

# Software-Managed Code Compression for Systems-on-a-Chip

**Charles Lefurgy**

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

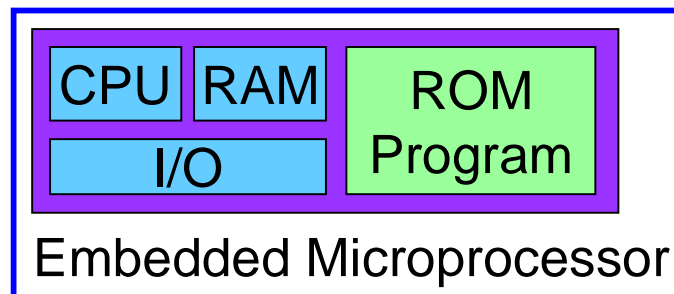
**Advanced Computer Architecture Laboratory  
Electrical Engineering and Computer Science Dept.  
The University of Michigan, Ann Arbor**



# The problem

---

- **Microprocessor die cost**
  - Low cost is critical for high-volume, low-margin embedded systems
  - Control cost by reducing area and increasing yield
- **Increasing amount of on-chip memory**
  - Memory is 40-80% of die area [ARM, MCore]
  - In control-oriented embedded systems, much of this is program memory
- **How can program memory be reduced without sacrificing performance?**



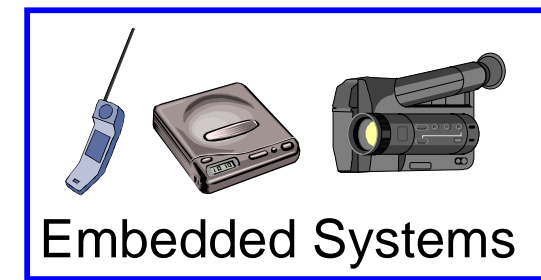
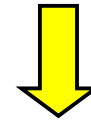
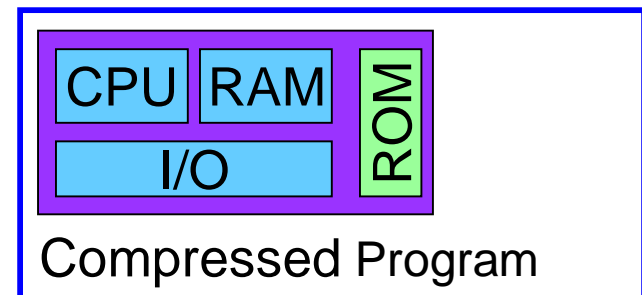
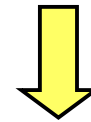
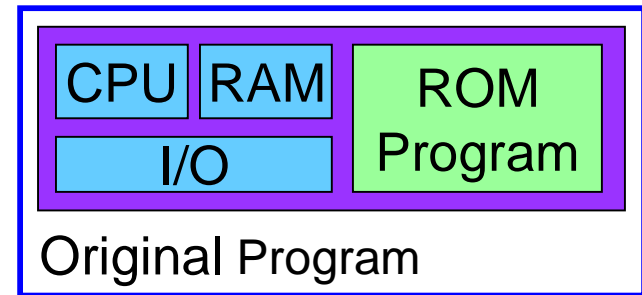
# Solution

- **Code compression**

- Reduce compiled code size
- Compress at compile-time
- Decompress at run-time

- **Implementation**

- Hardware or software?
- Code size?
- Execution speed?



# Research contributions

---

- **HW decompression** [MICRO-32, MICRO-30, CASES-98]
  - Dictionary compression method
  - Analysis of IBM CodePack algorithm
  - HW decompression increases performance
    - Removed decompression overhead
    - Improved bus utilization
    - Works well with narrow buses and long latency memory
- **SW decompression** [HPCA-6, CASES-99]
  - Near native code performance for media applications
  - Software-managed cache
  - Hybrid program optimization (new profiling method)
  - Memoization optimization

# Outline

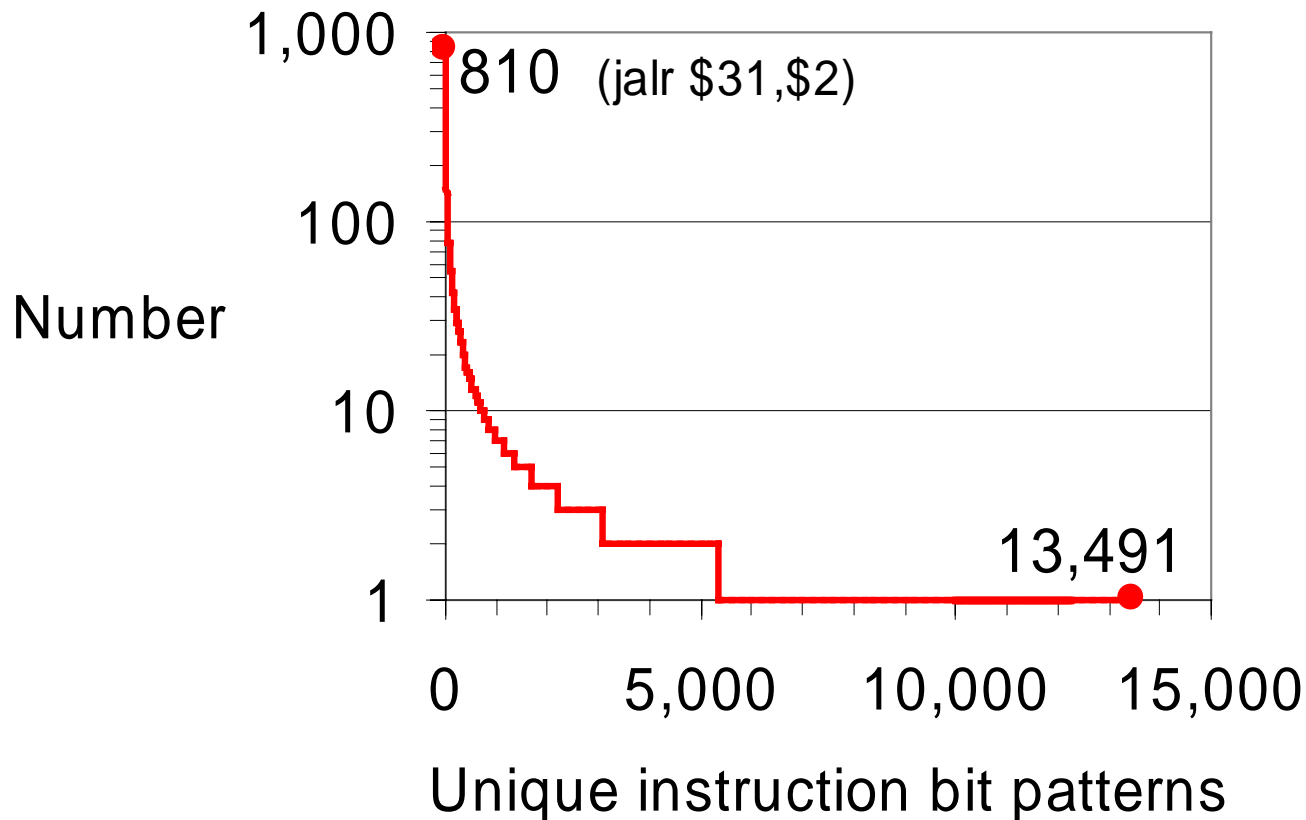
---

- **Compression overview and metrics**
- **SW compression algorithms**
  - Dictionary
  - CodePack
- **Hardware support**
- **Performance study**
- **Optimizations**
  - Hybrid programs
  - Memoization

