On the Interplay of Loop Caching, Code Compression, and Cache Configuration

Marisha Rawlins and Ann Gordon-Ross+

University of Florida
Department of Electrical and Computer Engineering

+ Also Affiliated with NSF Center for High-Performance Reconfigurable Computing

CHREC
NSF Center for High-Performance Reconfigurable Computing

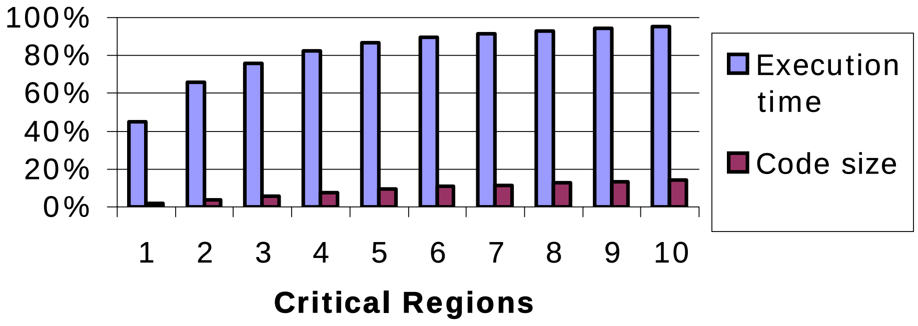
Gator Engineering
University of Florida

This work was supported by National Science Foundation (NSF) grant CNS-0953447

Instruction Cache Optimization

- The instruction cache is a good candidate for optimization
  - Large source of energy consumption
  - Predictable spatial and temporal locality
- Several optimizations exploit the 90-10 rule
  - 90% of execution is spent in 10% of code known as critical regions
  - Optimizations include loop caching, cache tuning, and code compression

Gordon-Ross '04



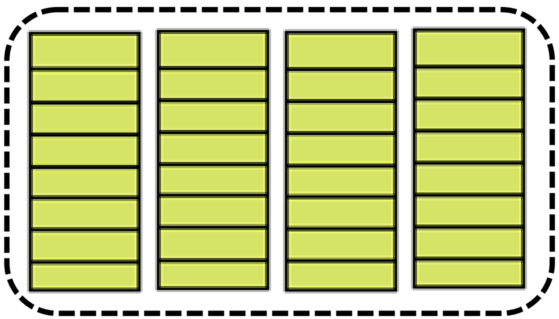
Instruction Cache Optimizations

Many instruction cache optimizations exist

code reordering

code compression

L1 Instruction Cache



loop caching

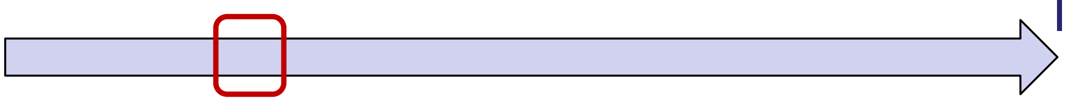
cache tuning

trace caching

filter caching

cache partitioning

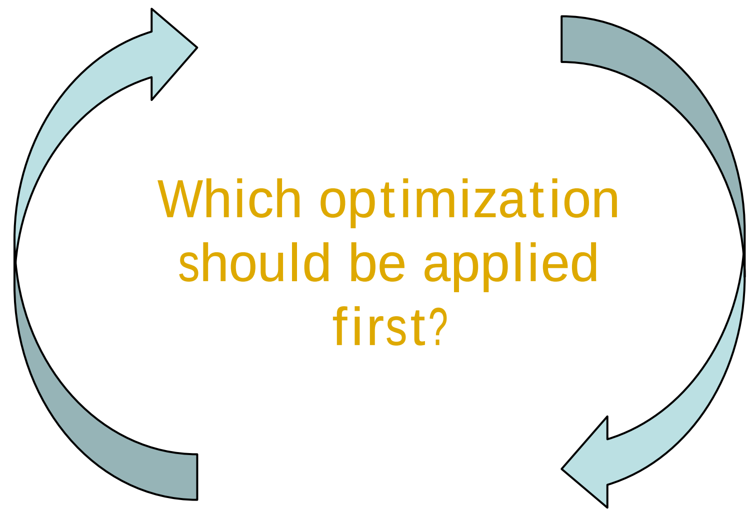
Which technique should we apply?



Increased Savings

apply optimization 1 ~~apply optimization 2~~

Can optimizations be combined?



How do optimizations interact?

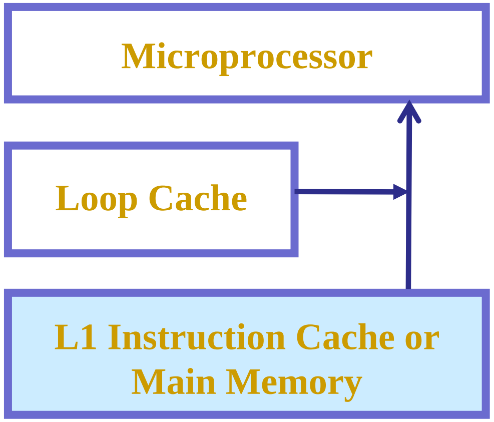
Complement

Degrade

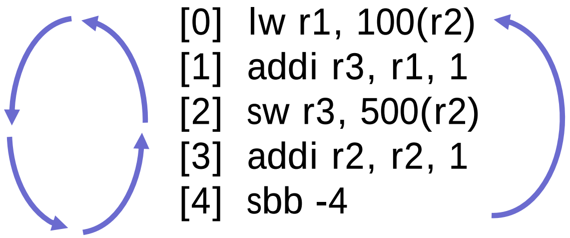
Obviate

Instruction Cache Optimization – Loop Caching

- The loop cache achieves energy savings by storing loops in a smaller device than the L1 cache
- Operation
  - Filled when a short backward branch is detected in the instruction stream
  - Provides the processor with instructions on the next loop iteration



