Multidimensional Synchronous Dataflow

Praveen K. Murthy, *Member, IEEE,* and Edward A. Lee, *Fellow, IEEE*

***Abstract*—Signal flow graphs with dataflow semantics have been used in signal processing system simulation, algorithm development, and real-time system design. Dataflow semantics implicitly expose function parallelism by imposing only a partial ordering constraint on the execution of functions. One particular form of dataflow called synchronous dataflow (SDF) has been quite popular in programming environments for digital signal processing (DSP) since it has strong formal properties and is ideally suited for expressing multirate DSP algorithms. However, SDF and other dataflow models use first-in first-out (FIFO) queues on the communication channels and are thus ideally suited only for one-dimensional (1-D) signal processing algorithms. While multidimensional systems can also be expressed by collapsing arrays into 1-D streams, such modeling is often awkward and can obscure potential data parallelism that might be present.**

SDF can be generalized to multiple dimensions; this model is called multidimensional synchronous dataflow (MDSDF). This paper presents MDSDF and shows how MDSDF can be efficiently used to model a variety of multidimensional DSP systems, as well as other types of systems that are not modeled elegantly in SDF. However, MDSDF generalizes the FIFO queues used in SDF to arrays and, thus, is capable only of expressing systems sampled on rectangular lattices. This paper also presents a generalization of MDSDF that is capable of handling arbitrary sampling lattices and lattice-changing operations such as nonrectangular decimation and interpolation. An example of a practical system is given to show the usefulness of this model. The key challenge in generalizing the MDSDF model is preserving static schedulability, which eliminates the overhead associated with dynamic scheduling, and preserving a model where data parallelism, as well as functional parallelism, is fully explicit.

I. Introduction

OVER the past few years, there has been increasing interest in dataflow models of computation for digital signal processing (DSP) because of the proliferation of block diagram programming environments for specifying and rapidly prototyping DSP systems. Dataflow is a very natural abstraction for a block-diagram language, and many subsets of dataflow have attractive mathematical properties that make them useful as the basis for these block-diagram programming environments. Visual languages have always been attractive in the engineering community, especially in computer-aided design, because engineers most often conceptualize their systems in terms of hierarchical block diagrams or flowcharts. The 1980s witnessed the acceptance in industry of logic-synthesis tools, in which circuits are usually described graphically by block diagrams, and one expects the trend to continue in the evolving field of high-level synthesis and rapid prototyping.

Synchronous dataflow and its variants have been quite popular in design environments for DSP. Reasons for its popularity include its strong formal properties like deadlock detection, determinacy, static schedulability, and, finally, its ability to model multirate DSP applications (like filterbanks) well, in addition to nonmultirate DSP applications (like IIR filters). Static schedulability is important because to get competitive real-time implementations of signal processing applications, dynamic sequencing, which adds overhead, should be avoided whenever possible. The overhead issue becomes even more crucial for image and video signal processing where the throughput requirements are even more stringent.

The SDF model suffers from the limitation that its streams are one-dimensional (1-D). For multidimensional signal processing algorithms, it is necessary to have a model where this restriction is not there so that effective use can be made of the inherent data-parallelism that exists in such systems. As is the case for 1-D systems, the specification model for multidimensional systems should expose, to the compiler or hardware synthesis tool, as much static information as possible so that run-time decisionmaking is avoided as much as possible and so that effective use can be made of both functional and data parallelism. Although a multidimensional stream can be embedded within a 1-D stream, it may be awkward to do so [10]. In particular, compile-time information about the flow of control may not be immediately evident. Most multidimensional signal processing systems also have a predictable flow of control, like 1-D systems, and for this reason, an extension of SDF, called multidimensional synchronous dataflow, was proposed in [20]. However, the MDSDF model developed in [20] is restricted to modeling systems that use rectangular sampling structures. Since there are many practical systems that use nonrectangular sampling and nonrectangular decimation and interpolation, it is of interest to have models capable of expressing these systems. Moreover, the model should be statically schedulable if possible, as already mentioned, and should expose all of the data and functional parallelism that might be present so that a good scheduler can make use of it. While there has been some progress in developing block-diagram environments for multidimensional signal processing, like the Khoros system [17], none, as far as we know, allow modeling of arbitrary sampling lattices at a fine-grained level, as shown in this paper.

Manuscript received March 2, 2001; revised February 28, 2002. This work was supported in part, by the Ptolemy project of the Defence Advanced Research Projects Agency, the U. S. Air Force under the RASSP program under Contract F33615–93-C-1317, Semiconductor Research Corporation under project 94-DC-008, the National Science Foundation under Grant MIP-9201605, the Office of Naval Technology (via Naval Research Laboratories), the State of California MICRO program, and the following companies: Bell Northern Research, Dolby, Hitachi, Mentor Graphics, Mitsubishi, NEC, Pacific Bell, Phillips, Rockwell, Sony, and Synopsys. The associate editor coordinating the review of this paper and approving it for publication was Dr. Edwin Hsing-Men Sha.

P. K. Murthy is with the Fujitsu Labs of America, Sunnyvale, CA 94085 USA (e-mail: p.murthy@fla.fujitsu.com).

E. A. Lee is with the Department of Electrical Engineering and Computer Science, University of California, Berkeley, CA 94720 USA (e-mail: eal@eecs.berkeley.edu)

Publisher Item Identifier 10.1109/TSP.2002.800830.

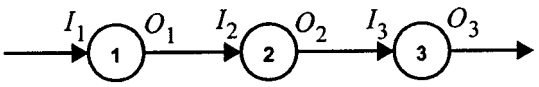

Fig. 1. Simple synchronous dataflow graph.

The paper is organized as follows. In Section I-A, we review the SDF model and describe the MDSDF model in Section II. In Section II-A–G, we describe the types of systems that may be described using MDSDF graphs. In Section III, we develop a generalization of the MDSDF model to allow arbitrary sampling lattices and arbitrary decimation and interpolation. We give an example of a practical video aspect ratio conversion system in Section IV that can be modeled in the generalized form of MDSDF. In Section V, we discuss related work of other researchers and conclude the paper in Section VI.

A. Synchronous Dataflow

For several years, we have been developing software environments for signal processing that are based on a special case of dataflow that we call synchronous dataflow (SDF) [19]. The Ptolemy [8], [11] program uses this model. It has also been used in Aachen, Germany, [29] in the COSSAP system and at Carnegie Mellon University, Pittsburgh, PA, [28] for programming the Warp. Industrial tools making use of dataflow models for signal processing include System Canvas and DSP Canvas from Angeles Design Systems [24], the Cocentric System Studio from Synopsys, and the Signal Processing Worksystem from Cadence Design Systems. SDF graphs consist of networks of actors connected by arcs that carry data. However, these actors are constrained to produce and consume a fixed integer number of tokens on each input or output path when they fire [19]. The term "synchronous" refers to this constraint and arises from the observation that the rates of production and consumption of tokens on all arcs are related by rational multiples. Unlike the "synchronous" languages Lustre [9] and Signal [2], however, there is no notion of clocks. Tokens form ordered sequences, where only the ordering is important.

Consider the simple graph in Fig. 1. The symbols adjacent to the inputs and outputs of the actors represent the number of tokens consumed or produced (also called rates). Most SDF properties follow from the *balance equations*, which for the graph in Fig. 1 are

$$r_1 O_1 = r_2 I_2,\ r_2 O_2 = r_3 I_3.$$

The symbols r_i represent the number of firings (repetitions) of an actor in a cyclic schedule and are collected in vector form as $\vec{r}^T = [r_1 \quad r_2 \quad r_3]$. Given a graph, the compiler solves the balance equations for these values r_i. As shown in [19], for a system of these balance equations, either there is no solution at all, in which case the SDF graph is deemed to defective due to inconsistent rates, or there are an infinite number of nonzero solutions. However, the infinite number of nonzero solutions are all integer multiples of the smallest solution $k\vec{r}, k = 0, 1, \ldots$, and this smallest solution $\vec{r}$ exists and is unique [19]. The number k is called the *blocking factor*. In this paper, we will assume that the solution to the balance equations is always this smallest, nonzero one (i.e, the blocking factor is 1). Given this solution, a precedence graph can be automatically constructed, specifying the partial ordering constrains between firings [19]. From this precedence graph, good compile-time multiprocessor schedules can be automatically constructed [30].

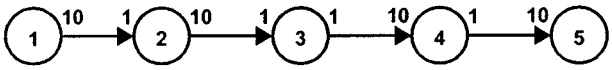

Fig. 2. Nested iteration described using SDF.

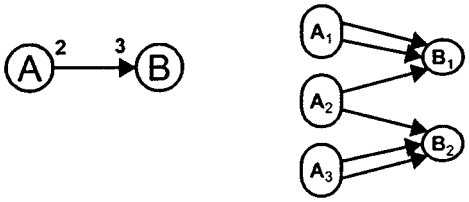

Fig. 3. SDF graph and its corresponding precedence graph.

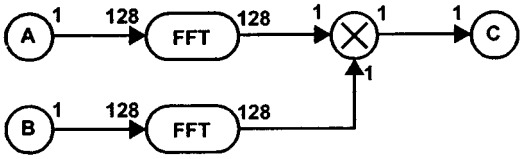

Fig. 4. Application of SDF to vector operations.

SDF allows a compact and intuitive expression of predictable control flow and is easy for a compiler to analyze. Consider, for instance, the SDF graph in Fig. 2. The balance equations can be solved to give the smallest nonzero integer repetitions for each actor (collected in vector form) as $\vec{r}^T = [1 \quad 10 \quad 100 \quad 10 \quad 1]$, which indicates that for every firing of actor 1, there will be ten firings of actor 2, 100 of 3, ten of 4, and one of 5. Hence, this represents nested iteration.

More interesting control flow can be specified using SDF. Fig. 3 shows two actors with a 2/3 producer/consumer relationship. From such a multirate SDF graph, we can construct a precedence graph that explicitly shows each invocation of the actor in the complete schedule and the precedence relations between different invocations of the actor. For the example of Fig. 3, the complete schedule requires three invocations of A and two of B. Hence, the precedence graph, shown to the right in Fig. 3, contains three A nodes and two B nodes, and the arcs in the graph reflect the order in which tokens are consumed in the SDF graph; for instance, the second firing of A produces tokens that are consumed by both the first and second firings of B. From the precedence graph, we can construct the sequential schedule $(A_1, A_2, B_1, A_3, B_2)$, among many possibilities. This schedule is not a simple nested loop, although schedules with simple nested loop structure can be constructed systematically [4]. Notice that unlike the "synchronous" languages Lustre and Signal, we do not need the notion of clocks to establish a relationship between the stream into actor A and the stream out of actor B.

