Dimension and Shape Invariant Array Programming: The Implementation and the Application

AHMED SAMEH & MANAL EZZAT
Department of Computer Science,
The American University in Cairo,
P.O.Box 2511, Cairo
EGYPT.

Abstract:-Designing an architecture to handle large conceptual data blocks is very much needed. This paper implements a model for the shape and dimension invariant programming of arrays based on the notation of the Mathematics of Arrays (MOA) algebra. It focuses on dimension and shape invariance and their effect in parallel computing. The MOA algebra is implemeted as a library of APIs, that contains object oriented classes implemented in C++. The APIs reduce the erroneous loops starts, strides, and stops used by programmers in the traditional models of handling multi-dimension arrays. The library defines the dimension and shape of the arrays at runtime, and gives the code of the problem at hand better chances to be automatically parallelized. An image-processing tool is implemented using the new MOA library, proving the correctness and effecicy of the model. Some video processing operations such as transformation of AVI frames and motion detection schemes are implemented using the MOA library. The parallelization factors inherent in the MOA implementation are demonstrated in terms of shape polymorphism, MOA parallel architecture, data redistribution, and tiling algorithms. Furthermore, pipelining MOA computations has also been demonstrated.

Key-Words:- Theory of Arrays, Image Processing, Video Processing, pipelining, tiling.

1. Introduction

The More’s Array Theory (AT) was formalized from the data concepts of APL, extending them to include nested arrays and systematic handling of second order functions [2]. AT is a formal description of nested arrays in a first order logic. It is a mathematical model of the ways in which the orthogonal arrangement of material bodies interacts with their hierarchy nesting. The theory stems out of geometric experience with finite collections. Originally, the motivation to formalize array theories and languages was to create a notation for subscripting sequences, vectors, matrices, and higher dimensional objects. Array interactions were always found in functional languages, whereas imperative languages defined arrays of static dimension and shape, and accessed them by indexing subscription.

2. MOA Notation

Mathematics of Arrays (MOA) is a notation that is based on the Psi-Calculus, which correspondes to the array theory (AT) and contains a set of notations for array interactions. It encapsulates the same array properties as AT, only differs in that its data elements are numeric scalars that can be extended to any homogenous scalar types. Scalars are defined to be arrays of no dimension and with empty shape vector. The MOA algebra is a set of operations proven to be useful for scientific algorithms. All operations are based on shapes and indexing functions. Previous research presented recommendations of the MOA extensions to functional languages [3][4][5].

The properties of the mathematics of arrays notation are as follows: Lazy Evaluation., Array Expression Simplification., Updating in Place, Avoiding Temporaries, Shift, Rotate, Take, or Drop segments of registers or memory locations (arrays), Fit Nicely in Cache Structures, so, Executes faster taking advantage of nearest memory location, even on uni-processor, In multi-processor environment, sub-array operations can be scheduled over various processors. These potential speed-ups could be fully realized in a parallel architecture, and could be partially realized on a uni-processor architecture. Therefore, a combination of an accurate model of memory caches, concurrency of sub-array operations, and low process communication would achieve good performance [8].

The MOA notation is divided into the following seven segments (see figure 1& Table 2 in the Appendix):

- The Measurement Operations: (Dimensionality: dx represent the rank of the array, Shape: rx represent the length of each dimension, Tau: tx represent the total number of elements , Ravel: rav x collapses the array to one dimension flat vector, Pi: px represent the product of the elements of any arbitrary vector).
- The Indexing Operations: (Psi: : $\vec{i}\, y x^{n}$ accesses selected index from the array, Gamma: $g\left(\vec{a};\vec{b}\right)$ returns an index in the flattened array, Gamma Reverse: g' returns the index of a scalar raveled array).
- The Array Constructing Operations: (Reshape: $\vec{s}\, r x$ binary infix operation the constructs arrays, Catenate: $x_1 \oplus x_2$ concatenates arrays over the required dimension, Iota: $i \quad n$ a vector whose entries are specified integers).
- The Scalar Operations: (Point-wise Extension: $x_1 \quad op \quad x_2$ apply an operator on the equally indexed pair of elements of two arrays, Scalar the elements of the array on a required dimension, Rotate : $\vec{s} q x$ rotates the array on every axe towards increasing indices, Transpose: $\vec{v}\Phi x$ apply reduction function, Compress: /• compresses the array on the positions given, Expand: •\ the revese of the previous function). The Reduction & Scan Operations: (Reduce: (op red) like the scalar function but applied on the components of the array cumulatively, Scan: ${}_{op}scan \quad x$ returns all the partial results of the array.

