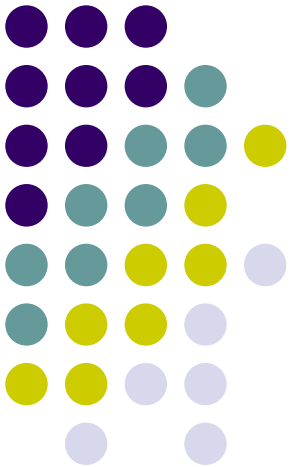
ECE 813
Survey Project Presentation

Code and Data Compression for Embedded Processors

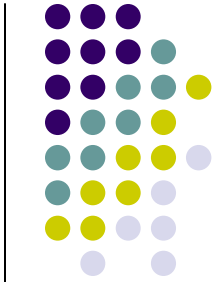


Awais M. Kamboh
kambohaw@msu.edu
Electrical and Computer Engineering
Michigan State University

MICHIGAN STATE UNIVERSITY

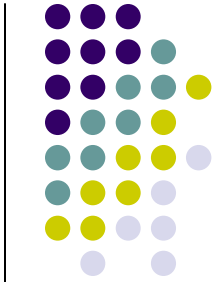
November 18th, 2006

Talk Organization



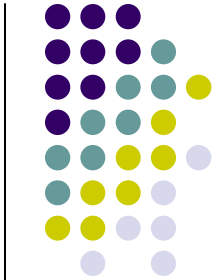
- What and why of compression
- Definition of terms
- Data and code compression
- Classification of techniques
- Resulting effects and advantages
- Overview of techniques
- Evaluation of methods
- Conclusions
- References
- Questions / Comments

What is Compression?



- The same amount of information can be stored using less space
- Executing fewer instructions and accessing external memory less frequently reduce energy needs.
- Storage media has always been limited
  - Saves money
  - Saves bandwidth

Definition of terms

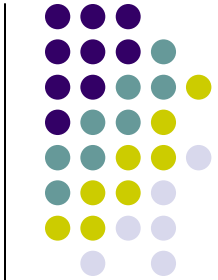


- Compression
- Theoretical background
  - Entropy

$$H(X) = -\sum_{x \in A_X} P(x) \log_2 P(x)$$

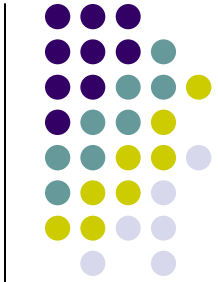
- Compression model
- Compression ratio
- Compaction

Data and Code Compression



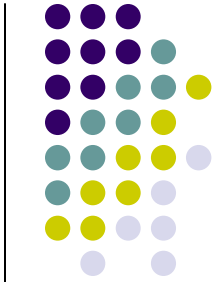
- Data Compression
  - Highly established
  - Communication systems
- Modeling techniques
  - Huffman coding
  - Arithmetic coding
  - Dictionary based methods
- Code Compression
  - Relatively unexplored
  - No standard nomenclature

Classification of techniques



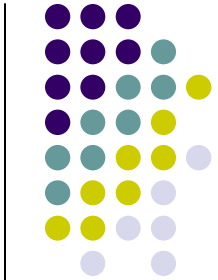
1. Basic size-reduction principle
2. Type of application hardware
3. Implementation of decompressor
4. Code equivalence
5. Type of subject program code
6. Granularity of program code

Classification of techniques (contd.)



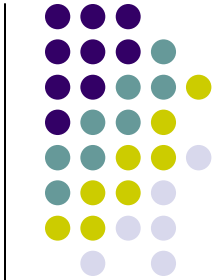
- Basic size-reduction principle
  - Without a decompressor
  - Decompressor is required
- Target hardware
  - Classification not applicable
  - System on a chip / embedded systems
  - Mainframe computers
- Code equivalence
  - Completely equivalent
  - Equivalent as seen by CPU
  - Functionally equivalent

Classification of techniques (contd.)



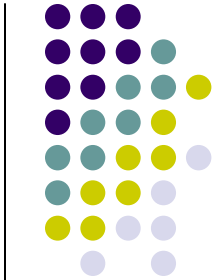
- Implementation of decompressor
  - Software based
  - Extra RAM required
  - Hardware decoding with CPU modification
  - Hardware decoding on cache based hardware
- Type of subject program code
  - Source code
  - Intermediate representation
  - Assembly
  - Machine
  - Special type

Classification of techniques (contd.)



