times q}{K_i} \right) \tag{3-12}$$

or

$$\underset{pk\times pq}{C_i} = \left( \underset{p}{I} \otimes \underset{k\times q}{K_i} \right) P \tag{3-13}$$

depending on whether the interconnection leads or follows in the permute/process stage pair.

## 3.2 Shuffled 2-D Processing Flow

### 3.2.1 Characterization of an $m \times n$-Parallel Processing Stage

Consider a sampled image $x(i,j)$ with $i = 0, 1, \ldots, m-1$ and $j = 0, 1, \ldots, n-1$. It is represented most naturally by an $m \times n$ matrix, $X$. A two-dimensional linear operation on the image $x(i,j)$ is defined as:

$$y(p,q) = \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} H(p,q;i,j)x(i,j) \tag{3-14}$$

Can this operation be represented by *matrix-space* notation? No, not in the general case. Matrix-space representations are only possible when the operation $\boldsymbol{H}$ is *separable*, i.e, it can be decomposed into a transformation of the rows followed by a transformation of the columns (or vice-versa). If the separate transformations are represented by two matrices, $\boldsymbol{H}_r$ and $\boldsymbol{H}_c$, respectively, then:

$$\boldsymbol{Y} = \boldsymbol{H}_c \cdot \boldsymbol{X} \cdot \boldsymbol{H}_r^T \tag{3-15}$$

Consider now representing the image $x(m,n)$ as a vector instead of a matrix. Such a vector may be formed by either row or column scanning of $\boldsymbol{X}$, i.e.:

$$\boldsymbol{x} = \text{vec}(\boldsymbol{X}) = \begin{cases} \text{vec}_r(\boldsymbol{X}) & \text{if} \quad \text{row} - \text{scanning} \\ \text{vec}_c(\boldsymbol{X}) & \text{if} \quad \text{column} - \text{scanning} \end{cases}$$

This vector-space representation allows (3–14) to be stated as:

$$\text{vec}(\boldsymbol{Y}) = \boldsymbol{H} \cdot \text{vec}(\boldsymbol{X}) \tag{3-16}$$

Note that by substituting (3–15) into (3–16) and applying (2–41) or (2–42) accordingly, it can be easily shown that in the separable case:

$$\boldsymbol{H} = \begin{cases} \boldsymbol{H}_r \otimes \boldsymbol{H}_c & \text{if} \quad \text{vec}(\cdot) \equiv \text{vec}_c(\cdot) \\ \boldsymbol{H}_c^T \otimes \boldsymbol{H}_r^T & \text{if} \quad \text{vec}(\cdot) \equiv \text{vec}_r(\cdot) \end{cases}$$

### 2-D Normal Factors

One would hope that a two-dimensional array of identical exchange or processing boxes would have a representation similar to that of normal factors in the one-dimensional case. Consider the case where each processing box is applied to a $2 \times 2$ matrix of input values represented by $\underset{2\times2}{\boldsymbol{X}}$, and outputs a corresponding $2 \times 2$ matrix, $\underset{2\times2}{\boldsymbol{Y}}$. The linear operation executed in each processing block is represented by the matrix $\underset{4\times4}{\boldsymbol{K}}$. In vector-space notation, this is represented as:

$$\text{vec}\left(\underset{2\times2}{\boldsymbol{Y}}\right) = \underset{4\times4}{\boldsymbol{K}} \cdot \text{vec}\left(\underset{2\times2}{\boldsymbol{X}}\right) \tag{3-17}$$

Imagine now, that the matrix $\boldsymbol{X}$ is of size $m \times n$ (where $m, n$ are multiples of 2) and subject to the same $2 \times 2$ processing window. This is conceptually illustrated in Fig. 3.6.

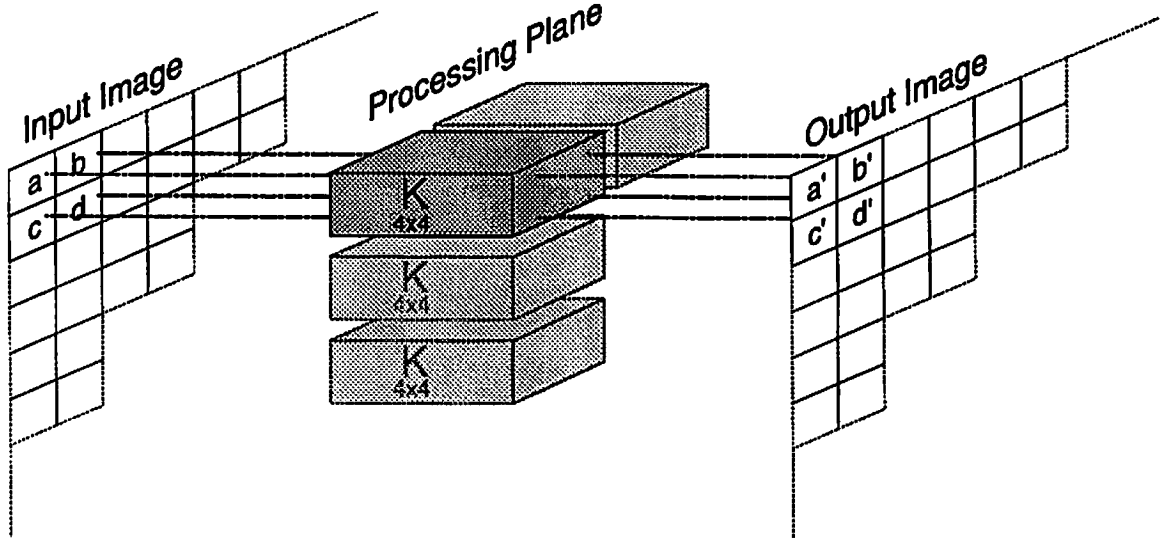



Figure 3.6: Isokernel Processing of $2 \times 2$ tiles

By partitioning the input matrix, $\boldsymbol{X}$, into submatrices $\boldsymbol{X}_{ij}$ of size $2 \times 2$, we may write:

$$\begin{bmatrix} \mathrm{vec}(\boldsymbol{Y}_{11}) & \mathrm{vec}(\boldsymbol{Y}_{12}) & \cdots & \mathrm{vec}\left(\boldsymbol{Y}_{1\frac{n}{2}}\right) \\ \mathrm{vec}(\boldsymbol{Y}_{21}) & \mathrm{vec}(\boldsymbol{Y}_{22}) & \cdots & \mathrm{vec}\left(\boldsymbol{Y}_{2\frac{n}{2}}\right) \\ \vdots & \vdots & \vdots & \vdots \\ \mathrm{vec}\left(\boldsymbol{Y}_{\frac{m}{2}1}\right) & \mathrm{vec}\left(\boldsymbol{Y}_{\frac{m}{2}2}\right) & \cdots & \mathrm{vec}\left(\boldsymbol{Y}_{\frac{m}{2}\frac{n}{2}}\right) \end{bmatrix}_{2m \times \frac{n}{2}} =$$

$$\begin{bmatrix} \underset{4\times4}{\boldsymbol{K}} & & & \\ & \underset{4\times4}{\boldsymbol{K}} & & \\ & & \ddots & \\ & & & \underset{4\times4}{\boldsymbol{K}} \end{bmatrix} \begin{bmatrix} \mathrm{vec}(\boldsymbol{X}_{11}) & \mathrm{vec}(\boldsymbol{X}_{12}) & \cdots & \mathrm{vec}\left(\boldsymbol{X}_{1\frac{n}{2}}\right) \\ \mathrm{vec}(\boldsymbol{X}_{21}) & \mathrm{vec}(\boldsymbol{X}_{22}) & \cdots & \mathrm{vec}\left(\boldsymbol{X}_{2\frac{n}{2}}\right) \\ \vdots & \vdots & \vdots & \vdots \\ \mathrm{vec}\left(\boldsymbol{X}_{\frac{m}{2}1}\right) & \mathrm{vec}\left(\boldsymbol{X}_{\frac{m}{2}2}\right) & \cdots & \mathrm{vec}\left(\boldsymbol{X}_{\frac{m}{2}\frac{n}{2}}\right) \end{bmatrix}_{2m \times \frac{n}{2}}$$

$$\Longrightarrow [\mathrm{vec}(\boldsymbol{Y}_{ij})] = \left(\underset{\frac{m}{2}}{\boldsymbol{I}} \otimes \underset{4\times4}{\boldsymbol{K}}\right) [\mathrm{vec}(\boldsymbol{X}_{ij})] \qquad (3\text{–}18)$$

Note that the input and output images are expressed in a "partitioned-matrix space" of size $2m \times \frac{n}{2}$. By vectorizing both sides of (3–18), the images are converted into a

permuted vector-space:

$$\begin{bmatrix} \mathrm{vec}(\boldsymbol{Y}_{11}) \\ \cdots \\ \mathrm{vec}\left(\boldsymbol{Y}_{\frac{m}{2}\frac{n}{2}}\right) \end{bmatrix}_{mn\times 1} = \begin{bmatrix} \underset{4\times 4}{\boldsymbol{K}} & & \\ & \ddots & \\ & & \underset{4\times 4}{\boldsymbol{K}} \end{bmatrix} \begin{bmatrix} \mathrm{vec}(\boldsymbol{X}_{11}) \\ \cdots \\ \mathrm{vec}\left(\boldsymbol{X}_{\frac{m}{2}\frac{n}{2}}\right) \end{bmatrix}_{mn\times 1}$$

$$\Longrightarrow \mathrm{vec}([\mathrm{vec}(\boldsymbol{Y}_{ij})]) = (\underset{mn/4}{\boldsymbol{I}} \otimes \underset{4\times 4}{\boldsymbol{K}})\mathrm{vec}([\mathrm{vec}(\boldsymbol{X}_{ij})]) \tag{3-19}$$

This expression, of course, is identical to that of the one-dimensional case except that the input and output vectors are permuted versions of vectorized matrices.

In order to derive the exact permutation, let's look at the following expansion by using the definition of column vectorization, (2-31):

$$\begin{aligned}
\mathrm{vec_c}([\mathrm{vec_c}(\boldsymbol{X}_{ij})]) &= \sum_{a=0}^{\frac{n}{2}-1} \underset{\frac{n}{2}}{\boldsymbol{u}}^{(a)} \otimes \left([\mathrm{vec_c}(\boldsymbol{X}_{ij})] \cdot \underset{\frac{n}{2}}{\boldsymbol{u}}^{(a)}\right) \\
&= \sum_{a=0}^{\frac{n}{2}-1} \underset{\frac{n}{2}}{\boldsymbol{u}}^{(a)} \otimes \left(\left(\sum_{b=0}^{\frac{m}{2}-1}\sum_{c=0}^{\frac{n}{2}-1} \underset{\frac{m}{2}\times\frac{n}{2}}{\boldsymbol{U}}^{(b,c)} \otimes \mathrm{vec_c}(\boldsymbol{X}_{bc})\right) \cdot \underset{\frac{n}{2}}{\boldsymbol{u}}^{(a)}\right) \\
&= \sum_{a=0}^{\frac{n}{2}-1}\sum_{b=0}^{\frac{m}{2}-1}\sum_{c=0}^{\frac{n}{2}-1} \underset{\frac{n}{2}}{\boldsymbol{u}}^{(a)} \otimes \left((\underset{\frac{m}{2}\times\frac{n}{2}}{\boldsymbol{U}}^{(b,c)} \cdot \underset{\frac{n}{2}}{\boldsymbol{u}}^{(a)}) \otimes \mathrm{vec_c}(\boldsymbol{X}_{bc})\right) \\
&= \sum_{b=0}^{\frac{m}{2}-1}\sum_{a=0}^{\frac{n}{2}-1}\sum_{c=0}^{\frac{n}{2}-1} \underset{\frac{n}{2}}{\boldsymbol{u}}^{(a)} \otimes \left((\underset{\frac{m}{2}}{\boldsymbol{u}}^{(b)} \cdot \underset{\frac{n}{2}}{\boldsymbol{u}}^{(c)^T} \cdot \underset{\frac{n}{2}}{\boldsymbol{u}}^{(a)}) \otimes \mathrm{vec_c}(\boldsymbol{X}_{bc})\right) \\
&= \sum_{b=0}^{\frac{m}{2}-1}\sum_{c=0}^{\frac{n}{2}-1} \underset{\frac{n}{2}}{\boldsymbol{u}}^{(c)} \otimes \underset{\frac{m}{2}}{\boldsymbol{u}}^{(b)} \otimes \mathrm{vec_c}(\boldsymbol{X}_{bc})
\end{aligned}$$

But, by the submatrix extraction definition, (2-30),

$$\begin{aligned}
\mathrm{vec_c}(\boldsymbol{X}_{bc}) &= \mathrm{vec_c}\left((\underset{\frac{m}{2}}{\boldsymbol{u}}^{(b)^T} \otimes \underset{2}{\boldsymbol{I}}) \underset{m\times n}{\boldsymbol{X}} (\underset{\frac{n}{2}}{\boldsymbol{u}}^{(c)} \otimes \underset{2}{\boldsymbol{I}})\right) \\
\textit{[by (2-41)]} \qquad &= \left((\underset{\frac{n}{2}}{\boldsymbol{u}}^{(c)} \otimes \underset{2}{\boldsymbol{I}})^T \otimes (\underset{\frac{m}{2}}{\boldsymbol{u}}^{(b)^T} \otimes \underset{2}{\boldsymbol{I}})\right) \mathrm{vec_c}(\boldsymbol{X}) \\
&= \left((\underset{\frac{n}{2}}{\boldsymbol{u}}^{(c)^T} \otimes \underset{2}{\boldsymbol{I}}) \otimes (\underset{\frac{m}{2}}{\boldsymbol{u}}^{(b)^T} \otimes \underset{2}{\boldsymbol{I}})\right) \mathrm{vec_c}(\boldsymbol{X}) \\
&= \left(\underset{\frac{n}{2}}{\boldsymbol{u}}^{(c)^T} \otimes \underset{2}{\boldsymbol{I}} \otimes \underset{\frac{m}{2}}{\boldsymbol{u}}^{(b)^T} \otimes \underset{2}{\boldsymbol{I}}\right) \mathrm{vec_c}(\boldsymbol{X})
\end{aligned}$$

Therefore,

$$
\begin{aligned}
\mathrm{vec_c}([\mathrm{vec_c}(\boldsymbol{X}_{ij})]) &= \sum_{b=0}^{\frac{m}{2}-1}\sum_{c=0}^{\frac{n}{2}-1} \underset{\frac{n}{2}}{\boldsymbol{u}}^{(c)} \otimes \underset{\frac{m}{2}}{\boldsymbol{u}}^{(b)} \otimes \left( \underset{\frac{n}{2}}{\boldsymbol{u}}^{(c)^T} \otimes \underset{2}{\boldsymbol{I}} \otimes \underset{\frac{m}{2}}{\boldsymbol{u}}^{(b)^T} \otimes \underset{2}{\boldsymbol{I}} \right) \mathrm{vec_c}(\boldsymbol{X}) \\
\textit{[by (2-18)]} &= \sum_{b=0}^{\frac{m}{2}-1}\sum_{c=0}^{\frac{n}{2}-1} \left( \underset{\frac{n}{2}}{\boldsymbol{u}}^{(c)} \otimes \underset{\frac{n}{2}}{\boldsymbol{u}}^{(c)^T} \otimes \underset{\frac{m}{2}}{\boldsymbol{u}}^{(b)} \otimes \underset{2}{\boldsymbol{I}} \otimes \underset{\frac{m}{2}}{\boldsymbol{u}}^{(b)^T} \otimes \underset{2}{\boldsymbol{I}} \right) \mathrm{vec_c}(\boldsymbol{X}) \\
\textit{[by (2-23)]} &= \sum_{b=0}^{\frac{m}{2}-1}\sum_{c=0}^{\frac{n}{2}-1} \left( \underset{\frac{n}{2}\times\frac{n}{2}}{\boldsymbol{U}}^{(c,c)} \otimes \underset{\frac{m}{2}}{\boldsymbol{u}}^{(b)} \otimes ( \underset{\frac{m}{2}}{\boldsymbol{u}}^{(b)^T} \otimes \underset{2}{\boldsymbol{I}}) \boldsymbol{S}_m \otimes \underset{2}{\boldsymbol{I}} \right) \mathrm{vec_c}(\boldsymbol{X}) \\
&= \sum_{b=0}^{\frac{m}{2}-1} \left( \sum_{c=0}^{\frac{n}{2}-1} \underset{\frac{n}{2}\times\frac{n}{2}}{\boldsymbol{U}}^{(c,c)} \otimes ( \underset{\frac{m}{2}}{\boldsymbol{u}}^{(b)} \otimes \underset{\frac{m}{2}}{\boldsymbol{u}}^{(b)^T} \otimes \underset{2}{\boldsymbol{I}}) \boldsymbol{S}_m \otimes \underset{2}{\boldsymbol{I}} \right) \mathrm{vec_c}(\boldsymbol{X}) \\
&= \sum_{b=0}^{\frac{m}{2}-1} \left( \underset{\frac{n}{2}}{\boldsymbol{I}} \otimes ( \underset{\frac{m}{2}\times\frac{m}{2}}{\boldsymbol{U}}^{(b,b)} \otimes \underset{2}{\boldsymbol{I}}) \boldsymbol{S}_m \otimes \underset{2}{\boldsymbol{I}} \right) \mathrm{vec_c}(\boldsymbol{X}) \\
&= \left( \underset{\frac{n}{2}}{\boldsymbol{I}} \otimes ( \sum_{b=0}^{\frac{m}{2}-1} \underset{\frac{m}{2}\times\frac{m}{2}}{\boldsymbol{U}}^{(b,b)} \otimes \underset{2}{\boldsymbol{I}}) \boldsymbol{S}_m \otimes \underset{2}{\boldsymbol{I}} \right) \mathrm{vec_c}(\boldsymbol{X}) \\
&= \left( \underset{\frac{n}{2}}{\boldsymbol{I}} \otimes ( \underset{\frac{m}{2}}{\boldsymbol{I}} \otimes \underset{2}{\boldsymbol{I}}) \boldsymbol{S}_m \otimes \underset{2}{\boldsymbol{I}} \right) \mathrm{vec_c}(\boldsymbol{X}) \\
&= ( \underset{\frac{n}{2}}{\boldsymbol{I}} \otimes \boldsymbol{S}_m \otimes \underset{2}{\boldsymbol{I}}) \mathrm{vec_c}(\boldsymbol{X}).
\end{aligned}
$$

Consequently, we may rewrite (3–19) as

$$
\begin{aligned}
( \underset{\frac{n}{2}}{\boldsymbol{I}} \otimes \boldsymbol{S}_m \otimes \underset{2}{\boldsymbol{I}}) \, \mathrm{vec_c}(\boldsymbol{Y}) &= ( \underset{\frac{mn}{4}}{\boldsymbol{I}} \otimes \underset{4\times4}{\boldsymbol{K}}) ( \underset{\frac{n}{2}}{\boldsymbol{I}} \otimes \boldsymbol{S}_m \otimes \underset{2}{\boldsymbol{I}}) \mathrm{vec_c}(\boldsymbol{X}) \\
\Longrightarrow \mathrm{vec_c}(\boldsymbol{Y}) &= ( \underset{\frac{n}{2}}{\boldsymbol{I}} \otimes \boldsymbol{S}_m \otimes \underset{2}{\boldsymbol{I}})^{-1} ( \underset{\frac{mn}{4}}{\boldsymbol{I}} \otimes \underset{4\times4}{\boldsymbol{K}}) ( \underset{\frac{n}{2}}{\boldsymbol{I}} \otimes \boldsymbol{S}_m \otimes \underset{2}{\boldsymbol{I}}) \, \mathrm{vec_c}(\boldsymbol{X}) \\
&= ( \underset{\frac{n}{2}}{\boldsymbol{I}} \otimes \boldsymbol{S}_m^{-1} \otimes \underset{2}{\boldsymbol{I}}) ( \underset{\frac{mn}{4}}{\boldsymbol{I}} \otimes \underset{4\times4}{\boldsymbol{K}}) ( \underset{\frac{n}{2}}{\boldsymbol{I}} \otimes \boldsymbol{S}_m \otimes \underset{2}{\boldsymbol{I}}) \, \mathrm{vec_c}(\boldsymbol{X}) \\
&= \left[ \underset{\frac{n}{2}}{\boldsymbol{I}} \otimes \left( (\boldsymbol{S}_m^{-1} \otimes \underset{2}{\boldsymbol{I}}) ( \underset{\frac{m}{2}}{\boldsymbol{I}} \otimes \underset{4\times4}{\boldsymbol{K}}) (\boldsymbol{S}_m \otimes \underset{2}{\boldsymbol{I}}) \right) \right] \mathrm{vec_c}(\boldsymbol{X})
\end{aligned}
$$

$$
\Longrightarrow \mathrm{vec_c}(\boldsymbol{Y}) = {}^2\boldsymbol{H}_c \cdot \mathrm{vec_c}(\boldsymbol{X}) \tag{3–20}
$$

where

$$
\boxed{ {}^2\boldsymbol{H}_c = \underset{\frac{n}{2}}{\boldsymbol{I}} \otimes \left( (\boldsymbol{S}_m^{-1} \otimes \underset{2}{\boldsymbol{I}}) ( \underset{\frac{m}{2}}{\boldsymbol{I}} \otimes \underset{4\times4}{\boldsymbol{K}}) (\boldsymbol{S}_m \otimes \underset{2}{\boldsymbol{I}}) \right) } \tag{3–21}
$$

It is evident that the 2-D processing stage representation expressed in (3–21) is not as elegant or as intuitive as the analogous representation of a 1-D processing stage, i.e.,

$$\boldsymbol{H} = \underset{\frac{n}{2}}{\boldsymbol{I}} \otimes \underset{2\times 2}{\boldsymbol{K}}.$$

However, it does bear some similarity to the 1-D normal factor in that there is a

$$\underset{\frac{n}{2}}{\boldsymbol{I}} \otimes \textit{something}$$

structure. This also means that a 1-D processing stage of $n/2$ nodes executing

$$\left[(\boldsymbol{S}_m^{-1} \otimes \underset{2}{\boldsymbol{I}})(\underset{\frac{m}{2}}{\boldsymbol{I}} \otimes \underset{4\times 4}{\boldsymbol{K}})(\boldsymbol{S}_m \otimes \underset{2}{\boldsymbol{I}})\right]_{2m\times 2m}$$

kernels can be used to simulate a 2-D processing stage of $\underset{4\times 4}{\boldsymbol{K}}$ kernels provided, of course, that the input and output matrices have been column-scanned.

An analogous 2-D normal factor for row-scanning may be obtained,

$$\boxed{{}^{2}\boldsymbol{H}_r = \underset{\frac{m}{2}}{\boldsymbol{I}} \otimes \left((\boldsymbol{S}_n^{-1} \otimes \underset{2}{\boldsymbol{I}})(\underset{\frac{n}{2}}{\boldsymbol{I}} \otimes \underset{4\times 4}{\boldsymbol{K}_\Gamma})(\boldsymbol{S}_n \otimes \underset{2}{\boldsymbol{I}})\right)} \tag{3–22}$$

for

$$\mathrm{vec_r}(\boldsymbol{Y}) = {}^{2}\boldsymbol{H}_r \cdot \mathrm{vec_r}(\boldsymbol{X}) \tag{3–23}$$

and

$$\mathrm{vec_r}\left(\boldsymbol{Y}_{ij}\right) = \underset{4\times 4}{\boldsymbol{K}_\Gamma} \cdot \mathrm{vec_r}\left(\boldsymbol{X}_{ij}\right) \tag{3–24}$$

Admittedly, the derivations of (3–21) and (3–22) are cumbersome and canonical forms of other types of 2-D processing planes would be just as laborious.

