Integrating VSIPL Support in the Dataflow Interchange Format

Chia-Jui Hsu and Shuvra S. Bhattacharyya

Department of Electrical and Computer Engineering,
and Institute for Advanced Computer Studies
University of Maryland, College Park, MD 20742, USA
{jerryhsu,ssb}@eng.umd.edu

Outline

- Dataflow models
- Objectives and developments
  - Dataflow Interchange Format
  - The DIF package
  - Porting mechanism
  - Software synthesis framework
- VSIPL integration
  - Intermediate VSIPL actor library
  - DIF-to-VSIPL synthesis capability
- Current Status

Dataflow Models of Computation

- Synchronous dataflow (SDF) [7]: static multi-rate behavior
- Boolean/integer dataflow: Turing complete models
- Multidimensional synchronous dataflow (MDSDF) [8]: matrix / image
- Scalable synchronous dataflow (SSDF): block processing
- Cyclo-static dataflow (CSDF): static phased behavior
- The processing graph method: US Naval Research Laboratory
- Parameterized dataflow: reconfigurable quasi-static DF

- Commercial tools
  - Agilent ADS, MCCI Autocoding Toolset, Synopsis Cocentric System Studio, Gedae, National Instruments LabVIEW, MLDesigner.
- Research-oriented tools
  - Compaan from Leiden University, Grape from K. U. Leuven, PeaCE from Seoul National University, Ptolemy II from U. C. Berkeley, StreamIt from MIT.

Objectives and Developments

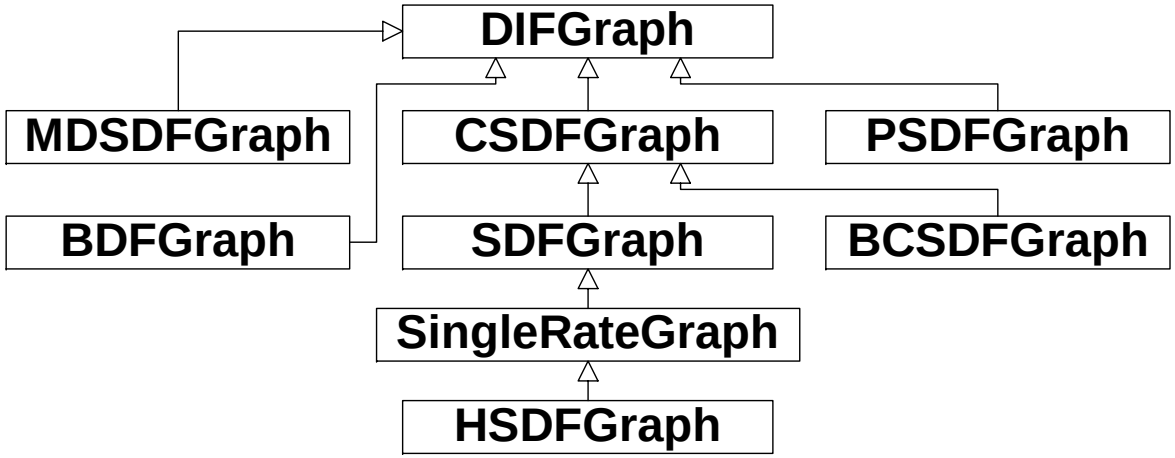
1. **Provide a standard language for specifying dataflow semantics.**
   - **Dataflow Interchange Format (DIF) [2]**
2. **Provide a software package for working with and developing dataflow models.**
   - **The DIF package [2]**
3. **Facilitate transferring DSP applications across dataflow-based design tools.**
   - **The DIF porting mechanism [3]**
   - **Intermediate VSIPL actor library [5]**
4. **Automate software implementations of DSP system designs from dataflow modeling specifications.**
   - **The DIF-to-C software synthesis framework [4]**
   - **DIF-to-VSIPL synthesis capability [5]**

