BLISS

Simultaneously Improving Code Size, Performance, & Energy in Embedded Processors

Ahmad Zmily and **Christos Kozyrakis**

Electrical Engineering Department
Stanford University
{zmily,kozyraki}@stanford.edu

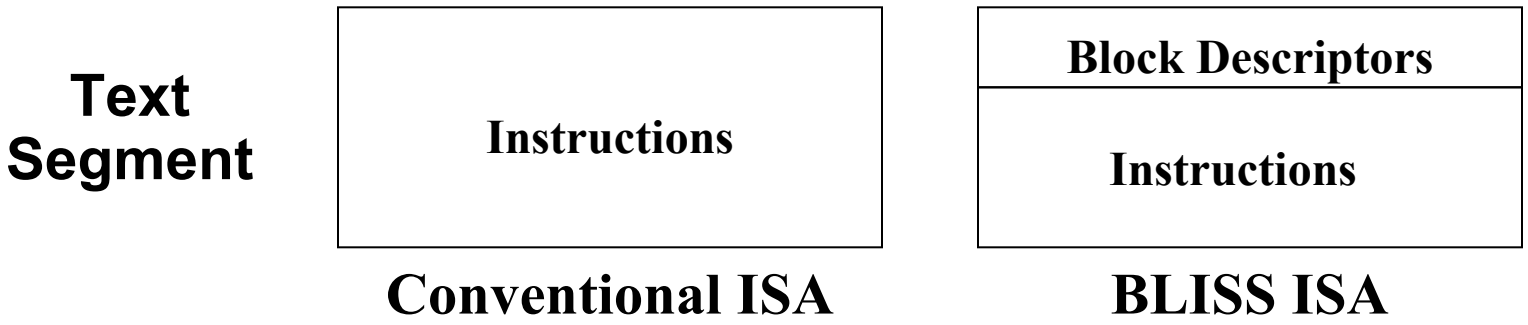
Efficiency Metrics for Embedded CPUs

- High performance
  - To meet soft or hard real-time constraints
  - Increasing demands from emerging applications
- Low energy consumption
  - Determines battery life-time
  - Critical for portable and deeply-embedded systems
- Low code size
  - Determines cost of program storage devices
    - Flash, ROM, instruction RAM/caches
  - A major cost component for the overall system

Code Size Trade-offs

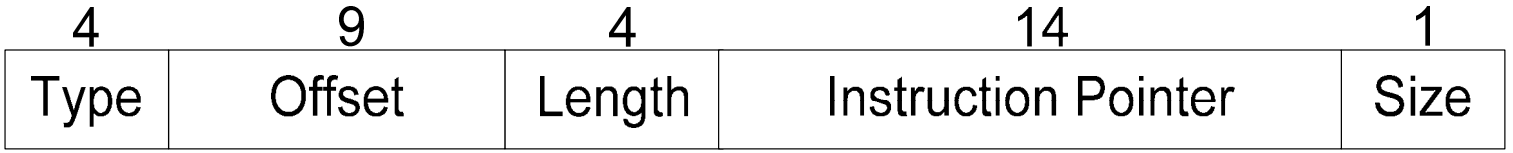
- 16-bit instructions
  - Better code size, worse performance & energy than 32-bit code
- Selective use of 16-bit instructions
  - Better code size, same performance & energy as 32-bit code
- This talk: can we improve all three simultaneously?

