Execution Trace Based Multi-Criteria Partitioning of Stream Programs

M. Michalska, S. Casale-Brunet, E. Bezati, M. Mattavelli
SCI-STI-MM Institute of Electrical Engineering
Outline

- Model of computation
- Problem
- Model of execution
- Partitioning algorithm
- Experiments
- Summary & Future works
Model of computation

Dataflow with firings

Actors
- Computational Kernel
- Communication via unbounded queues
- No notion of time
- Atomic execution of Actions
- Encapsulated State Variables

Actions
- Consume tokens
- Produce tokens
- Modify Internal state variables
Attractive features

Dataflow with firings has several advantages over other models:

The program is:
• modular → allows partitioning
• portable → can be easily implemented on different platforms
• expressive → efficiently handles dynamic behaviour
• atomic → no risk of races, unpredictable memory accesses etc.
Objective

How to partition the program onto a given set of processors?

Constraints:

• Maximize the throughput
• Minimize the number of processing units
• Allow implementing an efficient scheduling policy
• Take full advantage of the parallelism of an application

NP-complete problem
Problem statement

Theoretic notation: $P|\text{prec, groups}|C_{\text{max}}$

- Assignment of a set of tasks
- Identical processors with no pre-emption
- Minimized makespan (total execution time)
- For tasks assigned to the same unit only one can be executed at the same time

Some heuristic algorithmic approaches:
- Greedy placement
- Integer programing based methods
- Genetic algorithms

Common optimization criteria:
- Communication cost minimization
- Workload balance
- Multiple properties
Model of execution - Execution Trace Graph

- Every node represents a single action firing
- Every directed edge represents a dependency
- Each dependency defines an implicit execution order

The ETG is generated for a specific input stimulus
→ a way to handle dynamic applications

Types of dependency:
- State Variable (Read/Write or Write/Read)
- Finite State Machine
- Tokens
- Port (Read/Read or Write/Write)
- Guard (Enable/Disable)
Implications of the model

- There are intrinsic dependencies between action firings
- Firings are affiliated to the specific actor → maintained during the partitioning process
- A set of input stimuli must sufficiently cover the whole span of application behaviour
Implications of the model

• There are intrinsic dependencies between action firings
• Firings are affiliated to the specific actor → maintained during the partitioning process
• A set of input stimuli must sufficiently cover the whole span of application behaviour
Heuristic

Task:
Assign the actors to the target set of units

Two modes:
• Fixed number of processing units:

• Estimation of the optimal number of units:

Given input:
• ETG
• weights profiled for a target platform
Metrics

- **Actor Workload (AW):**
  sum of weights assigned to each action firing belonging to a given actor

- **Actor Preceding Workload (APW):**
  maximal sum of weights of each firing of each actor that precedes the given actor A in the network in terms of topological order

- **Actor Common Predecessors (ACP):**
  evaluated for each pair of actors, denoting the number of actors appearing on the topological list of predecessors for both from the pair

Calculation for D: the maximal sum of weights goes through actors E, F and G, therefore APW = 15

For actors C and G: actor A and actor B
Metrics

Indication of workload balance:

- **Average Partitioning Occupancy (APO)**
  average value of processing time of each unit expressed in percent

- **Standard Deviation of Occupancy (SDO)**
  statistical standard deviation for the population of units processing times expressed in percent

*High APO, low SDO → APO/SDO ratio monitored*
Flow of the algorithm

- Initial setup: all actors placed on separate units
- Candidate for reduction: unit with lowest APW
- Attach the candidate/spread its members
- Join the low-APW-partitions with the high-APW ones
- Maintain a high ACP within a partition
- Simulate and compare the performance

1st optimization procedure:
- Identify most idle actor
- Move to the most idle partition

2nd optimization procedure:
- Calculate token exchanges between partitions
- Move the significantly “communicative” actors
Flow of the algorithm

- Initial setup: all actors placed on separate units
- Candidate for reduction: unit with lowest APW
- Attach the candidate/spread its members
- Join the low-APW-partitions with the high-APW ones
- Maintain a high ACP within a partition
- Simulate and compare the performance

1st optimization procedure:
- Identify most idle actor
- Move to the most idle partition

2nd optimization procedure:
- Calculate token exchanges between partitions
- Move the significantly “communicative” actors
Flow of the algorithm

- Initial setup: all actors placed on separate units
- Candidate for reduction: unit with lowest APW
- Attach the candidate/spread its members
- Join the low-APW-partitions with the high-APW ones
- Maintain a high ACP within a partition
- Simulate and compare the performance

1st optimization procedure:
- Identify most idle actor
- Move to the most idle partition

2nd optimization procedure:
- Calculate token exchanges between partitions
- Move the significantly “communicative” actors
Flow of the algorithm

- Initial setup: all actors placed on separate units
- Candidate for reduction: unit with lowest APW
- Attach the candidate/spread its members
- Join the low-APW_partitions with the high-APW ones
- Maintain a high ACP within a partition
- Simulate and compare the performance

1st optimization procedure:
- Identify most idle actor
- Move to the most idle partition

2nd optimization procedure:
- Calculate token exchanges between partitions
- Move the significantly “communicative” actors
Flow of the algorithm

- Initial setup: all actors placed on separate units
- Candidate for reduction: unit with lowest APW
- Attach the candidate/spread its members
- Join the low-APW-partitions with the high-APW ones
- Maintain a high ACP within a partition
- Simulate and compare the performance

1st optimization procedure:
- Identify most idle actor
- Move to the most idle partition

