NISTIR 6115

Parallel Algorithms for Multi-Indexed Recurrence Relations with Applications to DPCM Image Compression*

Abdou Youssef

Department of EECS
The George Washington University
Washington, D.C. 22052
youssef@seas.gwu.edu
Tel: (202) 994-6569
Fax: (202) 994-0227

*This research was performed in part at:

U.S. DEPARTMENT OF COMMERCE
Technology Administration
National Institute of Standards and Technology
Information Technology Laboratory
Gaithersburg, MD 20899

NISTIR 6115

Parallel Algorithms for Multi-Indexed Recurrence Relations with Applications to DPCM Image Compression*

Abdou Youssef

Department of EECS
The George Washington University
Washington, D.C. 22052
youssef@seas.gwu.edu
Tel: (202) 994-6569
Fax: (202) 994-0227

*This research was performed in part at:

U.S. DEPARTMENT OF COMMERCE
Technology Administration
National Institute of Standards and Technology
Information Technology Laboratory
Gaithersburg, MD 20899

December 1998

U.S. DEPARTMENT OF COMMERCE
William M. Daley, Secretary

TECHNOLOGY ADMINISTRATION
Gary R. Bachula, Acting Under Secretary for Technology

NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY
Raymond G. Kammer, Director

Parallel Algorithms for Multi-Indexed Recurrence Relations with Applications to DPCM Image Compression [1]

Abdou Youssef

Department of EECS
The George Washington University
Washington, DC 22052
youssef@seas.gwu.edu
Tel: (202) 994-6569
Fax: (202) 994-0227

Abstract

Parallel solution of 1-indexed recurrence relations has received much attention, but no effort has been made to design parallel algorithms for multi-indexed recurrence relations, where the recurrence is in more than one index. Multi-indexed relations arise in JPEG-standard DPCM compression of images, which is an increasingly important area due to the explosion of imagery applications such as multimedia and medical imaging. In this paper we design and analyze novel parallel algorithms for solving multi-indexed recurrence relations of arbitrary order. To that end, we develop three techniques: *index sequencing*, *index decoupling*, and *dimension shifting*. Index sequencing leads to a parallel algorithm of $O(n \log n)$ time on n processors, where the input size is n^2. Index decoupling applies in limited situations, one of which is a commonly used special case of DPCM, and leads to a parallel algorithm of $O(\log n)$ time for $n \times n$ images on $n \times n$ processors. Dimension shifting is a universal technique for multi-indexed relations, and involves reducing the number of indices to 1 while clustering the terms of the relation into vectors. This is accomplished by means of special vector operators that we define and study. Our parallel algorithm for doing the dimension shifting and for solving the resulting 1-indexed recurrence relation takes $O(\log^2 N)$ time on meshes of partitionable buses or hypercubes, where N is the data input-output size.

Keywords: Compression, DPCM, Parallel Algorithms, Recurrence Relations, Dimension Shifting, Convolution, Fourier Transform.

[1] This research was performed in part at the National Institute of Standards and Technology.

1 Introduction

The massive amounts of imagery data in many applications demand that images be compressed to reduce their storage and transmission requirements. In certain applications, most notably medical imaging, the compression has to be lossless. One of the well-known lossless image compression techniques is differential pulse-code modulation (DPCM), which was adopted by the Joint Photographic Expert Group (JPEG) as an international standard [12].

Whenever an image is retrieved, locally or remotely, it has to be decompressed online for display. Therefore, decompression, also called decoding, must be as fast as possible. Compression, also called coding, is desired to be fast too, but its speed is not as critical as the decoding speed because coding is typically performed just once, often offline, whereas decoding is done many times, often online or in real time. Consequently, it is especially important to develop fast decoding algorithms.

DPCM decoding is essentially the computation of a 2-indexed scalar recurrence relation of low order; every term in the recurrence relation is indexed with two indices, the row and column positions of pixels. Similarly, DPCM of 3D and 4D images, which is conceivable in applications involving 3D visual modeling with or without motion, leads to 3-indexed and 4-indexed recurrence relations. Much work has been done on parallel algorithms for solving 1-indexed recurrence relations of arbitrary order [2, 6, 7, 13], due to their applicability in numerical computations [16, 17, 18]; the fastest parallel algorithms for solving those equations take logarithmic time, and are based on parallel prefix computation [1, 4, 8, 9, 10, 11, 13, 15]. However, no work has been reported on multi-indexed recurrence relations. Considering the importance of DPCM coding/decoding of imagery, and the need for fast decoding, parallel algorithms for solving multi-indexed recurrence relations merit serious study.

In this paper we design and analyze parallel algorithms for solving multi-indexed recurrence relations of arbitrary order, and identify the architectures best suited for the algorithms. For clarification, the multi-indexed relations are addressed in increasing order of complexity. Two-indexed recurrence relations of order 1 are addressed first, 2-indexed recurrence relations of arbitrary order are addressed second, and multi-indexed relations of arbitrary order, where the number of indices is greater than 2, are addressed last,

We develop three approaches for solving multi-indexed recurrence relations: *index sequencing*, *index decoupling*, and *dimension shifting*. The index sequencing approach for solving a 2-indexed relation to compute $n \times n$ terms, as in DPCM decoding of an $n \times n$ image[2], breaks down the relation into a sequence of n 1-indexed scalar recurrence relations that must be solved one after another. Each relation can be solved by a parallel $O(\log n)$ time algorithm on an n-processor hypercube or partitionable bus, and thus the n equations take $O(n \log n)$ time on n processors. This approach does not, however, exploit the full potential of parallelism.

Index decoupling, when applicable, breaks the 2-indexed relation into many independent 1-indexed recurrence relations, which can then be solved simultaneously, leading to an overall logarithmic time parallel algorithm. We will apply this approach to one special but commonly used DPCM. However, since index decoupling does not apply universally to arbitrary multi-

[2] All the algorithms in the paper apply equally well on $n \times m$ rectangular images, but for ease of presentation we will consider only $n \times n$ square images in the 2-indexed relations case.

indexed recurrence relations, alternative approaches must be sought.

Dimension shifting is a universal approach. To understand it, one must note the three dimensionalities in a multi-indexed recurrence relation: *the term dimensionality*, *the equation order*, and *the index dimensionality*. The first refers to the dimension of every term in the recurrence relation, and the second indicates the number of previous terms each term depends on. The index dimensionality is the number of indices q per term. Dimension shifting involves reducing the index dimension of q-indexed relations down to 1 while increasing the term dimension by a certain factor, then reducing the order dimension down to 1 while increasing the term dimension by another appropriate factor. This yields a 1-indexed vector recurrence relation of order one, where the vectors are $(q-1)$-dimensional arrays. The operations involved in the resulting recurrence relation are multidimensional-vector convolution and addition. Since multidimensional convolution can be done in parallel in logarithmic time using multidimensional FFT [3, 5, 20], and since standard parallel algorithms for 1-indexed recurrence relations of order 1 take a logarithmic number of operations [2, 6, 7, 13], the total time of the algorithm is square-logarithmic, using N processors configured as a q-dimensional mesh of partitionable buses or hypercubes, where N is the output size. Note that converting a q-indexed recurrence relation to a 1-indexed relation involves solving, recursively, a $(q-1)$-indexed recurrence relation.

Two points are noteworthy. First, dimension shifting has been applied to 1-indexed multi-order recurrence relations to reduce the order down to 1 and increase the term dimension [7]; however, dimension shifting of the index dimension to term dimension is novel, and involves some intricate operators and operator manipulation. Second, dimension shifting could be done more easily by shifting the index dimension to the order dimension first. For example, the 2-indexed terms of an $n \times n$ image can be re-indexed with one index by ordering the pixels in some manner such as row-wise or column-wise. This re-indexing turns the recurrence relation into a 1-indexed relation without changing the dimension of the terms (e.g., pixels), but it increases the order of the relation dramatically. Reducing the order afterwords requires an n-fold increase in the term dimension, and the corresponding parallel algorithm to solve the resulting equation takes $O(n \log n)$ time on an $n \times n$ processor system, which is considerably longer than the square-logarithmic time mentioned above. Therefore, our dimension shifting approach outlined in the previous paragraph is superior.

The paper is organized as follows. The next section overviews DPCM and formulates the DPCM decoding problem as a multi-indexed scalar recurrence relation. Sections 3 and 4 develop the index-sequencing and index-decoupling approaches, respectively. Section 5 presents a vector-operator based formulation of 2-indexed recurrence relations of order 1, defining the operators and proving some of their relevant properties. In section 6, parallel algorithms for setting up and solving the resulting 1-indexed vector recurrence relations are developed and analyzed. Section 7 addresses arbitrary-order 2-indexed recurrence relations, and develops a parallel algorithm for them. Section 8 will address the most general case, namely, multi-indexed recurrence relations of arbitrary order, where the number of indices is greater than 2. Finally, section 9 closes the paper with conclusions and future directions.

2 DPCM and Multi-indexed Recurrence Relations

The most widely used form of DPCM is overviewed first, and formulated into a 2-indexed recurrence relation of order 1. Afterwards, higher order and higher dimension DPCM is presented and formulated into a multi-indexed recurrence relation of the same order.

