Scheduling Synchronous Dataflow Graphs

Saman Amarasinghe and William Thies
Massachusetts Institute of Technology

PACT 2003
September 27, 2003
## Schedule

<table>
<thead>
<tr>
<th>Time</th>
<th>Session</th>
</tr>
</thead>
<tbody>
<tr>
<td>1:30-1:40</td>
<td>Overview (Saman)</td>
</tr>
<tr>
<td>1:40-2:20</td>
<td>Stream Architectures (Saman)</td>
</tr>
<tr>
<td>2:20-3:00</td>
<td>Stream Languages (Bill)</td>
</tr>
<tr>
<td>3:00-3:30</td>
<td>Break</td>
</tr>
<tr>
<td>3:30-3:55</td>
<td>Stream Compilers (Saman)</td>
</tr>
<tr>
<td>3:55-4:20</td>
<td>Domain-specific Optimizations (Saman)</td>
</tr>
<tr>
<td>4:20-5:00</td>
<td>Scheduling Algorithms (Bill)</td>
</tr>
</tbody>
</table>
Outline

- Introduction to Scheduling
  - Finding a Steady State
  - Finding a Schedule
  - Scheduling Tradeoffs

- Phased Scheduling Algorithm
  - Code Size / Buffer Size
  - Hierarchical scheduling
  - Results
Synchronous Dataflow (SDF)

- Consists of Filters and Channels
- Filters perform computation
  - Atomic execution step
  - Number of items produced / consumed on each firing is constant and known at compile time
- Channels act as FIFO queues for data between Filters
- For SDF, can statically determine:
  - Schedule of node firings
  - Buffer sizes
  - Deadlock conditions
- As we saw before, there are many generalizations
The Scheduling Problem

- Find a legal order in which filters can be executed
  - Nodes only fire when their inputs are ready
- Manage mismatched rates between filters
- Minimize data buffered up in channels between filters
- Minimize latency of data processing
Scheduling – Steady State

- Every valid stream graph has a Steady State
- Steady State does not change amount of data buffered between components
- Steady State can be executed repeatedly forever without growing buffers
Steady State Example

- 3:2 Rate Converter
  - First filter (A) upsamples by factor of 3
  - Second filter (B) downsamples by factor of 2
Steady State Example

- A executes 2 times
  - pushes $2 \times 3 = 6$ items
- B executes 3 times
  - pops $3 \times 2 = 6$ items
- Number of data items stored between Filters does not change

$\begin{align*}
\text{push} &= 3 \\
\text{push} &= 1 \\
\text{pop} &= 2 \\
\text{pop} &= 1 \\
\text{push} &= 3 \\
\text{pop} &= 1
\end{align*}$
Computing the Steady State

- **Balance equations**
  - For each edge (src, dst):
    \[ n(src) \times push(src) = n(dst) \times pop(dst) \]
  - Example:
Computing the Steady State

- **Balance equations**
  - For each edge \((\text{src}, \text{dst})\):
    \[ n(\text{src}) \times \text{push(\text{src})} = n(\text{dst}) \times \text{pop(\text{dst})} \]
  - **Example:**

```
pop = 0
X
push=2
push=2

pop = 1
A
push = 3

pop = 2
B
push = 1

pop=2
pop=3
Y
push = 0
```
Computing the Steady State

- Balance equations
  - For each edge \((\text{src}, \text{dst})\):
    \[ n(\text{src}) \times \text{push}(\text{src}) = n(\text{dst}) \times \text{pop}(\text{dst}) \]
  - Example:
    \[ n(X) \times 2 = n(Y) \times 2 \]
Computing the Steady State

- Balance equations
  - For each edge (src, dst):
    \[ n(\text{src}) \times \text{push}(\text{src}) = n(\text{dst}) \times \text{pop}(\text{dst}) \]
  - Example:
    \[ n(\text{X}) \times 2 = n(\text{Y}) \times 2 \]
    \[ n(\text{X}) \times 2 = n(\text{A}) \times 1 \]
Computing the Steady State