The application of this model to multirate signal processing is described in [7]. An application to vector operations is shown in Fig. 4, where two fast Fourier transforms (FFTs) are multiplied. Both function and data parallelism are evident in the

precedence graph that can be automatically constructed from this description. That precedence graph would show that the FFTs can proceed in parallel and that all 128 invocations of the multiplication can be invoked in parallel. Furthermore, the FFT might be internally specified as a dataflow graph, permitting exploitation of parallelism within each FFT as well. The Ptolemy system [8] can use this model to implement overlap-and-add or overlap-and-save convolution, for example.

II. Multidimensional Dataflow

The multidimensional SDF model is a straightforward extension of 1-D SDF. Fig. 5 shows a trivially simple 2-D SDF graph. The number of tokens produced and consumed are now given as M-tuples, for some natural number M. Instead of one balance equation for each arc, there are now M. The balance equations for Fig. 5 are

$$r_{A,1}O_{A,1} = r_{B,1}I_{B,1},\ r_{A,2}O_{A,2} = r_{B,2}I_{B,2}.$$

These equations should be solved for the smallest integers $r_{X,i}$, which then give the number of repetitions of actor X in dimension i. We can also associate a *blocking factor vector* with this solution, where the vector has M dimensions, and each dimension represents the blocking factor for the solution to the balance equations of that dimension.

A. Application to Image Processing

As a simple application of MDSDF, consider a portion of an image coding system that takes a 40×48 pixel image and divides it into 8×8 blocks on which it computes a DCT. At the top level of the hierarchy, the dataflow graph is shown in Fig. 6(a). The solution to the balance equations is given by $r_{A,1} = r_{A,2} = 1,\ r_{DCT,1} = 5,\ r_{DCT,2} = 6$.

A segment of the index space for the stream on the arc connecting actor A to the DCT is shown in Fig. 6(b). The segment corresponds to one firing of actor A. The space is divided into regions of tokens that are consumed on each of the five vertical firings of each of the six horizontal firings. The precedence graph constructed automatically from this would show that the 30 firings of the DCT are independent of one another and, hence, could proceed in parallel. Distribution of data to these independent firings can be automated.

B. Flexible Data Exchange

Application of MDSDF to multidimensional signal processing is obvious. There are, however, many less obvious applications. Consider the graph in Fig. 3. Note that the first firing of A produces two samples consumed by the first firing of B. Suppose instead that we wish for the firing of A_1 to produce the first sample for each of B_1 and B_2. This can be obtained using MDSDF as shown in Fig. 7. Here, each firing of A produces data consumed by each firing of B, resulting in a pattern of data exchange quite different from that in Fig. 3. The precedence graph in Fig. 7 shows this. The index space of the tokens transferred along the arc is also shown, with the left-most column indicating the tokens produced by the first firing of A and the top row indicating the tokens consumed by the first firing of B.

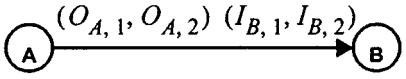

Fig. 5. Simple MDSDF graph.

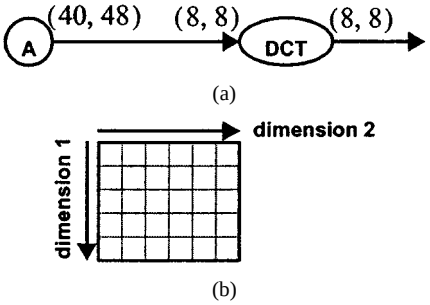

Fig. 6. (a) Image processing application in MDSDF. (b) Index space.

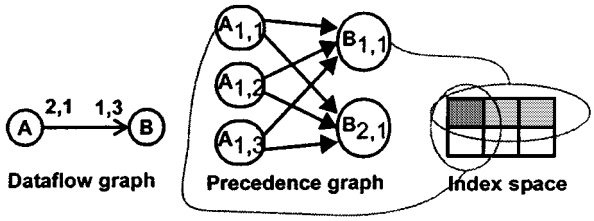

Fig. 7. Data exchange in an MDSDF graph.

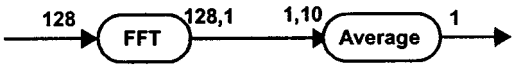

Fig. 8. Averaging successive FFT's using MDSDF.

A DSP application of this more flexible data exchange is shown in Fig. 8. Here, ten successive FFTs are averaged. Averaging in each frequency bin is independent and, hence, may proceed in parallel. The ten successive FFTs are also independent; therefore, if all input samples are available, they too may proceed in parallel.

A more complicated example of how the flexible data-exchange mechanism in an MDSDF graph can be useful in practice is shown in Fig. 9(a), which shows how a n-layer perceptron (with a nodes in the first layer, b nodes in the second layer, etc.) can be specified in a very compact way using only n nodes. However, as the precedence graph in Fig. 9(b) shows, none of the parallelism in the network is lost; it can be easily exploited by a good scheduler. Note that the net of Fig. 9(a) is used only for computation once the weights have been trained. Specifying the training mechanism as well would require feedback arcs with the appropriate delays and some control constructs; this is beyond the scope of this paper.

C. Delays

A delay in MDSDF is associated with a tuple, as shown in Fig. 10. It can be interpreted as specifying boundary conditions on the index space. Thus, for 2D-SDF, as shown in the figure, it specifies the number of initial rows and columns. It can also be interpreted as specifying the direction in the index space of a dependence between two single assignment variables, much as is done in reduced dependence graphs [18].

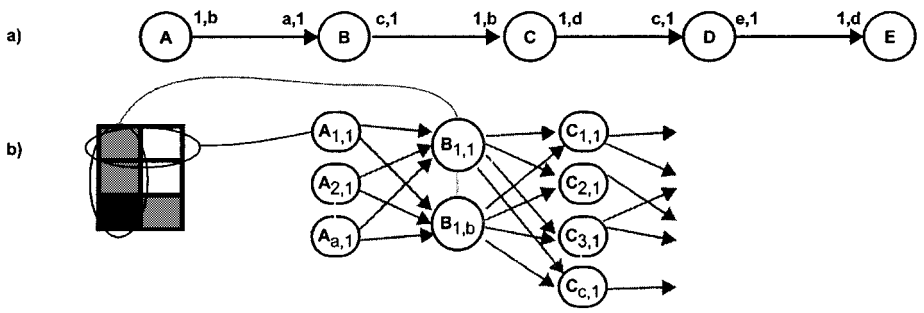

Fig. 9. (a) Multilayer perceptron expressed as an MDSDF graph. (b) Precedence graph.

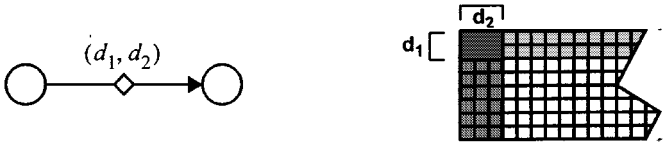

Fig. 10. Delay in MD-SDF is multidimensional.

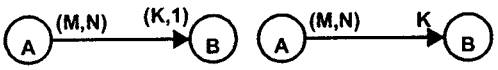

Fig. 11. Rule for augmenting the dimensionality of a producer or consumer.

D. Mixing Dimensionality

We can mix dimensionality. We use the following rule to avoid any ambiguity.

- The dimensionality of the index space for an arc is the maximum of the dimensionality of the producer and consumer. If the producer or the consumer specifies fewer dimensions than those of the arc, the specified dimensions are assumed to be the lower ones (lower number, earlier in the M-tuple), with the remaining dimensions assumed to be 1. Hence, the two graphs in Fig. 11 are equivalent.
- If the dimensionality specified for a delay is lower than the dimensionality of an arc, then the specified delay values correspond to the lower dimensions. The unspecified delay values are zero. Hence, the graphs in Fig. 12 are equivalent.

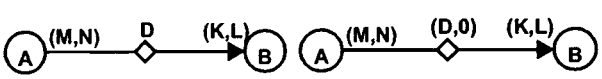

Fig. 12. Rule for augmenting the dimensionality of a delay.

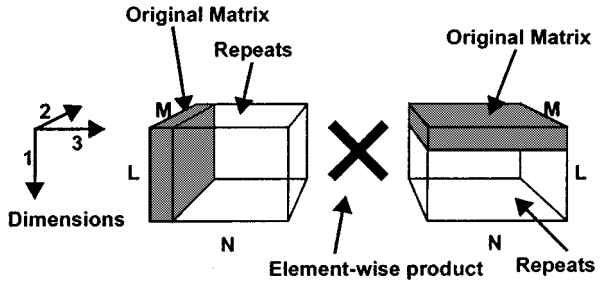

Fig. 13. Matrix multiplication represented schematically.

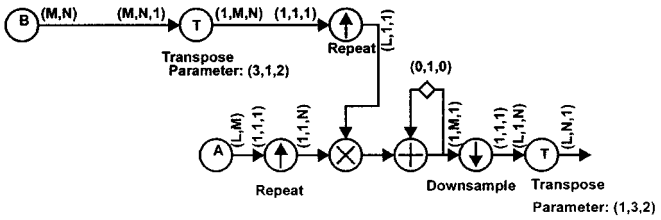

Fig. 14. Matrix multiplication in MDSDF.

E. Matrix Multiplication

As another example, consider a fine-grain specification of matrix multiplication. Suppose we are to multiply an $L \times M$ matrix by an $M \times N$ matrix. In a three-dimensional (3–D) index space, this can be accomplished as shown in Fig. 13. The original matrices are embedded in that index space, as shown by the shaded areas. The remainder of the index space is filled with repetitions of the matrices. These repetitions are analogous to assignments often needed in a single-assignment specification to carry a variable forward in the index space. An intelligent compiler need not actually copy the matrices to fill an area in memory. The data in the two cubes is then multiplied element-wise, and the resulting products are summed along dimension 2. The resulting sums give the $L \times N$ matrix product. The MDSDF graph implementing this is shown in Fig. 14. The key actors used for this are the following.

Repeat: In specified dimension(s), this consumes one and produces N, repeating values.

Down-sample: In specified dimension(s), this consumes N and produces 1, discarding samples.

Transpose: This consumes an M-dimensional block of samples and outputs them with the dimensions rearranged.

In addition, the following actor is also useful, although it is not used in the above example.

Upsample: In specified dimension(s), this consumes one and produces N, inserting zero values.

These are identified in Fig. 15. Note that all of these actors simply control the way tokens are exchanged and need not involve any run-time operations. Of course, a compiler then needs to understand the semantics of these operators.

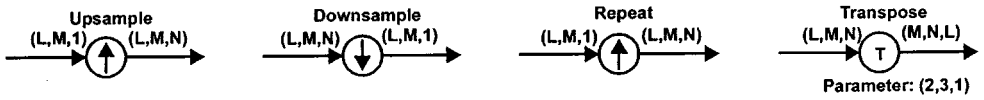

Fig. 15. Some key MDSDF actors that affect the flow of control.

F. Run-Time Implications

Several of the actors we have used perform no computation but instead control the way tokens are passed from one actor to another. In principle, a smart compiler can avoid run-time operations altogether, unless data movement is required to support parallel execution. We set the following objectives for a code generator using this language.