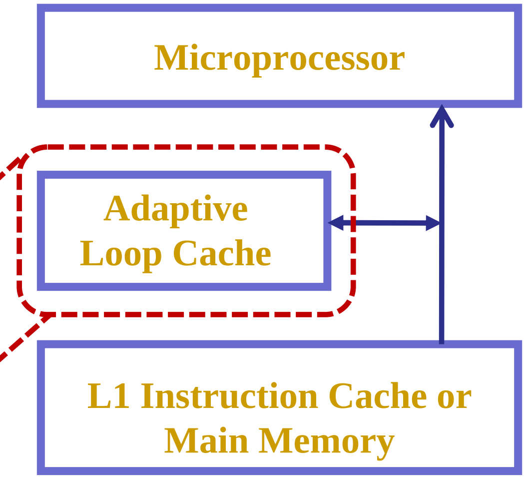
- Benefits
  - Smaller, tagless device → energy savings
  - Miss-less device → no performance penalty
    - Loop cache operation must guarantee a 100% hit rate
  - Loop cache operation invisible to user



Cannot cache loops with taken branches

Adaptive Loop Cache (ALC)

- Dynamically caches loops containing branches (Rawlins/Gordon-Ross 10)
  - Filled when a short backward branch is detected in the instruction stream
  - Valid bits are used to indicate the location of the next instruction fetch and are critical for maintaining a 100% hit rate



loop / branch

```
lw r1, 200(r2)
addi r3, r1, 1
sw r3, 500(r2)
bne r4, r3, 3
srl r4, r5, 10
or r6, r4, r1
addi r2, r2, 1
sbb -7
```

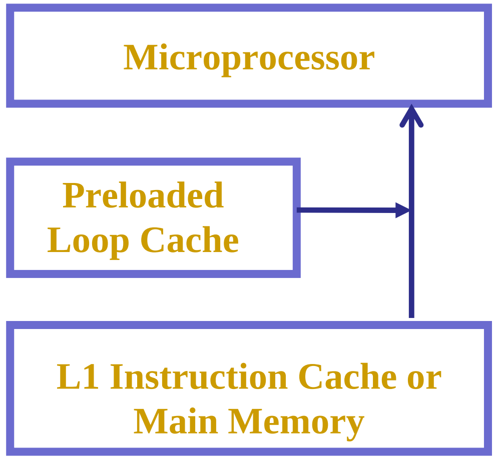
Instructions	nv	tnv
lw r1, 200(r2)	1	0
addi r3, r1, 1	1	0
sw r3, 500(r2)	1	0
bne r4, r3, 3	1	1
srl r4, r5, 10	1	0
or r6, r4, r1	1	0
addi r2, r2, 1	1	0
sbb -7	0	1

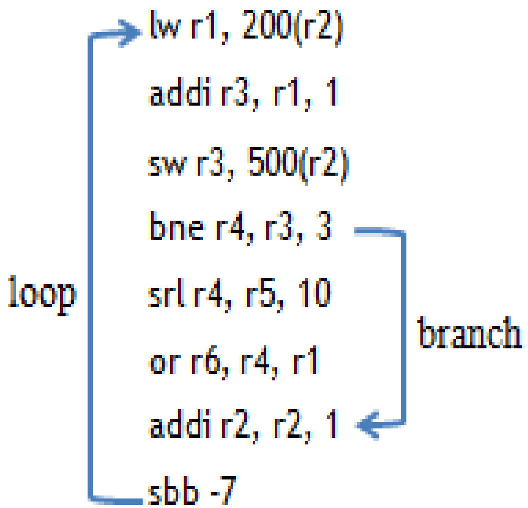
Energy savings as high as 69%

No designer effort

Preloaded Loop Cache (PLC)