### Array Algebra Representation

Consider, however, an array algebra representation of the described 2-D processing stage. Such a representation can be derived by first rewriting (3–21) as

$$\begin{aligned} {}^{2}\boldsymbol{H} &= (\underset{\frac{n}{2}}{\boldsymbol{I}} \otimes \boldsymbol{S}_m^{-1} \otimes \underset{2}{\boldsymbol{I}})(\underset{\frac{mn}{4}}{\boldsymbol{I}} \otimes \underset{4\times 4}{\boldsymbol{K}})(\underset{\frac{n}{2}}{\boldsymbol{I}} \otimes \boldsymbol{S}_m \otimes \underset{2}{\boldsymbol{I}}) \\ &= (\underset{\frac{n}{2}}{\boldsymbol{I}} \otimes (\underset{\frac{m}{2}}{\boldsymbol{I}} \star \underset{2}{\boldsymbol{I}}) \otimes \underset{2}{\boldsymbol{I}})\ \underset{mn\times mn}{\boldsymbol{Q}}\ (\underset{\frac{n}{2}}{\boldsymbol{I}} \otimes (\underset{2}{\boldsymbol{I}} \star \underset{\frac{m}{2}}{\boldsymbol{I}}) \otimes \underset{2}{\boldsymbol{I}}) \end{aligned} \tag{3–25}$$

where $\underset{mn\times mn}{Q} \triangleq \underset{\frac{mn}{4}}{I} \otimes \underset{4\times 4}{K}$.

An examination of (3–25) in the context of the rules of exchanging indices in array algebra (*see* (2–60)) reveals that

$$
\begin{aligned}
{}^{2}H &= \left(\underset{\frac{n}{2}}{I} \otimes \left(\underset{\frac{m}{2}}{I} * \underset{2}{I}\right) \otimes \underset{2}{I}\right) \underset{mn\times mn}{Q} \left(\underset{\frac{n}{2}}{I} \otimes \left(\underset{2}{I} * \underset{\frac{m}{2}}{I}\right) \otimes \underset{2}{I}\right) && (3\text{–}26)\\
&\Updownarrow \\
\mathcal{H}\vdots_{\alpha\beta\gamma\delta}^{abcd} &:= \mathcal{Q}\vdots_{\alpha\gamma\beta\delta}^{acbd} \\
&:= \mathcal{I}\vdots_{\alpha\gamma}^{ac}\, \mathcal{K}\vdots_{\beta\delta}^{bd} && (3\text{–}27)
\end{aligned}
$$

Let $\mathcal{X}\vdots_{\alpha\beta}^{\gamma\delta}$ and $\mathcal{Y}\vdots_{ab}^{cd}$ be the matrix-array representations of the input and output matrices, $\boldsymbol{X}$ and $\boldsymbol{Y}$, respectively. Then, the column vectorized matrix-arrays of $\boldsymbol{X}$ and $\boldsymbol{Y}$, are by definition (2–58) $\mathcal{Y}\vdash^{abcd}$ and $\mathcal{X}\vdash^{\alpha\beta\gamma\delta}$, respectively. The matrix-array representation of (3–20) is, consequently,

$$
\begin{aligned}
\mathrm{vec_c}(\boldsymbol{Y}) &= {}^{2}\boldsymbol{H}_c \cdot \mathrm{vec_c}(\boldsymbol{X}) \\
&\Updownarrow && (3\text{–}28)\\
\mathcal{Y}\vdash^{abcd} &:= \mathcal{H}\vdots_{\alpha\beta\gamma\delta}^{abcd}\, \mathcal{X}\vdash^{\alpha\beta\gamma\delta} \\
& && (3\text{–}29)
\end{aligned}
$$

where the row and column variables span counts representative of the tessellations of the input, output and processing planes. E.g., in the described 2-D 4-input/4-output node processing plane: $\langle b\rangle = \langle d\rangle = \langle\beta\rangle = \langle\delta\rangle = 2$ and $\langle a\rangle = \langle\alpha\rangle = \frac{m}{2}$ and $\langle c\rangle = \langle\gamma\rangle = \frac{n}{2}$.

Note that (3–27) may be used to describe any arbitrarily partitioned processing plane. E.g., the canonical form of the 1-D process plane may be derived by setting $\langle c\rangle = \langle\gamma\rangle = \langle d\rangle = \langle\delta\rangle = 1$ and $\langle a\rangle = \langle\alpha\rangle = \frac{n}{2}$ and $\langle b\rangle = \langle\beta\rangle = 2$,

$$
\begin{aligned}
\mathcal{H}\vdots_{\alpha\beta}^{ab} &:= \mathcal{I}\vdots_{\alpha}^{a}\, \mathcal{K}\vdots_{\beta}^{b} \\
&\Updownarrow \\
\boldsymbol{H} &= \underset{\frac{n}{2}}{I} \otimes \underset{2\times 2}{K}
\end{aligned}
$$

*Observation 3:* If $\underset{4\times 4}{K} = \underset{2\times 2}{K_r} \otimes \underset{2\times 2}{K_c}$, then

$$
\mathcal{H}\vdots_{\alpha\beta\gamma\delta}^{abcd} := \mathcal{I}\vdots_{\alpha\gamma}^{ac}\, \mathcal{K}\vdots_{\beta\delta}^{bd}
$$

$$
\begin{aligned}
&:= \mathcal{I}\vdash_{\alpha}^{a}\mathcal{I}\vdash_{\gamma}^{c}\mathcal{K}_{r}\vdash_{\beta}^{b}\mathcal{K}_{c}\vdash_{\delta}^{d} \\
&= \mathcal{I}\vdash_{\alpha}^{a}\mathcal{K}_{r}\vdash_{\beta}^{b}\mathcal{I}\vdash_{\gamma}^{c}\mathcal{K}_{c}\vdash_{\delta}^{d} \\
&:= (\underset{\frac{m}{2}}{\boldsymbol{I}}\otimes\underset{2\times 2}{\boldsymbol{K}_{r}})\vdash_{\alpha\beta}^{ab}(\underset{\frac{n}{2}}{\boldsymbol{I}}\otimes\underset{2\times 2}{\boldsymbol{K}_{c}})\vdash_{\gamma\delta}^{cd} \\
&\Updownarrow \\
{}^{2}\boldsymbol{H}_{c} &= (\underset{\frac{m}{2}}{\boldsymbol{I}}\otimes\underset{2\times 2}{\boldsymbol{K}_{r}})\otimes(\underset{\frac{n}{2}}{\boldsymbol{I}}\otimes\underset{2\times 2}{\boldsymbol{K}_{c}})
\end{aligned}
$$

assuming, of course, that $\langle a\rangle = \langle\alpha\rangle = \frac{m}{2}$ and $\langle c\rangle = \langle\gamma\rangle = \frac{n}{2}$. In other words,

$$\mathbf{vec}_{c}(\boldsymbol{Y}) = (\underset{\frac{m}{2}}{\boldsymbol{I}}\otimes\underset{2\times 2}{\boldsymbol{K}_{r}})\otimes(\underset{\frac{n}{2}}{\boldsymbol{I}}\otimes\underset{2\times 2}{\boldsymbol{K}_{c}})\mathbf{vec}_{c}(\boldsymbol{X}) \tag{3-30}$$

$$\textit{[by (2-41)]} \qquad = \mathbf{vec}_{c}\left((\underset{\frac{n}{2}}{\boldsymbol{I}}\otimes\underset{2\times 2}{\boldsymbol{K}_{c}})\boldsymbol{X}(\underset{\frac{m}{2}}{\boldsymbol{I}}\otimes\underset{2\times 2}{\boldsymbol{K}_{r}})^{T}\right) \tag{3-31}$$

$$\Longrightarrow \boldsymbol{Y} = (\underset{\frac{n}{2}}{\boldsymbol{I}}\otimes\underset{2\times 2}{\boldsymbol{K}_{c}})\boldsymbol{X}(\underset{\frac{m}{2}}{\boldsymbol{I}}\otimes\underset{2\times 2}{\boldsymbol{K}_{r}})^{T} \tag{3-32}$$

This proves that if the kernel is separable, then the processing may also be done in a separable manner using 1-D stages instead of 2-D stages.

### 3.2.2 Summary of Basic Canonical Forms

An implication of the previous section, is that there are array algebra canonical forms for each Kronecker algebra canonical form.

Implicit in the canonical forms examined so far, is the organization or partitioning of data into equal sized subgroups.

The typical $m \times n$ image is assumed to be partitioned according to the factorizations $m = r \cdot t$ and $n = p \cdot q$. It is implicit that the subgroups are of size $t \times q$ and are arranged over a $r \times p$ grid. This is represented more explicitly in the array domain by

$$\mathcal{X}\vdash_{\alpha\beta}^{\gamma\delta}$$

where $\langle\alpha\rangle = p$, $\langle\beta\rangle = q$, $\langle\gamma\rangle = r$, $\langle\delta\rangle = t$.

In the column vectorized domain,

$$
\begin{aligned}
\mathbf{vec}_{c}(\boldsymbol{Y}) &= \boldsymbol{M}\cdot\mathbf{vec}_{c}(\boldsymbol{X}) \\
&\Updownarrow \\
\mathcal{Y}\vdash^{abcd} &:= \mathcal{M}\vdash_{\alpha\beta\gamma\delta}^{abcd}\mathcal{X}\vdash^{\alpha\beta\gamma\delta}
\end{aligned}
$$

where $\boldsymbol{M}$ is one of the following basic shuffle/processing stages.

### 2-D Separable Shuffling Stage

$$
\begin{aligned}
{}^{2}\underset{mn\times mn}{S} &= S_{[p|q]} \otimes S_{[r|t]} \\
&= (\underset{p}{I} * \underset{q}{I}) \otimes (\underset{r}{I} * \underset{t}{I}) \\
&\Updownarrow \\
\mathcal{S}\vdash^{abcd}_{\alpha\beta\gamma\delta} &:= (\underset{p}{I} * \underset{q}{I})\vdash^{ab}_{\alpha\beta}\ (\underset{r}{I} * \underset{t}{I})\vdash^{cd}_{\gamma\delta} \\
&:= \mathcal{I}\vdash^{b}_{\alpha}\mathcal{I}\vdash^{a}_{\beta}\ \mathcal{I}\vdash^{d}_{\gamma}\mathcal{I}\vdash^{c}_{\delta}
\end{aligned}
$$

$$
\Longrightarrow \qquad \boxed{\begin{aligned}
{}^{2}\underset{mn\times mn}{S} &= S_{[p|q]} \otimes S_{[r|t]} \\
&\Updownarrow \\
\mathcal{S}\vdash^{abcd}_{\alpha\beta\gamma\delta} &:= \mathcal{I}\vdash^{b}_{\alpha}\mathcal{I}\vdash^{a}_{\beta}\mathcal{I}\vdash^{d}_{\gamma}\mathcal{I}\vdash^{c}_{\delta}
\end{aligned}} \tag{3-33}
$$

### 2-D Isokernel Processing Stage

Interestingly, the derivation of the 2-D normal factors can be achieved less laboriously then in the previous section by application of array algebra at an earlier stage. Namely, starting with a more general form of (3–19)

$$
\begin{aligned}
\mathbf{vec_c}\left([\mathbf{vec_c}\left(\underset{q\times t}{Y_{ij}}\right)]\right) &= (\underset{r}{I}\otimes\underset{p}{I}\otimes\underset{qt\times qt}{K})\ \mathbf{vec_c}\left([\mathbf{vec_c}\left(\underset{q\times t}{X_{ij}}\right)]\right) \\
\mathbf{vec_c}\left([\mathbf{vec_c}\left(\underset{q\times t}{Y_{ij}}\right)]\right) &= \underset{mn\times mn}{Q}\ \mathbf{vec_c}\left([\mathbf{vec_c}\left(\underset{q\times t}{X_{ij}}\right)]\right) \\
&\Updownarrow \\
\mathcal{Y}''\vdash^{acbd} &:= (\underset{mn\times mn}{Q})\vdash^{acbd}_{\alpha\gamma\beta\delta}\ \mathcal{X}''\vdash^{\alpha\gamma\beta\delta} \\
&:= (\underset{r}{I}\otimes\underset{p}{I}\otimes\underset{qt\times qt}{K})\vdash^{acbd}_{\alpha\gamma\beta\delta}\ \mathcal{X}''\vdash^{\alpha\gamma\beta\delta} \\
&:= \mathcal{I}\vdash^{a}_{\alpha}\mathcal{I}\vdash^{c}_{\gamma}\mathcal{K}\vdash^{bd}_{\beta\delta}\ \mathcal{X}''\vdash^{\alpha\gamma\beta\delta}
\end{aligned}
$$

Since the desired form of the array equation is one that involves the column vectorizations $\mathcal{Y}\vdash^{abcd}$ and $\mathcal{X}\vdash^{\alpha\beta\gamma\delta}$ we will need to switch the second and third variables in the row sequences to effect the assignments

$$
\mathcal{Y}\vdash^{abcd} := \mathcal{Y}''\vdash^{acbd} \qquad \text{and} \qquad \mathcal{X}\vdash^{\alpha\beta\gamma\delta} := \mathcal{X}''\vdash^{\alpha\gamma\beta\delta}
$$

So that

$$\mathcal{Y}\!\!\models^{abcd} := \mathcal{H}\!\!\models^{abcd}_{\alpha\beta\gamma\delta} \mathcal{X}\!\!\models^{\alpha\beta\gamma\delta}$$

which forces the assignment $\mathcal{H}\!\!\models^{abcd}_{\alpha\beta\gamma\delta} := \mathcal{Q}\!\!\models^{acbd}_{\alpha\gamma\beta\delta}$ or

$$\begin{aligned}
\mathcal{H}\!\!\models^{abcd}_{\alpha\beta\gamma\delta} &:= \mathcal{I}\!\!\models^{a}_{\alpha}\mathcal{I}\!\!\models^{c}_{\gamma}\mathcal{K}\!\!\models^{bd}_{\beta\delta} \\
&:= (\underset{r}{\boldsymbol{I}} \otimes \underset{p}{\boldsymbol{I}} \otimes \underset{qt\times qt}{\boldsymbol{K}})\!\!\models^{acbd}_{\alpha\gamma\beta\delta} \\
&:= (\underset{rp}{\boldsymbol{I}} \otimes \underset{qt\times qt}{\boldsymbol{K}})\!\!\models^{acbd}_{\alpha\gamma\beta\delta} \\
\textit{[by (2-60)]} \qquad &\Updownarrow \\
\underset{mn\times mn}{{}^{2}\boldsymbol{H}} &= (\underset{r}{\boldsymbol{I}} \otimes (\underset{p}{\boldsymbol{I}} \star \underset{q}{\boldsymbol{I}}) \otimes \underset{t}{\boldsymbol{I}})(\underset{rp}{\boldsymbol{I}} \otimes \underset{qt\times qt}{\boldsymbol{K}})(\underset{r}{\boldsymbol{I}} \otimes (\underset{q}{\boldsymbol{I}} \star \underset{p}{\boldsymbol{I}}) \otimes \underset{t}{\boldsymbol{I}}) \\
&= (\underset{r}{\boldsymbol{I}} \otimes \underset{[p|q]}{\boldsymbol{S}} \otimes \underset{t}{\boldsymbol{I}})(\underset{rp}{\boldsymbol{I}} \otimes \underset{qt\times qt}{\boldsymbol{K}})(\underset{r}{\boldsymbol{I}} \otimes \underset{[q|p]}{\boldsymbol{S}} \otimes \underset{t}{\boldsymbol{I}}) \\
&= \underset{r}{\boldsymbol{I}} \otimes \Big((\underset{[p|q]}{\boldsymbol{S}} \otimes \underset{t}{\boldsymbol{I}})(\underset{p}{\boldsymbol{I}} \otimes \underset{qt\times qt}{\boldsymbol{K}})(\underset{[q|p]}{\boldsymbol{S}} \otimes \underset{t}{\boldsymbol{I}})\Big)
\end{aligned}$$

$$\Longrightarrow \quad \boxed{\begin{aligned}
\underset{mn\times mn}{{}^{2}\boldsymbol{H}} &= \underset{r}{\boldsymbol{I}} \otimes \Big((\underset{[p|q]}{\boldsymbol{S}} \otimes \underset{t}{\boldsymbol{I}})(\underset{p}{\boldsymbol{I}} \otimes \underset{qt\times qt}{\boldsymbol{K}})(\underset{[q|p]}{\boldsymbol{S}} \otimes \underset{t}{\boldsymbol{I}})\Big) \\
&\Updownarrow \\
\mathcal{H}\!\!\models^{abcd}_{\alpha\beta\gamma\delta} &:= \mathcal{I}\!\!\models^{a}_{\alpha}\mathcal{I}\!\!\models^{c}_{\gamma}\mathcal{K}\!\!\models^{bd}_{\beta\delta}
\end{aligned}} \qquad (3\text{–}34)$$

There are two basic combinations of the separable shuffle and the isokernel processing plane.

### Isokernel-Process/Separable-Shuffle Stage

$$\begin{aligned}
\underset{mn\times mn}{{}^{2}\boldsymbol{T}} &= \underset{mn\times mn}{{}^{2}\boldsymbol{S}}\ \underset{mn\times mn}{{}^{2}\boldsymbol{H}} \\
&\Updownarrow \\
\mathcal{T}\!\!\models^{abcd}_{\alpha\beta\gamma\delta} &:= \mathcal{S}\!\!\models^{abcd}_{efgh}\mathcal{H}\!\!\models^{efgh}_{\alpha\beta\gamma\delta} \\
&:= \mathcal{I}\!\!\models^{b}_{e}\mathcal{I}\!\!\models^{a}_{f}\mathcal{I}\!\!\models^{d}_{g}\mathcal{I}\!\!\models^{c}_{h}\ \ \mathcal{I}\!\!\models^{e}_{\alpha}\mathcal{I}\!\!\models^{g}_{\gamma}\mathcal{K}\!\!\models^{fh}_{\beta\delta} \\
&:= \mathcal{I}\!\!\models^{b}_{\alpha}\mathcal{I}\!\!\models^{d}_{\gamma}\mathcal{K}\!\!\models^{ac}_{\beta\delta}
\end{aligned}$$

$$\Longrightarrow \quad \boxed{\begin{aligned}
\underset{mn\times mn}{{}^{2}\boldsymbol{S}}\ \underset{mn\times mn}{{}^{2}\boldsymbol{H}} &= (\underset{[p|q]}{\boldsymbol{S}} \otimes \underset{[r|t]}{\boldsymbol{S}})\Big[\underset{r}{\boldsymbol{I}} \otimes \Big((\underset{[p|q]}{\boldsymbol{S}} \otimes \underset{t}{\boldsymbol{I}})(\underset{p}{\boldsymbol{I}} \otimes \underset{qt\times qt}{\boldsymbol{K}})(\underset{[q|p]}{\boldsymbol{S}} \otimes \underset{t}{\boldsymbol{I}})\Big)\Big] \\
&\Updownarrow \\
\mathcal{S}\!\!\models^{abcd}_{efgh}\mathcal{H}\!\!\models^{efgh}_{\alpha\beta\gamma\delta} &:= \mathcal{I}\!\!\models^{b}_{\alpha}\mathcal{I}\!\!\models^{d}_{\gamma}\mathcal{K}\!\!\models^{ac}_{\beta\delta}
\end{aligned}} \qquad (3\text{–}35)$$

**Separable-Shuffle/Isokernel-Process Stage**

$$
\begin{array}{rcl}
\underset{mn\times mn}{{}^{2}\boldsymbol{T}} & = & \underset{mn\times mn}{{}^{2}\boldsymbol{H}}\ \underset{mn\times mn}{{}^{2}\boldsymbol{S}} \\
 & \Updownarrow & \\
\mathcal{T}\vdash_{\alpha\beta\gamma\delta}^{abcd} & := & \mathcal{H}\vdash_{efgh}^{abcd}\mathcal{S}\vdash_{\alpha\beta\gamma\delta}^{efgh} \\
 & := & \mathcal{I}\vdash_{e}^{a}\mathcal{I}\vdash_{g}^{c}\mathcal{K}\vdash_{fh}^{bd}\ \mathcal{I}\vdash_{\alpha}^{f}\mathcal{I}\vdash_{\beta}^{e}\mathcal{I}\vdash_{\gamma}^{h}\mathcal{I}\vdash_{\delta}^{g} \\
 & := & \mathcal{I}\vdash_{\beta}^{a}\mathcal{I}\vdash_{\delta}^{c}\mathcal{K}\vdash_{\alpha\gamma}^{bd}
\end{array}
$$

$$
\Longrightarrow \boxed{
\begin{array}{rcl}
\underset{mn\times mn}{{}^{2}\boldsymbol{H}}\ \underset{mn\times mn}{{}^{2}\boldsymbol{S}} & = & \left[\underset{r}{\boldsymbol{I}}\otimes\left((\boldsymbol{S}_{[p|q]}\otimes\underset{t}{\boldsymbol{I}})(\underset{p}{\boldsymbol{I}}\otimes\underset{qt\times qt}{\boldsymbol{K}})(\boldsymbol{S}_{[q|p]}\otimes\underset{t}{\boldsymbol{I}})\right)\right](\boldsymbol{S}_{[q|p]}\otimes\boldsymbol{S}_{[t|r]}) \\
 & \Updownarrow & \\
\mathcal{H}\vdash_{efgh}^{abcd}\mathcal{S}\vdash_{\alpha\beta\gamma\delta}^{efgh} & := & \mathcal{I}\vdash_{\beta}^{a}\mathcal{I}\vdash_{\delta}^{c}\mathcal{K}\vdash_{\alpha\gamma}^{bd}
\end{array}
} \qquad (3\text{–}36)
$$

A comparison of the column vectorized canonical forms in the two algebras is summarized in Table 3.1.