- **Balance equations**
  - For each edge \((\text{src, dst})\):
    \[ n(\text{src}) \times \text{push(\text{src})} = n(\text{dst}) \times \text{pop(\text{dst})} \]
  - **Example:**
    \[ n(X) \times 2 = n(Y) \times 2 \]
    \[ n(X) \times 2 = n(A) \times 1 \]
    \[ n(A) \times 3 = n(B) \times 2 \]
Computing the Steady State

- Balance equations
  - For each edge (src, dst):
    \[ n(\text{src}) \times \text{push}(\text{src}) = n(\text{dst}) \times \text{pop}(\text{dst}) \]
  - Example:
    \[ n(X) \times 2 = n(Y) \times 2 \]
    \[ n(X) \times 2 = n(A) \times 1 \]
    \[ n(A) \times 3 = n(B) \times 2 \]
    \[ n(B) \times 1 = n(Y) \times 3 \]
Computing the Steady State

- **Balance equations**
  - For each edge (src, dst):
    \[ n(\text{src}) \times \text{push}(\text{src}) = n(\text{dst}) \times \text{pop}(\text{dst}) \]
  - Example:
    \[ n(\text{X}) \times 2 = n(\text{Y}) \times 2 \]
    \[ n(\text{X}) \times 2 = n(\text{A}) \times 1 \]
    \[ n(\text{A}) \times 3 = n(\text{B}) \times 2 \]
    \[ n(\text{B}) \times 1 = n(\text{Y}) \times 3 \]

\[
\begin{bmatrix}
2 & 0 & 0 & -2 \\
2 & 0 & -1 & 0 \\
0 & 3 & 2 & 0 \\
0 & 0 & 1 & -3
\end{bmatrix}
\begin{bmatrix}
n(\text{X}) \\
n(\text{A}) \\
n(\text{B}) \\
n(\text{Y})
\end{bmatrix}
= 0
\]
Computing the Steady State

\[
\begin{bmatrix}
2 & 0 & 0 & -2 \\
2 & 0 & -1 & 0 \\
0 & 3 & 2 & 0 \\
0 & 0 & 1 & -3
\end{bmatrix}
\begin{bmatrix}
\mathbf{n}(X) \\
\mathbf{n}(A) \\
\mathbf{n}(B) \\
\mathbf{n}(Y)
\end{bmatrix} = \mathbf{0}
\]

Topology Matrix, \( \Gamma \)

- **Theorem (Lee ’86):**
  - A connected SDF graph with \( n \) actors has a periodic schedule iff its topology matrix \( \Gamma \) has rank \( n-1 \)
    - Rank > \( n-1 \) \( \Rightarrow \) no periodic schedule
    - Rank < \( n-1 \) \( \Rightarrow \) graph is not connected
  - If \( \Gamma \) has rank \( n-1 \) then there exists a unique smallest integer solution to \( \Gamma \mathbf{n} = 0 \)
Computing the Steady State

\[
\begin{bmatrix}
2 & 0 & 0 & -2 \\
2 & 0 & -1 & 0 \\
0 & 3 & 2 & 0 \\
0 & 0 & 1 & -3 \\
\end{bmatrix}
\begin{bmatrix}
n(X) \\
n(A) \\
n(B) \\
n(Y) \\
\end{bmatrix} = \begin{bmatrix}
\end{bmatrix}
\]

Topology Matrix, \( \Gamma \)

- Minimal solution:
  \[
  \begin{bmatrix}
n(X) \\
n(A) \\
n(B) \\
n(Y) \\
\end{bmatrix} = \begin{bmatrix}
1 \\
2 \\
3 \\
1 \\
\end{bmatrix}
  \]

- All multiples are valid steady-states
Outline

- Introduction to Scheduling
  - Finding a Steady State
  - Finding a Schedule
  - Scheduling Tradeoffs
- Phased Scheduling Algorithm
  - Code Size / Buffer Size
  - Hierarchical scheduling
  - Results
Computing the Schedule

- Schedule indicates exact ordering of nodes
- Steady state indicates only the multiplicity
  - A graph might have a valid steady-state without having any admissible schedule

To build legal schedule, fire any node that:
1. Has enough input items to execute
2. Has not exceeded its multiplicity in the steady state