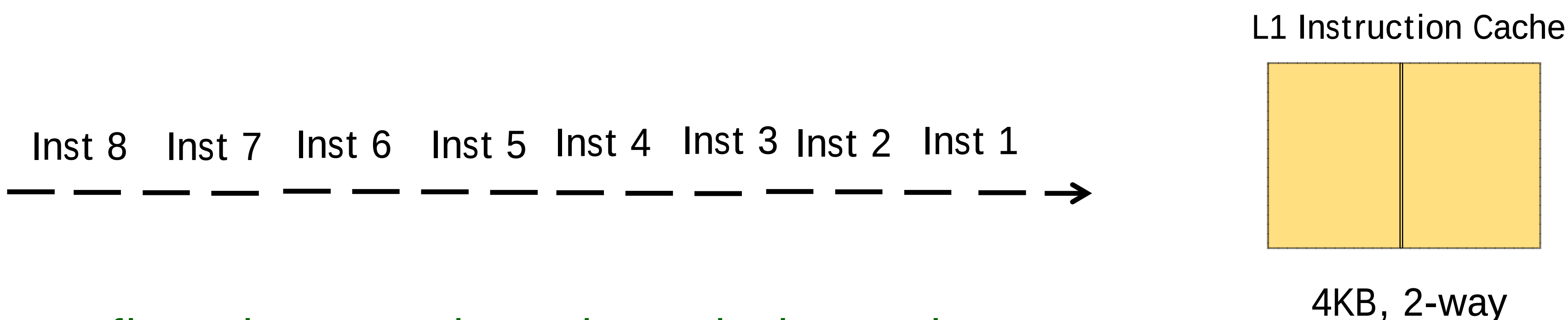
- Statically stores the most frequently executed loops (Gordon-Ross/Cotterell/Vahid 02)
  - Can cache loops containing branches and subroutines
- Operation
  - Application is profiled offline and critical regions are stored in the loop cache
  - PLC provides the instructions when a stored critical region is executed
  - Exit bits are used to indicate the location of the next instruction fetch
- No runtime fill cycles
- But requires designer effort and not appropriate for dynamic applications





Instruction Cache Optimization – Cache Configuration (Tuning)

- Different applications have vastly different cache requirements
  - Cache parameters that do not match an application's behavior can waste over 60% of energy (Gordon-Ross 05)
  - Cache tuning determines appropriate cache parameters (***cache configuration***) to meet optimization goals (e.g., lowest energy)
    - Configure cache parameters: size, line size, associativity



Cache configuration tunes the cache to the instruction stream

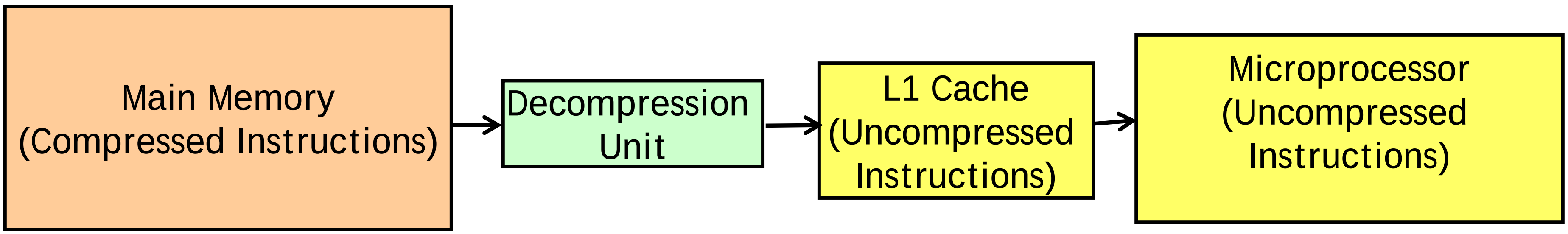
Average Energy Savings from Cache Tuning > 40%

Instruction Cache Optimization – Code Compression

- Code compression techniques were initially developed to reduce the static code size in embedded systems
- Code compression is typically performed off-line while decompression is performed during run-time
  - Area savings in main memory and perhaps the level one cache, depending on decompression location
- Since decompression done during runtime, decompression overhead must be minimized
  - Decompression overhead is defined as energy and performance expended while decompressing instructions

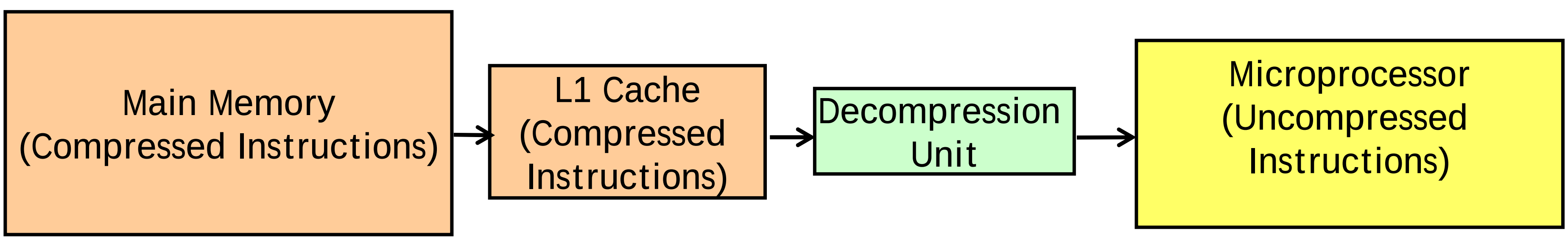
Code Compression Architectures

- Decompression on Cache Refill (DCR)



<u>Less overhead, no L1 area savings</u>

- Decompression on Fetch (DF)

