Custom Instructions with Multiple Exits: Generation and Execution

Hamid Noori, Farhad Mehdipour, Koji Inoue, Kazuaki Murakami, and Maziar Goudarzi

Kyushu University

January 23, 2007
Outline

○ Introduction
○ Overview of ADEXOR Architecture
○ Generating Multi-Exit Custom Instructions
  ● Motivation Example
  ● Tool Chain
  ● Generating MECIs
○ Proposing an Architecture for the CRFU
  ● Design Methodology
  ● Supporting Conditional Execution
  ● Proposed Architecture
  ● Integrating the CRFU with the base processor
○ Experiment Results
○ Future Work
Outline

- **Introduction**
- Overview of ADEXOR Architecture
- Generating Multi-Exit Custom Instructions
  - Motivation Example
  - Tool Chain
  - Generating MECIs
- Proposing an Architecture for the CRFU
  - Design Methodology
  - Supporting Conditional Execution
  - Proposed Architecture
  - Integrating the CRFU with the base processor
- Experiment Results
- Future Work
Introduction

- **ADaptive EXtensible ProcessOR (ADEXOR)**
  - Generating and adding custom instructions after chip-fabrication
  - Replacing *custom functional units* with a *reconfigurable functional unit (CRFU)*
    - Supporting adaptable custom instructions
    - Supporting more custom instructions
  - Custom instructions have single entry but multiple exits.
Outline

- Introduction
- Overview of ADEXOR Architecture
- Generating Multi-Exit Custom Instructions
  - Motivation Example
  - Tool Chain
  - Generating MECIs
- Proposing an Architecture for the CRFU
  - Design Methodology
  - Supporting Conditional Execution
  - Proposed Architecture
  - Integrating the CRFU with the base processor
- Experiment Results
- Future Work
Overview of ADEXOR Architecture (1/2)

- Main components
  - Base processor
    - 4-issue in-order RISC processor
  - Reconfigurable functional units (CRFU)
    - Coarse grain
    - Based on matrix of functional units (FUs)
    - Multi-cycle
    - Parallel with other functional units of the base processor
    - Read/write from/to register file
    - Functions and connections are controlled by configuration bits
  - Configuration memory
    - To keep the configuration data of the CRFU for multi-exit custom instructions (MECIs)
  - Counters
    - Control read/write ports of register file and select between CRFU and processor functional units
Overview of ADEXOR Architecture (2/2)
Outline

- Introduction
- Overview of ADEXOR Architecture
- Generating Multi-Exit Custom Instructions
  - Motivation Example
  - Tool Chain
  - Generating MECIs
- Proposing an Architecture for the CRFU
  - Design Methodology
  - Supporting Conditional Execution
  - Proposed Architecture
  - Integrating the CRFU with the base processor
- Experiment Results
- Future Work
Motivating Example (adpcm)

<table>
<thead>
<tr>
<th>Name of the block</th>
<th>No. of Exe. (M)</th>
<th>No. of Instr</th>
</tr>
</thead>
<tbody>
<tr>
<td>B1</td>
<td>11.6</td>
<td>5</td>
</tr>
<tr>
<td>B2</td>
<td>5.8</td>
<td>1</td>
</tr>
<tr>
<td>B3</td>
<td>5.8</td>
<td>4</td>
</tr>
<tr>
<td>B4</td>
<td>8.6</td>
<td>3</td>
</tr>
<tr>
<td>B5</td>
<td>5.2</td>
<td>3</td>
</tr>
<tr>
<td>B6</td>
<td>5.6</td>
<td>1</td>
</tr>
<tr>
<td>B7</td>
<td>5.8</td>
<td>2</td>
</tr>
<tr>
<td>B8</td>
<td>11.6</td>
<td>2</td>
</tr>
<tr>
<td>B9</td>
<td>11.6</td>
<td>6</td>
</tr>
<tr>
<td>B10</td>
<td>11.6</td>
<td>2</td>
</tr>
<tr>
<td>B11</td>
<td>11.6</td>
<td>4</td>
</tr>
<tr>
<td>B12</td>
<td>5.8</td>
<td>3</td>
</tr>
</tbody>
</table>
Multi-Exit Custom Instructions
Tool chain

Instruction Set Simulator (Simplescalar) → Profiler → Detecting Start Address of HBBs → Reading HBBs from Object code → Linking HBBs using Profiling Information and Generating HISs

