Managing Latency in Embedded Streaming Applications under Hard-Real-Time Scheduling

Mohamed A. Bamakhrama and Todor Stefanov
October 8, 2012 - CODES+ISSS - Tampere, Finland
Introduction

- Many modern embedded streaming systems require:
  - Real-time execution of applications on multiprocessor platforms
  - Ability to add/remove new applications at run-time

- Solution approach: Apply classical real-time scheduling theory to dataflow models
  - Dataflow model: Cyclo-Static Dataflow (CSDF)
  - The problem is classical real-time scheduling theory uses different task models than dataflow!
The periodic real-time task model

- An application is modeled as a set of independent tasks

- A task is characterized by 4 parameters: WCET, period, start time, and deadline

  - If deadline = period → implicit-deadline (IDP)
  - If deadline < period → constrained-deadline (CDP)
Cyclo-Static Dataflow (CSDF)

- An application is modeled as a set of dependent tasks (represented as a directed graph)
  - Graph nodes represent tasks (a.k.a. actors)
  - Graph edges represent data dependencies

- A graph must be consistent in order to be scheduled using bounded memory
  - A finite sequence of actor invocations that can be repeated infinitely
    - e.g. AABCD AABCD AABCD...
  - Each actor $v_i$ has a repetition $rep_i$ in such sequence. For example: $rep_A=2$, $rep_B=1$
Context of This Work

- Both models do not match!
  - Solution: Daedalus$^{\text{RT}}$ framework
  - Open-source, available at: http://daedalus.liacs.nl

Throughput Analysis

- The CSDF graph actors are scheduled as *periodic real-time tasks*
  - Such scheduling approach is called *strictly periodic scheduling* (SPS)

- SPS delivers the *maximum achievable throughput* for *matched I/O rates* graphs
  - Matched I/O rates graphs represent 80% of streaming applications
What about Latency?

- Latency is more important for some applications

- Defined as the time elapsed between the start of the first firing of the source in a path and the completion of the first firing of the sink in the same path.
Problem Statement

- Daedalus$^{RT}$ uses *implicit-deadline (IDP) model*

- IDP model delivers optimal throughput and latency for *balanced* graphs
  - $rep_i \text{WCET}_i = rep_j \text{WCET}_j$ for all $v_i, v_j$

- However, IDP *increases* latency significantly for unbalanced graphs
  - *Constrained-deadline (CDP) model* can be used to manage the latency
**Balanced vs. Unbalanced**

Graph 1 (Balanced)

1. **v1** (1) → **v2** (1) → **v3** (1)

Graph 2 (Unbalanced)

1. **v1** (1) → **v2** (9) → **v3** (1)

<table>
<thead>
<tr>
<th>Self-timed</th>
<th>IDP</th>
<th>CDP</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>v1 (1)</strong></td>
<td><strong>R</strong></td>
<td><strong>R</strong></td>
</tr>
<tr>
<td><strong>v2 (1)</strong></td>
<td><strong>L</strong></td>
<td><strong>L</strong></td>
</tr>
<tr>
<td><strong>v3 (1)</strong></td>
<td><strong>M</strong></td>
<td><strong>M</strong></td>
</tr>
</tbody>
</table>

| Graph 1 | 1/1 | 3 | 3 |
| Graph 2 | 1/9 | 11 | 2 |

| Graph 1 | 1/1 | 3 | 3 |
| Graph 2 | 1/9 | 27 | 2 |

**CDP can help in reducing L without compromising R**
Paper Contributions

Acyclic CSDF Graph $G$

Yes

Matched I/O?

No

$\hat{R}_{SPS} = \hat{R}$

Yes

Balanced?

No

$\hat{R}_{SPS} \leq \hat{R}$

Yes

IDP is LTO

No

$\bar{L}_{IDP} = \bar{L}$

Yes

CDP is LTO

No

$\bar{L}_{CDP} = \bar{L}$

Use CDP to reduce L

$\bar{L}_{CDP} \geq \bar{L}$
Managing Latency using the CDP model

- We propose an algorithm which checks for the bottleneck actors and assigns them earlier deadlines.

