A DISE Implementation of Dynamic Code Decompression

Marc L. Corliss
with E Christopher Lewis and Amir Roth
Department of Computer and Information Science
University of Pennsylvania
Philadelphia, Pennsylvania USA
{mcorliss,lewis,amir}@cis.upenn.edu
Motivation

Code size is important:

- Embedded systems
  - Cache and memory size limited
- Mobile computing
  - Network bandwidth limited
- General-purpose computing
  - Cache size effectively limited
Motivation

Code size is important:

- Embedded systems
  - Cache and memory size limited
- Mobile computing
  - Network bandwidth limited
- General-purpose computing
  - Cache size effectively limited

One solution: code compression
## Code Compression

<table>
<thead>
<tr>
<th>Compression algorithm</th>
<th>effectiveness</th>
<th>overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statistical</td>
<td>good +</td>
<td>high -</td>
</tr>
<tr>
<td>Dictionary</td>
<td>slightly worse -</td>
<td>low +</td>
</tr>
</tbody>
</table>
# Code Compression

## Compression algorithm

<table>
<thead>
<tr>
<th></th>
<th>effectiveness</th>
<th>overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statistical</td>
<td>good +</td>
<td>high -</td>
</tr>
<tr>
<td>Dictionary</td>
<td>slightly worse -</td>
<td>low +</td>
</tr>
</tbody>
</table>

## Decompressor position

<table>
<thead>
<tr>
<th>Inst-fill path</th>
<th>proc. core</th>
<th>penalty</th>
<th>I$</th>
<th>addr. trans.</th>
</tr>
</thead>
<tbody>
<tr>
<td>unmod. +</td>
<td>cache miss +</td>
<td>uncomp. -</td>
<td>yes -</td>
<td></td>
</tr>
</tbody>
</table>
Code Compression

Compression algorithm

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Effectiveness</th>
<th>Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statistical</td>
<td>good +</td>
<td>low -</td>
</tr>
<tr>
<td>Dictionary</td>
<td>slightly worse -</td>
<td>high +</td>
</tr>
</tbody>
</table>

Decompressor position

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Inst-fill path</td>
<td>unmod. +</td>
<td>cache miss +</td>
<td>uncomp. -</td>
<td>yes -</td>
</tr>
<tr>
<td>Post-fetch</td>
<td>minor mods -</td>
<td>all fetched -</td>
<td>comp. +</td>
<td>no +</td>
</tr>
</tbody>
</table>

Diagram:
- Memory (compressed)
- I$ (compressed)
- Decompression
- Fetch
- Rest of pipeline
Approach

Dictionary compression with a post-fetch decompressor

- Not a new approach [Lefurgy+ 97]
- Unique features
- Unique implementation strategy
- Evaluating other impacts
Approach

Dictionary compression with a post-fetch decompressor

- Not a new approach [Lefurgy+ 97]
- Unique features
- Unique implementation strategy
- Evaluating other impacts

We use Dynamic Instruction Stream Editing (DISE) [ISCA 03]

- Defines application-specific instruction behavior
Talk Outline

- Introduction ✓
- Dynamic Instruction Stream Editing (DISE)
- (De)Compression with DISE
- Results
  - Code size
  - Performance
  - Energy
- Summary
What is Dynamic Instruction Stream Editing (DISE)?

- Decoder-based instruction macro-expansion
  (Expands fetched instructions to instruction sequences)
What is Dynamic Instruction Stream Editing (DISE)?

- Decoder-based instruction macro-expansion
  (Expands fetched instructions to instruction sequences)

Example: Memory fault isolation

```plaintext
add a4, a4, #1
stq a4, 0(a2)
```
What is Dynamic Instruction Stream Editing (DISE)?

