Minimizing Buffer Requirements for Throughput Constrained Parallel Execution of Synchronous Dataflow Graph

Seoul National University, CAPLABT
Tae-ho Shin¹, Hyunok Oh² and Soonhoi Ha¹

1: Seoul National University
2: Hanyang University
Contents

- Introduction
  - Motivational Example
  - Related Work
  - Problem Definition

- Proposed Solution
  - Overall Structure
  - Proposed Dynamic Scheduling Method

- Experiments

- Conclusion
Motivational Example

- A (Simple) SDF Graph
  - node: computation block
  - arc: FIFO queue

- Sample rate: number of samples consumed or produced per node firing
- A node is fireable only after it has enough number of samples on all input arcs

- A mapping instance (nodes to processors)

<table>
<thead>
<tr>
<th>Node</th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mapped Processor</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Execution Time</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>
Arc buffer size affects the throughput!

- Scheduling result when the buffer size of arc AB is 4
- Scheduling result when the buffer size of arc AB is 6
Unfolding affects the throughput!

- Motivational Example 2

![Diagram showing a network of nodes A, B, C, D, and E with edges and execution times.]

<table>
<thead>
<tr>
<th>Node</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mapped Processor</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Execution Time</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<Scheduling result without unfolding>

<Scheduling result with 2-unfolding>
All previous work assumed “static scheduling”
The optimization problem is NP-hard
Extensive work has been performed recently – prove that the problem becomes practically important
Dynamic vs Static scheduling

- **Pros of dynamic scheduling over static scheduling**
  - Can get the effect of unfolding naturally
  - Easy to represent of schedule and uses less memory space
  - May improve system performance when the execution times are vary at run-time

- **But we need**
  - Run-time system to schedule the nodes dynamically
  - Priority assignment to the mapped nodes
Problem Definition

- **Input**
  - Target Architecture: A heterogeneous MPSoC
  - Input Information
    - An SDF graph with given execution time of nodes
    - A given static mapping of nodes to processors
    - A known dynamic scheduling policy on each processor
  - Constraints: Throughput

- **Problem**
  - Minimize the total buffer requirement and determine the buffer size of all arcs
  - (Determine the priority of the mapped nodes)
Proposed Solution

- Overall Optimization Flow

1. Generate SDF graph (without size of each buffer)
2. Schedule
3. feasible?
   - yes: fitness value = 1 / sum of buffer size
   - no: Fitness value = 0
GA-based Heuristic

- **JGAP package is used for current implementation**
  - The size of each buffer size is encoded into chromosome and GA evaluate chromosome by scheduling dynamically with encoded buffer size information.

  ![Chromosome Diagram]

  - Fitness value of chromosome is determined by feasibility of scheduling result based on given throughput constraint.
  - Optimization process is repeated until fitness value converges or pre-defined upper bound of generation steps.
Feasibility Analysis

- Simulate the system in which each processor performs dynamic scheduling of the mapped nodes for each candidate solution (given buffer sizes of all arcs)
  - All mapped nodes are assigned priorities
  - We consider the communication overhead between processors as well as execution time variation of the nodes
  - We repeat the execution of the graph until we obtain the throughput
Throughput Computation

- **Approximate throughput**
  
  Since there is no guarantee that the same scheduling pattern will be repeated in dynamic scheduling, the following equation is defined to calculate throughput in dynamic scheduling:

  \[
  T(G) = \lim_{n \to \infty} \frac{n}{\text{time to finish } n \text{ iterations}}
  \]

  If the number of iterations are increased to infinite, the value of equation converges to specific value and it can be considered as throughput.

  In most case, **after 10 iterations** the value converges.
Priority Assignment

- **Proposed heuristic**
  - We assign a different priority of each invocation for a same node
  - To set priority to each node invocation, calculate “as late as possible (ALAP)” scheduling time to sink node as following
    - \( P(N_{\text{last}}) = Ex(N) + \max\{P(K_i)\} \)
      where node \( K \) is in \{successors of node \( N \}\)
    - \( P(N_k) = P(N_{\text{last}}) + (\text{rep}(N) - k) \times Ex(N) \)
  - Optimal assignment is left as a future work
Experimental Results

- Comparison of total buffer size with an optimal solution in [14]

![Graph showing comparison of total buffer size with an optimal solution.](image-url)
Comparison with a pipelined method

- Pipelining is a popular way of throughput improvement
- But pipelining needs pipeline buffers.

<table>
<thead>
<tr>
<th></th>
<th>Throughput</th>
<th>Total buffer size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proposed Method</td>
<td>1/3</td>
<td>6</td>
</tr>
</tbody>
</table>
Scalability of the proposed technique

- Elapsed time with various input sets

<table>
<thead>
<tr>
<th># of instances</th>
<th># of processors</th>
<th># of edges</th>
<th>Throughput constraints</th>
<th>Elapsed time</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>3</td>
<td>5</td>
<td>1 / 100</td>
<td>190 s</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 / 44</td>
<td>192 s</td>
</tr>
<tr>
<td></td>
<td></td>
<td>32</td>
<td>1 / 100</td>
<td>134 s</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 / 34</td>
<td>133 s</td>
</tr>
<tr>
<td>100</td>
<td>7</td>
<td>20</td>
<td>1 / 100</td>
<td>1052 s</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 / 75</td>
<td>1059 s</td>
</tr>
<tr>
<td></td>
<td></td>
<td>54</td>
<td>1 / 100</td>
<td>588 s</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 / 79</td>
<td>665 s</td>
</tr>
</tbody>
</table>
Conclusion

- We propose a static mapping and dynamic scheduling method that has several benefits over static scheduling methods.
- The proposed GA_based algorithm minimizes the buffer requirement under the throughput constraints.
- A simple heuristic for priority assignment is also proposed – produces good results.
- The proposed technique is scalable, while producing near-optimal results.
Future work

- Find an optimal mapping
- Find an optimal priority assignment scheme

Thank you!