# Why are programs compressible?

---

- **Ijpeg benchmark** (MIPS gcc 2.7.2.3 -O2)
  - 49,566 static instructions
  - 13,491 unique instructions
  - 1% of unique instructions cover 29% of static instructions



# Previous results

---

Who	Architecture	Compression Ratio	Comment
CodePack	PowerPC	60%	Compress L1 lines
Thumb	ARM	70%	New 16-bit ISA
MIPS-16	MIPS	60%	New 16-bit ISA
Wolfe (CCRP)	MIPS	73%	Huffman on L1 lines
Lekatsas	MIPS, x86	50%, 80%	Arithmetic, stream division
Araujo	MIPS	43%	Op. factor., Huffman
Liao	TMS320C25	82%	Dictionary
Kirovski	SPARC	60%	Procedure compression
Ernst	SPARC	20%	Interpreted wire code

- **Evaluation Metrics**

- Compression ratio

$$\text{compression ratio} = \frac{\text{compressed size}}{\text{original size}}$$

- Decode efficiency
  - measure execution time

# Software decompression

---

- **Previous work**

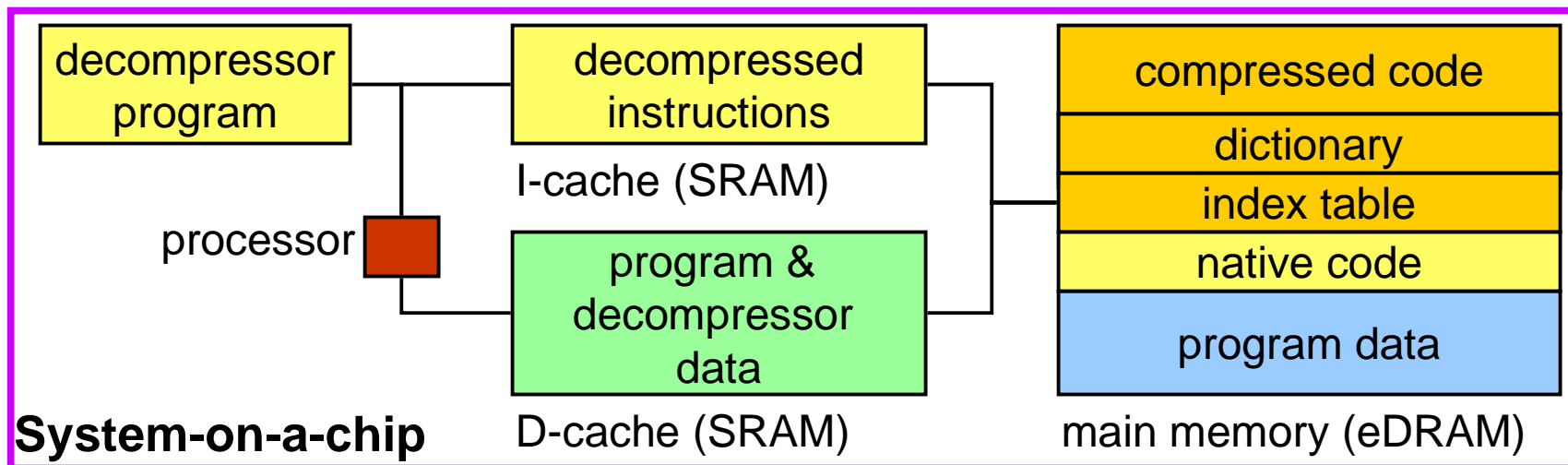
- Whole program compression [Tauton91]
  - Saved disk space
  - No memory savings
- Procedure compression [Kirovski97]
  - Requires large decompression memory
  - Fragmentation of decompression memory
  - Slow

- **My work**

- Decompression unit: 1 or 2 cache-lines
- High performance focus

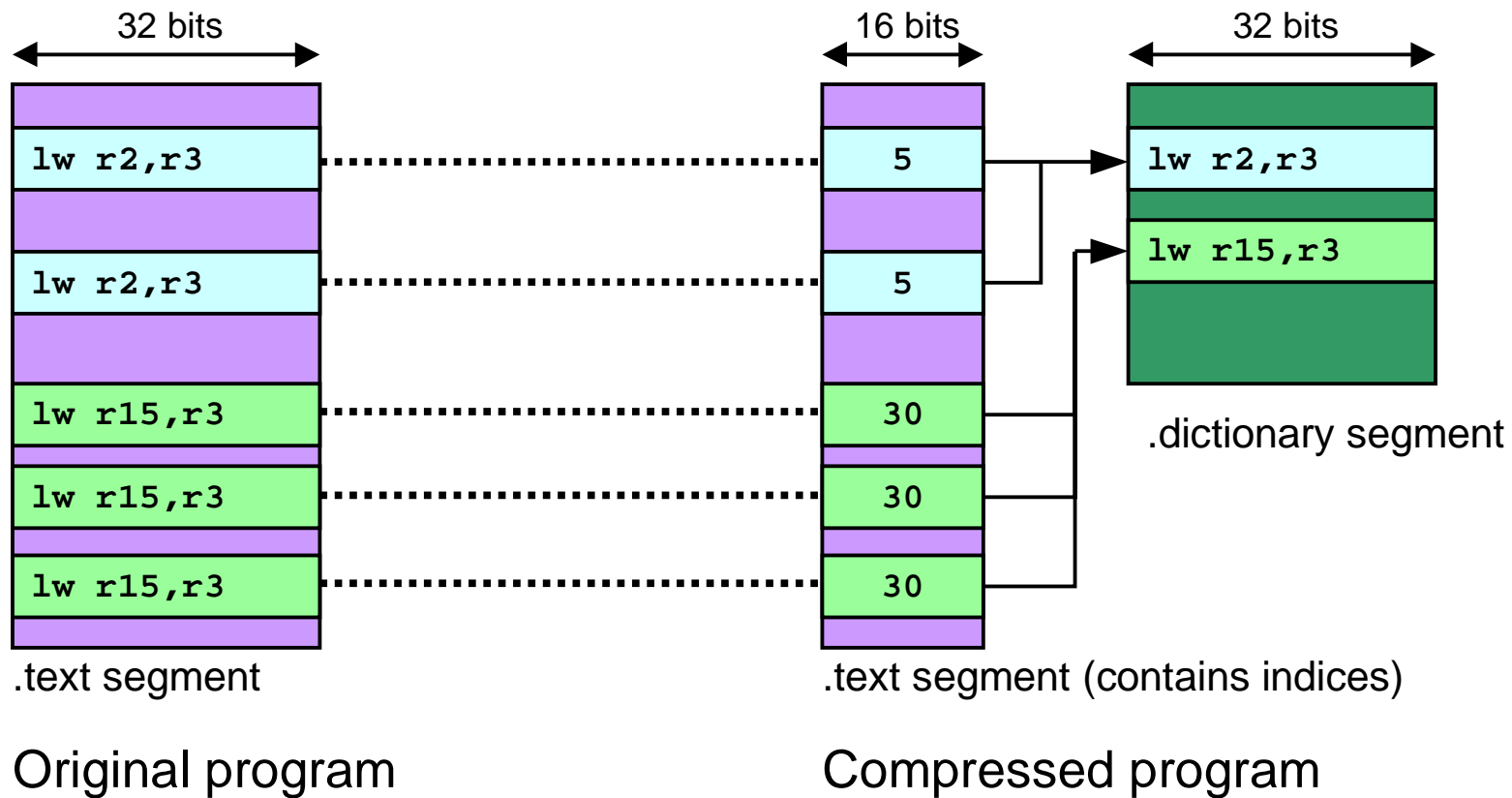
# How does code compression work?

