Exploiting Just-enough Parallelism When Mapping Streaming Applications in Hard Real-time Systems

Teddy Zhai, Mohamed Bamakhrama, Todor Stefanov
Leiden Embedded Research Center,
Leiden Institute of Advanced Computer Science
Leiden University, The Netherlands

tzhai@liacs.nl
Motivation

- Embedded streaming applications
- Both high performance and strict timing requirements

high performance ➞ hard real-time

Collision avoidance

Stereo navigation

need up to 64 processing elements

FP-7 parMerasa project
Significance of Problem

- Synchronous dataflow (SDF) graphs
- Performance gap: task-level parallelism only
- Need to exploit other forms of parallelism

![Graph showing speedup over single core with different number of cores](image-url)
Parallelization in Hard Real-time Systems

- Task-level parallelism + HRT scheduling
- Task + data-level (unfolding) parallelism + HRT scheduling

1) Which actor to unfold?
2) By what factor?
3) Actor allocation?

PE: Processing Element

![Diagram showing parallelization and allocation](attachment:parallelization_diagram.png)
Problem Statement

- Simultaneous actor unfolding and spatial allocation
  - Local temporal scheduling: EDF or RM

- **Find** minimum **unfolding factor** for each actor

- Primal problem
  - **Minimize** guaranteed periods (inverse of throughput), **subject to** #PEs under hard real-time scheduling

- Equivalent (dual) problem
  - **Maximize** total utilization of #PEs under hard real-time scheduling

EDF: Earliest Deadline First
RM: Rate Monotonic
Existing Approach 1

- 2-step approach: first actor unfolding and then allocation
  - Without timing constraints
  - Often excessive parallelism (unfolding too many times)
Existing Approach 2

• Simultaneous actor unfolding and allocation
  • Without timing constraints

• ILP formulation (exact solution)
  • Our problem is non-linear (real-time scheduling)
  • Intractability: takes ca. 70 hours to unfold and map 26 actors on 4 PEs

• Meta-heuristics (e.g.: genetic algorithm):
  • Solution quality: experimental comparison

**Insufficient** parallelism due to inappropriate solution space

ILP: Integer Linear Programming
Design Space of Unfolding Factors

Common assumption:
Maximum unfolding factor = #PEs

Our provable upper bounds
Non-optimal performance

A meaningful design space with impact on performance (period)
Contribution 2

- An efficient algorithm

1) SDF graph
2) #PEs: e.g., 128
3) Allocation algorithm: FFD, ...
4) Scheduling algorithm: EDF, RM, etc.
5) Quality: e.g., utilization 95% x 128 PEs

Algorithm

1) Guaranteed performance
2) Unfolded SDF graph
3) Allocation of actors (replicas)

FFD: First-Fit Decreasing
Experiments

- 11 StreamIT benchmarks (SDF)
  - #actors varies from 8 up to 120, #edges varies from 7 up to 147

- Map benchmarks onto 2 up to 128 PEs
  - Original SDF graphs
  - Unfolded graphs

- Compare against genetic algorithm based heuristics
  - Solution quality (unfolding factors)
  - Time complexity
Comparison with GA-based Meta-heuristic

- Time complexity: on avg. 104X faster
  - 813X faster in FMRadio benchmark
- Code size: on avg. 4X less
  - Smaller unfolding factors

![Graph comparing time complexity and code size between GA-based and new methods](chart.png)
Running time of our Algorithm

- Takes on avg. 4.8S to map a benchmark onto 128 PEs
- Longest running time: 78S for Serpent (120 actors)
Conclusion

- Exploiting just-enough parallelism \textit{w.r.t.} platform constraints
  - \#PEs
  - Scheduling on PEs

- Simultaneous actor unfolding and allocation under hard real-time scheduling
  - Derive better upper bound of unfolding factors (solution space)
  - Propose an efficient algorithm

- Experiments show that our algorithm
  - Results in large performance gain
  - Requires reasonable running time
  - Outperforms GA-based meta-heuristics in terms of solution quality and running time
Thank you for the attention!
Questions?
Unfolding SDF graphs

- Initial SDF graph

![Initial SDF graph diagram]

- Equivalent CSDF graph: unfolding $A_3$ by factor 3
  - Conceptually similar to adding splitter and joiner

![Equivalent CSDF graph diagram]

- Unfolding algorithm described in paper
Upper Bound of Unfolding Factors

- Initial SDF graph (previous slide)
- 2 PEs and EDF scheduling

- Optimal CSDF graph (unfolding $A_3$ by 4, greater than 2)
  - Period of $A_{5,1}$ 18, total utilization 2.0

## Benchmark Characteristics

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Num. of Actors</th>
<th>Num. of Edges</th>
<th>Has Stateful Actors?</th>
</tr>
</thead>
<tbody>
<tr>
<td>DCT</td>
<td>8</td>
<td>7</td>
<td>No</td>
</tr>
<tr>
<td>FFT</td>
<td>17</td>
<td>16</td>
<td>No</td>
</tr>
<tr>
<td>Filterbank</td>
<td>85</td>
<td>99</td>
<td>No</td>
</tr>
<tr>
<td>TDE</td>
<td>29</td>
<td>28</td>
<td>No</td>
</tr>
<tr>
<td>DES</td>
<td>53</td>
<td>60</td>
<td>No</td>
</tr>
<tr>
<td>Serpent</td>
<td>120</td>
<td>128</td>
<td>No</td>
</tr>
<tr>
<td>Bitonic</td>
<td>40</td>
<td>46</td>
<td>No</td>
</tr>
<tr>
<td>MPEG2</td>
<td>23</td>
<td>26</td>
<td>Yes</td>
</tr>
<tr>
<td>Vocoder</td>
<td>114</td>
<td>147</td>
<td>Yes</td>
</tr>
<tr>
<td>FMRadio</td>
<td>43</td>
<td>53</td>
<td>No</td>
</tr>
<tr>
<td>Channel</td>
<td>55</td>
<td>70</td>
<td>No</td>
</tr>
</tbody>
</table>
## GA Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Population size</td>
<td>80</td>
</tr>
<tr>
<td>Number of generations</td>
<td>300</td>
</tr>
<tr>
<td>Crossover rate</td>
<td>0.9</td>
</tr>
<tr>
<td>Mutation rate</td>
<td>0.05</td>
</tr>
<tr>
<td>Mating rate</td>
<td>0.1</td>
</tr>
</tbody>
</table>

Future Work

- Latency and Buffer Size Reduction

- Use Constrained Deadline for each actor
  - Deadline less than period
  - Significant latency and buffer size reduction

- Price to pay
  - More complex algorithm due to expensive QPA schedulability test