BLISS Instruction Set



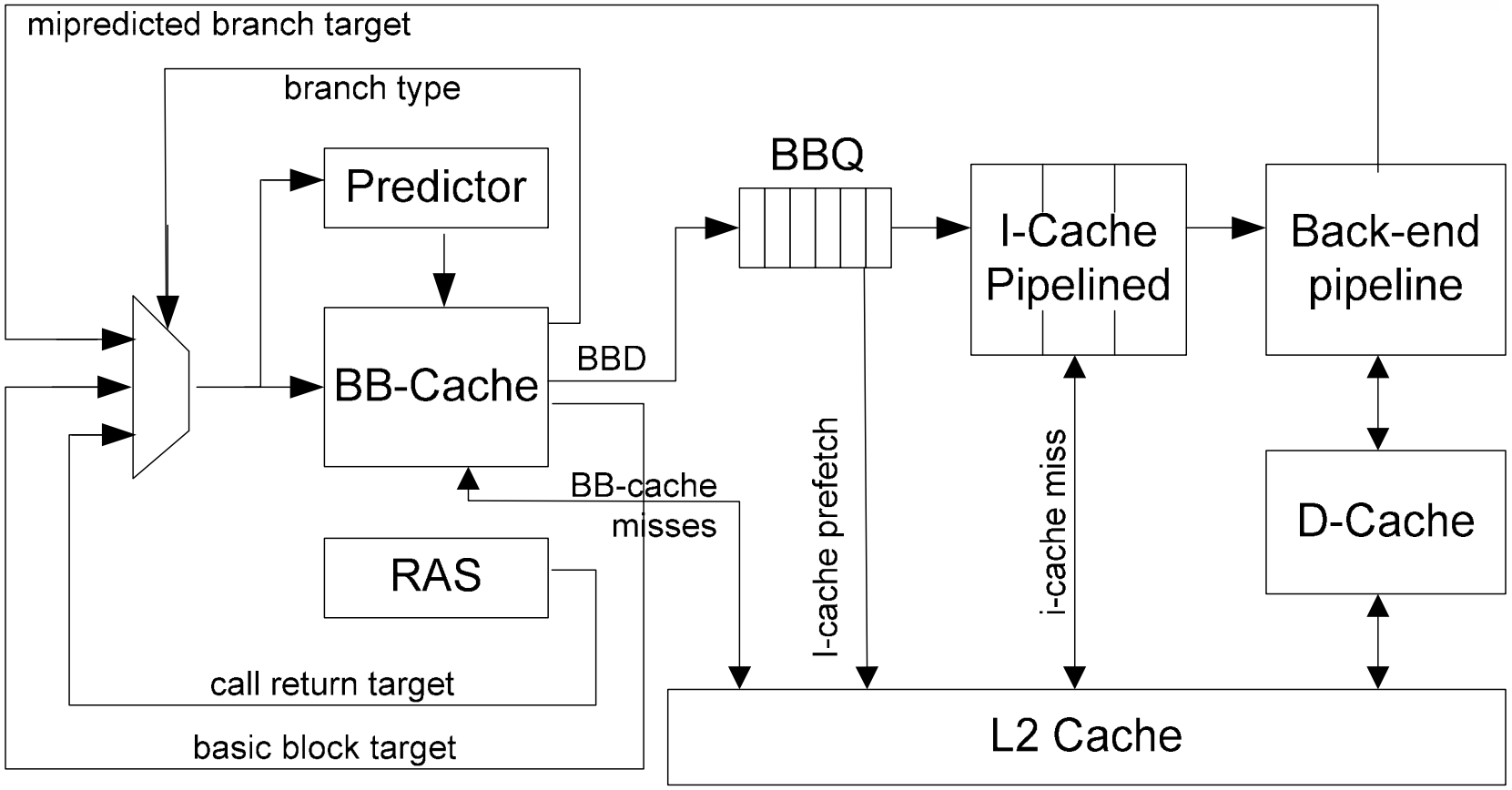
- BLISS = Block-aware Instruction Set
- Explicit basic block descriptors (BBDs)
  - Stored separately from instructions in the text segment
  - Describe control flow and identify associated instructions
- Execution model
  - PC always points to a BBD, not to instructions
  - Atomic execution of basic blocks

32-bit Basic Block Descriptor Format



- **Type**: type of terminating control-flow instruction
  - Fall-through, jump, jump register, branch, call, return
- **Offset**: displacement for PC-relative branches and jumps
  - Offset to target basic block descriptor
- **Length**: number of instruction in the basic block
  - 0 to 15 instructions
- **Instruction pointer**: address of the first instruction in the block
  - Remaining bits from TLB
- **Size**: indicate the encoding size of instructions in the block
  - 16-bit encoding or 32-bit encoding

