Towards Memory-Optimal Schedules for SDF

Mitchell Jones
Department of Computer Science
University of Illinois at Urbana-Champaign

Julian Mestre
School of Information Technologies
The University of Sydney

Bernhard Scholz
School of Information Technologies
The University of Sydney
Data-Flow Computation

• Stream programming paradigm based on Kahn’s processing model
• Processes large unbounded regular sequences of data forever
• Applications
  • Digital signal processing, audio, video, graphics, networking, and for big data
• Computations
  • Actors communicate via data channels only
  • Data-channel connects \textit{producer} with \textit{consumer}
  • Tokens are send and received on data channels
  • Actor invocations (\textit{aka. firing}) require coordination
    • Otherwise starvation of actors or memory depletion
Synchronous Data-Flow (SDF)

• Restricted Data-Flow Computational model
• Permits static scheduling / run in steady state
  • Pre-computes sequences of actor invocations (firings)
• Per actor firing data rates are fixed:
  • Actor produces a fixed number of tokens on outgoing channels
  • Actor consumes a fixed number of tokens from ingoing channels
• Data-channels implemented as FIFO-buffers

• Research Questions:
  • What are the sizes of the FIFO buffers?
Data-Channels as FIFO Buffers

• Data channels implemented as FIFO buffers
  • Tokens from producer stored in buffer
  • Consumer can retrieve them

• FIFO buffers require memory
  • Scarce resource
  • Cache effects
  • Small memory for embedded systems

• Different FIFO implementations
  • Static allocation in memory
  • Dynamic allocation
  • In hardware
Static FIFO Buffers for Data Channels

• FIFO buffer for a single data-channel between actor \( u \) and actor \( v \)
• Memory is \textit{not} shared between FIFO buffers; size is fixed.

```c
#include <stdio.h>

#define SZ_uv 100

typedef struct {
  token data;
} queue_t;

token queue_uv[SZ_uv]; // memory for data channel
int head_uv = 0, tail_uv = 0;

void produce_uv(token t) {
  queue_uv[tail_uv++] = t;
  tail_uv %= SZ_uv;
}

token consume_uv() {
  token t = queue_uv[head_uv++];
  head_uv %= SZ_uv;
  return t;
}
```

• \textit{Minimize the sum over all buffer sizes} \( \text{SZ}_{uv} \)?
Dynamic FIFO Buffers for Data Channels

• FIFO buffer for a single data-channel between actor $u$ and actor $v$
• Memory for tokens is shared between FIFO buffers; size is variable.

```c
tokenQ head_uv = NULL, tail_uv = NULL;
void produce_uv(token t) {
  tokenQ p = alloc(t);
  if (head_uv == NULL) head_uv = tail_uv = p;
  else tail_uv->next = p;
}
token consume_uv() {
  token t = head_uv->data;
  tokenQ p = head_uv;
  head_uv = head_uv->next; free(p);
  if (head_uv == NULL) tail_uv = NULL;
  return t;
}
```

• Minimize heap memory used by `alloc/free`!
Hardware Implementation for Data Channels

• Fixed-sized FIFO buffer for all data-channels

• May be relevant for hardware/FPGA implementations for SDF

• Minimize the total memory for all fixed sized FIFO buffers
Research Result for Minimizing FIFO Buffers

• Our research results for minimizing sizes of FIFO buffers

<table>
<thead>
<tr>
<th>Static FIFOs</th>
<th>Dynamic FIFOs</th>
<th>Hardware FIFOs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimal in $P$</td>
<td>Minimal in $NP$</td>
<td>Minimal in $P$</td>
</tr>
</tbody>
</table>

• Main Idea for Static FIFOs and HW FIFOs
  • Exploit properties of steady state scheduling theory
  • Online algorithm based on priority queues
    • Space complexity $O(n)$; run-time complexity $O(\log n)$ per actor invocation
    • Priority queue contains a single element for each actor
Steady-State Schedule

- Static periodic finite schedule for well-formed SDF programs
- Fill-state of buffers is the same before and after executing schedule
  - Also known as steady-state
- Static periodic finite schedule is executable ad-infinitum
- Basic Balance Equations for Steady State Schedule:
  \[ p(u, v) \cdot r(u) = c(u, v) \cdot r(v) \quad \forall (u, v) \in E \]
  \[ r(u) \geq 0 \quad \forall u \in V \]
- Production rate \( p(u,v) \); consumption rate \( c(u,v) \); repetitions \( r(u) \)
- Null-space of topological matrix
Example: Steady-State Schedule

• Balance Equations

\[
\begin{align*}
1 \cdot r(a) &= 2 \cdot r(b) \\
1 \cdot r(a) &= 1 \cdot r(c) \\
2 \cdot r(b) &= 1 \cdot r(c)
\end{align*}
\]

• Solution:

\[
\begin{align*}
 r(a) &= 2 \\
r(b) &= 1 \\
r(c) &= 2
\end{align*}
\]
Steady-State Schedules

