A Methodology for Automated Design of Hard-Real-Time Embedded Streaming Systems

Mohamed Bamakhrama, Jiali Teddy Zhai, Hristo Nikolov, and Todor Stefanov

Leiden University, The Netherlands

March 14, 2012

DATE 2012 – Dresden, Germany
Introduction

• Hard-real-time embedded streaming systems are becoming more complex
  – Increasing computational demands
    • Uniprocessor → multiprocessor
  – Integration of multiple applications into a single platform with support for adding applications at run-time
    • Platform must provide Quality-of-Service (QoS) guarantees and temporal isolation
Problem Statement

• How to design an embedded multiprocessor system that:
  1. Runs multiple streaming applications simultaneously,
  2. Provides hard-real-time QoS,
  3. Uses the minimum amount of resources,

While minimizing the design time and effort?
Current Solutions

- Existing flows can be classified based on QoS and *multiple applications support* into:
  - Soft-real-time/Best-effort, single app/multiple apps.
    - See [1] for survey
  - Hard-real-time, multiple apps.
    - PeaCE: Two Models-of-Computation (SPDF + fFSM)
    - NXP(Moreira et al.): SDF + TDM
    - CA-MPSoC: SDF + non-preemptive scheduling
    - MAMPS: SDF + TDM

All these flows share one thing: Use of *Design Space Exploration (DSE)* to determine the minimum number of processors needed to schedule the applications and the mapping of tasks to processors

Our Answer

• Utilize 40+ years of hard-real-time scheduling theory!
  – Bridge real-time scheduling and embedded MPSoC design
• Use hard-real-time scheduling theory/algorithms to:
  – Schedule the applications while providing temporal isolation and hard-real-time QoS
  – Determine the minimum number of processors needed to schedule the applications
  – Determine the mapping of tasks to processors

All of the above is achieved without performing DSE!
An integrated flow for system-level design of embedded streaming systems

- Model-of-Computation (MoC) based design

Consists of three main phases:
- Parallelization
- CSDF Model Derivation and Analysis
- System Synthesis
Key Ingredients (MoCs)

• A multi-dimensional MoC: *Polyhedral Process Networks (PPN)*
  – Expressiveness: applications operating on high dimensional arrays
  – Used for code generation

• The one-dimension counter-part: *Cyclo-Static Dataflow (CSDF)*
  – Temporal analysis / performance-constrained scheduling
Parallelization

• Realized using the pn compiler

• **Input:**
  – A set of static affine nested loop programs (SANLP) without cyclic dependencies written in the C programming language

• **Output:**
  – A set of Polyhedral Process Networks (PPNs) corresponding to the applications
int main()
{
    int i, j;
    int img[10][3], img1[10][3];

    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]=filter1(img[i][j]);
                else
                    img[i][j]=filter2(img[i][j]);

                snk(img[i][j], img1[i][j]);
            }
        }
        return 0;
    }
}
CSDF Model Derivation and Analysis

- Replaces the Design Space Exploration (DSE) phase in the original Daedalus flow

**Input:**
- A set of PPNs + Worst-case execution time (WCET) of each task

**Output:**
- Platform Specifications
- Mapping Specifications
Analysis Model Derivation

- Any non-parameterized acyclic PPN has an equivalent acyclic CSDF graph
- Processes and channels have one-to-one corresponding actors and edges

```
while(1){
    for(i=0;i<=9;i++){
        for(j=0;i<=2;i++){
            if(j<=1)
                READ(IP1,&in1);
            if(j==2)
                READ(IP2,&in1);
            READ(IP3,&in2);
            F(in1,in2,out);
            WRITE(OP3,out);
        }
    }
}
```
Model Derivation - Example

- **Variant domain**: In a variant domain, a PPN process always accesses the same set of input/output ports

V1 = { IP1, IP3, OP1}
V2 = { IP2, IP3, OP1}

```c
while(1){
    for (i=0; i<=9; i++) {
        for (j=0; j<=2; j++) {
            if (j<=1)
                READ(IP1, &in1);
            if (j==2)
                READ(IP2, &in1);
            READ(IP3, &in2);
            F(in1, in2, out);
            WRITE(OP3, out);
        }
    }
}
```
Domain Traversal

- Domain traversal according to lexicographic order

The traversal results in a sequence $S$

$$S = V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1V2V1V1
Compacting the Sequence

• Suffix tree
  – Tree construction: $O(L)$ in time ($L = |S|$)
  – The path from root node to any internal node represents a repetitive pattern
  – For any internal node: #occurrence of sub-string = #child nodes
• Find the shortest sub-string s. t.:
  $|\text{sub-string}| \times |\text{child nodes}| = L$

