Memory Optimal Single Appearance Schedule with Dynamic Loop Count for Synchronous Dataflow Graphs

Hyunok Oh, ARM Inc.,
Nikil Dutt, UC Irvine,
Soonhoi Ha, Seoul National Univ.
Outline

- Code synthesis from synchronous dataflow model
- Single appearance schedule
- Dynamic loop count single appearance schedule
  - Algorithm
  - Example
- Experiment
- Conclusion
Synchronous Dataflow (SDF) Model

- Useful to describe DSP algorithms
- A node represents a function block (ex: FIR, DCT)
- An arc represents data dependency: FIFO queue of samples
- Statically scheduled at compile-time.

\[
\begin{align*}
A & \xrightarrow{1} B \xrightarrow{2} D \\
C & \xrightarrow{2} B \xrightarrow{2} D
\end{align*}
\]

AACBDD
Software Synthesis Procedure

**SDF (Synchronous Dataflow)**

![Diagram of a Synchronous Dataflow graph]

**Schedule**

\[ = 2(A)CB2(D) \\
= \ldots \]

**Buffer allocation**

```plaintext
main() {
    int AB[2], AC[2], BD[2], CD[2];
    for(i=0;i<2;i++) {A}
    {C}
    {B}
    for(i=0;i<2;i++) {D}
}
```
Software Synthesis Problem

- Automatic code generation from data flow graph
  - The kernel code of a node is already optimized in the library.
  - Determine the schedule
  - Determine the buffer size
  - Codes are generated according to the scheduled sequence with buffer size

- Fundamental Question
  - Can we achieve the similar code quality as manually optimized code in terms of performance and memory requirement?
Memory Requirement

- Data memory: depends on schedule & buffer sharing & buffer management technique

A 3 3 → B 1 1 → C 4 3 → D

- <example>
  - < Schedule 1 > 3(ABC)4(D)
    - buffer size: 3 + 1 + 12 = 16
  - < Schedule 2 > (3A)(3B)(3C)(4D)
    - buffer size: 9 + 3 + 12 = 24
    - buffer sharing: 3 + max (9,12) = 15
  - < Schedule 3 > 3(ABCD)D
    - buffer size: 3 + 1 + 6 = 10
Single Appearance Schedule (SAS)

- Contains only one lexical appearance of each node
  - SAS: 3(ABC)4(D), (3A)(3B)(3C)(4D)
  - Non SAS: 3(ABCD)D
- Minimize code memory size
  - Each node has a single definition in a generated code
Problems of SAS (1/2)

- Large buffer size

- Buffer-optimal non SAS : ABC ABCC BCC
  - Buffer size on AB : 4
  - Buffer size on BC : 7

- SAS : (2A) (3B) (5C)
  - Buffer size on AB : 6
  - Buffer size on BC : 15
Buffer Memory Lower Bound

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

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

- For any schedule

$$LB(e) = \begin{cases} 
(a + b - c + (d \mod c)) & \text{if } d < a + b - c \\
\ d & \text{otherwise}
\end{cases}$$
Problems of SAS (2/2)

- Unschedulable for a graph with delay samples

- Buffer optimal non SAS: (2A)B AB
- SAS: N/A
Outline

- Code synthesis from synchronous dataflow model
- Single appearance schedule
- Dynamic loop count single appearance schedule
  - Algorithm
  - Example
- Experiment
- Conclusion
Dynamic Loop Count SAS (dlcSAS)

- Data memory size = non SAS
- Code memory size = SAS

- Change loop count at run time
- Data memory size
  - Equal to buffer optimal non SAS
- Code memory size
  - Similar to SAS except codes for loop count computation
Example 1: optimal data buffer size

