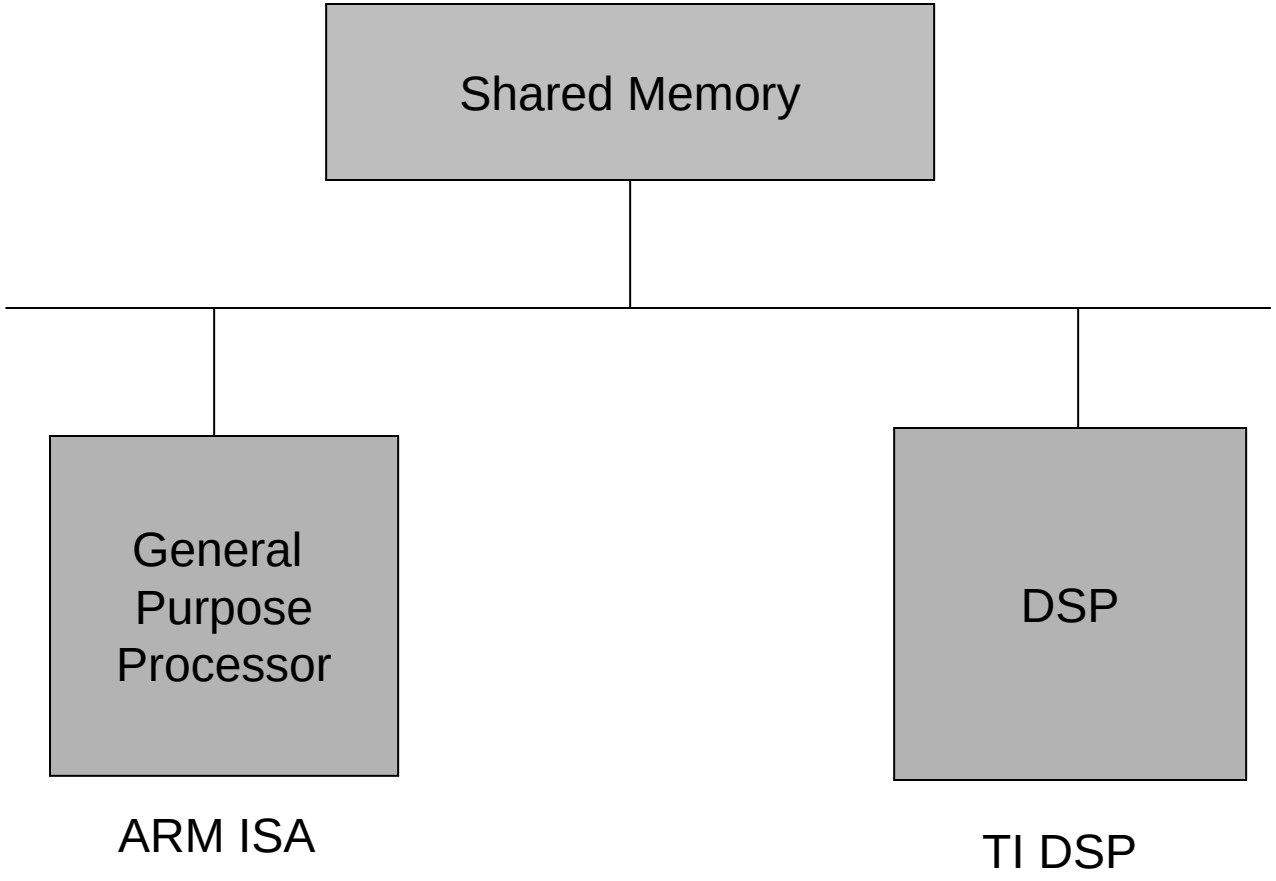
Is VLIW the Right Architecture for Embedded Processing?

Trevor Mudge
Bredt Professor of Engineering
Advanced Computer Architecture Lab
The University of Michigan of Engineering

With

- Prof. Wayne Wolf, Princeton Univ.
- Tiehan Lv, Princeton Univ.
- David Oehmke, Univ. Michigan
- Jeff Ringenberg, Univ. Michigan

What Kinds of Embedded Systems?