- **What is compressed?**
  - Individual instructions
- **When is decompression performed?**
  - During I-cache miss
- **How is decompression implemented?**
  - I-cache miss invokes exception handler
- **What is decompressed?**
  - 1 or 2 cache lines
- **Where are decompressed instructions stored?**
  - I-cache is the decompression buffer



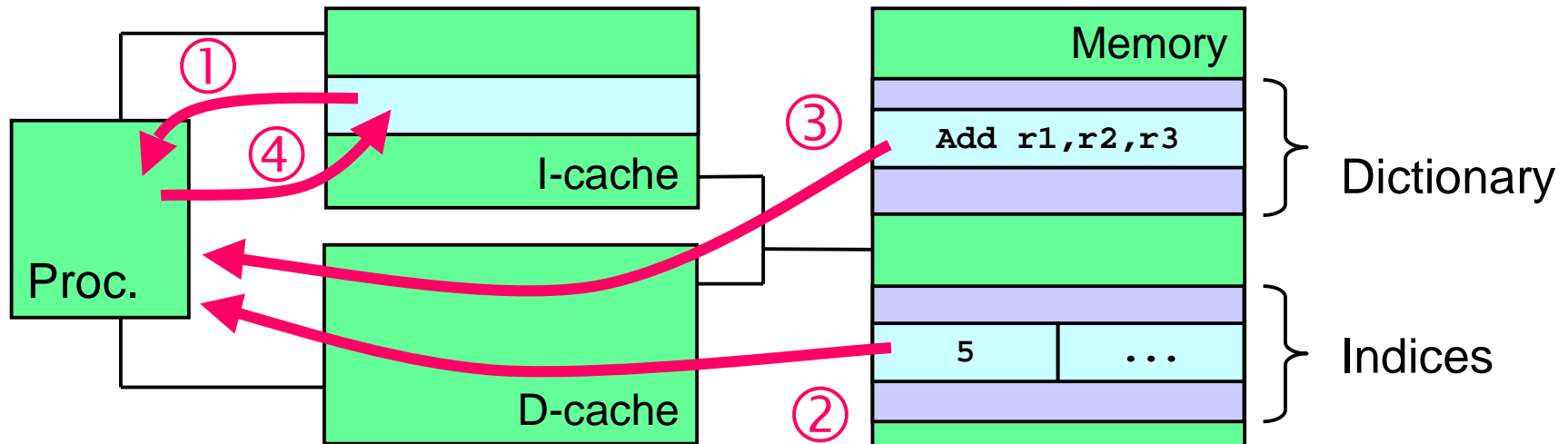
# Dictionary compression algorithm

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



# 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**



# Hardware support

---

- **Decompression exception**
  - Raise exception on I-cache miss
  - Exception not raised on native code section (allow hybrid programs)
  - Similar to Informing Memory [Horowitz98]
- **Store-instruction instruction**
  - MAJC, MIPS R10K

# Two software decompressors

---

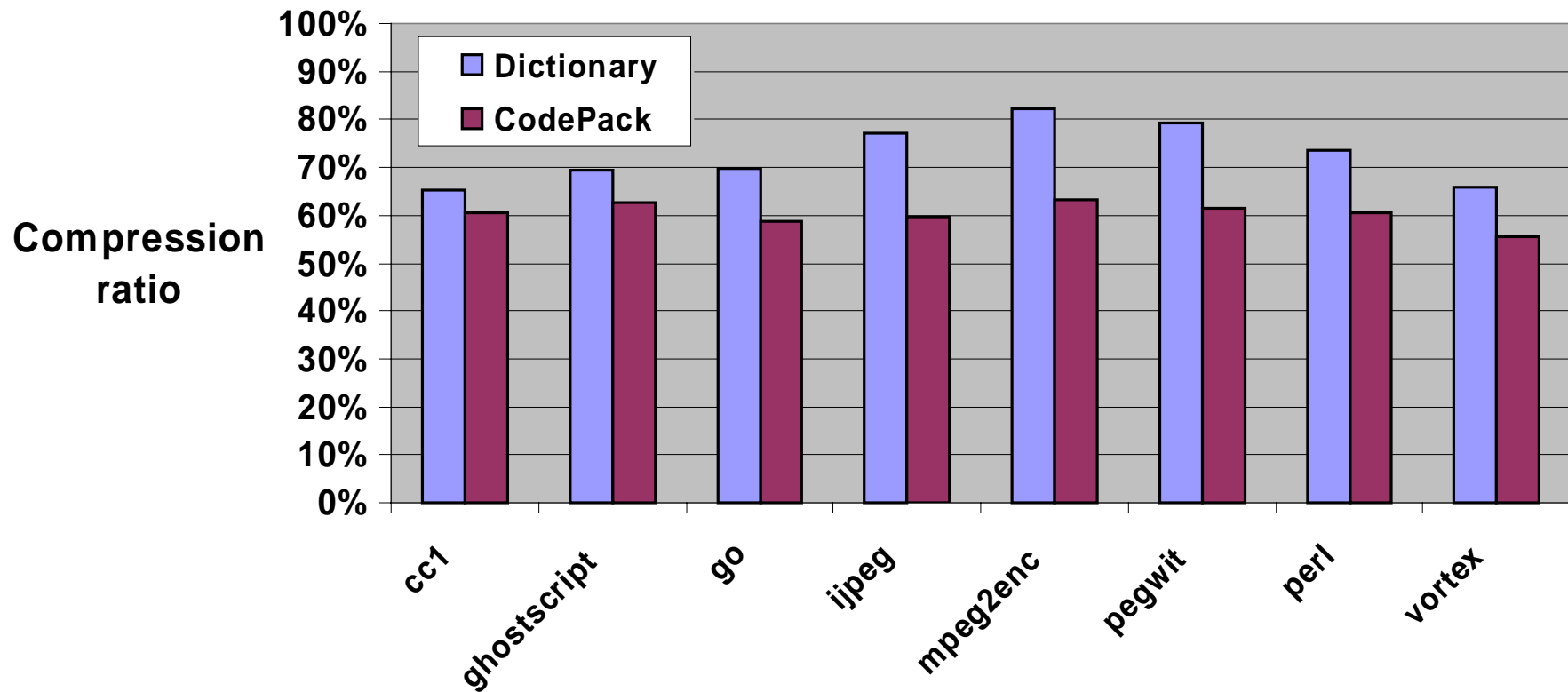
- **Dictionary**
  - Faster
  - Less compression
- **CodePack**
  - A software version of IBM's CodePack hardware
  - Slower
  - More compression