- Buffer-optimal non SAS: ABC ABCC BCC
- Previous SAS: (2A) (3B) (5C)
- dlcSAS
  - AB A(2B) → A {1,2}B
  - BC B(2C) B(2C) → B {1,2,2}C
  - 2(A {1,2}(B {1,2,2}C))
  - = ABC ABCC BCC: buffer-optimal non SAS
- Buffer size on AB: 4
- Buffer size on BC: 7
main()
{
    int n, i, j, a[4], b[7], iC = 0;
    int lB[2] = {1, 2}, lC[3] = {1, 2, 2};
    for (;;) {
        for (n = 0; n < 2; n++) {
            // A’s code */
            for (i = 0; i < lB[n]; i++) {
                // B’s code */
                for (j = 0; j < lC[iC]; j++) {
                    // C’s code */
                    iC = (iC + 1) % 3;
                }
            }
        }
    }
}
Example 2: graph with delays

\[
\text{dlcSAS} \\
2(\{2,1\}A \ B) \\
= (2A)B \ AB
\]

main()
{
    int i,j, a[4],b[4],lA[2]=\{2,1\};
    for(;;) {
        for(i=0;i<2;i++) {
            for(j=0;j<lA[i];j++) {
                // A’s code
            }
            // B’s code
        }
    }
}
Algorithm

- Determine optimal buffer size on each arc at compile time
- Compute loop count of each node at run time
  - Loop count is dependent on the number of samples on accumulated on input arcs and the available buffer size on output arcs
Example

- Optimal buffer size
  - a : 4, b : 7

- Schedule
  - l = loop count and r = # of samples
  - \( l_A = (4-r_a)/3 \)
  - \( l_B = \min(r_a/2, (7-r_b)/5) \)
  - \( l_C = r_b/3 \)
Example

\[ l_A = \frac{4}{3} = 1 \]
Example

$|A| = 4/3 = 1$
Example

\[ l_A = \frac{4}{3} = 1 \]
\[ l_B = \min\left(\frac{3}{2}, \frac{7}{5}\right) = 1 \]
Example

\[ l_A = \frac{4}{3} = 1 \]
\[ l_B = \min\left(\frac{3}{2}, \frac{7}{5}\right) = 1 \]
Example

\[ l_A = \frac{4}{3} = 1 \]
\[ l_B = \min(\frac{3}{2}, \frac{7}{5}) = 1 \]
\[ l_C = \frac{5}{3} = 1 \]
Example

\[ I_A = \frac{4}{3} = 1 \]
\[ I_B = \min(\frac{3}{2}, \frac{7}{5}) = 1 \]
\[ I_C = \frac{5}{3} = 1 \]
Example

\[ I_A = \frac{4}{3} = 1 \]
\[ I_B = \min\left(\frac{3}{2}, \frac{7}{5}\right) = 1 \]
\[ I_C = \frac{5}{3} = 1 \]

\[ I_A = \frac{3}{3} = 1 \]
Example

\[ l_A = \frac{4}{3} = 1 \]
\[ l_B = \min\left(\frac{3}{2}, \frac{7}{5}\right) = 1 \]
\[ l_C = \frac{5}{3} = 1 \]

\[ l_A = 3/3 = 1 \]
\[ l_B = \min\left(\frac{4}{2}, \frac{5}{5}\right) = 1 \]
Example

\[ I_A = \frac{4}{3} = 1 \]
\[ I_B = \min\left(\frac{3}{2}, \frac{7}{5}\right) = 1 \]
\[ I_C = \frac{5}{3} = 1 \]

\[ I_A = \frac{3}{3} = 1 \]
\[ I_B = \min\left(\frac{4}{2}, \frac{5}{5}\right) = 1 \]
Example

\[ I_A = \frac{4}{3} = 1 \]
\[ I_B = \min\left(\frac{3}{2}, \frac{7}{5}\right) = 1 \]
\[ I_C = \frac{5}{3} = 1 \]

\[ I_A = \frac{3}{3} = 1 \]
\[ I_B = \min\left(\frac{4}{2}, \frac{5}{5}\right) = 1 \]
\[ I_C = \frac{7}{3} = 2 \]
Example

\[ I_A = \frac{4}{3} = 1 \]
\[ I_B = \min\left(\frac{3}{2}, \frac{7}{5}\right) = 1 \]
\[ I_C = \frac{5}{3} = 1 \]

\[ I_A = \frac{3}{3} = 1 \]
\[ I_B = \min\left(\frac{4}{2}, \frac{5}{5}\right) = 1 \]
\[ I_C = \frac{7}{3} = 2 \]
Example

\[ l_A = 4/3 = 1 \]
\[ l_B = \min(3/2, 7/5) = 1 \]
\[ l_C = 5/3 = 1 \]