- Upsample: Zero-valued samples should not be produced, stored, or processed.
- Repeat: Repeated samples should not be produced or stored.
- Last-N: A circular buffer should be maintained and made directly available to downstream actors.
- Down-sample: Discarded samples should not be computed (similar to dead-code elimination in traditional compilers).
- Transpose: There should be no run-time operation at all: just compile-time bookkeeping.

It is too soon to tell how completely these objectives can be met.

G. State

For large-grain dataflow languages, it is desirable to permit actors to maintain state information. From the perspective of their dataflow model, an actor with state information simply has a self-loop with a delay. Consider the three actors with self loops shown in Fig. 16. Assume, as is common, that dimension 1 indexes the row in the index space and dimension 2 the column, as shown in Fig. 17(b). Then, each firing of actor A requires state information from the previous row of the index space for the state variable. Hence, each firing of A depends on the previous firing in the vertical direction, but there is no dependence in the horizontal direction. The first row in the state index space must be provided by the delay initial value specification. Actor B, by contrast, requires state information from the previous column in the index space. Hence, there is horizontal, but not vertical, dependence among firings. Actor C has both vertical and horizontal dependence, implying that both an initial row and an initial column must be specified. Note that this does imply that there is no parallelism since computations along a diagonal wavefront can still proceed in parallel. Moreover, this property is easy to detect automatically in a compiler. Indeed, all modern parallel scheduling methods based on projections of an index space [18] can be applied to programs defined using this model.

We can also show that these multidimensional delays do not cause any complications with deadlock or preservation of determinacy.

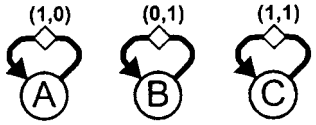

Fig. 16. Three macro actors with state represented as a self-loop.

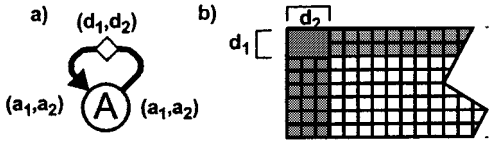

Fig. 17. (a) Actor with a self loop. (b) Data space on the arc.

Lemma 1: Suppose that an actor A has a self-loop as shown in Fig. 17(a). Actor deadlocks iff $a_1 > d_1$ and $a_2 > d_2$ both hold.

Proof: We use the notation $A_{[i,j]}$ to mean the (i,j)th invocation of actor A in a complete periodic schedule. If the inequalities both hold, then $A_{[0,0]}$ cannot fire since it requires a rectangle of data larger than that provided by the initial rows and columns intersected. The forward direction follows by looking at Fig. 17(b). If A deadlocks because $A_{[0,0]}$ cannot fire, then the inequalities must hold. If $A_{[0,0]}$ does fire, then it means that either $a_1 \leq d_1$ or $a_2 \leq d_2$. If $a_1 \leq d_1$, then clearly, $a_{[0,j]}$ can fire for any j since the initial rows provide the data for all these invocations. Then, $A_{[1,j]}$ can all fire since there are $a_1 + d_1$ rows of data now, and $2a_1 \leq a_1 + d_1$. Continuing this argument, we can see that can A fire as many times as it wants. The reasoning is that $a_2 \leq d_2$ is symmetric; in this case, $A_{[i,0]}$ can all fire, and then, $A_{[i,1]}$ can all fire, and so on. Therefore, actor A deadlocks iff $A_{[0,0]}$ is not firable, and $A_{[0,0]}$ is not firable iff the condition in the lemma holds. **Q.E.D.**

Corollary 1: In n dimensions, an actor A with a self-loop having $(d_1, \ldots, d_n)$ delays and producing and consuming hypercubes $(a_1, \ldots, a_n)$ deadlocks iff $a_1 > d_1 \forall i$.

Let us now consider the precedence constraints imposed by the self loop on the various invocations of A. Suppose that A fires (r_1, r_2) times. Then, the total array of data consumed is an array of size $(r_1 a_1, r_2 a_2)$. The same size array is written but shifted to the right and down of the origin by (d_1, d_2). In general, the rectangle of data read by a node is up and to the left of the rectangle of data written on this arc since we have assumed that the initial data is not being overwritten. Hence, an invocation $A_{[i,j]}$ can only depend on invocations $A_{[i^1,j^1]}$, where $i^1 \leq i, j^1 \leq j$. This motivates the following lemma.

Lemma 2: Suppose that actor A has a self loop as in the previous lemma, and suppose that A does not deadlock. Then, the looped schedule $(r_1, r_2)A$ is valid, and the order of nesting the loops does not matter. That is, the two programs that follow give the same result.

Proof: We have to show that the ordering of the $A_{[x,y]}$ in the loop is a valid linearization of the partial order given by the precedence constraints of the self loop. Suppose that in the first loop

```
for x = 0 : r1−1
  for y = 0 : r2 − 1
    fire A[x,y]
    end fory, forx
for y = 0 : r2 − 1
  for x = 0 : r2−1
    fire A[x,y]
    end forx, fory
```

the ordering is not a valid linearization. This means that there are indices (i_1, j_1) and (i_2, j_2) such that $A_{[i_2,j_2]}$ precedes $A_{[i_1,j_1]}$ in the partial order, but $A_{[i_1,j_1]}$ is executed before $A_{[i_2,j_2]}$ in the loop. Then, by the order of the loop indices, it must be that $i_1 \leq i_2$, but then, $A_{[i_2,j_2]}$ cannot precede $A_{[i_1,j_1]}$ in the partial order since this violates the right and down precedence ordering. The other loop is also valid by a symmetric argument. **Q.E.D.**.

The above result shows that the nesting order, which is an implementation detail not specified by the model itself, has no bearing on the correctness of the computation; this is important for preserving determinacy.

III. Modeling Arbitrary Sampling Lattices

The multidimensional dataflow model presented in the above section has been shown to be useful in a number of contexts, including expressing multidimensional signal processing programs and specifying flexible data-exchange mechanisms and scalable descriptions of computational modules. Perhaps the most compelling of these uses is the first one: for specifying multidimensional, multirate signal processing systems. This is because such systems, when specified in MDSDF, have the same intuitive semantics that 1-D systems have when expressed in SDF. However, the MDSDF model described so far is limited to modeling multidimensional systems sampled on the standard rectangular lattice. Since many multidimensional signals of practical interest are sampled on nonrectangular lattices [22], [32], for example, 2:1 interlaced video signals [13], and many multidimensional multirate systems use nonrectangular multirate operators like hexagonal decimators (see [1], [6], and [21], for example), it is of interest to have an extension of the MDSDF model that allows signals on arbitrary sampling lattices to be represented and that allows the use of nonrectangular downsamplers and upsamplers. The extended model we present here preserves compile-time schedulability.

A. Notation and Basics

The notation is taken from [33]. Consider the sequence of samples generated by $x(n_1, n_2) = x_a(a_{11}n_1 + a_{12}n_2, a_{21}n_1 + a_{22}n_2)$, where $x_a(t_1, t_2)$ is a continuous time signal. Notice that the sample locations retained are given by the equation

$$\hat{t} = \begin{bmatrix} t_1 \\ t_2 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} n_1 \\ n_2 \end{bmatrix} = V\hat{n}.$$

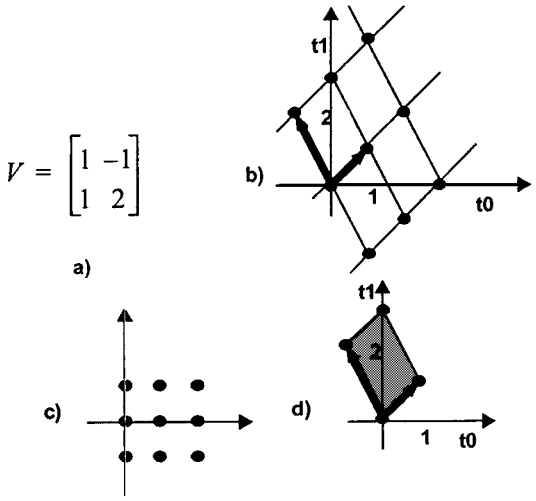

Fig. 18. Sampling on a nonrectangular lattice. (a) Sampling matrix V. (b) Samples on the lattice. (c) Renumbered samples of the lattice. (d) Fundamental parallelepiped for a matrix V.

The matrix V is called the *sampling matrix* (must be real and nonsingular). The sample locations are vectors $\hat{t}$ that are linear combinations of the columns of the sampling matrix V. Fig. 18(a) and (b) shows an example. The set of all sample points $\hat{t} = V\hat{n}, \hat{n} \in \aleph$ is called the *lattice* generated by V and is denoted $FPD(V)$. The matrix V is the *basis* that generates the lattice $LAT(V)$. Suppose that $\hat{n}$ is a point on $LAT(V)$. Then, there exists an integer vector $\hat{k}$ such that $\hat{n} = V\hat{K}$. The points $\hat{k}$ are called the *renumbered points* of $LAT(V)$. Fig. 18(c) shows the renumbered samples for the samples on $LAT(V)$ shown in Fig. 18(b) for the sampling matrix shown in Fig. 18(a).

The set of points $V\hat{x}$, where $\hat{x} = [x_1, \quad x_2]^T$, with $0 \leq x_1, x_2 < 1$, is called the *fundamental parallelepiped* of V and is denoted $LAT(V)$, as shown in Fig. 18(d) for the sampling matrix from Fig. 18(a). From geometry, it is well known that the volume of $FPD(V)$ is given by $|\det(V)|$. Since only one renumbered integer sample point falls inside $LAT(V)$, namely, the origin, the sampling density is given by the inverse of the volume of $FPD(V)$.

Definition 1: Denote the set of integer points within $FPD(V)$ as the set $N(V)$. That is, $N(V)$ is the set of integer vectors of the form $V\hat{x}, \hat{x} \in [0, 1)^m$.

The following well-known lemma (see [23] for a proof) characterizes the number of integer points that fall inside $FPD(V)$ or the size of the set $N(V)$.

Lemma 3: Let V be an integer matrix. The number of elements in $N(V)$ is given by $|N(V)| = |\det(V)|$.

1) Multidimensional Decimators: The two basic multirate operators for multidimensional systems are the decimator and expander. A decimator is a single-input-single-output (SISO) function that transmits only one sample for every n samples in the input; n is called the *decimation ratio*. For an MD signal $x(\hat{n})$ on $LAT(V_1)$, the M-fold decimated version is given by $y(\hat{n}) = x(\hat{n}), \hat{n} \in LAT(V_I M)$, where M is an $m \times m$ nonsingular integer matrix called the *decimation matrix*. Fig. 19 shows two examples of decimation. The example on the left is for a diagonal matrix M; this is called *rectangular decimation* because $FPD(M)$ is a rectangle rather than a

Fig. 19. (a) Rectangular decimation. (b) Hexagonal decimation.