Similar canonical forms may be easily derived for the row vectorized domain.

| Processing Flow | Canonical Forms: Kronecker Algebra *(Matrix Domain)* | Canonical Forms: Array Algebra *(M-Array Domain)* |
|---|---|---|
| Partitioned Matrix | $\underset{m\times n}{\boldsymbol{X}}$ $(m = r \cdot t,\ n = p \cdot q)$ | $\mathcal{X}\vdash_{\alpha\beta}^{\gamma\delta}$ $\langle\alpha\rangle = p,\ \langle\beta\rangle = q,\ \langle\gamma\rangle = r,\ \langle\delta\rangle = t$ |
| Vectorization | $\mathbf{vec_c}\left(\underset{m\times n}{\boldsymbol{X}}\right)$ <br> $\mathbf{vec_r}\left(\underset{m\times n}{\boldsymbol{X}}\right)$ | $\mathcal{X}\vdash^{\alpha\beta\gamma\delta}$ <br> $\mathcal{X}\vdash^{\gamma\delta\alpha\beta}$ |
| Shuffle | $\underset{mn\times mn}{{}^{2}\boldsymbol{S}} = \boldsymbol{S}_{[p\|q]} \otimes \boldsymbol{S}_{[r\|t]}$ | $\mathcal{S}\vdash_{\alpha\beta\gamma\delta}^{abcd} := \mathcal{I}\vdash_{\alpha}^{b}\mathcal{I}\vdash_{\beta}^{a}\mathcal{I}\vdash_{\gamma}^{d}\mathcal{I}\vdash_{\delta}^{c}$ |
| Processing Plane | $\underset{mn\times mn}{{}^{2}\boldsymbol{H}} = \underset{r}{\boldsymbol{I}} \otimes \left((\boldsymbol{S}_{[p\|q]} \otimes \underset{t}{\boldsymbol{I}})(\underset{p}{\boldsymbol{I}} \otimes \underset{qt\times qt}{\boldsymbol{K}})(\boldsymbol{S}_{[q\|p]} \otimes \underset{t}{\boldsymbol{I}})\right)$ | $\mathcal{H}\vdash_{\alpha\beta\gamma\delta}^{abcd} := \mathcal{I}\vdash_{\alpha}^{a}\mathcal{I}\vdash_{\gamma}^{c}\mathcal{K}\vdash_{\beta\delta}^{bd}$ |
| Process/Shuffle | $\underset{mn\times mn}{{}^{2}\boldsymbol{T}} = (\boldsymbol{S}_{[p\|q]} \otimes \boldsymbol{S}_{[r\|t]})\left[\underset{r}{\boldsymbol{I}} \otimes \left((\boldsymbol{S}_{[p\|q]} \otimes \underset{t}{\boldsymbol{I}})(\underset{p}{\boldsymbol{I}} \otimes \underset{qt\times qt}{\boldsymbol{K}})(\boldsymbol{S}_{[q\|p]} \otimes \underset{t}{\boldsymbol{I}})\right)\right]$ | $\mathcal{T}\vdash_{\alpha\beta\gamma\delta}^{abcd} := \mathcal{I}\vdash_{\alpha}^{b}\mathcal{I}\vdash_{\gamma}^{d}\mathcal{K}\vdash_{\beta\delta}^{ac}$ |
| Shuffle/Process | $\underset{mn\times mn}{{}^{2}\boldsymbol{T}} = \left[\underset{r}{\boldsymbol{I}} \otimes \left((\boldsymbol{S}_{[p\|q]} \otimes \underset{t}{\boldsymbol{I}})(\underset{p}{\boldsymbol{I}} \otimes \underset{qt\times qt}{\boldsymbol{K}})(\boldsymbol{S}_{[q\|p]} \otimes \underset{t}{\boldsymbol{I}})\right)\right](\boldsymbol{S}_{[p\|q]} \otimes \boldsymbol{S}_{[r\|t]})$ | $\mathcal{T}\vdash_{\alpha\beta\gamma\delta}^{abcd} := \mathcal{I}\vdash_{\beta}^{a}\mathcal{I}\vdash_{\delta}^{c}\mathcal{K}\vdash_{\alpha\gamma}^{bd}$ |

Table 3.1: Canonical forms of separable shuffles, isokernel processing planes, and their combinations. (Column vectorized domain)

### 2-D Isonode Processing Stage

The representation of a 2-D isonode processing plane of arbitrary weights is obtained by starting with an isonode version of (3–19), i.e.,

$$\text{vec}([\text{vec}(\boldsymbol{Y}_{ij})]) = \underbrace{(\sum_{i=0}^{r-1} \underset{r\times r}{\boldsymbol{U}^{(i,i)}} \otimes \sum_{j=0}^{p-1} \underset{p\times p}{\boldsymbol{U}^{(j,j)}} \otimes \underset{qt\times qt}{\boldsymbol{K}_{ij}})}_{\underset{mn\times mn}{{}^{2}\boldsymbol{Q}}} \text{vec}([\text{vec}(\boldsymbol{X}_{ij})]). \qquad (3\text{–}37)$$

By a comparison to the isokernel case, it can readily be seen that

$${}^{2}\underset{mn\times mn}{\boldsymbol{H}} = \left( (\underset{r}{\boldsymbol{I}} \otimes \boldsymbol{S}_{[p|q]} \otimes \underset{t}{\boldsymbol{I}}) \left( \sum_{i=0}^{r-1} \underset{r\times r}{\boldsymbol{U}^{(i,i)}} \otimes \sum_{j=0}^{p-1} \underset{p\times p}{\boldsymbol{U}^{(j,j)}} \otimes \underset{qt\times qt}{\boldsymbol{K}_{ij}} \right) (\underset{r}{\boldsymbol{I}} \otimes \boldsymbol{S}_{[q|p]} \otimes \underset{t}{\boldsymbol{I}}) \right) \qquad (3\text{–}38)$$

An equivalent array albebra canonical form can be derived for the general isokernel case.

## 3.3 2-D Omega Processor

The previously described 2-D processing stage of 4-input/4-output nodes, seems like a natural extension of the processing stage of the 1-D Omega processor for a 2-D Omega processor. Extending the concept of omega processing flow to two dimensions, however, requires a careful examination of the way in which the interconnection stages are to be extended. The question is what type of 2-D shuffle should be used?

Although one could argue that a folded 2-D perfect shuffle is appropriate, the 2-D separable perfect shuffle will be considered as the natural extension of the 1-D perfect shuffle. This has also been the choice of other researchers who have addressed 2-D omega networks such as Kumagai and Ikegaya [22] as well as Mei and Liu [31, 30].

Since Chapter 2 defined the canonical form of a 2-D separable shuffle and the previous section defined the canonical form of a 2-D processing stage, the canonical form of a 2-D Omega processor may be derived. An $N \times N$ input array is assumed so that the processing arrays contain $N/2 \times N/2$ 4-input/4-output nodes. The kernel matrices $\underset{4\times 4}{\boldsymbol{K}}$ and the two-dimensional shuffles are assumed to be defined for the appropriate vectorization domain.

An example of a $8 \times 8$ 2-D Omega processor is illustrated in Fig. 3.7 with the processing flow from top to bottom. The 2-D inverse Omega processor has the same structure but the data flows from the bottom to the top. The separable nature of

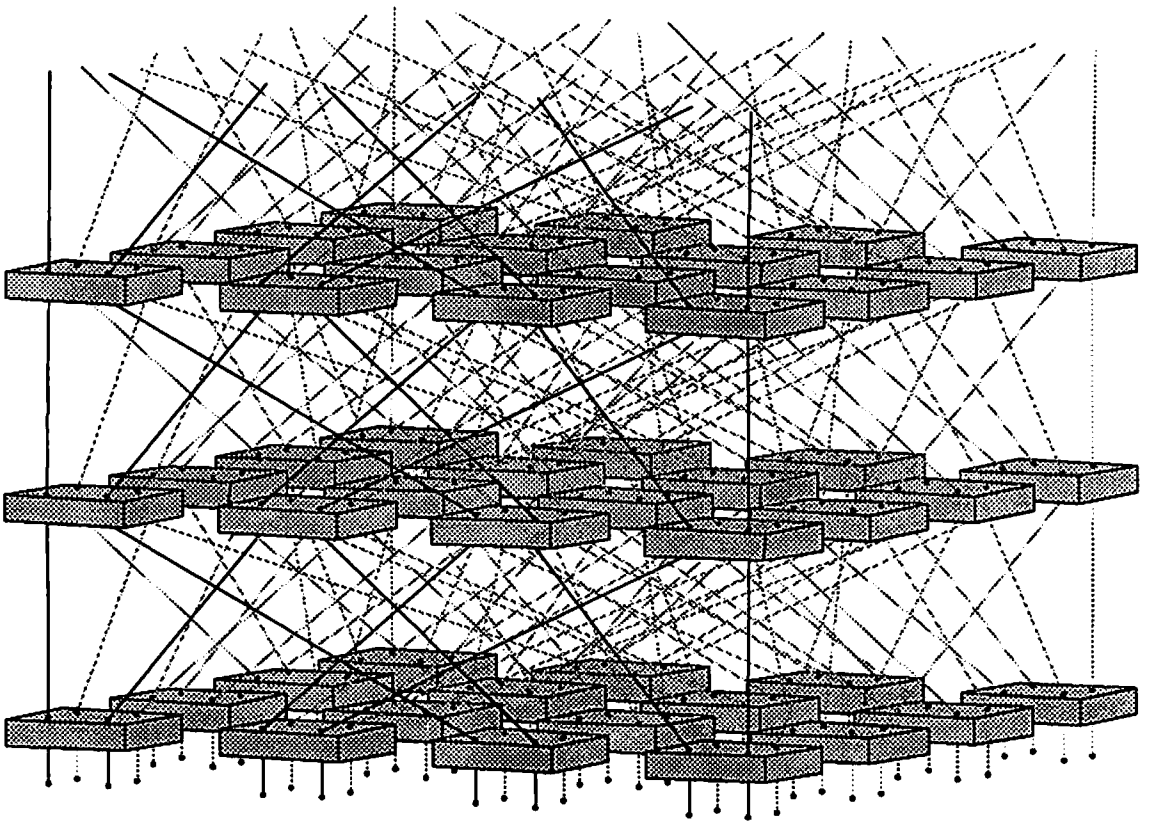


Figure 3.7: $8 \times 8$ Omega Processor. (Forward flow: top-down; Inverse flow: bottom-up)

the 2-D perfect shuffle is demonstrated by the cross-section slices at the edges of the interconnection stage. They are simply 1-D inverse perfect shuffles.

The canonical forms of 2-D Isokernel Omega Processors have the following Kronecker Algebra representations.

COROLLARY 23 *(2-D Local Isokernel Omega Canonical Form): The canonical form of a 2-D omega processing flow using identically valued 4-input/4-output processing kernels*

*within each processing plane is*

$$\mathfrak{Q} = \prod_{i=0}^{\log_2 N} \left[ \underbrace{\left[ \underset{\frac{N}{2}}{I} \otimes \left( (S_N^{-1} \otimes \underset{2}{I})( \underset{\frac{N}{2}}{I} \otimes \underset{4\times4}{K^i})(S_N \otimes \underset{2}{I}) \right) \right]}_{\text{2-D processing}} \underbrace{{}^2S_N}_{\text{2-D shuffle}} \right] \tag{3-39}$$

*where* $\underset{4\times4}{K^i}$ *is the kernel associated with stage i.*

And, of course,

COROLLARY 24 *(2-D Global Isokernel Omega Canonical Form): The canonical form of a 2-D omega processing flow using identically valued 4-input/4-output processing kernels within all processing planes is*

$$\mathfrak{Q} = \left[ \left[ \underset{\frac{N}{2}}{I} \otimes \left( (S_N^{-1} \otimes \underset{2}{I})( \underset{\frac{N}{2}}{I} \otimes \underset{4\times4}{K})(S_N \otimes \underset{2}{I}) \right) \right] {}^2S_N \right]^{\log_2 N} \tag{3-40}$$

*where* $\underset{4\times4}{K}$ *is the kernel applied at every stage.*

**2-D Inverse Omega** The canonical forms of the 2-D inverse omega processing flows are:

COROLLARY 25 *(2-D Local Isokernel Inverse Omega Canonical Form)*:

$$\mathfrak{Q}^{-1} = \prod_{i=0}^{\log_2 N} \left[ {}^2S_N^{-1} \left[ \underset{\frac{N}{2}}{I} \otimes \left( (S_N^{-1} \otimes \underset{2}{I})( \underset{\frac{N}{2}}{I} \otimes \underset{4\times4}{K^i})(S_N \otimes \underset{2}{I}) \right) \right] \right] \tag{3-41}$$

*where* $\underset{4\times4}{K^i}$ *is the kernel associated with stage i.*

And, of course,

COROLLARY 26 *(2-D Global Isokernel Inverse Omega Canonical Form)*:

$$\mathfrak{Q}^{-1} = \left[ {}^2S_N^{-1} \left[ \underset{\frac{N}{2}}{I} \otimes \left( (S_N^{-1} \otimes \underset{2}{I})( \underset{\frac{N}{2}}{I} \otimes \underset{4\times4}{K})(S_N \otimes \underset{2}{I}) \right) \right] \right]^{\log_2 N} \tag{3-42}$$

*where* $\underset{4\times4}{K}$ *is the kernel associated with every stage.*

Similar expressions may be derived in the context of array algebra.

# Chapter 4

# Task-to-Flow Mapping

The ultimate goal of introducing the tools of Chapter 2 is to relate a mathematical task to its potential implementations. In almost all modern day image processing applications, the data is collected in parallel either as a scanned line or a complete 2-D image or even as a multidimensional structure (multisensor observation). Therefore, the implementations of interest are those which take advantage of the simultaneous availability of data. The architectures presented in Chapter 3 are prototypical of the type of architectures needed and were characterized in terms of Kronecker and array algebra expressions. The problem now is, given a task, how does one determine if it can be performed on a given architecture?

In this chapter, a methodology is presented which maps a particular class of tasks to their corresponding isoplanar homosyndetic implementations.

## 4.1 1-D Mapping

In Chapter 3, the normal factor or canonical form of the normal isoplanar processing stage was presented, namely

$$\underset{p}{I} \otimes \underset{k \times q}{K}.$$

It is a fairly straight forward process to express other Kronecker products involving identity matrices in terms of normal factors and shuffles:

$$\underset{r\times q}{\boldsymbol{K}} \otimes \underset{n}{\boldsymbol{I}} = \boldsymbol{S}^{-1}_{[r|n]}(\underset{n}{\boldsymbol{I}} \otimes \underset{r\times q}{\boldsymbol{K}})\boldsymbol{S}_{[q|n]} \tag{4-1}$$

$$\underset{m}{\boldsymbol{I}} \otimes \underset{r\times q}{\boldsymbol{K}} \otimes \underset{n}{\boldsymbol{I}} = \boldsymbol{S}_{[n|mr]}(\underset{mn}{\boldsymbol{I}} \otimes \underset{r\times q}{\boldsymbol{K}})\boldsymbol{S}^{-1}_{[n|mq]} \tag{4-2}$$

$$\underset{m}{\boldsymbol{I}} \otimes \underset{r\times q}{\boldsymbol{K}} \otimes \underset{n}{\boldsymbol{I}} = (\underset{m}{\boldsymbol{I}} \otimes \boldsymbol{S}_{[n|r]})(\underset{mn}{\boldsymbol{I}} \otimes \underset{r\times q}{\boldsymbol{K}})(\underset{m}{\boldsymbol{I}} \otimes \boldsymbol{S}^{-1}_{[n|q]}) \tag{4-3}$$

These decompositions emphasize the important role that shuffle interconnections play in implementing structured transformations. For example, (4–1) can be implemented by a standard processing stage whose inputs and outputs have been shuffled and inverse shuffled respectively.

The block diagonal transformation is a special case of a more general class of transformations which can be expressed as Kronecker products of arbitrary matrices (not necessarily identities).

### General Kronecker Product Transformations

Any transformation matrix that can be expressed as a Kronecker product of $n$ smaller kernel matrices, i.e.,

$$\boldsymbol{H} = \boldsymbol{K}^{n-1} \otimes \cdots \otimes \boldsymbol{K}^1 \otimes \boldsymbol{K}^0 = \bigotimes_{i=n-1}^{0} \boldsymbol{K}^i$$

can be implemented by a (not necessarily homogeneous) cascade of $n$ processing stages and shuffle interconnections. This has been shown by Davio [9] based on the following factorization theorem.

THEOREM 27 *(Fundamental Kronecker Factorization): Let $\boldsymbol{M}^i$ represent a matrix of size $r_i \times c_i$ and let $\pi(\cdot)$ represent an arbitrary permutation. Then, the Kronecker product of $\boldsymbol{M}^i$, $i = 0, \ldots, n-1$, can be expressed by $n!$ different matrix products*

$$\bigotimes_{i=n-1}^{0} \boldsymbol{M}^i = \prod_{i=n-1}^{0} \left( \underset{(\alpha^i_{n-1}\cdots\alpha^i_{\pi(i)+1})}{\boldsymbol{I}} \otimes \boldsymbol{M}^{\pi(i)} \otimes \underset{(\alpha^i_{\pi(i)-1}\cdots\alpha^i_0)}{\boldsymbol{I}} \right) \tag{4-4}$$

| $i$ | 2 | 1 | 0 | $M^2 \otimes M^1 \otimes M^0$ |
|---|---|---|---|---|
| $\pi_1(i)$ | 2 | 1 | 0 | $(M^2 \otimes \underset{r_1 r_0}{I})(\underset{c_2}{I} \otimes M^1 \otimes \underset{r_0}{I})(\underset{c_2 c_1}{I} \otimes M^0)$ |
| $\pi_2(i)$ | 2 | 0 | 1 | $(M^2 \otimes \underset{r_1 r_0}{I})(\underset{c_2 r_1}{I} \otimes M^0)(\underset{c_2}{I} \otimes M^1 \otimes \underset{c_0}{I})$ |
| $\pi_3(i)$ | 1 | 2 | 0 | $(\underset{r_2}{I} \otimes M^1 \otimes \underset{r_0}{I})(M^2 \otimes \underset{c_1 r_0}{I})(\underset{c_2 c_1}{I} \otimes M^0)$ |
| $\pi_4(i)$ | 1 | 0 | 2 | $(\underset{r_2 r_1}{I} \otimes M^0)(M^2 \otimes \underset{r_1 c_0}{I})(\underset{c_2}{I} \otimes M^1 \otimes \underset{c_0}{I})$ |
| $\pi_5(i)$ | 0 | 2 | 1 | $(\underset{r_2}{I} \otimes M^1 \otimes \underset{r_0}{I})(\underset{r_2 c_1}{I} \otimes M^0)(M^2 \otimes \underset{c_1 c_0}{I})$ |
| $\pi_6(i)$ | 0 | 1 | 2 | $(\underset{r_2 r_1}{I} \otimes M^0)(\underset{r_2}{I} \otimes M^1 \otimes \underset{c_0}{I})(M^2 \otimes \underset{c_1 c_0}{I})$ |

Table 4.1: Factorizations of $M^2 \otimes M^1 \otimes M^0$

$$where \quad \alpha_k^i = \begin{cases} r_k & \text{if } \pi(i) > \pi(k) \\ c_k & \text{if } \pi(i) < \pi(k) \end{cases}$$

*Proof:* A proof is given in [9]. □

Note that the permutation $\pi(\cdot)$ determines the order of the kernel matrices on the right-hand side. As a result there are $n!$ possible factorizations. Therefore there are at least $n!$ basic ways of implementing the transformation. If one also considers that the Kronecker products of the right-hand side of (4–4) can be expressed as ordinary products of shuffles and normal factors in two different ways by (4–2) and (4–3), then there are $2^n \cdot n!$ distinct ways of implementing the transformation.

*Example 1:* For $n = 3$ there are $3! = 6$ possible implementations of $M^2 \otimes M^1 \otimes M^0$. That is, there are six unique permutations, $\pi_1(\cdot) \ldots \pi_6(\cdot)$, and corresponding factorizations as shown in Table 4.1. For example, using $\pi_1$,

$$\underset{2\times2}{C} \otimes \underset{2\times2}{B} \otimes \underset{2\times2}{A} = (\underset{2\times2}{C} \otimes \underset{4}{I})(\underset{2}{I} \otimes \underset{2\times2}{B} \otimes \underset{2}{I})(\underset{4}{I} \otimes \underset{2\times2}{A})$$

By using normal factor expansions, we can further derive

$$\underset{2\times2}{C} \otimes \underset{2\times2}{B} \otimes \underset{2\times2}{A} = \left[S_{[2|4]}^{-1}(\underset{4}{I} \otimes \underset{2\times2}{C})S_{[2|4]}\right]\left[S_{[2|4]}(\underset{4}{I} \otimes \underset{2\times2}{B})S_{[2|4]}^{-1}\right](\underset{4}{I} \otimes \underset{2\times2}{A})$$

$$= S_{[2|4]}^{-1}(\underset{4}{I} \otimes \underset{2\times 2}{C}) \cdot S_{[2|4]}^{-1}(\underset{4}{I} \otimes \underset{2\times 2}{B}) \cdot S_{[2|4]}^{-1}(\underset{4}{I} \otimes \underset{2\times 2}{A})$$
$$= S_8^{-1}(\underset{4}{I} \otimes \underset{2\times 2}{C}) \cdot S_8^{-1}(\underset{4}{I} \otimes \underset{2\times 2}{B}) \cdot S_8^{-1}(\underset{4}{I} \otimes \underset{2\times 2}{A})$$

which reveals a possible implementation in terms of pairs of inverse perfect shuffles and 2-input/2-output processing blocks. Such an implementation corresponds to that of a *1-D Inverse Omega processor.* Therefore, the permutation $\pi_1$ leads to implementations involving cascaded processing stages and inverse shuffles.

### 4.1.1 Shuffle/Process-Stage Implementation

If we wish to implement a Kronecker product by means of shuffle/process stages, then the following factorization [9] is useful:

COROLLARY 28 *(Factorization of Square Matrices): Let $\boldsymbol{K}^i$ represent a square matrix of size $q_i \times q_i$. Then, there is at least one factorization expressed purely in terms of pairs of inverse shuffles and processing stages.*

$$\bigotimes_{i=n-1}^{0} \underset{q_i\times q_i}{\boldsymbol{K}^i} = \prod_{i=n-1}^{0} \left[ S_{[q_i|Q_i]}^{-1}(\underset{Q_i}{\boldsymbol{I}} \otimes \underset{q_i\times q_i}{\boldsymbol{K}^i}) \right] \tag{4-5}$$

$$\text{where } Q_i = \prod_{\substack{j=0 \\ j\neq i}}^{n-1} q_j$$

*Proof:* Substitute (4–2) into Theorem 27 and consider only square matrices. □

*Example 2:* For $n = 2$, and $\boldsymbol{K}^0 = \boldsymbol{A}$, $\boldsymbol{K}^1 = \boldsymbol{B}$, and $q_0 = 2, q_1 = 4$:

$$\underset{4\times 4}{\boldsymbol{B}} \otimes \underset{2\times 2}{\boldsymbol{A}} = \underbrace{S_{[2|4]}^{-1}(\underset{2}{\boldsymbol{I}} \otimes \underset{4\times 4}{\boldsymbol{B}})}_{\text{stage 2}} \cdot \underbrace{S_{[4|2]}^{-1}(\underset{4}{\boldsymbol{I}} \otimes \underset{2\times 2}{\boldsymbol{A}})}_{\text{stage 1}}$$

COROLLARY 29 *(Factorization of Uniformly Square Matrices):*

$$\bigotimes_{i=n-1}^{0} \underset{q\times q}{\boldsymbol{K}^i} = \prod_{i=n-1}^{0} \left[ S_{[q|q^{n-1}]}^{-1}(\underset{q^{n-1}}{\boldsymbol{I}} \otimes \underset{q\times q}{\boldsymbol{K}^i}) \right] \tag{4-6}$$

*Proof:* Restrict the previous corollary to $q_i = q \ \forall \ i$. □

Note that such a factorization corresponds to an isoplanar (equal size normal factors) and homosyndetic (identical interconnection stages) architecture.

