Evaluation of a High Performance Code Compression Method

Charles Lefurgy, Eva Piccininni, and Trevor Mudge

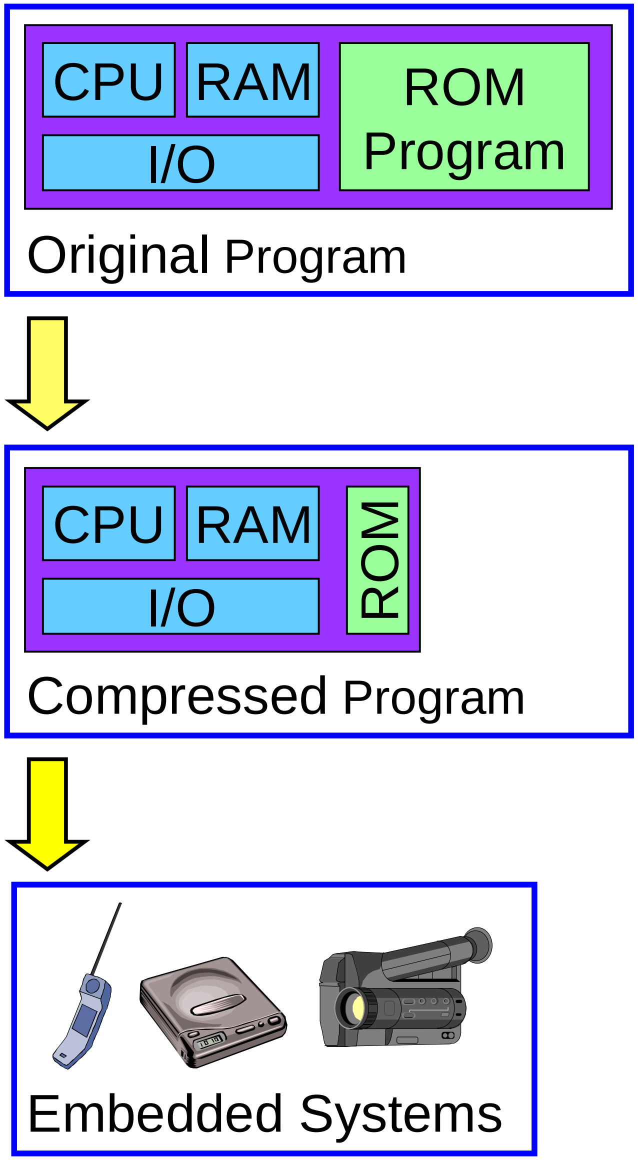
Advanced Computer Architecture Laboratory
Electrical Engineering and Computer Science Dept.
The University of Michigan, Ann Arbor

MICRO-32

November 16-18, 1999

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
  - Systems use cheaper processors with smaller on-chip memories
- **Implementation**
  - Code size?
  - Execution speed?

