System-level Scheduling of Real-time Streaming Applications using a Semi-partitioned Approach

Emanuele Cannella, Mohamed A. Bamakhrama, and Todor Stefanov

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University

March 27, 2014
Outline

Background and Motivation

Proposed Semi-partitioned Approach

Experimental Results

Conclusions
Modern embedded MPSoCs have tight constraints:

1. Real-time guarantees
2. Limited resources (#PE, memory)
3. Limited power/energy budget

Additional desirable features:

4. Multiple application support (temporal isolation)
5. Dynamic loading of applications (fast admission control)
Introduction

Modern embedded MPSoCs have tight constraints:

1. Real-time guarantees
2. Limited resources (#PE, memory)
3. Limited power/energy budget

Additional desirable features:

4. Multiple application support (temporal isolation)
5. Dynamic loading of applications (fast admission control)
Current State-of-the-Art

Applications modeled as acyclic Cyclo-Static Dataflow (CSDF) Graphs:

\[ v_1 \overset{[1]}{\rightarrow} v_2 \overset{[0,3]}{\rightarrow} v_3 \]

\[ e_1 \]

\[ e_2 \]

\[ v_1 \]
(1)

\[ v_2 \]
(2)

\[ v_3 \]
(2)

\[ [1,2] \]

\[ [1,2] \]

\[ [0,3] \]

\[ [1] \]

\[ [1] \]

\[ T_1 \]

\[ T_2 \]

\[ T_3 \]

\[ S_1 \]

\[ S_2 \]

\[ S_3 \]

This schedule is called \( SPS \) (Strictly Periodic Schedule).

For \( SPS \), classical real-time scheduling theory can be applied.

---

\(^1\)M. Bamakhrama and T. Stefanov. “Hard-real-time scheduling of data-dependent tasks in embedded streaming applications”. In: EMSOFT ’11.
Current State-of-the-Art

Applications modeled as acyclic Cyclo-Static Dataflow (CSDL) Graphs:

\[ v_1 \rightarrow v_2 \rightarrow v_3 \]

\[ e_1 \rightarrow e_2 \]

\[ v_1 \rightarrow v_2 \rightarrow v_3 \]

\[ e_1 \rightarrow e_2 \]

\[ v_1 \rightarrow v_2 \rightarrow v_3 \]

\[ e_1 \rightarrow e_2 \]

...can be converted to *implicit-deadline* periodic task sets\(^1\):

\[ \forall \text{ actor } v_i \Rightarrow \tau_i = (C_i, T_i, S_i) \]

- This schedule is called **SPS** (Strictly Periodic Schedule)
- For **SPS**, classical real-time scheduling theory can be applied

---

\(^1\)M. Bamakhrama and T. Stefanov. “Hard-real-time scheduling of data-dependent tasks in embedded streaming applications”. In: EMSOFT ’11.
Motivation

Related work on SPS only consider global or partitioned hard real-time (HRT) scheduling algorithms

- **Global approaches:**
  - Optimality
  - High scheduling overheads (preemptions, migrations)
  - High memory overhead in distributed memory systems

- **Partitioned approaches:**
  - No migration or memory overhead
  - Partitioning issues

Recently: **semi-partitioned** scheduling approaches

- Most tasks are statically allocated, a few can migrate
- So far only independent task sets have been considered

Can we benefit from semi-partitioned approaches when mapping data-dependent tasks to MPSoCs?
EDF-fm\textsuperscript{2}

Features:

- **Restricted migration** (less task state to be migrated, if any)
- Migrating tasks migrate between two processors only
- Implementation has minimal overhead compared to “standard” EDF

\[ S_{1,1} = \frac{3}{10}, \quad S_{3,1} = \frac{3}{10}, \quad S_{3,2} = \frac{1}{10}, \quad S_{4,2} = \frac{1}{2}, \quad S_{5,2} = \frac{2}{5}, \quad S_{5,3} = \frac{1}{10} \]

But: EDF-fm is a soft real-time (SRT) algorithm

- Tasks can miss deadlines by a bounded value: tardiness ($\Delta$)

---

Implications of the SRT Approach

The SPS approach assumes HRT scheduling (no deadline miss):

In the SRT approach, we have to:

1. Adjust start times
2. Change buffer sizes
3. Devise a task assignment heuristic to minimize memory/latency overheads
Adjusting Actors Start Times

Start time of $v_j$ depends on the source node production pattern:

- Each invocation of $v_i$ is affected by the maximum tardiness.
- No invocation of $v_j$ has tardiness.