*Example 3:* For $n = 2$, and $\boldsymbol{K}^0 = \boldsymbol{A}$, $\boldsymbol{K}^1 = \boldsymbol{B}$, and $q = 3$:

$$\underset{3\times3}{\boldsymbol{B}} \otimes \underset{3\times3}{\boldsymbol{A}} = \underbrace{\boldsymbol{S}^{-1}_{[3|3]}(\underset{3}{\boldsymbol{I}} \otimes \underset{3\times3}{\boldsymbol{B}})}_{\text{stage 2}} \cdot \underbrace{\boldsymbol{S}^{-1}_{[3|3]}(\underset{3}{\boldsymbol{I}} \otimes \underset{3\times3}{\boldsymbol{A}})}_{\text{stage 1}}$$

COROLLARY 30 *(2-D Inverse Omega Decomposition)*:

$$\bigotimes_{i=n-1}^{0} \underset{2\times2}{\boldsymbol{K}^i} = \prod_{i=n-1}^{0} \left[\boldsymbol{S}^{-1}_{2^n}(\underset{2^{n-1}}{\boldsymbol{I}} \otimes \underset{2\times2}{\boldsymbol{K}^i})\right] \tag{4–7}$$

*Proof:* Restrict the previous corollary to $q = 2$. □

*Example 4:* For $n = 3$, and $\boldsymbol{K}^0 = \boldsymbol{A}$, $\boldsymbol{K}^1 = \boldsymbol{B}$, $\boldsymbol{K}^2 = \boldsymbol{C}$:

$$\begin{aligned} \underset{2\times2}{\boldsymbol{C}} \otimes \underset{2\times2}{\boldsymbol{B}} \otimes \underset{2\times2}{\boldsymbol{A}} &= \boldsymbol{S}^{-1}_{[2|4]}(\underset{4}{\boldsymbol{I}} \otimes \underset{2\times2}{\boldsymbol{C}}) \cdot \boldsymbol{S}^{-1}_{[2|4]}(\underset{4}{\boldsymbol{I}} \otimes \underset{2\times2}{\boldsymbol{B}}) \cdot \boldsymbol{S}^{-1}_{[2|4]}(\underset{4}{\boldsymbol{I}} \otimes \underset{2\times2}{\boldsymbol{A}}) \\ &= \underbrace{\boldsymbol{S}^{-1}_{8}(\underset{4}{\boldsymbol{I}} \otimes \underset{2\times2}{\boldsymbol{C}})}_{\text{stage 3}} \cdot \underbrace{\boldsymbol{S}^{-1}_{8}(\underset{4}{\boldsymbol{I}} \otimes \underset{2\times2}{\boldsymbol{B}})}_{\text{stage 2}} \cdot \underbrace{\boldsymbol{S}^{-1}_{8}(\underset{4}{\boldsymbol{I}} \otimes \underset{2\times2}{\boldsymbol{A}})}_{\text{stage 1}} \end{aligned}$$

Implementation of $\boldsymbol{y} = (\boldsymbol{C} \otimes \boldsymbol{B} \otimes \boldsymbol{A})\boldsymbol{x}$ is shown in Fig. 4.1.

## 4.2 2-D Mapping

In Chapter 3, the canonical form of the 2-D Omega processing flow was presented. Mapping to such an architecture can be achieved using the theorems derived from the 1-D mapping.

## 4.3 Application Examples

### 4.3.1 Global Matrix Permutations

Geometrical operations (e.g., rotations, transposes, etc.) can be efficiently mapped onto isoplanar homosyndetic processing flows.

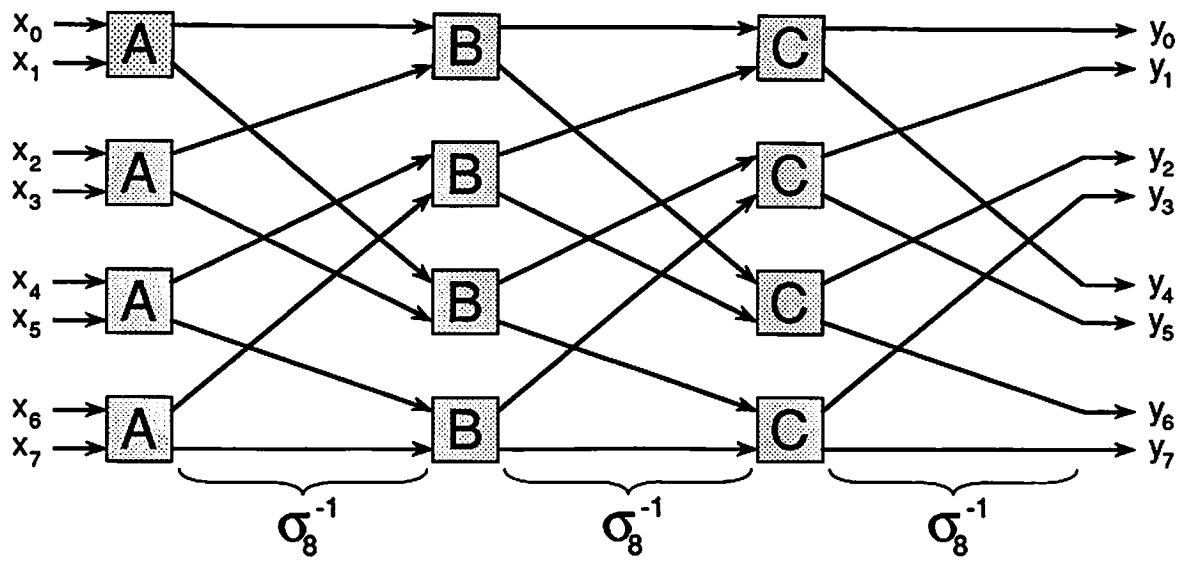



Figure 4.1: 1-D Inverse Omega Processing Flow of $\underset{2\times2}{C} \otimes \underset{2\times2}{B} \otimes \underset{2\times2}{A}$

### 90° Rotation

Consider the rotation of an $N \times N$ image, $\boldsymbol{X}$, by 90 degrees clockwise.

Kumagai and Ikegaya [22] suggested that a 2-D inverse omega processor may be used for rotating a matrix in 90° increments. However, no proof was given other than an example of a rotation of an $8 \times 8$ matrix.

Proving such a conjecture can be tedious at best without the help of Kronecker and array algebra.

The rotated matrix, $\boldsymbol{X}'$, can be expressed in matrix-space as

$$\begin{aligned} \underset{N\times N}{X'} = Rot_{-90°}(\underset{N\times N}{X}) &\triangleq \underset{N\times N}{X}^{T}\underset{N}{J} \\ &\triangleq (\underset{N}{J}\,\underset{N\times N}{X})^{T} \end{aligned}$$

where $\boldsymbol{J} = \begin{bmatrix} & & 1 \\ & \iddots & \\ 1 & & \end{bmatrix}$ represents a permutation which flips—about the axis of symmetry— the matrix to

which it is applied. If $J$ premultiplies $X$, then the effect is a vertical flip of $X$. If $J$ postmultiplies $X$, then a horizontal flip of $X$ about its middle axis takes place.

In column-scanned vector-space,

$$\begin{aligned} \mathrm{vec_c}\left(\underset{N\times N}{X'}\right) &= \mathrm{vec_c}\left((\underset{N}{J}\,\underset{N\times N}{X})^T\right) \\ \textit{[by Thm. 13]} \qquad &= S^{-1}_{[N|N]}\mathrm{vec_c}\left(\underset{N}{J}\,\underset{N\times N}{X}\right) \\ &= S_{[N|N]}\mathrm{vec_c}\left(\underset{N}{J}\,\underset{N\times N}{X}\,\underset{N}{I}\right) \\ \textit{[by (2-41)]} \qquad &= S^{-1}_{[N|N]}(\underset{N}{I}\otimes\underset{N}{J})\cdot\mathrm{vec_c}\left(\underset{N\times N}{X}\right) \end{aligned}$$

$$\Longrightarrow \qquad \mathrm{vec_c}\left(\underset{N\times N}{X'}\right) = S^{-1}_{[N|N]}(\underset{N}{I}\otimes\underset{N}{J})\cdot\mathrm{vec_c}\left(\underset{N\times N}{X}\right) \tag{4-8}$$

Consider the case where $N = 2^n$. Then,

$$\begin{aligned} S^{-1}_{[N|N]}(\underset{N}{I}\otimes\underset{N}{J}) &= S^{-1}_{[N|N]}\left[\overbrace{\underset{2}{I}\otimes\cdots\otimes\underset{2}{I}}^{n}\otimes\overbrace{\underset{2}{J}\otimes\cdots\otimes\underset{2}{J}}^{n}\right] \\ \textit{[by (4-6)]} \qquad &= S^{-1}_{[N|N]}\left[\left(S^{-1}_{[2|2^{2n-1}]}(\underset{2^{2n-1}}{I}\otimes\underset{2}{I})\right)^n\left(S^{-1}_{[2|2^{2n-1}]}(\underset{2^{2n-1}}{I}\otimes\underset{2}{J})\right)^n\right] \\ &= S^{-1}_{[N|N]}(S^{-1}_{N^2})^n\left[S^{-1}_{N^2}(\underset{N^2/2}{I}\otimes\underset{2}{J})\right]^n \end{aligned}$$

But,

$$\begin{aligned} S^{-1}_{[N|N]} &= S_{[N|2^n]} \\ \textit{[by (2-29)]} \qquad &= (S^{-1}_{N\cdot 2^n})^n \\ &= (S^{-1}_{N^2})^n \end{aligned}$$

Therefore,

$$\begin{aligned} S^{-1}_{[N|N]}(\underset{N}{I}\otimes\underset{N}{J}) &= \underbrace{(S^{-1}_{N^2})^{2n}}_{\underset{N^2}{I}}\left[S^{-1}_{N^2}(\underset{N^2/2}{I}\otimes\underset{2}{J})\right]^n \\ &= \left[S^{-1}_{N^2}(\underset{N^2/2}{I}\otimes\underset{2}{J})\right]^n \end{aligned}$$

which is a canonical form of a processing flow similar to the 1-D Inverse Omega processor except that the number of stages is only $n$—instead of $2n$ $(= \log_2 N^2)$.

The replicated processing stage will be shown to be equivalent to a two-dimensional canonical form by means of array algebra,

$$
\begin{aligned}
& & {}_{N^2\times N^2}\boldsymbol{T} &= \boldsymbol{S}^{-1}_{N^2}(\underset{N^2/2}{\boldsymbol{I}} \otimes \underset{2}{\boldsymbol{J}}) \\
& & &= \boldsymbol{S}_{[N^2/2|2]}(\underset{N^2/2}{\boldsymbol{I}} \otimes \underset{2}{\boldsymbol{J}}) \\
& \textit{[by (2–51)]} & &= \underset{N^2/2}{\boldsymbol{I}} \star \underset{2}{\boldsymbol{J}} \\
& \Longrightarrow & {}_{N^2\times N^2}\boldsymbol{T} &= (\underset{N/2}{\boldsymbol{I}} \otimes \underset{2}{\boldsymbol{I}} \otimes \underset{N/2}{\boldsymbol{I}}) \star \underset{2}{\boldsymbol{J}} \\
& & \Updownarrow & \\
& & \mathcal{T}\vert^{abcd}_{\alpha\beta\gamma\delta} &:= (\underset{N/2}{\boldsymbol{I}} \otimes \underset{2}{\boldsymbol{I}} \otimes \underset{N/2}{\boldsymbol{I}})\vert^{bcd}_{\alpha\beta\gamma}\ \mathcal{J}\vert^{a}_{\delta} \\
& & &:= \mathcal{I}\vert^{b}_{\alpha}\,\mathcal{I}\vert^{c}_{\beta}\,\mathcal{I}\vert^{d}_{\gamma}\ \mathcal{J}\vert^{a}_{\delta} \\
& & &:= \mathcal{I}\vert^{b}_{\alpha}\,\mathcal{I}\vert^{d}_{\gamma}\ (\mathcal{I}\vert^{c}_{\beta}\,\mathcal{J}\vert^{a}_{\delta}) \\
& \Longrightarrow & \mathcal{T}\vert^{abcd}_{\alpha\beta\gamma\delta} &:= \mathcal{I}\vert^{b}_{\alpha}\,\mathcal{I}\vert^{d}_{\gamma}\ (\underset{2}{\boldsymbol{I}} \star \underset{2}{\boldsymbol{J}})\vert^{ac}_{\beta\delta}
\end{aligned}
$$

Examination of this array equation reveals that it matches the array algebra canonical form (3–35) of an isokernel 2-D process/shuffle stage where

$$
\begin{aligned}
\boldsymbol{K} &= \underset{2}{\boldsymbol{I}} \star \underset{2}{\boldsymbol{J}} \\
\textit{[by (2–29)]} \qquad &= \boldsymbol{S}_{[2|2]}(\underset{2}{\boldsymbol{I}} \otimes \underset{2}{\boldsymbol{J}}).
\end{aligned}
$$

Consequently,

$$
\begin{aligned}
\mathcal{T}\vert^{abcd}_{\alpha\beta\gamma\delta} &:= \mathcal{I}\vert^{b}_{\alpha}\,\mathcal{I}\vert^{d}_{\gamma}\ (\underset{2}{\boldsymbol{I}} \star \underset{2}{\boldsymbol{J}})\vert^{ac}_{\beta\delta} \\
\Updownarrow & \\
{}_{N^2\times N^2}\boldsymbol{T} &= (\boldsymbol{S}^{-1}_{N} \otimes \boldsymbol{S}^{-1}_{N})\left[\underset{N/2}{\boldsymbol{I}} \otimes \left((\boldsymbol{S}^{-1}_{N} \otimes \underset{2}{\boldsymbol{I}})(\underset{N/2}{\boldsymbol{I}} \otimes (\underset{2}{\boldsymbol{I}} \star \underset{2}{\boldsymbol{J}}))(\boldsymbol{S}_{N} \otimes \underset{2}{\boldsymbol{I}})\right)\right] \\
&= (\boldsymbol{S}^{-1}_{N} \otimes \boldsymbol{S}^{-1}_{N})\left[\underset{N/2}{\boldsymbol{I}} \otimes \left((\boldsymbol{S}^{-1}_{N} \otimes \underset{2}{\boldsymbol{I}})(\underset{N/2}{\boldsymbol{I}} \otimes \boldsymbol{S}_{[2|2]}(\underset{2}{\boldsymbol{I}} \otimes \underset{2}{\boldsymbol{J}}))(\boldsymbol{S}_{N} \otimes \underset{2}{\boldsymbol{I}})\right)\right]
\end{aligned}
$$

Therefore, the 2-D processing flow of a 90° clockwise rotation is

$$
\boldsymbol{S}^{-1}_{[N|N]}(\underset{N}{\boldsymbol{I}} \otimes \underset{N}{\boldsymbol{J}}) = \left[{}_{N^2\times N^2}\boldsymbol{T}\right]^{n}
$$

$$
\begin{aligned}
&= \left[(S_N^{-1} \otimes S_N^{-1}) \right. \\
&\qquad \left. \left( \underset{N/2}{I} \otimes \left( (S_N^{-1} \otimes \underset{2}{I})( \underset{N/2}{I} \otimes S_{[2|2]}(\underset{2}{I} \otimes \underset{2}{J}))(S_N \otimes \underset{2}{I}) \right) \right) \right]^n \\
&= \left[ {}^{3}S_N^{-1} \left( \underset{N/2}{I} \otimes \left( (S_N^{-1} \otimes \underset{2}{I})( \underset{N/2}{I} \otimes S_{[2|2]}(\underset{2}{I} \otimes \underset{2}{J}))(S_N \otimes \underset{2}{I}) \right) \right) \right]^n \\
&\triangleq {}^{3}\underset{N \times N}{\Omega}^{-1} \Big|_{K = S_{[2|2]}(\underset{2}{I} \otimes \underset{2}{J})}
\end{aligned}
$$

$$
\Longrightarrow \qquad \mathrm{vec_c}\left( \underset{N \times N}{X'} \right) = {}^{3}\underset{N \times N}{\Omega}^{-1} \Big|_{K = S_{[2|2]}(\underset{2}{I} \otimes \underset{2}{J})} \cdot \mathrm{vec_c}\left( \underset{N \times N}{X} \right) \tag{4-9}
$$

Obviously, this is the canonical form of an isokernel inverse omega processing flow where each node executes $S_{[2|2]}(\underset{2}{I} \otimes \underset{2}{J})$ which rotates by 90° its $2 \times 2$ input, since

$$
\begin{aligned}
\mathrm{vec_c}\left( \underset{2 \times 2}{X'_{ij}} \right) &= S_{[2|2]}(\underset{2}{I} \otimes \underset{2}{J}) \ \mathrm{vec_c}\left( \underset{2 \times 2}{X_{ij}} \right) \\
&= S_{[2|2]} \ \mathrm{vec_c}\left( \underset{2}{J} \underset{2 \times 2}{X_{ij}} \underset{2}{I} \right) \\
&= S_{[2|2]}^{-1} \ \mathrm{vec_c}\left( \underset{2}{J} \underset{2 \times 2}{X_{ij}} \right) \\
&= \mathrm{vec_c}\left( (\underset{2}{J} \underset{2 \times 2}{X_{ij}})^T \right) \\
\Longrightarrow \underset{2 \times 2}{X'_{ij}} &= \left( \underset{2}{J} \underset{2 \times 2}{X_{ij}} \right)^T \\
&\triangleq Rot_{-90^\circ}(\underset{2 \times 2}{X_{ij}})
\end{aligned}
$$

In summary,

$$
\boxed{Rot_{-90^\circ}( \underset{2^n \times 2^n}{X} ) \mapsto {}^{3}\underset{2^n \times 2^n}{\Omega}^{-1} \Big|_{K = Rot_{-90^\circ}(\underset{2 \times 2}{X_{ij}})}}
$$

It is, certainly, intuitively pleasing to know that performing a 90° clockwise rotation within each node of an inverse omega processor effects a 90° rotation of the whole image.

Actually, this type of effect works for similar type of operations including 90° counter clockwise, where

$$
\underset{N \times N}{X''} = Rot_{90^\circ}( \underset{N \times N}{X} ) \triangleq ( \underset{N \times N}{X} \underset{N}{J})^T.
$$

and in column-scanned vector-space,

$$\begin{aligned}
\mathrm{vec_c}\left(\underset{N\times N}{X''}\right) &= \mathrm{vec_c}\left(\left(\underset{N\times N}{X}\,\underset{N}{J}\right)^T\right) \\
\textit{[by Thm. 13]} \qquad &= S^{-1}_{[N|N]}\mathrm{vec_c}\left(\underset{N\times N}{X}\,\underset{N}{J}\right) \\
&= S_{[N|N]}\mathrm{vec_c}\left(\underset{N}{I}\,\underset{N\times N}{X}\,\underset{N}{J}\right) \\
\textit{[by (2-41)]} \qquad &= S^{-1}_{[N|N]}\left(\underset{N}{J}\otimes\underset{N}{I}\right)\cdot\mathrm{vec_c}\left(\underset{N\times N}{X}\right)
\end{aligned}$$

It is easily shown that

$$\boxed{Rot_{90°}\left(\underset{2^n\times 2^n}{X}\right)\mapsto \left.\underset{2^n\times 2^n}{\Im\Omega}^{-1}\right|_{K=Rot_{90°}\left(\underset{2\times 2}{X_{ij}}\right)}}$$

## Transposition

Transposition was seen as an integral part of the rotation operations i.e.,

$$Rot_{-90°}\left(\underset{N\times N}{X}\right)\triangleq \underset{N\times N}{X}^T\underset{N}{J}.$$

By substituting, $\underset{N}{I}$ for $\underset{N}{J}$, it is obvious that an omega processing flow can perform a transposition with the kernel derived from the rotation kernel,

$$\begin{aligned}
K &= S_{[2|2]}\left(\underset{2}{I}\otimes\underset{2}{I}\right) \\
&= S_{[2|2]}\underset{4}{I} \\
&= S_{[2|2]} \\
&= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
\end{aligned}$$

which happens to effect a transposition of a $2 \times 2$ matrix, e.g.,

$$\mathrm{vec_c}\left(\underset{2\times 2}{X_{ij}}^T\right) = S_{[2|2]}\cdot\mathrm{vec_c}\left(\underset{2\times 2}{X_{ij}}\right)$$

Therefore,

$$\boxed{\underset{2^n\times 2^n}{X}^T\mapsto \left.\underset{2^n\times 2^n}{\Im\Omega}^{-1}\right|_{K=S_4}}$$

### 4.3.2 Unitary Transforms

The *Generalized Discrete Transform* (GDT) defined by [13] is a class of unitary transforms that is suited for implementation on omega processors.