- Integrated Framework (Partitioning and Mapping MECIs)
- Updating CDFG and Binary rewriting
- Generating Multi-Exit Custom Instructions
- Generating CDFG for HISs
Generating Hot Instruction Sequence

1. MECIs should not cross loop boundaries
2. Sorting loop from innermost to outermost
3. Reading HBBs and linking them to generate Hot Instruction Sequence
4. Sort other remaining HBBs in ascending order considering their start address

Function MAKE_HIS (objfile, HIS, start_addr)
1 if (HBB with start_addr is not included in previous MECIs) then read_add_HBB2HIS (objfile, HBB(start_addr), HIS) else return;
2 switch last_instruction(HBB)
3 case (indirect jump, return or call): return;
4 case (direct jump):
   MAKE_HIS(objfile, HIS, target address of jump);
5 case (branch):
5-1 if (it is hot backward) then return;
5-2 elsif (not-taken direction is hot) then MAKE_HIS(objfile, HIS, target address of not-taken direction) else return;
5-3 if (taken direction is hot) then MAKE_HIS(objfile, HIS, target address of taken direction) else return;
6 default: return;
Linking HBBs to generate HIS

- To avoid MECIs with some parts in the loop and some parts out of the loop, the algorithm starts from inner loops, then outer loops and then other hot parts of the code.
- The hot loops and HBBs are sorted according to their start address.
Generating CDFG for HIS

- While generating CDFG for a HIS all possible sources should be considered for each input plus the effective branches.
Generating Multi-Exit Custom Instructions

- In this implementation MECIs can only include fixed point instructions excluding *multiply*, *divide* and *load*. At most on *store* and five *branches*.
- A MECI can have at most four exit points
  - branch with only one hot direction
  - *indirect jump* and *return*
  - *Call*
  - hot backward branch
  - an instruction where its next instruction is *non-executable*.
- MECIs include the hot directions of branches
- If both directions of a branch are hot, both corresponding HBBs are added.
Generating Multi-Exit Custom Instructions

- Finding the largest sequence of instructions in the CFG of HIS
- Checking the anti-dependence and flow-dependence and moving executable instructions to the head and tail(s) of MECIs
- Rewriting the object code where instructions are going to be moved.
Example: a MECI with 2 exits

Detecting subgraphs in Moving instructions after checking data dependency (binary rewriting)

Overwriting the entry node with mtc1 instr
Executing MECIs

Kyushu University

ARC@Yokohama
Outline

- Introduction
- Overview of ADEXOR Architecture
- Generating Multi-Exit Custom Instructions
  - Motivation Example
  - Tool Chain
  - Generating MECIs
- Proposing an Architecture for the CRFU
  - Design Methodology
  - Supporting Conditional Execution
  - Proposed Architecture
  - Integrating the CRFU with the base processor
- Experiment Results
- Future Work
Design Methodology

First Phase
- Application
- Generating HIS
- Custom Instructions (CIs)
- Mapping (MapToolP1) and Analyzing CIs for Determining the CRFU Preliminary Architecture
- CRFU Primary Architecture
- Object Code

Second Phase
- Mapping Process (MapToolP2)
- Mappable CIs (MapCIP2)
- Rejected CIs
- Mapping is successful?
  - Yes
  - No

Third Phase
- Integrated Temporal Partitioning & Mapping (IntegFrameP3)
- Mappable CIs (MapCIP3)
- Fixing Final CRFU Architecture
- Final CRFU Architecture

Fourth Phase
- Modifying Mapping Process (MapToolP4) & IntegFrameP4
Mapping Tool

A Custom Instruction

1: SUBU R3, R0, R3
2: ADDU R10, R0, R0
3: SRA R8, R10, 0x3
4: SLT R2, R3, R8
5: BNE R0, R0, R0, 400488, R2

Data Flow Graph

RFU Map
Integrated Framework (1/2)

- Integrated Framework
  - Performs an integrated temporal partitioning and mapping process
  - Takes rejected CIs as input
  - Partitions them to appropriate mappable CIs
  - Adds nodes to the current partition while architectural constraints are satisfied
  - The ASAP level of nodes represents their order to execute according to their dependencies
Integrated Framework (2/2)