- The Higher Order Operations: (Omega: ${}_{op}\Omega a \quad x_1 \quad b \quad x_2$ high order function that applies operation on pairs of identically indexed subarrays, Unary Dot: • outer product, Binary Dot: • ’ inner product).

The Axis Transposition & Partition Operations: (Take: $\vec{i} \uparrow x$ this partition an array by returning a subarray, Drop: $\vec{i} \downarrow x$ returns the remaning of the array in the take function, Reverse: : $d f x$ reverses

Extension: $s \quad op \quad x$ applies an operation between the scalar and every element in the array).

In order to implement the MOA notation, the original equations were analyzed, and sometimes rewritten to suite the programming point of view. The following example for the reduction operation in the original equation, and the rewritten one, shows how recursion was eliminated in the new equation. It is followed by another example for the resulting shape after the take operation, showing the reduction to simplify programmability of the equation.

Original Equation

$${}_{op}red \quad x = 0 \quad y \quad x \quad op \quad \left({}_{op}red \quad \left(1 \quad \downarrow \quad x\right)\right)$$

$$r\left(\vec{i} \quad \uparrow \quad x\right) = \left(abs \quad \vec{i}\right) \quad . \quad \left(t\vec{i}\right) \quad \left(rx\right)$$

New Equation:

$$\vec{i} \quad y \quad \left({}_{op}red \ \ x\right) = op_{c=0}^{c=b[0]} \quad \left\langle c, \quad i[1], \quad \ldots, \quad i[j-1]\right\rangle \quad y \quad x$$

$$r\left(\vec{i} \quad \uparrow \quad x\right) = \begin{cases} \vec{i} & if \ \ j = k \\ \left\langle i[0], \quad \ldots, \quad i[j-1], \quad b[j], \quad \ldots, \quad b[k-1]\right\rangle & if \ \ 0 \le j < k \end{cases}$$

The Psi reduction theorem is applied in the implementation of the MOA notation. This theorem states that any MOA expression can be reduced to a simple expression based on the Psi function only. Also, reductions functions were implemented using the MOA red construct, and the compress construct. The elements of the Psi Correspondence Theorem (PCT) were formalized into three equations: expr (extracts slices from an array, according to their Psi index position, by definning the index of the slice required in every dimension in the array), mix (glues arrays formed using y on partial indices), and link (concatenates the items collected from the array in a previous expr operation). Equations for the three PCT notations are presented, and their usage is magnified later in the Video Processing Application (section 4).

An MOA library is implemented in three classes: CMOA, CMOAImage, and CMOAVideo. The main class CMOA is designed to operate all MOA notations on an MOA_rec structure that contains the dimension n, shape (n tuple of sizes – extents or upper bounds - of the n dimensions), and the data elements of the MOA array, stored as a flat (raveled) array in principal raw major order. The length of the flat array of the MOA array data elements is equal to the product of the bounds of its dimensions (the values of the shape vector elements). The shape and data of the MOA array are explicitly separated for the advantages of the shape theory. The type of the Data is fixed in this implementation to be a DWORD. The choice of DWORD data type is influenced by the image and video applications implemented as described later (sections 3 & 4). The image pixels, which are the array elements of the MOA arrays in both applications, require a minimum of one bit (black and white images) and a maximum of double word (32-bit pixel, describing the intensities of the blue, green, and red components of each pixels) space in memory to store a pixel value. However, data morphology (see section 4) can be implemented since C++ is a language that allows for overloading. Overloading can be employed to redefine the MOA functions to operate on different data types for array elements, based on different MOA structures denoting the different types.