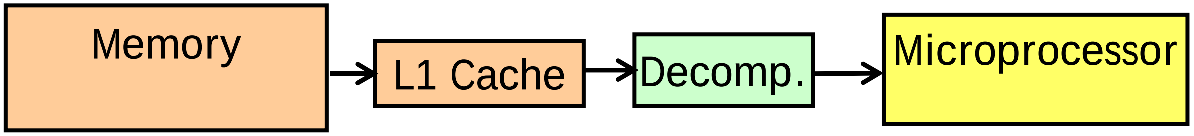


<u>More overhead, but L1 area savings</u>

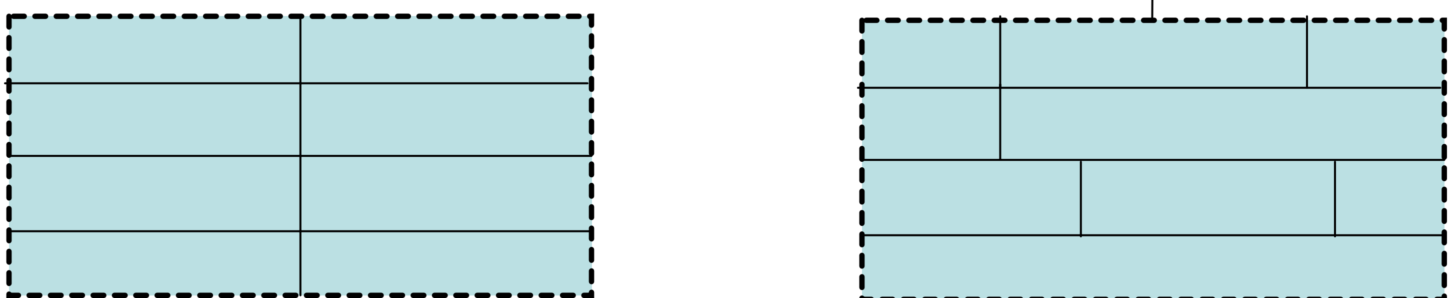
Code Compression (Energy Savings)

- Previous work on code compression achieved energy savings as high as 82% (Benini et al. 2001; Lekatsas 2000)

DF



- Decompression on Fetch (DF) architecture consumed lower energy than the Decompression on Cache Refill (DCR) architecture
  - Bit toggling and energy expended on busses were reduced
  - The L1 cache capacity was effectively increased



8 instructions 12 instructions

decompression unit is on the critical path → need low decompression overhead

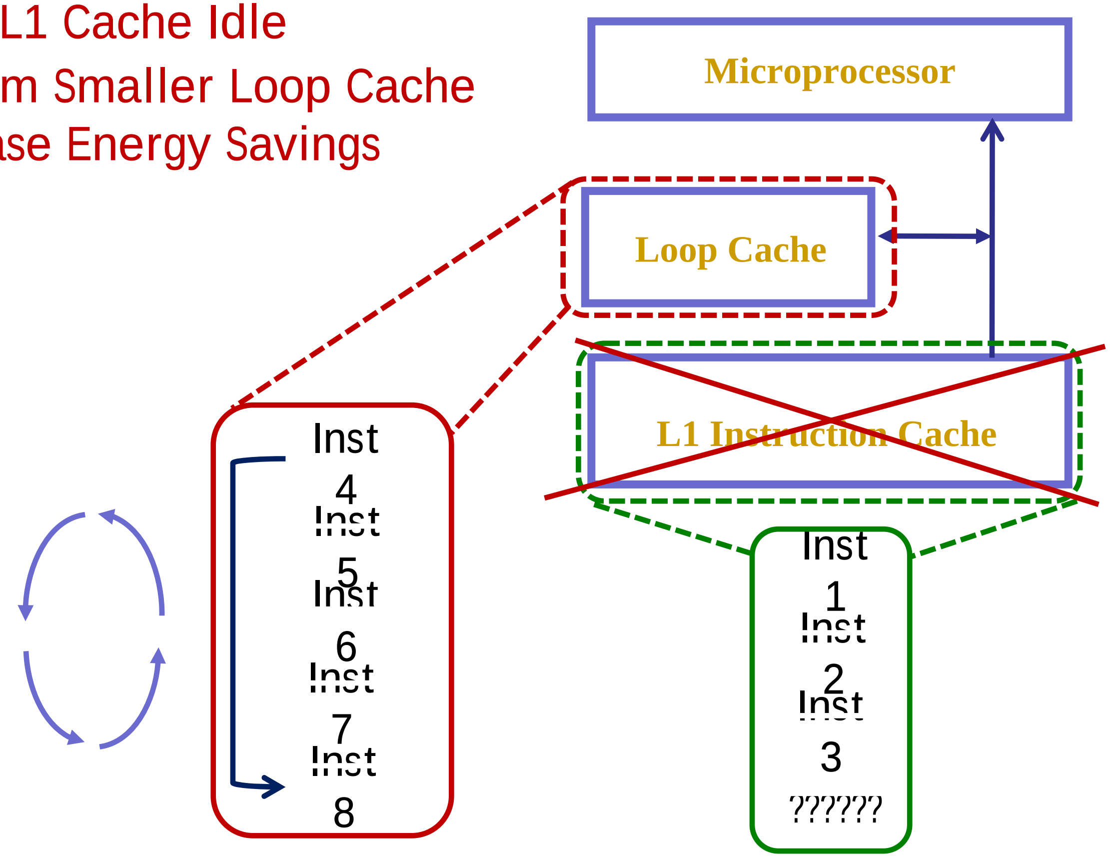
Combining Optimizations

- Studying the interaction of existing techniques reveals the practicality of combining optimization techniques
  - Combining certain techniques provides additional energy savings but the combination process may be non-trivial (e.g. circular dependencies for highly dependent techniques)
  - In these cases, new design techniques must be developed to maximize savings
  - Less dependent techniques may be easier to combine but may reveal little additional savings
  - Some combined techniques may even degrade each other