Neglecting the bit reversal reordering of the transformed output, if

$$\underset{N}{G} = \bigotimes_{i=n-1}^{0} \underset{2\times 2}{D_i}$$

then the 2-D transform of an image, $X$, can be expressed as:

$$Y = \underset{N}{G} \cdot \underset{N\times N}{X} \cdot \underset{N}{G^T}$$

In the column-scanned vector domain, the representation is

$$\begin{aligned} \mathrm{vec_c}(Y) &= \mathrm{vec_c}\left(\underset{N}{G} \cdot X \cdot \underset{N}{G^T}\right) \\ &= (\underset{N}{G} \otimes \underset{N}{G}) \cdot \mathrm{vec_c}(X) \\ &= \left(\bigotimes_{i=n-1}^{0} \underset{2\times 2}{D_i} \otimes \bigotimes_{i=n-1}^{0} \underset{2\times 2}{D_i}\right) \cdot \mathrm{vec_c}(X) \end{aligned}$$

Application of (4–6) yields

$$\begin{aligned} \underset{2}{H}^{\otimes n} \otimes \underset{2}{H}^{\otimes n} &= \left[\prod_{i=n-1}^{0} \left(S^{-1}_{[2|2^{n-1}]}(\underset{2^{n-1}}{I} \otimes \underset{2\times 2}{D_i})\right)\right. \\ &\qquad \left. \otimes \prod_{i=n-1}^{0} \left(S^{-1}_{[2|2^{n-1}]}(\underset{2^{n-1}}{I} \otimes \underset{2\times 2}{D_i})\right)\right] \\ \textit{[by (2–14)]} \qquad &= \prod_{i=n-1}^{0} \left[S^{-1}_{[2|2^{n-1}]}(\underset{2^{n-1}}{I} \otimes \underset{2\times 2}{D_i}) \otimes S^{-1}_{[2|2^{n-1}]}(\underset{2^{n-1}}{I} \otimes \underset{2\times 2}{D_i})\right] \\ &= \prod_{i=n-1}^{0} \left[(\underset{2^{n-1}}{I} \star \underset{2\times 2}{D_i}) \otimes (\underset{2^{n-1}}{I} \star \underset{2\times 2}{D_i})\right] \end{aligned}$$

Let

$$\begin{aligned} \underset{4^n\times 4^n}{T} &= (\underset{2^{n-1}}{I} \star \underset{2\times 2}{D_i}) \otimes (\underset{2^{n-1}}{I} \star \underset{2\times 2}{D_i}) \\ &\Updownarrow \\ T\Big|^{abcd}_{\alpha\beta\gamma\delta} &:= (\underset{2^{n-1}}{I} \star \underset{2\times 2}{D_i})\Big|^{ab}_{\alpha\beta} (\underset{2^{n-1}}{I} \star \underset{2\times 2}{D_i})\Big|^{cd}_{\gamma\delta} \end{aligned}$$

$$
\begin{aligned}
&:= \mathcal{I}_{\alpha}^{b}\,\mathcal{D}_{\beta}^{a}\;\mathcal{I}_{\gamma}^{d}\,\mathcal{D}_{\delta}^{c} \\
&:= \mathcal{I}_{\alpha}^{b}\,\mathcal{I}_{\gamma}^{d}\,\mathcal{D}_{\beta}^{a}\,\mathcal{D}_{\delta}^{c} \\
\Longrightarrow \mathcal{T}_{\alpha\beta\gamma\delta}^{abcd} &:= \mathcal{I}_{\alpha}^{b}\,\mathcal{I}_{\gamma}^{d}\,(\underset{2\times 2}{\boldsymbol{D}_{j}} \otimes \underset{2\times 2}{\boldsymbol{D}_{j}})_{\beta\delta}^{ac}
\end{aligned}
$$

This array equation matches the array algebra canonical form (3–35) of an isokernel 2-D process/shuffle stage where

$$
\boldsymbol{K} = \underset{2\times 2}{\boldsymbol{D}_{j}} \otimes \underset{2\times 2}{\boldsymbol{D}_{j}}
$$

Consequently,

$$
\begin{aligned}
\underset{N}{\boldsymbol{G}} \otimes \underset{N}{\boldsymbol{G}} \;=\; \prod_{i=n-1}^{0} \Big[ &(\boldsymbol{S}_{N}^{-1} \otimes \boldsymbol{S}_{N}^{-1}) \\
&\Big( \underset{N/2}{\boldsymbol{I}} \otimes \Big( (\boldsymbol{S}_{N}^{-1} \otimes \underset{2}{\boldsymbol{I}})( \underset{N/2}{\boldsymbol{I}} \otimes \underset{2\times 2}{\boldsymbol{D}_{j}} \otimes \underset{2\times 2}{\boldsymbol{D}_{j}})(\boldsymbol{S}_{N} \otimes \underset{2}{\boldsymbol{I}}) \Big) \Big) \Big]
\end{aligned}
\qquad (4\text{–}10)
$$

## Hadamard Transform

One of the simplest members of the *Generalized Discrete Transform* class defined by [13], is the Hadamard transform. It is useful in image bandwidth compression and filtering [16].

An ordered Hadamard transform of an $N \times N$ image is used to illustrate the mapping process. In addition to being separable and symmetric, the ordered Hadamard transform is recursively generated from a $2 \times 2$ kernel matrix:

$$
\underset{2}{\boldsymbol{H}} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}
$$

Matrices representing an $N = 2^n$ order transform, are based on the Kronecker relationship:

$$
\underset{2^n}{\boldsymbol{H}} = \underset{2}{\boldsymbol{H}}^{\otimes n} \qquad (4\text{–}11)
$$

where the superscript $^{\otimes n}$ denotes the Kronecker power of $n$, i.e., $\boldsymbol{Z}^{\otimes n} = \boldsymbol{Z} \otimes \boldsymbol{Z}^{\otimes n-1}$.

The transform also involves a scaling factor of $\sqrt{N}$, so the matrix representing a one-dimensional ordered Hadamard is

$$
\underset{N}{\boldsymbol{A}} = \frac{1}{\sqrt{N}} \underset{N}{\boldsymbol{H}}
$$

In the two-dimensional domain, the transform of an image, $X$, can be expressed as:

$$Y = \underset{N}{A} \cdot \underset{N\times N}{X} \cdot \underset{N}{A^T}$$

In the column-scanned vector domain, the representation is

$$\begin{aligned} \mathrm{vec_c}(Y) &= \mathrm{vec_c}\left(\underset{N}{A} \cdot X \cdot \underset{N}{A^T}\right) \\ &= (\underset{N}{A} \otimes \underset{N}{A}) \cdot \mathrm{vec_c}(X) \\ &= \frac{1}{N}(\underset{2}{H^{\otimes n}} \otimes \underset{2}{H^{\otimes n}}) \cdot \mathrm{vec_c}(X) \end{aligned}$$

Application of (4–6) yields

$$\begin{aligned} \underset{2}{H^{\otimes n}} \otimes \underset{2}{H^{\otimes n}} &= \left[\left(S^{-1}_{[2|2^{n-1}]}(\underset{2^{n-1}}{I} \otimes \underset{2}{H})\right)^n \otimes \left(S^{-1}_{[2|2^{n-1}]}(\underset{2^{n-1}}{I} \otimes \underset{2}{H})\right)^n\right] \\ \textit{[by (2–14)]} \qquad &= \left[S^{-1}_{[2|2^{n-1}]}(\underset{2^{n-1}}{I} \otimes \underset{2}{H}) \otimes S^{-1}_{[2|2^{n-1}]}(\underset{2^{n-1}}{I} \otimes \underset{2}{H})\right]^n \\ &= \left[(\underset{2^{n-1}}{I} \star \underset{2}{H}) \otimes (\underset{2^{n-1}}{I} \star \underset{2}{H})\right]^n \end{aligned}$$

Let

$$\begin{aligned} \underset{4^n\times 4^n}{T} &= (\underset{2^{n-1}}{I} \star \underset{2}{H}) \otimes (\underset{2^{n-1}}{I} \star \underset{2}{H}) \\ &\Updownarrow \\ \mathcal{T}\vert^{abcd}_{\alpha\beta\gamma\delta} &:= (\underset{2^{n-1}}{I} \star \underset{2}{H})\vert^{ab}_{\alpha\beta} (\underset{2^{n-1}}{I} \star \underset{2}{H})\vert^{cd}_{\gamma\delta} \\ &:= \mathcal{I}\vert^b_\alpha \mathcal{H}\vert^a_\beta \; \mathcal{I}\vert^d_\gamma \mathcal{H}\vert^c_\delta \\ &:= \mathcal{I}\vert^b_\alpha \mathcal{I}\vert^d_\gamma \mathcal{H}\vert^a_\beta \mathcal{H}\vert^c_\delta \\ &:= \mathcal{I}\vert^b_\alpha \mathcal{I}\vert^d_\gamma (\underset{2}{H} \otimes \underset{2}{H})\vert^{ac}_{\beta\delta} \\ \Longrightarrow \mathcal{T}\vert^{abcd}_{\alpha\beta\gamma\delta} &:= \mathcal{I}\vert^b_\alpha \mathcal{I}\vert^d_\gamma \; (\underset{4}{H})\vert^{ac}_{\beta\delta} \end{aligned}$$

This array equation matches the array algebra canonical form (3–35) of an isokernel 2-D process/shuffle stage where

$$K = \underset{4}{H}$$

Taking into account the scaling factor of $\frac{1}{N}$,

$$K = \frac{1}{N}\underset{4}{H} = \underset{N}{A}$$

Consequently,

$$\underset{N}{A} \otimes \underset{N}{A} = \left[ (S_N^{-1} \otimes S_N^{-1}) \left( \underset{N/2}{I} \otimes \left( (S_N^{-1} \otimes \underset{2}{I})( \underset{N/2}{I} \otimes \underset{N}{A})(S_N \otimes \underset{2}{I}) \right) \right) \right]^n \quad (4\text{–}12)$$

which is is the canonical form of a 2-D Omega processor with the kernel $\underset{2^n}{A}$ applied in every block of every processing plane. Such an operation will take $n$ process/shuffle steps to execute.

$$\boxed{\textit{2-D Hadamard}(\underset{2^n \times 2^n}{X}) \mapsto \underset{2^n \times 2^n}{\Omega^{-1}} \Big|_{K=\frac{1}{N}\underset{1}{H}}}$$

# Chapter 5

# Potential Implementations

## 5.1 Optical Shuffles

Optics affords a potential means of implementing complicated interconnection patterns including *one-* and *two-dimensional shuffles.* This may be achieved either by using classical optical components, such as lenses, or by use of more recent developments such as computer generated holograms. The information to be transmitted is encoded onto a beam of light which is routed in free space to its appropriate destination. The encoding method depends on the implementation technology although frequency or amplitude modulation predominate in most applications.

### 5.1.1 Shuffling with Simple Lenses

Consider an array[1] of light emitters to be interconnected to a similar array of detectors. A special subclass of shuffles may be realized in either one or two dimensions by strategic placement of simple lenses within a plane parallel to the emitter and detector planes. The use of convex thin lenses is assumed.

Analysis of such a system may be explained by means of Kronecker and array algebra as shown in the following sections. The result is a precise relationship between the system parameters (e.g., number and location of lenses, etc.) and the realized shuffle.

[1] In this section, the terms *array* and *image* are used in the more conventional optical sense.

The one-dimensional case will be examined next.

## Emitter/Detector Array Geometries

Consider a linear array of regularly spaced light emitters, which by means of one or more thin lenses, forms an array of magnified spotlights or *footprints* on a plane sampled by a linear array of detectors. These arrays are referred to as the *emitter array*, the *footprint array* and the *detector array*, respectively. The axis of symmetry through an array is the *array axis*. The axis of symmetry through a lens is the *lens axis*.

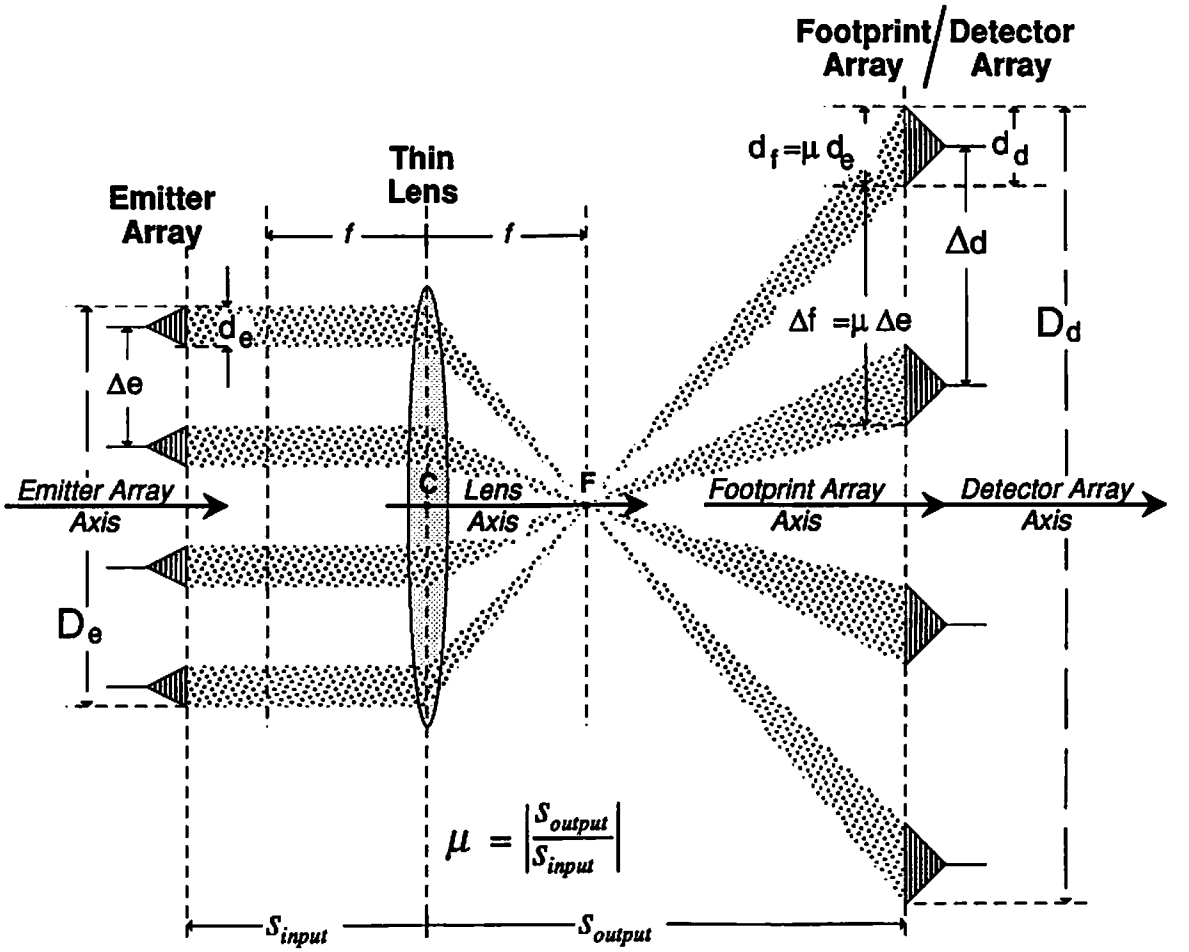



Figure 5.1: Emitter-detector array configuration.

As shown in Fig. 5.1, each emitter has a diameter $d_e$ and a center-to-center spacing of $\Delta e$. The length of an array of $n$ emitters is

$$D_e = d_e + (n-1)\Delta e.$$

The imaged footprint dimensions are magnified versions of the emitter array dimensions. For thin lenses, the Gaussian lens equation

$$\frac{1}{f} = \frac{1}{s_{input}} + \frac{1}{s_{output}}$$

defines the relationship of focal length $f$ to the distances of the emitter array and its image from the *optical center* of the lens. The imaged footprint of a single emitter, is the area seen at the imaged plane and is dependent on the *lateral* or *transverse magnification*

$$\mu = \left| \frac{s_{output}}{s_{input}} \right|.$$

Therefore, the size of a single footprint is $d_f = \mu d_e$ and their center-to-center separation is $\Delta f = \mu \Delta e$ (*see* Fig. 5.1).

**Detector-to-Footprint Interface.** The size and spacing of the detector elements are denoted as $d_d$ and $\Delta d$, respectively. Note that the detector elements may, conceivably, be of variable size and separation as long as they sample within their respective footprint areas. For optimal power transfer, however, the detector array should match the imaged footprint of the emitter, i.e.,

$$d_d = d_f = \mu d_e.$$

*Assumption 1:* Each detector element is centered within a unique imaged footprint. Therefore, the detector array and footprint array always share a common axis of symmetry. The length of the detector array is denoted by $D_d$ (*see* Fig. 5.1).

## Vector Representation of Spatially Discrete Optics

Expressing the optical imaging of an emitter array to a detector array by Kronecker algebra, requires a specific correspondence between the emitter, footprint and detector array physical attributes and their representative vectors. Let $\boldsymbol{x}$ represent the emitter

array, $\boldsymbol{y}$ the footprint array, and $\boldsymbol{x}'$ the detector array. The former is considered to be the *input vector.* The latter are the *footprint vector* and *detected vector*, respectively. Vectors are assumed to have some imaginary physical dimensions such as uniform element size and an axis of symmetry called the *vector axis.*

The correspondence of the emitter array to the input vector is straightforward. The input vector contains as many elements as the emitter array. A physical correspondence is shown in Fig. 5.2, where the emitter array and the input vector share a common axis and the imaginary height $\Delta_{\boldsymbol{x}}$ of an input vector element is equal to the physical separation $\Delta e$ of the emitter elements,

$$\Delta_{\boldsymbol{x}} = \Delta e. \tag{5-1}$$

Similarly, the detected vector, $\boldsymbol{x}'$, has as many elements as the detector array with each element height $\Delta_{\boldsymbol{x}'}$ equal to the physical separation $\Delta d$ of the detector elements.

Representation of the footprint array requires more thought. If the footprint vector contained only as many elements as imaged values, then the vector would be devoid of any magnification or offset information. From a schematic standpoint, this corresponds to setting the imaginary height $\Delta_{\boldsymbol{y}}$ of the footprint vector elements equal to the footprint element separation , i.e,

$$\Delta_{\boldsymbol{y}} = \Delta f = \mu \Delta e = \mu \Delta_{\boldsymbol{x}}.$$

If however, the size of the footprints and their separation can be expressed in integral multiples of the element height, then a vector could readily denote the separation between the nearest edges of the footprints by a number of "blank" elements. The two defining equations are,

$$d_f = p\Delta_{\boldsymbol{y}}$$

and

$$\Delta f - d_f = q\Delta_{\boldsymbol{y}}$$

$$\Longrightarrow \Delta f = (p+q)\Delta_{\boldsymbol{y}}$$

where $p$ and $q$ are nonnegative integers.

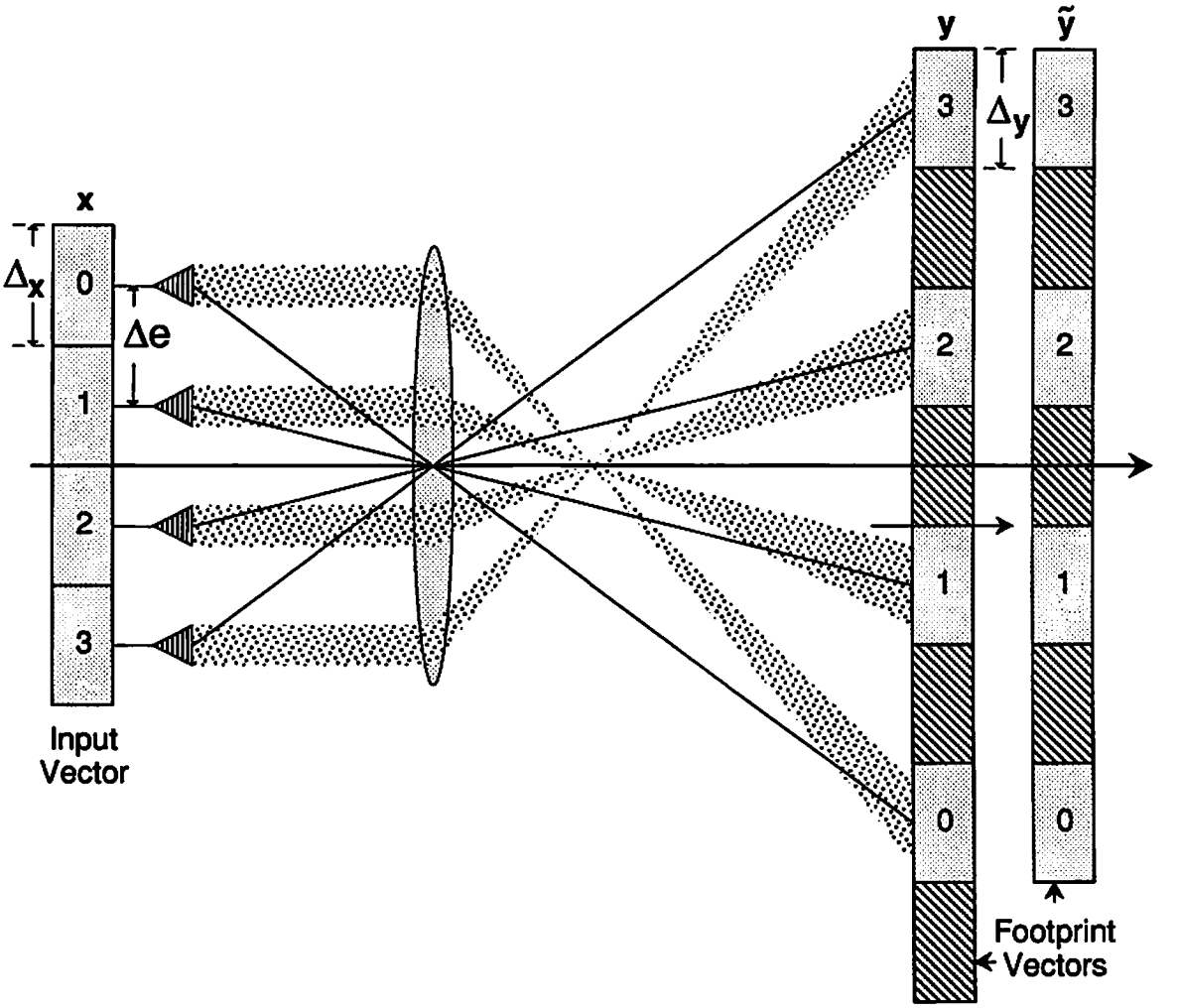



Figure 5.2: Correspondence of emitter and footprint arrays to their representative vectors.

Each element $x(i)$ of the input vector $\underset{n}{\boldsymbol{x}}$ is responsible for generating a segment $\underset{p+q}{\boldsymbol{y}}^{(n-i)}$ of the footprint vector, i.e.,

$$\underset{p+q}{\boldsymbol{y}}^{(n-i)} = \underset{p+q}{\boldsymbol{m}} \otimes x(i) = x(i) \otimes \underset{p+q}{\boldsymbol{m}}$$

where $\underset{p+q}{\boldsymbol{m}}$ is an *expansion vector* which contains $p$ contiguous non-blank elements representing the footprint area and $q$ blank elements representing the separation.

The expansion factor $\nu = p + q$ is related to the magnification $\mu$ as follows

$$\begin{aligned} \nu = p + q &= \frac{d_f}{\Delta y} + \frac{\Delta f - d_f}{\Delta y} \\ &= \frac{\Delta f}{\Delta y} \\ \Longleftrightarrow \nu &= \mu \frac{\Delta e}{\Delta y} \end{aligned} \tag{5-2}$$

*Assumption 2:* $\nu = \mu \Longrightarrow \Delta y = \Delta e$ . The total length of the footprint vector is a contention issue as there are two conceivable lengths, $\nu(n-1)$ or $\nu n$. This depends on whether the last footprint value is followed by blank elements as illustrated in Fig. 5.2. Both output vectors $\boldsymbol{y}$ and $\tilde{\boldsymbol{y}}$ are valid, but the former provides a simpler mathematical generation while the latter provides a common axis with the detector array.

When the input vector and lens axes coincide, the output vector, $\boldsymbol{y}$, is expressed as

$$\begin{aligned} \underset{\nu n}{\boldsymbol{y}} &= \sum_{k=0}^{n-1} \underset{n}{\boldsymbol{u}}^{(k)} \otimes \underset{\nu}{\boldsymbol{y}}^{(k)} \\ &= \sum_{k=0}^{n-1} \underset{n}{\boldsymbol{u}}^{(k)} \otimes (x(n-k) \otimes \underset{\nu}{\boldsymbol{m}}) \\ &= \left( \sum_{k=0}^{n-1} \underset{n}{\boldsymbol{u}}^{(k)} \otimes x(n-k) \right) \otimes \underset{\nu}{\boldsymbol{m}} \\ &= (\underset{n\ n}{\boldsymbol{J}} \boldsymbol{x}) \otimes \underset{\nu}{\boldsymbol{m}} \end{aligned} \tag{5-3}$$

where $\underset{n}{\boldsymbol{J}}$ is the *reflection matrix* to account for the imaging reversal (reflection about the axis of symmetry).