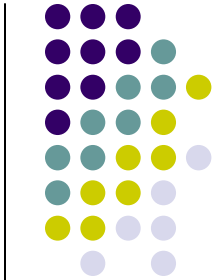
- Granularity of program code
  - Bit sequence
  - Character
  - Instruction
  - Block based (basic-block / cache-line)
    - Small (e.g. cache line)
    - Large (e.g. page)
  - Procedure
  - Program

Resulting effects



1. Size of compressed code
2. Number of executed instructions
3. Execution speed
4. Compression time / complexity
5. Decompression time / complexity
6. Behavior safety
7. Mutual effects

Overview of techniques

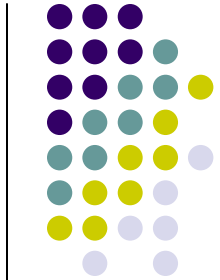


1. Cooper and McIntosh
2. Squeeze
3. Narrow-word Instructions for energy reduction
4. Code compression for cache based RISC
5. Araujo and Pannain
6. Fraser and Proebsting

Cooper and McIntosh

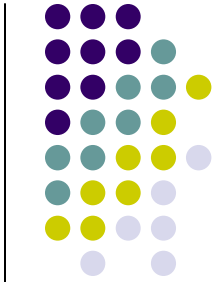
- Classification
  - Principle: No need of decompression
  - Target: Any hardware
  - Decompressor: Compactor, Software based
  - Equivalence: Functionally equivalent
  - Subject code: Computations on intermediate representation code
  - Granularity: Whole program at once
- Effects
  - Code size: 0.95 best, average 0.85
  - # of executed instructions: increase (requires additional instr.)
  - Speed: decrease
  - Compression complexity: Medium
  - Decompression: Not needed
  - Energy consumption: Probably increase
  - Mutual effects: Decrease in code-size results in smaller speed and more executed instructions

Squeeze



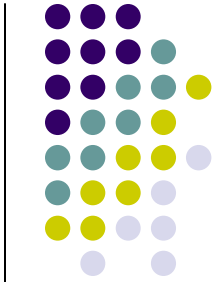
- Classification
  - Principle: No need of decompression
  - Target: Any hardware
  - Decompressor: Software based compactor
  - Equivalence: Functionally equivalent
  - Subject code: Machine code
  - Granularity: Whole program at once
- Effects
  - Code size: 0.7 best
  - # of executed instructions: Increase / decrease
  - Speed: Decrease / increase
  - Compression complexity: High
  - Decompression: Not needed
  - Behavior: Equivalent, could have problems with timing / real-time
  - Energy consumption: Probably decrease
  - Mutual effects: Behavior may be a problem, smaller code with fewer instructions gives better energy consumption.

Narrow-word Instructions for energy reduction



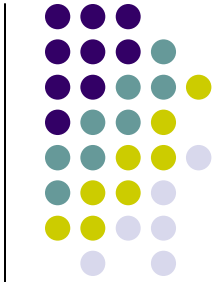
- Classification
  - Principle: Decompression by hardware
  - Target: Primarily RISC
  - Decompressor: Several modifications
  - Subject code: Computations on machine code
  - Granularity: Whole program at once
- Effects
  - Code size: Decreased
  - # of executed instructions: Decrease
  - Speed: No effect
  - Compression complexity: Simple
  - Decompression: Simple
  - Behavior: Equivalent
  - Energy consumption: 0.5
  - Mutual effects: More complex decompression may increase energy

Compression for cache based RISC



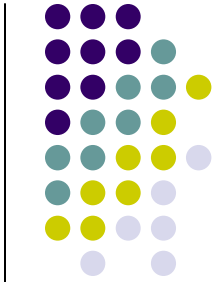
- Classification
  - Principle: Hardware decompression
  - Target: RISC architecture with cache
  - Decompressor: Line address table
  - Subject code: Machine code
  - Granularity: Cache line blocks
- Effects
  - Code size: 0.73 using Huffman
  - # of executed instructions: No effect
  - Speed: Slight decrease
  - Compression complexity: Medium
  - Decompression: Medium
  - Behavior: Equivalent, timing issues
  - Energy consumption: Probably increase
  - Mutual effects: Better compression scheme gives more compression, may lead to slow speed and high energy