- Incremental H\textit{TTT}
  - The node with the highest ASAP level is selected and moved to the subsequent partition.
- Nodes selection and moving order: 15, 13, 11, 9, 14, 12, 10, 8, 3 and 7.
Supporting Conditional Execution
Supporting Conditional Execution

![Diagram showing conditional execution support with a Selector-Mux and configuration bits.]
CRFU Architecture: A Quantitative Approach

○ 22 programs of MiBench were chosen
○ Simplescalar toolset was utilized for simulation
○ CRFU is a matrix of FUs
  ● No of Inputs
  ● No of Outputs
  ● No of FUs
  ● Connections
  ● Location of Inputs & Outputs
○ Some definitions:
  ● Considering frequency and weight in measurement
    ○ CI Execution Frequency
    ○ Weight (To equal number of executed instructions)
    ○ Average = for all CIs (ΣFreq*Weight)

  ● Rejection rate: Percentage of MECIs that could not be mapped on the CRFU
  ● Mapping rate: Percentage of MECIs that could be mapped on the CRFU
Inputs/Outputs

![Graph showing the mapping rate for inputs and outputs vs. the number of inputs/outputs.](image-url)
Width/Depth

![Graph showing mapping rate vs number of width and depth with and without constraints.]

- Width without constraints
- Depth without constraints
- Width with constraints
- Depth with constraints
CRFU Architecture
## Distribution of functions

<table>
<thead>
<tr>
<th></th>
<th>row1</th>
<th>row2</th>
<th>ro3</th>
<th>row4</th>
<th>row5</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>and</strong></td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td><strong>Or</strong></td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td><strong>Xor</strong></td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td><strong>Nor</strong></td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>Add/sub</strong></td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td><strong>Shift</strong></td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td><strong>Compare</strong></td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>
Synthesis result

- Synopsys tools
- Hitachi 0.18 \( \mu \text{m} \)
- Area: 2.1 mm\(^2\)
- Delay

<table>
<thead>
<tr>
<th>Depth of DFG of MECI</th>
<th>Delay (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.2</td>
</tr>
<tr>
<td>2</td>
<td>4.2</td>
</tr>
<tr>
<td>3</td>
<td>6.1</td>
</tr>
<tr>
<td>4</td>
<td>7.9</td>
</tr>
<tr>
<td>5</td>
<td>9.8</td>
</tr>
</tbody>
</table>
Control Bits & Immediate Data

- **375** bits are needed as Control Bits for
  - Multiplexers
  - Functional Units
- **240** bits are needed for Immediates
- Each CI configuration needs
  \[(308+204 = 615 \text{ bits})\]
Outline

- Introduction
- Overview of ADEXOR Architecture
- Generating Multi-Exit Custom Instructions
  - Motivation Example
  - Tool Chain
  - Generating MECIs
- Proposing an Architecture for the CRFU
  - Design Methodology
  - Supporting Conditional Execution
  - Proposed Architecture
  - Integrating the CRFU with the base processor
- Experiment Results
- Future Work
Experiment setup

- 22 applications of Mibench
- Simplescalar

<table>
<thead>
<tr>
<th>Issue</th>
<th>4-way</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1- I cache</td>
<td>32K, 2 way, 1 cycle latency</td>
</tr>
<tr>
<td>L1- D cache</td>
<td>32K, 4 way, 1 cycle latency</td>
</tr>
<tr>
<td>Unified L2</td>
<td>1M, 6 cycle latency</td>
</tr>
<tr>
<td>Execution units</td>
<td>4 integer, 4 floating point</td>
</tr>
<tr>
<td>RUU size &amp; Fetch queue size</td>
<td>64</td>
</tr>
<tr>
<td>Branch predictor</td>
<td>bimodal</td>
</tr>
<tr>
<td>Branch prediction table size</td>
<td>2048</td>
</tr>
<tr>
<td>Extra branch misprediction latency</td>
<td>3</td>
</tr>
</tbody>
</table>
Speedup CIs & MECIs

- The number of inputs, outputs and FUs are the same
- simpler connections and FUs and does not support conditional execution.
- Area: 1.15 mm²
- Delay for a CI with a critical length of five is 7.66 ns.
- Each CI configuration needs 512 bits.
- The average number of instructions included in CIs (one HBB) is 6.39 instructions and for MECIs is 7.85 instructions.
Speedup CIs & MECIs

The diagram shows the speedup of various programs using CIs and MECIs. The x-axis represents different programs, while the y-axis represents speedup. The bars indicate the speedup for CIs and MECIs.