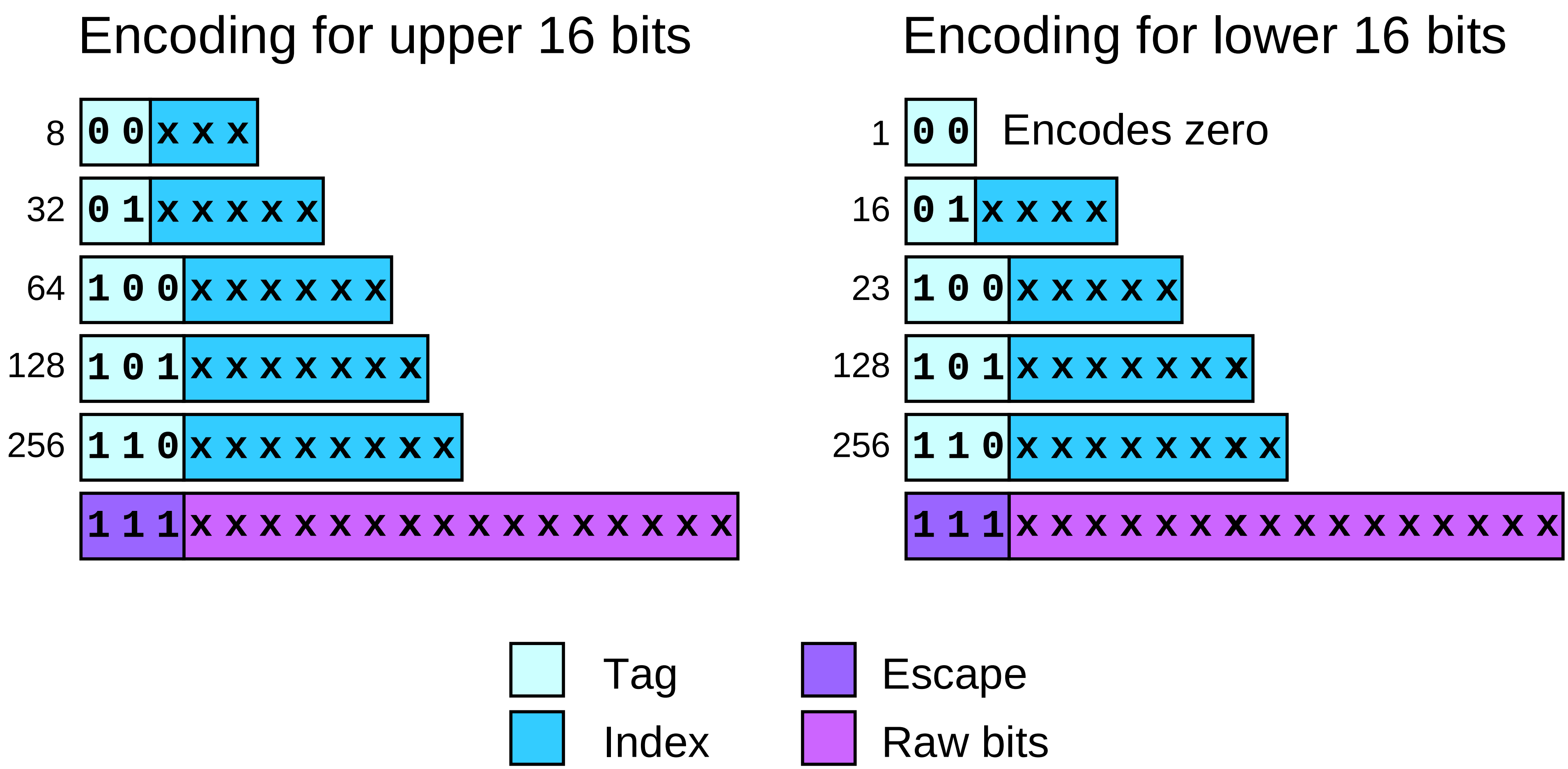


CodePack

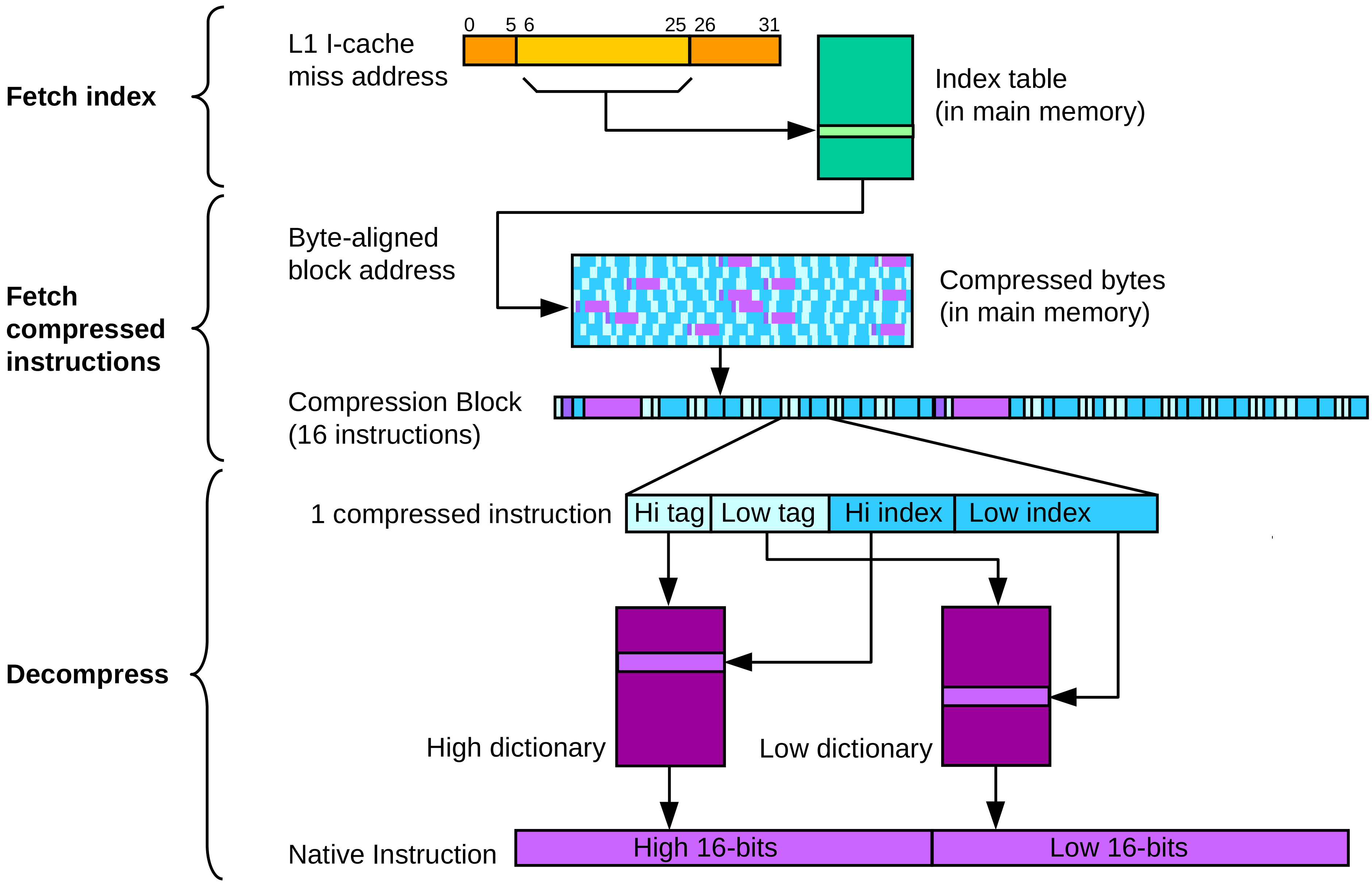
- **Overview**
  - IBM
  - PowerPC instruction set
  - First system with instruction stream compression
  - 60% compression ratio, ±10% performance [IBM]
    - performance gain due to prefetching
- **Implementation**
  - Binary executables are compressed after compilation
  - Compression dictionaries tuned to application
  - Decompression occurs on L1 cache miss
    - L1 caches hold decompressed data
    - Decompress 2 cache lines at a time (16 insns)
  - PowerPC core is unaware of compression

CodePack encoding

- **32-bit insn is split into 2 16-bit words**
- **Each 16-bit word compressed separately**