Trends

- General purpose computers
  - Superscalar
    - Deeper pipelines
    - Out-of-order execution
    - Multi-issue
- Digital signal processors
  - VLIW
    - Simpler hardware/function unit
    - Require more compiler support
      - Program in micro-code

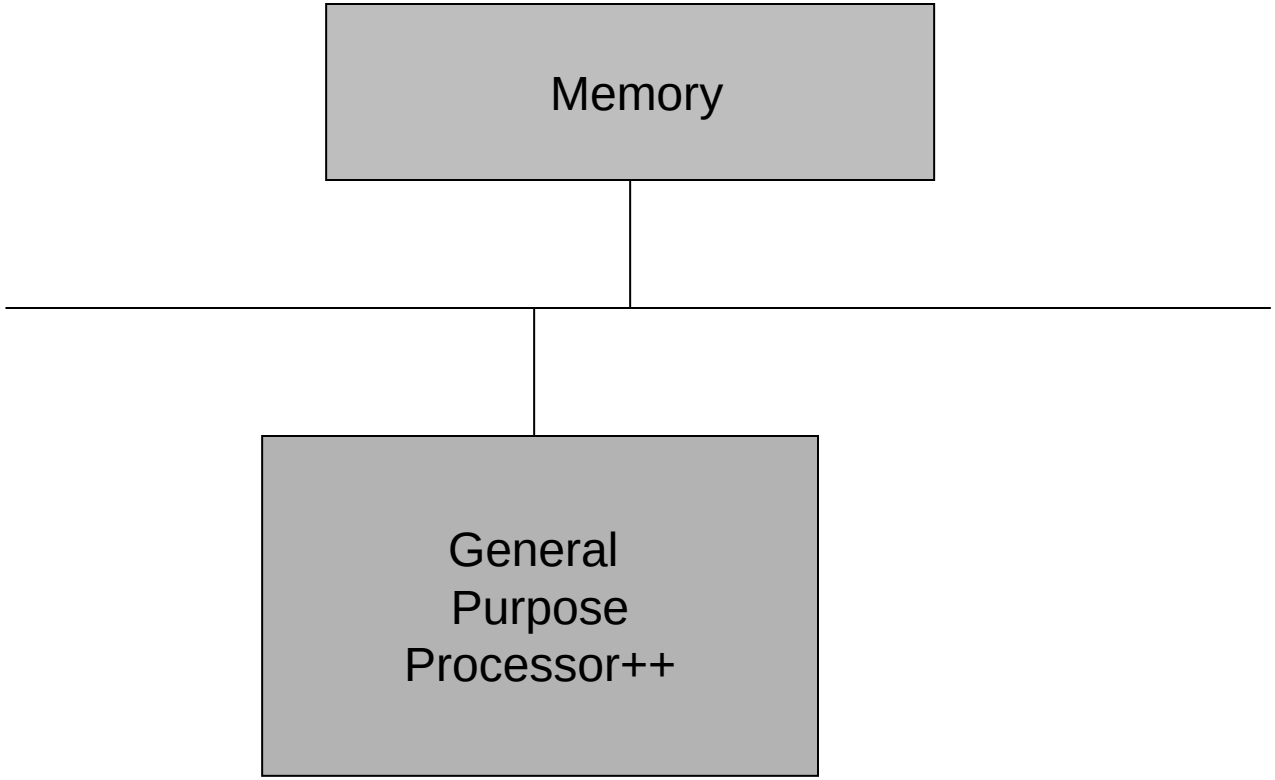
Observations

- VLIW machines unsuccessful in the GP world
- Itanium
  - Relaxed issue rules -> EPIC
  - Out-of-order memory support?
  - More like superscalar
- DSP solution to software
  - Libraries
  - Not sustainable

Thesis

- VLIW only makes sense in special purpose ultra-high performance data streaming apps
- Software is more expensive than hardware
  - Less automated
  - More faulty
  - Legacy languages with large time constants
- DSPs should look more like GP computers
  - Reduce VLIW features
  - More like GP computers
  - Easier to program

One Chip Solution?