If deadlock reached before steady state complete, then no valid schedule exists (Lee ‘86)
Initialization Schedule

- Filter Peeking provides a new challenge
- Just Steady State doesn’t work:
Initialization Schedule

- Filter Peeking provides a new challenge
- Just Steady State doesn’t work:
  - A

```
pop = 1
A
push = 3

peek = 3, pop = 2
B
push = 1
```
Initialization Schedule

- Filter Peeking provides a new challenge
- Just Steady State doesn’t work:
  - AA
Initialization Schedule

- Filter Peeking provides a new challenge
- Just Steady State doesn’t work:
  - AAB
Initialization Schedule

- Filter Peeking provides a new challenge
- Just Steady State doesn’t work:
  - AABB
  - Can’t execute B again!
Initialization Schedule

- Filter Peeking provides a new challenge
- Just Steady State doesn’t work:
  - AABB
  - Can’t execute B again!
- Can’t execute A one extra time:
  - AABB
Initialization Schedule

- Filter Peeking provides a new challenge
- Just Steady State doesn’t work:
  - AABB
  - Can’t execute B again!
- Can’t execute A one extra time:
  - AABBA
Initialization Schedule

- Filter Peeking provides a new challenge
- Just Steady State doesn’t work:
  - AABB
  - Can’t execute B again!
- Can’t execute A one extra time:
  - AABBAB
  - Left 3 items between A and B!
Initialization Schedule

- Must have data between A and B before starting execution of Steady State Schedule
- Construct two schedules:
  - One for Initialization
  - One for Steady State
- Initialization Schedule leaves data in buffers so Steady State can execute
Initialization Schedule

- Initialization Schedule:
  -
Initialization Schedule

- Initialization Schedule:
  - A

\[
\begin{align*}
\text{push} &= 3 \\
\text{peek} &= 3, \text{pop} = 2 \\
\text{push} &= 1 \\
\text{pop} &= 1 \\
\text{push} &= 3
\end{align*}
\]
Initialization Schedule

- Initialization Schedule:
  - A
  - Leave 3 items between A and B
- Steady State Schedule:

Initialization Schedule

- Initialization Schedule:
  - A
  - Leave 3 items between A and B
- Steady State Schedule:
  - A
Initialization Schedule

- Initialization Schedule:
  - A
  - Leave 3 items between A and B
- Steady State Schedule:
  - AA
Initialization Schedule

- Initialization Schedule:
  - A
  - Leave 3 items between A and B
- Steady State Schedule:
  - AAB
Initialization Schedule

- Initialization Schedule:
  - A
  - Leave 3 items between A and B
- Steady State Schedule:
  - AABB
Initialization Schedule

- Initialization Schedule:
  - A
  - Leave 3 items between A and B
- Steady State Schedule:
  - AABBB
Initialization Schedule

- Initialization Schedule:
  - A
  - Leave 3 items between A and B

- Steady State Schedule:
  - AABBB
  - Leave 3 items between A and B
Initialization Schedule

- Initialization Schedule:
  - A
  - Leave 3 items between A and B
- Steady State Schedule:
  - AABBB
  - Leave 3 items between A and B
- Number of items preserved

Diagram:
- pop = 1
- A
- push = 3
- peek = 3, pop = 2
- B
- push = 1
Outline

- Introduction to Scheduling
  - Finding a Steady State
  - Finding a Schedule
  - Scheduling Tradeoffs
- Phased Scheduling Algorithm
  - Code Size / Buffer Size
  - Hierarchical scheduling
  - Results
Scheduling Tradeoffs

- There are many possible schedules for a given steady-state
- Order of execution profoundly affects:
  - Latency
  - Buffer size
  - Code Size
- There is a wealth of literature that aims to optimize the schedule by various metrics
Scheduling Tradeoffs Example

- 3:2 Rate Converter
- First filter (A) upsamples by factor of 3
- Second filter (B) downsamples by factor of two
- Schedule:
  ```
  pop = 1
  A
  push = 3
  ```
Scheduling Tradeoffs Example

- 3:2 Rate Converter
  - First filter (A) upsamples by factor of 3
  - Second filter (B) downsamples by factor of two
- Schedule:
  - A
Scheduling Tradeoffs Example