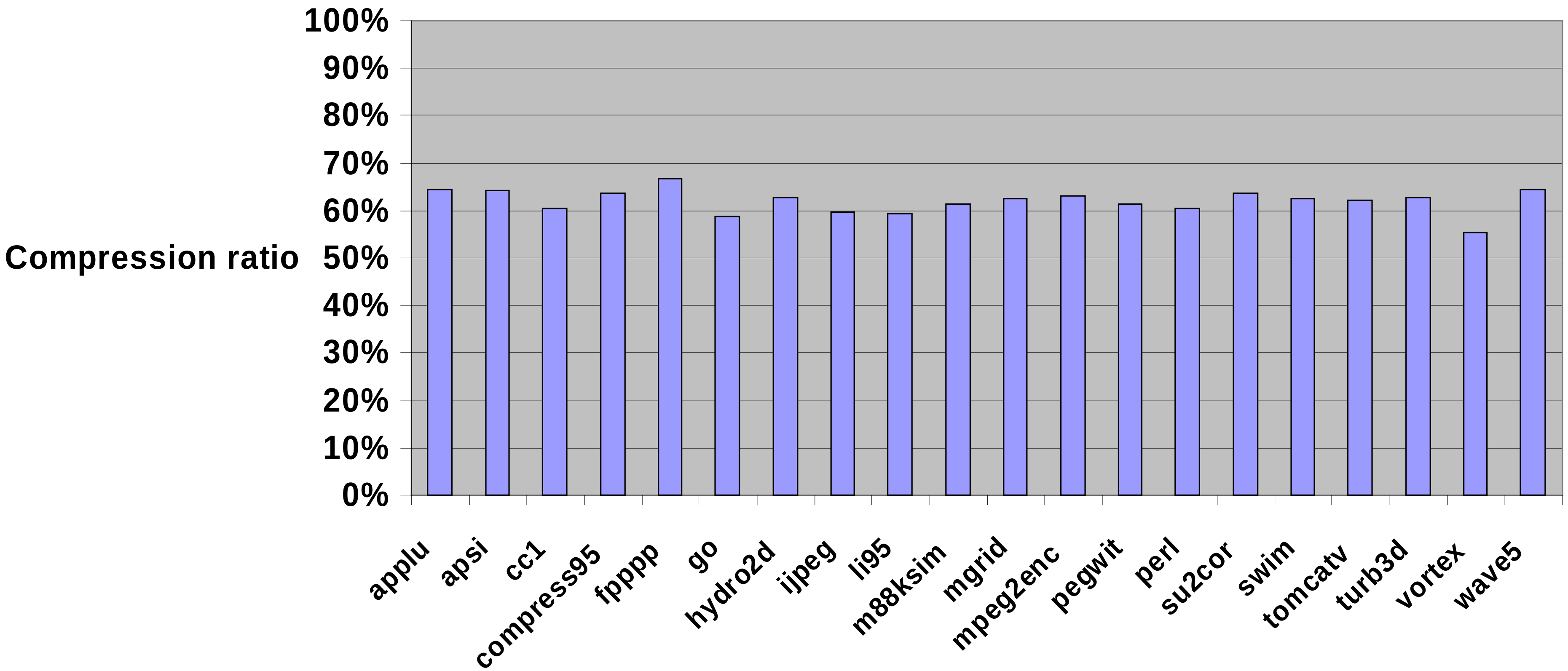


CodePack decompression



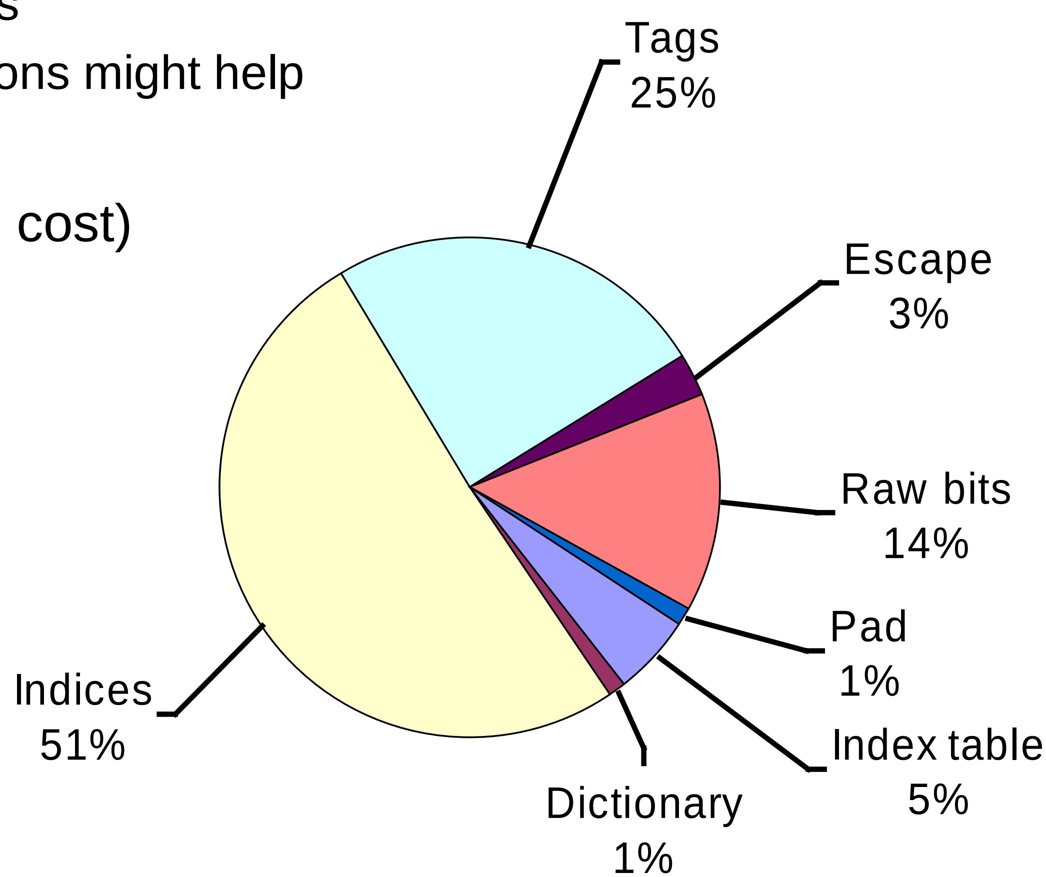
Compression ratio

- $$\textit{compression ratio} = \frac{\textit{compressed size}}{\textit{original size}}$$
- **Average: 62%**