Types of Computing

- Control intensive
  - Frequent conditional branches
  - (Super)scalar machines
- Data parallel
  - Loops
  - Vector/VLIW machines

Superscalar vs. VLIW

- Superscalar
  - complexity in the hardware
  - good performance on un-optimized software
  - known quantity
- VLIW
  - complexity in the compiler
  - requires hand coded kernels
  - portability compromised
  - life cycle costs

Software vs hardware

- Software – difficult to design and test
  - we accept buggy software
  - costly to produce
- Hardware – easier to design and test
  - we don’t accept bugs
  - many steps are automated
- Conclusion
  - be conservative about moving complexity from hardware to software
  - In those applications that are almost all data parallel => VLIW
  - Small traces of control code => Superscalar

Smart Camera Algorithms

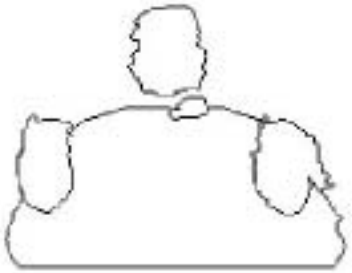
- Complete application
  - Background elimination
  - Skin area detection
  - Contour following
  - Super-ellipse fitting
  - Graph matching
- Mix of data parallel and control type algorithms

Processing a Frame

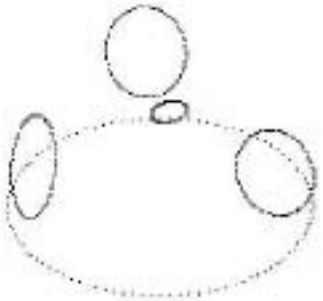
Original frame

Background elimination

Skin area detection



Contour following



Superellipse fitting

GSM Algorithms

- Coder and decoder – etsi.org
  - part of a digital cellular telecommunications system (GSM - Phase 2+) used for coding and decoding speech
  - specifically the GSM Enhanced Full Rate speech codec

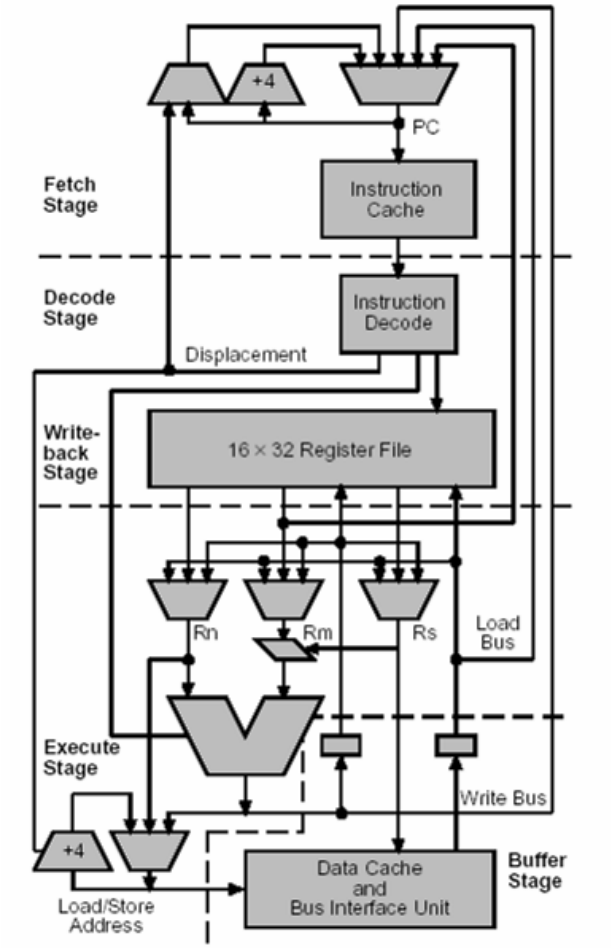
StrongARM

ARM7:

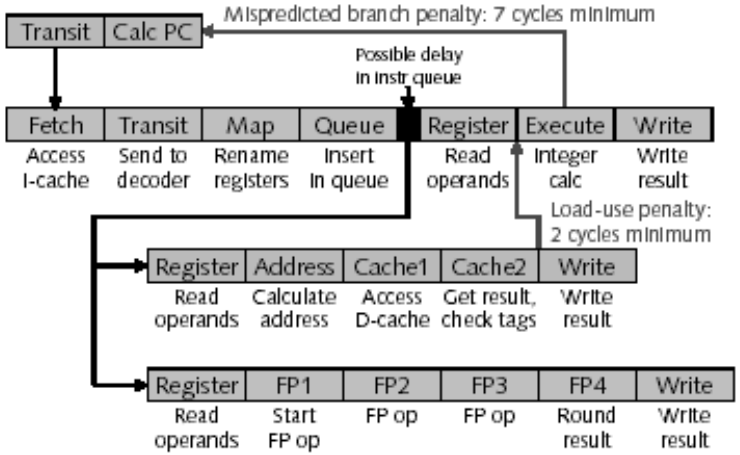
Fetch	Decode	Execute
latch instr	decode	shift/rotate ALU op D-cache access sign/zero ext commit result

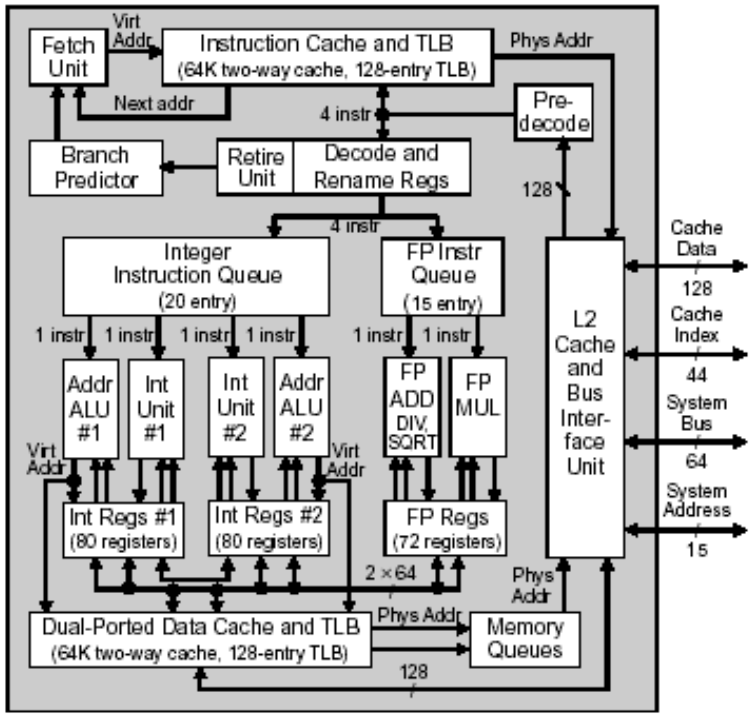
StrongArm:

Fetch	Decode	ALU	Cache	Writeback
calc PC I-cache access	decode cc mux clk ctrl CAM bypass reg file ctrl	shift/rotate ALU op	D-cache access sign/zero ext post result	commit write



Alpha 21264


Mispredicted branch penalty: 7 cycles minimum
Transit
Calc PC
Possible delay in instr queue
Fetch
Transit
Map
Queue
Register
Execute
Write
Access I-cache
Send to decoder
Rename registers
Insert in queue
Read operands
Integer calc
Write result
Load-use penalty: 2 cycles minimum
Register
Address
Cache1
Cache2
Write
Read operands
Calculate address
Access D-cache
Get result, check tags
Write result
Register
FP1
FP2
FP3
FP4
Write
Read operands
Start FP op
FP op
FP op
Round result
Write result


Fetch Unit
Virt Addr
Instruction Cache and TLB
(64K two-way cache, 128-entry TLB)
Phys Addr
Next addr
4 instr
Pre-decode
Branch Predictor
Retire Unit
Decode and Rename Regs
128
4 instr
Integer Instruction Queue
(20 entry)
FP Instr Queue
(15 entry)
1 instr
Addr ALU #1
Int Unit #1
Int Unit #2
Addr ALU #2
FP ADD DIV, SQRT
FP MUL
Virt Addr
Int Regs #1
(80 registers)
Int Regs #2
(80 registers)
FP Regs
(72 registers)
2 × 64
Phys Addr
Dual-Ported Data Cache and TLB
(64K two-way cache, 128-entry TLB)
Memory Queues
128
L2 Cache and Bus Interface Unit
Cache Data
128
Cache Index
44
System Bus
64
System Address
15