3. Image Processing with MOA

In the experimental part of this research, Image and Video files are mapped to MOA structures and processed using the MOA notations. The nesting of the MOA provides a high order functional efficiency. The Image was defined in MOA paradigm as a two-dimensional array, where the height of the image is denoted by the first dimension, and the second dimension denotes the width. The values of the array elements represent the pixel color value, of the specified x, y coordinates, which are the indices of the current position. The speed benefits of representing the image in MOA structures were not fully achieved because of the mapping between the traditional image.

structure and the MOA structure to apply MOA notations, and then transform back to traditional image structure for display. To fully utilize the MOA notation in image processing, and for achieving better performance results, an image MOA file format should be designed and a drawing function that reads directly from the new format.

This spatial representation allowed for reshaping, transposing, rotating, and reversing the elements of the array causing symmetrical effects on the image. Having a convolution function implemented in the tool, applied for all types of filtering, and even for morphology. The convolution is defined in MOA notation using the following equation:

$$\vec{i} \quad y \quad x_r \quad = \quad {}_{+}red \left[x_m \quad * \quad \left[rx_m \quad \Delta \quad \left(i \quad - \quad \frac{rx_{m-1}}{2} \right) \quad \nabla \quad x_i \right] \right]$$

with $\frac{rx_{m-1}}{2} \quad \leq \quad *\vec{i} \quad \leq \quad * rx_m \quad - \quad \frac{(rx_{m-1})}{2}$

This equation was refined to allow for double convolution, taking two masks to apply on the input array, and was refined again to allow for convoluting the image to select the maximum or minimum values in a defined area to execute the dilation or erosion morphology respectively. Then opening (erosion, then dilation), and closing (dilation, then erosion) of the images were implemented also. Figure 3 illustrates some filter effects, showing in image (a) the original image, in (b) the effect of a high pass filter as generated by the CMOAImage class, in (c) the effect of dilation, and finally in (d) the effect of erosion.

Also segmentation of an image can be done using the convolution function, by sending a detection mask, like horizontal line detection, vertical line detection, point detection,etc. Other segmentation techniques can be implemented by designing their algorithms based on MOA structures, like the region segmentation based on discontinuities detection, or applying thresholds. The region segmentation will then require a representation scheme. Representation and description of images have several alternatives that most of them can be easily mapped to the MOA image structures.

4. Video Processing with MOA

As a three-dimensional illustration of the MOA notation, a video processing application was implemented. Video processing is another field where MOA can be very beneficial. The AVI file format is

Frames designed to contain Audio-Video Interleaved streams. This means that the images forming the frames of a video file, are stored in form of streams, then another layer of an audio stream is interleaved in the file. This structure complicates the video processing operation. Since streams will need to be processed one by one, and will require more effort from the programmer to isolate frames while being able to process them collectively in a symmetric manner. The MOA representation for a video file is a 3-dimensional MOA array structure, where the first dimension is the time sequencer or the frame number, and the remaining two are the height and width of the frames images. Video processing operation on the Video frames can be processed individually using the Psi MOA indexing function to return the image of the frame number required, or collectively by working on all the frames at one shot (by applying all operations on the second dimension).

The CMOAVideo class provides the main interfacing between the MOA structure defined in the CMOA class and the AVI file format. The main objective achieved from this class is the decomposition of the AVI video stream into frames of the images, which the video displays. The pixels of all these frames are mapped to a 3-dimensional MOA.The Video MOA structure allows all the transformations discussed in the previous section, to be applied to image frames composing the video stream (flip vertical, flip horizontal, transpose). Extra transformations can be applied like the reversing the order of frames in the video stream symmetrically. Also, forward and rewind operations are a simple positioning of the current display index in the images frame.