	<b>Dictionary</b>	<b>CodePack</b>
Codewords (indices)	Fixed-length	Variable-length
Decompress granularity	1 cache line	2 cache lines
Static instructions	43	174
Dynamic instructions	43	1042-1062
Decompression overhead	73-105 cycles	1235-1266 cycles

# Compression ratio

- $compression\ ratio = \frac{compressed\ size}{original\ size}$

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



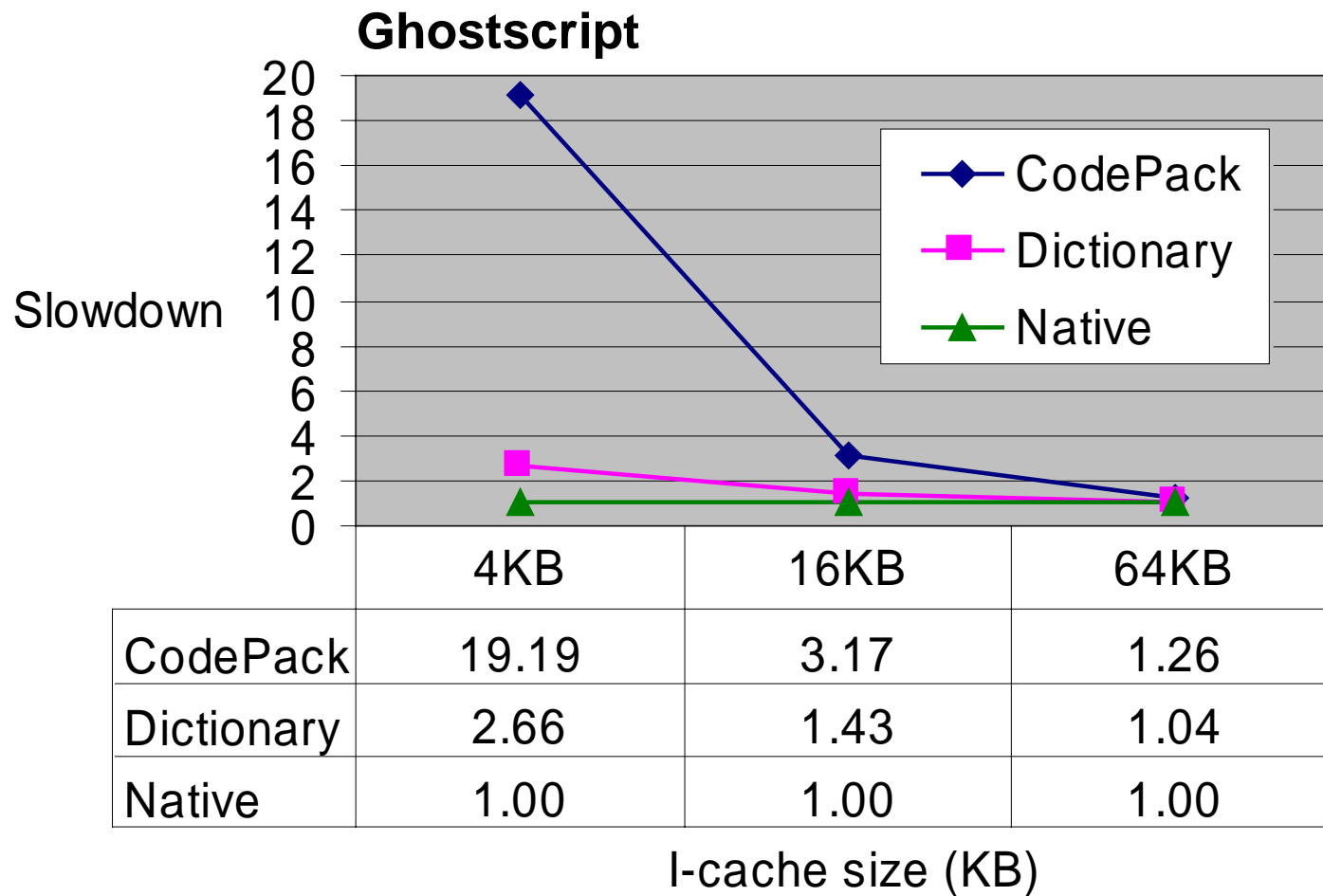
# Simulation environment

---

- **SimpleScalar**
- **Pipeline:** 5 stage, in-order
- **I-cache:** 4KB, 32B lines, 2-way
- **D-cache:** 8KB, 16B lines, 2-way
- **Memory:** embedded DRAM
  - 10 cycle latency
  - bus width = 1 cache line
  - 10x denser than SRAM caches
- **Performance: slowdown = 1 / speedup (1 = native code)**
- **Area results include:**
  - Main memory to hold program (compressed bytes, tables)
  - I-cache
  - Memory for decompressor optimizations (memorization, native code)