An alternative expression is

$$\underset{\nu n}{\boldsymbol{y}} = \sum_{i=0}^{n-1} \underset{\nu n \times n}{\boldsymbol{U}}^{(i\nu,i)} \underset{n\ n}{\boldsymbol{J}} \boldsymbol{x} \tag{5-4}$$

The common axis footprint vector, $\underset{\tilde{n}}{\tilde{\boldsymbol{y}}}$, is a truncation of $\boldsymbol{y}$,

$$\underset{\tilde{n}}{\tilde{\boldsymbol{y}}} = \sum_{i=0}^{\tilde{n}-1} \underset{\tilde{n}\times\nu n}{\boldsymbol{U}^{(i,i)}} \underset{\nu n}{\boldsymbol{y}} \tag{5-5}$$

where $\tilde{n} = \nu n - q$, and $\underset{r\times s}{\boldsymbol{U}^{(i,j)}}$ is the elementary matrix. Hereafter, the footprint vector refers to the common axis footprint vector, $\underset{\tilde{n}}{\tilde{\boldsymbol{y}}}$.

**Lens Displacement.** When the lens axis is displaced from the emitter axis by an amount $h$, then the footprint axis is displaced from the lens axis by an amount $h\mu$. The displacement of the footprint axis with respect to the emitter axis is

$$g = h(\mu + 1)$$

as illustrated in Fig. 5.3. In the following modeling description, all displacements are measured from the emitter array axis—the *reference axis.*

Consider a vector, $\boldsymbol{y}'$, which shares a common vector axis with the input and is large enough to include the footprint vector, $\underset{\tilde{n}}{\tilde{\boldsymbol{y}}}$. The common axis output vector, $\boldsymbol{y}'$, contains a minimum of $\tilde{n}$ output elements plus a number of "blank" elements induced by the lens offset. The number of output elements corresponding to the "offset side" of the plane is: half the length of footprint vector vector ($\tilde{n}/2$) plus $|g|\ \Delta y$ units. Thus, the size of the common axis output vector, $\boldsymbol{y}'$, is

$$\begin{aligned} n' &= 2[|g| + \tilde{n}/2] \\ &= 2\left(|h|(\nu+1) + \tilde{n}/2\right) \\ &= 2|h|(\nu+1) + \tilde{n}. \end{aligned}$$

The exact position of the output vector, $\boldsymbol{y}$, can be specified by premultiplying $\boldsymbol{y}$ with an identity matrix embedded within a larger zero matrix,

$$\underset{n'}{\boldsymbol{y}'} = \sum_{i=0}^{\tilde{n}-1} \underset{n'\times\tilde{n}}{\boldsymbol{U}^{(i+b',i)}} \underset{\tilde{n}}{\tilde{\boldsymbol{y}}} \tag{5-6}$$

where

$$\begin{aligned} b' &= |g| - g \\ &= (|h| - h)(\nu + 1). \end{aligned} \tag{5-7}$$

The sign of $h$ determines whether $\tilde{\boldsymbol{y}}$ is located in the lower or upper portion of $\boldsymbol{y}'$.

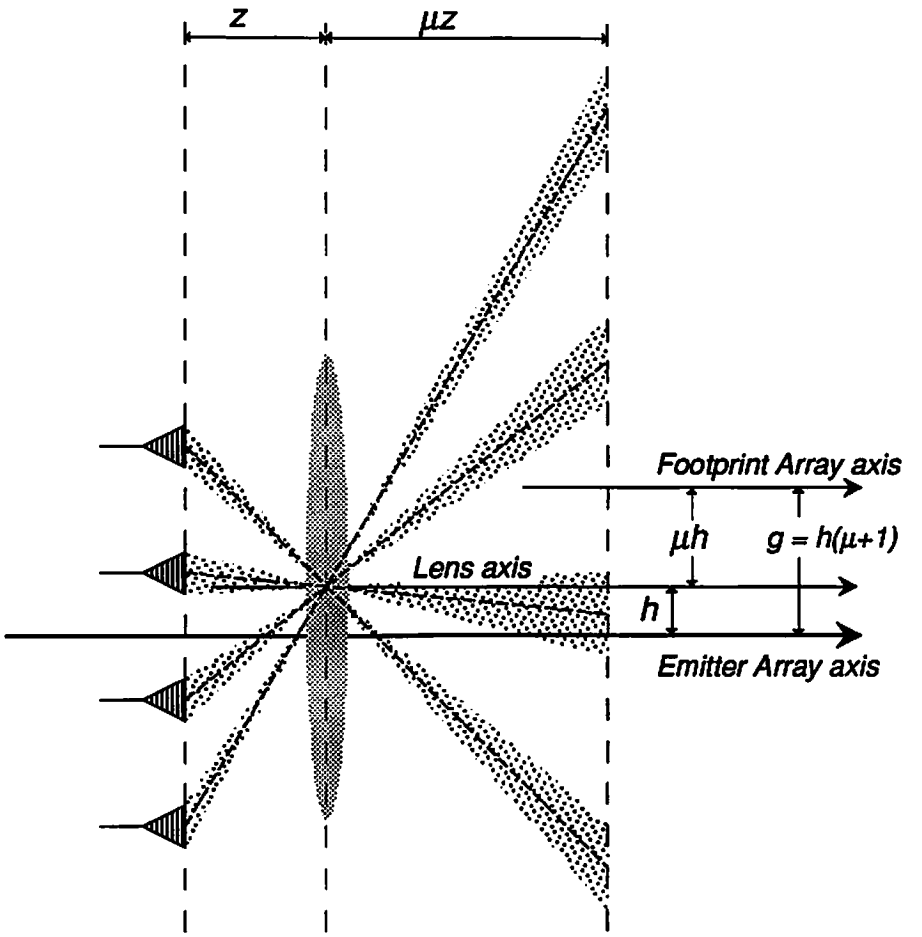



Figure 5.3: Effects of lens offset.

**Multilens systems** Combining the effects of two or more lenses yields some interesting interconnection patterns.

*Assumption 3:* All lenses are of the same focal length and lie within the same plane.

Combining the effects of multiple lenses yields an aggregate vector which may be larger than a single vector $y'_l$ due to a given offset $h_l$. The contribution of each lens to this aggregate vector, is defined by

$$y''_{n''_l} = \sum_{i=0}^{n'_l-1} {}_{n''}U_{x n'_l}^{(i+b''_l,i)} y'_{n'_l} \tag{5-8}$$

where

$$b_l'' = \frac{n'' - n_l'}{2} \tag{5-9}$$

An aggregate footprint vector, $y'''$, contains the output vectors output by each lens and shares a common axis with the input vector. The length of $y'''$ is the same as that of the $y'$ which corresponds to the lens which is displaced the farthest distance ($h_{max}$).

DEFINITION 23: *For L lenses located at unit distances of $h_0, h_1, \cdots, h_{L-1}$ from the input vector axis, the aggregate output vector is defined as,*

$$\underset{n''}{y'''} = \sum_{l=0}^{L-1} \sum_{j=0}^{n_l'-1} \underset{n'' \times n_l'}{U}^{(j+b_l'',j)} \underset{n_l'}{y_l'} \tag{5-10}$$

*where* $n'' = 2|h_{max}|(\nu+1) + \tilde{n}$, $|h_{max}| = max\{|h_0|, |h_1|, \cdots, |h_{L-1}|\}$ *and*

$$\begin{aligned} b_l'' &= \frac{1}{2}(n'' - n_l') \\ &= \frac{1}{2}\Big(2|h_{max}|(\nu+1) + \tilde{n} - (2|h_l|(\nu+1) + \tilde{n})\Big) \\ &= (|h_{max}| - |h_l|)(\nu+1) \end{aligned} \tag{5-11}$$

The use of more than one lens implies that the output vector contains more than one copy of the input elements. The intention of this multilens implementation is to overlay copies of offset footprint vectors in such a way that a select portion of the aggregate footprint vector contains the desired pattern. In the described model, the desired area is the subvector which shares the same size and vector axis as the input. This requires that the desired output vector $x'$ be extracted from the aggregate output vector by the operation,

$$\underset{n}{x'} = \sum_{j=0}^{n-1} \underset{n \times n''}{U}^{(j,j+b''')} \underset{n''}{y'''} \tag{5-12}$$

where

$$\begin{aligned} b''' &= \frac{1}{2}(n'' - n) \\ &= \frac{1}{2}\Big(2|h_{max}|(\nu+1) + \tilde{n} - n\Big) \\ &= |h_{max}|(\nu+1) + \frac{1}{2}(\tilde{n} - n) \end{aligned} \tag{5-13}$$

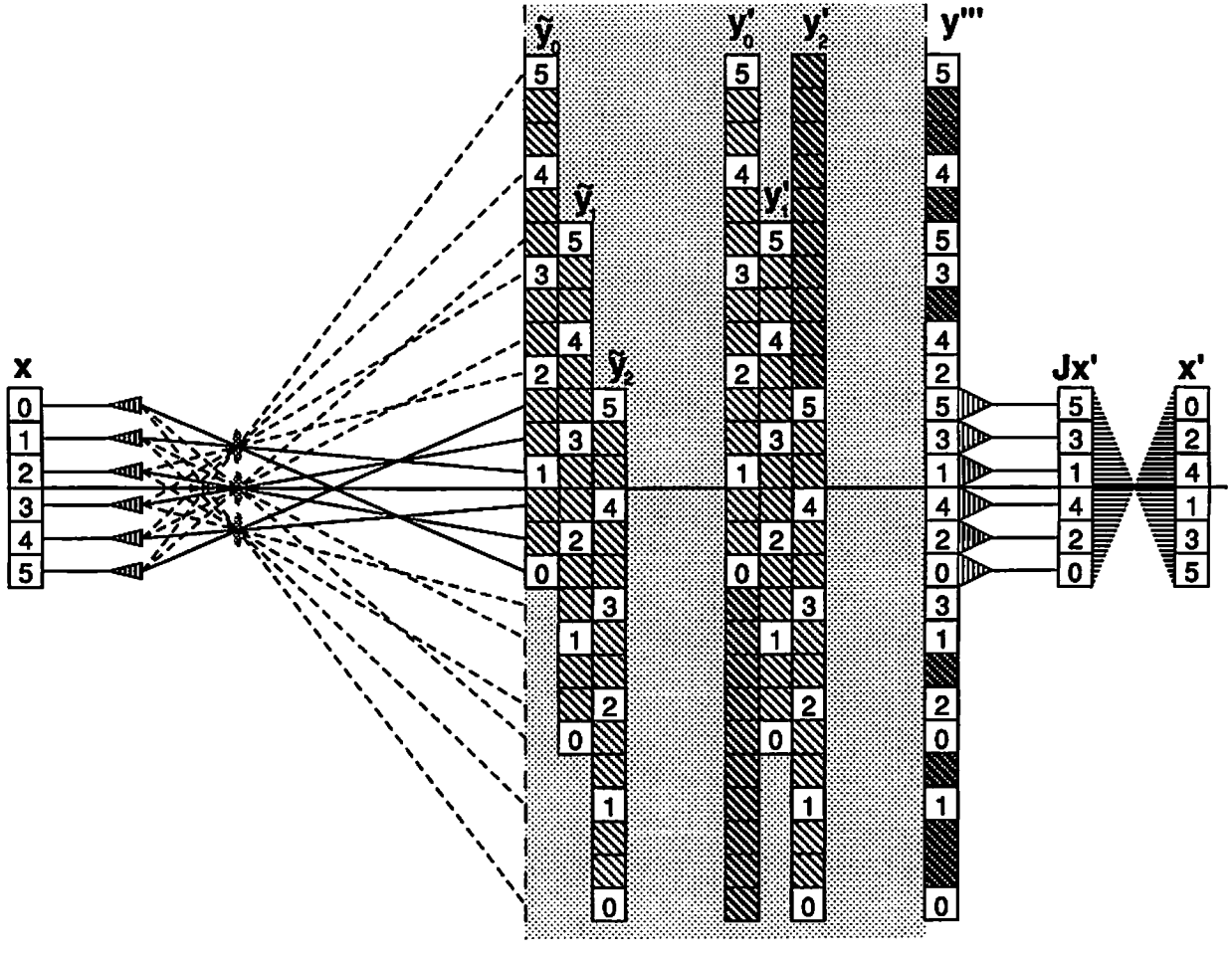



Figure 5.4: Optical implementation of an inverse perfect shuffle of a 6-element input vector

Fig. 5.4 illustrates the contributions of three lenses to the aggregate footprint vector, $\boldsymbol{y}'''$ as well as the extracted detector vector $\boldsymbol{x}'$.

By successively expanding each of the contributions to the aggregate footprint vector, we obtain

$$
\begin{aligned}
\underset{n''}{\boldsymbol{y}'''} &= \sum_{l=0}^{L-1} \sum_{j=0}^{n_l'-1} \underset{n'' \times n_l'}{\boldsymbol{U}}^{(j+b_l'',j)} \underset{n_l'}{\boldsymbol{y}_l'} \\
&= \sum_{l=0}^{L-1} \sum_{j=0}^{n_l'-1} \underset{n'' \times n_l'}{\boldsymbol{U}}^{(j+b_l'',j)} \left( \sum_{i=0}^{\tilde{n}-1} \underset{n_l' \times \tilde{n}}{\boldsymbol{U}}^{(i+b_l',i)} \underset{\tilde{n}}{\tilde{\boldsymbol{y}}} \right) \\
&= \sum_{l=0}^{L-1} \sum_{i=0}^{\tilde{n}-1} \sum_{j=0}^{n_l'-1} \underset{n'' \times n_l'}{\boldsymbol{U}}^{(j+b_l'',j)} \underset{n_l' \times \tilde{n}}{\boldsymbol{U}}^{(i+b_l',i)} \underset{\tilde{n}}{\tilde{\boldsymbol{y}}} \\
&= \sum_{l=0}^{L-1} \sum_{i=0}^{\tilde{n}-1} \sum_{j=0}^{n_l'-1} \underset{n''}{\boldsymbol{u}}^{(j+b_l'')} \underset{n_l'}{\boldsymbol{u}}^{(j)^T} \underset{n_l'}{\boldsymbol{u}}^{(i+b_l')} \underset{\tilde{n}}{\boldsymbol{u}}^{(i)^T} \underset{\tilde{n}}{\tilde{\boldsymbol{y}}} \\
&= \sum_{l=0}^{L-1} \sum_{i=0}^{\tilde{n}-1} \sum_{j=0}^{n_l'-1} \underset{n''}{\boldsymbol{u}}^{(j+b_l'')} \underbrace{\underset{n_l'}{\boldsymbol{u}}^{(j)^T} \underset{n_l'}{\boldsymbol{u}}^{(i+b_l')}}_{\neq 0 \text{ iff } j=i+b_l'} \underset{\tilde{n}}{\boldsymbol{u}}^{(i)^T} \underset{\tilde{n}}{\tilde{\boldsymbol{y}}} \\
&= \sum_{l=0}^{L-1} \sum_{i=0}^{\tilde{n}-1} \underset{n''}{\boldsymbol{u}}^{(i+b_l'+b_l'')} \underset{\tilde{n}}{\boldsymbol{u}}^{(i)^T} \underset{\tilde{n}}{\tilde{\boldsymbol{y}}} \\
&= \sum_{l=0}^{L-1} \sum_{i=0}^{\tilde{n}-1} \underset{n''}{\boldsymbol{u}}^{(i+b_l'+b_l'')} \underset{\tilde{n}}{\boldsymbol{u}}^{(i)^T} \underset{\tilde{n}}{\tilde{\boldsymbol{y}}}
\end{aligned}
$$

The extracted detector vector can now be written as

$$
\begin{aligned}
\underset{n}{\boldsymbol{x}'} &= \sum_{j=0}^{n-1} \underset{n \times n''}{\boldsymbol{U}}^{(j,j+b''')} \underset{n''}{\boldsymbol{y}'''} \\
&= \sum_{j=0}^{n-1} \underset{n \times n''}{\boldsymbol{U}}^{(j,j+b''')} \sum_{l=0}^{L-1} \sum_{i=0}^{\tilde{n}-1} \underset{n''}{\boldsymbol{u}}^{(i+b_l'+b_l'')} \underset{\tilde{n}}{\boldsymbol{u}}^{(i)^T} \underset{\tilde{n}}{\tilde{\boldsymbol{y}}} \\
&= \sum_{j=0}^{n-1} \sum_{l=0}^{L-1} \sum_{i=0}^{\tilde{n}-1} \underset{n}{\boldsymbol{u}}^{(j)} \underset{n''}{\boldsymbol{u}}^{(j+b''')^T} \underset{n''}{\boldsymbol{u}}^{(i+b_l'+b_l'')} \underset{\tilde{n}}{\boldsymbol{u}}^{(i)^T} \underset{\tilde{n}}{\tilde{\boldsymbol{y}}} \\
&= \sum_{j=0}^{n-1} \sum_{l=0}^{L-1} \sum_{i=0}^{\tilde{n}-1} \underset{n}{\boldsymbol{u}}^{(j)} \underbrace{\underset{n''}{\boldsymbol{u}}^{(j+b''')^T} \underset{n''}{\boldsymbol{u}}^{(i+b_l'+b_l'')}}_{\neq 0 \text{ iff } j+b'''=i+b_l'+b_l''} \underset{\tilde{n}}{\boldsymbol{u}}^{(i)^T} \underset{\tilde{n}}{\tilde{\boldsymbol{y}}} \\
&= \sum_{j=0}^{n-1} \sum_{l=0}^{L-1} \underset{n}{\boldsymbol{u}}^{(j)} \underset{\tilde{n}}{\boldsymbol{u}}^{(j+b'''-b_l''-b_l')^T} \underset{\tilde{n}}{\tilde{\boldsymbol{y}}}
\end{aligned}
$$

Let

$$\begin{aligned} B_l &= b''' - b_l'' - b_l' \\ &= |h_{max}|(\nu+1) + \frac{1}{2}(\tilde{n}-n) \\ &\quad -(|h_{max}| - |h_l|)(\nu+1) \\ &\quad -(|h_l| - h_l)(\nu+1) \\ &= \frac{1}{2}(\tilde{n}-n) + h_l(\nu+1) \\ &= \frac{1}{2}(\nu n - q - n) + h_l(\nu+1) \\ &= h_l(\nu+1) - \frac{q}{2} + \frac{n}{2}(\nu-1) \end{aligned} \tag{5-14}$$

(5–15)

Therefore,

$$\begin{aligned} \underset{n}{x'} &= \sum_{j=0}^{n-1} \sum_{l=0}^{L-1} \underset{n}{u^{(j)}} \underset{\tilde{n}}{u^{(j+B_l)^T}} \underset{\tilde{n}}{\tilde{y}} \\ &= \sum_{j=0}^{n-1} \sum_{l=0}^{L-1} \underset{n}{u^{(j)}} \underset{\tilde{n}}{u^{(j+B_l)^T}} \sum_{i=0}^{\tilde{n}-1} \underset{\tilde{n}\times\nu n}{U^{(i,i)}} \underset{\nu n}{y} \\ &= \sum_{j=0}^{n-1} \sum_{l=0}^{L-1} \sum_{i=0}^{\tilde{n}-1} \underset{n}{u^{(j)}} \underset{\tilde{n}}{u^{(j+B_l)^T}} \underset{\tilde{n}}{u^{(i)}} \underset{\nu n}{u^{(i)^T}} \underset{\nu n}{y} \\ &= \sum_{j=0}^{n-1} \sum_{l=0}^{L-1} \sum_{i=0}^{\tilde{n}-1} \underset{n}{u^{(j)}} \underbrace{\underset{\tilde{n}}{u^{(j+B_l)^T}} \underset{\tilde{n}}{u^{(i)}}}_{\neq 0 \text{ iff } j+B_l=i} \underset{\nu n}{u^{(i)^T}} \underset{\nu n}{y} \\ &= \sum_{j=0}^{n-1} \sum_{l=0}^{L-1} \underset{n}{u^{(j)}} \underset{\nu n}{u^{(j+B_l)^T}} \underset{\nu n}{y} \end{aligned}$$

Therefore,

$$\begin{aligned} \underset{n}{x'} &= \sum_{j=0}^{n-1} \sum_{l=0}^{L-1} \underset{n}{u^{(j)}} \underset{\nu n}{u^{(j+B_l)^T}} \underset{\nu n}{y} \\ &= \sum_{j=0}^{n-1} \sum_{l=0}^{L-1} \underset{n}{u^{(j)}} \underset{\nu n}{u^{(j+B_l)^T}} \left( \sum_{i=0}^{n-1} \underset{\nu n\times n}{U^{(i\nu,i)}} \underset{n\ n}{Jx} \right) \end{aligned}$$

$$= \sum_{j=0}^{n-1} \sum_{l=0}^{L-1} \sum_{i=0}^{n-1} \underset{n}{u^{(j)}} \underbrace{\underset{\nu n}{u^{(j+B_l)^T}} \underset{\nu n}{u^{(i\nu)}}}_{\neq 0 \text{ iff } j+B_l=i\nu} \underset{n}{u^{(i)^T}} \underset{n\,n}{Jx}$$

$$= \sum_{l=0}^{L-1} \sum_{i=0}^{n-1} \underset{n}{u^{(i\nu-B_l)}} \underset{n}{u^{(i)^T}} \underset{n\,n}{Jx}$$

*Assumption 4: $n = r\nu$* The number of elements is a integral multiple of the expansion factor $\nu$. Then, the index $i = mr + k \;\; \forall \;\; 0 \leq m \leq \nu - 1$ and $0 \leq k \leq r - 1$, i.e.,

$$\sum_{i=0}^{n-1} \equiv \sum_{m=0}^{\nu-1} \sum_{k=0}^{r-1}$$

and

$$\begin{aligned}
\underset{n}{x'} &= \sum_{l=0}^{\nu-1} \sum_{m=0}^{\nu-1} \sum_{k=0}^{r-1} \underset{r\nu}{u^{((mr+k)\nu-B_l)}} \underset{r\nu}{u^{(mr+k)^T}} \underset{n\,n}{Jx} \\
&= \sum_{l=0}^{\nu-1} \sum_{m=0}^{\nu-1} \sum_{k=0}^{r-1} \underset{r\nu}{u^{((mr\nu-B_l)+k\nu)}} \underset{r\nu}{u^{(mr+k)^T}} \underset{n\,n}{Jx} \\
&= \sum_{l=0}^{\nu-1} \sum_{m=0}^{\nu-1} \sum_{k=0}^{r-1} \underset{r\nu}{u^{(k\nu+(mn-B_l))}} \underset{r\nu}{u^{(mr+k)^T}} \underset{n\,n}{Jx}
\end{aligned}$$

Note that $\underset{r}{u^{(a)}} \otimes \underset{\nu}{u^{(b)}} = \underset{r\nu}{u^{(a\nu+b)}}$,

Thus,

$$\begin{aligned}
\underset{n}{x'} &= \sum_{l=0}^{L-1} \sum_{m=0}^{\nu-1} \sum_{k=0}^{r-1} \underset{k\nu+(mn-B_l)}{u^{(r\nu)}} \underset{r\nu}{u^{(mr+k)^T}} \underset{n\,n}{Jx} \\
&= \sum_{l=0}^{L-1} \sum_{m=0}^{\nu-1} \sum_{k=0}^{r-1} \left( \underset{r}{u^{(k)}} \otimes \underset{mn-B_l}{u^{(\nu)}} \right) \left( \underset{\nu}{u^{(m)}} \otimes \underset{r}{u^{(k)}} \right)^T \underset{n\,n}{Jx} \\
\textit{[by (2-5)]} \qquad &= \sum_{l=0}^{L-1} \sum_{m=0}^{\nu-1} \sum_{k=0}^{r-1} \left( \left( \underset{r}{u^{(k)}} \underset{\nu}{u^{(m)^T}} \right) \otimes \left( \underset{mn-B_l}{u^{(\nu)}} \underset{r}{u^{(k)^T}} \right) \right) \underset{n\,n}{Jx} \\
&= \sum_{l=0}^{L-1} \sum_{m=0}^{\nu-1} \sum_{k=0}^{r-1} \left( \underset{r,\nu}{U^{(k,m)}} \otimes \underset{\nu,r}{U^{(mn-B_l,k)}} \right) \underset{n\,n}{Jx} \\
\Longrightarrow \underset{n}{x'} &= \sum_{l=0}^{L-1} \sum_{m=0}^{\nu-1} \sum_{k=0}^{r-1} \left( (\underset{\nu,r}{U^{(m,k)}})^T \otimes \underset{\nu,r}{U^{(mn-B_l,k)}} \right) \underset{n\,n}{Jx}
\end{aligned}$$

*Assumption 5: $L = \nu$* There are as many lenses as the expansion factor $\nu$.