The DIF Language Syntax

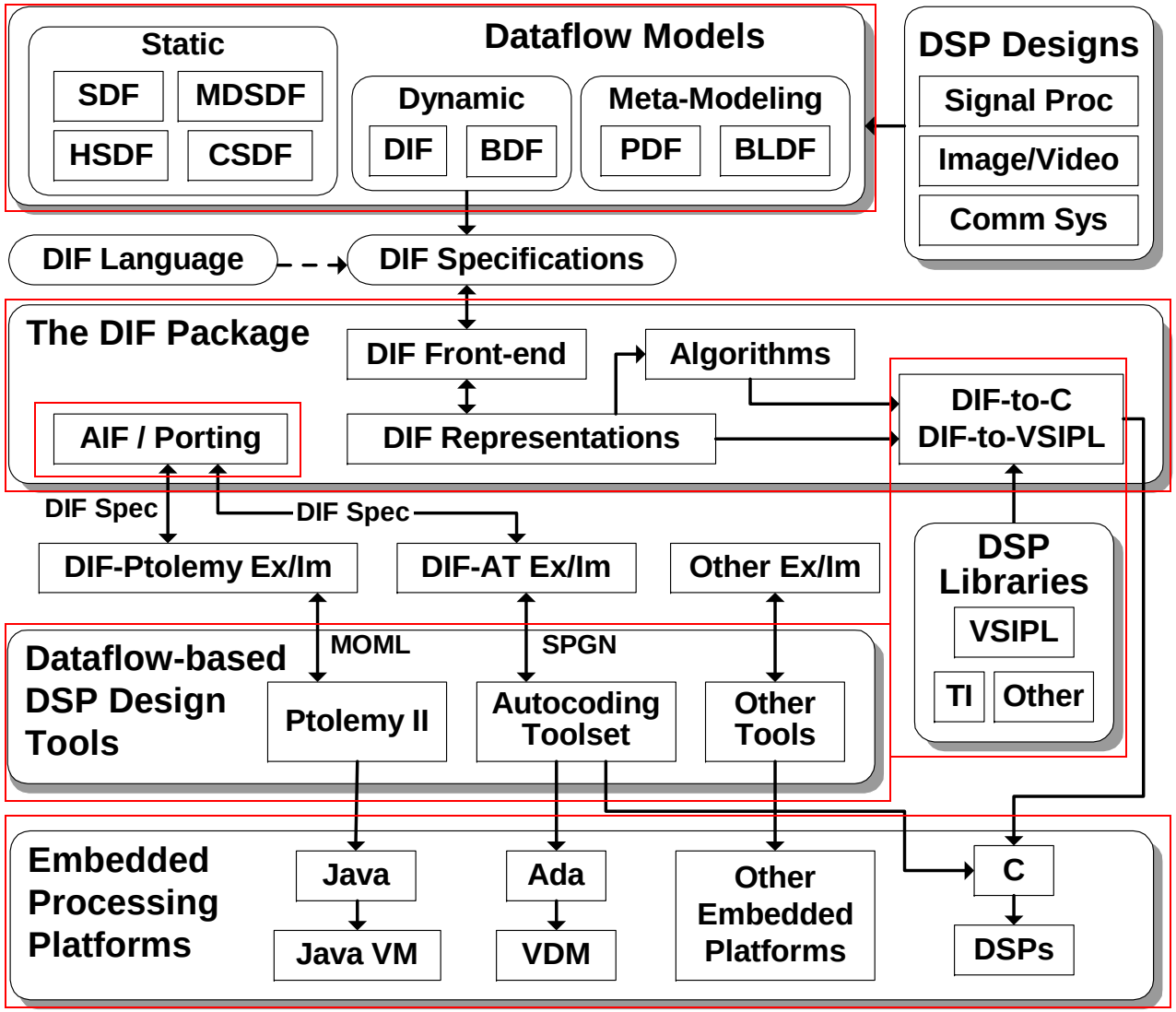
```
dataflowModel graphID {
  basedon { graphID; }
  topology {
    nodes = nodeID, ...;
    edges = edgeID (srcNodeID, snkNodeID), ...; }
  interface {
    inputs = portID [:nodeID], ...;
    outputs = portID [:nodeID], ...; }
  parameter {
    paramID [:dataType];
    paramID [:dataType] = value;
    paramID [:dataType] : range; }
  refinement {
    subgraphID = supernodeID;
    subPortID : edgeID;
    subParamID = paramID; }
  builtInAttr {
    [elementID] = value;
    [elementID] = id;
    [elementID] = id1, id2, ...; }
  attribute usrDefAttr{
    [elementID] = value;
    [elementID] = id;
    [elementID] = id1, id2, ...; }
  actor nodeID {
    computation = stringValue;
    attrID [:attrType] [:dataType] = value;
    attrID [:attrType] [:dataType] = id;
    attrID [:attrType] [:dataType] = id1, ...; }
}
```

The DIF Package

- **DIF representation**
  - Internal structures (Java classes) for representing and manipulating dataflow graphs.
- **Algorithm implementation**
  - Dataflow-based analysis, scheduling, and optimization.
- **DIF front-end (language parser)**
  - Translate between DIF specifications and DIF representations.
- **Infrastructure**
  - Porting
  - Software synthesis