Combining Cache Tuning and Loop Caching

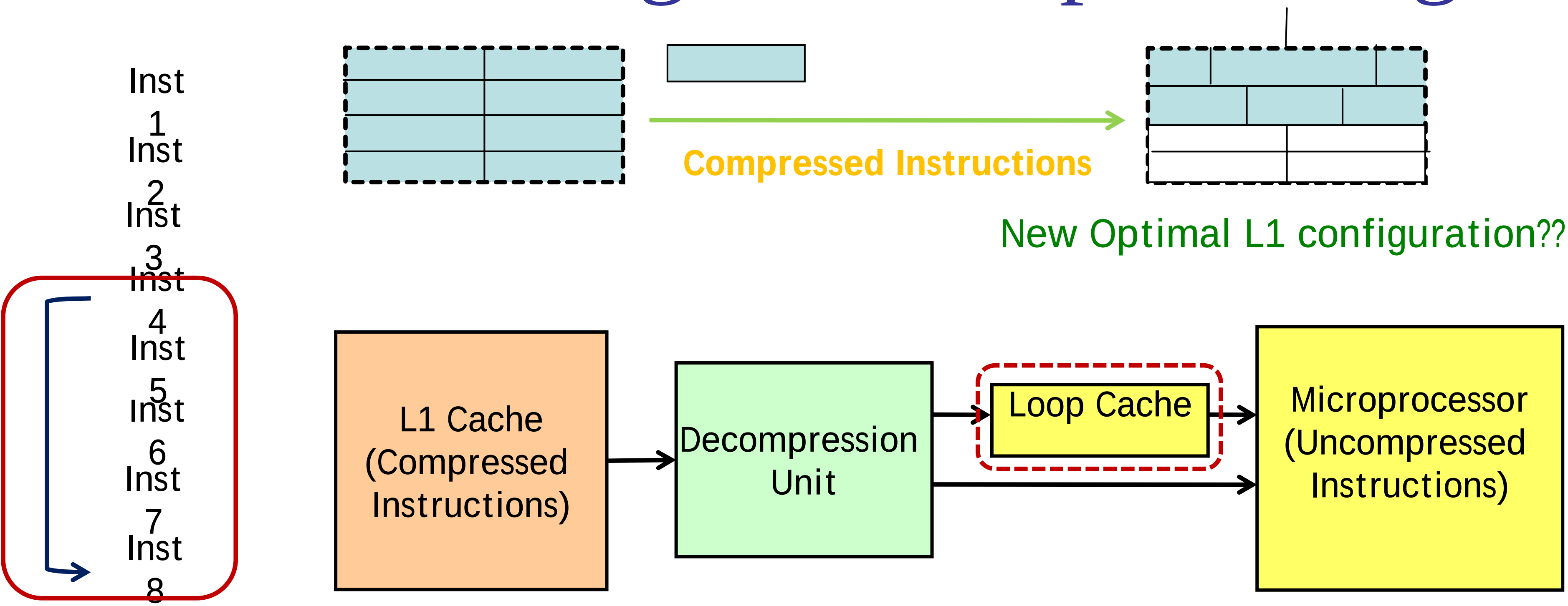
Inst 1
Inst 2
Inst 3
Inst 4
Inst 5
Inst 6
Inst 7
Inst 8

L1 Cache Idle
Fetch from Smaller Loop Cache
Increase Energy Savings



New Optimal L1 configuration??

Combining Code Compression, Cache Tuning, and Loop Caching



Loop Cache stores Uncompressed loop instructions

Decompression overhead eliminated when loops are fetched from the loop cache

Reduces overall energy consumption

Contribution

- Combining optimization techniques with respect to additional energy savings, desired designer effort, and dynamic flexibility
  - Adaptive Loop Cache (ALC) – No designer effort, most flexible
  - Preloaded Loop Cache (PLC) – Designer effort, less flexible but greater savings (no fill cycles)
- Interaction of cache tuning, loop caching, and code compression
  - Additional energy savings from combining loop caching and cache tuning
  - Identify benchmark characteristics and situations where combining loop caching and cache tuning are most effective
  - Investigate the practicality of using a loop cache to reduce decompression overhead
  - Indentify side effects from combining loop caching, code compression, and cache tuning

Loop Cache and Level One Cache Tuning

Experimental Setup

- Modified SimpleScalar[1] to implement the Adaptive Loop Cache (ALC) and Preloaded Loop Cache (PLC)
- 31 benchmarks from the EEMBC[2], Powerstone[3], and MiBench[4] suites
- Energy model based on access and miss statistics, (SimpleScalar) and energy values (CACTI[5])
- Energy savings calculated with respect to our base system (an 8kB, 4-way associative, 32 byte line size L1 cache[6] with no loop cache)