- Decoder-based instruction macro-expansion
  (Expands fetched instructions to instruction sequences)

Example: Memory fault isolation

```
add a4, a4, #1
stq a4, 0(a2)
```

```
add a4, a4, #1
move dr1, a2
srli dr2, 26, dr1
cmpeq dr2, dr3, dr2
bne dr2, error
stq a4,0(dr1)
```
Normal Processor

I$ (compressed) → decoder → execution engine in-order or out-of-order → D$

Normal processor pipeline
DISE Structures

Processor pipeline with DISE structures
Pattern Table (PT)

- Examines each instruction at decode
- Identifies matching instructions
  - E.g. All loads and stores
- Overrides normal decoding
- Indexes into RT on a match
Replacement Table (RT)

- Outputs predecoded instruction sequences
- Instructions may have “holes”

Ex: \text{stq } a4, 0(a2) \rightarrow \text{move } dr1, \_ \\
\text{srl} dr1, 26, dr1 \\
\text{cmpeq } dr1, dr2, dr1 \\
\text{bne } dr1, \text{error} \\
\text{stq } \_, \_, \_
DISE Structures

Instantiator
- Fills holes with information from fetched instruction

Ex: stq $a4, 0($a2)

move $dr1, $a2
srli $dr1, 26, $dr1
cmpeq $dr1, $dr2, $dr1
bne $dr1, error
stq $a4, 0($a2)
DISE Structures

**DISE Controller**
- Programs the PT/RT
- Abstracts the PT/RT internal representation
- Manages virtualization
DISE Summary

Characteristics:

- Programmable
- General-purpose
- Leverages existing technology
- Applicable for CISC or RISC
DISE Summary

Characteristics:
- Programmable
- General-purpose
- Leverages existing technology
- Applicable for CISC or RISC

Uses:
- Memory fault isolation [ISCA 03]
- Path profiling [TR 02]
- Code decompression [LCTES 03]
DISE (De)Compression

- Introduction ✓
- Dynamic Instruction Stream Editing (DISE) ✓
- (De)Compression with DISE
- Results
  - Code size
  - Performance
  - Energy
- Summary
Example

Static code

...  

lda a2, 8(a2)
ldq a4, 0(a2)
cmplt a4, a0, a5

...

lda a3, -8(a3)
ldq a4, 0(a3)
cmplt a4, a0, a5

...
Example

**Static code**

...  
lda a2, 8(a2)  
ldq a4, 0(a2)  
cmplt a4, a0, a5  
...  
lda a3, -8(a3)  
ldq a4, 0(a3)  
cmplt a4, a0, a5  
...

**Dictionary (RT)**

<table>
<thead>
<tr>
<th></th>
<th>lda a2, 8(a2)</th>
<th>ldq a4, 0(a2)</th>
<th>cmplt a4, a0, a5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>lda a3, -8(a3)</td>
<td>ldq a4, 0(a3)</td>
<td>cmplt a4, a0, a5</td>
</tr>
</tbody>
</table>

Put instruction sequences into RT
Replace instruction sequences with **codewords**
Example

Static code

... reserved #0 ...

reserved #3 ...

Eliminated 4 instructions

Dictionary (RT)

<table>
<thead>
<tr>
<th></th>
<th>lda a2, 8(a2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>ldq a4, 0(a2)</td>
</tr>
<tr>
<td></td>
<td>cmplt a4, a0, a5</td>
</tr>
<tr>
<td>3</td>
<td>lda a3, -8(a3)</td>
</tr>
<tr>
<td></td>
<td>ldq a4, 0(a3)</td>
</tr>
<tr>
<td></td>
<td>cmplt a4, a0, a5</td>
</tr>
</tbody>
</table>
Example

Static code

... 
reserved #0
...
reserved #3
...

<table>
<thead>
<tr>
<th>Dictionary (RT)</th>
<th>0</th>
<th>lda a2, 8(a2)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>ldq a4, 0(a2)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>cmplt a4, a0, a5</td>
</tr>
<tr>
<td>3</td>
<td>lda a3, -8(a3)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>ldq a4, 0(a3)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>cmplt a4, a0, a5</td>
</tr>
</tbody>
</table>

Observation: Many sequences are similar but not identical
Parameterization

Static code

... reserved #0 ...
... reserved #3 ...

Dictionary (RT)

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>lda a2, 8(a2)  ldq a4, 0(a2)  cmplt a4, a0, a5</td>
</tr>
<tr>
<td>3</td>
<td>lda a3, -8(a3) ldq a4, 0(a3)  cmplt a4, a0, a5</td>
</tr>
</tbody>
</table>
Parameterization

Static code

... reserved #0 ...

... reserved #3 ...