BLISS Decoupled Front-End



- BTB replaced by cache for block descriptors
- Decoupled descriptors fetch from instruction fetch

BLISS Performance & Energy

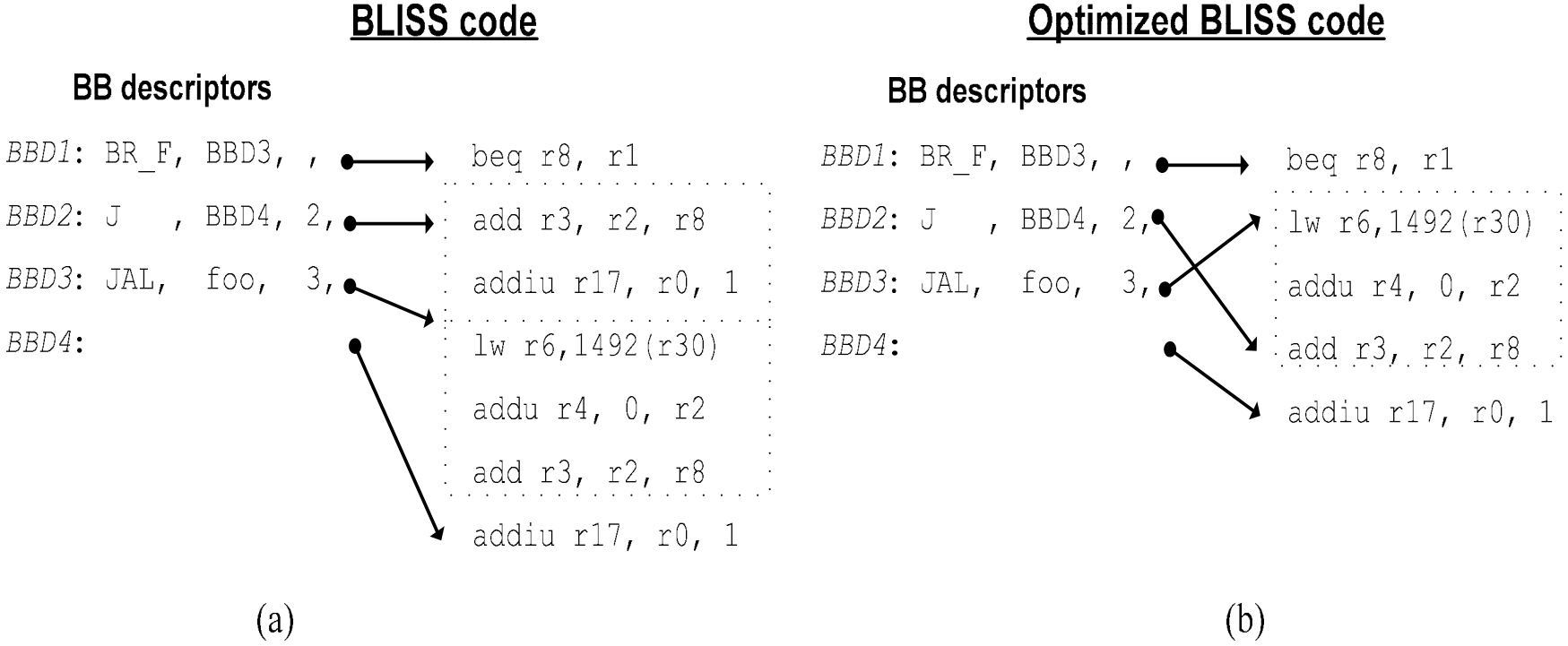
- Higher performance
  - Better branch prediction using software-defined info
    - L2 stores descriptors, better use of direction predictors, tolerate I-cache latency, …
  - I-cache prefetching using info in block descriptors
- Lower energy consumption
  - Energy saved by reducing mispredicted instructions
  - Judicious access to I-cache using software-define info
    - Merge accesses to sequential blocks, serial tag/data access, read needed words only, …
- See [ISLPED'05][EUROPAR'05] for details