DPCM assumes that the interpixel redundancies and correlations can be modeled as a 2D Markov model of some order [5]. For order 1, the model involves three parameter, a, b and c, as described next. Let $X(0:n-1, 0:n-1)$ be a matrix of pixels, representing an image. The model assumes that every pixel $X(i,j)$ can be largely "predicted" from its three neighbors in the north, west, and north-west as follows:

$$X(i,j) = aX(i,j-1) + bX(i-1,j) + cX(i-1,j-1) + E(i,j) \tag{1}$$

where $E(i,j)$ is *the prediction error*, or *residual*. Note that when i and j are outside their appropriate range, the corresponding value for $X(i,j)$ is assumed to be zero. Some of the commonly used values for (a,d,c) are (1,1,-1), (1/2,1/2,0), and (3/4,3/4,-1/2). The (1,1,-1) is often used, partly because of its arithmetic simplicity (no division or multiplication). This special case will be treated separately using the index decoupling approach.

DPCM coding involves first computing the residuals $E(i,j)$ using the following equation derived from the model of equation (1):

$$E(i,j) = X(i,j) - (aX(i,j-1) + bX(i-1,j) + cX(i-1,j-1)), \tag{2}$$

and then encode the residuals into a bit stream, using an entropy coder such as Huffman coding, run-length encoding, or arithmetic coding, which are lossless [14]. The residuals tend to be of small values and of high frequencies of occurrence, enabling the entropy coder to reduce the data size, typically by a factor of 2, 3 or 4.

The computation of E is easily parallelizable on an $n \times n$ mesh, assuming the whole matrix X is available in the processors ($X(i,j)$ in processor $P(i,j)$). Each $P(i,j)$ sends its data $X(i,j)$ to its three neighbors to the south, east, and south-east. Afterwards, each processor (i,j) has all the data it needs to compute $E(i,j)$ according to equation (2). The communication and the parallel computation take constant time. Of course the entropy coding of E will have to be parallelized, but this is left out of our scope because it calls for different approaches.

DPCM decoding decodes the bit stream back to the residuals $E(i,j)$, and then computes the pixel values $X(i,j)$ according to equation (1). Clearly, equation (1) is a 2-indexed scalar recurrence relation of order one, which will be solved in parallel in Sections 3 and 4.

The generalization of equation (1) comes from higher order DPCM and from multidimensional imagery. A DPCM of order (k,t) and parameter array $a(0:k, 0:t)$ assumes a 2D Markov model of images where every pixel can be largely predicted from its north-west rectangular neighborhood that extends k steps to the north and t steps to the west. The governing DPCM equation, which is a generalization of equation 1 is the following 2-indexed recurrence relation of order (k,t):

$$X(i,j) = \sum_{l=0}^{k} \sum_{s=0}^{t} a(l,s)X(i-l,j-s) + E(i,j) \tag{3}$$

where $a(0,0)$ is equal to 0 so that $X(i,j)$ does not depend on itself; the term $a(0,0)$ is kept to simplify the expression of the double summation in equation (3). Here again, when the indices step out of their appropriate bounds, the corresponding terms are assumed to be zero.

Finally, DPCM of order $(t_1, t_2, ..., t_q)$ and parameter array $a(0:t_1, 0:t_2, ..., 0:t_q)$, for q-dimensional imagery data, involves a q-indexed recurrence relation of order $(t_1, t_2, ..., t_q)$ in the form:

$$X(i_1, i_2, ..., i_q) = \sum_{r_1=0}^{t_1} \sum_{r_2=0}^{t_2} ... \sum_{r_q=0}^{t_q} a(r_1, r_2, ..., r_q) X(i_1 - r_1, i_2 - r_2, ..., i_q - r_q) + E(i_1, i_2, ... i_q), \quad (4)$$

where $a(0,0,...,0)$ is equal to 0 so that $X(i_1, i_2, ..., i_q)$ does not depend on itself. As before, any term is assumed to be zero if its indices are out of their defined bounds. Note that when $t_1 = t_2 = ... = t_q = t$, the recurrence relation is said to be of order t.

In this paper we will develop parallel algorithms for solving equations (1), (3) and (4), in that order of increasing difficulty. We will start with the index sequencing and index decoupling techniques first, due to their simplicity, and then move on to the general dimension shifting technique.

3 Index sequencing

Index sequencing applies to equation (1) to compute X from E as follows. For each fixed row i, in the sequential order $i = 0, 1, ..., n-1$, equation (1) is a 1-indexed recurrence relation of order 1, where the index is j, and the last three terms, i.e., $X(i-1,j)$, $(X(i-1,j-1)$ and $E(i,j)$, are known and available. Here is the algorithm, assuming an n-processor system configured as a hypercube or a partitionable bus. Processor j hosts column j of E, and is responsible for computing column j of X. Also, a, b and c are assumed to be stored in all the processors.

Algorithm Index-sequencing(**input:** X, a, b, c; **output:** X)
begin
1. **for** $i = 0$ **to** $n-1$ **do**
2. Each processor $j-1$ sends its $X(i-1,j-1)$ to processor j;
3. Each processors j computes $d_j = bX(i-1,j) + cX(i-1,j-1) + E(i,j)$;
4. All the processors solve in parallel the 1-indexed recurrence relation of order 1
$$X(i,j) = aX(i,j-1) + d_j,$$
where the recurrence index is j, thus computing row i of X;
endfor
end

Analysis

Step 2 takes $O(1)$ time on a partitionable bus, or $O(\log n)$ time on a hypercube, because the step is a translation (i.e., shift) communication step. Step 3 takes $O(1)$ time. Finally, step 4 takes $O(\log n)$ time using any standard parallel algorithm for solving scalar 1-indexed

recurrence relations on n processors [2, 6, 7, 13]. Consequently, step 1, and hence the whole algorithm, takes $O(n \log n)$ time. The corresponding speedup is $O(\frac{n^2}{n \log n}) = O(\frac{n}{\log n})$, and the efficiency is $O(1/\log n)$.

Clearly, the above algorithm has the advantages of good speedup, good efficiency, and relatively small number of processors. However, it does not exploit the full potential for parallelism, and thus other techniques are called for.

4 Index Decoupling: A Special Case of DPCM

As mentioned earlier, among the most common values for the parameter triplet (a, b, c) is $(1, 1, -1)$. In this section we will treat this special case due to its popularity. Index decoupling will apply to any situation where $a = 1$ and $c = -b$.

Under the assumption that $a = 1$ and $c = -b$, equation 1 becomes:

$$X(i,j) = X(i,j-1) + b\,(X(i-1,j) - X(i-1,j-1)) + E(i,j),$$

which is equivalent to

$$X(i,j) - X(i,j-1) = b(X(i-1,j) - X(i-1,j-1)) + E(i,j). \tag{5}$$

By defining

$$Y(i,j) = X(i,j) - X(i,j-1), \tag{6}$$

equation (5) becomes:

$$Y(i,j) = bY(i-1,j) + E(i,j). \tag{7}$$

Observe that equation (7) consists of n independent equations, one per column j, where each equation is a scalar 1-indexed recurrence relation of order 1, and the recurrence index is i. The initial value $Y(-1,j) = 0$ for all j. Due to independence, those n recurrence relations can be solved simultaneously, where each equation can be solved by a parallel recurrence relation algorithm on a separate subsystem.

Once Y is computed, X can be computed using a simple relation derived from equation (6):

$$X(i,j) = X(i,j-1) + Y(i,j). \tag{8}$$

Here again, equation (8) consists of n independent equations, one per row i, where each equations is a scalar 1-indexed recurrence relation of order 1, and the recurrence index is j. The initial value $X(i,-1) = 0$ for all i. Also, due to independence, those n recurrence relations can be solved simultaneously, where each equation can be solved by a parallel prefix algorithm on a separate subsystem. Thus, equations (7) and (8) form the essence of the index decoupling approach, since they decouple equation (1) into two separate sets of 1-indexed relations. The algorithm for solving equations (7) and (8) is given next, assuming an architecture of 2D mesh where every row and every column is a hypercube or a partitionable bus, because a hypercube or a partitionable bus is an ideal architecture for solving 1-indexed recurrence relations [19, 20].

Algorithm Index-decoupling(**input:** E; **output:** X)
begin

1. **for** j=0 **to** $n-1$ **pardo**
2. solve the relation $Y(i,j) = bY(i-1,j) + E(i,j)$ on a separate mesh column ;
   endfor

3. **for** i=0 **to** $n-1$ **pardo**
4. call parallel prefix on $X(i,j) = X(i,j-1) + Y(i,j)$ to run on a separate mesh row;
   endfor

end

Analysis

First assume an $n \times n$ mesh of hypercubes or partitionable buses. For each j, step 2 runs on column j of the mesh, and takes $O(\log n)$ time because it is a 1-indexed scalar recurrence relation or order 1. Therefore, the whole for-loop in step 1 takes $O(\log n)$. Similarly, for each i, step 4 runs on row i of the mesh and takes $O(\log n)$ time for the same reason. Accordingly, the whole for-loop in step 3 takes $O(\log n)$ time. Therefore, the whole algorithm takes $O(\log n)$ time, which is faster than the previous algorithm by a logarithmic factor. The speedup over the sequential algorithm is clearly $O(\frac{n^2}{\log n})$, and the efficiency is $O(\frac{1}{\log n})$.