• Repetition vector dictates the number of occurrences & length of schedule
• Exponential number of schedules $|S|$ for steady-state

\[
L = \sum_{u \in V} r(u)
\]

\[
|S| = \frac{L!}{\prod_{u \in V} r(u)!}
\]

• Which schedule is beneficial for minimizing memory?
  • Exhaustive search intractable
  • State-of-the-art: finds an ad-hoc solution
Optimal Schedule

- Schedule

Maximum Tokens: 1
Max fill-state:
- \((a,b)\Rightarrow 1\)
- \((a,c)\Rightarrow 0\)
- \((b,c)\Rightarrow 0\)

- Assume initial token on channel \((a,b)\)
Optimal Schedule

- Schedule

| a | b | c | a | c |

Maximum Tokens: 3
Max fill-state:
- (a,b) $\rightarrow$ 2
- (a,c) $\rightarrow$ 1
- (b,c) $\rightarrow$ 0
Optimal Schedule

• Schedule
  
  a b c a c

Maximum Tokens: 3
Max fill-state:
• (a,b) → 2
• (a,c) → 1
• (b,c) → 2
Optimal Schedule

• Schedule

```
| a | b | c | a | c |
```

Maximum Tokens: 3
Max fill-state:
• \((a,b)\rightarrow 2\)
• \((a,c)\rightarrow 1\)
• \((b,c)\rightarrow 2\)
Optimal Schedule

• Schedule

\[
\begin{array}{cccc}
 a & b & c & a & c \\
\end{array}
\]

Maximum Tokens: 3
Max fill-state:
• (a,b) \rightarrow 2
• (a,c) \rightarrow 1
• (b,c) \rightarrow 2
Optimal Schedule

• Schedule

Maximum Tokens: 3
Max fill-state:
• (a,b) \rightarrow 2
• (a,c) \rightarrow 1
• (b,c) \rightarrow 2
Greedy Algorithm

• Greedy algorithm developed by Battacharya, Murthy, and Lee
  • An actor is fireable if incoming edges have enough tokens to fire actor
  • An actor is deferrable if it is fireable and the consumers attached to its out-going edges are fireable
• Favor fireable and non-deferrable nodes to minimize buffer sizes
• If non-deferrable nodes do not exist, search for beneficial fireable node
Greedy Algorithm

**Algorithm 1** \textsc{greedy}\((\mathcal{G}, \mathcal{T}, \mathcal{P}, \mathcal{C}, \mathcal{T})\)

1. \(L \leftarrow \sum_{u \in \mathcal{V}} r(u)\)
2. let \(F\) be the set of fireable actors in \(\mathcal{V}\) using fill-state \(t\)
3. let \(D\) be the set of deferrable actors in \(F\)
4. \textbf{for} \(i = 1\) to \(L\) \textbf{do}
5. \textbf{if} \(F \setminus D \neq \emptyset\) \textbf{then}
6. \hspace{1em} \(u \leftarrow\) an actor from \(F \setminus D\)
7. \textbf{else}
8. \hspace{1em} \(u \leftarrow\) an actor in \(F\) that increases total number of tokens the least
9. \hspace{1em} add \(u\) to the schedule \(s\)
10. \(r(u) \leftarrow r(u) - 1\)
11. invoke actor \(u\)
12. \text{update} \(F\) and \(D\) \hspace{1em} // An actor \(u\) is not fireable if \(r(u) < 1\).
13. \textbf{return} \(s\)
Greedy Schedule

- Schedule

  a b a c c

Maximum Tokens: 1
Max fill-state:
- (a,b) \rightarrow 1
- (a,c) \rightarrow 0
- (b,c) \rightarrow 0
Greedy Schedule

• Schedule

Maximum Tokens: 3
Max fill-state:
• (a,b)\rightarrow 2
• (a,c)\rightarrow 1
• (b,c)\rightarrow 0
Greedy Schedule

• Schedule

Maximum Tokens: 3
Max fill-state:
• (a,b) $\Rightarrow$ 2
• (a,c) $\Rightarrow$ 1
• (b,c) $\Rightarrow$ 2
Greedy Schedule

• Schedule
  a b a c c

Maximum Tokens: 5
Max fill-state:
• (a,b) → 2
• (a,c) → 2
• (b,c) → 2
Greedy Schedule

- Schedule

  a  b  a  c  c

  Maximum Tokens: 5
  Max fill-state:
  - (a,b)  2
  - (a,c)  2
  - (b,c)  2
Greedy Schedule

• Schedule

a b a c c

Maximum Tokens: 3
Max fill-state:
• (a,b) → 2
• (a,c) → 2
• (b,c) → 2
Fill-State of Channels

• Define a fill-state function for channels \((u,v)\)
• Tracks fill-state of data-channel for a schedule

\[
 f_{s}^{i+1}(u,v) = \begin{cases} 
 f_{s}^{i}(u,v) + p(u,v), & \text{if } u = s(i + 1), \\
 f_{s}^{i}(u,v) - c(u,v), & \text{if } v = s(i + 1), \\
 f_{s}^{i}(u,v), & \text{otherwise.}
\end{cases}
\]

where \(s\) is the schedule, \(i\) is the step in the schedule.

