Early Performance-Cost Estimation of Application-Specific Data Path Pipelining

Jelena Trajkovic
Computer Science Department
École Polytechnique de Montréal, Canada
Email: jelena.trajkovic@polymtl.ca

Daniel D. Gajski
Center for Embedded Computer Systems
University of California, Irvine, USA
Email: gajski@uci.edu
Introduction

- Application-specific processors (ASPs) are used for optimized implementation of embedded systems

- Problem: determine the optimal pipeline configuration for a data path
  - Automatically estimating the application execution time
    - Estimated clock cycle length * estimated number of cycles

- Compute the cost of each pipelined design

- Our estimation enables fast, accurate and early analysis of different pipeline configurations
  - Each design is characterized by performance and cost
  - No need to create prototype or execute cycle-accurate model for each design

- High fidelity of performance estimation
Related Work

- Application Specific Instruction-set (IS) Processor and IS extension processors (Xtensa and Stretch processor)
  - Designer configures processor, including its pipeline
  - Based on SW profiling, not pipeline-specific estimation

- C-to-RTL tools (Catapult Synthesis)
  - Create data path and insert pipeline stages ‘on the fly’
  - Limited to small code size

- Our technique
  - Automatically produces performance and cost for early trade-off analysis
  - Can handle any size of C code (10K lines of C code)
  - Complementary to use of custom IS extensions and HW accelerators
Estimating the Number of Pipeline Stages

- Pipelining: partially overlap execution of multiple instructions
  - Increases throughput, not instruction execution time
  - Reduces clock cycle length
  - Ideal case: speedup factor = # pipeline stages

- Practical constraints:
  1. Can not divide a data path into equal parts
  2. Pipeline registers are required $\Rightarrow$ overhead to cycle time ($T_{clk}$)
  3. Dependences between adjacent instructions prevent partial overlapping

- #1 and #2 $\rightarrow$ Impact on clock cycle time ($T_{clk}$)
- #3 $\rightarrow$ Influence the number of execution cycles
Pipelining - Tool Flow

- **Inputs:**
  - (Optimized) non-pipelined data path
  - Application code
  - Component Library
    - Xilinx, but may be any other

- **Estimate:**
  - Execution time $T_{\text{exe}}$
    - Cycle length $T_{\text{clk}}$
    - Number of cycles
  - Cost
    - BRAM + Slices
Candidate Pipeline Configurations

- Uniform pipelining
  - Non-pipeline
  - Two stage
  - Three stage
  - Four stage

Jelena Trajkovic & Daniel D. Gajski © 2010
Cycle Time $T_{clk}$ Estimation

- Component is annotated with
  - Propagation delay
  - Setup time

- Eg: four stage pipelining
  \[ T_{clk} = \max \{ T_{clk1}, T_{clk2}, T_{clk3}, T_{clk4}, T_{clk5} \} \]

  \[ T_{clk1} = \max (RF.\text{prop \_ delay}) + \max (\text{interconnect.src}) + \max (\text{pipe \_ reg(T1). setup time}) \]
Number of Cycles Estimation

- Estimating code schedule
- Example:
  - Data path with Input pipelining
  - Memory access – 2 cycles
  - Add – 1 cycle
- Memory access time ⇒ stall cycle
- Estimated number of extra cycles compared to the non-pipelined

3-address code

Assumptions
**Experimental Results**

<table>
<thead>
<tr>
<th>Application</th>
<th>LoC</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sort</td>
<td>33</td>
<td>Bubble sort algorithm implementation</td>
</tr>
<tr>
<td>Bdist2</td>
<td>61</td>
<td>Part of motion estimation function from MPEG2 encoder</td>
</tr>
<tr>
<td>inv/forw 4x4</td>
<td>95</td>
<td>Integer transformation functions from video H.264 codecs</td>
</tr>
<tr>
<td>dct32</td>
<td>1006</td>
<td>DCT transformation implementation from Mp3</td>
</tr>
<tr>
<td>Mp3</td>
<td>13898</td>
<td>Reference Mp3 decoder implementation</td>
</tr>
</tbody>
</table>

- Mp3 – initial data path:
  - register file: 3 input/6 output ports, with 128 registers
  - 2 ALUs, 1 multiplier, 1 comparator and 1 divider

- Target architecture: NISC
  - No Instructions ⇒ no decoding step ⇒ compiler creates set of control signals (control words)
  - Our technique is applicable to any ASP design

Jelena Trajkovic & Daniel D. Gajski © 2010
Experimental Results – Mp3

- **Estimation error:**
  - Avg: 20%
  - Max: 33%

  ⇒ **Fidelity**
  - 94%
  - (71 out of 75)
  - 5 benchmarks, 6 configs each

- **Evaluation metric: Fidelity**
  - Estimated \((c_A)\) > Estimated \((c_B)\) ⇒ Measured \((c_A)\) > Measured \((c_B)\)
Conclusions:

Contributions:
- Developed automatic method for estimating application execution time and cost for different data path configurations from C code
- Implemented tool for automated early performance/cost estimation

Benefits
- Scalable to any size of C code
- Evaluates pipeline configurations in fraction of time compared to manual design, simulation and synthesis
  - Significantly reduces the # of configurations to be tested

Future work
- Support for forwarding and non-uniform pipeline designs
Acknowledgements

- Prof. Gabriela Nicolescu (École Polytechnique de Montréal)
  - For constructive feedback on this paper

- Center for Embedded Computer Systems
  - For supporting this research project
Thank You

- Questions