If the system is a $p \times p$ mesh of hypercubes or partitionable buses, where $p \leq n$, then divide each of the matrices X, Y, and E into $p \times p$ square blocks of size $\frac{n}{p} \times \frac{n}{p}$ each, and let the (i,j) block reside in processor (i,j). Since each 1-indexed recurrence relation takes $O(n/p + \log p)$ time on p processors, and since each row or column of processors has to solve n/p recurrence relations, it follows that each for-loop, and thus the whole algorithm, takes $O((n/p + \log p)n/p)$ time. If $p \log p \leq n$, then the algorithm takes $O(n^2/p^2)$ time, resulting in $O(p^2)$ speedup and $O(1)$ efficiency, which are optimal up to a constant factor.

Clearly, index dimension yields a time-optimal logarithmic parallel algorithm for the popular special case of DPCM where $(a,b,c) = (1,-1,1)$. However, it does not apply to the general case of arbitrary values of (a,b,c) or to multi-indexed relations of arbitrary order.

The remainder of the paper will develop the universal dimension-shifting approach to derive square-logarithmic-time parallel algorithms for arbitrary multi-indexed recurrence relations of arbitrary order, starting with 2-indexed relations of order 1, and then generalizing to 2-indexed relations of arbitrary order and finally to multi-indexed relations.

5 Dimension Shifting: Conversion to 1-indexed Vector Recurrence Relations

One fundamental idea in dimension shifting is the use of the *shift operator* $\mathcal{S}$ on column vectors. If $V(0:n-1)$ is a column vector, then $\mathcal{S}V$ is another vector $W(0:n-1)$ derived from V by shifting it down one position: for all $i > 0$, $W(i) = V(i-1)$, and $W(0) = 0$. The operator $\mathcal{S}$ can be applied repeatedly (say, r times) on a vector. This is equivalent to composition of operators. Denote by $\mathcal{S}^r$ the resulting operator. Clearly, $\mathcal{S}^r V$ is a vector derived from V by shifting the latter down r times, that is, for all $i = 0, 1, ..., n-1$, $(\mathcal{S}^r V)(i) = V(i-r)$, where a term is 0 when its index is out of bounds. Two special cases of r are worth noting: $r = 0$ and $r = n$. $\mathcal{S}^0$ is by convention the *identity operator* $\mathcal{I}$, that is, $\mathcal{I}V = V$ for any vector V. For convenience, we will sometimes use 1 instead $\mathcal{I}$. When $r = n$, observe that $\mathcal{S}^n V = 0$, the zero vector of length n. Accordingly, we simply denote the operator $\mathcal{S}^n$ by 0. Note that for all $r \geq n$, $\mathcal{S}^r = 0$. Polynomial operators in $\mathcal{S}$, that is, any linear combination of the powers of $\mathcal{S}$, can be easily defined as follows.

Definition 1 *Given a real sequence $a_0, a_1, ..., a_{n-1}$, a polynomial operator $\mathcal{T} = \sum_{l=0}^{n-1} a_l \mathcal{S}^l$ is defined to be such that for any vector V, $\mathcal{T}V = \sum_{l=0}^{n-1} a_l (\mathcal{S}^l V)$. The vector $T(0:n-1)$, where $T(i) = a_i$ for all i, is called the characteristic vector of operator $\mathcal{T}$.*

Throughout the paper, the following notational convention will be followed. For any matrix $M(0:n-1, 0:n-1)$, denote by M_j the j-column of M, so that $M_j(i) = M(i,j)$. Now we are in a position to cast equation 1 (and equation 3 later on) of DPCM into a new form.

Recall that equation 1 is

$$X(i,j) = aX(i,j-1) + bX(i-1,j) + cX(i-1,j-1) + E(i,j),$$

which, when using $M_j(i)$ for $M(i,j)$, becomes

$$X_j(i) = aX_{j-1}(i) + bX_j(i-1) + cX_{j-1}(i-1) + E_j(i).$$

Using the definition of $\mathcal{S}$, we have $X_j(i-1) = (\mathcal{S}X_j)(i)$ and $X_{j-1}(i-1) = (\mathcal{S}X_{j-1})(i)$. Consequently, the above equation becomes

$$X_j(i) = aX_{j-1}(i) + b(\mathcal{S}X_j)(i) + c(\mathcal{S}X_{j-1})(i) + E_j(i), \textit{for all } i.$$

Noting that the last equation involves only the i-th components of the various column vectors, the index i can be dropped, and the equation becomes a condensed vector equation:

$$X_j = aX_{j-1} + b\mathcal{S}X_j + c\mathcal{S}X_{j-1} + E_j.$$

By moving $b\mathcal{S}X_j$ to the left, and "factoring out" X_j on the left and X_{j-1} on the right, we obtain

$$(1 - b\mathcal{S})X_j = (a\mathcal{I} + c\mathcal{S})X_{j-1} + E_j. \tag{9}$$

The question now is how to get rid of the "coefficient" operator of X_j on the left hand side. Essentially, the question is whether there is an inverse operator $(1 - b\mathcal{S})^{-1}$; if such an inverse

exists, we can multiply both sides of the equation by that inverse to get rid of the coefficient operator of X_j. Fortunately, we know how to compute the infinite series expansion of $\frac{1}{1-x}$ for real values x: $\frac{1}{1-x} = 1 + x + x^2 + x^3 + ...$ This expansion and the fact that $\mathcal{S}^r = 0$ for $r \geq n$ suggest the following theorem for inverting $(1 - b\mathcal{S})$.

Theorem 1 $(1 - b\mathcal{S})^{-1} = 1 + b\mathcal{S} + b^2\mathcal{S}^2 + ... + b^{n-1}\mathcal{S}^{n-1}$.

Proof $(1 - b\mathcal{S}) \sum_{r=0}^{n-1} b^r\mathcal{S}^r = \sum_{r=0}^{n-1} b^r\mathcal{S}^r - \sum_{r=0}^{n-1} b^{r+1}\mathcal{S}^{r+1} = 1 - b^n\mathcal{S}^n = 1$, because $\mathcal{S}^n = 0$. ■

Multiplying equation (9) by $(1 - b\mathcal{S})^{-1}$ on both sides, and using Theorem 1, we have

$$X_j = (\sum_{r=0}^{n-1} b^r\mathcal{S}^r)(a\mathcal{I} + c\mathcal{S})X_{j-1} + (\sum_{r=0}^{n-1} b^r\mathcal{S}^r)E_j. \tag{10}$$

Denote by $\mathcal{A}$ the operator that is the coefficient of X_{j-1} in the last equation. Since operator polynomials multiply in the same manner as the more familiar real polynomials, it can be easily seen that

$$\mathcal{A} = (\sum_{r=0}^{n-1} b^r\mathcal{S}^r)(a\mathcal{I} + c\mathcal{S}) = a + \sum_{r=1}^{n-1}(ab^r + cb^{r-1})\mathcal{S}^r. \tag{11}$$

Denote by $\mathcal{B}$ the coefficient of E_j in equation (10), and let $F_j = \mathcal{B}E_j$,that is,

$$\mathcal{B} = \sum_{r=0}^{n-1} b^r\mathcal{S}^r, \text{ and } F_j = \mathcal{B}E_j. \tag{12}$$

Now equation (10) can be written as a 1-indexed vector recurrence relation of order 1:

$$X_j = \mathcal{A}X_{j-1} + F_j \tag{13}$$

where the X_j's are the unknowns (to be computed), $\mathcal{A}$ is a known operator with known parameters (which depend on the given values a, b and c, as shown in equation (11), and the F_j's are known vectors because they are computable using equation (12).

Although equation (13) has the simple form of a linear recurrence relation of order 1, the "multiplication" of a vector by an operator is not usually encountered in the literature on parallel algorithms for recurrence relations. Neither is it clear how to parallelize that multiplication operation. However, the next theorem below will show that multiplying a vector by a polynomial operator is nothing but a convolution, a partial convolution to be precise.

Recall that the convolution of two vectors $U(0 : n-1)$ and $V(0 : n-1)$ is a vector $W(0 : 2n-1)$ where $W(i) = \sum_{r=0}^{i} U(r)V(i-r)$, using again the convention that when indices are out of their bounds, the corresponding terms are zeros. Only the first n components of W will be of use to us. Therefore, we will denote by $\otimes$ the partial convolution operation that returns the first n components of the answer of a full convolution. For ease of reference, we will define partial convolution formally.

Definition 2 *Let $U(0 : n-1)$ and $V(0 : n-1)$ be two vectors. The partial convolution of U and V, denoted $U \otimes V$, is a vector $W(0 : n-1)$ of the same length as U and V such that $W(i) = \sum_{r=0}^{i} U(r)V(i-r)$, for $i = 0, 1, ..., n-1$.*

Since full convolution is an associative operation, it can be easily shown that partial convolution $\otimes$ is associative too.