Dictionary (RT)

<table>
<thead>
<tr>
<th>0</th>
<th>lda a2, 8(a2)</th>
<th>ldq a4, 0(a2)</th>
<th>cmplt a4, a0, a5</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>lda a3, -8(a3)</td>
<td>ldq a4, 0(a3)</td>
<td>cmplt a4, a0, a5</td>
</tr>
</tbody>
</table>

- Parameterize register in RT entries
Parameterization

Static code

... reserved a2, #0 ...

... reserved a3, #3 ...

Dictionary (RT)

<table>
<thead>
<tr>
<th></th>
<th>lda P1, 8(P1)</th>
<th>ldq a4, 0(P1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>cmplt a4, a0, a5</td>
</tr>
<tr>
<td>3</td>
<td>lda P1, -8(P1)</td>
<td>ldq a4, 0(P1)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>cmplt a4, a0, a5</td>
</tr>
</tbody>
</table>

- Parameterize register in RT entries
Parameterization

Static code

...  
reserved a2, #0  
...  
reserved a3, #3  
...  

Dictionary (RT)

|   | lda P1, 8(P1)  
|---|---------------  
| 0 | ldq a4, 0(P1)  
|   | cmplt a4, a0, a5  
| 3 | lda P1, -8(P1)  
|   | ldq a4, 0(P1)  
|   | cmplt a4, a0, a5  

- Parameterize register in RT entries
- Parameterize immediate in RT entries
Parameterization

Static code

... reserved a2, 8, #0
... reserved a3, -8, #3
... 

Dictionary (RT)

<table>
<thead>
<tr>
<th></th>
<th>lda P1, P2(P1)</th>
<th>ldq a4, 0(P1)</th>
<th>cmplt a4, a0, a5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>lda P1, P2(P1)</td>
<td>ldq a4, 0(P1)</td>
<td>cmplt a4, a0, a5</td>
</tr>
</tbody>
</table>

- Parameterize register in RT entries
- Parameterize immediate in RT entries
Parameterization

Static code

... reserved a2, 8, #0 ...

... reserved a3, -8, #3 ...

Dictionary (RT)

<table>
<thead>
<tr>
<th></th>
<th>lda P1, P2(P1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>ldq a4, 0(P1)</td>
</tr>
<tr>
<td></td>
<td>cmplt a4, a0, a5</td>
</tr>
<tr>
<td>3</td>
<td>lda P1, P2(P1)</td>
</tr>
<tr>
<td></td>
<td>ldq a4, 0(P1)</td>
</tr>
<tr>
<td></td>
<td>cmplt a4, a0, a5</td>
</tr>
</tbody>
</table>

- Parameterize register in RT entries
- Parameterize immediate in RT entries
- Eliminate redundant dictionary entry
Parameterization

Static code

... reserved a2, 8, #0 ...
... reserved a3, -8, #0 ...

Dictionary (RT)

| 0 | lda P1, P2(P1)   |
|   | ldq a4, 0(P1)   |
|   | cmplt a4, a0, a5 |

- Parameterize register in RT entries
- Parameterize immediate in RT entries
- Eliminate redundant dictionary entry
Parameters and Branches

Static code

... 

\texttt{ldq a5, 0(a4)}

\texttt{bne a5, -64}

... 

... 

\texttt{ldq a5, 0(a4)}

\texttt{bne a5, -64}

...
Parameters and Branches

Static code

... 
reserved #0
... 
reserved #0
... 

Dictionary (RT)

<table>
<thead>
<tr>
<th>0</th>
<th>ldq a5, 0(a4)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>bne a5, -64</td>
</tr>
</tbody>
</table>

The offset is now wrong for both codewords

**Problem:** offsets might not be identical anymore!
Parameters and Branches

Static code

... reserved off1, #0

... reserved off2, #0

... The offset is now wrong for both codewords

Problem: offsets might not be identical anymore!

Solution: parameterize branch offsets

Dictionary (RT)

<table>
<thead>
<tr>
<th></th>
<th>ldq a5, 0(a4)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>bne a5, P1</td>
</tr>
</tbody>
</table>
Parameterization Summary

Benefits

+ Can make more effective use of dictionary
+ Can compress PC-relative branches
Parameterization Summary

Benefits

+ Can make more effective use of dictionary
+ Can compress PC-relative branches

How many parameters should be used?