CodePack programs

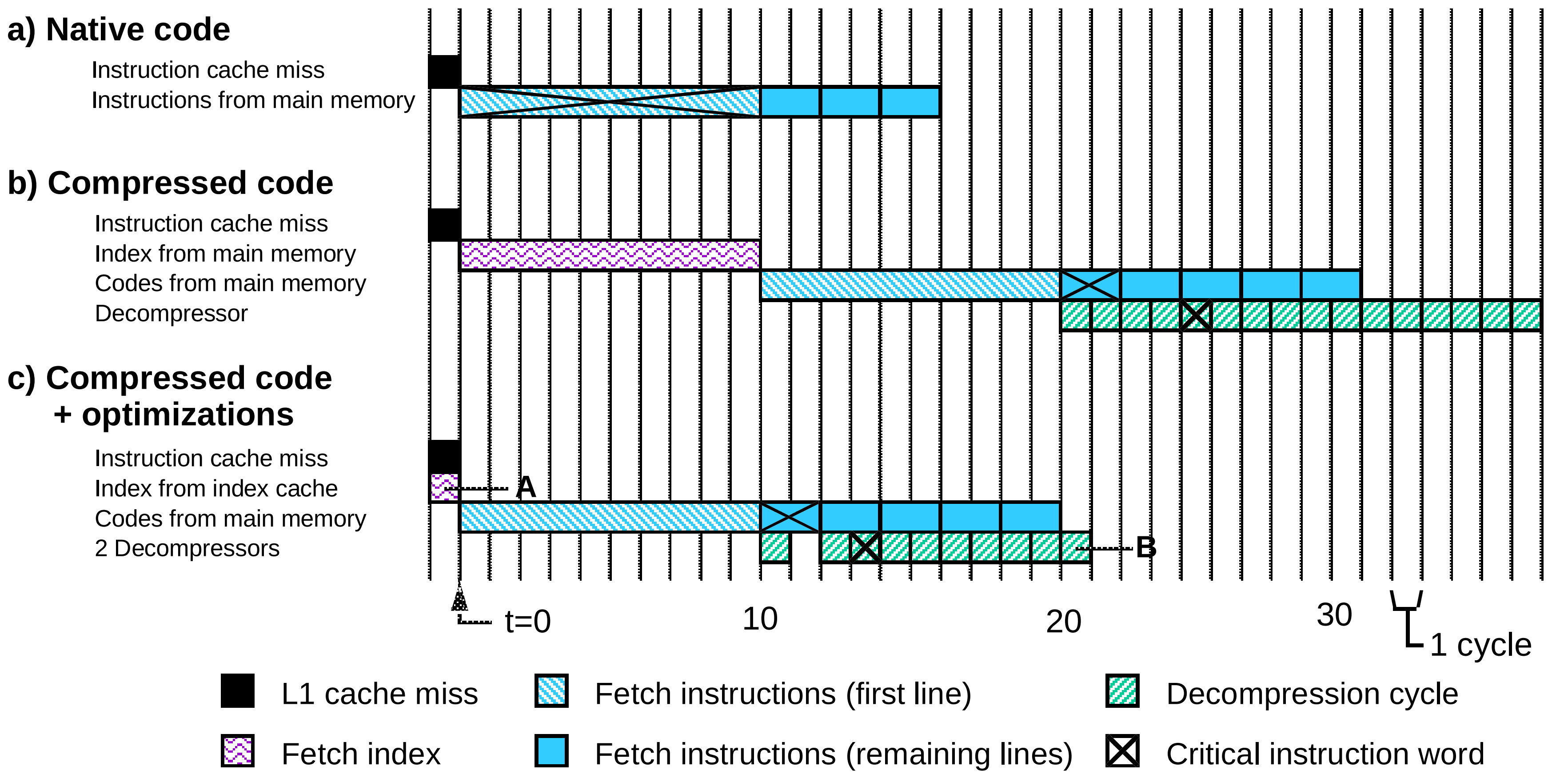
- **Compressed executable**
  - 17%-25% raw bits: not compressed!
    - Includes escape bits
    - Compiler optimizations might help
  - 5% index table
  - 2KB dictionary (fixed cost)
  - 1% pad bits