2nd optimization procedure:
- Calculate token exchanges between partitions
- Move the significantly “communicative” actors
Flow of the algorithm

- Initial setup: all actors placed on separate units
- Candidate for reduction: unit with lowest APW
- Attach the candidate/spread its members
- Join the low-APW-partitions with the high-APW ones
- Maintain a high ACP within a partition
- Simulate and compare the performance

1st optimization procedure:
- Identify most idle actor
- Move to the most idle partition

2nd optimization procedure:
- Calculate token exchanges between partitions
- Move the significantly “communicative” actors
Flow of the algorithm

- Initial setup: all actors placed on separate units
- Candidate for reduction: unit with lowest APW
- Attach the candidate/spread its members
- Join the low-APW-partitions with the high-APW ones
- Maintain a high ACP within a partition
- Simulate and compare the performance

1st optimization procedure:
- Identify most idle actor
- Move to the most idle partition

2nd optimization procedure:
- Calculate token exchanges between partitions
- Move the significantly “communicative” actors
Flow of the algorithm

- Initial setup: all actors placed on separate units
- Candidate for reduction: unit with lowest APW
- Attach the candidate/spread its members
- Join the low-APW-partitions with the high-APW ones
- Maintain a high ACP within a partition
- Simulate and compare the performance

1st optimization procedure:
- Identify most idle actor
- Move to the most idle partition

2nd optimization procedure:
- Calculate token exchanges between partitions
- Move the significantly “communicative” actors
Flow of the algorithm

- Initial setup: all actors placed on separate units
- Candidate for reduction: unit with lowest APW
- Attach the candidate/spread its members
- Join the low-APW-partitions with the high-APW ones
- Maintain a high ACP within a partition
- Simulate and compare the performance

1st optimization procedure:
- Identify most idle actor
- Move to the most idle partition

2nd optimization procedure:
- Calculate token exchanges between partitions
- Move the significantly “communicative” actors
Flow of the algorithm

• Initial setup: all actors placed on separate units
• Candidate for reduction: unit with lowest APW
• Attach the candidate/spread its members
• Join the low-APW-partitions with the high-APW ones
• Maintain a high ACP within a partition
• Simulate and compare the performance

1st optimization procedure:
• Identify most idle actor
• Move to the most idle partition

2nd optimization procedure:
• Calculate token exchanges between partitions
• Move the significantly “communicative” actors
Flow of the algorithm

• Initial setup: all actors placed on separate units
• Candidate for reduction: unit with lowest APW
• Attach the candidate/spread its members
• Join the low-APW-partitions with the high-APW ones
• Maintain a high ACP within a partition
• Simulate and compare the performance

1st optimization procedure:
• Identify most idle actor
• Move to the most idle partition

2nd optimization procedure:
• Calculate token exchanges between partitions
• Move the significantly “communicative” actors
Flow of the algorithm

- Initial setup: all actors placed on separate units
- Candidate for reduction: unit with lowest APW
- Attach the candidate/spread its members
- Join the low-APW-partitions with the high-APW ones
- Maintain a high ACP within a partition
- Simulate and compare the performance

1st optimization procedure:
- Identify most idle actor
- Move to the most idle partition

2nd optimization procedure:
- Calculate token exchanges between partitions
- Move the significantly “communicative” actors
Methodology of experiments

Tested designs:

- JPEG decoder
  - ✓ 6 actors
  - ✓ 45 actions
  - ✓ for 5 frames QCIF Foreman sequence: 227 156 firings

- MPEG4-SP decoder
  - ✓ 34 actors
  - ✓ 174 actions
  - ✓ for 5 frames QCIF Foreman sequence: 283 287 firings

Platform: 4-cores Intel i7-3770 3.4 GHz
## Experimental results

### JPEG

<table>
<thead>
<tr>
<th>No. of units</th>
<th>Basic [fps]</th>
<th>+Min-Idle [fps]</th>
<th>+Min-Comm [fps]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2217</td>
<td>2217</td>
<td>2217</td>
</tr>
<tr>
<td>2</td>
<td>3631</td>
<td>2239</td>
<td>3462</td>
</tr>
<tr>
<td>3</td>
<td>3843</td>
<td>2085</td>
<td>3298</td>
</tr>
<tr>
<td>4</td>
<td>3481</td>
<td>3498</td>
<td>3448</td>
</tr>
</tbody>
</table>

### MPEG

<table>
<thead>
<tr>
<th>No. of units</th>
<th>Basic [fps]</th>
<th>+Min-Idle [fps]</th>
<th>+Min-Comm [fps]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>536</td>
<td>536</td>
<td>536</td>
</tr>
<tr>
<td>2</td>
<td>868</td>
<td>984</td>
<td>1015</td>
</tr>
<tr>
<td>3</td>
<td>1327</td>
<td>1332</td>
<td>1075</td>
</tr>
<tr>
<td>4</td>
<td>1130</td>
<td>1222</td>
<td>1036</td>
</tr>
</tbody>
</table>
Summary

• Partitioning heuristic outperformed the referenced solution
• Difficult to compare with other software - no conformance with our model
• Satisfactory values of speed-up:
  ✓ for 2 cores - 1.9 (MPEG)
  ✓ for 3 cores - 2.5 (MPEG)
• Model of execution based on the analysis of the ETG
• Single profiling and trace generation - multiple extraction of properties and simulation of the performance
Future work

• Optimizations of the heuristic (performance, reliability)
• Implementation of a more robust scheduling policy
• Introduction of further target functions to the model:
  ✓ communication cost minimization
  ✓ memory access optimization
  ✓ monitoring of the shared caches occupancy
Thank you.