The Methodology of Using DIF



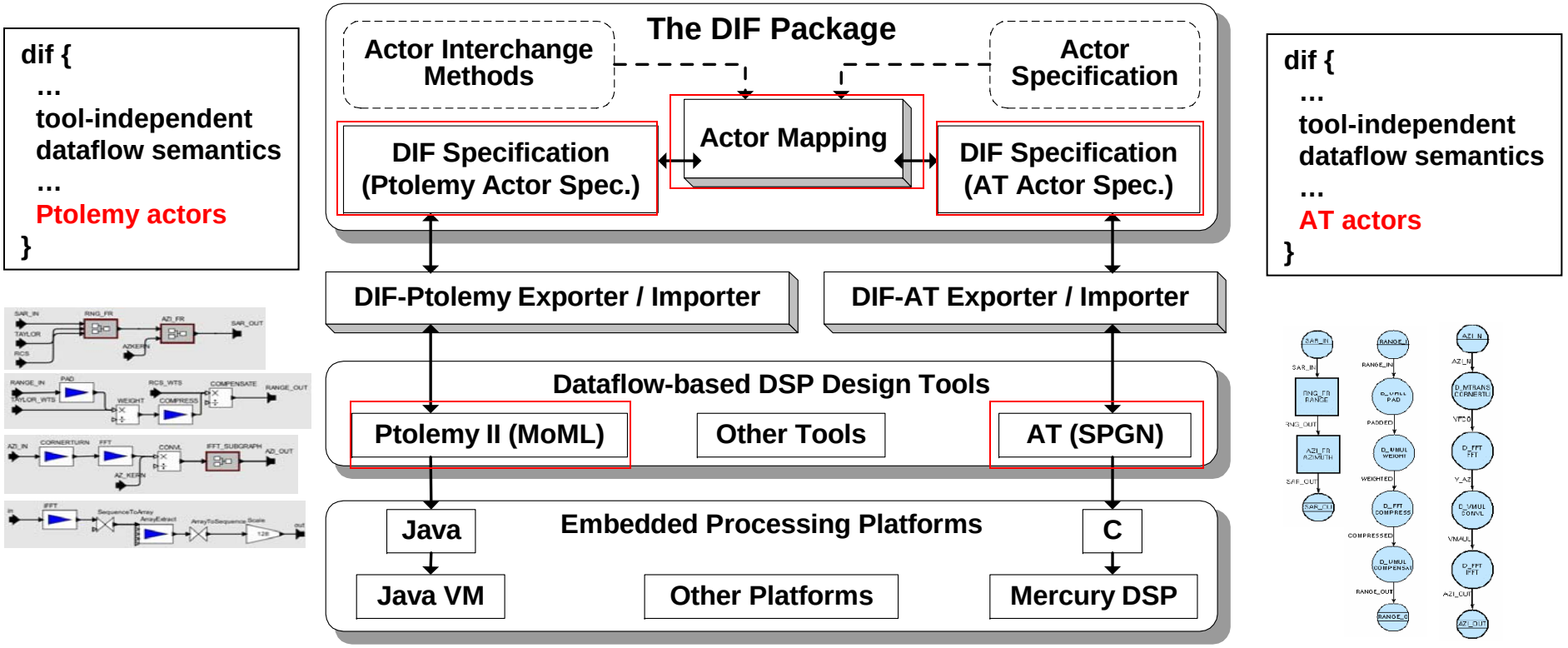
Porting DSP Applications

- **Tools may have complementary features:**
  - **simulation vs. synthesis, hardware vs. software support, etc.**
- **Portability across tools is equivalent to portability across all underlying platforms supported by tools.**

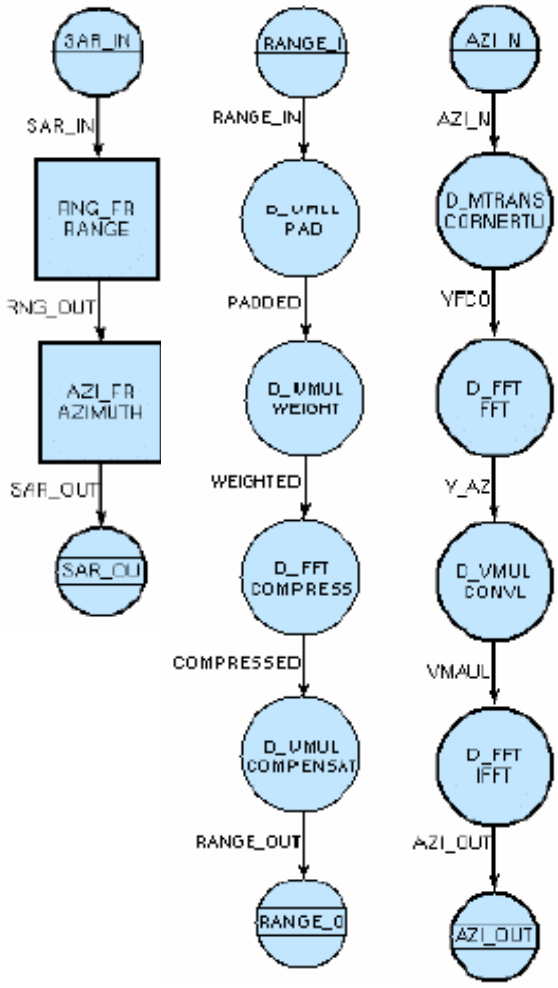
- **Except for the actor information, the DIF specifications for a DSP application are tool-independent in dataflow semantics.**
- **Porting DSP applications can be achieved by properly mapping the tool-dependent actors while transferring the dataflow semantics unaltered.**
- **Actor Interchange Format: specify how to map actors across a pair of tools.**

Porting Mechanism

1. Exporting: Export a DSP application from a design tool to DIF.
2. Actor mapping: Interchange actor information in DIF specification.
3. Importing: Import DIF specification to another design tool.



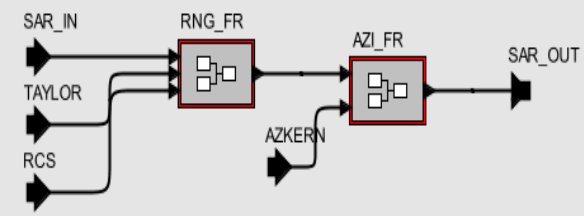
Porting Demonstration

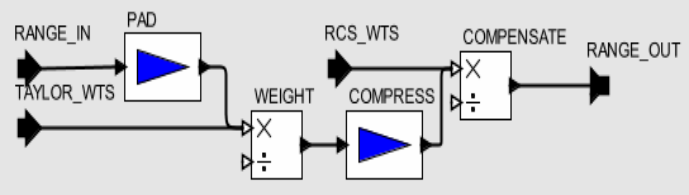


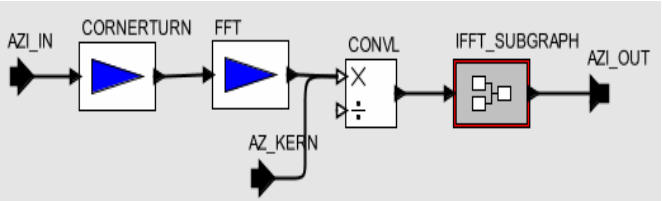

```
dif RNG_FR {
  topology { nodes = PAD, WEIGHT, COMPRESS, COMPENSATE;
    edges = PADDED(PAD,WEIGHT), WEIGHTED(WEIGHT,COMPRESS), …;}
  interface { inputs = RANGE_IN : PAD, …; outputs = RANGE_OUT : … ;}
  parameter { NFFT; NR; NPAD; PAD_VAL = (0.0, 0.0); }
  actor WEIGHT { computation = "D_VMUL";
    N = NFFT; X = PADDED; Y = TAYLOR_WTS; Z = WEIGHTED; }
  …
}
dif AZI_FR { …
  actor CORNERTURN { computation = "D_MTRAN"; M = NFFT;…;}
  actor FFT { computation = "D_FFT"; N = NFFT; FI = 0; X = YFCO; Y =Y_AZ;}
  …}
dif FR_SAR { … }
```