• Initial fill-state is known as delay.
Problem Definitions with Fill-States

• Minimize Buffers

\[(P1) \quad \min_{(s,t)} \max_{0 \leq I \leq L} \max_{(u,v) \in E} f_s^I(u, v)\]

\[(P2) \quad \min_{(s,t)} \sum_{(u,v) \in E} \max_{0 \leq I \leq L} f_s^I(u, v)\]

\[(P3) \quad \min_{(s,t)} \max_{0 \leq I \leq L} \sum_{(u,v) \in E} f_s^I(u, v)\]

• \((P1) \iff\) HW buffers; \((P2) \iff\) static buffers; \((P3) \iff\) dynamic buffers
Lower-Bound on Fill-State

• Assume production rate $p$, consumption rate $c$

• Lower-Bound for FIFO buffer size
  • $LB(u,v) = p(u,v) + c(u,v) + \gcd(p(u,v), c(u,v))$
  • $\forall i: LB(u,v) \leq f^i(u,v)$

• Follows Euclid’s argument (cf. Lemma 1)

• Lower-bound is maintainable as upper-bound (cf. Lemma 2)
Canonical Algorithm

- **(P1)** and **(P2)** permit an optimal algorithm in P
  - Requires an arbitrary total order for actors $\pi$
  - Compute initial fill-state for data-channels
  - Each actor has a priority $x$ and is added to a priority queue
    - Initial priority of an actor is 0 and added to queue
    - Ties are broken with total order
    - New priority is $x' = x + \frac{1}{r(u)}$ where $x$ is old priority of an actor
  - Use a priority queue in size of the number of actors
  - Break ties with a fixed order

- **(P3)** is a NP hard problem
  - Reduction uses the Minimum Feedback-Arc-set (FAS) Problem
  - See Appendix of paper
Canonical Algorithm

Algorithm 2 CANONICAL((V, E, p, c), π)

1. for (u, v) ∈ E do
2. t_π(u, v) ← \begin{cases} c(u, v) - \gcd(p(u, v), c(u, v)) & \text{if } \pi(u) < \pi(v) \\ c(u, v) & \text{if } \pi(v) < \pi(u) \end{cases}
3. let Q be an empty priority queue
4. for u ∈ V do
5. insert u with priority 0 into Q
6. while true do
7. (u, x) ← delete-min(Q) // break ties using the π order
8. execute actor u
9. insert u with priority x + \frac{1}{r(u)} into Q
### Experiments: Quality & Runtime

<table>
<thead>
<tr>
<th>Instance</th>
<th>CANONICAL</th>
<th></th>
<th>GREEDY</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(P1)</td>
<td>(P2)</td>
<td>time (s)</td>
<td>(P2)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>586</td>
<td>0.0030</td>
<td>1226</td>
</tr>
<tr>
<td>15</td>
<td>28</td>
<td>1048</td>
<td>0.0047</td>
<td>2937</td>
</tr>
<tr>
<td>20</td>
<td>32</td>
<td>2483</td>
<td>0.0081</td>
<td>7818</td>
</tr>
<tr>
<td>25</td>
<td>48</td>
<td>6131</td>
<td>0.0143</td>
<td>30306</td>
</tr>
<tr>
<td>30</td>
<td>60</td>
<td>9486</td>
<td>0.0188</td>
<td>47979</td>
</tr>
<tr>
<td>35</td>
<td>70</td>
<td>16782</td>
<td>0.0291</td>
<td>68126</td>
</tr>
<tr>
<td>40</td>
<td>80</td>
<td>22927</td>
<td>0.0352</td>
<td>149469</td>
</tr>
<tr>
<td>45</td>
<td>84</td>
<td>29781</td>
<td>0.0454</td>
<td>244380</td>
</tr>
<tr>
<td>50</td>
<td>100</td>
<td>46203</td>
<td>0.0567</td>
<td>347676</td>
</tr>
</tbody>
</table>

- **Greedy**
  - computation of fireable / deferrable takes too long.
  - uses more memory
Experiments

- Canonical: faster and better performance for random instances
Conclusion

• Minimizing buffer sizes for SDF programs is important
  • Impact on data caches
  • large data to process
  • increases with complex SDFs

• Three notions of memory optimality
  • Static FIFO buffers (in P)
  • Dynamic FIFO buffers (in NP)
  • Hardware FIFO buffers (in P)

• Introduce Canonical Algorithm using a Priority Queue
  • Show that lower-bound is maintainable as upper-bound
  • Faster than state-of-the-art algorithm
  • Optimality guarantees
  • Space complexity $O(n)$; run-time complexity $O(\log n)$ per actor invocation

• Provided some preliminary experimental results