Decidable Dataflow Graphs

Soonhoi Ha
Seoul National University, KOREA
Outline

- Introduction: “Decidable” Dataflow Graphs
- SDF (Synchronous Data Flow) Graph
- Software Synthesis from SDF Graphs

- CSDF (Cyclo-Static Data Flow) Graph
- Other SDF Variants
  - FRDF (Fractional Rate Data Flow) Graph
  - SPDF (Synchronous Piggybacked Data Flow) Graph
  - SSDF (Scalable SDF) Graph

- MDSDF (Multi-Dimensional Synchronous Data Flow) Graph
- WSDF (Windowed Synchronous Data Flow) Graph
System Behavior Specification

- **Wish list**
  - User friendliness
  - Executable / simulatable
  - Implementation Independence
  - Design validation / error detection capability
  - Design Maintenance and collaboration
  - Well defined path to synthesis

- **Language-based approach vs model-based approach**
  - Language based approach is **NOT** good for
    - Parallel implementation
    - Design validation: simulation-based verification
    - Code reuse and maintenance
Actor-Model

- Model the system behavior as a composition of active modules (actors)
  - Express concurrency naturally
- Separate computation and communication
- Implementation independent
  - Can explore wide design space
- Design reuse
- Easy to understand the system behavior
Well-known Actor Models

- **Dataflow model**
- **Discrete-event model**
  - SystemC model
- **Concurrent process models**
  - CSP, Kahn process networks, Dataflow process network
- **SIMULINK model**

- **Models differ from each other**
  - When is a node “fireable” (executable)
  - What is the semantics of channel
Comparison: H.264 encoder

- **Language-based specification**
  - Deep nested function calls
  - Shared variables

```c
main()
encode_one_frame()

encode_one_slice() {
  ...
  start_macroblock();
  encode_one_macroblock();
  write_on_macroblock();
}

encode_one_macroblock() {
  ...
  Mode_Decision_for_Intra4x4Macroblock();
  ...
  Mode_Decision_for_8x8IntraBlocks();
  ...
  Mode_Decision_for_4x4IntraBlocks() {
    intrapred_luma();
    ...
    dct_luma();
    ...
  }
```

- DCT
- Quantization
- CofAcGen
- Dequantization
- IDCT
Comparison – cont’d

- **Actor-model specification**

```
Read Frame → Macroblock Initialize → I-Frame prediction → Prediction Type selection → I-frame Encoding
                          |                                               |                        → CAVLC
                          v                                               v                        → Write Frame
P-Frame prediction → P-frame Encoding
                          |                                               |                        → Deblocking Filter
                          v                                               v                        → "delay: initial sample"
```

"Much easier to understand!"
Dataflow (Graph) Model

- **A directed graph**
  - A node represents computation (or function) and an arc represents data channel

- **No shared variable between nodes: no side effect**
  - nodes communicate each other through arcs

- **Data-driven execution**
  - A node is fireable (can start execution) when the required number of data samples exist on the incoming arcs.

- **Suitable for signal processing applications or stream-based applications**

- **Node granularity may vary**
  - Fine grain (single operation) vs. coarse grain (function)

- **An example**

```
Read ➔ Filter ➔ Play
    
    Store ➔ Filter
```
Scheduling

- **Scheduling** is to determine where and when to execute the nodes on a given target platform
  - Issues
    - **resource requirement**: Can it run without buffer overflow? How large buffer should be assigned for each arc?
    - **correctness**: Is it free of deadlock?
    - **performance**: Can we estimate the performance on a multi-core platform?
  - Two approaches
    - Dynamic scheduling vs. Static scheduling
  - Usually assume that the graph will be repeatedly executed

- There are numerous ways of scheduling a dataflow graph
Decidable Dataflow Graphs

- **“Decidable” Dataflow Graph**
  - Dataflow graph with restricted semantics so that scheduling decision can be made at compile-time
  - We can “decide” the possibility of buffer overflow or deadlock before running the program
    - If a graph is deadlocked, no static schedule can be made.
    - Suppose we execute a graph infinitely according to the static schedule. A correct graph will have NO buffer overflow on all arcs.
  - Static analysis helps design validation in an early stage – very desirable feature
  - (But, the graph still may be scheduled dynamically)
**SDF (Synchronous Data Flow)**

- **Non-uniform sample rates are allowed**
  - The number of samples to consume (or produce) on an arc can be greater than 1: we call this number as input (or output) sample rate.
    - Useful to describe multi-rate DSP algorithms
  - The sample rates are constant at run-time

- **A dataflow model with strict firing condition**
  - A node is fireable only when all input arcs have no fewer samples than the associated input sample rates

- **We can determine the static schedule of the graph**
Example: H.263 Encoder

- **SDF specification**

![Diagram of H.263 Encoder process]

Delay - initial sample
Homogeneous v.s. Synchronous

- **example: (10-point) FFT block**

  - **homogeneous data-flow:** consume 1 sample per arc
    - 10 firings of FFT block to perform 10-point FFT
    - do nothing with first 9 samples inside the FFT block
    - hidden pipelining to produce 1 sample at a time

  - **synchronous data-flow:** consumes 10 samples at once and produce 10 samples at once
    - 1 firing of FFT block
    - natural representation
Two Implementations of Multi-rate