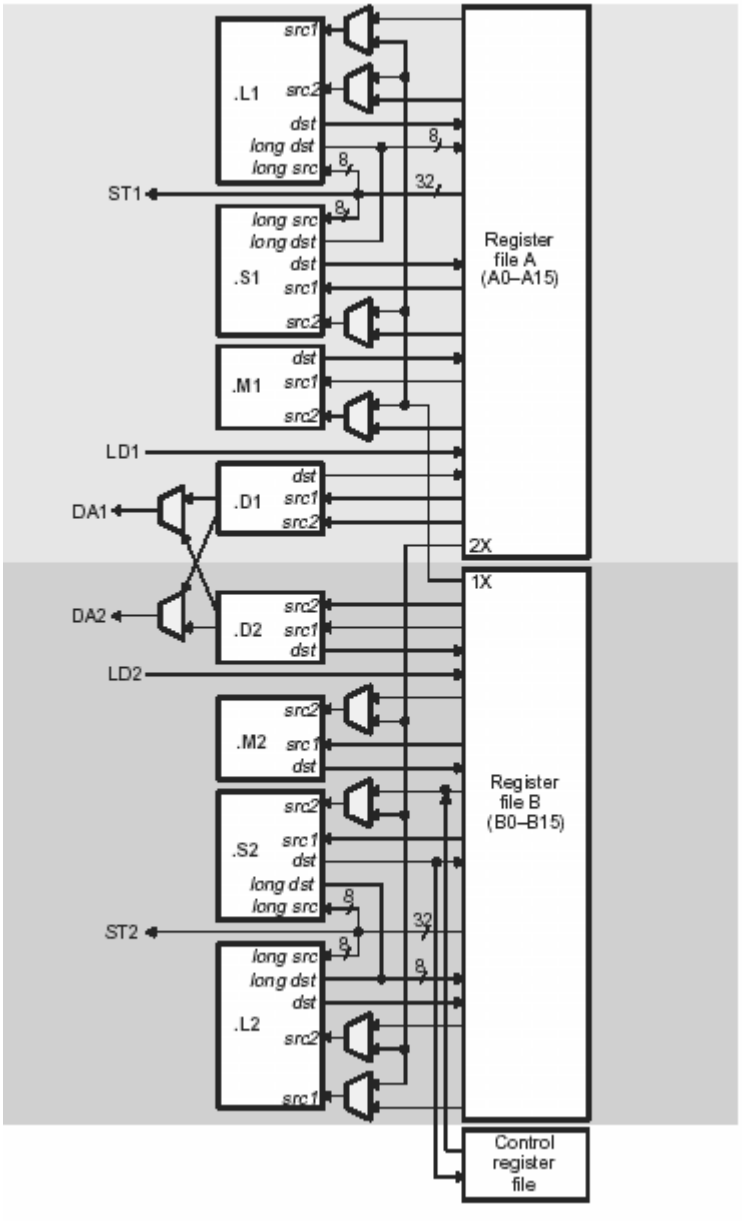
SA1 and Alpha Core Configs

	SA1 Core	Alpha Core
Instruction Fetch Queue Size	8	32
Extra Branch Misprediction Latency	1	3
Branch Predictor	Not Taken	Gshare (4096 entry, 8 bit hist)
BTB	NA	1024 sets, 4 way associative
Pipeline	in order	out of order
Decode Width	1	4
Issue Width	1	4
Commit Width	1	4
Register Update Unit Size	4	128
Load/Store Queue Size	4	64
Memory Disambiguation	perfect	perfect
Cache	perfect	perfect
Memory Latency	1	1
Memory Width	4 bytes	8 bytes
Pipelined Memory	no	yes
# Integer ALUs	1	4
# Integer Mult	1	1
# Memory Ports	1	2
# Floating Point ALUs	1	4
# Floating Point Mult	1	1

TI C62x

- VLIW
  - Up to 8, 32-bit instructions per cycle
  - RISC-like ISA
- 2 - Cluster Architecture
- Per cluster:
  - 16 General Purpose Registers
  - 4 Fully-Pipelined Functional Units
  - One crosspath to other cluster
- Predicated execution
- Multi-cycle latency instructions