- 3:2 Rate Converter
- First filter (A) upsamples by factor of 3
- Second filter (B) downsamples by factor of two
- Schedule:
  - A

```
push = 1
A
push = 3

pop = 2
B
push = 1
```
Scheduling Tradeoffs Example

- 3:2 Rate Converter
  - First filter (A) upsamples by factor of 3
  - Second filter (B) downsamples by factor of two
- Schedule:
  - A

```
pop = 1
A
push = 3

pop = 2
B
push = 1
```
Scheduling Tradeoffs Example

- 3:2 Rate Converter
- First filter (A) upsamples by factor of 3
- Second filter (B) downsamples by factor of two
- Schedule:
  - AA

```
      ▼
     0  1
       ▼
      pop = 1
      A
      push = 3

      ▼
     3
       ▼
      pop = 2
      B
      push = 1

      ▼
     0
```
Scheduling Tradeoffs Example

- 3:2 Rate Converter
  - First filter (A) upsamples by factor of 3
  - Second filter (B) downsamples by factor of two
- Schedule:
  - AA

```
{ pop = 1
  A
  push = 3
}

pop = 2
B
push = 1
```
Scheduling Tradeoffs Example

- 3:2 Rate Converter
- First filter (A) upsamples by factor of 3
- Second filter (B) downsamples by factor of two
- Schedule:
  - AA
Scheduling Tradeoffs Example

- 3:2 Rate Converter
- First filter (A) upsamples by factor of 3
- Second filter (B) downsamples by factor of two
- Schedule:
  - AAB
Scheduling Tradeoffs Example

- 3:2 Rate Converter
  - First filter (A) upsamples by factor of 3
  - Second filter (B) downsamples by factor of two
- Schedule:
  - AAB
Scheduling Tradeoffs Example

- 3:2 Rate Converter
  - First filter (A) upsamples by factor of 3
  - Second filter (B) downsamples by factor of two
- Schedule:
  - AAB
Scheduling Tradeoffs Example

- 3:2 Rate Converter
- First filter (A) upsamples by factor of 3
- Second filter (B) downsamples by factor of two
- Schedule:
  - AABB
Scheduling Tradeoffs Example

- 3:2 Rate Converter
- First filter (A) upsamples by factor of 3
- Second filter (B) downsamples by factor of two
- Schedule:
  - AABB
Scheduling Tradeoffs Example

- 3:2 Rate Converter
- First filter (A) upsamples by factor of 3
- Second filter (B) downsamples by factor of two
- Schedule:
  - AABB
Scheduling Tradeoffs Example

- 3:2 Rate Converter
- First filter (A) upsamples by factor of 3
- Second filter (B) downsamples by factor of two
- Schedule:
  - AABBB
Scheduling Tradeoffs Example

- 3:2 Rate Converter
- First filter (A) upsamples by factor of 3
- Second filter (B) downsamples by factor of two
- Schedule:
  - AABBB
Scheduling Tradeoffs Example

- 3:2 Rate Converter
- First filter (A) upsamples by factor of 3
- Second filter (B) downsamples by factor of two
- Schedule:
  - AABBB
Scheduling Tradeoffs Example

- 3:2 Rate Converter
  - First filter (A) upsamples by factor of 3
  - Second filter (B) downsamples by factor of two
- Schedule:
  - AABBB
Scheduling Tradeoffs Example

- 3:2 Rate Converter
- First filter (A) upsamples by factor of 3
- Second filter (B) downsamples by factor of two
- Schedule:
  - AABBB
  - A
Scheduling Tradeoffs Example

- 3:2 Rate Converter
- First filter (A) upsamples by factor of 3
- Second filter (B) downsamples by factor of two
- Schedule:
  - AABB
  - A

```
pop = 1
A
push = 3

pop = 2
B
push = 1
```
Scheduling Tradeoffs Example

- 3:2 Rate Converter
- First filter (A) upsamples by factor of 3
- Second filter (B) downsamples by factor of two
- Schedule:
  - AABB
  - A

```
pop = 1
A
push = 3
```

```
pop = 2
B
push = 1
```

```
0
```

```
1
```

```
3
```