Object tracking (Motion detection) is implemented in this research work, using a simple xoring technique, which cumulatively xores the equally indexed pixels in every frame with the previous cumulative differential frame, applying the following formula:

$$d_{ij}(x, \quad y) = \begin{cases} 1 \quad if \quad | f(t_i, x, y) \quad xor \quad f(t_j, x, y) | \quad > \quad q \\ \quad 0 \quad otherwise \end{cases}$$

Where x and y are the spatial coordinates of pixels in images of the video stream, and t is the time of the image display. It starts from the first frame as the initial cumulative frame, and produces a final cumulative xored frame that shows the boundary of the moving object along its track of movement through all images in the video stream. This is implemented by partitioning the video MOA structures (3-D MOA arrays) to image frames (2-D MOA arrays) and applying a point-wise extension to each frame and the differential image. This technique zeros out the still background. In case of noise, thresholds need to be applied first to ensure that only moving objects will be detected in the final result. Different methods can be applied to show different analysis for the moving object. Figure 4 gives an example as analyzed by the implemented object tracking algorithm. It shows all the images in the video stream of an AVI file as decomposed by the MOA partitioning operations (partitioning on the time – 0th- dimension), and the resulted cumulative xored differential image showing the movement of the pointers of the clock AVI file. Also, audio and video correspondence in an AVI video file can be implemented in the MOA video structure by applying the Psi Correspondence Theorem (PCT), by relating the audio stream and the video stream coded in the AVI file format as two MOA structures in correspondence of each other. Streaming is a technique of using a small buffer to play a large file by filling the buffer with data from the file at the same rate that data is taken from the buffer and played.

5. Parallelism, Pipelining and Hardware Implementation

The existence of the parallelisation factors in the MOA deign, and the possibility of pipelining its execution are investigated. Also, a hardware package of the notation was implementing using the VHDL language on Renoir [1]. Parallelism in MOA is discussed in terms of Implicit Parallelism in MOA design, Shape Separation benefits, MOA Parallel Architecture Mapping Scheme, MOA in Data Redistribution, and MOA & Tiling Algorithms.
Since MOA is a higher construct for array interactions, compilers can extract implicit parallelism from the code. MOA as based on Psi-Calculus, is a mathematical formalization that decides the data placement from the array expression, allowing the array mapping to processors to be based on the Psi-Reduction theorem, and Psi-Correspondence theorem on two levels. The first is the functional normal form, which is a minimal semantic form expressed in terms of selections using Cartesian coordinates, employing Psi- Tiling is the process of applying geometric transformations in the iteration space in order to restructure the loop nest (execution of statements within the loop, or the loop iteration), in order to improve the performance. These transformations need also to preserve a reasonable tradeoff between communication, computation, and allow for automatic parallelisation.

Pipelining is one of the parallel processing techniques, Accessing arrays by loops allows pipelining to be a process of scheduling technique that starts a loop iteration, before the previous iteration completes. This part of the research work [1], considered the Monolithic Array Pipelining Algorithm in terms of MOA. The main constructors of the algorithm were fount to be:

- IGEN (Index Generator Sub-graph).
- AGEN (Array Generator Sub-Graph).
- Selection Operation with Index Calculation expression.

As MOA can implement the above constructors effectively invariant of shape & dimension, it is safe to say that pipelining is feasible using the MOA structure implemented.

The hardware implementation is intended to allow for designing hardware accelerators for different applications based on the MOA package implemented in this research work. The hardware implementation was done using VHDL – a hardware description language that has syntax like C – on Renoir [1]. It was simulated using the ModelSim Simulator [1]. The VHDL MOA package was implemented using the same logic applied in C++. The simulation produced the same results except for the lack of dynamic size allocations that was compensated by a constant upper bound.

Reduction theorem. The second level is the transformation of the functional normal form to its equivalent operational normal form, which describes the implementation results, in terms of starts, strides, and lengths to select from the linear arrangement of the items, employing the Psi-Correspondence Theorem.

Shape definition is useful for determining communication strategies, load balancing, evaluation strategies, etc. Parallel Computing (can be referred to as shape-based computing) requires separating the shape of the data structure away from the data itself. Dynamic and Static shape analysis steps are usually done by compilers. Data types are divided into shapely data-types (regular & irregular arrays, graphs, records, variant records, lists and trees) and non-shapely data-types (functions and sets). In this work, the MOA structures are implemented with separate vectors for shape, eliminating the need for extra analysis, and providing shape polymorphism.