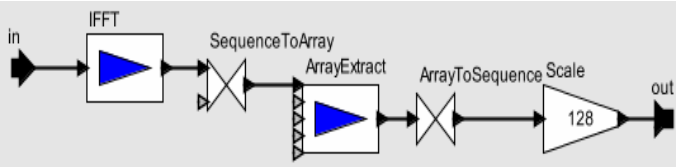
```
actor ptolemy.domains.sdf.lib.FFT <- D_FFT | ifExpression("FI == 0") {
    order : PARAMETER <- N | conditionalAssign("log(N)/log(2)", "(log(N)/log(2))-
        rint(log(N)/log(2)) == 0");
    input : INPUT <- X; output : OUTPUT <- Y; }
actor ptolemy.actor.lib.MultiplyDivide <- D_VMUL {
    multiply : INPUT <- X, Y; output : OUTPUT <- Z;}
graph ptolemy.actor.TypedCompositeActor <- D_FFT |
        ifExpression("FI == 1 && M != N") { … }
…
```

Porting Demonstration

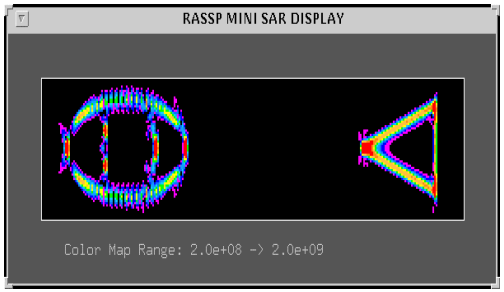


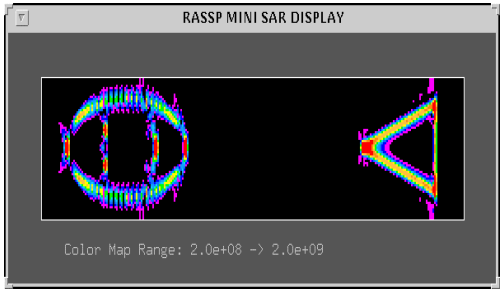






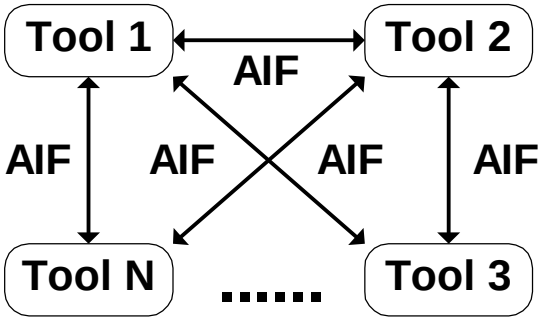

```
dif IFFT_SUBGRAPH { …
  actor IFFT { computation = "ptolemy.domains.sdf.lib.IFFT";
    order : PARAMETER = 7.0; input : INPUT = in; output : OUTPUT = e1; }
  … }
dif RNG_FR { …
  actor WEIGHT { computation = "ptolemy.actor.lib.MultiplyDivide";
    multiply : INPUT = PADDED,TAYLOR_WTS; …;
  …}
dif AZI_FR { …
  actor FFT { computation = "ptolemy.domains.sdf.lib.FFT";
    order : PARAMETER = 7.0; input : INPUT = YFCO; …;}
  … }
dif FR_SAR { … }
```



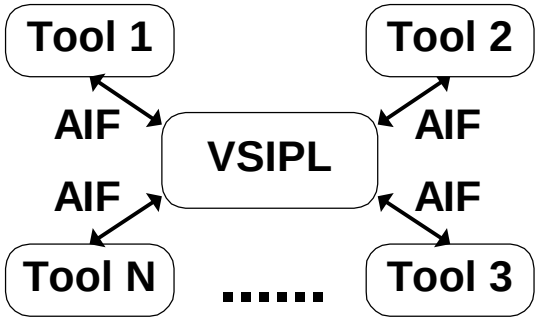


Intermediate VSIPL Actor Library

- The DIF / AIF porting mechanism provides an efficient approach in porting across a pair of tools.
- When many tools in the porting space, the requirement of actor mapping specifications grows quadratically.



N(N-1)/2



N

- We use VSIPL as an intermediate actor library, and the actor mapping operates by mapping “to” and “from” VSIPL.

