Simultaneously Improving Code Size, Performance, & Energy in Embedded Processors

Ahmad Zmily and Christos Kozyrakis

Electrical Engineering Department
Stanford University
{zmily,kozyraki}@stanford.edu
Efficiency Metrics for Embedded CPUs

- High performance
  - To meet soft or hard real-time constraints
  - Increasing demands from emerging applications

- Low energy consumption
  - Determines battery life-time
  - Critical for portable and deeply-embedded systems

- Low code size
  - Determines cost of program storage devices
    - Flash, ROM, instruction RAM/caches
  - A major cost component for the overall system
Code Size Trade-offs

- 16-bit instructions
  - Better code size, worse performance & energy than 32-bit code
- Selective use of 16-bit instructions
  - Better code size, same performance & energy as 32-bit code
- This talk: can we improve all three simultaneously?

C. Kozyrakis, 3/2006
BLISS = Block-aware Instruction Set

- Explicit basic block descriptors (BBDs)
  - Stored separately from instructions in the text segment
  - Describe control flow and identify associated instructions

- Execution model
  - PC always points to a BBD, not to instructions
  - Atomic execution of basic blocks
### 32-bit Basic Block Descriptor Format

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>9</td>
<td>4</td>
<td>14</td>
<td>1</td>
</tr>
<tr>
<td>Type</td>
<td>Offset</td>
<td>Length</td>
<td>Instruction Pointer</td>
<td>Size</td>
</tr>
</tbody>
</table>

- **Type**: type of terminating control-flow instruction
  - Fall-through, jump, jump register, branch, call, return
- **Offset**: displacement for PC-relative branches and jumps
  - Offset to target basic block descriptor
- **Length**: number of instruction in the basic block
  - 0 to 15 instructions
- **Instruction pointer**: address of the first instruction in the block
  - Remaining bits from TLB
- **Size**: indicate the encoding size of instructions in the block
  - 16-bit encoding or 32-bit encoding
BLISS Decoupled Front-End

- BTB replaced by cache for block descriptors
- Decoupled descriptors fetch from instruction fetch
Higher performance
- Better branch prediction using software-defined info
  - L2 stores descriptors, better use of direction predictors, tolerate I-cache latency, …
- I-cache prefetching using info in block descriptors

Lower energy consumption
- Energy saved by reducing mispredicted instructions
- Judicious access to I-cache using software-define info
  - Merge accesses to sequential blocks, serial tag/data access, read needed words only, …

See [ISLPED’05][EUROPAR’05] for details
BLISS Code Size

- Naïve code generation
  - 10-20% code size increase compared to 32-bit RISC!
  - A block descriptor per 5 to 10 instructions

- Basic code size optimizations
  - All jump instruction are removed
    - BBD defines both control-flow type and the offset
  - Many conditional branches can be removed
    - Simple condition test encoded in the producing opcode
    - Branch target is provided by the block descriptor
Block Subsetting Optimization

- Idea: duplicate descriptors but never instructions
  - Eliminate all instruction in a block if exact sequence found elsewhere in the binary
  - Adjust instruction pointer in block descriptor

**BLISS code**

<table>
<thead>
<tr>
<th>BB descriptors</th>
<th>BLISS code</th>
</tr>
</thead>
<tbody>
<tr>
<td>BBD1: BR_F, BBD3,</td>
<td>beg r8, r1</td>
</tr>
<tr>
<td>BBD2: J , BBD4, 2,</td>
<td>add r3, r2, r8</td>
</tr>
<tr>
<td>BBD3: JAL, foo, 3,</td>
<td>addiu r17, r0, 1</td>
</tr>
<tr>
<td>BBD4:</td>
<td>lw r6,1492(r30)</td>
</tr>
<tr>
<td></td>
<td>add r4, 0, r2</td>
</tr>
<tr>
<td></td>
<td>add r3, r2, r8</td>
</tr>
<tr>
<td></td>
<td>addiu r17, r0, 1</td>
</tr>
</tbody>
</table>