\[ l_A = 3/3 = 1 \]
\[ l_B = \min(4/2, 5/5) = 1 \]
\[ l_C = 7/3 = 2 \]

\[ l_A = 2/3 = 0 \]
Example

\[ l_A = \frac{4}{3} = 1 \]
\[ l_B = \min\left(\frac{3}{2}, \frac{7}{5}\right) = 1 \]
\[ l_C = \frac{5}{3} = 1 \]

\[ l_A = \frac{3}{3} = 1 \]
\[ l_B = \min\left(\frac{4}{2}, \frac{5}{5}\right) = 1 \]
\[ l_C = \frac{7}{3} = 2 \]

\[ l_A = \frac{2}{3} = 0 \]
\[ l_B = \min\left(\frac{2}{2}, \frac{6}{5}\right) = 1 \]
Example

A 3 a 2 B 5 b 3 C

$l_A = 4/3 = 1$  \hspace{1cm}  $l_B = \min(3/2, 7/5) = 1$  \hspace{1cm}  $l_C = 5/3 = 1$

$A = 3/3 = 1$  \hspace{1cm}  $B = \min(4/2, 5/5) = 1$  \hspace{1cm}  $C = 7/3 = 2$

$A = 2/3 = 0$  \hspace{1cm}  $B = \min(2/2, 6/5) = 1$
Example

\[ l_A = \frac{4}{3} = 1 \]
\[ l_B = \min\left(\frac{3}{2}, \frac{7}{5}\right) = 1 \]
\[ l_C = \frac{5}{3} = 1 \]

\[ l_A = \frac{3}{3} = 1 \]
\[ l_B = \min\left(\frac{4}{2}, \frac{5}{5}\right) = 1 \]
\[ l_C = \frac{7}{3} = 2 \]

\[ l_A = \frac{2}{3} = 0 \]
\[ l_B = \min\left(\frac{2}{2}, \frac{6}{5}\right) = 1 \]
\[ l_C = \frac{6}{3} = 2 \]
Example

\[
\begin{align*}
A &= 3 \\
B &= 5 \\
C &= 3
\end{align*}
\]

\[
\begin{align*}
A &= \frac{4}{3} = 1 \\
B &= \min\left(\frac{3}{2}, \frac{7}{5}\right) = 1 \\
C &= \frac{5}{3} = 1
\end{align*}
\]

\[
\begin{align*}
A &= \frac{3}{3} = 1 \\
B &= \min\left(\frac{4}{2}, \frac{5}{5}\right) = 1 \\
C &= \frac{7}{3} = 2
\end{align*}
\]

\[
\begin{align*}
A &= \frac{2}{3} = 0 \\
B &= \min\left(\frac{2}{2}, \frac{6}{5}\right) = 1 \\
C &= \frac{6}{3} = 2
\end{align*}
\]
Code Generation

main() {
    int i, j, k a[4], b[7], IA, IB, IC, ra=0, rb=0;
    for(;;) {
        IA = (4-ra)/3; ra+=3*IA;
        for(i=0; i<IA; i++)
            { /* A’s code */ }
        IB = min(ra/2, (7-rb)/5);
        ra-=2*IB; rb+=5*IB;
        for(j=0; j<IB; j++)
            { /* B’s code */ }
        IC = rb/3; rb-= 3*IC;
        for(k=0; k<IC; k++)
            { /* C’s code */ }
    }
}
Experiments

Filter Bank

<table>
<thead>
<tr>
<th>previous SAS</th>
<th>dlcSAS</th>
<th>Ratio (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>code memory</td>
<td>13128 bytes</td>
<td>13540 bytes</td>
</tr>
<tr>
<td>data memory</td>
<td>15720 bytes</td>
<td>9664 bytes</td>
</tr>
<tr>
<td>total memory</td>
<td>28848 bytes</td>
<td>23204 bytes</td>
</tr>
<tr>
<td>cycles</td>
<td>71060 Kcycl</td>
<td>71363 Kcycl</td>
</tr>
</tbody>
</table>

ARM 9
Conclusion

- A new single appearance schedule
  - Dynamic loop count single appearance schedule
  - Data buffer size is equal to buffer optimal non SAS
  - Code size is equal to single appearance schedule except loop count computation
  - 20% total memory reduction
  - Less than 1% performance overhead
Thanks!