- **Multiple clocking**
  - dynamic scheduling of A: one firing of B produces 1 sample repeat (A-B-C-skip-B-C-skip-B-C-skip-B-C)
  - buffer size between B and C is 1

- **Multiple Samples (ex: Ptolemy/PeaCE)**
  - produce 4 samples at once A-B-4(C): buffer size on BC is 4
Static Scheduling

- **Ratio of node repetition counts should be maintained**
  - Repetition counts of A, B: \( r(A), r(B) \)
  - A produces (\( \text{prod}(A) = 2 \)) samples and B consumes (\( \text{cons}(B) = 3 \)) samples per execution

  - **Balance Equation:** \( r(A) \text{prod}(A) = r(B) \text{cons}(B) \)

  - \( r(A) : r(B) = \text{cons}(B) : \text{prod}(A) = 3:2 \)

  - Then, \( r(B) : r(C) = ? \)
    \( r(A):r(B):r(C)=? \)

- **Iteration period:** the minimum cycle that satisfies the ratio of node repetition counts
PASS: Periodic Admissible Sequential Schedule

- **periodic schedule**: repetitively applying the same program on an infinite stream of data
- **topology matrix**

\[
T = \begin{bmatrix}
c & -e & 0 \\
d & 0 & -f \\
0 & i & -g \\
\end{bmatrix}
\]

- **PASS: periodic ASS**
  - \(<예>\) \{A,B,C,C\} but not \{B,A,C,C\} or \{A,B,C\}
  - A = B, A = 2C, B = 2C
Snapshot of PASS

- **FIFO queue on each arc: buffer size** $b(n)$
  - delay: initial token (or sample)
- **Invocation vector** $v(n) \rightarrow v(n) = [1,0,0]^T, [0,1,0]^T, \text{or } [0,0,1]^T$
  - $b(n+1) = b(n) + T v(n)$
- **Initial buffer state** $b(0): b(0) = [1,3,2]$
- **Buffer size trace**
  - $b(1) = b(0) + T v(A) = [2,5,2]$
  - $b(2) = b(1) + T v(B) = [1,5,4]$
  - $b(3) = b(2) + T v(C) = [1,4,3]$
  - $b(4) = b(3) + T v(C) = [1,3,2] = b(0)$
Existance of PASS

- admissible sequential schedule: nonempty ordered list of nodes
  - buffer size should be non-negative and bounded
  - each node must appear at least once
- rank(T) = s - 1 (s: number of blocks in the graph)
  - for connected graph, rank(T) >= s-1, i.e. rank(T) = s-1 or s. < proof> tree graph: rank(T) = s-1

- Proof
  - b(p) = b(0) + T (v(0)+v(1)+ ... + v(p-1)) = b(0) + Tq: period p
  - b(np) = b(0) + nTq
  - buffer should be bounded -> Tq = 0 and q != 0
  - rank (T) < s -> rank(T) = s-1
PASS Construction

- **Invocation vector q: \( Tq = 0 \)**
  - positive integer vector exists
  - \( q \) is in the null space of \( T \) whose dimension is 1.
  - We may choose the minimum period \( |q| \) by choosing the smallest possible integer vector \( q \)
  - If no \( q \) is found, the graph is not consistent: buffer overflow will occur

- **PASS construction (class S algorithm) for sequential execution**
  - Given \( q \) \( (Tq = 0) \) and initial buffer size \( b(0) \),
  - schedule a runnable node \( k \) if it has not been run \( q_k \) times.
  - stop when no node is runnable
  - if all nodes are scheduled as many times as \( q \) vector indicates, done. Else, the graph is deadlocked
Syntax check

- **Sample rate inconsistency**
  - Some arc accumulates tokens without bound
- **Deadlock**
  - Graph cannot start its execution
**SDF Scheduling**

- **repetition counts**
  - A: 3  
  - B: 2  
  - C: 4

- **Valid schedules are not unique because the graph describes the partial order (true dependency) between blocks**
  - (ex) AABCCABCC – schedule for minimum buffer size
  - (3A)(2(B(2C))) – schedule with loop structure: **looped schedule**

- **What is the best schedule?**
  - Depends on the design objectives
  - (informal) programming describes just one schedule – no guarantee of optimality!
Looped Schedule

- **Looped schedule: Σ2 and Σ3**
  - Allows compact representation of the schedule while sacrificing the buffer requirement
  - **Single appearance (SA) schedule**: Every node appears exactly once in the schedule – minimum code memory
  - Σ3 is a Flat SA schedule that has no nested loop

Periodic schedules
- Σ1: ADADBBCCADDBCC
- Σ2: 3(AD)2(B2(C))
- Σ3: 3A3D2B4C
Exercise 1

- Graph (a) is deadlocked. To avoid deadlock, we insert initial tokens on the arc between C and A. Find the minimum number of samples to avoid deadlock.
Exercise 2

- **Answer the followings. (numbers on arcs indicate the number of samples consumed/produces per node execution)**
  - (a) Find the smallest integers of $x$, $y$ to make the SDF graph consistent
  - (b) With the answers of (a), compute the repetition counts of all nodes in a single iteration
  - (c) Find an optimal schedule with minimum buffer requirement and the required buffer size of all arcs