**Optimized BLISS code**

<table>
<thead>
<tr>
<th>BB descriptors</th>
<th>Optimized BLISS code</th>
</tr>
</thead>
<tbody>
<tr>
<td>BBD1: BR_F, BBD3,</td>
<td>beg r8, r1</td>
</tr>
<tr>
<td>BBD2: J , BBD4, 2,</td>
<td>lw r6,1492(r30)</td>
</tr>
<tr>
<td>BBD3: JAL, foo, 3,</td>
<td>add r4, 0, r2</td>
</tr>
<tr>
<td>BBD4:</td>
<td>add r3, r2, r8</td>
</tr>
<tr>
<td></td>
<td>addiu r17, r0, 1</td>
</tr>
</tbody>
</table>
Efficient 16-bit/32-bit Code Interleaving

- **MIPS16**: interleaving at function-level
  - JALX instruction is used to switch between functions
  - But functions include both perf. critical & non-critical code

- **Thumb-2, rISA**: instruction-level interleaving
  - A couple of instructions per switch
  - Can switch encoding at arbitrary points

- **BLISS**: basic block interleaving
  - A block is either fully perf. critical or fully non-critical
  - Descriptor indicates the encoding size for each block
  - No other overhead/instructions for switch
Evaluation Summary

- Up to 60% average compression ratio
  - For mediabench applications

- Performance & energy for Xscale PXA270
  - +10% performance, -20% total energy, see paper for details
  - Similar results for high-end embedded CPUs
    - IBM PowerPC 750GX
## Code Size Statistics

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>MIPS32</th>
<th>BLISS basic Optimization</th>
<th>Block Subset</th>
<th>Selective 16/32 Blocks</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Code Size (kb)</td>
<td>J/B inst. Removed</td>
<td># BB</td>
<td>Inst. eliminated</td>
</tr>
<tr>
<td>adpcm</td>
<td>36</td>
<td>1671</td>
<td>2607</td>
<td>2536</td>
</tr>
<tr>
<td>epic</td>
<td>64</td>
<td>2808</td>
<td>4345</td>
<td>4771</td>
</tr>
<tr>
<td>g721</td>
<td>42</td>
<td>1920</td>
<td>2942</td>
<td>3015</td>
</tr>
<tr>
<td>gsm</td>
<td>69</td>
<td>2866</td>
<td>4409</td>
<td>4483</td>
</tr>
<tr>
<td>jpeg</td>
<td>109</td>
<td>4535</td>
<td>6609</td>
<td>8033</td>
</tr>
<tr>
<td>mesa</td>
<td>430</td>
<td>16692</td>
<td>24054</td>
<td>36628</td>
</tr>
<tr>
<td>mpeg2.dec</td>
<td>77</td>
<td>3514</td>
<td>5128</td>
<td>5168</td>
</tr>
<tr>
<td>mpeg2.enc</td>
<td>104</td>
<td>4390</td>
<td>6502</td>
<td>6895</td>
</tr>
<tr>
<td>pegwit</td>
<td>74</td>
<td>2735</td>
<td>4094</td>
<td>5229</td>
</tr>
<tr>
<td>rasta</td>
<td>226</td>
<td>10628</td>
<td>13788</td>
<td>19191</td>
</tr>
</tbody>
</table>
BLISS Vs. Selective 16-bit code

- BLISS achieves similar code size reduction with
  - 10% performance improvement
  - 21% total energy improvement
Conclusions

- **BLISS: a block-aware instruction set**
  - Defines basic block descriptors separate from instructions
  - Expressive ISA to communicate software info and hints

- **Enabled code optimizations**
  - Eliminate redundant jump and branch instructions
  - Remove blocks which appear elsewhere in the code
  - Interleaving 16-bit & 32-bit code at basic-block level without overhead

- **Improved code size and energy consumption and performance:**
  - 40% code size reduction
  - 10% performance improvement
  - 21% reduction in total energy consumption