Fig. 20. (a) Rectangular expansion. (b) Nonrectangular expansion.

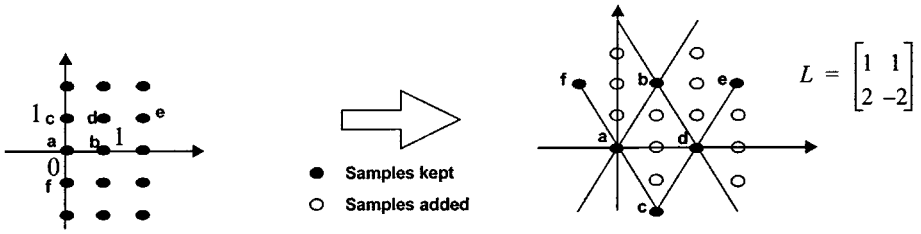

Fig. 21. Renumbered samples from the expanders output.

parallelepiped. In general, a rectangular decimator is one for which the decimation matrix is diagonal. The example on the right is for a nondiagonal M and is loosely termed *hexagonal decimation*. Note that $LAT(V_I) \supseteq LAT(V_I M)$.

The decimation ratio for a decimator with decimation matrix M is given by $|N(M)| = |det(M)|$. The decimation ratio for the example on the left in Fig. 19 is 6, and it is 4 for the example on the right.

2) Multidimensional Expanders: In the multidimensional case, the "expanded" output $y(\hat{n})$ of an input signal $x(n)$ is given by

$$y(n) = \begin{pmatrix} x(n) & n \in LAT(V_I) \\ 0 & \text{otherwise} \end{pmatrix} \forall n \in LAT(V_I L^{-1})$$

where V_1 is the input lattice to the expander. Note that $LAT(V_I) \subseteq LAT(V_I L^{-1})$. The expansion ratio, which is defined as the number of points added to the output lattice for each point in the input lattice, is given by $|\det(L)|$. Fig. 20 shows two examples of expansion. In the example on the left, the output lattice is also rectangular and is generated by diag$(0.5,\ 0.5)$[1] . The example on the right shows nonrectangular expansion, where the lattice is generated by

$$L^{-1} = \begin{bmatrix} 0.5 & 0.25 \\ 0.5 & -0.25 \end{bmatrix}.$$

An equivalent way to view Fig. 20 is to plot the renumbered samples. Notice that the samples from the input will now lie on $LAT(L)$ (see Fig. 21). Some of the points have been labeled with letters to show where they would map to on the output signal.

[1]We use the notation diag$(a_1, \ldots, a_n)$ to denote a diagonal $n \times n$ matrix with the a_I on the diagonal.

B. Semantics of the Generalized Model

Consider the system depicted in Fig. 22, where a source actor produces an array of 6×6 samples each time it fires [(6,6) in MDSDF parlance]. This actor is connected to the decimator with a nondiagonal decimation matrix. The circled samples indicate the samples that fall on the decimators output lattice; these are retained by the decimator. In order to represent these samples on the decimator's output, we will think of the buffers on the arcs as containing the renumbered equivalent of the samples on a lattice. For a decimator, if we renumber the samples at the output according to $LAT(V_1 M)$, then the samples get written to a parallelogram-shaped array rather than a rectangular array. To see what this parallelogram is, we introduce the concept of a "support matrix" that describes precisely the region of the rectangular lattice where samples have been produced. Fig. 22 illustrates this for a decimation matrix, where the retained samples have been renumbered according to $LAT(M)$ and plotted on the right. The labels on the samples show the mapping. The renumbered samples can be viewed as the set of integer points lying inside the parallelogram that is shown in the figure. In other words, the *support* of the renumbered samples can be described as $FPD(Q)$, where $Q = \begin{bmatrix} 3 & 1.5 \\ 3 & -1.5 \end{bmatrix}$.

We will call Q the *support matrix* for the samples on the output arc. In the same way, we can describe the support of the samples on the input arc to the decimator as $FPD(P)$, where $P = \text{diag}(6, 6)$. It turns out that $Q = M^{-1}P$.

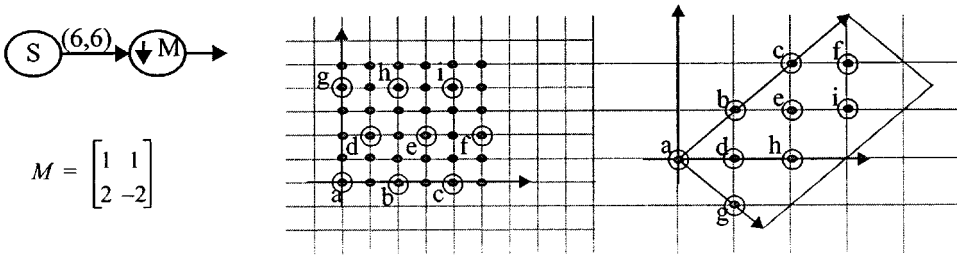

Fig. 22. Output samples from the decimator renumbered to illustrate concept of support matrix.

Definition 2: Let X be a set of integer points in $\Re^m$. We say that X satisfies the *containability condition* if there exists $m \times m$ an rational-valued matrix W such that $N(W) = X$. In other words, that there is a fundamental parallelepiped whose set of integer points equals X.

Definition 3: Given a sampling matrix V_S, a set of samples ζ is called a *production set on* V_S if each sample in ζ lies on the lattice $LAT(V_S)$ and the set $\overline{\zeta} = \left\{V_S^{-1}\hat{n} : \hat{n} \in \zeta\right\}$, which is the set of integer points consisting of the points of ζ renumbered by $LAT(V_S)$, satisfies the containability condition.

We will assume that any source actor in the system produces data according to the *source data production method*, where a source S outputs a production set on V_S, which is the sampling matrix on the output of S.

Given a decimator with decimation matrix M, as shown in Fig. 23(a), we make the following definitions and statements. Denoting the input arc to the decimator as e and the output arc as f, V_e and V_f are the bases for the input and output lattice, respectively. W_e and W_f are the support matrices for the input and output arcs, respectively, in the sense that samples, numbered according to the respective lattices, are the integer points of fundamental parallelepipeds of the respective support matrices. Similarly, we can also define these quantities for the expander L depicted in Fig. 23(b). With this notation, we can state the following.

Theorem 1: The relationships between the input and output lattices, and the input and output support matrices for the decimator and expander depicted in Fig. 23, are

$$\begin{aligned} \textbf{Decimator} \quad & V_f = V_e M, \quad W_f = M^{-1} W_e. \\ \textbf{Expander} \quad & V_f = V_e L^{-1}, \quad W_f = L W_e. \end{aligned}$$

Proof: The relationships between the input and output lattices follow from the definition of the expander and decimator. Consider a point n on the decimator's input lattice. There exists an integer vector k such that $n = V_e k$. If $M^{-1}k$ is an integer vector, then this point will be kept by the decimator since it will fall on the output lattice, i.e, $n = V_e M k'$, where $k' = M^{-1}k$. This point n is renumbered as $k' = M^{-1}V_e^{-1}n = M^{-1}k$ by the output lattice. Since k was the renumbered point corresponding to n on the input lattice and, hence, in $N(W_e)$, every point k in $N(W_e)$ that is kept by the decimator is mapped to $M^{-1}k$ by the output lattice. Now, $k \in N(W_e) \Rightarrow \exists z \in [0,1)^2 \text{s.t. } k = W_e z$. Therefore, $M^{-1}k \in N(M^{-1}W_e)$ because $M^{-1}k = M^{-1}W_e z$. Conversely, let j be any point in $N(M^{-1}W_e)$. Then, $\exists z \in [0,1)^2 \text{s.t. } j = M^{-1}W_e z$. Since $W_e z = Mj$, we have that $Mj \in N(W_e)$. In addition, the corresponding point to this on the input lattice is $V_e M j$, implying that the point is retained by the decimator. Hence, $W_f = M^{-1}W_e$. The derivation for the expander is identical, only with different expressions. **Q.E.D.**

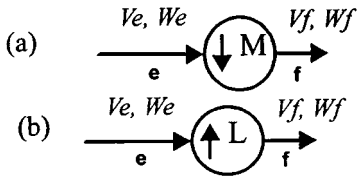

Fig. 23. (a) Generalized decimator and (b) expander with arbitrary input lattices and support matrices.

Corollary 2: In an acyclic network of actors, where the only actors that are allowed to change the sampling lattice are the decimator and expander in the manner given by Theorem 1, and where all source actors produce data according to the source data production method of Section III-B, the set of samples on every arc, renumbered according to the sampling lattice on that arc, satisfies the containability condition.

Proof: The proof is immediate from the theorem.

In the following, we develop the semantics of a model that can express these nonrectangular systems by going through a detailed example. In general, our model for the production and consumption of tokens will be the following: An expander produces $FPD(L)$ samples on each firing where L is the upsampling matrix. The decimator consumes a "rectangle" of samples, where the "rectangle" has to be suitably defined by looking at the actor that produces the tokens that the decimator consumes.

Definition 4: An *integer* (a, b) *rectangle* is defined to be the set of integer points in $[0, a) \times [0, b)$, where a and b are arbitrary real numbers.

Definition 5: Let X be a set of points in $\Re^2$, and let x and y be two positive integers such that $xy = |X|$. X is said to be organized as a *generalized* (x, y) *rectangle* of points, or just a generalized (x, y) rectangle, by associating a *rectangularizing* function with X that maps the points of X to an integer (x, y) rectangle.

Example 1: Consider the system where a decimator follows an expander [see Fig. 24(a)]

We start by specifying the lattice and support matrix for the arc *SA*. Let $V_{SA} = \text{diag}(1, 1)$ and $W_{SA} = \text{diag}(3, 3)$. Therefore, the source produces (3,3) in MDSDF parlance since the lattice on *SA* is the normal rectangular lattice, and the support matrix represents an FPD that is a 3×3 rectangle. For the system above, we can compute the lattice and support matrices for all

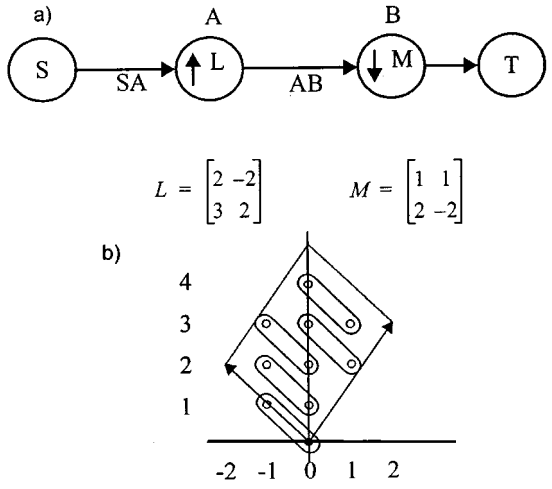

Fig. 24. Example to illustrate balance equations and the need for some additional constraints. (a) System. (b) Ordering of data into a 5×2 rectangle inside FPD(L).

other arcs given these. We will need to specify the scanning order for each arc as well that which tells the node the order in which samples should be consumed. Assume for the moment that the expander will consume the samples on arc SA in some natural order, for example, scanning by rows. We need to specify what the expander produces on each firing. The natural way to specify this is that the expander produces $FPD(L)$ samples on each firing; these samples are organized as a generalized (L_1, L_2) rectangle. This allows us to say that the expander produces (L_1, L_2) samples per firing; this is understood to be the set $FPD(L)$ of points organized as a generalized (L_1, L_2) rectangle. Note that in the rectangular MDSDF case, we could define upsamplers that did their upsampling on only a subset of the input dimensions (see Section II-E). This is possible since the rectangular lattice is separable; for nonrectangular lattices, it is not possible to think of upsampling (or downsampling) occurring along only some dimensions. We have to see upsampling and downsampling as lattice transforming operations and deal with the appropriate matrices.

