Multi-Objective Aware Extraction of Parallelism for Embedded Systems

Daniel Cordes
TU Dortmund University
Department of Computer Science 12
Dortmund
Germany
Outline

1. Introduction
2. Summary of previous approaches
3. GA-based, Multi-Objective Aware Parallelization approach
4. Results
5. Conclusion & Future Work
Introduction

- Nowadays multiple cores available in embedded MPSoCs
  - most embedded applications written in sequential C code
  - manual parallelization error-prone & time-consuming

- Many (semi-) automatic parallelization tools exist
  - only very limited applicability for embedded devices
  - written for desktop- or high-performance computers

- Restrictions of embedded architectures
  - low computational power
  - expensive communication structure
  - small memories
  - battery-driven
  - ...

→ Parallelization tools should be aware of this
Summary of previous presentations

- Already presented two extraction techniques @Rheinfels

- coarse-grained Task-Level Parallelization approach (2010)

- more fine-grained Pipeline Parallelization approach (2011)
Task-Level Parallelization approach

1. Extract hierarchical graph
2. Parallelize nodes bottom-up
3. Transform to ILP
4. Solve ILP for different task limitations
5. Transform solutions to HTG
6. Attach results and continue with other nodes

D. Cordes, P. Marwedel, A. Mallik:
Automatic parallelization of embedded software using hierarchical task graphs and integer linear programming,
In Proc. of CODES/ISSS 2010
Pipeline Parallelization approach

1. Extract PDG
2. Create Sub-PDG for Loop(-nest)
3. Transform to ILP
   \[ \begin{align*}
   (1) \quad & \min \sum x_i \cdot c_i \\
   (2) \quad & \sum b_i \cdot x_i \leq C \\
   (3) \quad & \forall x_i \leq 1 \\
   (4) \quad & \forall x_i \geq 0
   \end{align*} \]
4. Solve ILP for different task limitations
5. Collect best solution candidates
6. Attach results and continue with other loops
7. Combine solution candidates of loops to optimal result
8. Annotate Source Code
9. Implement parallelism

D. Cordes, A. Heinig, P. Marwedel:
Automatic Extraction of Pipeline Parallelism for Embedded Software Using Linear Programming, Proc. of ICPADs 2011
Characteristics of previous approaches

- Input: Sequential C-Code
- Output: Parallelized application
- Used techniques:
  - Hierarchical Task Graph (task-level)
  - Program Dependence Graph (pipeline)
  - Integer Linear Programming (ILP)
- Integer linear programming:
  - System of linear inequalities
  - Solution optimal with respect to the used model
  - Task-overhead and communication costs modeled in ILP
Characteristics of previous approaches

- **Input:** Sequential C-code
- **Output:** Parallelized application
- **Used techniques:**
  - Hierarchical Task Graph (task-level)
  - Program Dependence Graph (pipeline)
  - Integer Linear Programming (ILP)

- **Integer linear programming:**
  - System of linear inequalities
  - Solution optimal with respect to the used model
  - Task-overhead and communication costs modeled in ILP

- Previous approaches perform well for many embedded applications
- However, how can we optimize applicability for embedded systems even more?
- Consider multiple objectives (like, e.g., execution time, energy consumption) while extracting parallelism
- Many mapping tools do this, but no parallelization tool performs like this
- Problem: ILPs only optimize for one objective

→ **Solution:** Use Genetic Algorithms (GA) to extract multi-objective aware parallelism
GA-Approach: Ideas & Characteristics*

- Start with multi-objective aware task-level extraction approach. Consider the following objectives:
  - execution time
  - energy consumption
  - communication overhead

- Re-Use Hierarchical Task Graph as employed intermediate representation

- Use high level models for fast evaluation of solution candidates
  - A lot of solutions can be generated and evaluated

GA-Approach: Hierarchical Task Graph

- Hierarchical Task Graph very useful to extract Task-Level Parallelism
- Hierarchical structure of graph equal to structure of application
- E.g., loops create new hierarchical levels
- Nodes and edges annotated with cost information
- Each hierarchical level processed in isolation due to encapsulated communication
GA-Approach: Tool Flow

1. Start to parallelize inner-most hierarchical nodes
2. Attach Pareto-optimal results to hierarchical nodes
3. Go upwards in hierarchy
4. Create new tasks + recombine
5. Return final Pareto-front to user
GA-Approach: Chromosome Representation

- **First part of gene:** Node2Task Mapping
- Mapping influences dependencies
- **Second part:** Hierarchical solution
- One solution chosen for each node
- Directly influences objectives

### Chromosome Representation

- **Node-to-Task Mapping**
  - \( T_1 \) to \( N_1 \)
  - \( T_1 \) to \( N_2 \)
  - \( T_2 \) to \( N_3 \)
  - \( T_3 \) to \( N_4 \)
  - \( T_3 \) to \( N_5 \)
  - \( T_4 \) to \( N_6 \)
  - \( T_4 \) to \( N_7 \)