Architectures, Languages, and Compilers for the Streaming Domain
PACT 2003 Tutorial - Saman Amarasinghe, William Thies - MIT CSAIL
Scheduling Tradeoffs Example

- 3:2 Rate Converter
  - First filter (A) upsamples by factor of 3
  - Second filter (B) downsamples by factor of two
- Schedule:
  - AABBB
  - AB
Scheduling Tradeoffs Example

- 3:2 Rate Converter
  - First filter (A) upsamples by factor of 3
  - Second filter (B) downsamples by factor of two
- Schedule:
  - ABB
  - AB
Scheduling Tradeoffs Example

- 3:2 Rate Converter
- First filter (A) upsamples by factor of 3
- Second filter (B) downsamples by factor of two
- Schedule:
  - AABBB
  - AB
Scheduling Tradeoffs Example

- 3:2 Rate Converter
  - First filter (A) upsamples by factor of 3
  - Second filter (B) downsamples by factor of two
- Schedule:
  - AABBB
  - ABA
Scheduling Tradeoffs Example

- 3:2 Rate Converter
- First filter (A) upsamples by factor of 3
- Second filter (B) downsamples by factor of two
- Schedule:
  - AABBB
  - ABA
Scheduling Tradeoffs Example

- 3:2 Rate Converter
- First filter (A) upsamples by factor of 3
- Second filter (B) downsamples by factor of two
- Schedule:
  - AABBB
  - ABA
Scheduling Tradeoffs Example

- 3:2 Rate Converter
- First filter (A) upsamples by factor of 3
- Second filter (B) downsamples by factor of two
- Schedule:
  - AABBB
  - ABAB
Scheduling Tradeoffs Example

- 3:2 Rate Converter
- First filter (A) upsamples by factor of 3
- Second filter (B) downsamples by factor of two
- Schedule:
  - AABBB
  - ABAB
Scheduling Tradeoffs Example

- 3:2 Rate Converter
  - First filter (A) upsamples by factor of 3
  - Second filter (B) downsamples by factor of two
- Schedule:
  - AABBB
  - ABAB
Scheduling Tradeoffs Example

- 3:2 Rate Converter
- First filter (A) upsamples by factor of 3
- Second filter (B) downsamples by factor of two
- Schedule:
  - AABBB
  - ABABB
Scheduling Tradeoffs Example

- 3:2 Rate Converter
- First filter (A) upsamples by factor of 3
- Second filter (B) downsamples by factor of two
- Schedule:
  - AABBB
  - ABABB
Scheduling Tradeoffs Example

- 3:2 Rate Converter
  - First filter (A) upsamples by factor of 3
  - Second filter (B) downsamples by factor of two
- Schedule:
  - AABBB
  - ABABB
Scheduling Tradeoffs Example

- 3:2 Rate Converter
- First filter (A) upsamples by factor of 3
- Second filter (B) downsamples by factor of two
- Schedule:
  - AABBB
  - ABABB
Scheduling Tradeoffs – Latency

- AABBB – First data item output after third execution of a filter
  - Also A already consumed 2 data items

- ABABB – First data item output after second execution of a filter
  - A consumed only 1 data item

Architectures, Languages, and Compilers for the Streaming Domain
PACT 2003 Tutorial - Saman Amarasinghe, William Thies - MIT CSAIL
Scheduling Tradeoffs – Buffer Size

- AABBB requires 6 data items of buffer space between filters A and B

- ABABB requires 4 data items of buffer space between filters A and B
Scheduling Tradeoffs – Code Size

- AABBB – Can be compressed into a loop nest with two appearances of the filters:
  - \( \{5A\}\{3B\} \)

- ABABB – Requires three appearances of the filters:
  - \( 2\{AB\}B \)
Scheduling Tradeoffs – Code Size

- AABBB – Can be compressed into a loop nest with two appearances of the filters:
  - \{5A\}{3B\}
  - “Single Appearance Schedule”

- ABABB – Requires three appearances of the filters:
  - 2{AB}B
Single Appearance Scheduling (SAS)

- Every Filter is listed only once in the loop nest denoting the schedule
  - Example: \(5\{4\{AB\}\} \ 6\{C \ 3D\}\)
  - There are multiple SAS schedules for a given graph