Execution Core



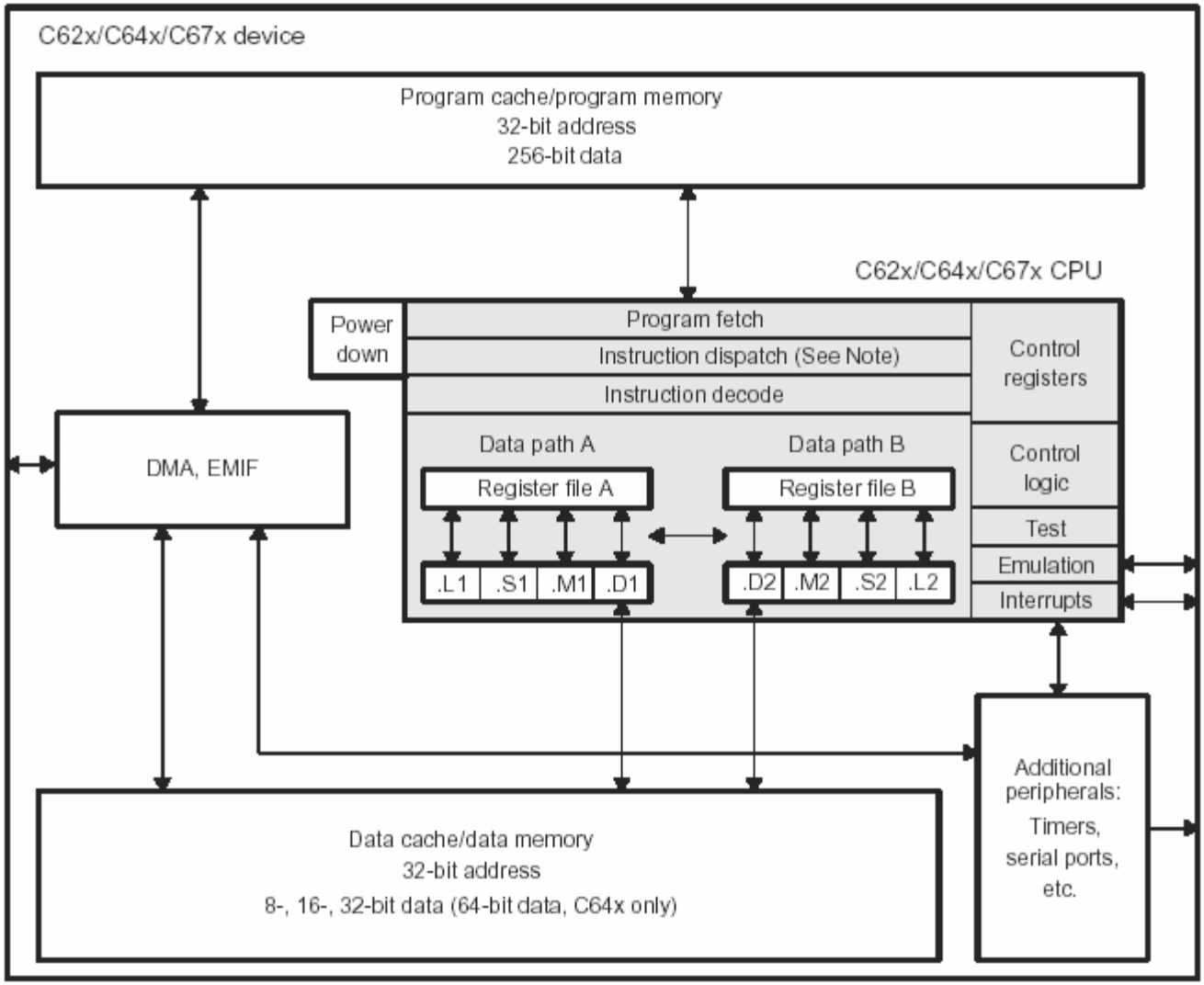
Functional Units

- L-Unit
  - 32/40-bit Arithmetic
  - 32-bit Logical Operations
  - 32/40-bit Compare Operations
  - Leftmost 1 or 0 counting for 32-bit
  - Normalization count for 32 and 40-bit
- D-Unit
  - 32-bit Add and Subtract (linear and circular addressing)
  - Loads and Stores with 5-bit constant offset
  - Loads and Stores with 15-bit constant offset (.D2 unit only)