go

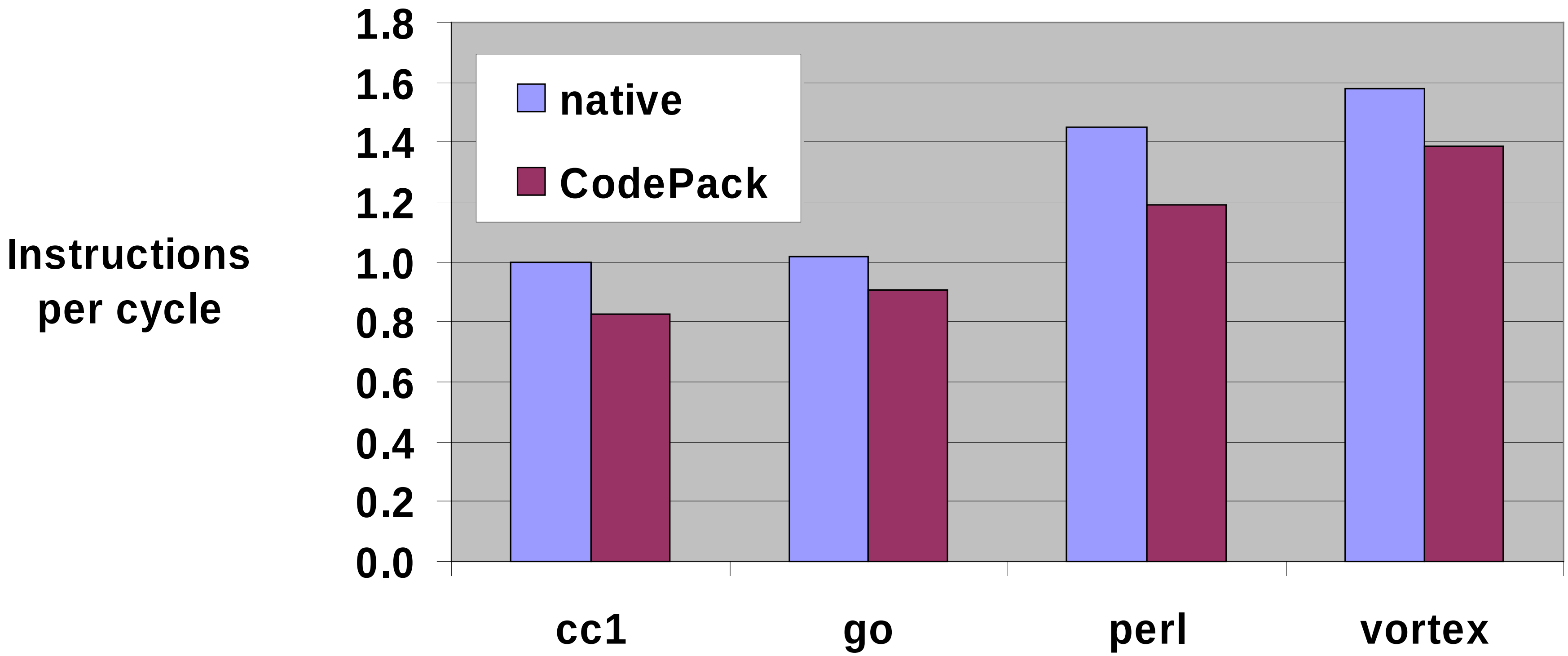
I-cache miss timing

- **Native code uses critical word first**
- **Compressed code must be fetched sequentially**
- **Example shows miss to 5th instruction in cache line**
  - 32-bit insns, 64-bit bus



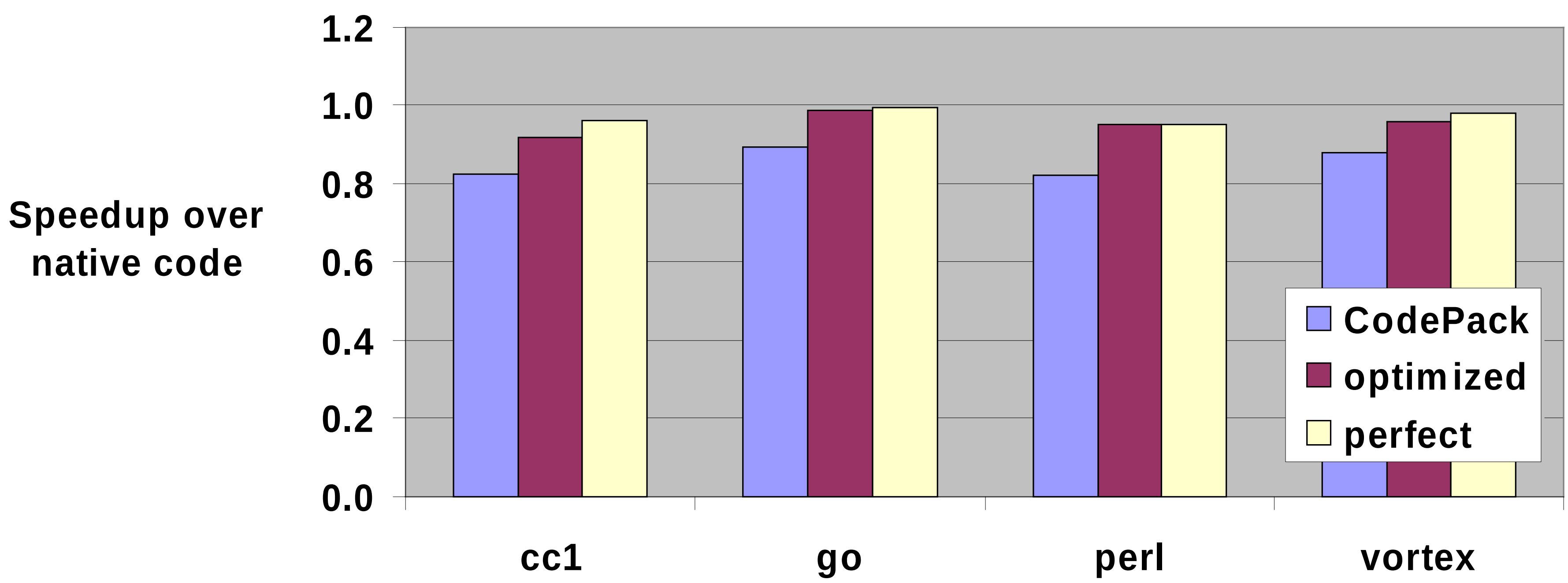
Baseline results

- **CodePack causes up to 18% performance loss**
  - SimpleScalar
  - 4-issue, out-of-order
  - 16 KB caches
  - Main memory: 10 cycle latency, 2 cycle rate