Programs included in the diagram are: bitcnts, basicmath, qsort, susan, cjpeg, djpeg, lame, dijkstra, patricia, stringsearch, blowfish, rijndael, sha, adpcm, crc, fft, gsm, and Average.
MECIs vs. Cl

%Speedup enhancement

bitcnts basicmath qsort susan jpeg jpeg lame dijkstra patricia stringsearch blowfish ndcuel sha adpcm crc fft gsm Average

Kyushu University

ARC@Yokohama
Effect of clock frequency of speedup

![Graph showing the effect of clock frequency on speedup for various programs. The x-axis represents the clock frequencies from 200 MHz to 400 MHz, and the y-axis represents the speedup. Different programs such as bitcnts, basicmath, qsort, susan, jpeg, jpeg, lame, dijkstra, patricia, stringsearch, blowfish, rjndael, sha, adpcm, crc, fft, gsm, and average are shown with bars indicating their speedup at each frequency.](image-url)
## Execution time of configuration phase

<table>
<thead>
<tr>
<th>Application</th>
<th>Exec. time (Seconds)</th>
<th>Application</th>
<th>Exec. time (Seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>adpcm</td>
<td>225</td>
<td>gsm</td>
<td>461</td>
</tr>
<tr>
<td>bitcounts</td>
<td>331</td>
<td>lame</td>
<td>526</td>
</tr>
<tr>
<td>blowfish</td>
<td>94</td>
<td>patricia</td>
<td>84</td>
</tr>
<tr>
<td>basicmath</td>
<td>34</td>
<td>qsort</td>
<td>233</td>
</tr>
<tr>
<td>cjjpeg</td>
<td>75</td>
<td>rijndael</td>
<td>68</td>
</tr>
<tr>
<td>crc</td>
<td>132</td>
<td>sha</td>
<td>29</td>
</tr>
<tr>
<td>dijkstra</td>
<td>101</td>
<td>stringsearch</td>
<td>3</td>
</tr>
<tr>
<td>djjpeg</td>
<td>9</td>
<td>susan</td>
<td>122</td>
</tr>
<tr>
<td>fft</td>
<td>36</td>
<td>Average</td>
<td>150.8</td>
</tr>
</tbody>
</table>
Effect of connections on mapping rate

- By deleting the connections with length more than one, 24.2% of MECIs can not be mapped.
Conclusion

- Our experimental results show that by extending custom instructions over multiple HBBs the average speedup increases by 46% compared to the custom instructions which are limited to only one HBB. This is achieved in return for 83% more hardware and 20% more configuration bits. Utilizing connections with different length are helpful for supporting larger custom instructions with the available number of FUs.
Future work

- Energy evaluation of the ADEXOR
- Exploring the design space of CRFU architecture (To study the effect of number of inputs, outputs, FU on the speedup, area and power)
Thank you for your attention
Introduction (1/2)

- Approaches for designing Embedded SoCs
  - Application Specific Integrated Circuits (ASIC)
    - High performance
    - Low power
    - No flexibility
    - High cost and long design time
  - General Purpose Processors (GPPs)
    - Low performance
    - High power consumption
    - Programmable
    - Availability of tools
  - Application Specific Instruction-set Processors (ASIP)
    - Higher performance than GPPs
    - More flexibility than ASICs
    - Expensive and has long design turnaround time
  - Extensible Processor
    - Base processor + custom functional unit
    - Accelerating frequently executed subgraphs using custom functional units
    - Manufacturing new processor for each application (significant non-recurring engineering cost)
Generating MECIs

- Block 3 (B3) is selected as the biggest instructions sequence that can be executed on the ACC.
- Block 2 (B2) can not be executed on ACC.
- Block 1 (B1) can be executed on ACC.
- If there is no flow and anti-dependency between B1 and B2 exchange them.
- This is done for B3, B4 and B5.
Profiling

○ Configuration phase
  ● Instruction Set Simulator (e.g. SimpleScalar)
  ● Profiling
    ○ Program Counter (PC)
    ○ Committed Instructions

<table>
<thead>
<tr>
<th>Basic Block Addr</th>
<th>Exec. Freq.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Branch Addr</th>
<th>Exec. Freq</th>
<th>Taken Freq.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Kyushu University

ARC@Yokohama