DIF-to-C Software Synthesis Framework

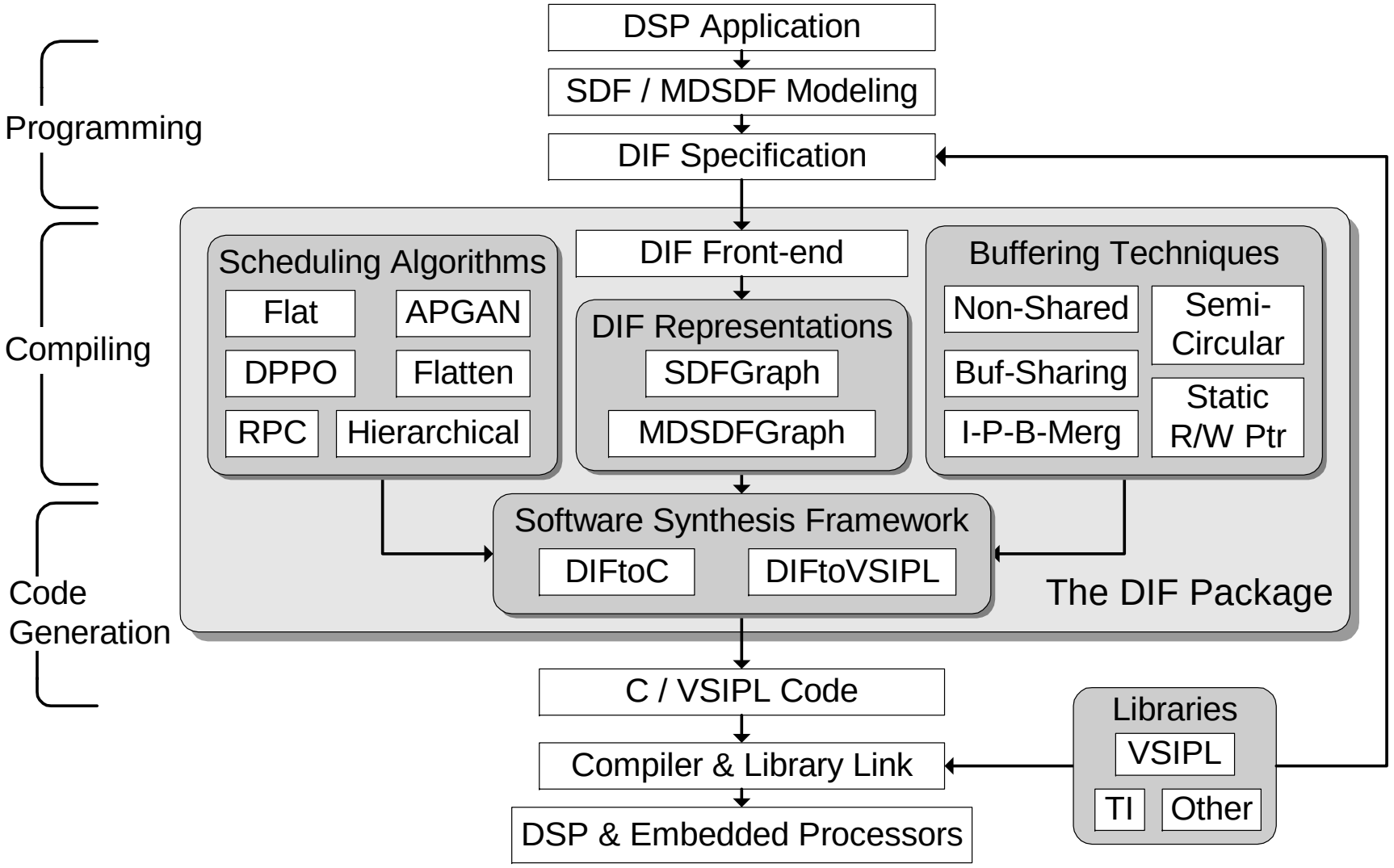
- **Automatically generate C implementations from dataflow modeling of DSP systems programmed in DIF.**
- **Designers need only to specify:**
  - **Dataflow graph topology**
  - **Hierarchy**
  - **Dataflow attributes (productions, consumptions, delays, etc.)**
  - **Actor attributes (function associations, edge/port connections, parameters, etc.)**
  - **Relevant software attributes (data types, etc.)**
- **Synchronous dataflow (SDF) semantics: SDF is the most mature model for dataflow-based DSP design.**

Features

- **Library-neutral: DIF programmers can associate actors with desired C functions (either user-designed or from libraries).**
  - **Most PDSPs and embedded processors provide C compilers; and many PDSP vendors and third-party companies provide hand-optimized C libraries.**
  - **Link between coarse-grain dataflow techniques with fine-grain actor optimizations and platform-dependent compilers.**

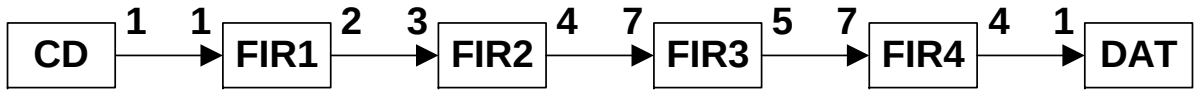
- **The DIF package provides various dataflow models, scheduling algorithms, and buffering techniques.**
  - **Designers can easily explore different combinations and determine trade-offs.**

DIF-to-C/VSIPL Software Synthesis Flow



DIF-to-C Demonstration

CD to DAT:

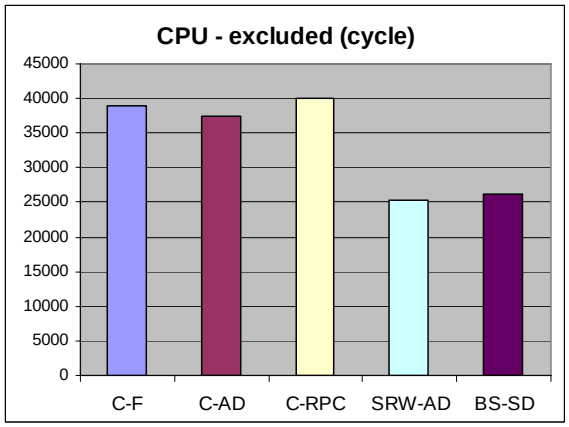


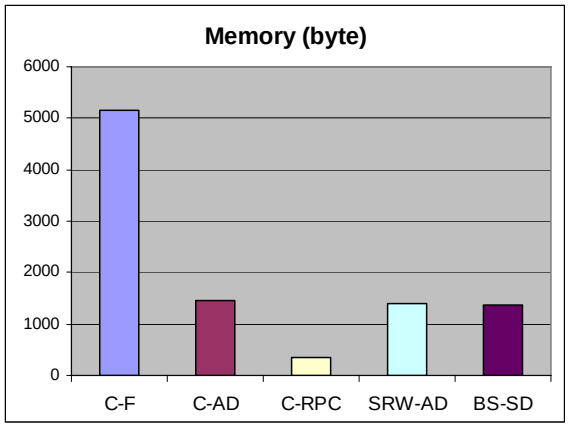

```
sdf cd2dat{
   topology {
      nodes = A, B, C, D, E, F;
      edges = e1(A,B), e2(B,C), e3(C,D), e4(D,E), e5(E,F); }
   production { e1 = 1; e2 = 2; e3 = 4; e4 = 5; e5 = 4; }
   consumption { e1 = 1; e2 = 3; e3 = 7; e4 = 7; e5 = 1; }
   attribute datatype { e1 = "float"; e2 = "float"; …; }
   parameter {
      coefs1 : "float" = [1.252648e-004, 5.114661e-004, … ];
      coefs2 : "float" = [1.032540e-004, 6.546872e-004, … ];
      coefs3 : "float" = [3.177298e-004, 5.464669e-004, … ];
      … }
   actor A {
      computation = "WavRead"; output = e1; readerFP = reader; }
   actor B {
      computation = "FIRAmp";
      input = e1;   output = e2; taps = coefs1;   feedback = dataB;
      interpolation = 2;   decimation = 1; decimationPhase = 0;
  … }
…
```