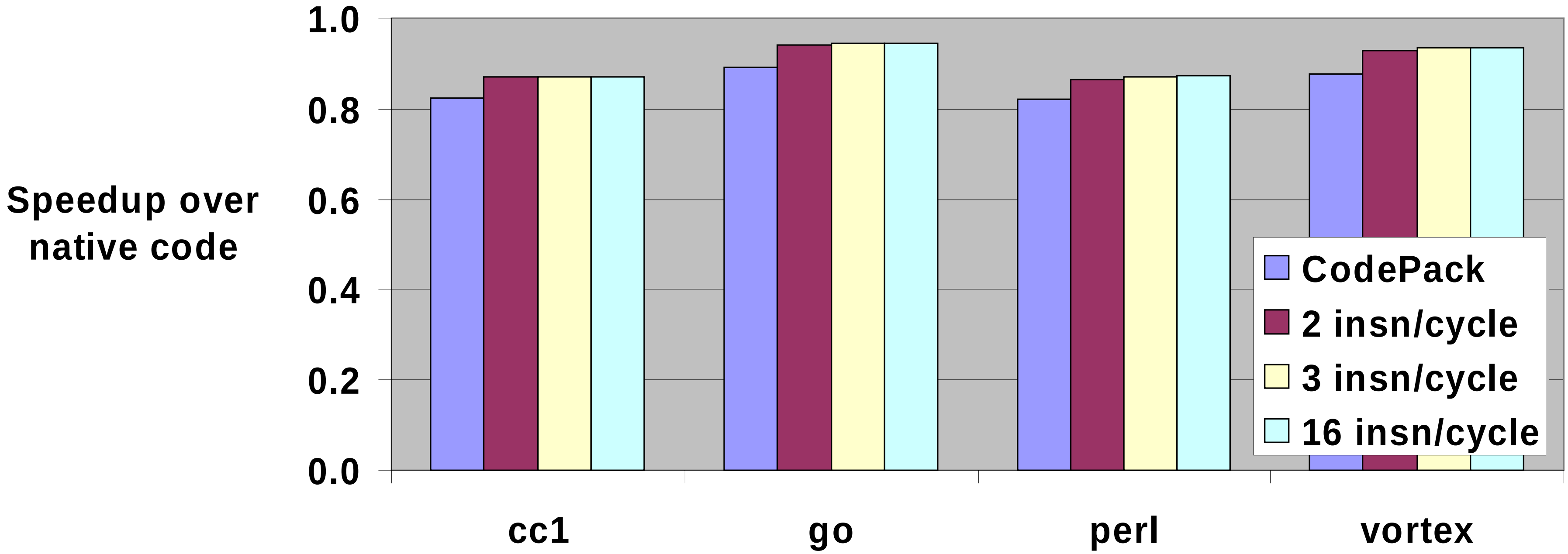
Optimization A: Index cache

- **Remove index table access with a cache**
  - A cache hit removes main memory access for index
  - optimized: 64 lines, fully assoc., 4 indices/line (<15% miss ratio)
    - Within 8% of native code
  - perfect: an infinite sized index cache
    - Within 5% of native code



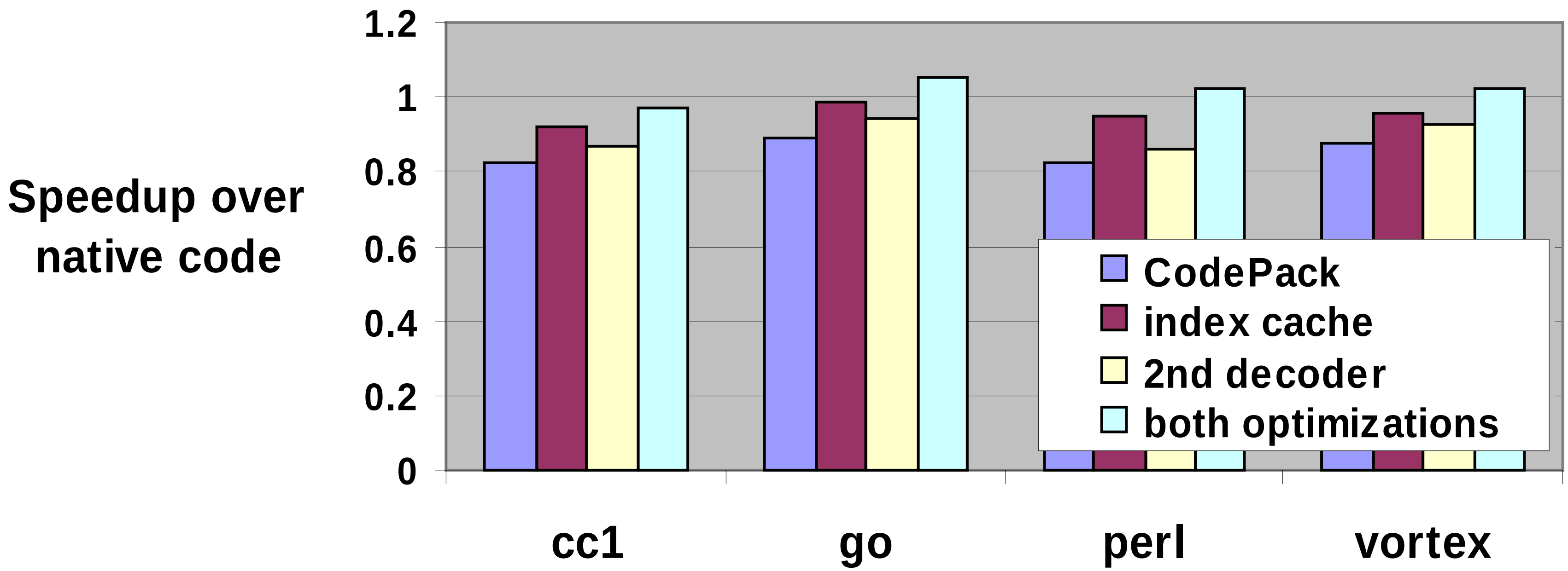
Optimization B: More decoders

- **Codeword tags enable fast extraction of codewords**
  - Enables parallel decoding
- **Try adding more decoders for faster decompression**
- **2 decoders: performance within 13% of native code**



