Reducing Code Size with Run-time Decompression

Charles Lefurgy, Eva Piccininni, and Trevor Mudge

Advanced Computer Architecture Laboratory
Electrical Engineering and Computer Science Dept.
The University of Michigan, Ann Arbor

High-Performance Computer Architecture (HPCA-6)
January 10-12, 2000
Motivation

• **Problem: embedded code size**
  – Constraints: cost, area, and power
  – Fit program in on-chip memory
  – Compilers vs. hand-coded assembly
    • Portability
    • Development costs
  – Code bloat

• **Solution: code compression**
  – Reduce compiled code size
  – Take advantage of instruction repetition

• **Implementation**
  – Hardware or software?
  – Code size?
  – Execution speed?
Software decompression

- **Previous work**
  - Decompression unit: whole program [Tauton91]
    - No memory savings
  - Decompression unit: procedures [Kirovski97][Ernst97]
    - Requires large decompression memory
    - Fragmentation of decompression memory
    - Slow

- **Our work**
  - Decompression unit: 1 or 2 cache-lines
  - High performance focus
  - New profiling method
Dictionary compression algorithm

- **Goal**: fast decompression
- **Dictionary contains unique instructions**
- **Replace program instructions with short index**

![Diagram showing original and compressed program segments]

**Original program**

```
lw r2, r3
lw r2, r3
lw r15, r3
lw r15, r3
lw r15, r3
```

**Compressed program**

```
lw r2, r3
lw r15, r3
```

32 bits

16 bits

32 bits
Decompression

• Algorithm
  1. I-cache miss invokes decompressor (exception handler)
  2. Fetch index
  3. Fetch dictionary word
  4. Place instruction in I-cache (special instruction)

• Write directly into I-cache
• Decompressed instructions only exist in I-cache
CodePack

• **Overview**
  – IBM
  – PowerPC
  – First system with instruction stream compression
  – Decompress during I-cache miss

• **Software CodePack**

<table>
<thead>
<tr>
<th></th>
<th>Dictionary</th>
<th>CodePack</th>
</tr>
</thead>
<tbody>
<tr>
<td>Codewords (indices)</td>
<td>Fixed-length</td>
<td>Variable-length</td>
</tr>
<tr>
<td>Decompress granularity</td>
<td>1 cache line</td>
<td>2 cache lines</td>
</tr>
<tr>
<td>Decompression overhead</td>
<td>75 instructions</td>
<td>1120 instructions</td>
</tr>
</tbody>
</table>
Compression ratio

- \( \text{compression ratio} = \frac{\text{compressed size}}{\text{original size}} \)

- CodePack: 55% - 63%
- Dictionary: 65% - 82%
Simulation environment

- **SimpleScalar**
- **Pipeline**: 5 stage, in-order
- **I-cache**: 16KB, 32B lines, 2-way
- **D-cache**: 8KB, 16B lines, 2-way
- **Memory**: 10 cycle latency, 2 cycle rate
Performance

- **CodePack**: very high overhead
- **Reduce overhead by reducing cache misses**

![Graph](image-url)

- **Slowdown relative to native code**
- **Go**
- **I-cache size (KB)**
  - 4KB
  - 16KB
  - 64KB
- **CodePack**
- **Dictionary**
- **Native**
Cache miss

• Control slowdown by optimizing I-cache miss ratio
Selective compression

• **Hybrid programs**
  – Only compress some procedures
  – Trade size for speed
  – Avoid decompression overhead

• **Profile methods**
  – Count dynamic instructions
    • Example: Thumb
    • Use when compressed code has more instructions
    • Reduce number of executed instructions
  – Count cache misses
    • Example: CodePack
    • Use when compressed code has longer cache miss latency
    • Reduce cache miss latency
**Cache miss profiling**

- **Cache miss profile reduces overhead 50%**
- **Loop-oriented benchmarks benefit most**
  - Approach performance of native code

![Graph](graph.png)

**Pegwit (encryption)**

- **CodePack: dynamic instructions**
- **CodePack: cache miss**
CodePack vs. Dictionary

- **More compression may have better performance**
  - CodePack has smaller size than Dictionary compression
  - Even with some native code, CodePack is smaller
  - CodePack is faster due to using more native code

![Graph showing comparison between CodePack and Dictionary compression]

**Ghostscript**

- CodePack: cache miss
- Dictionary: cache miss

**Slowdown relative to native code**

**Comression ratio**

- 60%
- 70%
- 80%
- 90%
- 100%
Conclusions

• **High-performance SW decompression possible**
  – Dictionary faster than CodePack, but 5-25% compression ratio difference
  – Hardware support
    • I-cache miss exception
    • Store-instruction instruction

• **Tune performance by reducing cache misses**
  – Cache size
  – Code placement

• **Selective compression**
  – Use cache miss profile for loop-oriented benchmarks

• **Code placement affects decompression overhead**
  – Future: unify code placement and compression
Web page

http://www.eecs.umich.edu/compress