- By metric of DSP community, SAS schedules guarantee **minimal code size**
  - Schedule size = \# appearances of filters in schedule
  - Filter invocations are often inlined, and consume more space than the loop nests

- Due to their analyzability, SAS schedules have been the target of almost all optimization research
  - Heuristics for finding SAS with minimal buffer size
  - “Buffer merging” for SAS schedules
  - Etc., etc. (see Bhattacharyya ’99 for review)
Shortcomings of SAS

1. Buffer size explosion for hierarchical components
   - If a large hierarchical component must execute all at once, then its I/O rates are huge
   - Critical consideration for separate compilation

2. Restricted space of schedules considered
   - Hampers effectiveness of buffer, latency optimization

Is there a place for multiple-appearance schedules?
Outline

- Introduction to Scheduling
  - Finding a Steady State
  - Finding a Schedule
  - Scheduling Tradeoffs

- Phased Scheduling Algorithm
  - Code Size / Buffer Size
  - Hierarchical scheduling
  - Results
Our recent work: “Phased Scheduling”

- Implements a multiple-appearance schedule
- Approach:
  - Allows code size to grow to a fixed number of SAS “phases”
- Benefits:
  - Small buffer sizes for hierarchical programs
  - Fine grained control over code size vs buffer size
  - Always avoids deadlock in separate compilation
SAS Example – Buffer Size

- Example: CD-DAT
- CD to Digital Audio Tape rate converter
- Mismatched rates cause large number of executions in Steady State
SAS Example – Buffer Size

- Naïve SAS schedule:
  - 147A 98B 28C 32D
  - Required Buffer Size: 714
  - Unnecessarily large buffer requirements!
SAS Example – Buffer Size

- Naïve SAS schedule:
  - 147A 98B 28C 32D
  - Required Buffer Size: 714
  - Unnecessarily large buffer requirements!

- Optimal SAS CD-DAT schedule:
  - 49{3A 2B} 4{7C 8D}
  - Required Buffer size: 258
Phased Scheduling

- Idea:
  - What if we take the naïve SAS schedule, and divide it into n roughly equal phases?
  - Buffer requirements would reduce roughly by factor of n
  - Schedule size would increase by factor of n
  - May be OK, because buffer requirements dominate schedule size anyway!
Phased Scheduling

- Try \( n = 2 \):
  - Two phases are:
    - 74A 49B 14C 16D
    - 73A 49B 14C 16D
  - Total Buffer Size: 358
  - Small schedule increase
  - Greater \( n \) for bigger savings
Phased Scheduling

- Try $n = 3$:
  - Three phases are:
    - 48A 32B 9C 10D
    - 53A 35B 10C 11D
    - 46A 31B 9C 11D
  - Total Buffer Size: 259
  - Basically matched best SAS result
    - Best SAS was 258
Phased Scheduling

- Try \( n = 28 \):
- The phases are:
  - 6A 4B 1C 1D
  - 5A 3B 1C 1D
  - ...
  - 4A 3B 1C 2D
- Total Buffer Size: 35
- Drastically beat best SAS result
  - Best SAS was 258
A Lower Bound on Buffer Size: Pull Scheduling

- **Pull Scheduling:**
  - Always execute the bottom-most element possible

- **CD-DAT schedule:**
  - 2A B A B 2A B A B C D ... A B C 2D
  - Required Buffer Size: 26
  - 251 entries in the schedule

- Hard to implement efficiently, as schedule is VERY large
CD-DAT Comparison: SAS vs Pull vs Phased

<table>
<thead>
<tr>
<th></th>
<th>Buffer Size</th>
<th>Schedule Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAS</td>
<td>258</td>
<td>4</td>
</tr>
<tr>
<td>Pull Schedule</td>
<td>26</td>
<td>251</td>
</tr>
<tr>
<td>Phased Schedule</td>
<td>35</td>
<td>52</td>
</tr>
</tbody>
</table>
Outline