+ More parameters give better compression
- More parameters increase codeword size

One possible encoding of a DISE codeword:

<table>
<thead>
<tr>
<th>31</th>
<th>26 25</th>
<th>21 20</th>
<th>16 15</th>
<th>11 10</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>reserved</td>
<td>param1</td>
<td>param2</td>
<td>param3</td>
<td>RT index</td>
<td></td>
</tr>
</tbody>
</table>
Dictionary construction algorithm

- Enumerate all static instruction sequences
- Iterate
  - Order sequences by (benefit - cost)
    - Benefit = bytes reduced from the static code
    - Cost = bytes of the dictionary entry
  - Greedily select best sequence
    - Combine using parameterization
    - Otherwise make a new entry
- Terminate when cost outweighs benefit
Results

- Introduction ✓
- Dynamic Instruction Stream Editing (DISE) ✓
- (De)Compression with DISE ✓

Results
  - Code size
  - Performance
  - Energy

Summary
Methodology

- Simulator
  - Cycle-level simulation tool built using SimpleScalar
  - Energy modeled using Wattch framework and CACTI-3

General-purpose
- 4-way superscalar, 12-stage out-of-order pipeline
- 32K inst cache and data cache, 1MB L2

Embedded
- 2-way superscalar, 5-stage in-order pipeline
- 16K inst cache and data cache, no L2

Benchmarks
- 12 SPEC2000 integer benchmarks
- 16 Mediabench benchmarks
Methodology

- Simulator
  - Cycle-level simulation tool built using SimpleScalar
  - Energy modeled using Wattch framework and CACTI-3

- Machine configurations
  - General-purpose
    - 4-way superscalar, 12-stage out-of-order pipeline
    - 32K inst cache and data cache, 1MB L2
  - Embedded
    - 2-way superscalar, 5-stage in-order pipeline
    - 16K inst cache and data cache, no L2

Benchmarks
- 12 SPEC2000 integer benchmarks
- 16 Mediabench benchmarks
Methodology

- Simulator
  - Cycle-level simulation tool built using SimpleScalar
  - Energy modeled using Wattch framework and CACTI-3

- Machine configurations
  - General-purpose
    - 4-way superscalar, 12-stage out-of-order pipeline
    - 32K inst cache and data cache, 1MB L2
  - Embedded
    - 2-way superscalar, 5-stage in-order pipeline
    - 16K inst cache and data cache, no L2

- Benchmarks
  - 12 SPEC2000 integer benchmarks
  - 16 Mediabench benchmarks
DISE vs Dedicated

DISE

Dedicated Hardware [Lefurgy+ 97]
DISE vs Dedicated

DISE
+ General-purpose

Dedicated Hardware [Lefurgy+ 97]
- Compression-specific
DISE vs Dedicated

DISE
- General-purpose
- Parameterization

Dedicated Hardware [Lefurgy+ 97]
- Compression-specific
- No parameterization
DISE vs Dedicated Hardware [Lefurgy+ 97]

DISE

- General-purpose
- Parameterization
- Branches

Dedicated Hardware

- Compression-specific
- No parameterization
- No branches
## DISE vs Dedicated

<table>
<thead>
<tr>
<th>DISE</th>
<th>Dedicated Hardware [Lefurgy+ 97]</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ General-purpose</td>
<td>- Compression-specific</td>
</tr>
<tr>
<td>+ Parameterization</td>
<td>- No parameterization</td>
</tr>
<tr>
<td>+ Branches</td>
<td>- No branches</td>
</tr>
<tr>
<td>+ Customizable dictionary</td>
<td>+ Customizable dictionary</td>
</tr>
</tbody>
</table>
DISE vs Dedicated Hardware [Lefurgy+ 97]

**DISE**
- General-purpose
- Parameterization
- Branches
- Customizable dictionary
- Larger codewords

**Dedicated Hardware**
- Compression-specific
- No parameterization
- No branches
- Customizable dictionary
- Smaller codewords
# DISE vs Dedicated