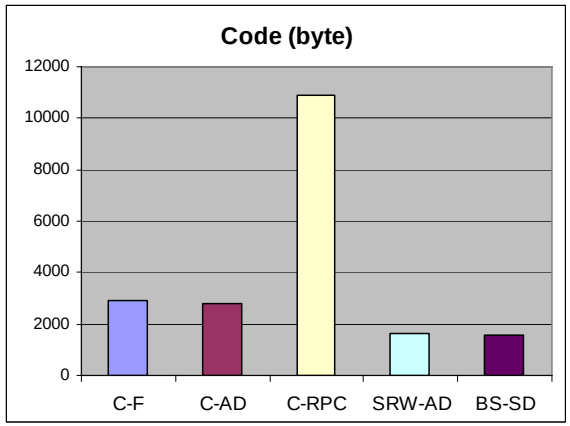
```
int main(int argc, char* argv[]) {
   float e1[1];      int e1_r = 0, e1_w = 0;
   float e2[6];      int e2_r = 0, e2_w = 0;
   float e3[56];     int e3_r = 0, e3_w = 0;
   float e4[280];    int e4_r = 0, e4_w = 0;
   float e5[4];      int e5_r = 0, e5_w = 0;
   …
   for (i=0; i<atoi(argv[1]); i++) {
      e4_r = 0; e4_w = 0;
      for (i2=0; i2<7; i2++) {
         e3_r = 0; e3_w = 0;
         for (i3=0; i3<7; i3++) {
            e2_r = 0; e2_w = 0;
            for (i4=0; i4<3; i4++) {
               e1_r = 0; e1_w = 0;
               WavRead(e1 + e1_w, reader);
               e1_w = (e1_w + 1);
               FIRAmp(e1 + e1_r, e2 + e2_w, …);
               e1_r = (e1_r + 1); e2_w = (e2_w + 2);
            } …
```

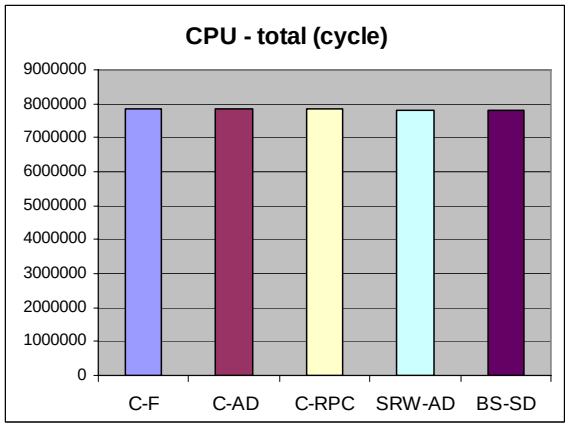
DIF-to-C Demonstration

- **Generated C code:**
  - **A looped sequence of function invocations interleaved with buffer management routines.**
  - **Buffer allocation, parameter declaration, subroutine generation.**
- **Memory, code size, CPU, vs different scheduling and buffering combinations.**
  - **Compile and simulate on TI CSS without any compiler optimization.**