Theorem 2 *Let $\mathcal{T} = \sum_{r=0}^{n-1} a_r \mathcal{S}^r$ be a polynomial operator of characteristic vector T, where $T(i) = a_i$, and let $V(0:n-1)$ be a vector. Then $\mathcal{T}V = T \otimes V$.*

Proof: Let $W = \mathcal{T}V = \sum_{r=0}^{n-1} a_r \mathcal{S}^r V$, and recall that $(\mathcal{S}^r V)(i) = V(i-r)$. Therefore, $W(i) = \sum_{r=0}^{n-1} a_r (\mathcal{S}^r V)(i) = \sum_{r=0}^{n-1} a_r V(i-r) = \sum_{r=0}^{n-1} T(r) V(i-r)$, for all $i = 0, 1, ..., n-1$. It follows that $W = T \otimes V$, and thus $\mathcal{T}V = T \otimes V$. ■

Denote by A and B the characteristic vectors of operators $\mathcal{A}$ and $\mathcal{B}$ defined in equations (11) and (12), respectively. Then, by the previous theorem, equation (13) and the accompanying definitions can now be put in a condensed form:

$$X_j = A \otimes X_{j-1} + F_j, \ j = 0, 1, ..., n-1 \quad (14.a)$$
$$F_j = B \otimes E_j \quad (14.b)$$
$$A(0) = a, \ A(r) = ab^r + cb^{r-1} \ for \ r = 1, 2, ..., n-1 \quad (14.c)$$
$$B(r) = b^r \ for \ r = 0, 1, ..., n-1. \quad (14.d)$$

Note that the initial value of equation (14.a) is X_{-1} being the zero column vector.

In the next section a parallel algorithm for computing first the parameters A and F_j's, and then the vectors X_j's, using equations 14, will be presented and analyzed.

6 Parallel Algorithm for 2-Indexed Relations of Order 1

To arrive at a suitable architecture for solving equations (14), two observations must be noted. First, first-order 1-indexed scalar recurrence relations are best solved on a hypercube or a fully partitionable bus [19, 20]. Second, since each term in equation 14.a is a vector, and the operations involved are vector operations, namely, $\otimes$ and vector addition, every processor in the scalar case is best replaced by a column of processors, preferably connected as a hypercube or a fully partitionable bus, to run $\otimes$ fast. Therefore, a good topology for the overall system is an $n \times n$ mesh where every row and every column is either a hypercube or a fully partitionable bus. (Note that an n^2-node hypercube can be configured as an $n \times n$ mesh of subcubes.) Each processor (i, j) hosts $X_j(i) = X(i, j)$, $E(i, j)$, $F_j(i)$, $A(j)$ and $B(j)$. The algorithm will first compute vectors A and B in the leftmost column of the mesh, and then broadcast them row-wise to the remaining mesh columns; afterwards, each mesh column j computes $F_j = B \otimes E_j$; finally, equation 14.a is solved in parallel by all the n columns of the mesh, where every column is treated as a subsystem in which $\otimes$ and vector addition are computed in parallel.

Note that the system may be a $p \times p$ mesh of the same topology as before, for $p \leq n$. In that case, each of the matrices X, E and F is divided into $p \times p$ square blocks of size $\frac{n}{p} \times \frac{n}{p}$ each, such that blocks (i, j) reside in processor (i, j). Also, A and B are divided into p subcolumns of size n/p each; they are computed in the leftmost column of the mesh, and then the i-th subcolumn of A and of B are broadcast row-wise to all the processors in row i of the mesh, for all i. The computations are parallel across blocks, but sequential within the blocks, because each block is allocated to a single processor.

The algorithm is given next.

Algorithm Index-2-order-1(**input:** E, a, b, c; **output:** X)
begin
/* Setting up of equation 14.a: computing B, A and the F_j's */
1. The 1st column of the mesh performs parallel prefix to compute $B(0 : n-1)$ where $B(r) = b^r$;
2. The 1st column of the mesh first computes vector $D = \mathcal{S}B$ by shifting down vector B one step, and then computes vector A in parallel as $A = a * B + c * D$;
3. The 1st column of the mesh broadcasts vectors A and B row-wise, that is, for each row i, the 1st processor hosting $A(i)$ and $B(i)$ broadcasts them to all the processors in its row;
4. **for** j=0 **to** $n-1$ **pardo**
5. Compute $F_j = B \otimes E_j$ by a parallel convolution algorithm on a separate column;
endfor

/* Solving recurrence relation (14.a) */
6. Solve the 1-indexed vector recurrence relation $X_j = A \otimes X_{j-1} + F_j$;
Note that since each term in the recurrence relation is a column vector, each vector is handled by a separate mesh column, and each vector operation involved ($\otimes$ and $+$) is performed in parallel within the host column of the mesh architecture.
end

Analysis

First assume an $n \times n$ mesh of hypercubes or partitionable buses. Step 1 takes $O(\log n)$ time. Step 2 takes $O(1)$ time if using partitionable buses; however, if the leftmost column is a hypercube, the shifting down of B by one step is a mapping $i \rightarrow i+1$, which requires $O(\log n)$ time to execute, and the rest of the computation of A takes $O(1)$ time. Step 3 takes $O(1)$ time if the rows are buses, and $O(\log n)$ if the rows are hypercubes. Step 5 (and thus Step 4) takes $O(\log n)$ times on either topology using FFT [19, 20]. Finally, step 6 takes $O((\log n)(T_\otimes + T_+))$, where $T_\otimes$ is the time to execute one $\otimes$ operation on one mesh column, and T_+ is the time to add two vectors in one mesh column. Clearly, $T_\otimes = O(\log n)$ as indicated earlier, and $T_+ = O(1)$. Therefore, step 6 takes a dominant $O(\log^2 n)$ time, leading to an overall algorithm time complexity of $O(\log^2 n)$. The corresponding speedup is $O(\frac{n^2}{\log^2 n})$, and the efficiency of the algorithm is $O(\frac{1}{\log^2 n})$.

If the mesh is $p \times p$ where $p \leq n$, the same analysis applies with a few adjustments. Essentially, one convolution operation $\otimes$ takes $O(n/p + \log p)$ on p processors. Therefore, step 1 takes $O(n/p + \log p)$ time. One can bypass step 3 by broadcasting (a, b, c) to all the processors, and letting each column perform steps 1 and 2 redundantly and independently in $O(n/p + \log p)$ time. Steps 4 and 5 involve n/p convolutions per column of processors, thus taking $O((n/p)(n/p + \log p))$ time. Finally, step 6 takes $O((n/p + \log p)(T_\otimes + T_+))$, where now $T_\otimes = O(n/p + \log p)$ and $T_+ = O(n/p)$ because of using only p processors per each operation. Therefore, step 6, and consequently the whole algorithm take $O((n/p + \log p)^2)$

time. If $p \log p \leq n$, then $\log p \leq n/p$ and the whole algorithm takes $O((n/p)^2)$ time, resulting in speedup of $O(n^2/(n/p)^2) = O(p^2)$ and efficiency $O(1)$.

7 Generalization to 2-Indexed Relations of Arbitrary Order

As mentioned earlier, predicting values in DPCM can involve more than one previous term in each matrix dimension. That leads to higher order DPCM, or specifically DPCM of order (k, t), for which the DPCM decoding relation is equation (3) which we re-state here as equation (15):

$$X(i,j) = \sum_{l=0}^{k} \sum_{s=0}^{t} a(l,s) X(i-l, j-s) + E(i,j), \tag{15}$$

where $a(0,0)$ is equal to 0 so that $X(i,j)$ does not depend on itself. The array $a(0:k, 0:t)$ of DPCM parameters is given, and k and t are typically small positive integers.

Equation (15) will be converted to a 1-indexed t-th order vector recurrence relation, using the same shift operator $\mathcal{S}$. Using the notation $M_j(i)$ for $M(i,j)$, equation (15) becomes

$$X_j(i) = \sum_{l=0}^{k} \sum_{s=0}^{t} a(l,s) X_{j-s}(i-l) + E_j(i)$$

Since $X_{j-s}(i-l) = (\mathcal{S}^l X_{j-s})(i)$, the above equation transforms to

$$X_j(i) = \sum_{l=0}^{k} \sum_{s=0}^{t} (a(l,s) \mathcal{S}^l X_{j-s})(i) + E_j(i), \ \textit{for all } i.$$

Because the last equation involves the i-th components of the various column vectors, the index i can be dropped, and the equation becomes a condensed vector equation:

$$X_j = \sum_{s=0}^{t} \sum_{l=0}^{k} (a(l,s) \mathcal{S}^l X_{j-s}) + E_j,$$

where the order of the double summation is reversed. By splitting the summation over s into two parts, one for $s = 0$ and the other for $s \neq 0$, and by recalling that $a(0,0) = 0$, we obtain

$$X_j = (\sum_{l=1}^{k} a(l,0) \mathcal{S}^l X_j) + \sum_{s=1}^{t} \sum_{l=0}^{k} a(l,s) \mathcal{S}^l X_{j-s} + E_j.$$

Next, move the first sum on the right hand side to the left, and factor X_j out, resulting in

$$(1 - \sum_{l=1}^{k} a(l,0) \mathcal{S}^l) X_j = \sum_{s=1}^{t} (\sum_{l=0}^{k} a(l,s) \mathcal{S}^l) X_{j-s} + E_j. \tag{16}$$

To simplify the notation, let

$$\mathcal{T}_0 = 1 - \sum_{l=1}^{k} a(l,0)\mathcal{S}^l, \quad \mathcal{T}_s = \sum_{l=0}^{k} a(l,s)\mathcal{S}^l$$

so that equation (16) becomes $\mathcal{T}_0 X_j = \sum_{s=1}^{t} \mathcal{T}_s X_{j-s} + E_j$, leading to

$$X_j = \sum_{s=1}^{t} \mathcal{T}_0^{-1}\mathcal{T}_s X_{j-s} + \mathcal{T}_0^{-1} E_j. \tag{17}$$

As we did with equation (5) in Section 4, we have to invert the polynomial operator $\mathcal{T}_0$. The inversion here is, however, more complex than it was in the case of equation (5). As the next theorem will show, $\mathcal{T}_0^{-1}$ does exist, and it is a polynomial operator whose coefficients will be the solution of a scalar recurrence relation of order k.