![Diagram of SDF graph with nodes A, B, C, D and arcs labeled with numbers 1, 2, 3, 1, 2, y, x, and x, y]
Software Synthesis from SDF

- **Automatic code generation from data flow graph**
  - SDF semantics should be preserved - “refinement”
  - The kernel code of a block is already optimized in the library
  - Determine the schedule and coding style
  - Codes are generated according to the scheduled sequence

- **Key question**
  - Can we achieve the similar code quality as manually optimized code in terms of performance and memory requirement?
Style of Synthesized Code

- Inlined code
- Function calls
- Switch-case

Concurrent processes:
- A, B, and C are generated as threads that can be scheduled dynamically or statically
Memory Requirement

- **Code memory: depends on coding style**
  - Inlined-style, Switch-style
    - SAS (Single Appearance Schedule) is preferred.
    - In case of no-SAS, define a function in the body.
    - Good for fine granule node
  - Code sharing is helpful if the block context is smaller than the code size

- **Data (buffer) memory: depends on schedule & buffer sharing**
Buffer Memory Requirement

- **<example>**
  - **< Schedule 1 >** 3(ABC)4(D)
    - minimum code size
    - buffer size: 3 + 1 + 12 = 16

  - **< Schedule 2 >** (3A)(3B)(3C)(4D)
    - minimum code size (inlined style)
    - buffer size: 9 + 3 + 12 = 24
    - buffer sharing: 3 + max (9,12) = 15

  - **< Schedule 3 >** 3(ABCD)D
    - code overhead
    - buffer size: 3 + 1 + 6 = 10
Efforts to Reduce Memory Space

- **Single Appearance Schedule (SAS):** APGAN, RPMC
  - [by Bhattacharyya et. al.] in Ptolemy Group
  - SAS guarantees the minimum code size (without code sharing)
  - APGAN, RPMC: heuristics to find data minimized SAS schedule

- **ILP formulation for data memory minimization**
  - [by Ritz et. al.] in Meyr Group
  - Flat single appearance schedule + sharing of data buffer

- **Rate optimal compile time schedule**
  - [by Govindarajan et. al.] in Gao Group
  - Tried to minimize the buffer requirement using linear programming
Buffer Memory Lower Bound

- For single appearance schedule,
  \( a = \text{produced}(e), b = \text{consumed}(e), c = \gcd\{a,b\}, d = \text{delay } (e) \)

\[
BMLB(e) = \begin{cases} 
(\eta(e) + d) & \text{if } d < \eta(e) \\
 d & \text{if } d \geq \eta(e)
\end{cases}, \quad \text{where } \eta(e) = \frac{ab}{c}
\]

- For any schedule

\[
LB(e) = \begin{cases} 
 a + b - c + (d \mod c) & \text{if } d < a + b - c \\
 d & \text{otherwise}
\end{cases}
\]
Optimal SAS Schedule for Chained Graph

- Find an optimal factoring transformation
- No buffer sharing is considered
- Dynamic programming: $O(n^3)$

- $b[i,j] = \min\{(b[i,k] + b[k+1,j] + C[k] | (l <= k < j))\}$
  $\{B,C,D\}: (3(B)(C))(2D): 1+6 = 7$, $(3B)((3C)(2D)): 3+6 = 9$
- $\{A,B,C,D\}: 9A4\{B,C,D\} = (9A)((3B)(C))(2D)): 36+7 = 43$
  $3\{A,B\}4\{C,D\} = (3(3A)(4B))(4(3C)(2D)): 12 + 6 + 12 = 30$
  $3\{A,B,C\}8D = (3(3A)(4(B)(C)))(8D): 13+24 = 37$
Extensions

- **Chain-structured graph with delays**
- **Well-ordered graph: partial order is a total order**
  - topological sort gives a single list to traverse.
- **Acyclic SDF graph**
  - the number of topological sort is exponential.
  - It is proved that the problem of constructing buffer-optimal single appearance schedules for acyclic graphs with delays is NP complete
    - reduced to vertex cover problem
  - Without delays, open problem.
Multiprocessor Scheduling

- **Map the nodes to processors and determine the execution order on each processor**
  - Static mapping and dynamic scheduling
  - Static mapping and static scheduling
  - Dynamic mapping and scheduling

- **Objectives**
  - Minimize the throughput for stream-based applications
  - Minimize the latency for a single execution of the graph
  - Minimize the buffer requirement under a given performance constraint
APEG Translation

- Translate a SDF graph to an APEG (Acyclic Precedence Expanded Graph), shortly EG (Expanded Graph)
Multiprocessor scheduling example

A1 -> A2 -> B1
A3 -> B2

D1 -> A1
D2 -> A2
D3 -> A3

B1 -> C1
B2 -> C2

C1 -> C3
C2 -> C4

P1: A1  A2  B1  C1  C2
P2: D1  A3  B2  C3  C4
P3: D2  D3

processor

time
Multiprocessor Scheduling Issues

- **Execution time estimation**
  - How to determine the execution time of each node statically?
  - How to consider the communication overhead?