- The deadline is controlled by the designer using *deadline factors*. For actor $v_i$, a deadline factor $d_i \in [0,1]$ specifies the deadline as follows:

$$Deadline_i = WCET_i + d_i(Period_i - WCET_i)$$
Example 1: $\vec{d} = \vec{1}$

```
int main()
{
    int i, j;
    int img[11][4], img1[11][4];

    while(1) {
        for(i=1;i<=10;i++) {
            for(j=1;j<=3;j++) {
                src(&img[i][j],&img1[i][j]);
                if(j<=2)
                    img[i][j]=f1(img[i][j]);
                else
                    img[i][j]=f2(img[i][j]);
                snk(img[i][j],img1[i][j]);
            }
        }
        return 0;
    }
}
```

```
Latency = 56
Min. Latency = 43
```

```
<table>
<thead>
<tr>
<th>Actor</th>
<th>Min. period</th>
<th>Start time (absolute time)</th>
<th>Deadline</th>
</tr>
</thead>
<tbody>
<tr>
<td>v1</td>
<td>8</td>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td>v2</td>
<td>12</td>
<td>8</td>
<td>12</td>
</tr>
<tr>
<td>v3</td>
<td>24</td>
<td>24</td>
<td>24</td>
</tr>
<tr>
<td>v4</td>
<td>8</td>
<td>32</td>
<td>8</td>
</tr>
</tbody>
</table>
```
**Example 1: \( \vec{d} = 0.5 \)**

```c
int main()
{
    int i, j;
    int img[11][4], img1[11][4];

    while(1) {
        for(i=1;i<=10;i++) {
            for(j=1;j<=3;j++) {
                src(&img[i][j],&img1[i][j]);

                if(j<=2)
                    img[i][j]=f1(img[i][j]);
                else
                    img[i][j]=f2(img[i][j]);

                snk(img[i][j],img1[i][j]);
            }
        }
        return 0;
    }
}
```

### Table: Actor Schedule

<table>
<thead>
<tr>
<th>Actor</th>
<th>Min. period</th>
<th>Start time (absolute time)</th>
<th>Deadline</th>
</tr>
</thead>
<tbody>
<tr>
<td>v1</td>
<td>8</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>v2</td>
<td>12</td>
<td>6</td>
<td>12</td>
</tr>
<tr>
<td>v3</td>
<td>24</td>
<td>22</td>
<td>24</td>
</tr>
<tr>
<td>v4</td>
<td>8</td>
<td>30</td>
<td>6</td>
</tr>
</tbody>
</table>

**Latency = 52**

**Min. Latency = 43**
Example 1: \( \vec{d} = \vec{0} \)

\[
\begin{align*}
\text{int main()} &
\{ \\
\text{int i, j;} &
\text{int img[11][4], img1[11][4];} \\
\text{while(1) {} } &
\{
\text{for(i=1;i<=10;i++){} } &
\{ \\
\text{for(j=1;j<=3;j++){} } &
\{ \\
\text{src(&img[i][j],&img1[i][j]);} &
\}
\}
\}
\}
\text{return 0;} &
\}
\end{align*}
\]

\[
\begin{align*}
\text{Latency} = 49 &
\text{ Min. Latency} = 43
\end{align*}
\]

\[
\begin{array}{|c|c|c|c|}
\hline
\text{Actor} & \text{Min. period} & \text{Start time (absolute time)} & \text{Deadline} \\
\hline
v1 & 8 & 0 & 5 \\
v2 & 12 & 5 & 12 \\
v3 & 24 & 21 & 24 \\
v4 & 8 & 29 & 4 \\
\hline
\end{array}
\]
Example 2: H.263 decoder

![Diagram of H.263 decoder process with actor names and latencies]

<table>
<thead>
<tr>
<th>Actor</th>
<th>Min. Period</th>
<th>Deadline $d = 1$</th>
<th>Deadline $d = 0.5$</th>
<th>Deadline $d = 0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>vld</td>
<td>1782</td>
<td>1782</td>
<td>896</td>
<td>10</td>
</tr>
<tr>
<td>iq</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>idct</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>mc</td>
<td>1782</td>
<td>1782</td>
<td>895</td>
<td>8</td>
</tr>
<tr>
<td><strong>Latency</strong></td>
<td><strong>5349</strong></td>
<td><strong>3575</strong></td>
<td><strong>1802</strong></td>
<td></td>
</tr>
</tbody>
</table>

Min. Latency $= 1802$
Evaluation

- We evaluate the proposed technique using 19 real applications

- We consider the impact of deadline factors on latency and resource requirements by using different values of the deadline factor
Impact on Latency

Ratio to the minimum achievable latency

- $d_i = 1$
- $d_i = 0.5$
- $d_i = 0$

Graph showing the impact on latency with different values of $d_i$ for various devices.
Impact on Resource Usage

- The number of processors is computed using the following sufficient test: \( M = \left\lceil \sum \frac{WCET_i}{\min(Deadline_i, Period_i)} \right\rceil \)
Relationship between Latency and Required Resources

$L$ scales linearly while $M$ does not for the sufficient test
Conclusions

- We propose a decision tree to assist the designer in choosing the suitable task model in terms of latency and throughput

- IDP model is latency and throughput optimal for balanced graphs, however, it increases latency of unbalanced graphs

- CDP model can be used to reduce latency without impacting the throughput

- Choosing the deadline involves a tradeoff between latency and resource requirements
Thank you