Theorem 3 *1)* $\mathcal{T}_0^{-1} = \sum_{l=0}^{n-1} C(l)\mathcal{S}^l$, *where the* $C(l)$*'s satisfy the following 1-indexed scalar recurrence relation of order* k*:*

$$C(0) = 1, \; C(r) = \sum_{l=1}^{k} a(l,0)C(r-l), \; \textit{for } 1 \le r \le n-1. \tag{18}$$

2) For every $s = 1, 2, \ldots, t$, $\mathcal{T}_0^{-1}\mathcal{T}_s = \sum_{l=0}^{k} B_s(l)\mathcal{S}^l$, *where*

$$B_s = C \otimes a(0 : n-1, s). \tag{19}$$

Note that for $m > k$, $a(m,s) = 0$.

Proof: 1) The $C(l)'s$ satisfy $(1 - \sum_{l=1}^{k} a(l,0)\mathcal{S}^l)(\sum_{l=0}^{n-1} C(l)\mathcal{S}^l) = 1$. In that identity relation, the coefficient of $\mathcal{S}^0$ is $C(0)$ on the left side and 1 on the right side, leading to $C(0) = 1$. For $r = 1, 2, \ldots n-1$, the coefficient of $\mathcal{S}^r$ is $C(r) - \sum_{l=1}^{min(k,r)} a(l,0)C(r-l)$ on the left hand side, and 0 on the right hand side. Thus, $C(r) = \sum_{l=1}^{min(k,r)} a(l,0)C(r-l)$. By taking $C(m) = 0$ for $m < 0$, the value of $C(r)$ simplifies to $C(r) = \sum_{l=1}^{k} a(l,0)C(r-l)$.

2) Thr proof of this part follows immediately from observing that operator polynomials multiply in the same way as real polynomials, that polynomial multiplication is equivalent to convolution of the coefficients, and that $\mathcal{S}^r = 0$ for all $r \ge n$. ■

Using Theorem 3, and recalling from Theorem 2 that applying a polynomial operator to a vector is a convolution $\otimes$, equation 17 becomes

$$X_j = \sum_{s=1}^{t} B_s \otimes X_{j-s} + F_j \tag{20.a}$$
$$F_j = C \otimes E_j. \tag{20.b}$$

Clearly, equation 20.a is a 1-indexed vector recurrence relation of order t. This completes the dimension shifting from index dimension to term dimension. It remains to shift the order dimension to term dimension.

The standard method for solving 1-indexed recurrence relations of order $t > 1$ is to convert the equation into a vector equation of order 1 such that every vector has t terms, where each

term has the same dimensionality as the terms of the original recurrence relation. Specifically, equation 20.a will be converted to the following form

$$\overline{X}_j = A \odot \overline{X}_{j-1} + \overline{F}_j \tag{21}$$

where $\overline{X}_j$ is a column vector of t vectors

$$\overline{X}_j = [X_{tj-1}\ X_{tj-2}\ X_{tj-2}\ \ldots X_{tj-t}]^T,\ j = 1, 2, ..., n/t,$$

and A, $\overline{F}_j$ and operation $\odot$ are to be defined. Note that in the definition of $\overline{X}_j$, the exponent T stands for matrix transpose. Note also that $\overline{X}_0$ is the zero vector of length $t \times n$.

The conversion from order t to order 1 is usually straightforward, but because each term in equation 20.a is a long vector and the operations involved, especially $\otimes$, are fairly costly, the conversion becomes a considerable and time-consuming task, and thus warrants special attention to parallelize it. The conversion entails computing the matrix A and the vectors $\overline{F}_j$'s, and defining the operation $\odot$.

The values of A and the $\overline{F}_j$'s are derived by repeated application of equation (20.a) so that each X_{tj-r}, for $r = 0, 1, ..., t-1$, is expressed in terms of $X_{t(j-1)-1}$, $X_{t(j-1)-2}$,...,$X_{t(j-1)-t}$ and F_{tj-r}, F_{tj-r-1},...,F_{tj-t}. We summarize next the outcome of that derivation/conversion process, leaving out the derivation details for the sake of brevity.

- The matrix A in equation (21) is a $t \times t$ matrix where each term $A_{i,s}$ is a vector of length n, for $i = 1, ..., t$ and $s = 1, 2, ..., t$. Specifically, $A_{i,s}$ satisfies the following equation

  $$A_{t,s} = B_s,\ A_{i-1,s} = B_s \otimes A_{i,1} + A_{i,s+1},\ i = t, t-1, ..., 2,\ s = 1, 2, ..., t \tag{22}$$

  where B_s was defined in equation (19) within Theorem 3.

- For every column vector $Y = [Y_1\ Y_2\ \ldots\ Y_t]^T$ where each Y_i is a vector of length n,

  $$A \odot Y = Z,\ where\ Z(i) = \sum_{s=1}^{t} A_{i,s} \otimes Y_s. \tag{23}$$

  Note that $A \odot Y$ is similar to a matrix-vector product except that the scalar multiplication operation is replaced by the vector convolution operation $\otimes$. Note also that if $\odot$ is performed on a system consisting of t subsystems, where subsystem i computes $Z(i)$, then the time $T_\odot = t \times T_\otimes + (t-1) \times T_+$. In particular, if each subsystem has n processors, then $T_\otimes = O(\log n)$ time by performing parallel convolution through FFT, and $T_+ = O(1)$ because it is a parallel vector addition. Hence, $T_\odot = O(tT_\otimes) = O(t \log n)$.

- Each column vectors $\overline{F}_j$ in equation 21 is of the form

  $$\overline{F}_j = [F_{j,1}\ F_{j,2}\ \ldots\ F_{j,t}]^T, \tag{24}$$

  where the terms $F_{j,r}$ are themselves vectors of length n, satisfying the following equation

  $$F_{j,t} = F_{tj-t},\ and$$

  $$for\ r = 1, 2, ..., t-1, F_{j,r} = F_{tj-r} + \sum_{s=r+1}^{t} A_{t+r+1-s,1} \otimes F_{tj-s},\ j = 1, 2, ..., n/t. \tag{25}$$

  Note that the F_{tj-r}'s are defined according to equation 20.b.

The general DPCM decoding algorithm must solve equation (21). For that purpose, it must first compute equations (18), (19), (20.b), (22) and (25). To understand the algorithm, it helps to view the $n \times n$ mesh architecture in two ways, one way as a sequence of n columns labeled globally $0, 1, \ldots, n-1$, and another way as partitioned into n/t contiguous subsystems of t contiguous columns each (n/t is assumed to be an integer for simplicity). The subsystems are labeled $1, ..., n/t$, and the columns within each subsystem are labeled locally $1, 2, ..., t$.

Data Assignment

Assume that each subsystem has the array a such that each column s of array a is stored in the s-th column of each subsystem, one term per processor. Recall again that for $m > k$, $a(m, s) = 0$. In addition to column $a(., 1)$, column 0 of the mesh has $a(., 0)$. The columns E_j and F_j are stored in column j of the mesh, that is, $E(i, j)$ and $F(i, j)$ are in processor (i, j). Each local column s of each subsystem j hosts vectors B_s, $A_{i,s}$ for all $i = 1, 2, ..., t$, and $F_{j,s}$.

Algorithm Index-2-order-k-t(**input:** $E, a(0 : k, 0 : t)$; **output:** $X(0 : n-1, 0 : n-1)$)
begin