We can adapt existing formulas for start times, by considering the production pattern of $\tilde{v}_i$ instead of $v_i$. 
Adjusting Actors Start Times

Start time of $v_j$ depends on the source node production pattern:

Worst case pattern for start time calculation:

- Each invocation of $v_i$ is affected by the maximum tardiness
- No invocation of $v_j$ has tardiness

⇒ We can adapt existing formulas for start times, by considering the production pattern of $\tilde{v}_i$ instead of $v_i$
Changing Buffer Sizes

After tardiness-aware start times have been derived

- We re-consider each source-destination pair:

\[ \Delta_j \]
\[ \Delta_j \]

\[ Vi \rightarrow e \rightarrow Vj \]

Worst case pattern for buffer sizes calculation:

- No invocation of \( v_i \) has tardiness
- Each invocation of \( v_j \) is affected by the maximum tardiness

\[ \Rightarrow \text{We can adapt existing formulas for buffer sizes by considering the worst-case consumption pattern} \]
FFD-SP Assignment Heuristic

1. Assign tasks using First-Fit Decreasing (FFD)

2. If FFD fails for a task, try semi-partitioning
   - Assign the first share to the largest available utilization among processors
   - Find the best fit for the remaining share

3. Optimize the assignment
   - If possible, move “small shares” to processors which run less tasks

The optimization step reduces the number of tasks with tardiness
**FFD-SP Assignment Heuristic**

1. Assign tasks using First-Fit Decreasing (FFD)
2. If FFD fails for a task, try semi-partitioning
   - Assign the first share to the largest available utilization among processors
   - Find the best fit for the remaining share
3. Optimize the assignment
   - If possible, move “small shares” to processors which run less tasks

The optimization step reduces the number of tasks with tardiness
**FFD-SP Assignment Heuristic**

1. Assign tasks using First-Fit Decreasing (FFD)
2. If FFD fails for a task, try semi-partitioning
   - Assign the first share to the largest available utilization among processors
   - Find the best fit for the remaining share
3. Optimize the assignment
   - If possible, move “small shares” to processors which run less tasks

The optimization step reduces the number of tasks with tardiness
FFD-SP Assignment Heuristic

1. Assign tasks using First-Fit Decreasing (FFD)
2. If FFD fails for a task, try semi-partitioning
   - Assign the first share to the largest available utilization among processors
   - Find the best fit for the remaining share
3. Optimize the assignment
   - If possible, move “small shares” to processors which run less tasks

The optimization step reduces the number of tasks with tardiness
**FFD-SP Assignment Heuristic**

1. Assign tasks using First-Fit Decreasing (FFD)
2. If FFD fails for a task, try semi-partitioning
   - Assign the first share to the largest available utilization among processors
   - Find the best fit for the remaining share
3. Optimize the assignment
   - If possible, move “small shares” to processors which run less tasks

The optimization step reduces the number of tasks with tardiness
Reduction in Number of Processors

We compare the number of processors required by:

- **OPT**: an optimal, global scheduler
- **SP**: semi-partitioned + FFD-SP (our approach)
- **FFD**: partitioned EDF + FFD

![Graph showing the comparison of processors required by OPT, SP, and FFD for different applications.](image)

**FFD** on average requires **18%** more processors than **OPT**

**SP** on average requires **2%** more processors than **OPT**
Overheads: Memory and Latency

**Memory overhead:** increase in code size and buffer sizes

$\Rightarrow$ **SP** on average requires 24% more memory than **FFD**

**SP latency overhead:** 29% (on average) compared to FFD
Conclusions

• A semi-partitioned soft real-time scheduler can be effectively used for real-time streaming applications
  - Hard real-time behavior at the I/O interfaces can be guaranteed

• Over our benchmarks, the proposed approach can:
  - Reduce the number of required processors (-16%)
  - Keep reasonable memory (+24%) and latency (+29%) overheads

• With the **same application throughput**, the proposed approach allows a trade off between:
  - The number of required processors
  - Required memory and latency guarantee
Thanks for your attention!

Questions?