- **Mapping**
  - Can we assign multiple instances of a node to multiple processors?
    - Overhead for synchronization and communication
    - Overhead for code storage

- **Scheduling**
  - Static scheduling pre-determines the execution order of nodes while dynamic scheduling determines the execution order at run-time

- **Others**
  - Graph unfolding
  - Objectives: throughput vs latency
SDF Limitations

- **Expression Capability**
  - SDF model cannot express varying sample rates -> CSDF, FRDF
  - SDF model does not allow shared memory (global states) between blocks due to side effect. -> SPDF
    - why? Memory update order may vary depending on the schedule.

- **Performance issues**
  - Reduce the memory access counts -> SSDF
  - Reduce the memory requirement -> CSDF, FRDF

- **Multi-dimensional extension**
  - MDsDF and WSDF
  - FRDF allows message-type token as a unit
Summary of Lecture 1 (SDF)

- **Dataflow Model** is a formal model of computation that has been successfully used for DSP algorithm specification in particular.

- SDF has a nice feature of static analyzability that determine the static schedule and resource requirement.

- But SDF has some weaknesses to be used for general purpose:
  - Restricted expression capability
  - No shared memory

- Multiprocessor scheduling of SDF graph has many issues to consider:
  - Open for further research
Outline

- Introduction: “Decidable” Dataflow Graphs
- SDF (Synchronous Data Flow) Graph
- Software Synthesis from SDF Graphs

- CSDF (Cyclo-Static Data Flow) Graph
- Other SDF Variants
  - FRDF (Fractional Rate Data Flow) Graph
  - SPDF (Synchronous Piggybacked Data Flow) Graph
  - SSDF (Scalable SDF) Graph

- MDSDF (Multi-Dimensional Synchronous Data Flow) Graph
- WSDF (Windowed Synchronous Data Flow) Graph
CSDF (Cyclo Static Data Flow)

- **Key extension**
  - Allow periodic sample rate change

- **For each node A in a CSDF graph**
  - Basic period $\tau(A) \in \{1, 2, \ldots\}$
    - Can specify the phases of operation
    - Input sampling rate: $\tau(A)$-tuple $\{C_{e,1}, C_{e,2}, \ldots, C_{e,\tau(A)}\}$
    - Output sampling rate: $\tau(A)$-tuple $\{P_{e,1}, P_{e,2}, \ldots, P_{e,\tau(A)}\}$

- **Example**

![Diagram showing SDF and CSDF examples](image-url)
Translation: CSDF to SDF

- SDF is a special case of CSDF: the period of all nodes are 1
- CSDF node A can be translated to SDF node A’
  - For each output arc e’
    - \( \text{delay}(e') = \text{delay}(e) \)
  - \( \text{prod}(e') = \sum_{i=1}^{T(A)} p_{e,i} \)
  - For each input arc e’,
    - \( \text{cons}(e') = \sum_{i=1}^{T(A)} C_{e,i} \)

- Sample rate consistency can be checked with the translated SDF
- The translated SDF graph can be deadlocked while the original CSDF graph is not
Delay-free Loop without Deadlock

- Even possible to form a deadlock-free feedback-loop without any delay on the arc

(a) CSDF version

CSDF cycle => schedulable

(b) SDF version

delay-free SDF cycle => deadlock
More Freedom for Composition

- **CSDF gives better chance of actor composition**

More freedom to make a composite actor

(a) CSDF version

(b) SDF version

Dead-lock!
Buffer Size Reduction

- CSDF requires smaller buffer size than SDF

(a) SDF version

Src. \[\begin{array}{c}
1 \\
\end{array}\] \rightarrow \text{Dist.} \[\begin{array}{c}
N \\
2N \\
N \\
1 \\
\end{array}\] \rightarrow \text{M1} \quad \text{Buffer size = 4N}

(b) CSDF version

Src. \[\begin{array}{c}
1 \\
{N,N} \\
\end{array}\] \rightarrow \text{Dist.} \[\begin{array}{c}
{N,N} \\
{N,0} \\
{0,N} \\
1 \\
\end{array}\] \rightarrow \text{M1} \quad \text{Buffer size = 3N}
FRDF (Fractional Rate Dataflow)

- **Motivation – H.263 encoder in SDF**
  - More memory requirement than reference C code
  - Separate buffers for frame data and macroblock data

<table>
<thead>
<tr>
<th>Reference Code (TMN 2.0)</th>
<th>SDF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buffer size</td>
<td></td>
</tr>
<tr>
<td>361KB</td>
<td>686KB</td>
</tr>
</tbody>
</table>

Schedule: \((\text{ME,D})99(\text{EN..})\)
Observation

- **ME(motion estimation) node in hand-optimized code**
  - need not produce the frame-size output at once
  - Generates output samples at the unit of macro block for short latency and minimizing buffer memory

```c
for(i=0; i<99; i++) {
    MotionEstimation(motion_vector, macroblock, currFrame, prevFrame);
    DCT(macroblock);
    Quantization(macroblock);
}
```
Fractional Rate Dataflow

- **Key extension**
  - Use message type tokens (for multi-dimensional applications)
  - Express the relationship between two different message types with the novel concept of “fractional rate”

- **Example**

  ![Diagram](image)

  Schedule: 99(ME, EN)

  The macro block is 1/99 of the video frame in its size. For each execution of the ME node, it consumes a macro-block from the input frame, and produces a macro-block output.
H.263 Encoder in FRDF

Reference Code | SDF  | FRDF |
---|---|---|
Buffer size | 361KB | 686KB | 225KB |
# frame size variables | 5 | 8 | 3 |
H.263 Encoder in CSDF

Buffer size: 291KB
Interpretation of Fractional Rate

- **Composite (Message) Type**

  - Output rate is integer
  - Output and input data have same access sequences
  - DDUDU is a valid schedule (SDF schedule)

- The consumer and the producer should have the same interpretation on the fraction. Otherwise, it is regarded as atomic type.

