[8] J. Buck, S. Ha, E. A. Lee, D. G. Messerschmitt, "Ptolemy: a Framework for Simulating and Prototyping Heterogeneous Systems", *International Journal of Computer Simulation*, to appear, 1993.

[9] J. Buck and E. A. Lee, "The Token Flow Model," presented at *Data Flow Workshop*, Hamilton Island, Australia, May, 1992.

[10] P. Caspi, D. Pilaud, N. Halbwachs, and J. A. Plaice, "LUSTRE: A Declarative Language for Programming Synchronous Systems," *Conference Record of the 14th Annual ACM Symp. on Principles of Programming Languages,* Munich, Germany, January, 1987.

[11] A. L. Davis and R. M. Keller, "Data Flow Program Graphs", *Computer*, **15(2)**, February, 1982.

[12] J.B. Dennis, "First Version Data Flow Procedure Language", Technical Memo MAC TM61, May, 1975, MIT Laboratory for Computer Science.

[13] P. N. Hilfinger, "Silage Reference Manual, DRAFT Release 2.0", Computer Science Division, EECS Dept., University of California, Berkeley, CA 94720, July 8, 1989.

[14] R. Jagannathan and A. A. Faustini, "The GLU Programming Language," Tech. Report SRI-CSL-90-11, Computer Science Laboratory, SRI International, Menlo Park, CA 94025, USA, November 1990.

[15] S. Y. Kung, *VLSI Array Processors*, Prentice-Hall, Englewood Cliffs, New Jersey, 1988.

[16] E. A. Lee and D. G. Messerschmitt, "Static Scheduling of Synchronous Data Flow Programs for Digital Signal Processing" *IEEE Transactions on Computers*, January, 1987.

[17] E. A. Lee and D. G. Messerschmitt, "Synchronous Data Flow" *IEEE Proceedings*, September, 1987.

[18] E. A. Lee, "Consistency in Dataflow Graphs", *IEEE Transactions on Parallel and Distributed Systems",* Vol. 2, No. 2, April 1991.

[19] J. McGraw, "Sisal: Streams and Iteration in a Single Assignment Language", *Language Reference Manual,* Lawrence Livermore National Laboratory, Livermore, CA 94550.

[20] H. Printz, "Automatic Mapping of Large Signal Processing Systems to a Parallel Machine," Memorandum CMU-CS-91-101, School of Computer Science, Carnegie Mellon University, Ph.D. Thesis, May 15, 1991.

[21] S. Ritz, M. Pankert, and H. Meyr, "High Level Software Synthesis for Signal Processing Systems," in *Proc. of the Int. Conf. on Application Specific Array Processors,* IEEE Computer Society Press, August 1992.

[22] G.C. Sih, E.A. Lee, "A Compile-Time Scheduling Heuristic for Interconnection-Constrained Heterogeneous Processor Architectures", to appear, *IEEE Trans. on Parallel and Distributed Systems,* 1992.

[23] G. C. Sih and E. A. Lee, "Declustering: A New Multiprocessor Scheduling Technique," to appear in *IEEE Trans. on Parallel and Distributed Systems,* 1992.

[24] D. B. Skillcorn, "Stream Languages and Data-Flow," in *Advanced Topics in Dataflow Computing,* ed. L. Bic and J.-L. Gaudiot.

[25] P. A. Suhler, J. Biswas, K. M. Korner, J. C. Browne, "TDFL: A Task-Level Dataflow Language", *J. on Parallel and Distributed Systems,* 9(2), June 1990.

6.0 Caveats

A programming model based on dataflow that supports multidimensional streams has been outlined. However, we have only defined a language in sufficient detail to illustrate some simple examples. It is not clear at this point that a language based on these principles will be easy to use. Certainly the matrix multiplication program in figure 18 is not very readable. Algorithms with less regular structure will only be more obtuse. This difficulty will be exacerbated when a multidimensional DF language based on the token flow model is developed. However, the analytical properties of programs expressed this way are compelling. Parallelizing compilers should be able to do extremely well with these programs without relying on runtime overhead for task allocation and scheduling. We conclude, therefore, that further investigation is certainly warranted. At the very least, the method looks promising to supplement large-grain dataflow languages. 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.

7.0 References

[1] Arvind and J. D. Brock, "Resource Managers in Functional Programming," *J. of Parallel and Distributed Computing,* Vol. 1, No. 5-21, 1984

[2] E. A. Ashcroft, "Proving Assertions about Parallel Programs," *J. of Computer and Systems Science,* Vol. 10, No. 1, pp. 110-135, 1975.

[3] E. A. Ashcroft and R. Jagannathan, "Operator Nets," in *Proc. IFIP TC-10 Working Conf. on Fifth-Generation Computer Architectures,* North-Holland, The Netherlands, 1985.

[4] A. Benveniste, B. Le Goff, and P. Le Guernic, "Hybrid Dynamical Systems Theory and the Language 'SIGNAL'", Research Report No. 838, Institut National de Recherche en Informatique at en Automatique (INRIA), Domain de Voluceau, Rocquencourt, B. P. 105, 78153 Le Chesnay Cedex, France, April 1988.