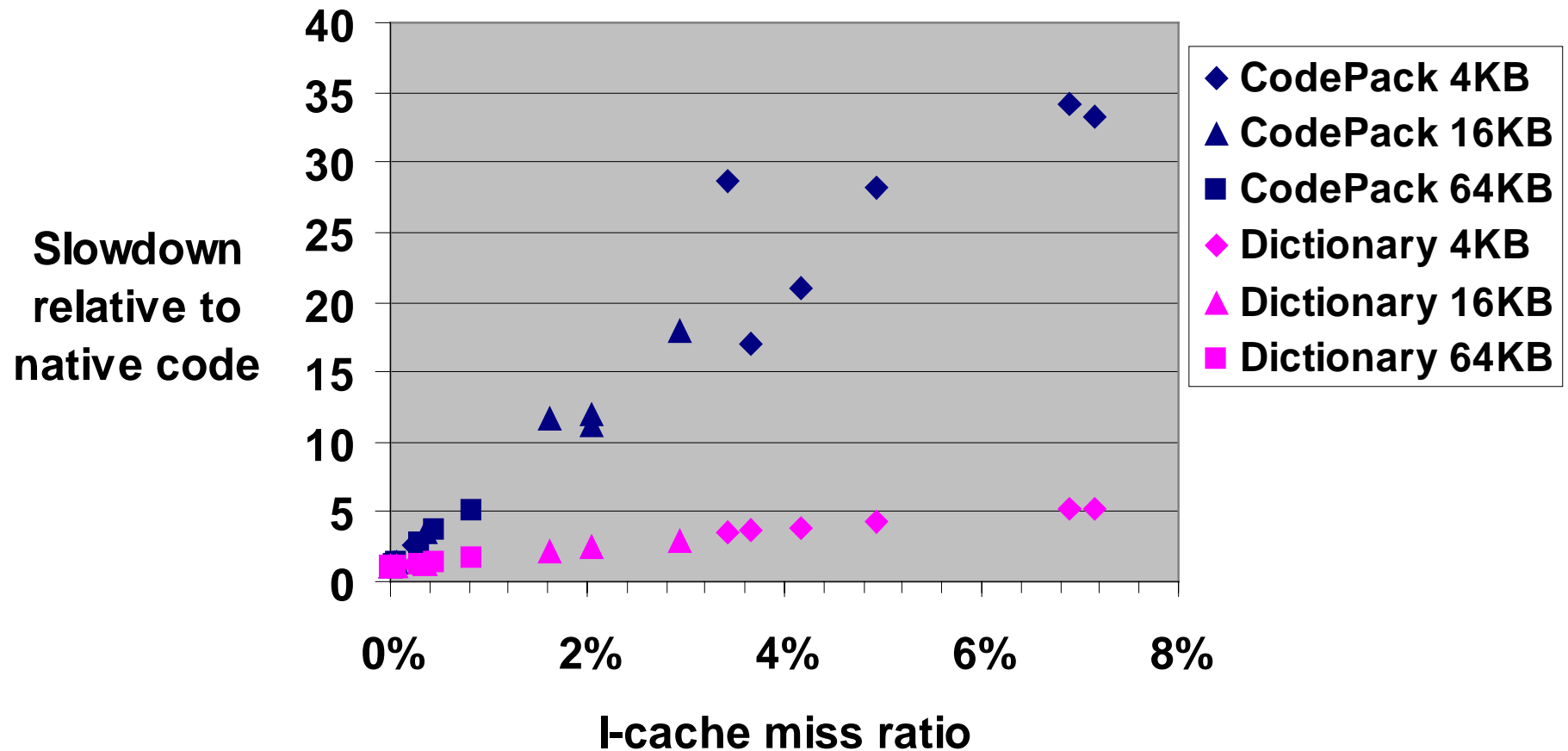
# Performance

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



# Cache miss

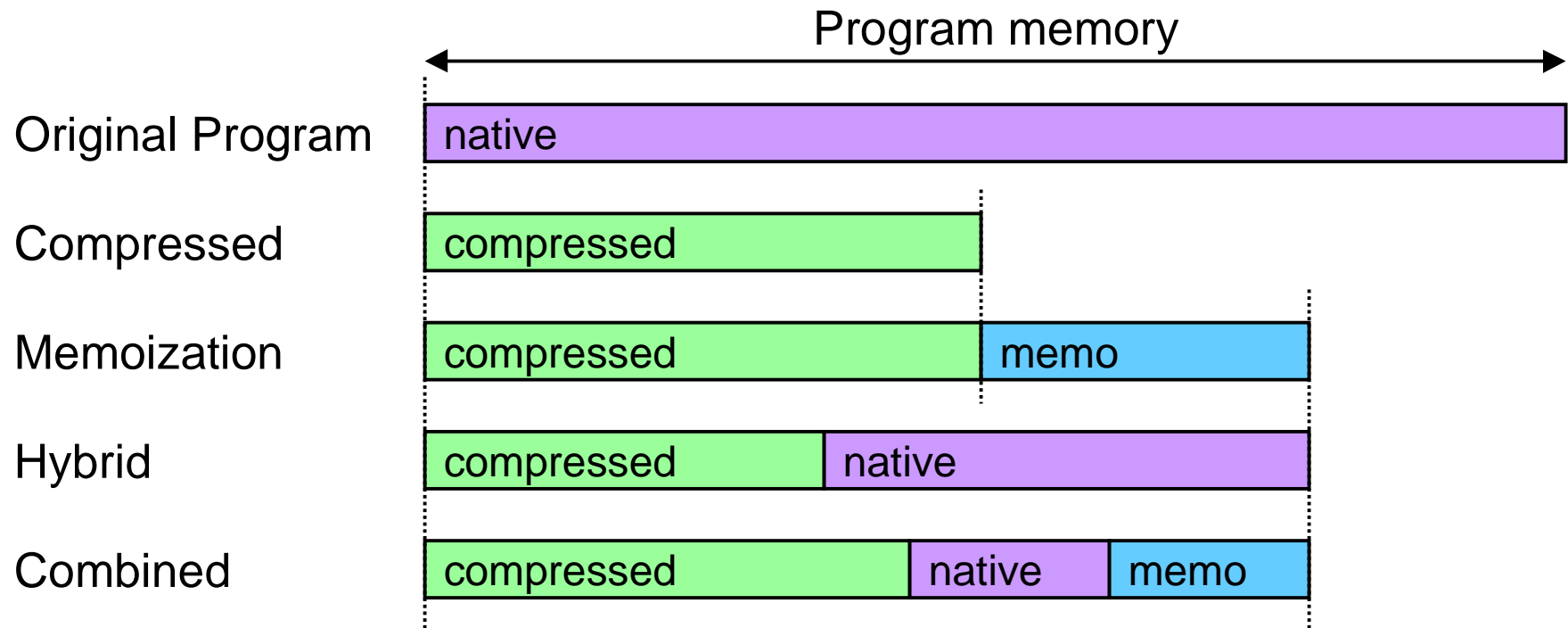
- Control slowdown by optimizing I-cache miss ratio
- Small change in miss ratio → large performance impact



# Two optimizations

---

- **Hybrid programs (static)**
  - Both compressed and native code
- **Memoization (dynamic)**
  - Cache recent decompressions in main memory
- **Both can be applied to any compression algorithm**



# Hybrid programs

---

- **Selective compression**

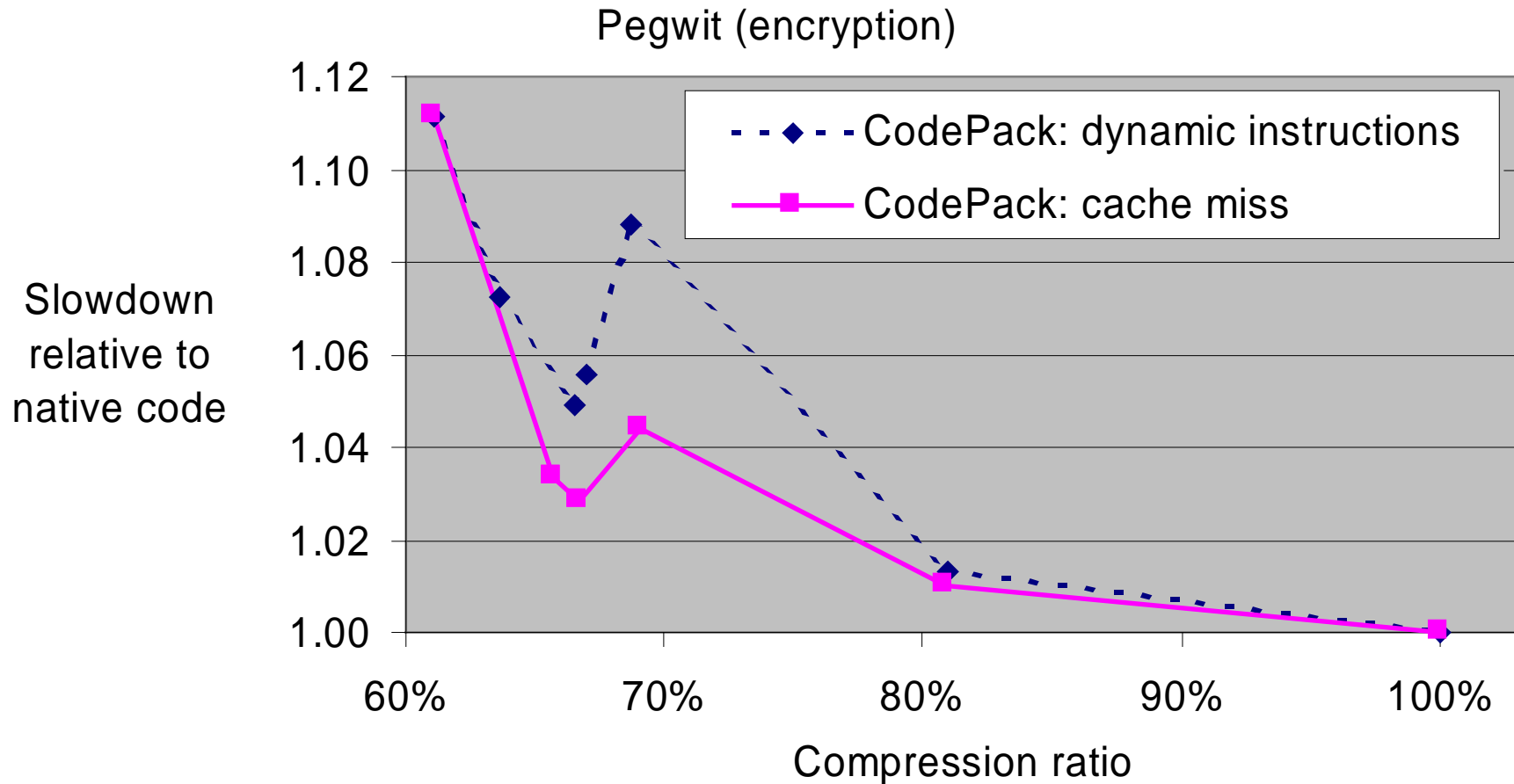
- Only compress some procedures
- Trade size for speed
- Avoid decompression overhead