6. Conclusion

This research work attempted to link the MOA notation with the current application requirements in image and video processing and provid a model that can be used to achieve parallel processing in imperative languages. This research work has shown that the MOA implementation neither need a compiler nor extensions to functional languages. It can be implemented as APIs in an imperative Object Oriented Language like C++.

7. Bibliography

[1] Manal E. Helal, "Dimension and Shape Invariant Programming: The Application and the Implementation", A Master of Science Thesis Dissertation, American University in Cairo, January 2001.

[2] Mullin, Lenore, "A Mathematics of Arrays", Ph. D. Dissertation, Syracuse University, December 1988.

[3] Lenore Mullin, Scott Thibault, "A Reduction Semantics for Arrays Expressions: The PSI Compiler", Department of Computer Science, University of Missouri-Rolla, Rolla, Missouri 65401, CSC-94-05, March 9, 1994.

[4] Lenore M. R. Mullin, Michael A. Jenkins, "Effective Data Parallel Computation using the PSI-Calculus", July 25, 1995.

[5] Lenore R. Mullin (PA), Werner Kluge, "On Programming Scientific Applications in a Functional Language Extended by a - Calculus Subsystem for Array Operations", March, 13, 1995

[6] Rafael C. Gonzalez, Richard E. Woods, "Digital Image Processing", Addison-Wesley Publishing Company, 1993.

[7] Hoda A. Khalil, "Tiling as a Loop Parallelisation Technique", AMaster of Science Thesis Dissertation, American University In Cairo, May 2000.

[8] C. Barry Jay, "Shape Analysis for Parallel Computing", School of Computing Sciences, University of Technology, Sydney.

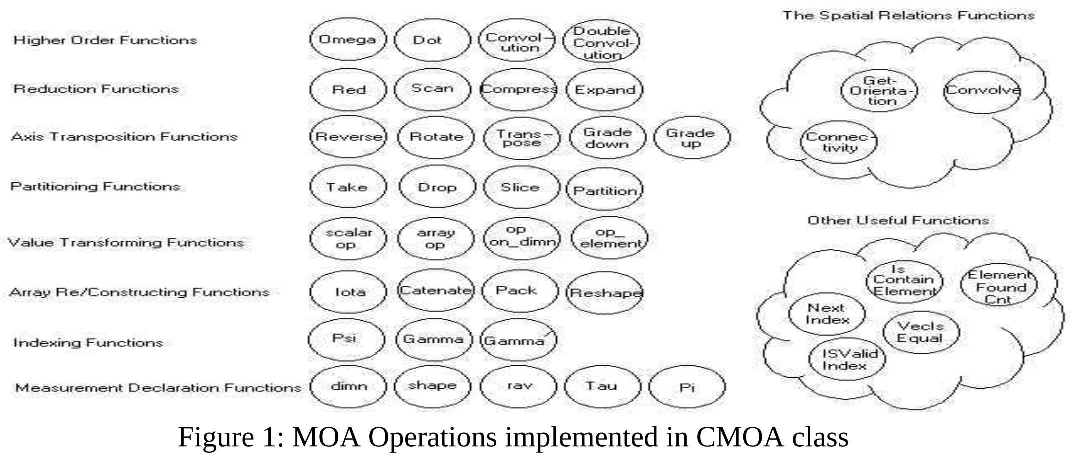
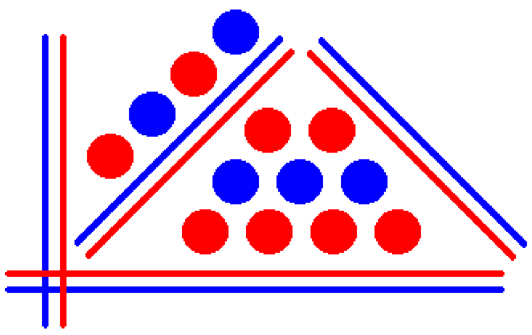
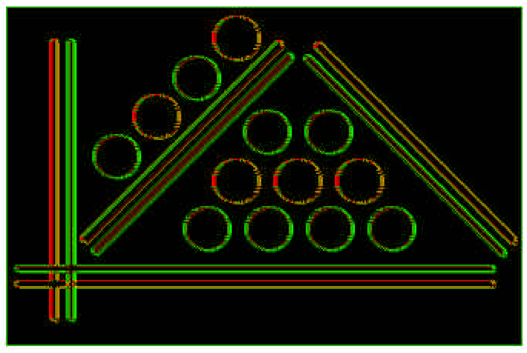