[5] S. Bhattacharyya and E. A. Lee, "Scheduling Synchronous Dataflow Graphs for Efficient Looping," to appear in *J. of VLSI Signal Processing,* 1992.

[6] J. Bier, E. Goei, W. Ho, P. Lapsley, M. O'Reilly, G. Sih and E.A. Lee, "Gabriel: A Design Environment for DSP," *IEEE Micro*, October 1990, Vol. 10, No. 5, pp. 28-45.

[7] J. Buck, S. Ha, E. A. Lee, and D. G. Messerschmitt, "Multirate Signal Processing in Ptolemy", *Proc. of the Int. Conf. on Acoustics, Speech, and Signal Processing*, Toronto, Canada, April, 1991.

A data-driven or demand-driven style of computation can ameliorate such problems. We will now show, however, that neither data-driven nor demand driven execution is a panacea. Consider the two SDF graphs in figure 21. *Source* actors, such as A and B, have no inputs, and hence are always enabled. They model internally generated token streams, or external inputs, which have side effects. Modeling such external inputs was a principal motivation for the development of stream data types in Id [1]. *Sink* actors, such as E and F can similarly model program outputs. Suppose that $M \neq N$. If a data-driven style of execution is used at run time, ignoring the information about the number of tokens produced and consumed, then some additional control is required to keep A or B from producing an unbounded number of tokens that will accumulate in memory. That additional control is not provided by the data-driven model. A demand-driven style of execution solves the problem for the A-B-C graph, but encounters the same problem with the D-E-F graph. Again, additional control is needed. A typical approach in data-driven execution is to throttle a producer when its tokens are not being consumed, but this approach incurs run-time overhead.

SDF, MD-SDF, and the dynamic extensions based on the token flow model solve all of these problems. The relative firing rates for the graphs in figure 21 are determined at compile time, so the model of execution is neither data driven nor demand driven. Moreover, the execution model is complete, requiring no additional control, and there is no loss of concurrency. Similar results are obtained using the token flow for graphs that do not fit the SDF model [9]. For multidimensional streams, permissible firing patterns are again determined at compile time. For example, suppose the index space for the self-loop of actor C in figure 20 is doubly infinite (infinite in both dimensions). The permissible wavefront pattern of execution and all the parallelism it implies can be determined at compile time, and appropriate run-time control flow can be synthesized.

Figure 21. Two SDF graphs that illustrate problems with both demand and data-driven execution.

giving the number of samples produced and consumed by an actor. This is necessary to use multi-dimensional dataflow over non-rectangular index spaces.

5.0 Neither Demand nor Data-Driven Execution

Since streams are data structures like any other, streams of streams can be formed, although functional operations on such composite objects can be ambiguous. Consider for example a stream s of streams p_i of infinite length. Consider a function $f(s)$ applied to the stream. This constitutes the nested infinite iteration

forall i in (0 ... ∞) { forall j in (0 ... ∞) { $f(p_{i,j})$ }},

where $p_{i,j}$ represents the $j-th$ element of the stream p_i.We assume non-strict semantics (necessary for such operators on infinite streams to make sense) and functional semantics (so that we can use the "forall" construct, which implies no ordering constraint). For functional operators on finite sets of data elements, the order of execution does not matter. However, the order of execution of the above iteration, although unspecified, is clearly important. Consider the difference between

for i in (0 ... ∞) { forall j in (0 ... ∞) { $f(p_{i,j})$ }}.

and

forall i in (0 ... ∞) { for j in (0 ... ∞) { $f(p_{i,j})$ }},

where the "for" construct implies sequential execution.

For practical applications, we can restrict streams of streams so that they are infinite in one dimension only, in which case there is no ambiguity of specification. There remains, however, some ambiguity of execution. Consider the nested iteration

forall i in (0 ... M) { forall j in (0 ... ∞) { $f(p_{i,j})$ }}

If this is executed as

for i in (0 ... M) { for j in (0 ... ∞) { $f(p_{i,j})$ }},

then only the infinite stream for $i = 0$ will be processed.

These are identified in figure 19. 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.

4.6 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 figure 20. Assume, as is common, that dimension 1 indexes the row in the index space, and dimension 2 the column, as shown in figure 9. 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 [15] can be applied to programs defined using this model.

4.7 Asynchronous Actors

The token flow model, which permits SWITCH and SELECT actors, can be easily extended to multiple dimensions by simply allowing symbolic placeholders inside the M-tuples

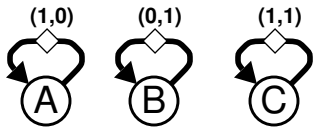

Figure 20. Three macro actors with state represented as a self-loop.

needed in a single-assignment specification to carry a variable forward in the index space [15]. 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 LxN matrix product. The MD-SDF graph implementing this is shown in figure 18. The key actors used for this are:

Upsample: In specified dimension(s), consumes 1 and produces N, inserting zero values.

Repeat: In specified dimension(s), consumes 1 and produces N, repeating values.

Downsample: In specified dimension(s), consumes N and produces 1, discarding samples.