[1] (Burger/Austin/Bennet 96), [2](http://www.eembc.org/), [3](Lee/Arends/Moyer 98), [4](Guthaus/Ringenberg/Ernst/Austin/Mudge/Brown 01), [5](Shivakumar/Jouppi 01) [6](Zhang/Vahid/Najjar 00)

Experimental Setup

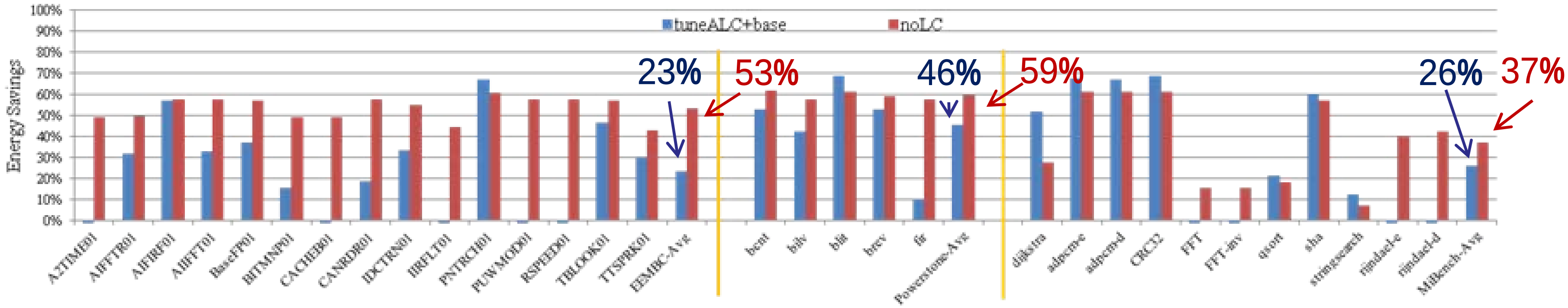
- Tunable cache parameters (based on [6, 7])
  - L1 cache size: 2kB,4kB, and 8kB
  - L1 cache line size: 16 bytes, 32 bytes, and 64 bytes
  - L1 cache associativity: 1-, 2-, and 4-way associative
  - Loop cache sizes: 4 – 256 entries
- Experiments
  - Tuned the L1 cache with a fixed size ALC
  - Tuned both the L1 cache the ALC
  - Tuned the L1 cache with fixed size PLC
- For comparison purposes we reported
  - Tuned ALC with a fixed L1 base cache
  - Tuned the L1 cache in a system with no loop cache

ALC - Adaptive Loop Cache
PLC - Preloaded Loop Cache

[6](Zhang/Vahid/Najjar 00), [7](Rawlins/Gordon-Ross 10)