- **Profile methods**

- Count dynamic instructions
  - Example: ARM/Thumb
  - Use when compressed code has more instructions
  - Reduce number of executed instructions
- Count cache misses **New!**
  - Example: CodePack
  - Use when compressed code has longer cache miss latency
  - Reduce cache miss latency

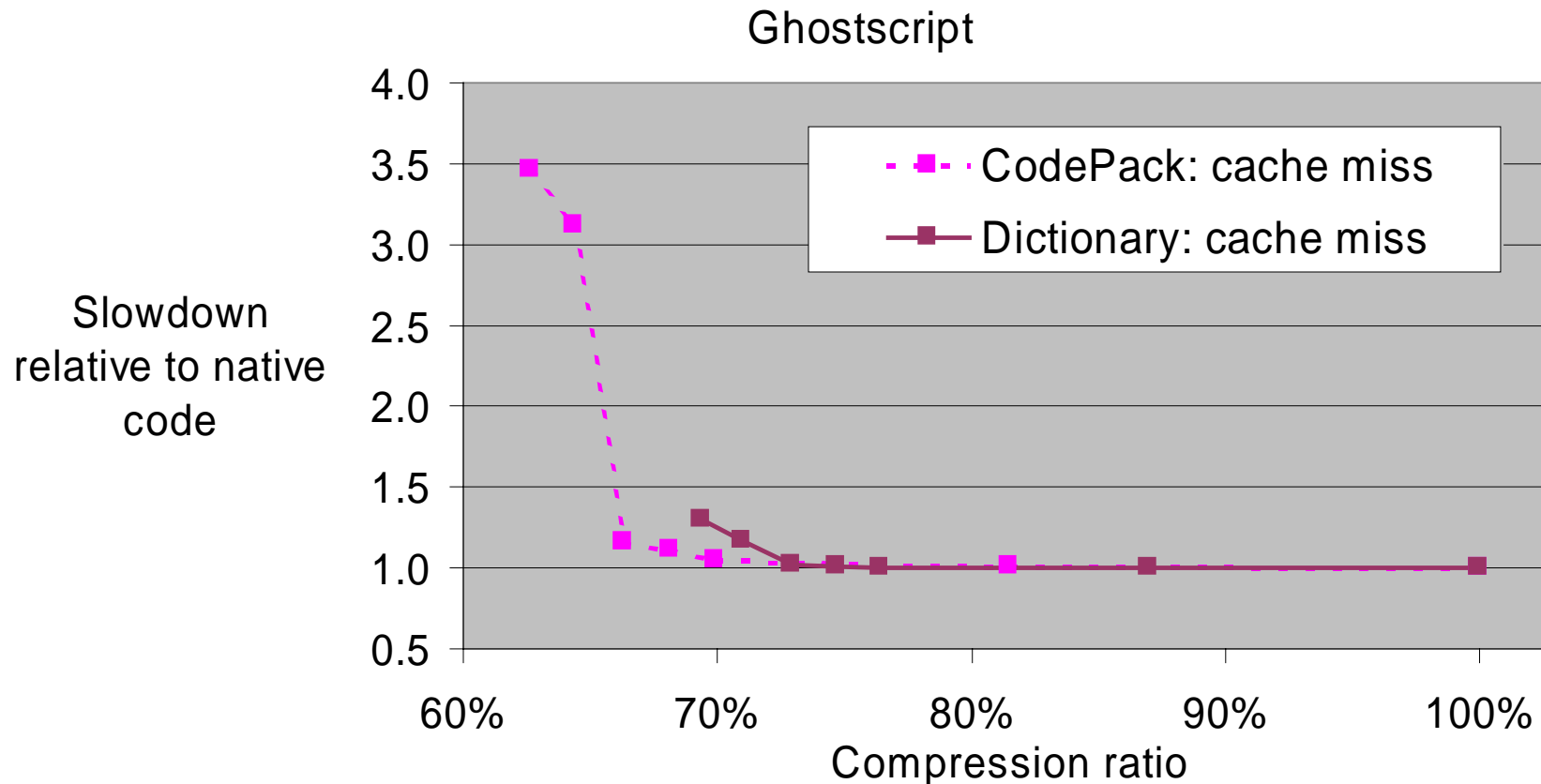
# Cache miss profiling

- **Cache miss profile reduces overhead 50%**
- **Loop-oriented benchmarks benefit most**
  - Profiles are different in loop regions



# CodePack vs. Dictionary

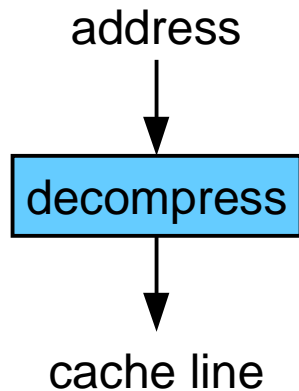
- **More compression may have better performance**
  - CodePack has smaller size than Dictionary compression
  - Even with some native code, CodePack is smaller
  - CodePack is faster due to using more native code