  - 3(D) 2(U)
Interpretation for Atomic Type

- **Statistical interpretation**

  \[ \frac{p}{q} = \text{p samples per q executions} \]

- **FRDF may require less buffer space!**

  \[
  \begin{array}{c}
  \text{A} \\
  \text{D} \\
  \text{B} \\
  \text{C}
  \end{array}
  \quad
  \begin{array}{c}
  \text{A} \\
  \text{D} \\
  \text{B} \\
  \text{C}
  \end{array}
  \]

  \(2(A) \text{ (Mux,B,C)}\)  \(2(A,\text{Mux}) \text{ (B,C)}\)
**Transformation from SDF to FRDF**

- **Equivalence Relationship between SDF and FRDF**

\[
\frac{p_i}{q_i} \quad \text{(FRDF)} \quad \Rightarrow \quad \frac{p_i}{q_i} \times Q \quad \text{(SDF)}
\]

Where \( Q \) is the I/O period = LCM(repetition periods of all ports)

![Diagram](image.png)
An FR-SDF (fractional rate extension of SDF) is consistent if the equivalent SDF graph is consistent.

Firing Condition of FRDF node
- If data-type is composite, there must be at least as large fraction of samples stored as the fractional input rate
- Otherwise, there must be at least as many samples stored as the numerator value of the fractional sample rate

atomic: $4(AB)2(C)$

composite: $2(2(AB)C)$
CD2DAT Example

- SDF representation requires 6+56+280 buffers

\[
7(7(3(FIR1)2(FIR2))8(FIR3))40(FIR4)
\]

- FRDF representation requires 6+4+40 buffers

\[
7(7(3(FIR1)2(FIR2))4(FIR3)))40(FIR4))
\]
CD2DAT Example: CSDF

- CSDF representation gives the same buffer size as FRDF
- But representation is much complicated
Exercise

- Answer the following questions for the FRDF graph below.

  - (a) Find the repetition counts of all nodes.
  - (b) Find an optimal schedule with minimum buffer requirement.
  - (c) Draw a CSDF graph with the same buffer requirement and the same schedule.
**Motivation:**
- Need of global states arises in many multimedia applications (frame-based application): MP3 decoder, OpenGL machine, etc

**MPEG-1 frame structure**
- Sync, system information
- Side information
- Scale factors
- Subband information
- block_type, global_gain, subblock_gain[], scale[]

**MPEG-1 decoder structure**
- Packet Decoder (PD)
- Huffman Decoding → Dequantize → Inverse MDCT → Synthesis Filterbank
- Use shared global states!
- State update (SU) request
Adhoc Approaches in SDF (1)

- To duplicate a state update (SU) request for each data sample
  - Redundant copy problem
  - Block reuse problem

- To group data samples as a unit to remove redundant copy
  - Large buffer problem
  - Block reuse problem
Ad-hoc Approaches in SDF (2)

- To bypass the intermediate nodes

- Synchronization problem: initial sample, multirate
- Redundant copy problem
- Still have the block reuse problem for node C
- Visibility problem in a hierarchical graph
Synchronous Piggybacked Dataflow

- Global structure for global states with limited access
- Each data sample is piggybacked with an SU request
- A block updates its internal parameters depending on applicability of SU request

<table>
<thead>
<tr>
<th>State name</th>
<th>1st iteration</th>
<th>2nd iteration</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;gain&quot;</td>
<td>10</td>
<td>20</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

PD → piggyback → A → B → C
Extensions to SDF

- **Global State Table (GST)**
  - maintain the outstanding values of global states

- **Piggybacking Block (PB)**
  - models the coupling of data samples and the pointers to the GST
  - single source of state update

<table>
<thead>
<tr>
<th>State name</th>
<th>1st iteration</th>
<th>2nd iteration</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;gain&quot;</td>
<td>10</td>
<td>20</td>
</tr>
<tr>
<td>....</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Diagram:

- **PD**
- **SU**
- **GST**
- **A**
- **B**
- **C**
- **Data**
- **piggyback**

Arrow connections and labels indicate state transitions and data flow.
SPDF Solution

- Solves the following problem
  - Block reusability, synchronization, buffer size from grouping
- Redundant copy problem remains to be unsolved!
- Overhead of extracting GST-related information from data sample!