1. Column 0 of the mesh computes $C(0 : n-1)$ by solving the 1-indexed scalar recurrence relation (18) of order k, using any standard parallel recurrence relation solver. Afterwards, C is broadcast row-wise to all the columns in the mesh.
2. **for** j=0 **to** $n-1$ **pardo**
3. Grid-column j does: $F_j = C \otimes E_j$ by a parallel convolution algorithm;
   endfor
4. **for** $s = 1$ **to** t **pardo**
5. Local column s of each subsystem does: $B_s = C \otimes a(0 : n-1, s)$; $A_{t,s} = B_s$;
   endfor
6. **for** $i = t$ **down to** 2 **do**
7. Local column 1 of each subsystem broadcasts $A_{i,1}$ rowwise to the columns of its subsystem;
8. Each column $s = 2, 3, ..., t$ in each subsystem sends rowwise the column vector $A_{i,s}$ to the previous mesh column;
   /* Steps 7 and 8 deliver data to processors to compute equation (22) next. */
9. **for** $s = 1$ **to** t **pardo**
10. Local column s of each subsystem does: $A_{i-1,s} = B_s \otimes A_{i,1} + A_{i,s+1}$;
    endfor
    endfor
11. **for** j=1 **to** n/t **pardo**
12. **for** r=1 **to** t **pardo**
13. Local column r of subsystem j does:
    it gets F_{tj-s} from local column s of subsystem j, for $s = r, r+1, ..., t$;
    it then computes $F_{j,r} = F_{tj-r} + \sum_{s=r+1}^{t} A_{t+r+1-s,1} \otimes F_{tj-s}$;
    endfor
    endfor
14. All the subsystems of the mesh work together to solve the 1-indexed recurrence relation $\overline{X}_j = A \odot \overline{X}_{j-1} + \overline{F}_j$ in parallel, where $\overline{X}_j$ resides in subsystem j;

end

Analysis

Step 1, being a 1-index scalar recurrence relation of order k, takes $O(k \log \frac{n}{k})$ time. The broadcast of C afterwards takes $O(1)$ time on partitionable buses, or $O(\log n)$ on a mesh of hypercubes. Step 3 (and thus step 2) takes $O(\log n)$ time, being a convolution. Similarly, steps 4 and 5 take $O(\log n)$ time for the same reason. The communication steps 7 and 8 take $O(1)$ time on a mesh of partitionable buses, or $O(\log n)$ on a mesh of hypercubes. Steps 9 and 10 take $O(\log n)$ time, since they involve a convolution and a vector addition. Step 13 requires each local column s in each subsystem j to broadcast its vector F_{tj-s} to several columns $r = 1, 2, ..., s-1$ within the same subsystem. Each such broadcast takes $O(1)$ time on partitionable buses, or $O(\log n)$ time on a mesh of hypercubes. Since step 13 requires t such broadcasts, which have to be performed one after another, the communication time of step 13 is $O(t)$ on partitionable buses, or $O(t \log n)$ on a mesh of hypercubes. After the broadcasts, the computation of $F_{j,r}$ in step 13 takes $O(t \log n)$. Step 14, being a 1-indexed recurrence relation of order 1, takes $O((T_{\odot} + T_{+}) \log \frac{n}{t})$. The time $T_{\odot} = O(tT_{\otimes}) = O(t \log n)$ as was explained right after equation (23). Therefore, the total time complexity of the algorithm is $O(k \log \frac{n}{k} + t \log n \log \frac{n}{t})$, which is dominated by $O(t \log^2 n)$ for small values of t and k. Since the sequential time of computing the original recurrence relation (15) is $O(ktn^2)$, it follows that the speedup is $O(kn^2 / \log^2 n)$ and the efficiency is $O(k / \log^2 n)$.

8 Generalization to Multi-indexed Relations

In this section we treat the most general case, namely, q-indexed relations of arbitrary order, where $q \geq 2$. The treatment will require a generalization of: (1) the 1D shift operator to $(q-1)$-dimensional shift operators, (2) the subsequent generalization of the "single-variable" polynomial operator to something that resembles a multivariable polynomial operator, and (3) finally the use of $(q-1)$-dimensional convolution rather than 1D convolution. The conversion process from k-indexed arbitrary-order relations to 1-indexed relations of order 1 is otherwise the same as in the previous sections.

Recall from equation (4) that a q-indexed recurrence relation of order $(t_1, t_2, ..., t_q)$ has the following form:

$$X(i_1, i_2, ..., i_q) = \sum_{r_1=0}^{t_1} \sum_{r_2=0}^{t_2} ... \sum_{r_q=0}^{t_q} a(r_1, r_2, ..., r_q) X(i_1 - r_1, i_2 - r_2, ..., i_q - r_q) + E(i_1, i_2, ..., i_q) \quad (26)$$

where $i_l = 0, 1, ..., N_l - 1$ for $l = 1, 2, ..., q$, the array $E(0 : N_1 - 1, 0 : N_2 - 1, ..., 0 : N_q - 1)$ and the array $a(0 : t_1, 0 : t_2, ..., 0 : t_q)$ of parameters are given, $a(0, 0, ..., 0)$ is equal to 0 so that $X(i_1, i_2, ..., i_q)$ does not depend on itself, and $t_l << N_l$ for $l = 1, 2, ..., q$. As before, any term is assumed to be zero if its indices are out of their defined bounds.

To simplify the multi-index cumbersome notation, we will introduce some notations and conventions. Let

- $\mathcal{R} = \{(r_1, r_2, ..., r_q) \mid r_l = 0, 1, ...t_l \; for \; all \; l = 1, 2, ..., q\}$, and any $\mathbf{r}$ in $\mathcal{R}$ is by convention a vector $\mathbf{r} = (r_1, r_2, ..., r_q)$.

- $\mathcal{R}' = \{(r_1, r_2, ..., r_{q-1}) \mid r_l = 0, 1, ...t_l \ for\ all\ l = 1, 2, ..., q-1\}$.
- $\mathcal{R}^* = \mathcal{R} - \{(0, 0, ..., 0)\}$, that is, the set $\mathcal{R}$ except of the zero-tuple.
- $\mathcal{R}'^* = \mathcal{R}' - \{(0, 0, ..., 0)\}$.
- $\mathcal{I} = \{(i_1, i_2, ..., i_q) \mid i_l = 0, 1, ...N_l - 1\ for\ all\ l = 1, 2, ..., q\}$, and any $\imath$ in $\mathcal{I}$ is by convention a vector $\imath = (i_1, i_2, ..., i_q)$.
- $\mathcal{I}' = \{(i_1, i_2, ..., i_{q-1}) \mid i_l = 0, 1, ...N_l - 1\ for\ all\ l = 1, 2, ..., q-1\}$.
- Convention: If $\imath = (r_1, r_2, ..., r_q)$ is an index vector in $\mathcal{R}$, then $\imath'$ is vector derived from $\imath$ by dropping the last term of $\imath$. Clearly, $\imath' = (r_1, r_2, ..., r_{q-1}) \in \mathcal{R}'$, and $(\imath', r_q) = \imath$. Similarly, for any index vector $\imath$ in $\mathcal{I}$, denote by $\imath'$ the vector derived from $\imath$ by dropping the last term from the latter.
- Convention: For any q-dimensional array M and any integer i_q, denote by M_{i_q} the $(q-1)$-dimensional array such that $M_{i_q}(i_1, i_2, ..., i_{q-1}) = M(i_1, i_2, ..., i_{q-1}, i_q)$. Equivalently, $M_{i_q}(\imath') = M(\imath)$.

Accordingly, equation (26) becomes

$$X(\imath) = \sum_{\imath \in \mathcal{R}^*} a(\imath) X(\imath - \imath) + E(\imath). \tag{27}$$

Using the last two conventions indicated above, the last equation translates to

$$X_{i_q}(\imath') = \sum_{\imath \in \mathcal{R}^*} a(\imath) X_{i_q - r_q}(\imath' - \imath') + E_{i_q}(\imath').$$

Split the multiple summation on the right hand side of the previous equation into two parts: one for $r_q = 0$ and another for $r_q \neq 0$. This leads to

$$X_{i_q}(\imath') = \sum_{\imath' \in \mathcal{R}'^*} a(\imath', 0) X_{i_q}(\imath' - \imath') + \sum_{r_q=1}^{t_q} \sum_{\imath' \in \mathcal{R}'} a(\imath', r_q) X_{i_q - r_q}(\imath' - \imath') + E_{i_q}(\imath'). \tag{28}$$

At this point we introduce the generalization of the shift operator. For each vector $\imath' = (r_1, r_2, ..., r_{q-1})$ of non-negative components, define the shift operator $\mathcal{S}_{\imath'}$ to operate on $(q-1)$-dimensional arrays V of size $N_1 \times N_2 \times \cdots \times N_{q-1}$ as follows: $\mathcal{S}_{\imath'} V$ is vector of the same dimensional structure as V such that $(\mathcal{S}_{\imath'} V)(\imath') = V(\imath' - \imath')$ for all $\imath' \in \mathcal{I}'$. Therefore, $X_{i_q}(\imath' - \imath') = (\mathcal{S}_{\imath'} X_{i_q})(\imath')$. Note that if $r_l \geq N_l$ for some l, then $\mathcal{S}_{\imath'}$ is the zero operator, that is $\mathcal{S}_{\imath'} V$ is equal to the zero array of size $N_1 \times N_2 \times \cdots \times N_{q-1}$. Note also that if $\imath'$ and $\jmath'$ are two index vectors in $\mathcal{R}'$, then it can be easily verified that $\mathcal{S}_{\imath'} \mathcal{S}_{\jmath'} = \mathcal{S}_{\imath' + \jmath'}$. Indeed, we can correspond to $\mathcal{S}_{\imath'}$ a polynomial term $x_1^{r_1} x_2^{r_2} ... x_{q-1}^{r_{q-1}}$ of $q-1$ variables; the product (i.e., the composition) of two operators $\mathcal{S}_{\imath'} \mathcal{S}_{\jmath'}$ is equivalent to the product of their corresponding polynomial terms. Hence, one can define "multivariable" polynomial operators.