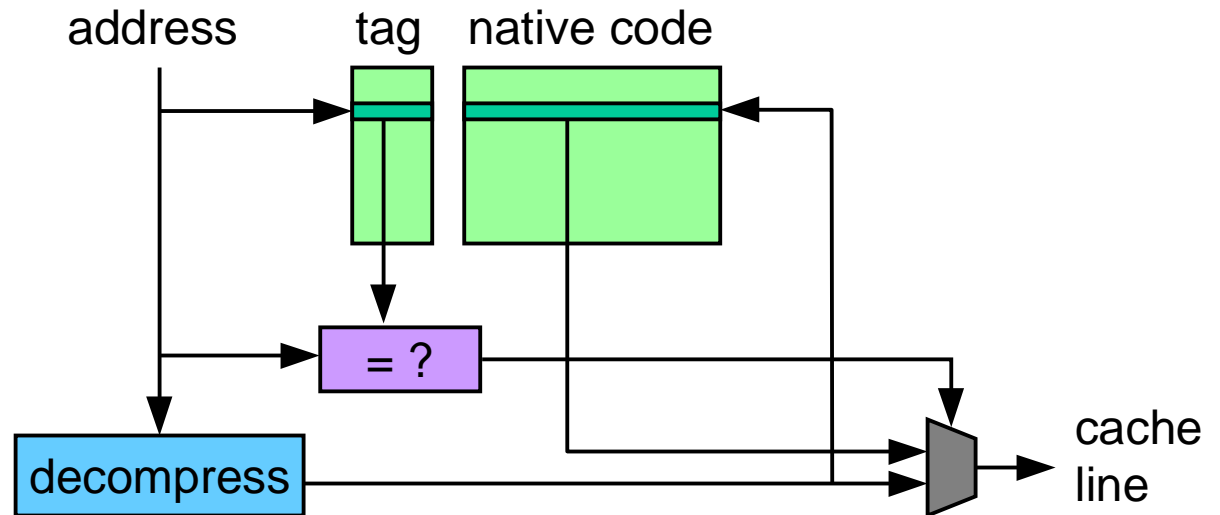
# Memoization

- **Reserve main memory for caching decompressed insns.**
  - Use high density DRAM to store more than I-cache in less area
  - Manage as a cache: data and tags
- **Algorithm**
  - Decompressor checks memo table before decompressing
  - On hit, copy instructions into I-cache (no decompression)
  - On miss, decompress into I-cache and update memo table

## No memoization

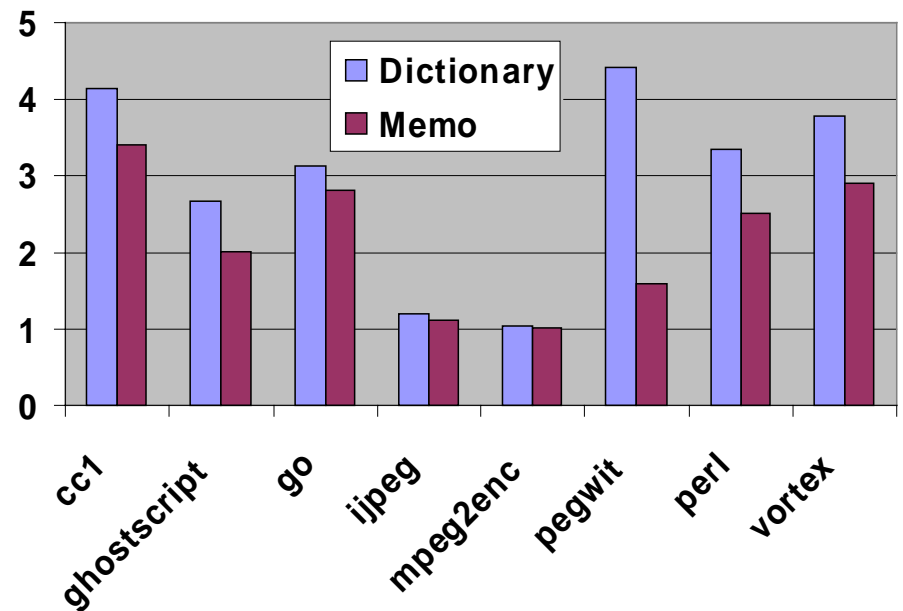
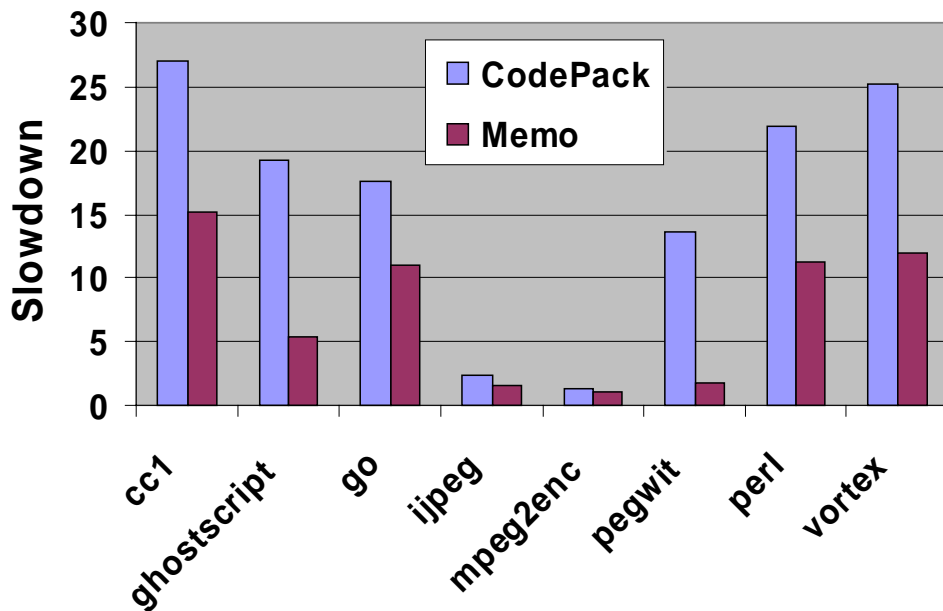