Energy Savings

Cache Tuning & Loop Caching Applied Individually

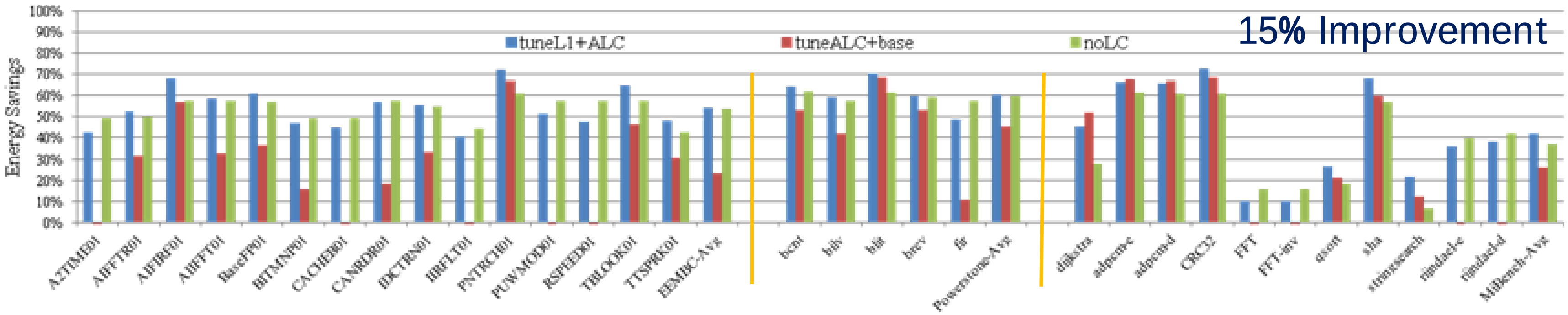


- In general, loop caching alone does not match cache tuning alone

ALC - Adaptive Loop Cache

Energy Savings

Combining a Fixed Sized ALC with L1 Cache Tuning

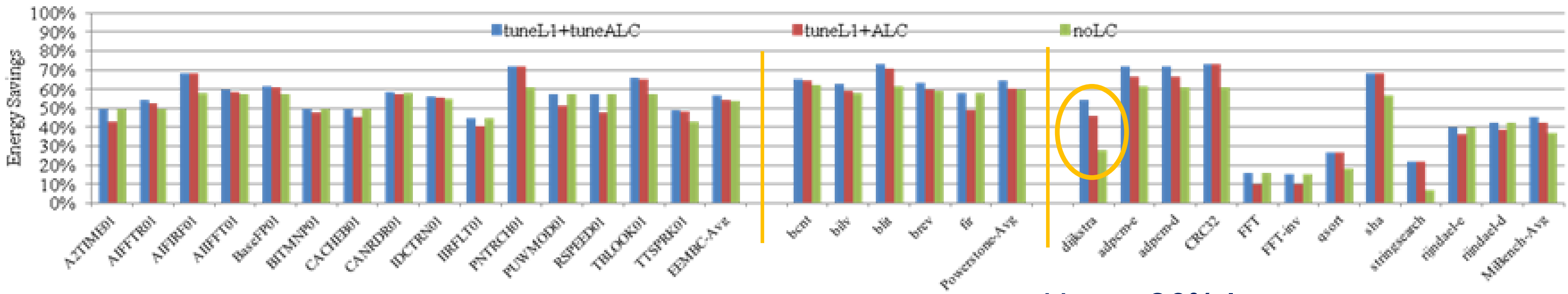


- Small average improvement in energy savings compared to cache tuning alone
- L1 cache tuning dominates overall energy savings

ALC - Adaptive Loop Cache

Energy Savings

Combining ALC Tuning with L1 Cache Tuning



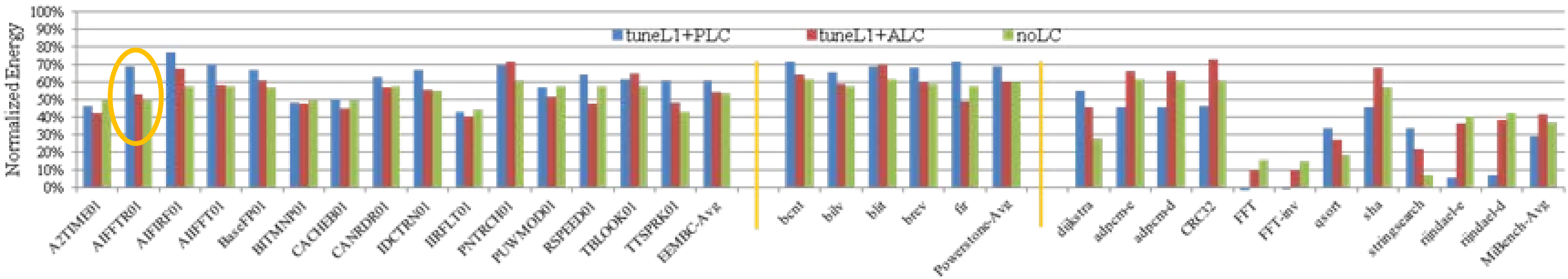
Up to 26% Improvement

- Small improvement in savings
- L1 cache tuning therefore obviates the need for ALC tuning
  - Adding an appropriately sized ALC is sufficient
  - Reduces design space exploration
    - No need to try each ALC configuration with each L1 cache configuration

ALC - Adaptive Loop Cache

Energy Savings

Combining a Fixed Sized PLC with L1 Cache Tuning



Up to 27% Improvement

10% Average Improvement

- PLC results in higher energy savings compared with the ALC
- Using a PLC can result in a different optimal L1 configuration
  - PLC removes instructions from instruction stream
  - Achieves area savings up to 33% for 14 benchmarks

ALC - Adaptive Loop Cache
PLC - Preloaded Loop Cache

Code Compression, Loop Caching, and Cache Tuning

Experimental Setup

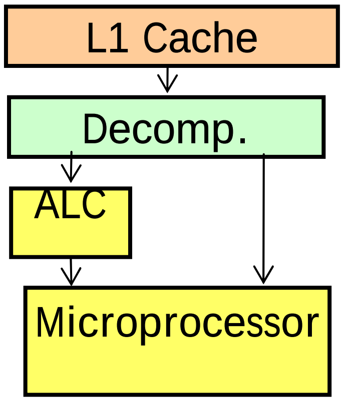
- Decompression on Fetch architecture with Huffman encoding
- 32 entry ALC; 64 entry PLC (based on [7])
- Decompression unit, Line Address Table, ALC, and PLC implemented in SimpleScalar
- Branch targets were byte aligned for random access
- Energy model modified for decompression energy
- Measured performance (# cycles needed to complete execution)

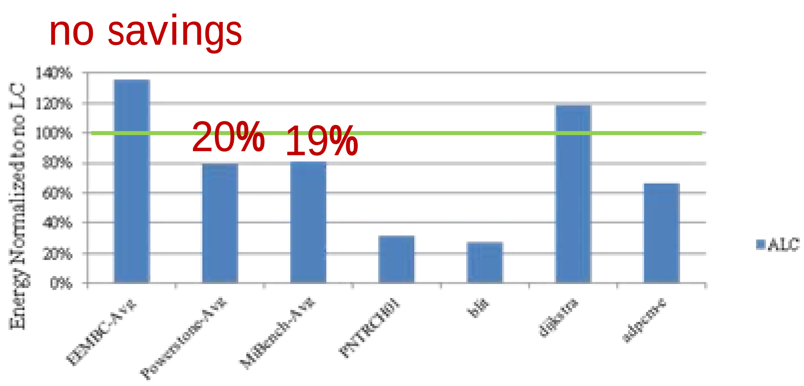
ALC - Adaptive Loop Cache
PLC - Preloaded Loop Cache