Use “static scheduling” to remove these overhead

- **Period**: the repetition period of updating the states in terms of node’s execution
- **Offset**: the starting position of data samples to which the state update is coupled

```
<table>
<thead>
<tr>
<th>Node</th>
<th>Gain</th>
<th>Period</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>PD</td>
<td>20</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>PB</td>
<td>10</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
Scheduling Procedure

- Examine Piggyback star
- Identify stars with global state
- Starting from each piggyback star
  - 1) propagate \{ period, offset \} pair to the target star
  - 2) construct data structure for global states
  - 3) attach preamble for the target star

\[
\text{PD,SC,3(PB,A,B,(SU,C))}
\]

If (\text{count==offset}){
  \text{new_value = read_state_port();}
  \text{update_GST("gain",new_value);}
}
\text{data_out[]=data_in[];}
\text{if (++count >= period) count=0;}

{ // SU request code
  if (\text{count==offset})
    \text{gain=read_GST("gain");}
  \text{if (++count >= period) count=0;}
}
Experimental Results
Scalable-SDF (SSDF) Graph

- **SSDF**
  - Same SDF graph, but with non-unity blocking factor ($N_g$)
    - Repetition counts are multiplied by $N_g$
    - The blocking factor is determined by scheduler or optimizer

- **Motivation**
  - Trade-off between memory requirement and memory access overhead

### Example

<table>
<thead>
<tr>
<th></th>
<th>$r(A) = 3$</th>
<th>$r(B) = 1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N_g = 3$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$3AB$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>$r(A) = 9$</th>
<th>$r(B) = 3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$9A3B$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
SSDF Graph with Feedback Loop

- **For blocking factor \(N_g = 5\)**
  - If delay = 1, schedule: 5A 5(EBCD) 5F
  - If delay = 2, schedule: 5A 2(2E2B2C2D) EBCD 5F

- **Scheduling procedure**
  - First cluster strongly-connected components (BCDE)
  - Schedule the graph considering the clusters as a unit
    - 5A 5(cluster) 5F
  - Schedule the cluster inside
    - Make a blocking schedule as much as possible
To overcome the limitations of SDF, several extended models have been proposed: CSDF, FRDF, SPDF, SSDF, etc.

- **CSDF Model**
  - Expresses the cyclically varying sample rates
  - Can reduce the buffer size requirement

- **FRDF Model**
  - Use message type tokens and express the relationship between two different message types
  - For atomic type data, it can reduce the buffer size requirement

- **SPDF Model**
  - Allows shared states for efficient code generation

- **SSDF Model**
  - Reduce the memory access frequency by increasing the blocking factor
Outline

- Introduction: “Decidable” Dataflow Graphs
- SDF (Synchronous Data Flow) Graph
- Software Synthesis from SDF Graphs
- CSDF (Cyclo-Static Data Flow) Graph
- Some SDF Variants
  - FRDF (Fractional Rate Data Flow) Graph
  - SPDF (Synchronous Piggybacked Data Flow) Graph
  - SSDF (Scalable SDF) Graph
- MDSDF (Multi-Dimensional Synchronous Data Flow) Graph
- WSDF (Windowed Synchronous Data Flow) Graph
MDSDF Basics

- **Motivation**
  - Original SDF uses 1-D FIFO channel for communication
    - Awkward and difficult to present 2-D or 3-D data samples that are widely used for image/video processing

- **Multi-dimensional (vector-valued) samples**

  ![](diagram.png)

  - Array arc buffer and an execution scenario
    - B1
    - B2
    - B3
    - A3
    - A2
    - A1
MDSDF Balance Equation

- **Multi-dimensional (vector-valued) samples**

- **Repetition matrix and balance equation**

\[
R = \begin{pmatrix}
  r_{A,1} & r_{B,1} \\
  r_{A,2} & r_{B,2}
\end{pmatrix}
\]

\[
r_{A,1} \times 3 = r_{B,1} \times 1, \quad r_{A,2} \times 1 = r_{B,2} \times 3 \quad \rightarrow \quad (r_{A,1}, r_{A,2}) = (1, 3) \]

\[
(r_{B,1}, r_{B,2}) = (3, 1)
\]

- **Delay is also multidimensional**
**Examples**

- **Image processing**
  
  - A (frame (40,48), macroblock (8,8)) → DCT
  
  \[
  (r_{A,1}, r_{A,2}) = (1,1) \\
  (r_{DCT,1}, r_{DCT,2}) = (5,6)