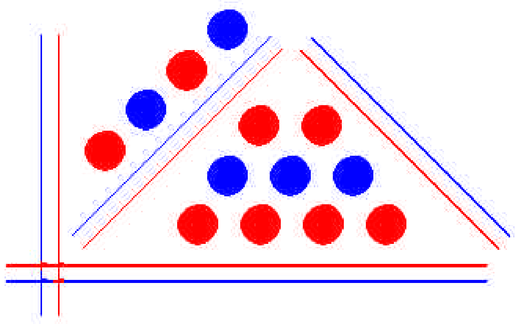
Figure 1: MOA Operations implemented in CMOA class



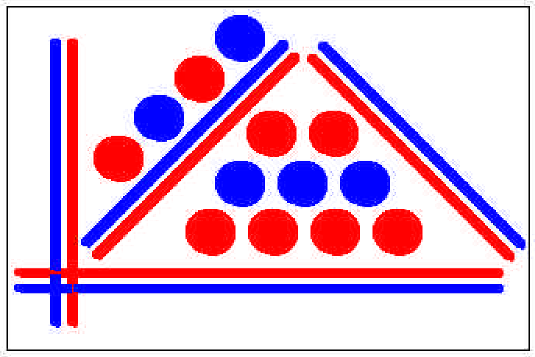
a) Original Image



b) High Pass filter Effect

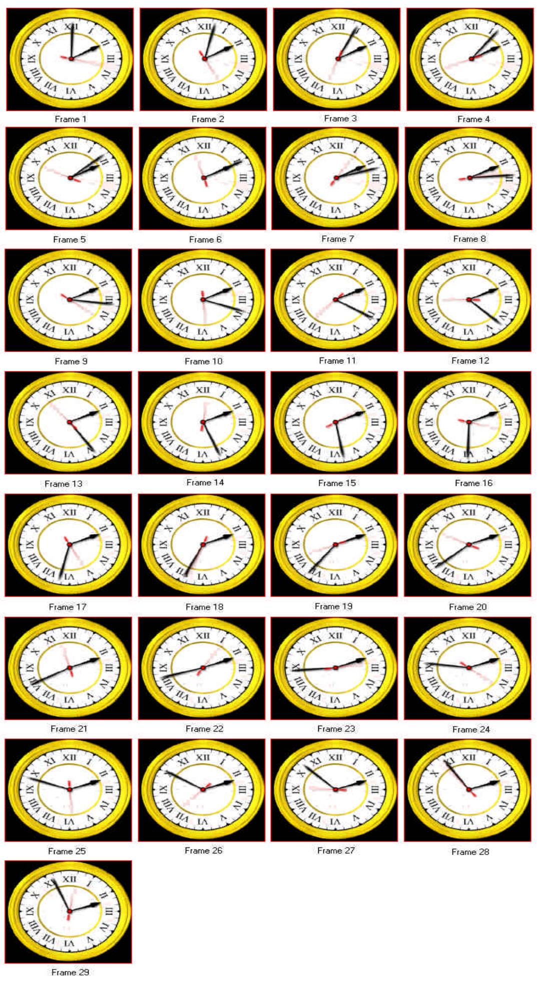


c) Dilation Effect

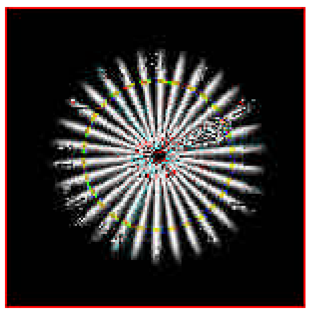


d) Erosion Effect

Figure 2: Image Filters Effects



a) Clock Images in the Video Stream



b) Cumulative Xored Result

Figure 3: Object tracking algorithm applied on a Moving Clock