- Introduction to Scheduling
  - Finding a Steady State
  - Finding a Schedule
  - Scheduling Tradeoffs
- Phased Scheduling Algorithm
  - Code Size / Buffer Size
  - Hierarchical scheduling
  - Results
Hierarchical Phased Scheduling

- Apply technique hierarchically
- Children have several phases which all have to be executed
- Automatically supports cyclo-static filters
- Children pop/push less data, so can manage parent’s buffer sizes more efficiently
Hierarchical Phased Scheduling

- What if a Steady State of a component of a Feedback Loop required more data than available?
- Single Appearance couldn’t do separate compilation!
- Phased Scheduling can provide a fine-grained schedule, which will always allow separate compilation (if possible at all)
Minimal Latency Schedule

- Every Phase consumes as few items as possible to produce at least one data item
- Every Phase produces as many data items as possible
- Guarantees any schedulable program will be scheduled without deadlock
- Allows for separate compilation
- For details, see LCTES ’03 paper
Minimal Latency Scheduling

- Simple FeedbackLoop with a tight *delay* constraint
- Not possible to schedule using SAS
- Can schedule using Phased Scheduling
  - Use Minimal Latency Scheduling

*delay = 10*

Architectures, Languages, and Compilers for the Streaming Domain
PACT 2003 Tutorial - Saman Amarasinghe, William Thies - MIT CSAIL
Minimal Latency Scheduling

- Minimal Latency Phased Schedule:
Minimal Latency Scheduling

- Minimal Latency Phased Schedule:
  - join 2B 5split L

Diagram:

- Nodes labeled with numbers and letters
- Arrows indicating sequence and dependencies
- Note: \( \text{delay} = 10 \)
Minimal Latency Scheduling

- Minimal Latency Phased Schedule:
  - join 2B 5split L
  - join 2B 5split L
Minimal Latency Scheduling

- Minimal Latency Phased Schedule:
  - join 2B 5split L
  - join 2B 5split L
  - join 2B 5split L
Minimal Latency Scheduling

- Minimal Latency Phased Schedule:
  - join 2B 5split L
  - join 2B 5split L
  - join 2B 5split L
  - join 2B 5split 2L

\[ \text{delay} = 10 \]
Minimal Latency Schedule

- Minimal Latency Phased Schedule:
  - join 2B 5split L
  - join 2B 5split L
  - join 2B 5split L
  - join 2B 5split 2L
- Can also be expressed as:
  - 3 {join 2B 5split L}
  - join 2B 5split 2L
- Common to have repeated Phases
Why not SAS?

- Naïve SAS schedule
  - 4join 8B 20split 5L:
    - Not valid because 4join consumes 20 data items
- Would like to form a loop-nest that includes join and L
- But multiplicity of executions of L and join have no common divisors

\[ \text{delay} = 10 \]
Outline

- Introduction to Scheduling
  - Finding a Steady State
  - Finding a Schedule
  - Scheduling Tradeoffs

- Phased Scheduling Algorithm
  - Code Size / Buffer Size
  - Hierarchical scheduling

- Results
Results

- SAS vs Minimal Latency
- Used 17 applications
  - 9 from our ASPLOS ’02 paper
  - 2 artificial benchmarks
  - 2 from Murthy99
  - Remaining 4 from our internal applications
Results - Buffer Size

(Buffer Size (Min latency / Single Appearance))

- S:Peak 21
- HDTV
- CD:DAT
- CFAR
- S:Peak 1024
- Block Max
- MuL
- Vocoder
- Radar
- Bitonic Soft
- 3GPP
- Trellis
- FIR
- Filter Bank
- GMF
- Radio
- FFT
Results – Schedule Size

- Code Size (Min latency / Single Appearance)
- S/Pack1, HDTV, CD, DAT, CIF, CIF2, Block Matrix Mul, Vocoder, Radar, BioVSSort, 3GPP, Trellis, FIR, Filter Bank, QMF, Radio, FFT

- S/Pack1: 517% 1067%
Results - Combined
Conclusion

Presented Phased Scheduling Algorithm

- Provides efficient interface for hierarchical scheduling
- Enables separate compilation with safety from deadlock
- Provides flexible buffer / schedule size trade-off
- Reduces latency of data throughput

Step towards a large scale hierarchical stream programming model