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

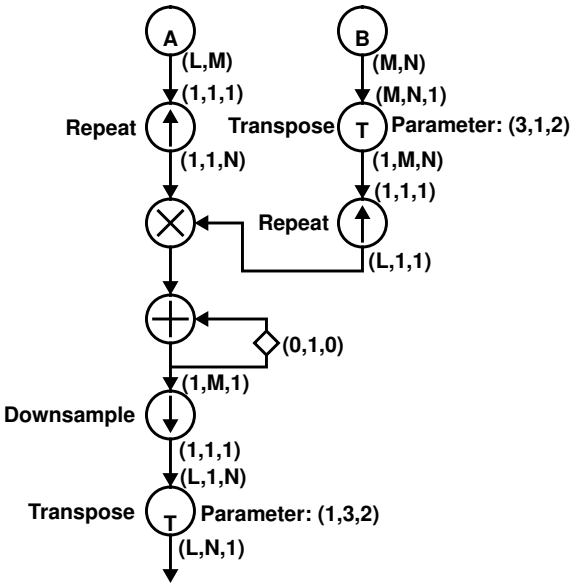

Figure 18. Matrix multiplication in MD-SDF.

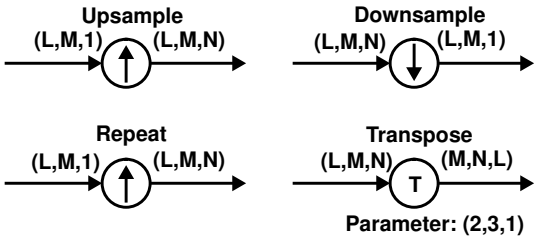

Figure 19. : Some key MD-SDF actors that affect the flow of control.

of the arc, the specified dimensions are assumed to be the lower ones (lower number, earlier in the M-tuple). Hence, the two graphs in figure 15 are equivalent.

We can specify a comparable rule for delays:

- 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 figure 16 are equivalent.

4.5 Matrix Multiplication

As another example, consider a fine-grain specification of matrix multiplication. Suppose we are to multiply an LxM matrix by an MxN matrix. In a three dimensional index space, this can be accomplished as shown in figure 17. A 3D index space is used. 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

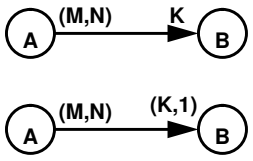

Figure 15. Rule for augmenting the dimensionality of a producer or consumer.

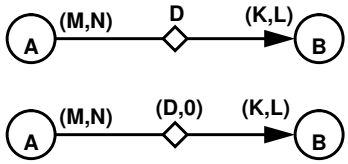

Figure 16. Rule for augmenting the dimensionality of a delay.

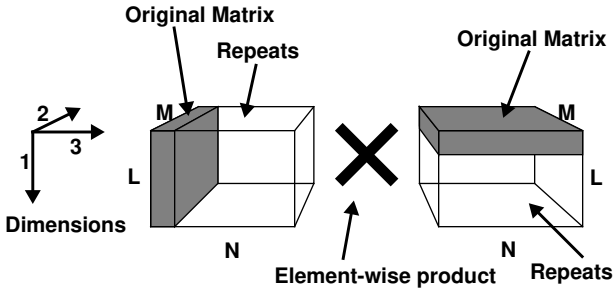

Figure 17. Matrix multiplication represented schematically.

A delay in MD-SDF in associated with a tuple as shown in figure 13. 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 done in reduced dependence graphs [15].

Using MD-SDF delays, the repeated inner product can be specified as shown in figure 14. The only significant difference between this and figure 13 is the multidimensional delay. Its effect is illustrated schematically in figure 14, where the index space for the output of the delay is shown. The shaded area is the initial condition specified by the delay. Note that the index space for each arc in the system can be (and usually will be) different, unlike reduced dependence graphs [15].

4.4 Mixing Dimensionality

Note that in figure 14, 2D and 1D-SDF are mixed. 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

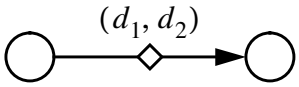

Figure 13. A delay in MD-SDF is multidimensional.

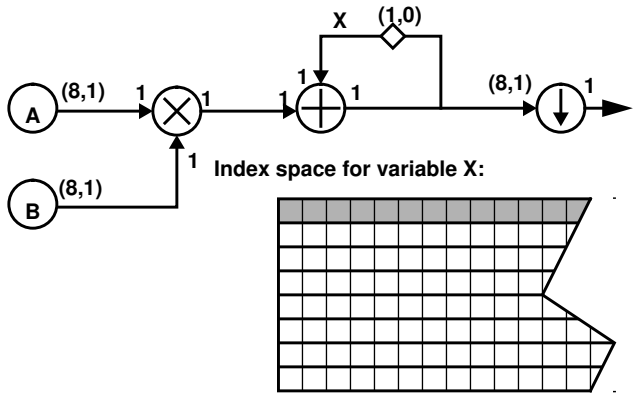

Figure 14. Repeated inner products in MD-SDF.