SDG2KPN: System Dependency Graph to Function-level KPN generation of Legacy Code for MPSoCs

Jude Angelo Ambrose, Jorgen Peddersen, Alvin Labios, Yusuke Yachide, Sri Parameswaran
Legacy Code Usage

- Legacy codes/systems
  - Around 70% of the people claim that they have more than 50% legacy systems
    
    ![Bar Chart](image)

    **Figure 1. Percentage of IT systems states label as “Legacy Systems,” N=29**

- However, porting the existing code for other platforms with effective performance is much difficult because of unfamiliar code
  - Challenge is to parallelise the legacy code which was not designed for parallelism

*Source: NASCIO’s 2008 National Survey on Legacy Systems and Modernization in the States*
Traditional Processing
Traditional Processing

TASK

TASK

PROCESSOR

TASK

TASK
Software Balancing using Pipeline
Choosing Hardware Configurations
Hardware Balancing for Pipeline
Overview

• Our goal

• Related work

• SDG2KPN Methodology

• Experiments

• Conclusion
Our Goal: Legacy Code to Function-level KPN to Pipeline MPSoC

Function call sites to convert to KPN and then map to individual processors.
Related Work

• **KPNGen from the Daedalus framework** (Nikolov et al., DAC’08)
  - ✓ KPN is generated by analysing the source code
  - × manual modification required to the source code
  - × code and MPSoC platform not automatically generated
  - × shared variables not supported

• **COMPAAN** (Kienhuis et al., CODES’00)
  - ✓ a commercial tool, generating multicore code
  - × supporting only Affine Nested Loop Programs using Polyhedral technique
  - × shared variables, such as globals and pointers are not supported

Our SDG2KPN methodology utilizes a rule-based traversal (static analysis) of the SDG of any legacy code to find dependencies between functions, supporting analysis of shared variables which was hitherto not supported.
SDG2KPN Flow

Legacy Code → Abtractor → System Dependency Graph (SDG) → Networker

Kahn Process Network (KPN) → Mapper → Target Generator → MPSoc System

Balance Checker → Load Annotated KPN → Annotator → Simulator

Graph Optimizer → Design Constraints → Tuner → Final MPSoc System

Application Input Data
SDG Creation

Legacy Code → Abstractor → System Dependency Graph (SDG)

Input Legacy Code

```c
int main()
{
    int a=20, h, e;
    for(i=0;i<20;i++)
    {
        h=10;
        e=add(a, &h);
        sub(e, a, &h);
    }
}

void add(int c, int *d)
{
    *d = *d+10 + c;
}

void sub(int e, int f, int *g)
{
    *g = *g+f+e;
}
```

System Dependency Graph
KPN Generation

- System Dependency Graph (SDG)
- Traverser
- Function Mapper
- User Specifications
- Traversal Rules
- Networker
- Binder
- Kahn Process Network (KPN)
User Specifies the functions to be converted to KPN

User Specifications

Function Mapper

Traverser

Binder

Networker

Rule: actual-in links to an intra-predecessor, which is an actual-out

Dependency: return variable passed as parameter
# Experiments – Rule Evaluation

<table>
<thead>
<tr>
<th>Apps</th>
<th>No. KPN Nodes</th>
<th>No. of Rules</th>
<th>Generation Time (sec)</th>
<th>Estimated Manual Gen.</th>
<th>LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES</td>
<td>2</td>
<td>16</td>
<td>4.8</td>
<td>2 days</td>
<td>950</td>
</tr>
<tr>
<td>MPEG-2 (enc)</td>
<td>10</td>
<td>97</td>
<td>21.4</td>
<td>2 months</td>
<td>8k</td>
</tr>
<tr>
<td>MJPEG</td>
<td>6</td>
<td>75</td>
<td>8.0</td>
<td>3 weeks</td>
<td>2k</td>
</tr>
<tr>
<td>H.264 (enc)</td>
<td>9</td>
<td>1064</td>
<td>164.9</td>
<td>6 months</td>
<td>58k</td>
</tr>
<tr>
<td>ADPCM (enc)</td>
<td>6</td>
<td>27</td>
<td>4.3</td>
<td>1 week</td>
<td>285</td>
</tr>
</tbody>
</table>
## Experiments – MPSoC Executions

<table>
<thead>
<tr>
<th>Apps</th>
<th>No. of Processors</th>
<th>Latency KPN (Mcycles)</th>
<th>Latency single (Mcycles)</th>
<th>Power KPN (mW)</th>
<th>Power single (mW)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES</td>
<td>2</td>
<td>131</td>
<td>135</td>
<td>180.54</td>
<td>123.34</td>
</tr>
<tr>
<td>MJPEG (f)</td>
<td>6</td>
<td>445</td>
<td>536</td>
<td>494.39</td>
<td>113.55</td>
</tr>
<tr>
<td>MJPEG (mb)</td>
<td>6</td>
<td>508</td>
<td>536</td>
<td>367.02</td>
<td>113.55</td>
</tr>
<tr>
<td>ADPCM (enc)</td>
<td>6</td>
<td>224</td>
<td>113</td>
<td>579.19</td>
<td>131.25</td>
</tr>
</tbody>
</table>

- **AES**: No speedup due to feedbacks
Conclusion

- a novel SDG to KPN conversion methodology is proposed

- a rule based traversal of the SDG is performed to create dependencies of variables across functions

- all the variable constructs of the legacy code, including shared variables such as globals and pointers, are supported, which was hitherto not possible.
THANK YOU