BLISS Code Size

- Naïve code generation
  - 10-20% code size increase compared to 32-bit RISC!
  - A block descriptor per 5 to 10 instructions
- Basic code size optimizations
  - All jump instruction are removed
    - BBD defines both control-flow type and the offset
  - Many conditional branches can be removed
    - Simple condition test encoded in the producing opcode
    - Branch target is provided by the block descriptor

Block Subsetting Optimization

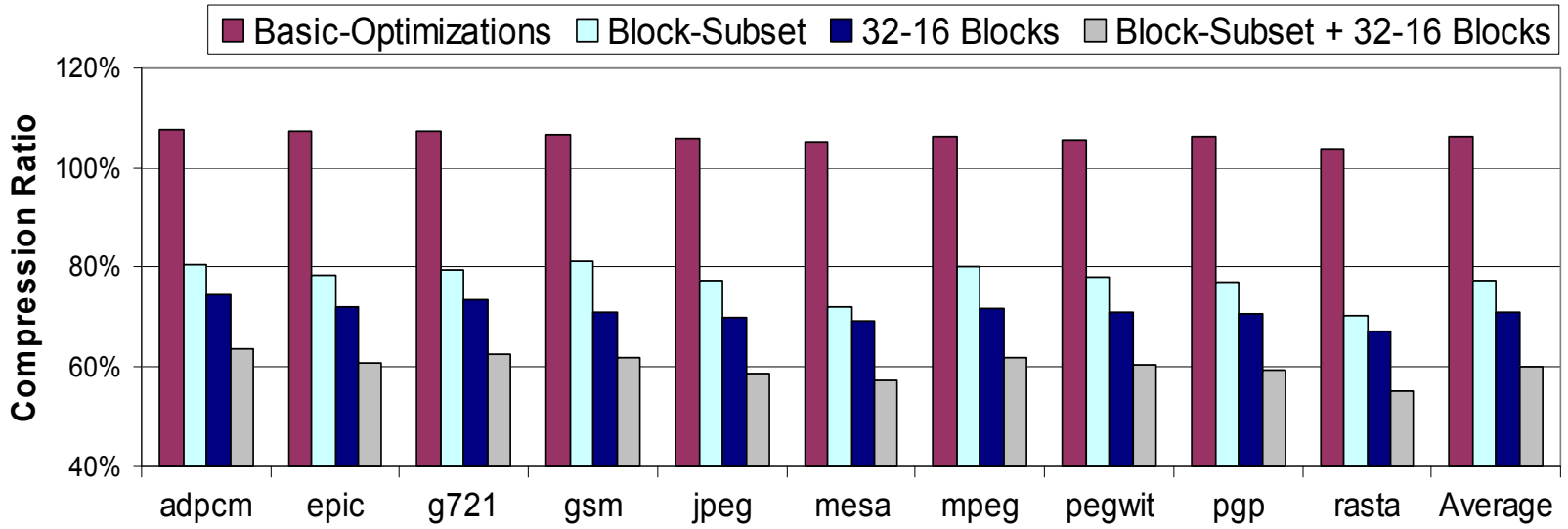
- Idea: duplicate descriptors but never instructions
  - Eliminate all instruction in a block if exact sequence found elsewhere in the binary
  - Adjust instruction pointer in block descriptor



Efficient 16-bit/32-bit Code Interleaving

- MIPS16: interleaving at function-level
  - JALX instruction is used to switch between functions
  - But functions include both perf. critical & non-critical code
- Thumb-2, rISA: instruction-level interleaving
  - A couple of instructions per switch
  - Can switch encoding at arbitrary points
- BLISS: basic block interleaving
  - A block is either fully perf. critical or fully non-critical
  - Descriptor indicates the encoding size for each block
  - No other overhead/instructions for switch