Suppose we choose the factorization 5×2 for $|\det(L)|$. Consider Fig. 24(b), where the samples in $FPD(L)$ are shown. One way to map the samples into an integer (5, 2) rectangle is as shown by the groupings. Notice that the horizontal direction for $FPD(L)$ is the direction of the vector $[2 \quad 3]^T$, and the vertical direction is the direction of the vector $[-2 \quad 2]^T$. We need to number the samples in $FPD(L)$; the numbering is needed in order to establish some common reference point for referring to these samples since the downstream actor may consume only some subset of these samples. One way to number the samples is to number them as sample points in a 5×2 rectangle, as shown in Table I.

Hence, $FPD(L)$ is a generalized $(5,2)$ rectangle if we associate the function given in Table I with it as the rectangularizing function. Given a factoring of the determinant of L, the function given previously can be computed easily, for example, by ordering the samples according to their Euclidean distance from the two vectors that correspond to the horizontal and vertical directions (we should be convinced that given a factorization $n \times m$ for $|\det(L)|$, clearly, there are many

TABLE I
ORDERING THE SAMPLES PRODUCED BY THE EXPANDER

Original sample	(0,0)	(0,1)	(0,2)	(1,2)	(1,3)	(-1,1)	(-1,2)	(-1,3)	(0,3)	(0,4)
Renumbered sample	(0,0)	(1,0)	(2,0)	(3,0)	(4,0)	(0,1)	(1,1)	(2,1)	(3,1)	(4,1)

functions that map the nm points in $FPD(L)$ to the set $\{(0,0), \ldots, (0, m-1), \ldots, (n-1, 0), \ldots, (n-1, m-1)\}$; any such function would suffice). The scanning order for the expander across invocations is determined by the numbering of the input sample on the output lattice. For example, the sample at (1,0) that the source produces maps to location (2,3) on the lattice at the expanders output ($L[1 \quad 0]^T$). Hence, consuming samples in the $[1 \quad 0]$ direction on arc SA results in 5×2 samples (i.e, $FPD(L)$ samples but ordered according to the table) being produced along the vector $[2 \quad 3]$ on the output. Similarly, the sample (0,1) produced by the source corresponds to $(-2,2)$ on the output lattice. A global ordering on the samples is imposed by the following renumbering. The sample at (2,3) lies on the lattice generated by L and is generated by the vector $[1 \quad 0]^T$. Hence, (1,0) is the renumbered point corresponding to (2,3), but because there are more points in the output than simply the points on $LAT(L)$, clearly, (1,0) cannot be the renumbered point. In fact, since we organized $FPD(L)$ as a generalized (5,2) rectangle and renumbered the points inside the *FPD* as in the table, the actual renumbered point corresponding to (2,3) is given by $(1 * 5, 0 * 2) = (5, 0)$. Similarly, the lattice point (0,5) is generated by (1,1), meaning that it should be renumbered as $(1*5, 1*2) = (5, 2)$. With this global ordering, it becomes clear what the semantics for the decimator should be. Again, choose a factorization of $|\det(M)|$, and consume a "rectangle" of those samples, where the "rectangle" is deduced from the global ordering imposed previously. For example, if we choose 2×2 as the factorization, then the (0,0) invocation of the decimator consumes the (original) samples at (0,0), $(-1,1)$, (0,1), and $(-1,2)$. The (0,2)th invocation of the decimator would consume the (original) samples at (1,3), (0,4), (2,3), and (1,4). The decimator would have to determine which of these samples falls on its lattice; this can be done easily. Note that the global ordering of the data is not a restriction in any way since this ordering is determined by the scheduler and can be determined on the basis of implementation efficiency if required. The designer does not have to worry about this behind-the-scenes determination of ordering.

We have already mentioned the manner in which the source produces data. We add that the subsequent firings of the source are always along the directions established by the vectors in the support matrix on the output arc of the source.

Now, we can write a set of "balance" equations using the "rectangles" that we have defined. Denote the repetitions of a node X in the "horizontal" direction by $r_{X,1}$ and the "vertical" direction as $r_{X,2}$. These directions are dependent on the geometries that have been defined on the various arcs. Thus, for example, the directions are different on the input arc to the expander from the directions on the output arc. We have

$$3r_{S,1} = 1r_{A,1} \quad 5r_{A,1} = 2r_{B,1} \quad r_{B,1} = r_{T,1}$$
$$3r_{S,2} = 1r_{A,2} \quad 2r_{A,2} = 2r_{B,2} \quad r_{B,2} = r_{T,2}$$

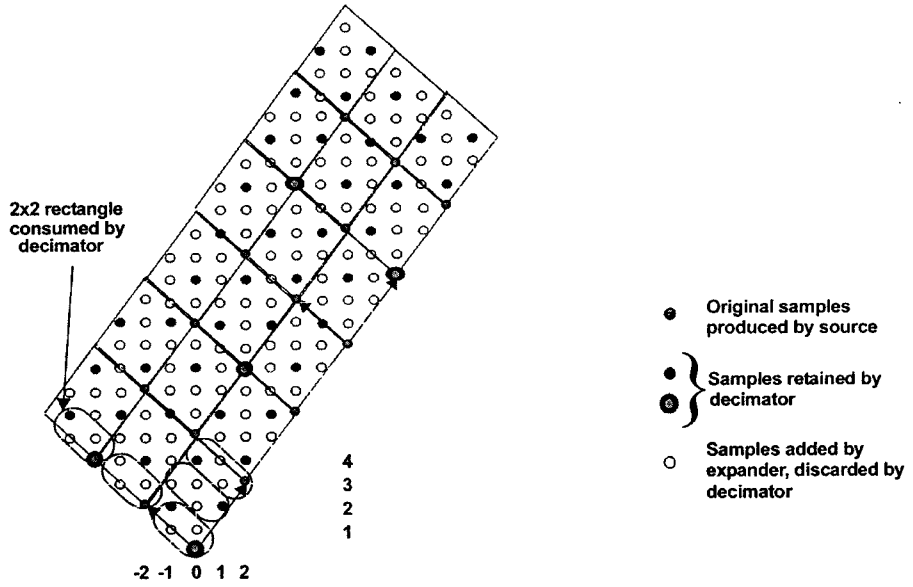

Fig. 25. Total amount of data produced by the source in one iteration of the periodic schedule determined by the balance equations in (1).

where we have assumed that the sink actor T consumes (1,1) for simplicity. We have also made the assumption that the decimator produces exactly (1,1) every time it fires. This assumption is usually invalid, but the calculations done in the following are still valid, as will be discussed later. Since these equations fall under the same class as SDF balance equations described in Section I-A, the properties about the existence of the smallest unique solution applies here as well. These equations can be solved to yield the following smallest, unique solution:

$$\begin{aligned} r_{S,1} &= 2 \quad r_{A,1} = 6 \quad r_{B,1} = 15 \quad r_{T,1} = 15 \\ r_{S,2} &= 1 \quad r_{A,2} = 3 \quad r_{B,2} = 3 \quad r_{T,2} = 3. \end{aligned} \tag{1}$$

Fig. 25 shows the data space on arc AB with this solution to the balance equations. As we can see, the assumption that the decimator produces (1,1) on each invocation is not valid; sometimes, it produces no samples at all and sometimes two samples or one sample. Hence, we have to see if the total number of samples retained by the decimator is equal to the total number of samples it consumes divided by the decimation ratio.

In order to compute the number of samples output by the decimator, we have to compute the support matrices for the various arcs assuming that the source is invoked (2,1) times (so that we have the total number of samples being exchanged in one schedule period). We can do this symbolically using $r_{S,1}$ and $r_{S,2}$ and substitute the values later. We get

$$\begin{aligned} W_{SA} &= \begin{bmatrix} 3 & 0 \\ 0 & 3 \end{bmatrix} \begin{bmatrix} r_{S,1} & 0 \\ 0 & r_{S,2} \end{bmatrix} = \begin{bmatrix} 3r_{S,1} & 0 \\ 0 & 3r_{S,2} \end{bmatrix} \\ W_{AB} &= LW_{SA} = \begin{bmatrix} 6r_{S,1} & -6r_{S,2} \\ 9r_{S,1} & 6r_{S,2} \end{bmatrix}, \text{and} \\ W_{BT} &= M^{-1}W_{AB} = \frac{1}{4}\begin{bmatrix} 21r_{S,1} & -6r_{S,2} \\ 3r_{S,1} & -18r_{S,2} \end{bmatrix}. \end{aligned} \tag{2}$$

Recall that the samples that the decimator produces are the integer points in $FPD(W_{BT})$. Hence, we want to know if

$$|N(W_{BT})| = \frac{|N(W_{AB})|}{|M|} \tag{3}$$

is satisfied by our solution to the balance equations. By Lemma 3, the size of the set $N(A)$ for an integer matrix A is given by $|\det(A)|$. Since W_{AB} is an integer matrix for any value of $r_{S,1}$, $r_{S,2}$, we have $|N(W_{AB})| = |\det(W_{AB})| = 90r_{S,1}r_{S,2}$. The right-hand side of (3) becomes $(90r_{S,1}r_{S,2})/4 = (45r_{S,1}r_{S,2})/2$. Hence, our first requirement is that $r_{S,1}r_{S,2} = 2k \quad k = 0, 1, 2 \ldots$. The balance equations gave us $r_{S,1} = 2, r_{S,2} = 1$; this satisfies the requirement. With these values, we get

$$W_{BT} = \begin{bmatrix} \frac{21}{2} & \frac{-3}{2} \\ \frac{3}{2} & \frac{-9}{2} \end{bmatrix}.$$

Since this matrix is not integer-valued, Lemma 3 cannot be invoked to calculate the number of integer points in $FPD(W_{BT})$. For noninteger matrices, there does not seem to be a polynomial-time method of computing $|N(W_{BT})|$, although a method that is much better than the brute force approach is given in [23]. Using that method, it can be determined that there are 47 points inside $FPD(W_{BT})$. Hence, (3) is not satisfied. One way to satisfy (3) is to force W_{BT} to be an integer matrix. This implies that $r_{S,1} = 4k,\ k = 0, 1, 2 \ldots$ and $r_{S,2} = 2k,\ k = 0, 1, 2 \ldots$. The smallest values that make W_{BT} integer valued are $r_{S,1} = 4,\ r_{S,2} = 2$. From this, the repetitions of the other nodes are also multiplied by 2. Note that the solution to the balance equations by themselves are not "wrong"; it is just that for nonrectangular systems, (3) gives a new constraint that must also be satisfied. We address concerns about efficiency that the increase in repetitions entails in Section III-B1. We can formalize the ideas developed in the previous example in the following.