### Task Graph

- \( T_1 \) to \( N_1 \)
- \( T_3 \) to \( N_2 \)
- \( T_2 \) to \( N_3 \)
- \( T_4 \) to \( N_5 \)

#### Selected Parallel Solution for Node \( N_1 \)

- \( S_{1,4} \) to \( N_1 \)

#### Node \( N_1 \) (Selected Point \( S_4 \)):
- Execution costs: 162 cycles
- Energy consumption: 0.16 nJ
- Communicated data: 200
- Number hierarchical tasks: 3

### Hierarchical Parallel Solutions (Pareto-frontiers)

- \( S_{1,6} \) to \( N_1 \)
- \( S_{2,7} \) to \( N_2 \)
- \( S_{3,5} \) to \( N_3 \)
- \( S_{4,9} \) to \( N_4 \)
- \( S_{5,9} \) to \( N_5 \)
- \( S_{6,7} \) to \( N_6 \)
- \( S_{7,7} \) to \( N_7 \)

### GA-Approach: Chromosome Representation

- **Chromosome Representation**
- **Task Graph**
- **Hierarchical Parallel Solutions**
GA-Approach: Results

- **Edge detect** benchmark from **UTDSP** benchmark suite
- Many Pareto-optimal solutions
- 1.6x speedup → energy: 220%
- Max speedup → energy: 340%
- Many options for „best“ solution
- Target architecture: MPARM

- Pareto-front is based on model
- Model vs. simulated values →
- Both curves very similar
- Trend of exec. time and energy consumption validated
GA-Approach: Results

- Matrix multiplication benchmark from UTDSP benchmark suite
- Less Pareto-optimal solutions
- Many dominated solutions
- Exec. Time ranges up to 2.9x
- Energy between 100% - 300%

Model vs. Simulated values →
Trend of exec. time and energy consumption similar, as well
### GA-Approach: Results - GA

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Time*</th>
<th>#Nodes</th>
<th>#Populations</th>
<th>#Individuals</th>
<th>#Mutations</th>
<th>#Crossover</th>
<th>#Solut.</th>
</tr>
</thead>
<tbody>
<tr>
<td>adpcm enc.</td>
<td>00:49</td>
<td>27</td>
<td>1,376</td>
<td>146,003</td>
<td>27,164</td>
<td>91,616</td>
<td>5</td>
</tr>
<tr>
<td>bound. value</td>
<td>01:04</td>
<td>6</td>
<td>532</td>
<td>79,983</td>
<td>15,122</td>
<td>50,492</td>
<td>4</td>
</tr>
<tr>
<td>compress</td>
<td>09:07</td>
<td>131</td>
<td>6,552</td>
<td>592,124</td>
<td>110,426</td>
<td>371,048</td>
<td>5</td>
</tr>
<tr>
<td>edge detect</td>
<td>02:08</td>
<td>2,088</td>
<td>175,203</td>
<td>32,506</td>
<td>109,528</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>filterbank</td>
<td>01:36</td>
<td>212</td>
<td>22,198</td>
<td>4,223</td>
<td>14,012</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>fir 256</td>
<td>00:18</td>
<td>7</td>
<td>292</td>
<td>4,883</td>
<td>16,620</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>iir 4</td>
<td>00:51</td>
<td>7</td>
<td>564</td>
<td>82,224</td>
<td>186,988</td>
<td>33</td>
<td></td>
</tr>
<tr>
<td>jpeg 2000</td>
<td>04:36</td>
<td>43</td>
<td>2,468</td>
<td>4,883</td>
<td>120,084</td>
<td>42</td>
<td></td>
</tr>
<tr>
<td>latnrm 32</td>
<td>00:13</td>
<td>11</td>
<td>460</td>
<td>37,122</td>
<td>23,172</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>mult 10</td>
<td>00:14</td>
<td>10</td>
<td>404</td>
<td>34,511</td>
<td>21,500</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>spectral</td>
<td>01:36</td>
<td>38</td>
<td>1,948</td>
<td>190,266</td>
<td>120,084</td>
<td>33</td>
<td></td>
</tr>
</tbody>
</table>

* Overall 9 minutes: 0.9 milli sec. / individual

* Number of solutions range from 4 - 42

* Over ½ million created and evaluated individuals

* AMD Opteron @ 2.4 GHz
Conclusions & Future Work

Conclusion

- Presented different parallelization approaches for embedded systems
- Previous ILP-based approaches
- Multi-objective aware approach
- Huge optimization potential for multi-objective aware parallelization

Future Work

- Extend for heterogeneous architectures
- Adapt ILP-based pipeline parallelism approach to be multi-objective aware

* Already done. Paper is currently in review phase.
Thank you for your attention!

Questions?