Evaluation Summary



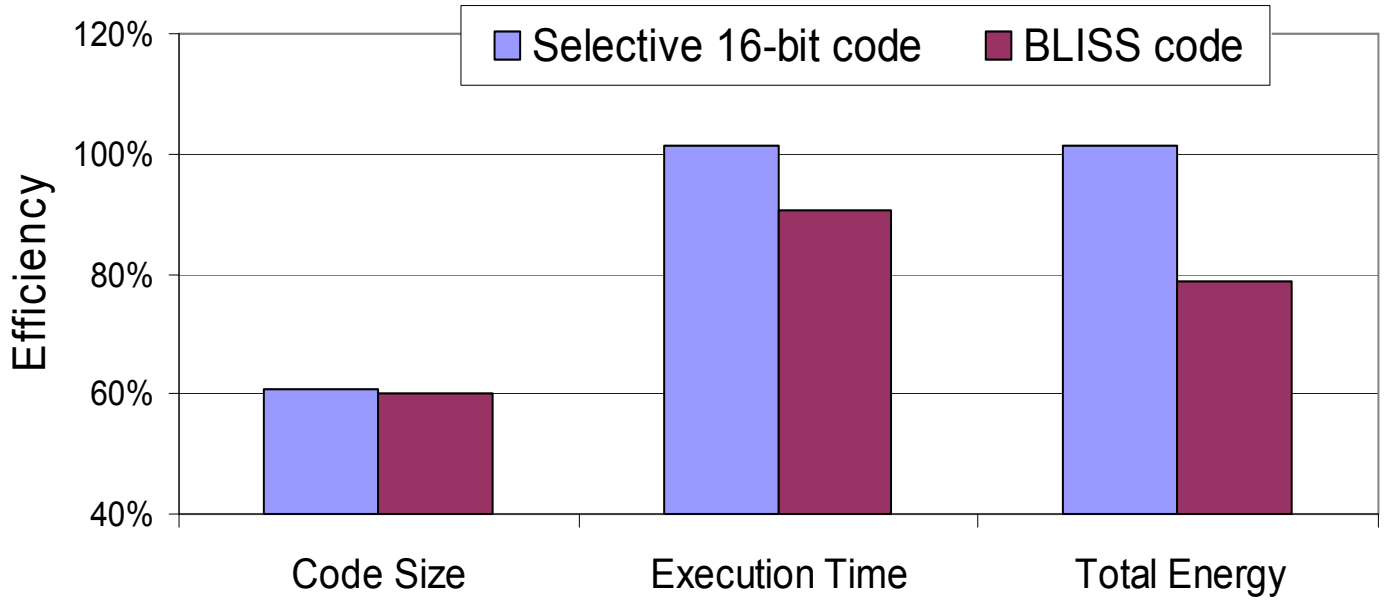
- Up to 60% average compression ratio
  - For mediabench applications
- Performance & energy for Xscale PXA270
  - +10% performance, -20% total energy, see paper for details
  - Similar results for high-end embedded CPUs
    - IBM PowerPC 750GX

Evaluation

Code Size Statistics

Benchmark	MIPS32	BLISS basic Optimization		Block Subset	Selective 16/32 Blocks	
	Code Size (kb)	J/B inst. Removed	# BB	Inst. eliminated	% inst Converted to 16-bit	Number of extra inst Added
adpcm	36	1671	2607	2536	94%	730
epic	64	2808	4345	4771	96%	823
g721	42	1920	2942	3015	93%	750
gsm	69	2866	4409	4483	94%	948
jpeg	109	4535	6609	8033	96%	1322
mesa	**430**	**16692**	**24054**	**36628**	**95%**	**6305**
mpeg2.dec	77	3514	5128	5168	96%	1242
mpeg2.enc	104	4390	6502	6895	95%	1820
pegwit	74	2735	4094	5229	95%	1666
rasta	226	10628	13788	19191	96%	1040

BLISS Vs. Selective 16-bit code



- BLISS achieves similar code size reduction with
  - 10% performance improvement
  - 21% total energy improvement

Conclusions

- BLISS: a block-aware instruction set
  - Defines basic block descriptors separate from instructions
  - Expressive ISA to communicate software info and hints
- Enabled code optimizations
  - Eliminate redundant jump and branch instructions
  - Remove blocks which appear elsewhere in the code
  - Interleaving 16-bit & 32-bit code at basic-block level without overhead
- Improved code size **and** energy consumption **and** performance:
  - 40% code size reduction
  - 10% performance improvement
  - 21% reduction in total energy consumption