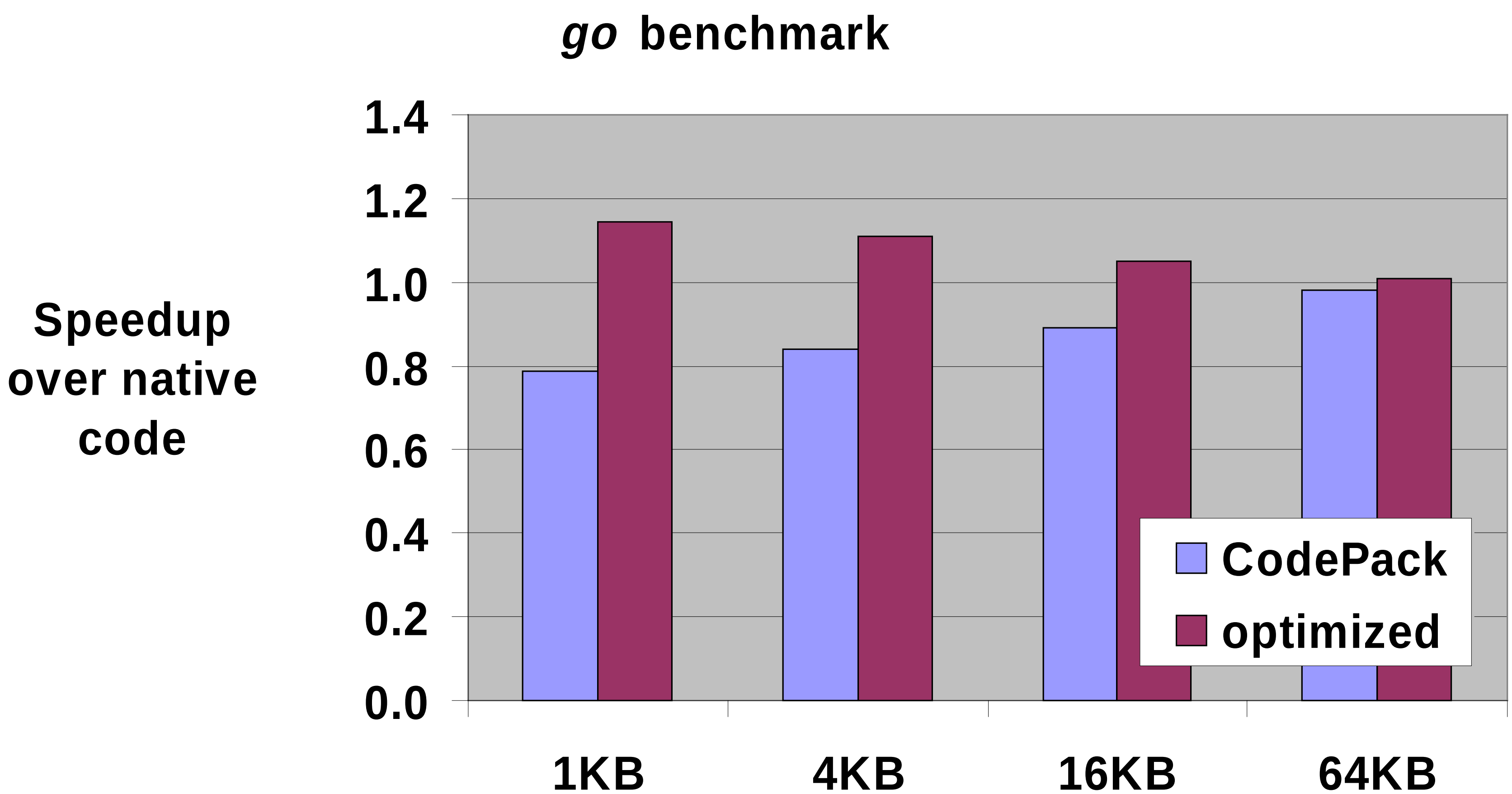
Comparison of optimizations

- **Index cache provides largest benefit**
- **Optimizations**
  - index cache: 64 lines, 4 indices/line, fully assoc.
  - 2nd decoder
- **Speedup over native code: 0.97 to 1.05**
- **Speedup over CodePack: 1.17 to 1.25**