<table>
<thead>
<tr>
<th>DISE</th>
<th>Dedicated Hardware [Lefurgy+ 97]</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ General-purpose</td>
<td>- Compression-specific</td>
</tr>
<tr>
<td>+ Parameterization</td>
<td>- No parameterization</td>
</tr>
<tr>
<td>+ Branches</td>
<td>- No branches</td>
</tr>
<tr>
<td>+ Customizable dictionary</td>
<td>+ Customizable dictionary</td>
</tr>
<tr>
<td>- Larger codewords</td>
<td>+ Smaller codewords</td>
</tr>
<tr>
<td>- No length one sequences</td>
<td>+ Length one sequences</td>
</tr>
</tbody>
</table>
## DISE vs Dedicated

<table>
<thead>
<tr>
<th>DISE</th>
<th>Dedicated Hardware [Lefurgy+ 97]</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ General-purpose</td>
<td>- Compression-specific</td>
</tr>
<tr>
<td>+ Parameterization</td>
<td>- No parameterization</td>
</tr>
<tr>
<td>+ Branches</td>
<td>- No branches</td>
</tr>
<tr>
<td>+ Customizable dictionary</td>
<td>+ Customizable dictionary</td>
</tr>
<tr>
<td>- Larger codewords</td>
<td>+ Smaller codewords</td>
</tr>
<tr>
<td>- No length one sequences</td>
<td>+ Length one sequences</td>
</tr>
<tr>
<td>- Larger dictionary entries</td>
<td>+ Smaller dictionary entries</td>
</tr>
</tbody>
</table>
Code Size

Dedicated → DISE

Length 1 seqs, Codeword = 2 bytes, Dict entry = 4 bytes,
No params, No comp. branches
Code Size

Dedicated → DISE

No length 1 seqs, Codeword = 2 bytes, Dict entry = 4 bytes,
No params, No comp. branches
Code Size

Dedicated $\rightarrow$ DISE

No length 1 seqs,

**Codeword = 4 bytes,**

No params,

Dict entry = 4 bytes,

No comp. branches
Code Size

Dedicated → DISE

No length 1 seqs, Codeword = 4 bytes, Dict entry = 8 bytes,
No params, No comp. branches
Code Size

Dedicated → DISE

No length 1 seqs, 
Codeword = 4 bytes, 
Dict entry = 8 bytes, 
3 params, 
No comp. branches
Code Size

No length 1 seqs, Codeword = 4 bytes, Dict entry = 8 bytes,
3 params, Comp. branches
Code Size

DISE decompression does better than dedicated
DISE decompression makes more efficient use of dictionary
Parameterization

![Graph showing code size (norm. to uncompressed) for various benchmarks.

- SPEC benchmarks: crafty, eon, perlbmk, vortex
- MEDIABENCH benchmarks: g721.dec, ghostscr.gs, gsm.toast, mpeg2.enc

The graph compares the code size relative to the uncompressed size for each benchmark.
Parameterization

![Diagram showing code size (norm. to uncompressed) for various SPEC and MEDIABENCH benchmarks. The x-axis labels are crafty, eon, perlbmk, vortex, g721.dec, ghostscr.gs, gsm.toast, and mpeg2.enc. The y-axis represents code size normalized to uncompressed size. The legend indicates two categories: 0 and 1, possibly representing different parameterizations or variations.]
Parameterization
Parameterization

![Graph showing code size (norm. to uncompressed) for SPEC and MEDIABENCH benchmarks.

The x-axis represents benchmarks: crafty, eon, perlbmk, vortex, g721.dec, ghostscr.gs, gsm.toast, mpeg2.enc.

The y-axis represents code size normalized to uncompressed size, ranging from 0.5 to 1.0.

Different colors represent different parameterizations: 0, 1, 2, 3.

Below the graph, there is a legend for the color key.

Dictionary is indicated by a small box on the right side of the graph.