  \]

- **Mixing dimensionality**
  
  - Missing dimension becomes 1

\[
\begin{align*}
\text{A} & \quad (M,N) \quad (K,1) \quad \text{B} \\
\text{A} & \quad (M,N) \quad K \quad \text{B}
\end{align*}
\]
Arbitrary Sampling

- Sampling matrix V
  - Sample points define a lattice LAT(V) $t = Vn$
    - Points n are called renumbered points of LAT(V)
  - Fundamental parallelepiped: FPD(V) $\rightarrow$ area = $|\det(V)|$

- Rectangular sampling example

$$V = \begin{bmatrix} 2 & 0 \\ 0 & 3 \end{bmatrix}$$

P = $\text{diag}(7,6)$

renumbered samples
Arbitrary Sampling (Example)

- Nonrectangular sampling example with

  \[ V = \begin{bmatrix} 2 & 2 \\ 2 & -2 \end{bmatrix} \]

- (1) Find LAT(V) with the support of \( P = \text{diag}(7, 6) \)
- (2) Find the FPD(V): area = \( |\det(V)| = 8 \)
- (3) Find the renumbered points
MD Downsampler

- **Decimation matrix M**: nonsingular integral matrix
  - Sample points define LAT(VM): output sampling matrix $V_0=VM$
- **Decimation ratio** = $|\det(M)|$

\[
\begin{align*}
V &= \begin{bmatrix} 2 & 0 \\ 0 & 3 \end{bmatrix}, \\
M &= \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, \\
VM &= \begin{bmatrix} 2 & 2 \\ 3 & -3 \end{bmatrix}
\end{align*}
\]
MD upsampler

- **Expansion matrix L**: nonsingular integral matrix
  - Sample points define $\text{LAT}(V L^{-1})$: output sampling matrix $V_o = V L^{-1}$
  - Expanded samples are set to 0

$$V = \begin{bmatrix} 2 & 2 \\ 2 & -2 \end{bmatrix}$$

$$L = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, \quad L^{-1} = \begin{bmatrix} 0.5 & 0.5 \\ 0.5 & -0.5 \end{bmatrix}, \quad V L^{-1} = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$$
Example

- **Nonrectangular expansion**

  \[
  L = \begin{bmatrix} 1 & 1 \\ 2 & -2 \end{bmatrix}, \quad L^{-1} = \begin{bmatrix} 0.5 & 0.25 \\ 0.5 & -0.25 \end{bmatrix}
  \]

- **Renumbered samples**
Support matrix

- **Support matrix** $W$: region of the rectangular lattice where samples are placed

\[ W = \begin{bmatrix} 6 & 0 \\ 0 & 6 \end{bmatrix} \]

\[ M = \begin{bmatrix} 1 & 1 \\ 2 & -2 \end{bmatrix} \]

\[ W_f = M^{-1}W = \begin{bmatrix} 0.5 & 0.25 & 6 & 0 \\ 0.5 & -0.25 & 0 & 6 \end{bmatrix} = \begin{bmatrix} 3 & 1.5 \\ 3 & -1.5 \end{bmatrix} \]
Example of Arbitrary Sampling

- Various factorization possibilities of downsampling
  - There is always a factorization that ensures that the decimator produces (1,1) for all invocations

\[ M = \begin{bmatrix} 1 & 1 \\ 2 & -2 \end{bmatrix} \]
Example of Arbitrary Sampling

- Ordering of data into a 5x2 rectangle inside FPD(L)

\[ L = \begin{bmatrix} 2 & -2 \\ 3 & 2 \end{bmatrix} \quad M = \begin{bmatrix} 1 & 1 \\ 2 & -2 \end{bmatrix} \]

<table>
<thead>
<tr>
<th>original</th>
<th>(0,0)</th>
<th>(0,1)</th>
<th>(0,2)</th>
<th>(1,2)</th>
<th>(1,3)</th>
<th>(-1,1)</th>
<th>(-1,2)</th>
<th>(-1,3)</th>
<th>(0.3)</th>
<th>(0.4)</th>
</tr>
</thead>
<tbody>
<tr>
<td>renumbered</td>
<td>(0,0)</td>
<td>(1,0)</td>
<td>(2,0)</td>
<td>(3,0)</td>
<td>(4,0)</td>
<td>(0,1)</td>
<td>(1,1)</td>
<td>(2,1)</td>
<td>(3,1)</td>
<td>(4,1)</td>
</tr>
</tbody>
</table>
Two downsampling factorizations

\[
L = \begin{bmatrix} 2 & -2 \\ 3 & 2 \end{bmatrix} \quad M = \begin{bmatrix} 1 & 1 \\ 2 & -2 \end{bmatrix}
\]

2x2

1x4
Example of Arbitrary Sampling

\[ L = \begin{bmatrix} 2 & -2 \\ 3 & 2 \end{bmatrix} \quad M = \begin{bmatrix} 1 & 1 \\ 2 & -2 \end{bmatrix} \]

- Balance equations with 1x4 downsampling factorization

\[ 3r_{S,1} = r_{L,1}, \quad 3r_{S,2} = r_{L,2}, \quad 5r_{L,1} = r_{M,1}, \quad 2r_{L,2} = 4r_{M,2}, \quad r_{M,1} = r_{T,1}, \quad r_{M,2} = r_{T,2} \]

\[ r_{S,1} = 1, \quad r_{S,2} = 2, \quad r_{L,1} = 3, \quad r_{L,2} = 6, \quad r_{M,1} = 15, \quad r_{M,2} = 3, \quad r_{T,1} = 15, \quad r_{T,2} = 3 \]
Initial Tokens on Arcs

- Delays mean translations of buffers along the vectors in the basis for the sampling lattice

\[ V = \begin{bmatrix} 2 & 1 \\ 0 & 1 \end{bmatrix} \]
WSDF (Windowed SDF): Motivation

- **Motivation**
  - Sliding window algorithms are popular in image/video processing
    - Noise filtering, edge detection, medical imaging, etc
  - Common characteristics of sliding window algorithms