Example:
$S = V1V2V1V2V1V2\$ 

Compact pattern ($V1V2$): $2 \times 3 = 6$
Consumption/Production Rates Generation

Compact pattern: \{V1, V1, V2\}
V1 = \{ IP1, IP3, OP1 \}
V2 = \{ IP2, IP3, OP1 \}

<table>
<thead>
<tr>
<th></th>
<th>IP1</th>
<th>IP2</th>
<th>IP3</th>
<th>OP1</th>
</tr>
</thead>
<tbody>
<tr>
<td>V1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>V1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>V2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

PPN Process

```c
while(1) {
    for (i=0; i<=9; i++) {
        for (j=0; i<=2; i++) {
            if (j<=1)
                READ(IP1, &in1);
            if (j==2)
                READ(IP2, &in1);
            READ(IP3, &in2);
            F(in1, in2, out);
            WRITE(OP3, out);
        }
    }
}
```

CSDF Actor

[0, 0, 1] -> [1, 1, 0] -> [1, 1, 1]
[1, 1, 1]
Analysis

• The actors in any acyclic CSDF can be scheduled as a set of periodic tasks
• Utilize this result to apply hard-real-time schedulability analysis so that we can derive:
  – The minimum number of processors needed to schedule the application,
  – The mapping of tasks to processors

Apply hard-real-time schedulability analysis! 😊
Analysis - Background

• For a graph \( G \), the repetition vector is \( \tilde{q}_G = [q_1, q_2, \ldots, q_N]^T \)
• Each actor \( v_i \) has a period \( \lambda_i \) given by
  \[
  \lambda_i = \frac{Q}{q_i} \left\lfloor \frac{\eta}{Q} \right\rfloor
  \]
Where:
  – \( Q = \text{lcm}\{q_1, q_2, \ldots, q_N\} \)
  – \( \eta = \max\{WCET(v_i) \cdot q_i\} \)
• Based on these periods, we can derive:
  – Start times of the actors
  – Buffer sizes of the channels
  – Number of processors needed to schedule the tasks
Platform Sizing

- After computing the periods, we compute the minimum number of processors needed to schedule the tasks, denoted by $M$.
- Computing $M$ depends on the algorithm used.
- Example: $\tau = \{A, B, C, D\}$

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>$WCET_i$</td>
<td>5</td>
<td>2</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>$Period_i$</td>
<td>8</td>
<td>8</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>$U_i = \frac{WCET_i}{Period_i}$</td>
<td>5/8</td>
<td>2/8</td>
<td>3/4</td>
<td>2/6</td>
</tr>
<tr>
<td>$U_{sum} = \sum_{\tau_i \in \tau} U_i$</td>
<td></td>
<td>$\frac{47}{24} = 1.9583$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$M$ (Optimal)</td>
<td></td>
<td></td>
<td></td>
<td>$[\frac{47}{24}] = 2$</td>
</tr>
<tr>
<td>$M$ (P-EDF+FF)</td>
<td></td>
<td></td>
<td></td>
<td>$\min{x \in N: B \text{ is } x\text{-partition of } \tau \text{ and } U_{sum} \leq 1 \forall y \in B} = 3$</td>
</tr>
</tbody>
</table>
Results: Flow Execution Time

- Applications: edge-detection filter (sobel), Motion JPEG decoder and encoder

<table>
<thead>
<tr>
<th>Phase</th>
<th>Daedalus&lt;sup&gt;RT&lt;/sup&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of applications</td>
<td>3</td>
</tr>
<tr>
<td><strong>Phase</strong></td>
<td><strong>Time</strong></td>
</tr>
<tr>
<td>Parallelization</td>
<td>0.48 sec.</td>
</tr>
<tr>
<td>WCET Analysis</td>
<td>1 day</td>
</tr>
<tr>
<td>Deriving the CSDF</td>
<td>5 sec.</td>
</tr>
<tr>
<td>Deriving the Platform/Mapping</td>
<td>0.03 sec.</td>
</tr>
<tr>
<td>System Synthesis</td>
<td>2.16 sec.</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>~1 day</td>
</tr>
<tr>
<td><strong>Total excluding WCET</strong></td>
<td>~8 sec.</td>
</tr>
</tbody>
</table>
Conclusions

- Daedalus\textsuperscript{RT} provides automatic parallelization of applications together with \textit{fully automatic derivation} of the code generation and analysis models.

- Daedalus\textsuperscript{RT} uses hard-real-time schedulability analysis to derive the platform and mapping specifications in a \textit{very fast yet accurate} manner.

- The throughput resulting from strictly periodic scheduling is equal to the maximum achievable throughput for more than 80% of the applications.
Thank you!

Demonstration @ University Booth
Tuesday 13 March: 1230 - 1430
Thursday 15 March: 1200 - 1400