[7](Rawlins/Gordon-Ross 10)

Energy Savings (ALC)

Combining Code Compression with L1 Cache Tuning





Using the ALC to store uncompressed instructions

ALC - Adaptive Loop Cache

- Powerstone and MiBench benchmarks contain few loops which iterate several times
- EEMBC benchmarks contain several loops which iterate fewer times than Powerstone/MiBench
  - EEMBC benchmarks spend little time fetching uncompressed instructions from the ALC before the decompression unit is invoked again

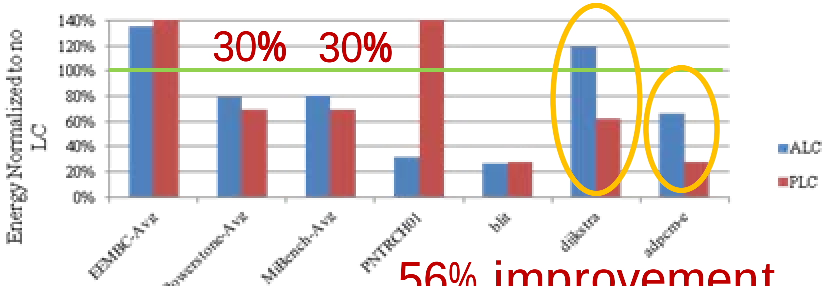
Energy Savings (PLC)

Combining Code Compression with L1 Cache Tuning

no savings

Additional 10% energy savings

30% 30%



Using the PLC to store uncompressed instructions

ALC - Adaptive Loop Cache
PLC - Preloaded Loop Cache

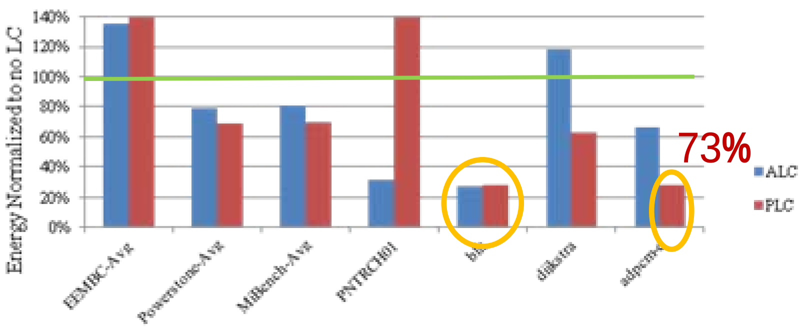
56% improvement

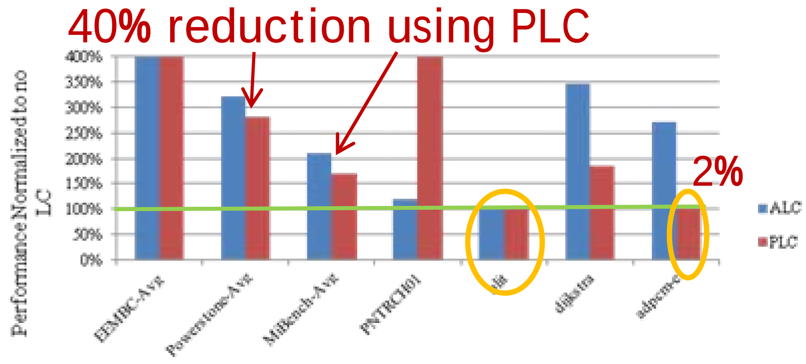
38% improvement

– Eliminates the decompression overhead (energy) which would have been consumed while filling the ALC

Performance (ALC & PLC)

Combining Code Compression with L1 Cache Tuning





Average increase in execution time (decompression overhead):
1.7x – 4.7x

- PLC smaller performance penalty than ALC
- Combining code compression and L1 cache tuning is possible when loop caching eliminates decompression overhead
- In some cases, combining code compression and L1 cache tuning is only possible using the PLC

ALC - Adaptive Loop Cache
PLC - Preloaded Loop Cache

Area (ALC & PLC)

Combining Code Compression with L1 Cache Tuning

- Storing compressed instructions in the L1 cache resulted in smaller optimal L1 configurations for 12 benchmarks

Original Optimal L1 Cache Size	New Optimal L1 Cache Size	Area Savings
8KB	2KB	50%
8KB	4KB	30%
4KB	2KB	20%

- For the remaining benchmarks the L1 cache configuration did not change
  - Thus adding a loop cache increased the area of the system
- Some benchmarks achieved energy savings but not area savings

ALC - Adaptive Loop Cache
PLC - Preloaded Loop Cache

Conclusions

- We investigated the effects of combining loop caching with level one cache tuning
  - In general, cache tuning dominates overall energy savings indicating that cache tuning is sufficient for energy savings
  - However, we observed that adding a loop cache to an optimal (lowest energy) level one cache can increase energy savings by as much as 26%
- We investigated the possibility of using a loop cache to minimize run-time decompression overhead and quantified the effects of combining code compression with cache tuning
  - Our results showed that a loop cache effectively reduces the decompression overhead resulting in energy savings of up to 73%
  - However, to fully exploit combining cache tuning, code compression, and loop caching, a compression/decompression algorithm with a lower overhead than the Huffman encoding technique is required