Note that if $mn - B_l = m$ for $l = m$, then

$$\underset{n}{x'} = \sum_{m=0}^{\nu-1}\sum_{k=0}^{r-1}\left((\underset{\nu,r}{U}^{(m,k)})^T \otimes \underset{\nu,r}{U}^{(m,k)}\right)\underset{n\ n}{J}\underset{n}{x}$$

*[by (2-1)]* $$\triangleq \underset{n\ n}{S_{[\nu|r]}J}\underset{n}{x}$$

$$\Longrightarrow \boxed{\underset{n}{x'} = \underset{n}{J}\underset{n}{S_{[\nu|\frac{n}{\nu}]}}\underset{n}{x}}$$

The restriction $mn - B_l = m$ for $0 \leq l = m \leq \nu - 1$, will provide the proper location of the lenses in order to achieve this shuffle, i.e.,

$$\begin{aligned}
mn - B_m &= m \\
m(n-1) &= B_m \\
\textit{[by (5-14)]} \qquad m(n-1) &= h_m(\nu+1) - \frac{q}{2} + \frac{n}{2}(\nu-1) \\
\Longrightarrow h_m(\nu+1) &= m(n-1) + \frac{q}{2} - \frac{n}{2}(\nu-1) \\
\Longrightarrow h_m &= \frac{1}{(\nu+1)}[m(n-1) + \frac{q}{2} - \frac{n}{2}(\nu-1)]
\end{aligned}$$

*Assumption 6:* $p = 1 \Longrightarrow q = \nu - 1$

$$\begin{aligned}
\Longrightarrow h_m &= \frac{1}{2(\nu+1)}[2m(n-1) + (\nu-1) - n(\nu-1)] \\
&= \frac{1}{2(\nu+1)}[2m(n-1) - (n-1)(\nu-1)] \\
&= \frac{(n-1)}{2(\nu+1)}[2m - (\nu-1)]
\end{aligned}$$

Therefore, the set of lens offsets $\mathcal{H}$ is

$$\mathcal{H} = \left\{h : h = \frac{(n-1)(2m-\nu+1)}{2(\nu+1)},\ m = 0, 1, \ldots, \nu-1\right\}$$

The above results are summarized into the following statement:

THEOREM 31: *The $\sigma_{\nu|\frac{n}{\nu}}$ shuffle of an n-element vector may be implemented optically by using an array of $\nu$ thin lenses of equal focal length displaced from the vector axis by offsets*

$$h_k = \frac{(n-1)(2k-\nu+1)}{2(\nu+1)},\ k = 0,1,\ldots,\nu-1 \qquad (5\text{–}16)$$

Note that the lens placement is dependent on the length of the input vector.

*Example 1:* For $n = 6$, the inverse perfect shuffle, $S_6^{-1}$, may be implemented with three lenses. This was depicted in Fig. 5.4. The lens positions are given, respectively, as $h_0 = -\frac{5}{4},\ h_1 = 0,\ h_2 = \frac{5}{4}$.

*Example 2:* For $n = 8$, the inverse perfect shuffle, $S_8^{-1}$, may be implemented with four lenses as depicted in Fig. 5.5. The positioning of the lenses is in accordance with the offset rule, i.e.,

| *Lens* | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| *Offset ($h_k$)* | -2.1 | -0.7 | 0.7 | 2.1 |

*Example 3:* For $n = 16$, the $S_{16}$, $S_{[4|4]}$, and $S_{16}^{-1}$ shuffles may be optically implemented using two, four and eight lenses respectively, and positioned as follows:

- For two lenses: $h = \mp 2.5$
- For four lenses: $h = \mp 4.5$ and $h = \mp 1.5$
- For eight lenses: $h = \mp 35/6$, $h = \mp 25/6$, $h = \mp 15/6$ and $h = \mp 5/6$

### Two-Dimensional system

Two-dimensional optical interconnections can also be achieved using an array of lens, although the lenses may not be tessellated in a rectangular manner.

An analysis analogous to that of the one-dimensional case will yield lens locations appropriate for achieving a number of 2-D shuffles. The implementation of a 2-D separable perfect shuffle using four lenses in a plane has been demonstrated by Sawchuk

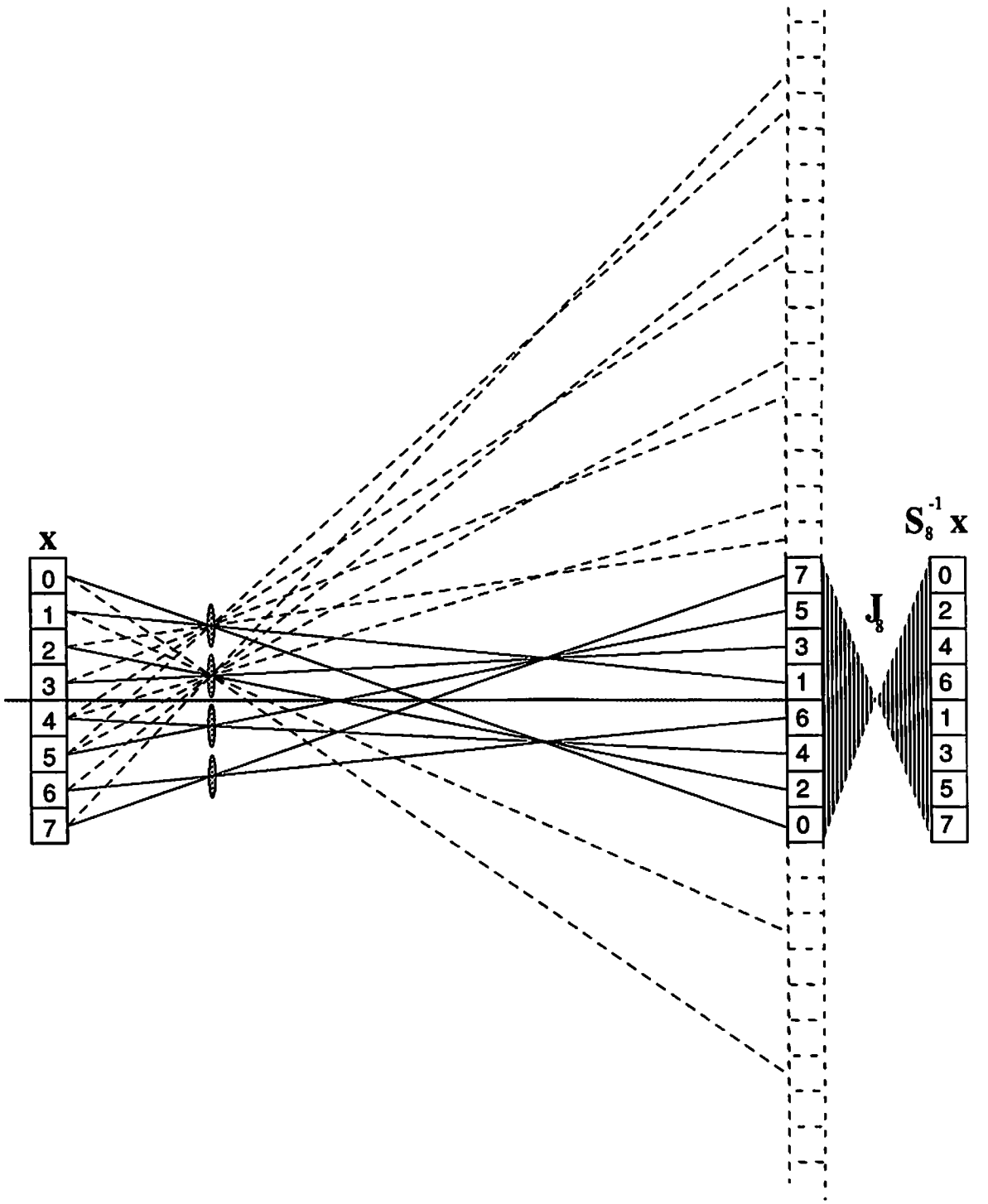



Figure 5.5: Optical implementation of an inverse perfect shuffle of an 8-element input vector

and Glaser [53]. Fig. 5.6 illustrates the ray tracing for a $16 \times 16$ 2-D separable perfect shuffle.

There are four points of concentration which correspond to the centers of the lenses.

### 5.1.2 Shuffling with Holograms

*Computer generated holograms (CGH)* comprise another class of devices which may be used to implement shuffles and all other possible permutations of two-dimensional arrangements of light sources.

Consider an isoplanar homosyndetic architecture that has been reduced to one stage of processing/communication. In order to emulate the full structure, a feedback path is needed that interconnects the output light sources with the input light receptors. When the sources and receptors are located on the same side of the processing plane, then the loop back mechanism may be simplified by means of CGHs. This may make the physical architecture more compact.

Both major types of CGHs, *transmissive* and *reflective*, are well suited for such an application. This is illustrated schematically in Fig. 5.7 and Fig. 5.8.

In Fig. 5.7, the hologram reflects the normal incident beams to receptors of target processing elements in the processing plane. The angle of reflection in encoded into the hologram by computer. Any fixed interconnection pattern may be realized by this technique.

In Fig. 5.8, the incident beams are transmitted through the hologram, but diverted by such an angle so that their subsequent reflection by a simple mirror directs them to another part of the hologram where they are normalized to the target receptors.

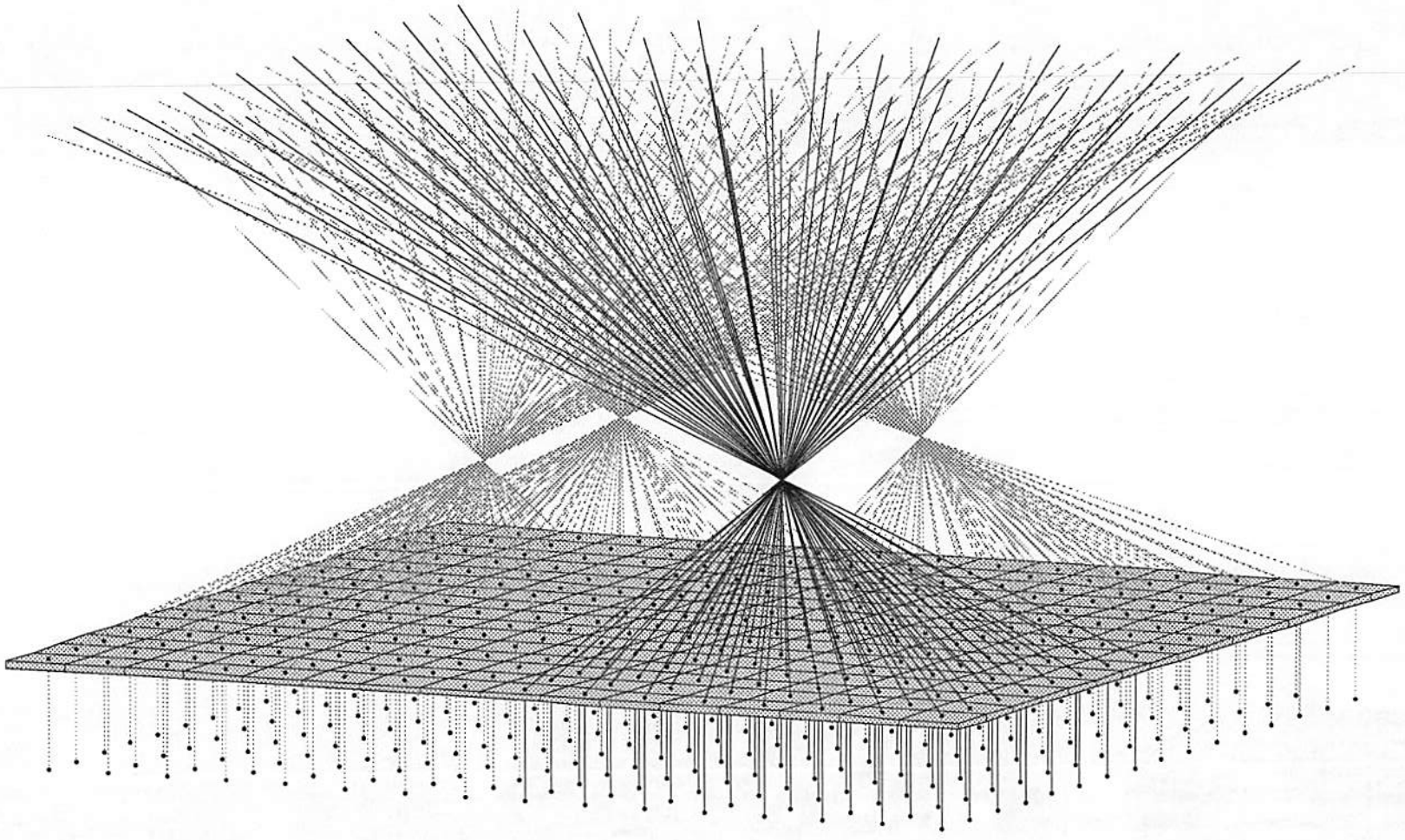


Figure 5.6: Optical implementation of a two dimensional separable perfect shuffle of a $16 \times 16$ input array

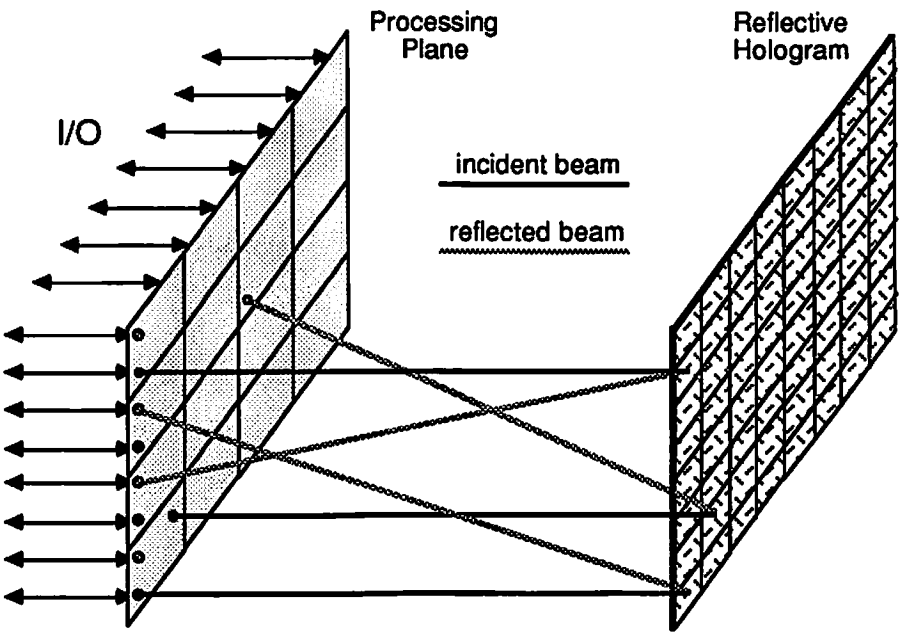



Figure 5.7: Free-space interconnections by reflective CGH

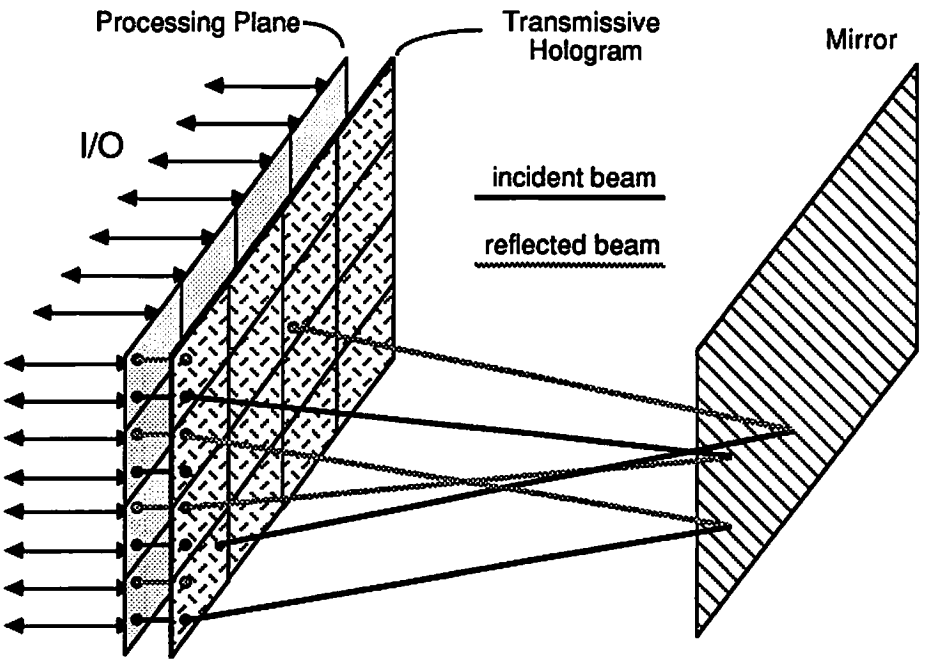



Figure 5.8: Free-space interconnections by transmissive CGH

# Chapter 6

# Conclusions and Direction of Future Work

This dissertation presented a mathematical framework by which it is possible to analyze and synthesize isoplanar homosyndetic processing structures that may be used—amongst other applications—for image processing.

This work was spurred by:

- the potential near-term availability of optical computing structures, especially *free-space* interconnection techniques, and
- the immediate need for real-time processing of large digitized images.

The possibilities of an optical computing technology motivate the researcher to step back and take a look at the broad picture of parallel computing. Popular past and present design techniques appear painfully inadequate for efficiently mapping solutions to massively parallel computing structures.

A universal parallel design methodology, though highly desirable, may never materialize. In the interim, specialized design techniques for broad subclasses of parallel computing problems seems like a reasonable alternative. In this vein, the use of Kronecker and array algebra is helpful in the design of two- and three-dimensional multistage architectures like the *Omega* computers which employ shuffled interconnections. The

contributions of this dissertation as well as expected and suggested follow-up work are summarized in the following two sections.

## 6.1 Research Contributions

This thesis introduced new definitions and design methodologies to the field of parallel processing. These contributions include:

- New terms conveying more precision with regards to the dimensionality and design of massively parallel computing structures.
- Consolidation of a mathematical outer-product algebra based on Kronecker products, shuffles and *array algebra* concepts.
- Derivation of useful identities.
- Expanded interpretation of array algebra.
- Classification of one- and two-dimensional isoplanar homosyndetic computing structures and derivation of their mathematical canonical forms in both Kronecker and array algebra.
- Demonstration of the applicability of the methodology to two-dimensional task-to-flow mapping. Examples included one-to-one matrix transformations as well as a subclass of unitary transformations.
- The mathematical analysis of the optical implementation (with simple lenses) of a perfect shuffle in one-dimensional space. The use of Kronecker and array algebra proved their usefulness beyond that of *task-to-flow* mapping.

## 6.2 Future Work

The comprehensive analysis and synthesis of isoplanar homosyndetic architectures by means of Kronecker and array algebras requires a continuation of the work presented in this dissertation. Specifically, with regards to array algebra,

- The index variables must be allowed to decrement as well as increment in order to facilitate characterization of "reversal" operations such as that represented by the *reflection matrix* $\boldsymbol{J}$. A notation for indicating this reverse index counting must be developed as well as identities relating "forward" and "reverse" indexed arrays.

- Uncommon types of matrix operations which have not been presented here need to be stated in array algebra so that more operations and tasks can be easily derived.

- Achieving a fluency in array/Kronecker algebra, would be aided by developing more solution paradigms and "rules-of-thumb". As with any algebra, solutions can fall out quite easily and elegantly by using appropriate "tricks".

In regards to task-to-flow mapping problems, an examination of tasks which can be mapped to 2-D isonodal—arbitrary kernel weights—architectures should be further pursued in order to broaden the applicability of such structures.

In general, the more experience one accumulates by working out several problems in Kronecker and array algebra the more insight will be gained into the various tasks and ways of mapping them to homosyndetic isoplanar and other types of parallel architectures.

# Appendix A

The benefits of using array algebra can be demonstrated by working through two different proofs of the pseudo-commutativity theorem (Theorem 6),

$$\underset{p\times q}{\boldsymbol{A}} \otimes \underset{r\times t}{\boldsymbol{B}} = \boldsymbol{S}^{-1}_{[p|r]}(\underset{r\times t}{\boldsymbol{B}} \otimes \underset{p\times q}{\boldsymbol{A}})\boldsymbol{S}_{[q|t]}. \tag{1-1}$$

First, a Kronecker algebra based proof is detailed, as outlined by [7]. It is followed by an array algebra based proof. The comparison is quite dramatic.