    - Windows do overlap when successive output tokens are computed
    - The borders of the images need special treatment
    - Reading of tokens in a window should not be destructive
    - Samples are multi-dimensional
WSDF Specification

- **WSDF graph** $G=(V,E,p,v,c,\Delta c,u,d,b_u,b_l)$
  - $p$: (effective) token produced by the source node
  - $v$: virtual token that forms a unit data elements belonging together and model for instance images or blocks.
  - $c$: token consumed by the sink node
  - $\Delta c$: defines the window movement
  - $b_u(i,j)$: border extension up $(i)$ and left $(j)$
  - $b_l(i,j)$: border extension down $(i)$ and right $(j)$
  - $d$: delay
  - $u$: virtual token union
WSDF Specification (2)

\[ p = (1, 2) \]
\[ v = (2, 3) \]
\[ d = (d_1, d_2) \]
\[ b^u = (1, 1) \]
\[ b^l = (1, 2) \]
\[ c = (3, 5) \]
\[ \Delta c = (1, 1) \]

\& u = (1, 2)

virtual token union \( u = (1, 2) \)
WSDF Balance Equation (1)

- For each arc and for the associated sink actor,
  - For each dimension, compute the minimum number of times actors have to be invoked to ensure that complete virtual tokens are consumed

\[ w_e = \frac{u_e v_e + b^u_e + b^l_e - c_e}{\Delta c_e} + 1 \]

\[ w_{e,1} = \frac{1 \times 2 + 1 + 1 - 3}{1} + 1 = 2, \quad w_{e,2} = \frac{2 \times 3 + 2 + 1 - 5}{1} + 1 = 4 \]
WSDF Balance Equation (2)

- For each arc, the source node and the sink node exchange complete virtual tokens to return to the initial state
  - Let $N_A$ be the LCM of $w_e$ among the set of input edges of actor $A$
    - If there is no input edge, $N_A$ becomes 1

$$N_{src(e)} p_{e} q_{src(e)} = \frac{N_{snk(e)}}{w_e} v_{e} u_{e} q_{snk(e)}$$

$1 \times 1 \times q_{A,1} = \frac{2}{2} \times 1 \times q_{B,1}, \quad 1 \times 2 \times q_{A,2} = \frac{4}{3} \times 2 \times q_{B,1}$

$$Q = \begin{pmatrix} q_{A,1} & q_{B,1} \\ q_{A,2} & q_{B,2} \end{pmatrix} = \begin{pmatrix} 2 & 1 \\ 3 & 1 \end{pmatrix}$$

- Repetition vector $r = Nq$

$$\begin{pmatrix} r_{A,1} \\ r_{B,1} \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 2 \end{pmatrix} \begin{pmatrix} 2 \\ 1 \end{pmatrix} = \begin{pmatrix} 2 \\ 2 \end{pmatrix}, \quad \begin{pmatrix} r_{A,2} \\ r_{B,2} \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 4 \end{pmatrix} \begin{pmatrix} 3 \\ 1 \end{pmatrix} = \begin{pmatrix} 3 \\ 4 \end{pmatrix}$$

$r_A = (2, 3), \quad r_B = (2, 4)$
Example

- Find the repetition vectors of two actors

\[ p = (1,1), \quad v = (7,4), \quad b^u = (1,1), \quad b^l = (1,1), \quad c = (3,3), \quad \Delta c = (1,1) \]

\[ q_1 = \begin{pmatrix} 1 & 0 \\ 0 & 7 \end{pmatrix}, \quad q_2 = \begin{pmatrix} 7 & 1 \\ 0 & 4 \end{pmatrix} \]

\[ w_{e,1} = \frac{1\times7 + 1 + 1 - 3}{1} + 1 = 7, \quad w_{e,2} = \frac{1\times4 + 1 + 1 - 3}{1} + 1 = 4 \]

\[ 1 \times 1 \times q_{A,1} = \frac{7}{7} \times 1 \times q_{B,1}, \quad 1 \times 1 \times q_{A,2} = \frac{4}{4} \times 1 \times q_{B,1} \]

\[ Q = \begin{pmatrix} q_{A,1} & q_{B,1} \\ q_{A,2} & q_{B,2} \end{pmatrix} = \begin{pmatrix} 7 & 1 \\ 4 & 1 \end{pmatrix} \]

\[ \begin{pmatrix} r_{A,1} \\ r_{B,1} \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 7 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & 4 \end{pmatrix} \begin{pmatrix} 7 \\ 1 \end{pmatrix} = \begin{pmatrix} 4 \\ 4 \end{pmatrix} \]

\[ r_A = (7,4), \quad r_B = (7,4) \]
Summary of Lecture 3

- Multi-dimensional signal processing applications are becoming popular.
- MDSDF extends the SDF model with multi-dimensional samples, preserving the decidability properties of the SDF model.
- WSDF further extends the MDSDF for sliding window algorithms that are popular in image/video signal processing applications.
- FRDF considers a message type token as a unit sample for multi-dimensional signal processing.
References

- **SDF**

- **CSDF**

- **SPDF**
FRDF


SSDF


MDSDF


WSDF

- Contact information
  - Prof. Soonhoi Ha ([sha@snu.ac.kr](mailto:sha@snu.ac.kr))
    CAP (codesign and parallel processing) laboratory
    School of Computer Science and Engineering
    Seoul National University, Korea
    [http://peace.snu.ac.kr/sha](http://peace.snu.ac.kr/sha)

Thank you!