Definition 3 *Given a $(q-1)$-dimensional real array $T(\imath')$, $\imath' \in \mathcal{I}'$, the operator $\mathcal{T} = \sum_{\imath' \in \mathcal{I}'} T(\imath')\mathcal{S}_{\imath'}$, called the multivariable polynomial operator of characteristic array T, is such that for any array V of size $N_1 \times N_2 \times \cdots \times N_{q-1}$, $\mathcal{T}V = \sum_{\imath' \in \mathcal{I}'} T(\imath')(\mathcal{S}_{\imath'}V)$.*

Equation (28) can now be put in the following form:

$$X_{i_q}(\imath') = (\sum_{R' \in \mathcal{R}'^*} a(R',0)\mathcal{S}_{R'}X_{i_q})(\imath') + \sum_{r_q=1}^{t_q}(\sum_{R' \in \mathcal{R}'} a(R',r_q)\mathcal{S}_{R'}X_{i_q-r_q})(\imath') + E_{i_q}(\imath'). \tag{29}$$

Since, the vector-index $\imath'$ is the same in all the terms of the previous equation, we can drop it resulting in a compact $(q-1)$-dimensional array equation

$$X_{i_q} = \sum_{R' \in \mathcal{R}'^*} a(R',0)\mathcal{S}_{R'}X_{i_q} + \sum_{r_q=1}^{t_q}\sum_{R' \in \mathcal{R}'} a(R',r_q)\mathcal{S}_{R'}X_{i_q-r_q} + E_{i_q}. \tag{30}$$

Next, move the first summation on the right hand side to the left, and factor X_{i_q} out. This results in

$$(1 - \sum_{R' \in \mathcal{R}'^*} a(R',0)\mathcal{S}_{R'})X_{i_q} = \sum_{r_q=1}^{t_q}\sum_{R' \in \mathcal{R}'} a(R',r_q)\mathcal{S}_{R'}X_{i_q-r_q} + E_{i_q}. \tag{31}$$

We simplify the notation by defining

$$\mathcal{T}_0 = 1 - \sum_{R' \in \mathcal{R}'^*} a(R',0)\mathcal{S}_{R'}, \quad \mathcal{T}_{r_q} = \sum_{R' \in \mathcal{R}'} a(R',r_q)\mathcal{S}_{R'}, \ for\ r_q = 1,2,...,t_q.$$

As a result, equation (31) becomes $\mathcal{T}_0 X_{i_q} = \sum_{r_q=1}^{t_q} \mathcal{T}_{r_q} X_{i_q-r_q} + E_{i_q}$, leading to

$$X_{i_q} = \sum_{r_q=1}^{t_q} \mathcal{T}_0^{-1}\mathcal{T}_{r_q} X_{i_q-r_q} + \mathcal{T}_0^{-1}E_{i_q} \tag{32}$$

The next theorem, which combines and generalizes Theorems 2 and 3 to multivariable polynomial operators, will show that the application of a multivariable polynomial operator to an an array is equivalent to multidimensional partial convolution, that the inverse of $\mathcal{T}$ exists and can be computed by solving recursively a $(q-1)$-index recurrence relation of order $(t_1, t_2, ..., t_{q-1})$, and that the product of two multivariable polynomial operators is a polynomial operator whose characteristic array is the partial convolution of the characteristic arrays of the factor operators. To that effect, we first define partial multidimensional convolution, and then prove the theorem afterwards.

Definition 4 *The partial convolution of two $N_1 \times N_2 \cdots \times N_{q-1}$ arrays U and V is an $N_1 \times N_2 \cdots \times N_{q-1}$ array W, denoted $W = U \otimes V$, such that*
$W(i_1, i_2, ..., i_{q-1}) = \sum_{r_1=0}^{i_1}\sum_{r_2=0}^{i_2} \cdots \sum_{r_{q-1}=0}^{i_{q-1}} U(r_1, r_2, ..., r_{q-1})V(i_1 - r_1, i_2 - r_2, ..., i_{q-1} - r_{q-1})$.

Theorem 4 *1) Let T and V be two $N_1 \times N_2 \times \cdots \times N_{q-1}$ arrays, and $\mathcal{T}$ the multivariable polynomial operator of characteristic array T. Then, $\mathcal{T}V = T \otimes V$.*
2) $\mathcal{T}_0^{-1} = \sum_{I' \in \mathcal{I}'} C(I')\mathcal{S}_{I'}$, where C is an $N_1 \times N_2 \times \cdots \times N_{q-1}$ array that satisfies the following $(q-1)$-index recurrence relation of order $(t_1, t_2, ..., t_{q-1})$:

$$C(0.0, ... 0) = 1, \quad C(I') = \sum_{R' \in \mathcal{R}'} a(R', 0)C(I' - R') \tag{33}$$

3) For every $i_q = 1, 2, ..., t_q$, $\mathcal{T}_0^{-1}\mathcal{T}_{i_q} = \sum_{I' \in hIp} B_{i_q}(I')\mathcal{S}_{I'}$, where

$$B_{i_q} = C \otimes a_{i_q}, \tag{34}$$

and a_{i_q} is an $N_1 \times N_2 \times \cdots \times N_{q-1}$ array such that $a_{i_q}(i_1, i_2, ..., i_{q-1}) = a(i_1, i_2, ..., i_{q-1}, i_q)$ if $(i_1, i_2, ..., i_{q-1})$ is in $\mathcal{R}'$, and equal to zero otherwise.

Proof: The proof of (1) follows the same reasoning as in the proof of Theorem 2. The proof of (2) and (3) follows the same reasoning as in the proof of Theorem 3. ■

Now, by using the previous theorem, we convert equation (32) to a form identical to equations (20) (correspond i_q to j and r_q to s):

$$X_{i_q} = \sum_{r_q=1}^{t_q} B_{r_q} \otimes X_{i_q - r_q} + F_{i_q} \tag{35.a}$$
$$F_{i_q} = C \otimes E_{i_q}. \tag{35.b}$$

Clearly, equation (35.a) is a 1-indexed recurrence relation of order t_q, where every term is an $N_1 \times N_2 \times \cdots \times N_{q-1}$ array. This completes the dimension shifting from index dimension to term dimension. The shifting of the order dimension to term dimension from here on mirrors the same process conducted in the previous section starting from equation (21) onward, and, therefore, will not be repeated. The resulting 1-index recurrence relation of order 1, similar to equation (21), is

$$\overline{X}_{i_q} = A \odot \overline{X}_{i_q - 1} + \overline{F}_{i_q} \tag{36}$$

The definitions of A, $\overline{X}_{i_q}$, $\overline{F}_{i_q}$, F_{i_q, r_q}, and $\odot$ are formally the same as in the previous section, with the proper substitutions. Note in particular that A is a $t_q \times t_q$ matrix where every term $A_{i,s}$ is an $N_1 \times N_2 \times \cdots \times N_{q-1}$ array, and the convolution $\otimes$ involved in the definition of $\odot$ is now a $(q-1)$-dimensional convolution on $N_1 \times N_2 \times \cdots \times N_{q-1}$ arrays.

The architecture is naturally an $N_1 \times N_2 \times \cdots \times N_q$ mesh of hypercubes or of partitionable buses. As in the previous section, the mesh can be viewed as N_q *submeshes* (as opposed to columns), each being an $N_1 \times N_2 \times \cdots \times N_{q-1}$ mesh. The submeshes are labeled $0, 1, ..., N_q - 1$. On the other hand, the whole mesh can be viewed as partitioned into N_q/t_q contiguous *subsystems* of t_q contiguous submeshes each. The subsystems are labeled $1, 2, ..., N_q/t_q$, and the submeshes within each subsystem are labeled locally $1, 2, ..., t_q$.

Data Assignment
Assume that each subsystem has the parameter array a such that the (sub)array a_{r_q} is stored in the r_q-th submesh of each subsystem, one term per processor. In addition to array a_1, submesh 0 of the mesh has the array a_0. The arrays E_{i_q} and F_{i_q} are stored in submesh i_q, that is, $E(i_1, i_2, ..., i_q)$ and $F(i_1, i_2, ..., i_q)$ are in processor $(i_1, i_2, ..., i_q)$. Each local submesh r_q of each subsystem i_q hosts arrays B_{r_q}, A_{i, r_q} for all $i = 1, 2, ..., t_q$, and F_{i_q, r_q}.

The algorithm for solving equation (36) is largely identical to the Index-2-order-k-t algorithm for equation (21) presented in the previous section. To map the latter algorithm to an algorithm for equation (36), only three major changes are needed:

- Step 1 of the algorithm must be a recursive call to compute the array C of the $(q-1)$-indexed recurrence relation (33) of order $(t_1, t_2, ..., t_{q-1})$.
- Add a basis step corresponding to the case where $q = 2$, in order for recursion to work.
- Substitute "submesh" for "column" and "Grid-column", "$(q-1)$-dimensional convolution" for "convolution", "array" for "vector column", t_q for t, N_q for n, i_q for j, r_q for s.