Lemma 4: The support matrices in the network can each be written as functions of the repetitions variables of one particular source actor in the network.

Proof: The proof is immediate from the fact that all of the repetitions variables are related to each other via the balance equations.

Lemma 5: In a multidimensional system, the jth column of the support matrix on any arc can be expressed as a matrix that has entries of the form $a_{ij}r_{S,j}$, where $r_{S,j}$ is the repetitions variable in the jth dimension of some particular source actor S in the network, and a_{ij} are rationals.

Proof: Without loss of generality, assume that there are two dimensions. Let the support matrix on the output arc of source S for one firing be given by

$$W_S = \begin{bmatrix} p & q \\ r & s \end{bmatrix}.$$

For $r_{S,1}, r_{S,2}$ firings in the "horizontal" and "vertical" directions (these are the directions of the columns of W_S), the support matrix becomes

$$W_S = \begin{bmatrix} p & q \\ r & s \end{bmatrix} \begin{bmatrix} r_{S,1} & 0 \\ o & r_{S,2} \end{bmatrix} = \begin{bmatrix} pr_{S,1} & qr_{S,2} \\ rr_{S,1} & sr_{S,2} \end{bmatrix}$$

(in multiple dimensions, the right multiplicand would be a diagonal matrix with $r_{S,j}$ in the jth row).

Now, consider an arbitrary arc (u, v) in the graph. Since the graph is connected, there is at least one undirected path P from source S to node u. Since the only actors that change the sampling lattice (and, thus, the support matrix) are the decimator and expander, all of the transformations that occur to the support matrix W_S along P are left multiplications by some rational-valued matrix. Hence, the support matrix on arc e (W_e) can be expressed as $W_e = AW_S$, where A is some rational valued matrix. The claim of the lemma follows from this.

Q.E.D.

Theorem 2: In an acyclic network of actors, where the only actors that are allowed to change the sampling lattice are the decimator and expander in the manner given by Theorem 1, and where all source actors produce data according to the source data production method of Section III-B, whenever the balance equations for the network have a solution, there exists a blocking factor vector J such that increasing the repetitions of each node in each dimension by the corresponding factor in J will result in the support matrices being integer valued for all arcs in the network.

Proof: By Lemma 5, a term in an entry in the jth column of the support matrix on any arc is always a product of a rational number and repetitions variable $r_{S,j}$ of source S. We force this term to be integer valued by dictating that each repetition's variable $r_{S,j}$ be the lcm of the values needed to force each entry in the jth column to be an integer. Such a value can be computed for each support matrix in the network. The lcm of all these values and the balance equations solution for the source would then give a repetition's vector for the source that makes all of the support matrices in the network integer valued and solves the balance equations. **Q.E.D.**

It can be easily shown that the constraint of the type in (3) is always satisfied by the solution to the balance equations when all of the lattices and matrices are diagonal [23].

The fact that the decimator produces a varying number of samples per invocation might suggest that it falls nicely into the class of cyclostatic actors. However, there are a couple of differences. In the CSDF model of [5], the number of cyclostatic phases are assumed to be known beforehand and is only a function of the parameters of the actor, like the decimation factor. In our model for the decimator, the number of phases is not just a function of the decimation matrix; it is also a function of the sampling lattice on the input to the decimator (which, in turn, depends on the actor that is feeding the arc) and the factorization choice that is made by the scheduler. Second, in CSDF, SDF actors are represented as cyclostatic by decomposing their input/output behavior over one invocation. For example, a CSDF decimator behaves exactly like the SDF decimator, except that the CSDF decimator does not need all M data inputs to be present before it fires; instead, it has a four-phase firing pattern. In each phase, it will consume one token but will produce one token only in the first phase and produce 0 tokens in the other phases. In our case, the cyclostatic behavior of the decimator is arising across invocations rather than within an invocation. It is as if the CSDF decimator with decimation factor 4 were to consume {4,4,4,4,4,4} and produce {2,0,1,1,0,2} instead of consuming {1,1,1,1} and producing {1,0,0,0}.

One way to avoid dealing with constraints of the type in (3) would be to choose a factorization of $|\det(M)|$ that ensured that the decimator produced one sample on each invocation. For example, if we were to choose the factorization 1×4 for the previous example, the solution to the balance equations would automatically satisfy (3). As we show later, we can find factorizations where the decimator produces one sample on every invocation in certain situations, but generalizing this result appears to be a difficult problem since there does not seem to be an analytical way of writing down the renumbering transformation that was shown in Table I.

1) Implications of the Previous Example for Streams: In SDF, there is only one dimension, and the stream is in that direction. Hence, whenever the number of repetitions of a node is greater than unity, the data processed by that node then corresponds to data along the stream. In MDSDF, only one of the directions is the stream. Hence, if the number of repetitions of a node, especially a source node, is greater than unity for the nonstream directions, the physical meaning of invocations in those directions becomes unclear. For example, consider a 3-D MDSDF model for representing a progressively scanned video system. Of these three dimensions, two of them correspond to the height and width of the image, and the third dimension is time. Hence, a source actor that produces the video signal might produce something like (512,512,1), meaning 1 512×512 images per invocation. If the balance equations dictated that this source should fire (2,2,3) times, for example, then it is not clear what the two repetitions each in the height and width directions signify since they certainly do not result in data from the next iteration being processed, where an iteration corresponds to the processing of an image at the next sampling instant. Only the repetitions of three along the time dimension

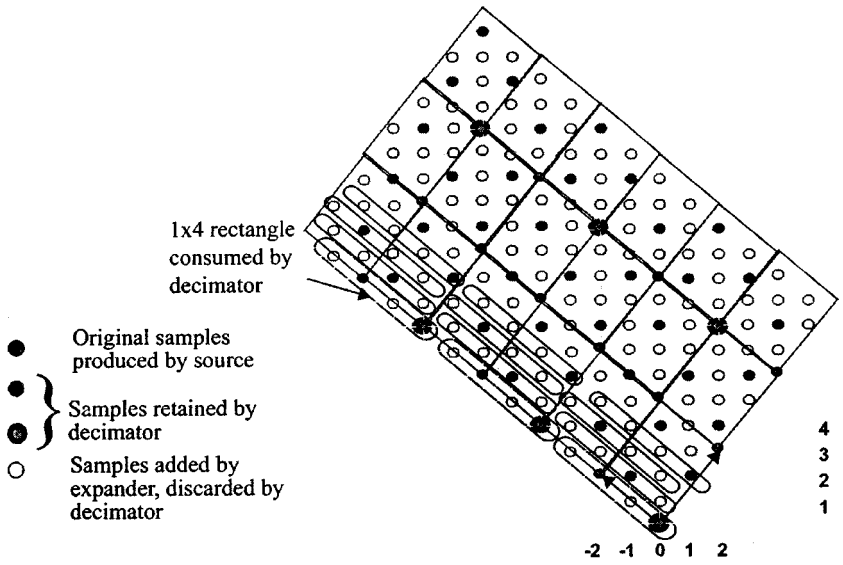

Fig. 26. Total amount of data produced by the source in one iteration of the periodic schedule determined by the balance equations in (4). The samples kept by the decimator are the lightly shaded samples.

makes physical sense. Hence, there is potentially room for great inefficiency if the user of the system has not made sure that the rates in the graph match up appropriately so that we do not actually end up generating images of size 1024×1024 when the actual image size is 512×512. In rectangular MDSDF, it might be reasonable to assume that the user is capable of setting the MDSDF parameters such that they do not result in absurd repetitions being generated in the nonstream directions since this can usually be done by inspection. However, for nonrectangular systems, we would like to have more formal techniques for keeping the repetition's matrix in check since it is much less obvious how to do this by inspection. The number of variables are also greater for nonrectangular systems since different factorizations for the decimation or expansion matrices give different solutions for the balance equations.

To explore the different factoring choices, suppose we use 1×4 for the decimator instead of 2×2. The solution to the balance equations become

$$\begin{aligned} r_{S,1} &= 1 \quad r_{A,1} = 3 \quad r_{B,1} = 15 \quad r_{T,1} = 15 \\ r_{S,2} &= 2 \quad r_{A,2} = 6 \quad r_{B,2} = 3 \quad r_{T,2} = 3. \end{aligned} \tag{4}$$

From (2), W_{BT} is given by

$$W_{BT} = \begin{bmatrix} \frac{21}{4} & -3 \\ \frac{3}{4} & -9 \end{bmatrix}$$

and it can be determined that $|N(W_{BT})| = 45$, as required. Therefore, in this case, we do not need to increase the blocking factor to make W_{BT} an integer matrix, and this is because the decimator is producing one token on every firing, as shown in Fig. 26.

However, if the stream in the above direction were in the horizontal direction (from the point of view of the source), then the solution given by the balance (4) may not be satisfactory, for reasons already mentioned. For example, the source may be forced to produce only zeros for invocation (0,1). One way to incorporate such constraints into the balance equation's computation is to specify the repetition's vector instead of the number produced or consumed. That is, for the source, we specify that $r_{S,2} = 1$ leave the number it produces in the vertical direction unspecified (this is the strategy used in programming the Philips VSP, for example). The balance equations will give us a set of acceptable solutions involving the number produced vertically; we can then pick the smallest such number that is greater than or equal to three. Denoting the number produced vertically by y_s, our balance equations become

$$\begin{aligned} 3r_{S,1} &= 1r_{A,1} \quad 5r_{A,1} = 1r_{B,1} \quad r_{B,1} = r_{T,1} \\ y_S 1 &= 1r_{A,2} \quad 2r_{A,2} = 4r_{B,2} \quad r_{B,2} = r_{r,2}. \end{aligned} \tag{5}$$

The solution to this is given by

$$\begin{aligned} r_{S,1} &= 1 \quad r_{A,1} = 3 \quad r_{B,1} = 15 \quad r_{T,1} = 15 \\ y_S &= 2k \quad r_{A,2} = 2k \quad r_{B,2} = k \quad r_{r,2} = 3 \\ k &= 1, 2, \ldots \end{aligned}$$

and we see that $k = 2$ satisfies our constraint. Recalculating the other quantities

$$W_{BT} = M^{-1} W_{AB} = \frac{1}{4} \begin{bmatrix} 21 r_{S,1} & -8 r_{S,2} \\ 3 r_{S,1} & -24 r_{S,2} \end{bmatrix} = \begin{bmatrix} \frac{21}{4} & -2 \\ \frac{3}{4} & -6 \end{bmatrix}$$

and we can determine that $|N(W_{BT})| = 30$ as required (i.e., $3 \times 4 \times 10/4 = 30$). Hence, we get away with having to produce only one extra row rather than three, assuming that the source can only produce three meaningful rows of data (and any number of columns).