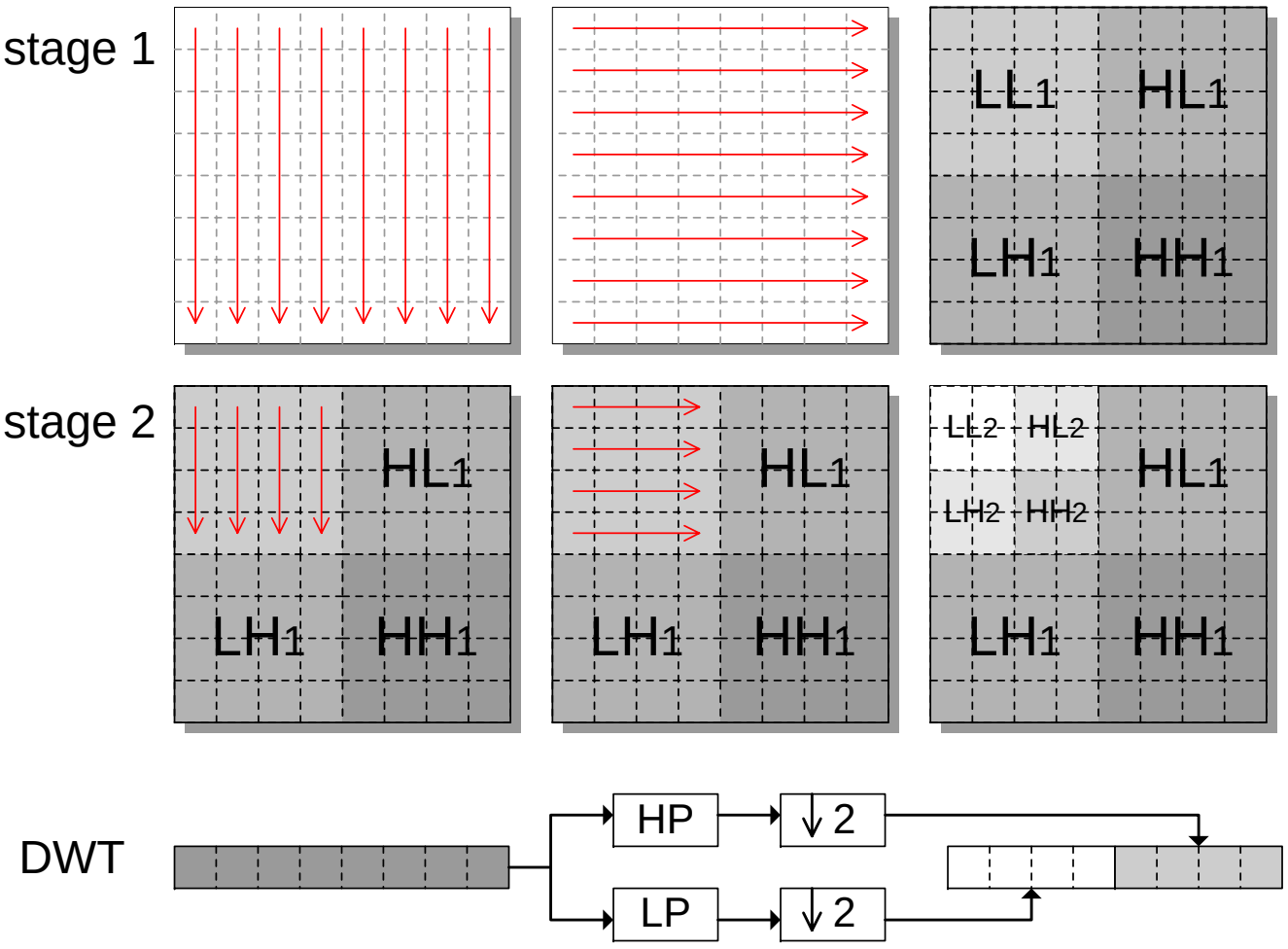


Limitations of SDF and generic C

- **Stream-based SDF semantics is insufficient to model multi-dimensional DSP systems.**
  - **Data dimension, locality, and data-level parallelism.**
- **Multi-dimensional synchronous dataflow (MDSDF) supports multi-dimensional representation in dataflow.**

- **Major issues in software synthesis for MDSDF:**
  - **Multi-dimensional buffer space v.s. one-dimensional memory layout.**
  - **Multi-dimensional production / consumption rate v.s. one-dimensional C pointer**
    - **function(int*, …)**

2D Discrete Wavelet Transform



- **Generic C: Hard to access just by 1D C pointer.**
- **VSIPL: Matrix view, Vector view, Submatrix view.**

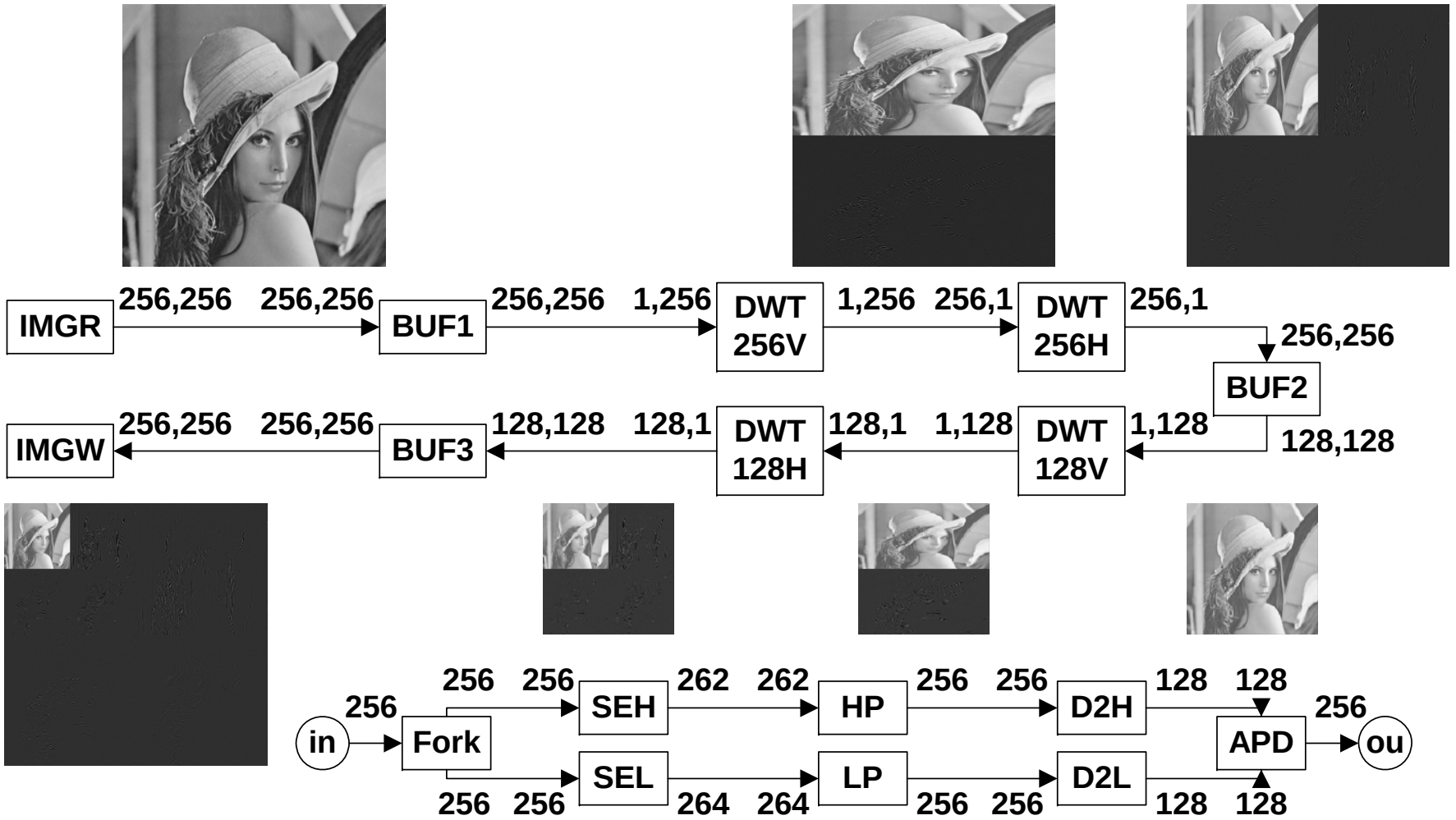
VSIPL

- **Vector, Signal, and Image Processing Library (VSIPL)**
- **VSIPL adds a layer of abstraction (blocks and views):**
  - **VSIPL blocks: contiguous memory spaces where data is stored.**
  - **VSIPL views: VSIPL functions operate on views in a way that sets or subsets of data can be accessed as vectors (1-D), matrices (2-D), or tensors (3-D).**
- **This feature makes VSIPL a good match for multi-dimensional software synthesis.**