SPEC and MEDIABENCH benchmarks are listed on the x-axis.
Parameterization
Parameterization

SPEC MEDIA BENCH
crafty eon perlbmk vortex g721.dec ghostscr.gs gsm.toast mpeg2.enc
Parameterization

SPEC

MEDIABENCH

code size (norm. to uncompressed)

0 1 2 3 6 unbounded 6+6byteCW dictionary
Parameterization

- 3 params gives best compression w/o increasing codeword
Performance

IPC (norm. to baseline)

- crafty
- eon
- perlbmk
- vortex
- g721.dec
- ghostscr.gs
- gsm.toast
- mpeg2.enc

uncompressed
Performance

![Graph showing performance comparison for various benchmarks and memory configurations. The x-axis represents different benchmarks and memory configurations, while the y-axis shows IPC (IPC norm. to baseline). The graph compares uncompressed performance against baseline.]
## Performance

<table>
<thead>
<tr>
<th>SPEC</th>
<th>MEDIABENCH</th>
</tr>
</thead>
<tbody>
<tr>
<td>crafty</td>
<td>32K 16K 8K</td>
</tr>
<tr>
<td>eon</td>
<td>32K 16K 8K</td>
</tr>
<tr>
<td>perl-bmk</td>
<td>32K 16K 8K</td>
</tr>
<tr>
<td>vortex</td>
<td>32K 16K 8K</td>
</tr>
<tr>
<td>g721.dec</td>
<td>16K 8K 4K</td>
</tr>
<tr>
<td>ghostcr.gs</td>
<td>16K 8K 4K</td>
</tr>
<tr>
<td>gsm.toast</td>
<td>16K 8K 4K</td>
</tr>
<tr>
<td>mpeg2.enc</td>
<td>16K 8K 4K</td>
</tr>
</tbody>
</table>

**IPC (norm. to baseline)**

- Uncompressed

### Performance across benchmarks

The chart above illustrates the performance of various benchmarks across different memory configurations. The X-axis represents the benchmarks, and the Y-axis shows the IPC (instructions per cycle) normalized to the baseline. The bars indicate the performance of the benchmarks under uncompressed conditions.

**Legend:**
- Blue: uncompressed
Performance

The diagram shows the performance in terms of IPC (Instruction Per Cycle) normalized to a baseline for various benchmarks across different dictionary sizes. The benchmarks include SPEC (crafty, eon, perlbench, vortex, g721.dec, ghostscr.gs, gsm.toast, mpeg2.enc) and MEDIABENCH (128, 768B). The dictionary sizes tested are 32K, 16K, 8K, and 4K for uncompressed and dictionary sizes 128 (768B) for compressed cases.
Performance

- Performance improves even with small dictionary
Performance

- Performance improves even with small dictionary
Performance improves even with small dictionary
DISE decompression can compensate for small I$
Performance

- Performance improves even with small dictionary
- DISE decompression can compensate for small I$
Performance

- Performance improves even with small dictionary
- DISE decompression can compensate for small I$
- Huge dictionaries are not necessary
Reducing Energy

- Performance perspective
  - Compression compensates for small I$
    (32KB I$ == 16KB I$ + 3KB RT)
Reducing Energy

- Performance perspective
  - Compression compensates for small I$
    (32KB I$ == 16KB I$ + 3KB RT)

- Energy perspective
  - Move accesses from big structure (I$) to small one (RT)
  - Use dynamic compression algorithm
    - Compress dynamically frequent sequences
Energy

Fixed I$, increasing RT size

![Energy Consumption Graph]

- **Energy (norm. to baseline)**
- **DISE energy**
- **I$ energy**
- **other energy**

### SPEC
- **crafty**
- **eon**
- **perlbmk**
- **vortex**
- **g721.dec**
- **ghostscr.gs**
- **gsm.toast**
- **mpeg2.enc**

### MEDIABENCH
- **none**
- **128**
- **512**
- **2K**

- **other energy**
- **I$ energy**
- **DISE energy**
Energy

Fixed I$, increasing RT size

DISE decomp. dramatically reduces energy consumption
Energy

Fixed I\$, increasing RT size

- DISE decompression dramatically reduces energy consumption
Energy

Fixed I$, increasing RT size

- DISE decompression dramatically reduces energy consumption
Energy

DISE decompression dramatically reduces energy consumption
DISE decompression dramatically reduces energy-delay product
Summary

- Decreased code size by as much as 35%
  - Evaluated impact of parameterization
  - Accounted for 10-20% compression

- Achieved performance improvements of 5-20%

- Reduced energy consumption by 10% and edp by 20%

- Leveraged general-purpose facility (DISE)