Algorithm General(**input:** $E(0:N_1,...,0:N_q), a(0:t_1,...,0:t_q)$; **out:** $X(0:N_1,...,0:N_q)$)
begin

0. Basis step: if $q = 2$, call algorithm Index-2-order-k-t and return;
1. Submesh 0 calls the algorithm "General" recursively to compute $C(0:N_1,...,0:N_{q-1})$ by solving the $(q-1)$-indexed scalar recurrence relation (33) of order $(t_1, t_2, ..., t_{q-1})$. Afterwards, C is broadcast along dimension q to all the submeshes.
2. **for** i_q=0 **to** $N_q - 1$ **pardo**
3. Submesh i_q does: $F_{i_q} = C \otimes E_{i_q}$ by a parallel convolution algorithm;
   endfor
4. **for** $r_q = 1$ **to** t_q **pardo**
5. Local submesh r_q of each subsystem does: $B_{r_q} = C \otimes a_{r_q}$; $A_{t_q,r_q} = B_{r_q}$;
   endfor
6. **for** $i = t_q$ **down to** 2 **do**
7. Local submesh 1 of each subsystem broadcasts $A_{i,1}$ along dimension q to all the submeshes of its subsystem;
8. Each submesh $r_q = 2, 3, ..., t_q$ in each subsystem sends the vector array A_{i,r_q} one step down along dimension q in the mesh;
   /* Steps 7 and 8 deliver data to processors to compute A next. */
9. **for** $r_q = 1$ **to** t_q **pardo**
10. Local submesh r_q of each subsystem does: $A_{i-1,r_q} = B_{r_q} \otimes A_{i,1} + A_{i,r_q+1}$;
    endfor
   endfor
11. **for** i_q=1 **to** N_q/t_q **pardo**
12. **for** r=1 **to** t_q **pardo**
13. Local submesh r of subsystem i_q does:
    it gets $F_{t_q i_q - r_q}$ from local submesh r_q of subsystem i_q, for $r_q = r, r+1, ..., t_q$;
    it then computes $F_{i_q,r} = F_{t_q i_q - r} + \sum_{r_q=r+1}^{t_q} A_{t_q+r+1-r_q,1} \otimes F_{t_q i_q - r_q}$;
    endfor
   endfor
14. All the subsystems of the mesh work together to solve the 1-indexed recurrence relation $\overline{X}_{i_q} = A \odot \overline{X}_{i_q-1} + \overline{F}_{i_q}$ in parallel, where $\overline{X}_{i_q}$ resides in subsystem i_q;

end

Analysis

Let $T(N_1, t_1, N_2, t_2, ..., N_q, t_q)$ be the parallel time complexity of the whole algorithm, and let $N = N_1 N_2 ... N_q$. Step 1, being a recursive step, takes $T(N_1, t_1, N_2, t_2, ..., N_{q-1}, t_{q-1})$ time. The time of all the remaining steps is dominated by the last step, step 14, which is the solution of the 1-indexed recurrence relation of order 1, equation (36), and thus takes $O((T_{\odot}+T_{+}) \log \frac{N_q}{t_q})$ time. Note that $\odot$ is a matrix-vector product where the matrix is $t_q \times t_q$ and the multiplication operation is replaced by $(q-1)$-dimensional $\otimes$. Multidimensional convolution can be implemented by multidimensional FFT, performed one dimension after another, in $O(\log N_1 + \log N_2 + \cdots + \log N_{q-1}) = O(\log(N_1 N_2 ... N_{q-1}))$ time because FFT in dimension l takes $O(\log N_l)$ time. Since $T_{\odot} = O(t_q T_{\otimes})$ (as was explained after equation 23), it follows that $T_{\odot} = O(t_q \log(N/N_q))$. Since also $T_{+} = O(1)$, it follows that step 14 takes $O(t_q \log(N_1 N_2 ... N_{q-1}) \log(N_q/t_q))$. Consequently, we have

$$T(N_1, t_1, ..., N_q, t_q) = T(N_1, t_1, ..., N_{q-1}, t_{q-1}) + O(t_q \log(N_1 ... N_{q-1}) \log(N_q/t_q)).$$

This recurrence relation easily yields that

$$T(N_1, t_1, N_2, t_2, ..., N_q, t_q) = T(N_1, t_1) + O(\textstyle\sum_{l=2}^{q} t_l \log(N_1 ... N_{l-1}) \log(N_l/t_l))$$

Clearly, $T(N_1, t_1) = O(t_1 \log \frac{N_1}{t_1})$ because it corresponds to solving a scalar 1-indexed recurrence relation of order t_1. Since $\log(N_1 ... N_{l-1}) \log(N_l/t_l) \leq \log N \log(N_l/t_l) \leq \log N \log N_l \leq \log^2 N$, the time formula for the whole algorithm simplifies to

$$T(N_1, t_1, N_2, t_2, ..., N_q, t_q) = O((\sum_{l=1}^{q} t_l) \log^2 N).$$

Since the sequential time of solving equation (26) is $O(t_1 t_2 ... t_q N)$, it follows that the speedup is $O(\frac{t_1 t_2 ... t_q}{\sum_{l=1}^{q} t_l} \frac{N}{\log^2 N})$, and the efficiency is $O(\frac{t_1 t_2 ... t_q}{(\sum_{l=1}^{q} t_l) \log^2 N})$, where N is the size of the output array X. In the typical cases where the t_l's are small constants, the overall time becomes $O(\log^2 N)$, and the speedup and efficiency become $O(N/\log^2 N)$ and $O(1/\log^2 N)$, respectively.

9 Conclusions

In this paper we developed a dimension-shifting approach for novel parallel solution of multi-indexed recurrence relations of arbitrary order in square-logarithmic time, using a mesh of hypercubes or of partitionable buses. Multi-indexed recurrence relations have not been addressed before, but hold special significance due to their application to DPCM, which is a standard image compression technique.

We introduced two other techniques, index sequencing and index decoupling. The first is very simple and works well for a small number of processors, but does not exploit all the potential for parallelism. The second is limited in its applicability, but when applicable it is faster than the other two techniques. Indeed, it takes logarithmic time, which is optimal.

For future work, it is of practical interest to consider how best to load-balance and possibly pipeline the algorithms "Index-2-order-k-t" and "General" on a system of fewer processors than the size of the output X, for the purpose of minimizing the parallel time complexity.

References

[1] Blelloch, G. E., "Scans as Primitive Parallel Operations," *IEEE Trans. Comput.,* C-38(11),pp. 1526–1538, Nov. 1989.

[2] S. Chen and D. J. Kuck, "Time and Parallel Processor Bounds for Linear Recurrence Systems," *IEEE Trans. Comput.,* C-24(7), July 1975.

[3] J. W. Cooley and J. W. Tuckey, "An Algorithm for the Machine Calculation of Complex Fourier Series," *Math. of Comput.,* Vol. 19, pp. 297–301, 1965.

[4] F.E. Fich, "New Bounds for Parallel Prefix Circuits," *Proceedings ACM Symposium on Theory of Computing,* pp. 100–109, Apr. 1983.

[5] R. Gonzales and R. Woods, *Digital Image Processing,* (Chapter 3) Addison-Wesley, 1992.

[6] L. Hyafil and H. T. Kung, "The Complexity of Parallel Evaluation of Linear Recurrences," *Journal of the Association for Computing Machinery,* 24(3), pp. 513–521, July 1977.

[7] P. M. Kogge and H. S. Stone, "A parallel algorithm for the efficient solution of a general class of recurrence equations," *IEEE Trans. Comput.,* Vol. C-22, No. 8, pp. 786–793, Aug. 1973.

[8] C. P. Kruskal, L. Rudolph and M. Snir, "The power of parallel prefix," *IEEE Trans. Comput.'* Vol. 34, pp. 965–968, 1985.

[9] R. E. Ladner and M. J. Fischer, "Parallel prefix Computation," *Journal of ACM* Vol. 27, pp. 831–838, 1980.

[10] S. Lakshmivarahan, C. M. Yang, and S. K. Dhall, "Optimal Parallel Prefix Circuits with (Size+Depth)=$2n - n$ and $\lceil \log n \rceil \leq depth \leq \lceil 2 \log n \rceil - 3$," *Proceedings International Conference on Parallel Processing,* pp. 58–65, Aug. 1987.

[11] B. D. Lubachevsky and A. G. Greenberg, "Efficient Asynchronous Parallel Prefix Algorithms," *Proceedings International Conference on Parallel Processing,* pp. 66-69, Aug. 1987.

[12] B. Pennebaker and J. L. Mitchell, *JPEG Still Image Data Compression Standard,* Van Nostrand Reinhold, New York, 1993.

[13] J. H. Reif, Ed., *Synthesis of Parallel Algorithms,* Chapter 1, Morgan Kaufmann Publishers, 1993.

[14] K. Sayood, *Introduction to Data Compression,* Morgan Kauffmann Publishers, San Fransisco, California, 1996.

[15] M. Snir, "Depth-size trade-off for parallel prefix computation," *Journal of Algorithms,* Vol. 7, pp. 185–201, 1986.

[16] H. S. Stone, "Parallel Processing with the Perfect Shuffle," *IEEE Transactions on Computers,* C-20(2), pp. 153-161, 1971.

[17] H. S. Stone, "An Efficient Parallel Algorithm for the Solution of a Tridiagonal Linear System of Equations," *Journal of the Association for Computing Machinery,* 20(1),pp. 27–38, Jan. 1973.

[18] H. S. Stone, "Parallel Tridiagonal Equation Solvers," *ACM Transactions on Mathematical Software,* 1(4), pp. 289–307, Dec. 1975.

[19] A. Youssef, "On-Line Communication on Circuit-Switched Fixed Routing Meshes," *the Sixth International Parallel Processing Symposium,* Beverly Hills, California, pp. 390-397, March 1992.

[20] A. Youssef, "Translation of Serial Recursive Codes to Parallel SIMD Codes," *Proc. of the 3rd Conference on Parallel Architecture and Compilation Techniques,* Limassol, Cyprus, pp. 254–263, July 1995.