2) Eliminating Cyclostatic Behavior: The fact that the decimator does not behave in a cyclostatic manner in Fig. 26 raises the question of whether factorizations that result in noncyclostatic behavior in the decimator can always be found. The following example and lemma give an answer to this question for the special case of a decimator whose input is a rectangular lattice.

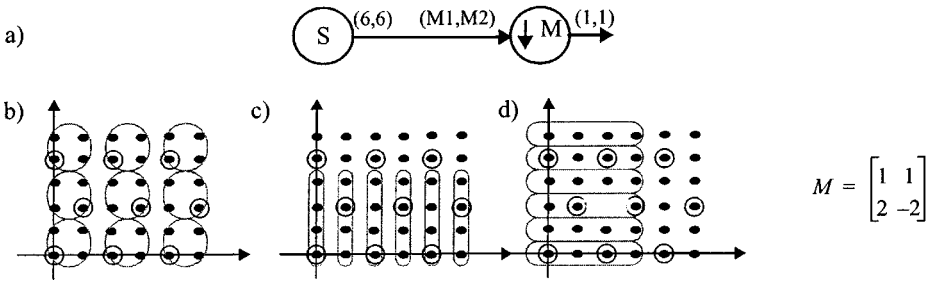

Fig. 27. Example to illustrate that two factorizations always exist that result in noncyclostatic behavior with the decimator. (a) System. (b) $M1 = 2, M2 = 2$. (c) $M1 = 1, M2 = 4$. (d) $M1 = 4, M2 = 1$.

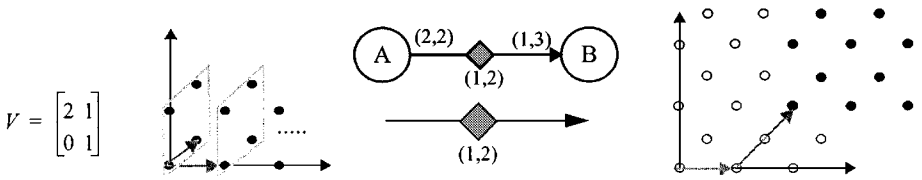

Fig. 28. Delays on nonrectangular lattices.

Example 2: Consider the system in Fig. 27(a), where a 2-D decimator is connected to a source actor that produces an array of (6,6) samples on each firing. The black dots represent the samples produced by the source, and the circled black dots show the samples that the decimator should retain. Since $|det(M)| = 4$, there are three possible ways to choose M_1, M_2. For two of the factorizations, the decimator behaves statically, that is, it produces one sample on each firing [see Fig. 27(b) and (c). However, in Fig. 27(d), we see that on some invocations, no samples are produced [that is, (0,0) samples are produced], while in some invocations, two samples are produced. This raises the question of whether there is always a factorization that ensures that the decimator produces (1,1) for all invocations. The following lemma ensures that for any matrix, there are always two factorizations of the determinant such that the decimator produces (1,1) for all invocations.

Lemma 6 [23]: If $M = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$ is any nonsingular, integer 2×2 matrix, then there are at most two factorizations (and at least one) of $|\det(M)|$, $A_1B_2 = |\det(M)|$, and $A_2B_2 = |(M)|$ such that if $M_1 = A_1$, $M_2 = B_1$ or $M_1 - A_2$, $M_2 = B_2$ in Fig. 27, then the decimator produces (1,1) for all invocations. Moreover

$$A_1 = \gcd(a,b),\ B_1 = \frac{|\det(M)|}{\gcd(a,b)}$$

and

$$A_2 = \frac{|\det(M)|}{\gcd(c,d)},\ B_2 = \gcd(c,d).$$

Remark: Note that $gcd(a,0) = a$; hence, if M is diagonal, the two factorizations are the same, and there is only one unique factorization. This implies that for rectangular decimation, there is only one way to set the MDSDF parameters and get noncyclostatic behavior.

Example 2 illustrates two other points. First, it is only *sufficient* that the decimator produce 1 sample on each invocation for the additional constraints on decimator outputs to be satisfied by the balance equation solution. Second, it is only *sufficient* that the support matrix on the decimators output be integer valued for the additional constraints to be satisfied. Indeed, we have

$$W_{SM} = \begin{bmatrix} 6r_{S,1} & 0 \\ 0 & 6r_{S,2} \end{bmatrix},\ W_{MO} = \begin{bmatrix} 3r_{S,1} & 1.5r_{S,2} \\ 3r_{S,1} & -1.5r_{S,2} \end{bmatrix}$$

where W_{MO} is the support matrix on the decimators output. For the case where $M1 = 4$, $M2 = 1$, we have $r_{S,1} = 2$, $r_{S,2} = 1$, making W_{MO} noninteger valued. However, we do have that $|N(W_{MO})| = |N(W_{SM})|/|\det(M)|$, despite the fact that W_{MO} is noninteger valued and the decimator is cyclostatic.

3) Delays in the Generalized Model: Delays can be interpreted as translations of the buffer of produced values along the vectors of the support matrix (in the renumbered data space) or along the vectors in the basis for the sampling lattice (in the lattice data space). Fig. 28 illustrates a delay of (1,2) on a nonrectangular lattice.

C. Summary of Generalized Model

In summary, our generalized model for expressing nonrectangular systems has the following semantics.

- Sources produce data in accordance with the source data production method of Section III-B. The support matrix and lattice-generating matrix on the source's output arcs are specified by the source. The source produces a generalized (S_1, S_2) rectangle of data on each firing.
- An expander with expansion matrix L consumes (1,1) and produces the set of samples in $FPD(L)$ that is ordered as a generalized (L_1, L_2) rectangle of data, where L_1, L_2 are positive integers such that $L_1, L_2 = |\det(L)|$.
- A decimator with decimation matrix M consumes a rectangle (M_1, M_2) of data where this rectangle is interpreted according to the way it has been ordered (by the use of some rectangularizing function) by the actor feeding the decimator. It produces (1,1) on average. Unfortunately, there does not seem to be any way of making the decimators output any more concrete.

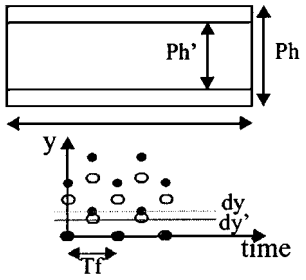

Fig. 29. Picture sizes and lattices for the two aspect ratios 4/3 and 16/9.

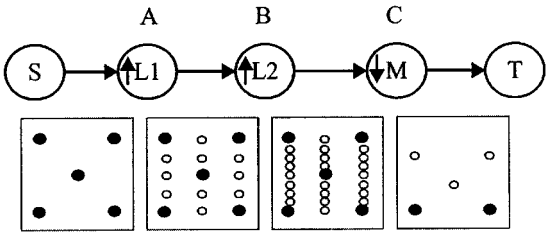

Fig. 30. System for doing multistage sampling structure conversion from 4/3 aspect ratio to 16/9 aspect ratio for a 2:1 interlaced TV signal.

- On any arc, the global ordering of the samples on that arc is established by the actor feeding the arc. The actor consuming the samples follows this ordering.

A set of balance equations are written down using the various factorizations. Additional constraints for arcs that feed a decimator are also written down. These are solved to yield the repetitions matrix for the network. A scheduler can then construct a static schedule by firing firable nodes in the graph until each node has been fired the requisite number of times, as given by the repetitions matrix.

IV. Multistage Sampling Structure Conversion Example

An application of considerable interest in current television practice is the format conversion from 4/3 aspect ratio to 16/9 aspect ratio for 2:1 interlaced TV signals. It is well known in 1-D signal processing theory that sample rate conversion can be done efficiently in many stages. Similarly, it is more efficient to do both sampling rate and sampling structure conversion in stages for multidimensional systems. The two aspect ratios and the two lattices are shown in Fig. 29. One way to do the conversion between the two lattices as shown previously is as shown in Fig. 30 [21]. We can easily calculate the various lattices and support matrices for this system, solve the balance equations, and develop a schedule [23].

V. Related Work

In [36], Watlington and Bove discuss a stream-based computing paradigm for programming video processing applications. Rather than dealing with multidimensional dataspaces directly, as is done in this paper, the authors sketch some ideas of how multidimensional arrays can be collapsed into 1-D streams using simple horizontal/vertical scanning techniques. They propose to exploit data parallelism from the 1-D stream model of the multidimensional system and to use dynamic (run-time) scheduling, in contrast to our approach in this paper of using a multidimensional stream model with static scheduling.

The Philips Video Signal Processor (VSP) is a commercially available processor designed for video processing applications [34]. A single VSP chip contains 12 arithmetic/logic units, four memory elements, six on-chip buffers, and ports for six off-chip buffers. These are all interconnected through a full cross-point switch. Philips provides a programming environment for developing applications on the VSP. Programs are specified as signal flow graphs. Streams are 1-D, as in [36]. Multirate operations are supported by associating a clock period with every operation. Because all of the streams are unidimensional, data-parallelism has to be exploited by inserting actors like multiplexors and demultiplexors into the signal flow graphs.

There has been interesting work done at Thomson-CSF in developing the array-oriented language (AOL) [12]. AOL is a specification formalism that tries to formalize the notion of array access patterns. The observation is that in many multidimensional signal processing algorithms, a chief problem is in specifying how multidimensional arrays are accessed. AOL allows the user to graphically specify the data tokens that need to be accessed on each firing by some block and how this pattern of accesses changes with firings.

The concrete data structures of Kahn and Plotkin [16], and later of Berry and Curien [3], is an interesting model of computation that may include MDSDF as a subset. Concrete data structures model most forms of real-world data structures such as lists, arrays, trees, etc. Essentially, Berry and Curien in [3] develop semantics for dataflow networks where the arcs hold concrete data structures, and nodes implement Kahn-Plotkin sequential functions. As future work, a combination of the scheduling techniques developed in this paper, the semantics work of [3], and the graphical syntax of [12] might prove to be a powerful model of computation for multidimensional programming.