## With memoization



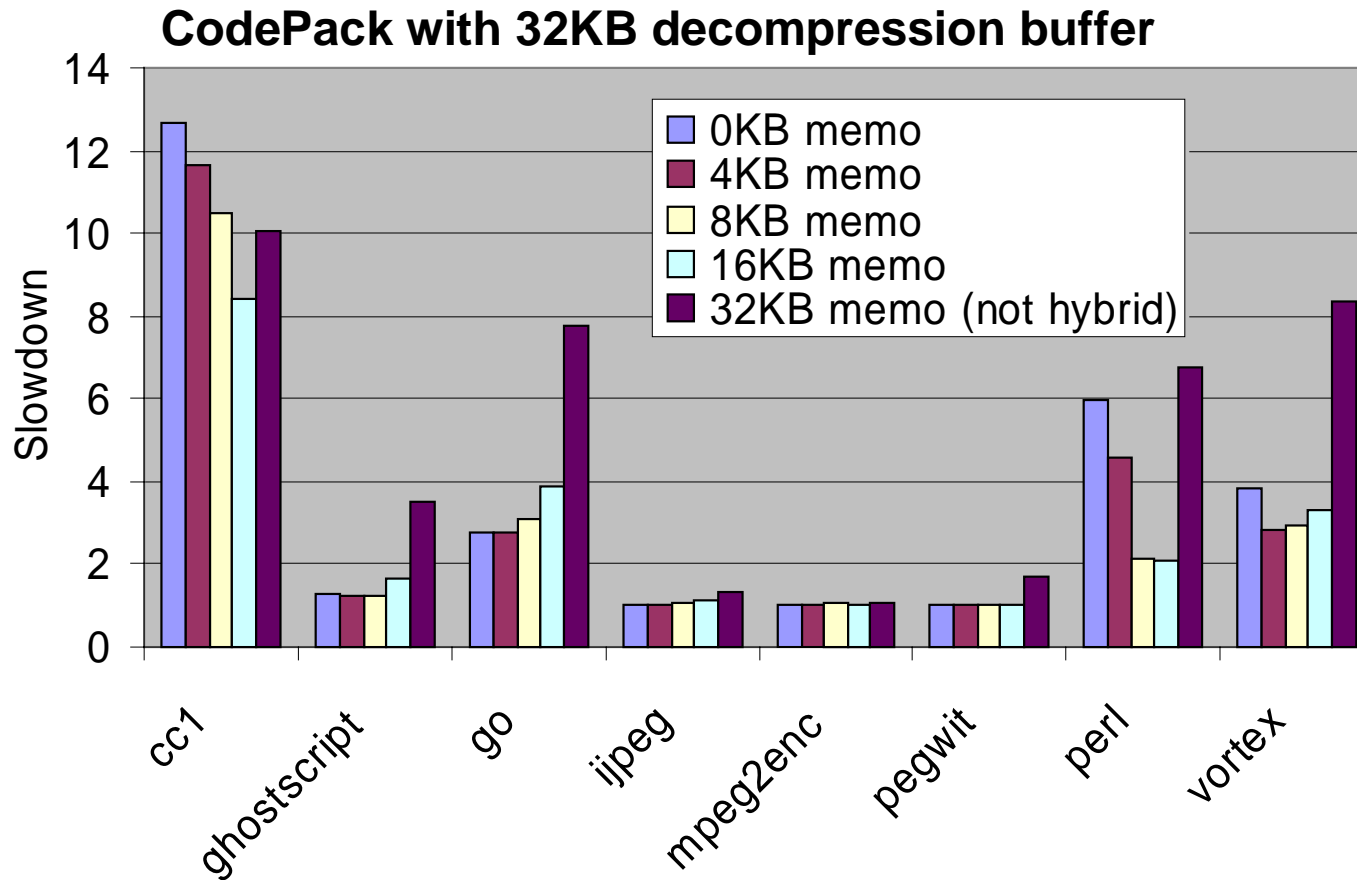
# Memoization results

- **16KB memoization table**
- **CodePack: large improvement**
- **Dictionary is already fast: small improvement**



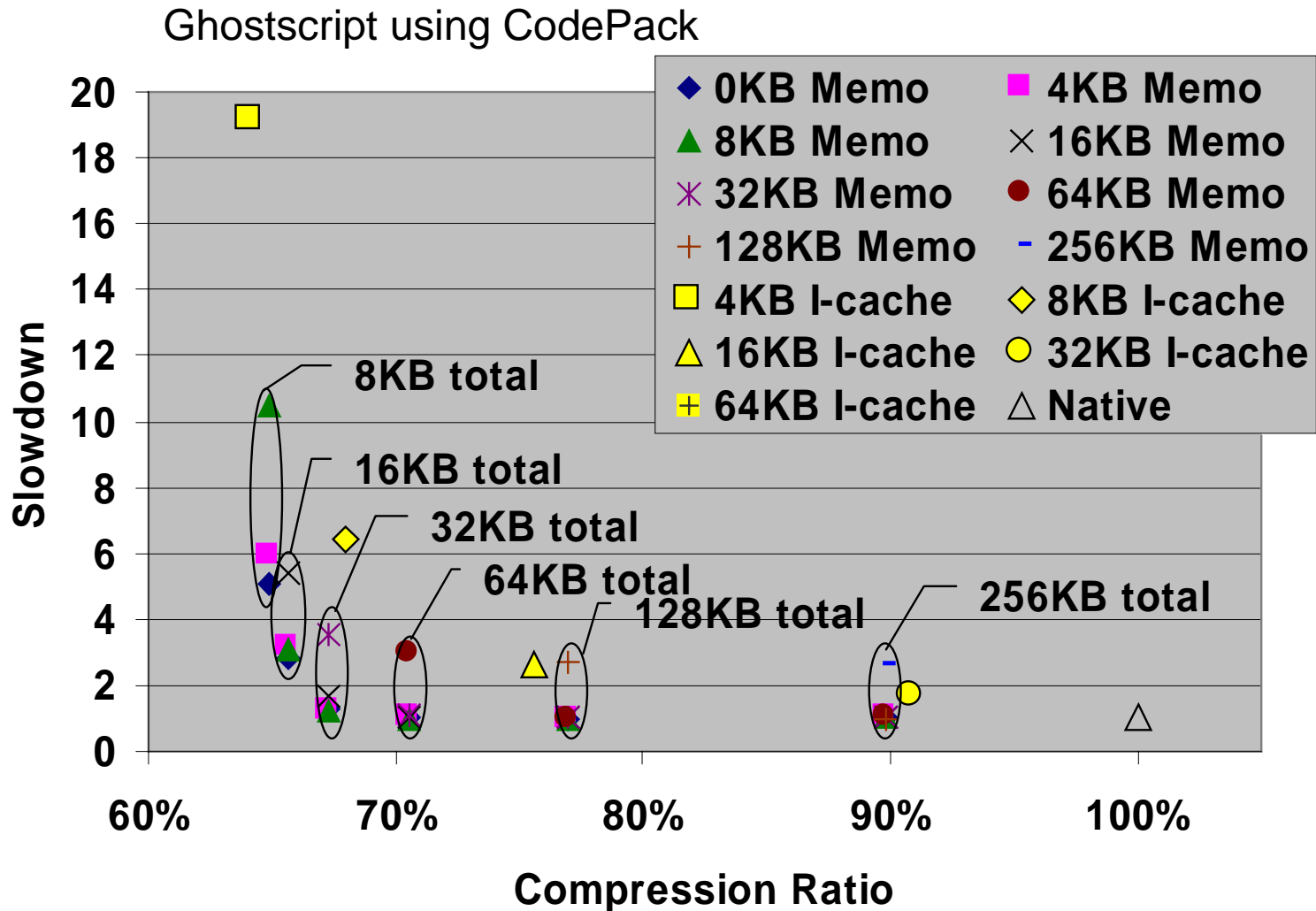
# Combined

- **Use memoization on hybrid programs**
  - Keep area constant. Partition memory for best solution.
- **Combined solution is often the best**



# Combined

- Adding DRAM is better than larger SRAM cache



# 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**
  - Cache size
  - Hybrid programs, memoization
- **Hybrid programs**
  - Use cache miss profile for loop-oriented benchmarks
- **Memoization**
  - A little memoization can be very effective on large working sets

# Future work

---

- **Compiler optimization for compression**
  - Improve compression ratio without affecting performance
- **Unify selective compression and code placement**
  - Reduce I-cache miss to improve performance
- **Energy consumption**
  - Increasing run-time may use too much energy
  - Improving bus utilization saves energy
- **Dynamic code generation: Crusoe, Dynamo**
  - Memory management, code stitching
- **Compression for high-performance**
  - Lower L2 miss rate

# Web page

---

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