DIF-to-VSIPL Software Synthesis [5]

- **Given the multi-dimensional buffer size of a dataflow edge:**
  - **Create a VSIPL *block* with total buffer size.**
  - **Create two VSIPL *views* (vector, matrix, or tensor based on the dimensionality) associated with the block for source and sink VSIPL functions.**
  - **The *length* attributes of the views are decided by the multi-dimensional production and consumption rates.**
  - **The *stride* attributes are determined by the m-D buffer space.**
  - **The *offset* attributes are adjusted between VSIPL function invocations based on the m-D looped schedule and the m-D token transfer rates.**

MDSDF Modeling of 2DDWT



DIF-to-VSIPL Demonstration

```
mdsdf dwt256h {
   topology { nodes = FRK, SEH, SEL, HP, LP, …;
              edges = e1(FRK,SEH), e2(SEH,HP), …; }
   interface { inputs = in:FRK; outputs = out:APD; }
   production { e1 = [256]; e2 = [262]; …; }
   consumption { e1 = [256]; e2 = [262]; …; }
   parameter { … }
   …
   actor HP { computation = "vsip_convolve1d_d";
      conv1d = ConvObjH1; x = e2; y = e3; }
   …
}
…
mdsdf TDDWT {
   topology { nodes = IMGR, BUF1, DWT256V,
      DWT256H, …;
   edges = e1(IMGR,BUF1), …; }
   refinement { dwt256v = DWT256V; in : e2; out : e3; }
   production { e1 = [256,256]; e2 = [256,256]; …; }
   consumption { e1 = [256,256]; e4 = [256,256]; …; }
   …
}
```

```
#include <vsip.h>
void dwt256v(vsip_mview_d*, vsip_mview_d*);
…
int main(int argc, char* argv[]) {
   vsip_block_d *e1 = vsip_blockcreate_d(65536, 0);
   vsip_mview_d* e1_rv = vsip_mbind_d(e1,0,256,256,…);
   vsip_mview_d* e1_wv = vsip_mbind_d(e1,0,256,256,…);
   …
   for (i1d0=0; i1d0<256; i1d0++) {
      for (i1d1=0; i1d1<1; i1d1++) {
         vsip_mputoffset_d(e2_rv,(i1d1*256)*256+(i1d0*1));
         vsip_mputoffset_d(e3_wv,(i1d1*256)*256+(i1d0*1));
         dwt256v(e2_rv, e3_wv);
      }
   }
   for (i1d0=0; i1d0<1; i1d0++) {
      for (i1d1=0; i1d1<256; i1d1++) {
         vsip_mputoffset_d(e3_rv,(i1d1*1)*256+(i1d0*256));
         vsip_mputoffset_d(e4_wv,(i1d1*1)*256+(i1d0*256));
         dwt256h(e3_rv, e4_wv);
      }
   }
…
```

Current Status

- **DIF is being developed in the University of Maryland DSPCAD Research Group.**
- **DIF is being evaluated and used by a number of research partners.**
- **A general public release of DIF is being planned for the near future.**

References

1. J. Eker, J. W. Janneck, E. A. Lee, J. Liu, X. Liu, J. Ludvig, S. Neuendorffer, S. Sachs, and Y. Xiong, "Taming heterogeneity - the Ptolemy approach," *Proceedings of the IEEE*, vol. 91, no. 1, Jan. 2003.
2. C. Hsu and S. S. Bhattacharyya, "Dataflow Interchange Format Version 0.2," Technical Report, UMIACS-TR-2004-66, Institute for Advanced Computer Studies, University of Maryland at College Park, November 2004.
3. C. Hsu and S. S. Bhattacharyya, "Porting DSP applications across design tools using the dataflow interchange format," In *Proceedings of the International Workshop on Rapid System Prototyping*, Montreal, Canada, June 2005.
4. C. Hsu, M. Ko, and S. S. Bhattacharyya, "Software synthesis from the dataflow interchange format," In *Proceedings of the International Workshop on Software and Compilers for Embedded Systems*, Dallas, Texas, September 2005.
5. C. Hsu and S. S. Bhattacharyya, "Integrating VSIPL support in the Dataflow Interchange Format," In *Proceedings of the Annual Workshop on High Performance Embedded Computing*, Lexington, Massachusetts, September 2005.
6. R. Janka, R. Judd, J. Lebak, M. Richards, and D. Campbell, "VSIPL: An object-based open standard API for vector, signal, and image processing," In *Proceedings of the 2001 IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol.2, pp.949–952.
7. E. A. Lee and D. G. Messerschmitt, "Synchronous dataflow," *Proceedings of the IEEE*, 75(9):1235-1245, September 1987.
8. P. K. Murthy and E. A. Lee, "Multidimensional synchronous dataflow," *IEEE Transactions on Signal Processing*, vol. 50, no. 8, pp. 2064-2079, August 2002.
9. C. B. Robbins, "Autocoding toolset software tools for automatic generation of parallel application software," Technical report, Management Communications and Control, Inc., 2002.

Thank You