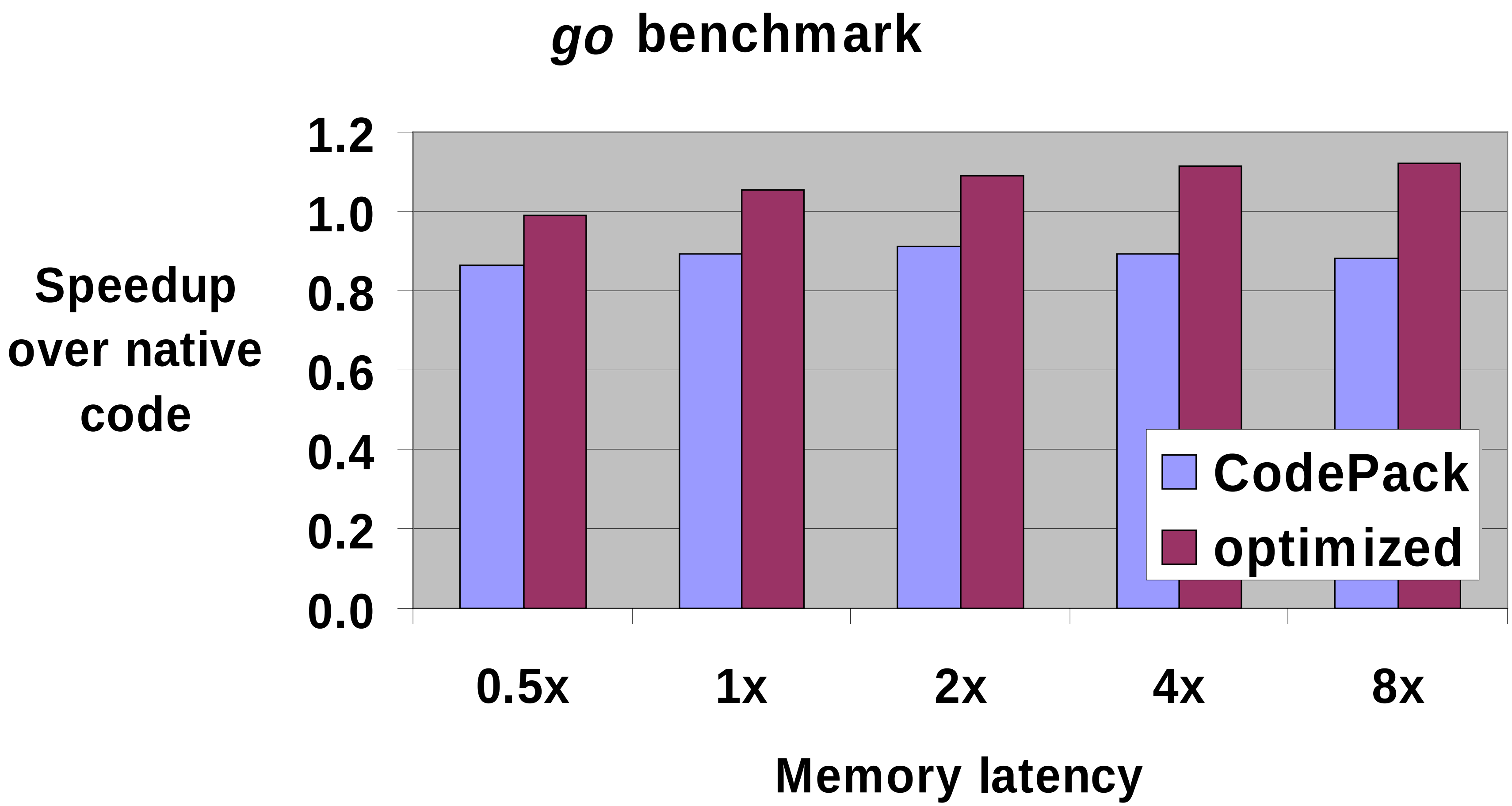
Cache effects

- **Cache size controls normal CodePack slowdown**
- **Optimizations do well on small caches: 1.14 speedup**



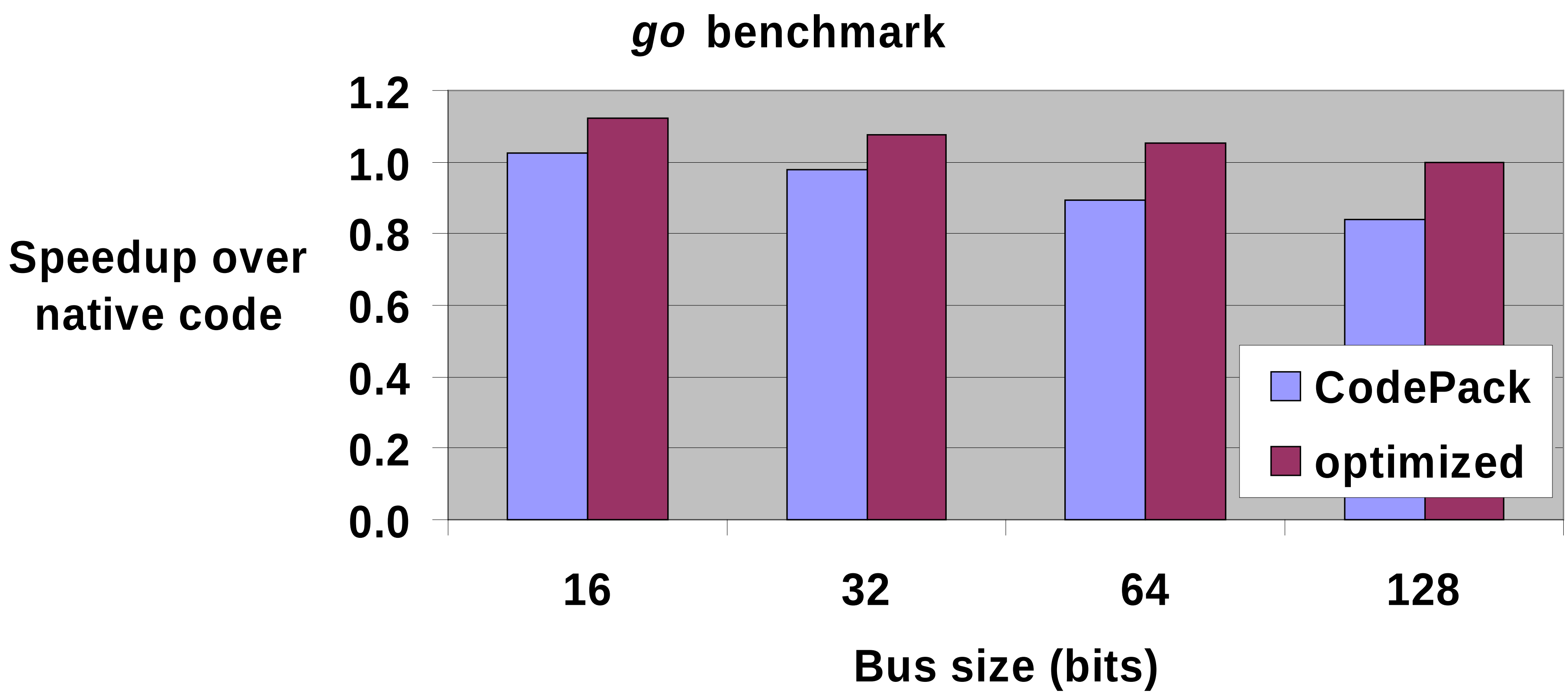
Memory latency

- **Optimized CodePack performs better with slow memories**
  - Fewer memory accesses than native code



Memory width

- **CodePack provides speedup for small buses**
- **Optimizations help performance degrade gracefully as bus size increases**



Conclusions

- **CodePack works for other instruction sets than PowerPC**
- **Performance can be improved at modest cost**
  - Remove decompression overhead: index lookup, dictionary lookup
- **Compression can speedup execution**
  - Compressed code requires fewer main memory accesses
  - CodePack includes simple prefetching
- **Systems that benefit most from compression**
  - Narrow buses
  - Slow memories
- **Workstations might benefit from compression**
  - Fewer L2 misses
  - Less disk access

Web page

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