Processor Reconfiguration through Instruction Set Metamorphosis

Presented by: Easwar Hariharan
ECGR 6185: Advanced Embedded System Design
Overview

- Introduction and Background
- Processor Reconfiguration through Instruction Set Metamorphosis (PRISM): The approach
- PRISM-I: The prototype
- Experiments and Results
- Q & A
Introduction and Background

- General Purpose Processors
  - Examples: desktop machines, laptops, etc
  - Acceptable performance on wide variety of tasks
- Application Specific Architectures
  - Examples: ASIC, Embedded Systems
  - High performance on specific tasks
- How can we get high performance on a wide variety of tasks?
Introduction and Background (contd)

- Hook: Maximum amount of execution time is spent on small amounts of code
- Amdahl's law: Make common case fast
- Applications in computationally intensive tasks:
  - Examples:
    - Scientific computation
      - SETI@Home, Folding@Home
    - Video editing
    - Gaming
Introduction and Background (contd)

Why so far unfeasible

- Burden on programmers
- Hardware expertise
- Multiple languages

Why now

- Integration of identification and synthesis possible
- 10 – 2,700 fold speedups
- Reaching limit of extracting performance by conventional methods
- Appropriate languages and hardware platforms now available
PRISM: The approach

- Modifying the compiler
  - Called “configuration compiler”
  - Accepts normal C program file
  - Outputs hardware image and software image
- Hardware image
  - Consists of physical specifications to be mapped
- Software image
  - Consists of normal object code
PRISM: The approach (contd)

- Hardware synthesized functions act as native capabilities
- Needs:
  - Close integration of reconfigurable resource with CPU
  - High data bandwidth between reconfigurable resource and CPU
- How it works:
  - Input arguments are passed from CPU to FPGA
  - Function is evaluated in hardware
  - Outputs are returned via bus
PRISM-I: The prototype

- Configuration compiler
  - Language: C
  - Why?
    - Widely accepted
    - Regularly used for programming computationally intensive tasks

- Stages:
  - Function identification and extraction
  - Hardware image synthesis
  - Access/Merge redefinition
  - Compiler optimizations
PRISM-I: The prototype

Figure 2. High-level block diagram illustrating the complete PRISM-I configuration compilation process.
PRISM-I: The prototype (contd)

- Function identification and extraction
  - Separate hardware candidates from software components
  - Selection criteria: Greatest impact on performance
  - Requires estimating runtime behavior
    - Modeling hardware architecture and instruction set
    - Modeling conditional execution with probability
    - Rule based analysis
- Limitations
  - Runtime data dependencies
  - Branch prediction
  - Loop iteration counts
PRISM-I: The prototype (contd)

- Hardware image synthesis
  - Syntax check
  - Parsing
    - Converts function description to manipulable representation
    - Creates software definitions to link function calls to hardware
  - Data Flow Graph (DFG)
  - Hardware structures from libraries to instantiate elements
  - Rule based analysis converts to boolean equations
  - Partitioning/Decomposition
    - Optimizes boolean equations
  - Netlist generation, place and route handled by Xilinx utilities
PRISM-I: The prototype

Figure 3. Detail flow diagram for the hardware-image synthesis in the PRISM-I configuration compilation process.
PRISM-I: The prototype (contd)

- Access/Merge redefinition
  - Merges access definition file with compiled program files
- Compiler optimizations
  - Produces software image file
Experiments and Results

- Compilation time: 15 minutes
- Speedups over software-only evaluation from 2.9 to 54
- Caveats:
  - Hindered by slow interface (16 bit)
  - Limited implementation
    - No floating point capabilities
    - All inputs through arguments
    - Arguments and results limited to 32 bits
## Results

Table 1. Compilation and performance results of functions from the PRISM-1 compiler running on a Sun Sparc IPC workstation. Speedup factors represent the improvement of executing on a 10-MHz M68010-based Armstrong node with PRISM-I versus executing on the node without PRISM-I. Compilation times do not include target place-and-route times.

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Description (input bytes/output bytes)</th>
<th>Compilation Time (min.)</th>
<th>Percent Utilization of XC3090 FPGA</th>
<th>Speedup Factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hamming(x, y)</td>
<td>Hamming metric calculation (4/2)</td>
<td>6</td>
<td>38</td>
<td>24</td>
</tr>
<tr>
<td>Bitrev(x)</td>
<td>Bit-reversal function (4/4)</td>
<td>2</td>
<td>0</td>
<td>26</td>
</tr>
<tr>
<td>Neuron(x, y)</td>
<td>Cascadable 4-input n-net function (4/4)</td>
<td>12</td>
<td>52</td>
<td>12</td>
</tr>
<tr>
<td>MultAccm(x, y)</td>
<td>Multiply/accumulate function (4/4)</td>
<td>11</td>
<td>58</td>
<td>2.9</td>
</tr>
<tr>
<td>LogicEv(x)</td>
<td>Logic-simulation engine function (4/4)</td>
<td>12</td>
<td>40</td>
<td>18</td>
</tr>
<tr>
<td>ECC(x, y)</td>
<td>Error-correction coder/decoder (3/2)</td>
<td>6</td>
<td>14</td>
<td>24</td>
</tr>
<tr>
<td>Find_first_1(x)</td>
<td>First “1” in input locater (4/1)</td>
<td>3</td>
<td>11</td>
<td>42</td>
</tr>
<tr>
<td>Piecewise(x)</td>
<td>Five-section piecewise linear segmentation (4/4)</td>
<td>24</td>
<td>77</td>
<td>5.1</td>
</tr>
<tr>
<td>ALog2(x)</td>
<td>Base-2 A*log(x) computation (4/4)</td>
<td>16</td>
<td>74</td>
<td>54</td>
</tr>
</tbody>
</table>
Questions?