*Proof:* The right-hand side of (1–1) can be reduced by first expanding the shuffles by their definition:

$$\begin{aligned}
&\boldsymbol{S}^{-1}_{[p|r]}(\underset{r\times t}{\boldsymbol{B}} \otimes \underset{p\times q}{\boldsymbol{A}})\boldsymbol{S}_{[q|t]} \\
= \quad &\left(\sum_{i=0}^{p-1}\sum_{k=0}^{r-1} \underset{p\times r}{\boldsymbol{U}^{(i,k)}} \otimes \underset{r\times p}{\boldsymbol{U}^{(k,i)}}\right)(\underset{r\times t}{\boldsymbol{B}} \otimes \underset{p\times q}{\boldsymbol{A}})\left(\sum_{m=0}^{q-1}\sum_{n=0}^{t-1} \underset{t\times q}{\boldsymbol{U}^{(n,m)}} \otimes \underset{q\times t}{\boldsymbol{U}^{(m,n)}}\right) \\
= \quad &(\underset{p\times r}{\boldsymbol{U}^{(00)}} \otimes \underset{r\times p}{\boldsymbol{U}^{(00)}}) \\
&\quad \left[(\underset{r\times t}{\boldsymbol{B}} \otimes \underset{p\times q}{\boldsymbol{A}})(\underset{t\times q}{\boldsymbol{U}^{(00)}} \otimes \underset{q\times t}{\boldsymbol{U}^{(00)}}) + (\underset{r\times t}{\boldsymbol{B}} \otimes \underset{p\times q}{\boldsymbol{A}})(\underset{t\times q}{\boldsymbol{U}^{(01)}} \otimes \underset{q\times t}{\boldsymbol{U}^{(10)}}) + \cdots\right]
\end{aligned}$$

$$
\begin{aligned}
&+ \ (\underset{p\times r}{U}^{(01)} \otimes \underset{r\times p}{U}^{(10)}) \\
&\qquad \left[(\underset{r\times t}{B} \otimes \underset{p\times q}{A})(\underset{t\times q}{U}^{(00)} \otimes \underset{q\times t}{U}^{(00)}) + (\underset{r\times t}{B} \otimes \underset{p\times q}{A})(\underset{t\times q}{U}^{(01)} \otimes \underset{q\times t}{U}^{(10)}) + \cdots\right] \\
&\vdots \\
&+ \ (\underset{p\times r}{U}^{((p-1)(r-1))} \otimes \underset{r\times p}{U}^{((p-1)(r-1))}) \\
&\qquad \left[(\underset{r\times t}{B} \otimes \underset{p\times q}{A})(\underset{t\times q}{U}^{(00)} \otimes \underset{q\times t}{U}^{(00)}) + (\underset{r\times t}{B} \otimes \underset{p\times q}{A})(\underset{t\times q}{U}^{(01)} \otimes \underset{q\times t}{U}^{(10)}) + \cdots\right]
\end{aligned}
$$

$$
\begin{aligned}
= \quad & (\underset{p\times r}{U}^{(00)} \otimes \underset{r\times p}{U}^{(00)}) \left[(\underset{r\times t}{B}\,\underset{t\times q}{U}^{(00)} \otimes \underset{p\times q}{A}\,\underset{q\times t}{U}^{(00)}) + (\underset{r\times t}{B}\,\underset{t\times q}{U}^{(01)} \otimes \underset{p\times q}{A}\,\underset{q\times t}{U}^{(10)}) + \cdots\right] \\
&+ \ (\underset{p\times r}{U}^{(01)} \otimes \underset{r\times p}{U}^{(10)}) \left[(\underset{r\times t}{B}\,\underset{t\times q}{U}^{(00)} \otimes \underset{p\times q}{A}\,\underset{q\times t}{U}^{(00)}) + (\underset{r\times t}{B}\,\underset{t\times q}{U}^{(01)} \otimes \underset{p\times q}{A}\,\underset{q\times t}{U}^{(10)}) + \cdots\right] \\
&\vdots \\
&+ \ (\underset{p\times r}{U}^{((p-1)(r-1))} \otimes \underset{r\times p}{U}^{((p-1)(r-1))}) \\
&\qquad \left[(\underset{r\times t}{B}\,\underset{t\times q}{U}^{(00)} \otimes \underset{p\times q}{A}\,\underset{q\times t}{U}^{(00)}) + (\underset{r\times t}{B}\,\underset{t\times q}{U}^{(01)} \otimes \underset{p\times q}{A}\,\underset{q\times t}{U}^{(10)}) + \cdots\right]
\end{aligned}
$$

$$
\begin{aligned}
= \quad & \underset{p\times r}{U}^{(00)}\,\underset{r\times t}{B}\,\underset{t\times q}{U}^{(00)} \otimes \underset{r\times p}{U}^{(00)}\,\underset{p\times q}{A}\,\underset{q\times t}{U}^{(00)} + \underset{p\times r}{U}^{(00)}\,\underset{r\times t}{B}\,\underset{t\times q}{U}^{(01)} \otimes \underset{r\times p}{U}^{(00)}\,\underset{p\times q}{A}\,\underset{q\times t}{U}^{(10)} + \cdots \\
&+ \ \underset{p\times r}{U}^{(01)}\,\underset{r\times t}{B}\,\underset{t\times q}{U}^{(00)} \otimes \underset{r\times p}{U}^{(10)}\,\underset{p\times q}{A}\,\underset{q\times t}{U}^{(00)} + \underset{p\times r}{U}^{(01)}\,\underset{r\times t}{B}\,\underset{t\times q}{U}^{(01)} \otimes \underset{r\times p}{U}^{(10)}\,\underset{p\times q}{A}\,\underset{q\times t}{U}^{(10)} + \cdots \\
&\vdots \\
&+ \ \underset{p\times r}{U}^{((p-1)(r-1))}\,\underset{r\times t}{B}\,\underset{t\times q}{U}^{(00)} \otimes \underset{r\times p}{U}^{((p-1)(r-1))}\,\underset{p\times q}{A}\,\underset{q\times t}{U}^{(00)} \\
&\quad + \underset{p\times r}{U}^{((p-1)(r-1))}\,\underset{r\times t}{B}\,\underset{t\times q}{U}^{(01)} \otimes \underset{r\times p}{U}^{((p-1)(r-1))}\,\underset{p\times q}{A}\,\underset{q\times t}{U}^{(10)} + \cdots
\end{aligned}
$$

By recognizing the fact that, $\underset{r\times p}{U}^{(k,i)}\,\underset{p\times q}{A}\,\underset{q\times t}{U}^{(m,n)} = a_{im}\,\underset{r\times t}{U}^{(k,n)}$ , we can write:

$$
\begin{aligned}
& S^{-1}_{[p|r]}(\underset{r\times t}{B} \otimes \underset{p\times q}{A})S_{[q|t]} = \\
&\qquad b_{00}\,\underset{p\times q}{U}^{(00)} \otimes a_{00}\,\underset{r\times t}{U}^{(00)} + b_{00}\,\underset{p\times q}{U}^{(01)} \otimes a_{01}\,\underset{r\times t}{U}^{(00)} + b_{00}\,\underset{p\times q}{U}^{(02)} \otimes a_{02}\,\underset{r\times t}{U}^{(00)} + \cdots \\
&\quad + \ b_{10}\,\underset{p\times q}{U}^{(00)} \otimes a_{00}\,\underset{r\times t}{U}^{(00)} + b_{10}\,\underset{p\times q}{U}^{(01)} \otimes a_{01}\,\underset{r\times t}{U}^{(00)} + b_{10}\,\underset{p\times q}{U}^{(02)} \otimes a_{02}\,\underset{r\times t}{U}^{(00)} + \cdots \\
&\quad \vdots \\
&\quad + \ b_{(r-1)(t-1)}\,\underset{p\times q}{U}^{(00)} \otimes a_{00}\,\underset{r\times t}{U}^{(00)} + b_{(r-1)(t-1)}\,\underset{p\times q}{U}^{(01)} \otimes a_{01}\,\underset{r\times t}{U}^{(00)} \\
&\qquad\quad + b_{(r-1)(t-1)}\,\underset{p\times q}{U}^{(02)} \otimes a_{02}\,\underset{r\times t}{U}^{(00)} + \cdots
\end{aligned}
$$

$$
\begin{aligned}
= & \left[a_{00}\underset{p\times q}{U}^{(00)} + a_{01}\underset{p\times q}{U}^{(01)} + \cdots + a_{(p-1)(q-1)}\underset{p\times q}{U}^{((p-1)(q-1))}\right] \otimes b_{00}\underset{r\times t}{U}^{(00)} \\
+ & \left[a_{00}\underset{p\times q}{U}^{(00)} + a_{01}\underset{p\times q}{U}^{(01)} + \cdots + a_{(p-1)(q-1)}\underset{p\times q}{U}^{((p-1)(q-1))}\right] \otimes b_{10}\underset{r\times t}{U}^{(10)} \\
& \vdots \\
+ & \left[a_{00}\underset{p\times q}{U}^{(00)} + a_{01}\underset{p\times q}{U}^{(01)} + \cdots + a_{(p-1)(q-1)}\underset{p\times q}{U}^{((p-1)(q-1))}\right] \\
& \otimes b_{(r-1)(t-1)}\underset{r\times t}{U}^{((r-1)(t-1))}
\end{aligned}
$$

and since $\underset{p\times r}{A} = \sum_{i=0}^{p-1}\sum_{k=0}^{r-1} a_{ik}\underset{p\times r}{U}^{(i,k)}$, we can write:

$$
\begin{aligned}
S^{-1}_{[p|r]}(\underset{r\times t}{B} \otimes \underset{p\times q}{A})S_{[q|t]} = & \quad \underset{p\times q}{A} \otimes b_{00}\underset{r\times t}{U}^{(00)} \\
& + \underset{p\times q}{A} \otimes b_{10}\underset{r\times t}{U}^{(10)} \\
& \vdots \\
& + \underset{p\times q}{A} \otimes b_{(r-1)(t-1)}\underset{r\times t}{U}^{((r-1)(t-1))} \\
= & \quad \underset{p\times q}{A} \otimes \left[b_{00}\underset{r\times t}{U}^{(00)} + b_{10}\underset{r\times t}{U}^{(10)} + \cdots \right. \\
& \qquad \left. + b_{(r-1)(t-1)}\underset{r\times t}{U}^{((r-1)(t-1))}\right]
\end{aligned}
$$

$$
\Longrightarrow \boxed{S^{-1}_{[p|r]}(\underset{r\times t}{B} \otimes \underset{p\times q}{A})S_{[q|t]} = \underset{p\times q}{A} \otimes \underset{r\times t}{B}}
$$

□

In contrast, the array algebra based proof is, simply:

*Proof:* Let $\underset{pr\times qt}{C} = \underset{p\times q}{A} \otimes \underset{r\times t}{B}$. Then,

$$
\begin{aligned}
\underset{pr\times qt}{C} &= \underset{p\times q}{A} \otimes \underset{r\times t}{B} \\
&\Updownarrow \\
C|^{ij}_{mn} &:= A|^{i}_{m}\, B|^{j}_{n} \\
&= I|^{i}_{x}\, A|^{x}_{m}\; I|^{i}_{y}\, B|^{y}_{n} \\
&= I|^{i}_{x}\, A|^{x}_{k}\, I|^{k}_{m}\; I|^{j}_{y}\, B|^{y}_{l}\, I|^{l}_{n} \\
&= I|^{j}_{y}\, I|^{i}_{x}\; B|^{y}_{l}\, A|^{x}_{k}\; I|^{k}_{m}\, I|^{l}_{n}
\end{aligned}
$$

$$
\begin{aligned}
&:= \quad [\underset{r}{I} \star \underset{p}{I}]^{ij}_{yx}\,[B \otimes A]^{yx}_{lk}\,[\underset{q}{I} \star \underset{t}{I}]^{lk}_{mn} \\
&:= \quad [S_{[r|p]}(B \otimes A)S_{[q|t]}\,]^{ij}_{mn} \\
&\Updownarrow \\
\underset{pr\times qt}{C} \quad &= \quad S_{[r|p]}(B \otimes A)S_{[q|t]}
\end{aligned}
$$

$$
\Longrightarrow \boxed{\underset{p\times q}{A} \otimes \underset{r\times t}{B} = S^{-1}_{[p|r]}(\underset{r\times t}{B} \otimes \underset{p\times q}{A})S_{[q|t]}}
$$

□

# References


[1] N. Alexandridis, "Adaptable Software and Hardware: Problems and Solutions," *IEEE Computer*, Vol. 19, pp. 29–39, Feb. 1986.

[2] H. C. Andrews and J. Kane, "Kronecker Matrices, Computer Implementation, and Generalized Spectra," *Journal of the Association for Computing Machinery*, Vol. 17, pp. 260–268, Apr. 1969.

[3] D. Baldwin, "Why We Can't Program Multiprocessors the Way We're Trying to Do it Now," Tech. Rep. 224, University of Rochester, Aug. 1987.

[4] H. S. Barad, *The SCOOP Pyramid: An Object-Oriented Prototype of a Pyramid Architecture for Computer Vision*, Ph.D. dissertation, Electrical Engineering Dept., University of Southern California, Los Angeles, California, Dec. 1987. USC SIPI Report 115.

[5] R. Bellman, *Introduction to Matrix Analysis*, New York: McGraw-Hill, 1960.

[6] S. H. Bokhari, "On the Mapping Problem," *Proceedings of the 1979 International Conference on Parallel Processing*, pp. 239–248, Aug. 1979.

[7] J. W. Brewer, "Kronecker Products and Matrix Calculus in System Theory," *IEEE Trans. on Circuits and Systems*, Vol. CAS-25, pp. 772–781, Sept. 1978.

[8] J. W. Cooley, P. A. W. Lewis, and P. D. Welch, "Historical Notes on the Fast Fourier Transform," *IEEE Transactions on Audio and Electroacoustics*, Vol. AU-15, No. 2, pp. 76–79, 1967.

[9] M. Davio, "Kronecker Products and Shuffle Algebra," *IEEE Trans. on Computers*, Vol. C-30, pp. 116–125, Feb. 1981.

[10] P. J. Davis, *Circulant Matrices*, New York: John Wiley & Sons, 1979.

[11] T. J. Drabik, M. A. Title, and S. H. Lee, "Architectures and Algorithms for Digital Optical Computing Systems with Applications to Numerical Transforms and Partial Differential Equations," *Proceedings of SPIE*, Vol. 625, Jan. 1986.

[12] J. A. Eisele and R. M. Mason, *Applied Matrix and Tensor Analysis*, New York: Wiley-Interscience, 1970.

[13] D. F. Elliott and K. R. Rao, *Fast Transforms: Algorithms, Analyses, Applications*, New York, N.Y.: Academic Press, 1982.

[14] B. J. Fino and V. R. Algazi, "A Unified Treatment of Discrete Fast Unitary Transforms," *SIAM Journal on Computing*, Vol. 6, pp. 700–717, Dec. 1977.

[15] M. H. Fitzpatrick, *Kronecker and Other Special Products of Matrices*, Ph.D. dissertation, Dept. of Mathematics, Auburn University, Auburn, Alabama, June 1964.

[16] R. C. Gonzalez and P. Wintz, *Digital Image Processing*, Addison Wesley, 1987.

[17] A. Graham, *Kronecker Products and Matrix Calculus with Applications*, New York: John Wiley & Sons, 1981.

[18] W. Hillis, "The Connection Machine: A Computer Architecture Based on Cellular Automata," *Physica*, Vol. 10D, pp. 213–228, 1984.

[19] W. Hillis, *The Connection Machine*, ACM Distinguished Dissertations, Cambridge, Massachusetts: The MIT Press, 1985.

[20] K. Hwang and F. A. Briggs, *Computer Architecture and Parallel Processing*, McGraw-Hill, 1984.

[21] B. Jenkins, *Optical Logic Systems: Implementation and Architectural Implications*, Ph.D. dissertation, Dept. of Electrical Engineering, USC, Los Angeles, Calif., Apr. 1984. USC-IPI Report 1110.

[22] T. Kumagai and K. Ikegaya, "The Two-Dimensional Inverse Omega Network," *Proceedings of the 1985 International Conference on Parallel Processing*, pp. 325–327, 1985.

[23] S. Y. Kung, *VLSI Array Processors*, Englewood Cliffs, New Jersey: Prentice-Hall, 1988.

[24] S. Y. Kung, H. J. Whitehouse, and T. Kailath, eds., *VLSI and Modern Signal Processing*, Englewood Cliffs, New Jersey: Prentice-Hall, 1985.

[25] T. Kushner, A. Y. Wu, and A. Rosenfeld, "Image Processing on MPP:1," *Pattern Recognition*, Vol. 15, No. 3, pp. 121–130, 1982.

[26] T. Kushner, A. Y. Wu, and A. Rosenfeld, "Image Processing on ZMOB," *IEEE Transactions on Computers*, Vol. C-13, pp. 943–951, Oct. 1982.

[27] T. R. Kushner and A. Rosenfeld, "A Model of Interprocessor Communication for Parallel Image Processing," *IEEE Transactions on Systems, Man and Cybernetics*, Vol. SMC-13, pp. 600–618, July/August 1983.

[28] S. Y. Lee and J. K. Aggarwal, "A Problem-Driven Approach to Parallel Image Processing: System Design and Scheduling," in *IEEE International Conference on Systems, Man and Cybernetics*, Nov. 1985.

[29] S. P. Levitan, *Parallel Algorithms and Architectures: A Programmer's Perspective*, Ph.D. dissertation, Dept. of Computer and Information Sciences, University of Massachusetts, Amherst, Mass., May 1984. COINS Technical Report 84-11.

[30] W. Liu and G.-G. Mei, "A Class of Two-Dimensional Problems: VLSI Algorithms and Architectures," *International Conference on Computer Design: VLSI in Computers*, pp. 194–197, Oct. 1986.

[31] G.-G. Mei and W. Liu, "Parallel Algorithms for Computer Vision Primitives," *International Conference on Computer Design: VLSI in Computers*, pp. 506–509, Oct. 1986.

[32] D. I. Moldovan, "On the Analysis and Synthesis of VLSI Algorithms," *IEEE Transactions on Computers*, Vol. C-31, pp. 1121–1126, Nov. 1982.

[33] D. I. Moldovan, "On the Design of Algorithms for VLSI Systolic Arrays," *IEEE Proceeding*, pp. 113–120, Jan. 1983.

[34] H. Neudecker, "A Note on Kronecker Matrix Products and Matrix Equation Systems," *SIAM Journal of Applied Mathematics*, Vol. 17, pp. 603–606, May 1969.

[35] H. J. Nussbaumer and P. Quandalle, "Fast Computation of Discrete Fourier Transforms Using Polynomial Transforms," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, Vol. ASSP-27, pp. 169–181, Apr. 1979.

[36] R. Offen, ed., *VLSI Image Processing*, McGraw-Hill, 1985.

[37] A. V. Oppenheim and R. W. Schafer, *Digital Signal Processing*, Prentice-Hall, Inc., 1975.

[38] D. S. Parker, Jr., "Notes on Shuffle/Exchange-Type Switching Networks," *IEEE Transactions on Computers*, Vol. c-29, pp. 213–222, Mar. 1980.

[39] M. Pease, "The Indirect Binary n-Cube Microprocessor Array," *IEEE Transactions on Computers*, Vol. c-26, pp. 458–473, May 1977.

[40] W. K. Pratt, *Digital Image Processing*, John Wiley & Sons, 1978.

[41] K. Preston, Jr., "Cellular Logic Algorithms for Graylevel Image Processing," in *Multicomputers and Image Processing Algorithms and Programs*, pp. 135–147, Academic Press, 1982.

[42] K. Preston, Jr., M. J. B. Duff, S. Levialdi, P. E. Norgen, and J.-I. Toriwaki, "Basics of Cellular Logic with Some Applications in Medical Image Processing," *Proceedings of the IEEE*, Vol. 67, pp. 826–856, May 1979.

[43] P. Quinton, "Automatic Synthesis of Systolic Arrays from Uniform Recurrent Equations," $11^{th}$ *Annual International Symposium on Computer Architecture*, pp. 208–214, June 1984.

[44] A. Ralston and E. D. Reilly, Jr., eds., *Encyclopedia of Computer Science & Engineering*, New York: Van Nostrand Co., second ed., 1983.

[45] S. K. Rao, *Regular Iterative Algorithms and their Implementations on Processor Arrays*, Ph.D. dissertation, Stanford Univ., Oct. 1985.

[46] A. P. Reeves, "Parallel Computer Architectures for Image Processing," *Computer Vision, Graphics and Image Processing*, Vol. 25, pp. 68–88, 1984.

[47] G. X. Ritter and P. D. Gader, "Image Algebra Techniques for Parallel Image Processing," *Journal of Parallel and Distributed Processing*, Vol. 4, pp. 7–44, 1987.

[48] D. J. Rose, "Matrix Identities of the Fast Fourier Transform," *Linear Algebra and its Applications*, Vol. 29, pp. 423–443, 1980.

[49] A. Rosenfeld, "Cellular Architectures: From Automata to Hardware," in *Multicomputers and Image Processing Algorithms and Programs*, pp. 254–260, Academic Press, 1982.

[50] A. Rosenfeld, "The Prism Machine: An Alternative to the Pyramid," *Journal of Parallel and Distributed Computing*, Vol. 2, pp. 404–411, 1985.

[51] A. Rosenfeld and A. C. Kak, *Digital Picture Processing*, vol. 1, Academic Press, second ed., 1982.

[52] A. A. M. Saleh, "Matrix Analysis of Mildly Nonlinear, Multiple-Input, Multiple-Output Systems with Memory," *Bell System Technical Journal*, Vol. 61, pp. 2221–2243, Nov. 1982.

[53] A. Sawchuk and I. Glaser, "Geometries for Optical Implementations of the Perfect Shuffle," in *Proc. International Optical Computing Conference*, (Toulon, France), 29 Aug.–2 Sept. 1988.

[54] R. C. Singleton, "A Method for Computing the Fast Fourier Transform with Auxiliary Memory and Limited High-Speed Storage," *IEEE Transactions on Audio and Electroacoustics*, Vol. AU-15, No. 2, pp. 91–97, 1967.

[55] F. Soviš, "Uniform Theory of Shuffle-Exchange Type Permutation Networks," *Proceedings $10^{th}$ Annual Symp. on Computer Architecture*, pp. 185–191, June 1983.

[56] H. Stone, "Parallel Processing with the Perfect Shuffle," *IEEE Transactions on Computers*, Vol. C20, pp. 153–161, Feb. 1971.

[57] P. H. Swain, H. J. Siegel, and J. el Achkar, "Multiprocessor Implementation of Image Pattern Recognition: A general Approach," *Proc. Conf. Pattern Recognition and Image Processing*, pp. 309–317, Dec. 1980.

[58] G. Tait, W. Vetter, and P. Bélanger, "The Array Matrix for Multivariate Analysis," *Third International Symposium on Large Engineering Systems*, pp. 69–76, July 1980.

[59] G. R. Tait, *The Array-Matrix Concept – A New Approach to Multivariate Analysis*, Ph.D. dissertation, Dept. of Electrical Engineering, McGill University, Montreal, Canada, Mar. 1971.

[60] S. L. Tanimoto, "A Hierarchical Cellular Logic for Pyramid Computers," *Journal of Parallel and Distributed Computing*, Vol. 1, pp. 105–132, Nov. 1984.

[61] S. L. Tanimoto, J.-P. Crettez, and J.-C. Simon, "Alternative Hierarchies for Cellular Logic," *Journal of Parallel and Distributive Computing*, 1984.

[62] L. Uhr, *Algorithm-Structured Computer Arrays and Networks*, Computer Science and Applied Mathematics, Academic Press, 1984. Architectures and Processes for Images, Percepts, Models, Information.

[63] L. Uhr, "Parallel, Hierarchical Software/Hardware Pyramid Architectures," Tech. Rep. Computer Sciences # 646, University of Wisconsin-Madison, June 1986.

[64] S. Unger, "A Computer Oriented Toward Spatial Problems," *Proceedings of the IRE*, pp. 1744–1750, Oct. 1958.

[65] W. J. Vetter, "Matrix Calculus Operations," *SIAM Review*, Vol. 15, pp. 352–369, Apr. 1973.

[66] W. J. Vetter, "On Matrix Operations for Large System Studies," *Third International Symposium on Large Engineering Systems*, pp. 55–61, July 1980.

[67] W. J. Vetter, "The Array Matrix Generalization for Signal Processing," *International Concference on Acoustics, Speech, and Signal Processing*, pp. 297–300, Apr. 1986.

[68] R. J. Weidner, *A Stochastic Extension of the Structural Properties of Compartment Models via a Kronecker Algebra*, Ph.D. dissertation, Oklahoma State University, Stillwater, Oklahoma, May 1981.