Araujo and Pannain



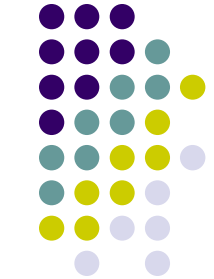
- Classification
  - Principle: Decompression by hardware
  - Target: Primarily RISC, special application to DSP
  - Decompressor: Special logic for the instruction fetch of the CPU
  - Equivalence: Functionally Equivalent
  - Subject code: Machine code
- Effects
  - Code size: 0.43, (0.75 for DSP)
  - # of executed instructions: No effect
  - Speed: No effect
  - Compression complexity: Standard Huffman
  - Decompression: 1.08 (1.47 for DSP)
  - Behavior: Equivalent
  - Energy consumption: Probably increase
  - Mutual effects: Compression ratio is inversely proportional to decompression engine complexity

Fraser and Proebsting



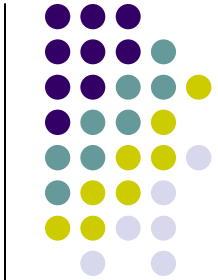
- Classification
  - Principle: Interpreter is required
  - Target: Mainframe computers
  - Decompressor: Complicated decompression mechanism
  - Equivalence: Functional equivalence
  - Subject code: Intermediate representation
- Effects
  - Code size: 0.2 best for wire code, 0.6 for BRISC
  - # of executed instructions: Input is in another form
  - Speed: Decreases
  - Compression complexity: High
  - Decompression: High
  - Energy consumption: Probably increase
  - Mutual effects: Very high compression ratio have a tradeoff with compression/decompression complexity

Evaluation of methods



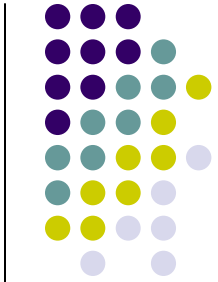
Method	Coding	Code Size	# of Instr.	Speed	Comp	Decomp	Energy
Cooper and McIntosh	Software	0.95	↑	↓	M	N/A	?
Squeeze	Software	0.70	↓	↑	H	N/A	?
Narrow Word	Dictionary	↓	↓	-	L	-	0.5
Cache based RISC	Huffman	0.73	-	↓	M	M	↓
Araujo & Pannain	Dictionary	0.43	-	-	M	1.08	?
Fraser & Proebsting	Dictionary	0.2	N/A	↓	H	H	?

Conclusions



- Code compression uses the same theory as data compression.
- There is no standard nomenclature and no standard classification of code compression techniques
- There is no “best” method for code-size reduction.
- Different methods perform well in different usage contexts.
- Most methods result in a trade-off between code-size, hardware complexity and energy.

References



1. Ahmad Zmily and Christos Kozyrakis, "Simultaneously Improving Code Size, Performance, and Energy in Embedded Processors," *Proceedings of the Conference on Design Automation and Test in Europe (DATE),* March 2006.
2. Árpád Beszédes , Rudolf Ferenc , Tibor Gyimóthy , André Dolenc , Konsta Karsisto, “Survey of code-size reduction methods,” *ACM Computing Surveys (CSUR)*, v.35 n.3, p.223-267, September 2003.
3. Haris Lekatsas , Jorg Henkel , Venkata Jakkula , Srimat Chakradhar, “A Unified Architecture for Adaptive Compression of Data and Code on Embedded Systems,” *Proceedings of the 18th International Conference on VLSI Design held jointly with 4th International Conference on Embedded Systems Design (VLSID'05)*, p.117-123, January 2005.
4. Chang Hong Lin , Yuan Xie , Wayne Wolf, “LZW-Based Code Compression for VLIW Embedded Systems,” *Proceedings of the conference on Design, automation and test in Europe*, p.30076, February 2004
5. Martin Thuresson, Per Stenstrom, “Evaluation of extended dictionary-based static code compression schemes” *Conference On Computing Frontiers Proceedings of the 2nd conference on Computing frontiers*, p.77 - 86, 2005
6. Waldman, I. and Pinter, S. S. “Profile-driven compression scheme for embedded systems.” In *Proceedings of the 3rd Conference on Computing Frontiers.* p.95-104, May 2006.

Questions & Comments