There is a body of work that extends scheduling and retiming techniques for 1-D, single-rate dataflow graphs, for example [25], to single-rate multidimensional dataflow graphs [26] (retiming) [27], [35] (scheduling). Architectural synthesis from multirate, MDDFGs for rectangularly sampled systems is proposed in [31]. These works contrast with ours in that they do not consider modeling arbitrary sampling lattices, nor do they consider multidimensional dataflow as a high-level coordination language that can be used in high-level graphical programming environments for specifying multidimensional systems. Instead, they focus on graphs that model multidimensional nested loops and optimize the execution of such loops via retiming and efficient multiprocessor scheduling.

VI. Conclusion

A graphical programming model called multidimensional synchronous dataflow (MDSDF), based on dataflow that supports multidimensional streams, has been presented. We have

shown that the use of multidimensional streams is not limited to specifying multidimensional signal processing systems but can also be used to specify more general data exchange mechanisms, although it is not clear at this point whether these principles will be easy to use in a programming environment. Certainly, the matrix multiplication program in Fig. 14 is not very readable. An algorithm with less regular structure will only be more obtuse. However, the analytical properties of programs expressed this way are compelling. Parallelizing compilers and hardware synthesis tools should be able to do extremely well with these programs without relying on runtime overhead for task allocation and scheduling. At the very least, the method looks promising to supplement large-grain dataflow languages, much like the GLU "coordination language" makes the multidimensional streams of Lucid available in large-grain environment [15]. It may lead to special purpose languages but could also ultimately form a basis for a language that, like Lucid, supports multidimensional streams but is easier to analyze, partition, and schedule at compile time.

However, this coordination language appears to be most useful for specifying multidimensional, multirate signal processing systems, including systems that make use of nonrectangular sampling lattices and nonrectangular decimators and interpolators. The extension to nonrectangular lattices has been nontrivial and involves the inclusion of geometric constraints in the balance equations. We have illustrated the usefulness of our model by presenting a practical application (video format conversion) that can be programmed using our model.

REFERENCES

[1] R. H. Bamberger, "The directional filterbank: A multirate filterbank for the directional decomposition of images," Ph.D. dissertation, Georgia Inst. Technol., Atlanta, 1990.

[2] A. Benveniste, B. Le Goff, and P. Le Guernic, "Hybrid dynamical systems theory and the language "SIGNAL"," Institut National de Recherche en Informatique at en Automatique (INRIA), Le Chesnay Cedex, France, 838, 1988.

[3] G. Berry and P-L Curien, "Theory and practice of sequential algorithms: the kernel of the programming language CDS," in *Algebraic Methods in Semantics*. Cambridge, U.K.: Cambridge Univ. Press, 1988, pp. 35–88.

[4] S. S. Bhattacharyya, P. K. Murthy, and E. A. Lee, *Software Synthesis from Dataflow Graphs*. Norwood, MA: Kluwer, 1996.

[5] G. Bilsen, M. Engels, R. Lauwereins, and J. Peperstraete, "Static scheduling of multi-rate cyclo-static DSP applications," in *Proc. IEEE Workshop VLSI Signal Process.*, San Diego, CA, Oct. 1994, pp. 137–146.

[6] F. Bosveld, R. L. Lagendijk, and J. Biemond, "Compatible Spatio-Temporal Subband Encoding of HDTV," *Signal Process.*, vol. 28, no. 3, pp. 271–289, Sept. 1992.

[7] J. Buck, S. Ha, E. A. Lee, and D. G. Messerschmitt, "Multirate signal processing in Ptolemy," in *Proc. Int. Conf. Acoust., Speech, Signal Process.*, Toronto, ON, Canada, Apr. 1991, pp. 1245–48.

[8] ——, "Ptolemy: A framework for simulating and prototyping heterogeneous systems," *Int. J. Comput. Simulation*, vol. 4, pp. 155–182, Apr. 1994.

[9] P. Caspi, D. Pilaud, N. Halbwachs, and J. A. Plaice, "LUSTRE: A declarative language for programming synchronous systems," in *Conf. Rec. 14th Annu. ACM Symp. Principles Programming Languages*, Munich, Germany, Jan 1987, pp. 178–88.

[10] M. C. Chen, "Developing a multidimensional synchronous dataflow domain in Ptolemy," Dept. Elect. Eng. Comput. Sci., Univ. California, Berkeley, CA, 1994.

[11] J. Davis *et al.*, "Heterogeneous concurrent modeling and design in Java," Univ. California, Berkeley, Tech. Memo. UCB/ERL M01/12, EECS, 2001.

[12] A. Demeure, "Formalisme de Traitement du Signal: Array-OL," Thomson SINTRA ASM, Sophia Antipolis, Valbonne, France, 1994.

[13] E. Dubois, "The sampling and reconstruction of time-varying imagery with applications in video systems," *Proc. IEEE*, vol. 73, pp. 502–522, Apr. 1985.

[14] R. Hopkins, "Progress on HDTV broadcasting standards in the United States," *Signal Processing: Image Commun.*, vol. 5, pp. 355–78, Dec. 1993.

[15] R. Jagannathan and A. A. Faustini, "The GLU programming language," SRI Int., Comput. Sci. Lab., Menlo Park, CA, SRI-CSL-90–11, 1990.

[16] G. Kahn and G. D. Plotkin, "Concrete domains," *Theoretical Comput. Sci.*, vol. 121, no. 1-2, pp. 187–277, Dec. 1993.

[17] K. Konstantinides and J. R. Rasure, "The Khoros software development environment for image and signal processing," *IEEE Trans. Image Processing*, vol. 3, pp. 243–52, May 1994.

[18] S. Y. Kung, *VLSI Array Processors*. Englewood Cliffs, NJ: Prentice-Hall, 1988.

[19] E. A. Lee and D. G. Messerschmitt, "Static scheduling of synchronous data flow programs for digital signal processing," *IEEE Trans. Comput.*, vol. C-36, pp. 24–35, Jan 1987.

[20] E. A. Lee, "Multidimensional Streams Rooted in Dataflow," in *Proc. IFIP Working Conf. Architectures Compilation Techn. Fine Medium Grained Parallelism*, Orlando, FL, Jan. 1993.

[21] R. Manduchi, G. M. Cortelazzo, and G. A. Mian, "Multistage sampling structure conversion of video signals," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 3, pp. 325–40, Oct. 1993.

[22] R. M. Mersereau and T. C. Speake, "The processing of periodically sampled multidimensional signals," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-31, pp. 188–194, Feb. 1983.

[23] P. K. Murthy, "Scheduling Techniques for Synchronous and Multidimensional Synchronous Dataflow," Ph.D, Electron. Res. Lab., Univ. California, Berkeley, CA, 1996.

[24] P. K. Murthy, E. Cohen, and S. Rowland, "System canvas–A new design environment for embedded dsp and telecommunication systems," in *Proc. Ninth Int. Symp. Hardware/Software Codesign*, Copenhagen, Denmark, Apr. 2001, pp. 54–59.

[25] K. K. Parhi and D. G. Messerschmitt, "Static rate-optimal scheduling of iterative data-flow programs via optimum unfolding," *IEEE Trans. Comput.*, vol. 40, pp. 178–95, Feb. 1991.

[26] N. L. Passos and E. H. -M Sha, "Synchronous circuit optimization via multidimensional retiming," *IEEE Trans. Circuits Syst. II*, vol. 43, pp. 507–19, July 1996.

[27] ——, "Scheduling of uniform multidimensional systems under resource constraints," *IEEE Trans. VLSI Syst.*, vol. 6, pp. 719–30, Dec. 1998.

[28] H. Printz, "Automatic mapping of large signal processing systems to a parallel machine," Ph.D. dissertation, Carnegie Mellon Univ., Pittsburgh, PA, 1991.

[29] S. Ritz, M. Pankert, and H. Meyr, "High level software synthesis for signal processing systems," in *Proc. Int. Conf. Applicat.-Specific Array Process.*, Aug. 1992, pp. 679–93.

[30] G. C. Sih and E. A. Lee, "A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures," *IEEE Trans. Parallel Distrib. Syst.*, vol. 4, pp. 175–187, Feb. 1993.

[31] V. Sundararajan and K. Parhi, "Synthesis of folded, pipelined architectures for multidimensional multirate systems," in *Proc. ICASSP*, May 1998, pp. 3089–3092.

[32] P. P. Vaidyanathan, "Fundamentals of multidimensional multirate digital signal processing," *Sadhana*, vol. 15, pp. 157–176, Nov. 1990.

[33] ——, *Multirate Systems and Filter Banks*. Englewood Cliffs, NJ: Prentice-Hall, 1993.

[34] K. A. Vissers *et al.*, "Architecture and Programming of Two Generations Video Signal Processors," *Microprocess. Microprogram.*, vol. 41, no. 5-6, pp. 373–90, Oct 1995.

[35] J. Q. Wang, E. H.-M. Sha, and N. L. Passos, "Minimization of memory access overhead for multidimensional DSP applications via multilevel partitioning and scheduling," *IEEE Trans. Circuits Syst. II*, vol. 44, pp. 741–53, Sept. 1997.

[36] J. A. Watlington and V. M. Bove Jr, "Stream-based computing and future television," in *Proc. 137th SMPTE Techn. Conf.*, Apr. 1995.

Praveen K. Murthy (M'97) received the B.S.E.E degree from the Georgia institute of Technology, Atlanta, in 1989 and the M.S. and Ph.D. degrees in electrical engineering and computer science from the University of California, Berkeley, in 1993 and 1996, respectively.

He is a Member of Research Staff of the Advanced CAD Research Group at Fujitsu Labs of America (FLA), Sunnyvale, CA. His research interests span all areas of system-level design, verification, and synthesis including simulation, techniques for producing optimized software implementations, semantics of different models of computation, and software tools for rapid prototyping. Prior to joining FLA, he was with Angeles Design Systems and Cadence Design Systems. He has also consulted for Berkeley Design Technologies, Inc., in the area of DSP architectures and tools. He has co-authored numerous refereed papers and the book *Software Synthesis from Dataflow Graphs* (Boston, MA: Kluwer).

Edward A. Lee (F'94) received the B.S. degree from Yale University, New Haven, CT, in 1979, the S.M. degree from the Massachusetts Institute of Technology, Cambridge, in 1981, and the Ph.D. degree from the University of California at Berkeley (UC Berkeley) in 1986.

He is a Professor with the Electrical Engineering and Computer Science Department, UC Berkeley. His research interests center on design, modeling, and simulation of embedded, real-time computational systems. He is Director of the Ptolemy project at UC Berkeley. He is co-author of four books and numerous papers. From 1979 to 1982, he was a Member of Technical Staff at Bell Telephone Laboratories, Holmdel, NJ, in the Advanced Data Communications Laboratory. He is a co-founder of BDTI, Inc., where he is currently a Senior Technical Advisor, is co-founder of Agile Design, Inc., and has consulted for a number of other companies.

Dr. Lee was a National Science Foundation Presidential Young Investigator and won the 1997 Frederick Emmons Terman Award for Engineering Education.