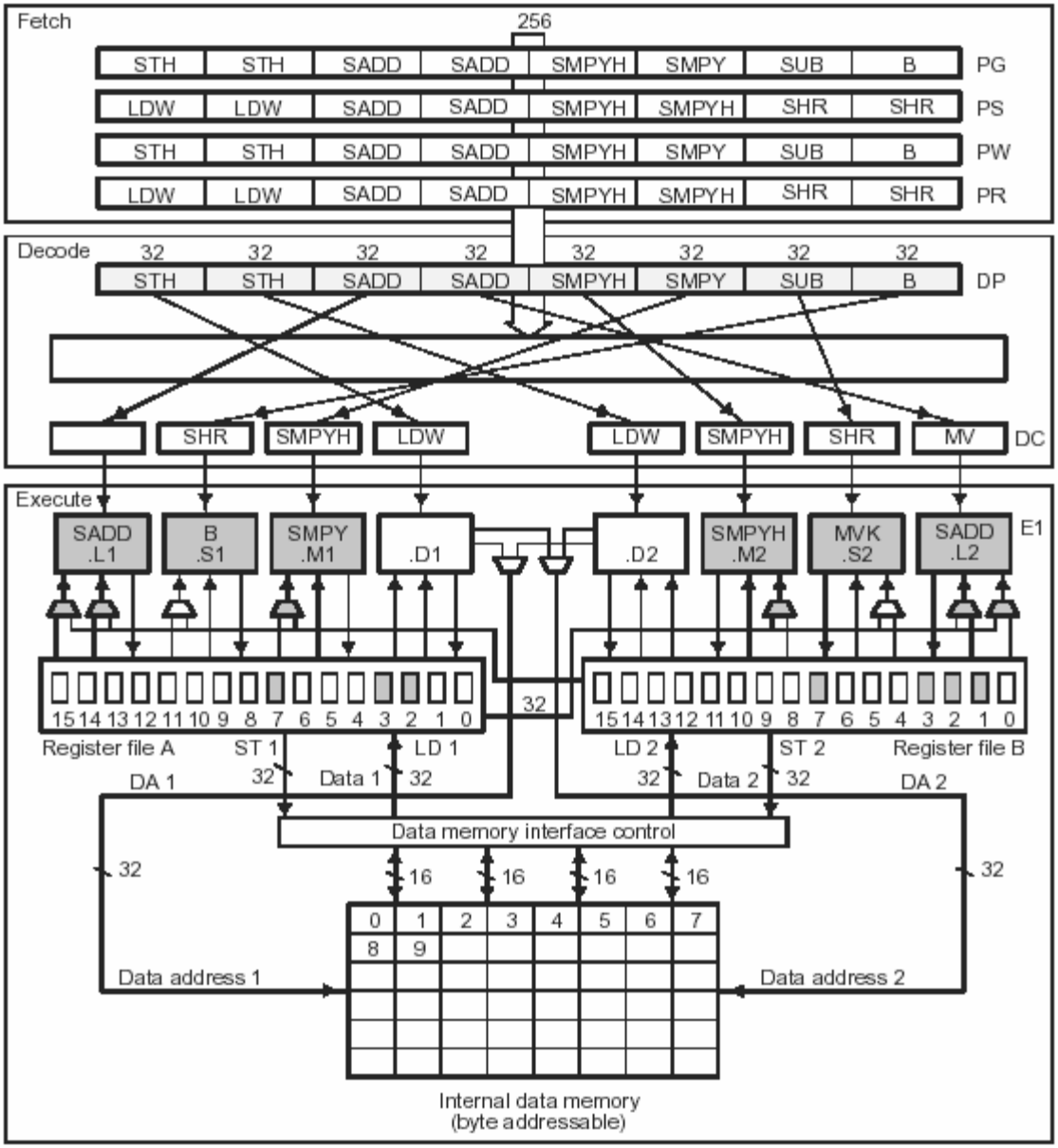
Functional Units

- S-Unit
  - 32-bit Arithmetic
  - 32-bit Logical Operations
  - 32/40-bit Shifts
  - 32-bit Bit-field Operations
  - Branches
  - Constant Generation
  - Control Register Access (.S2 unit only)
- M-Unit
  - 16x16 Multiply

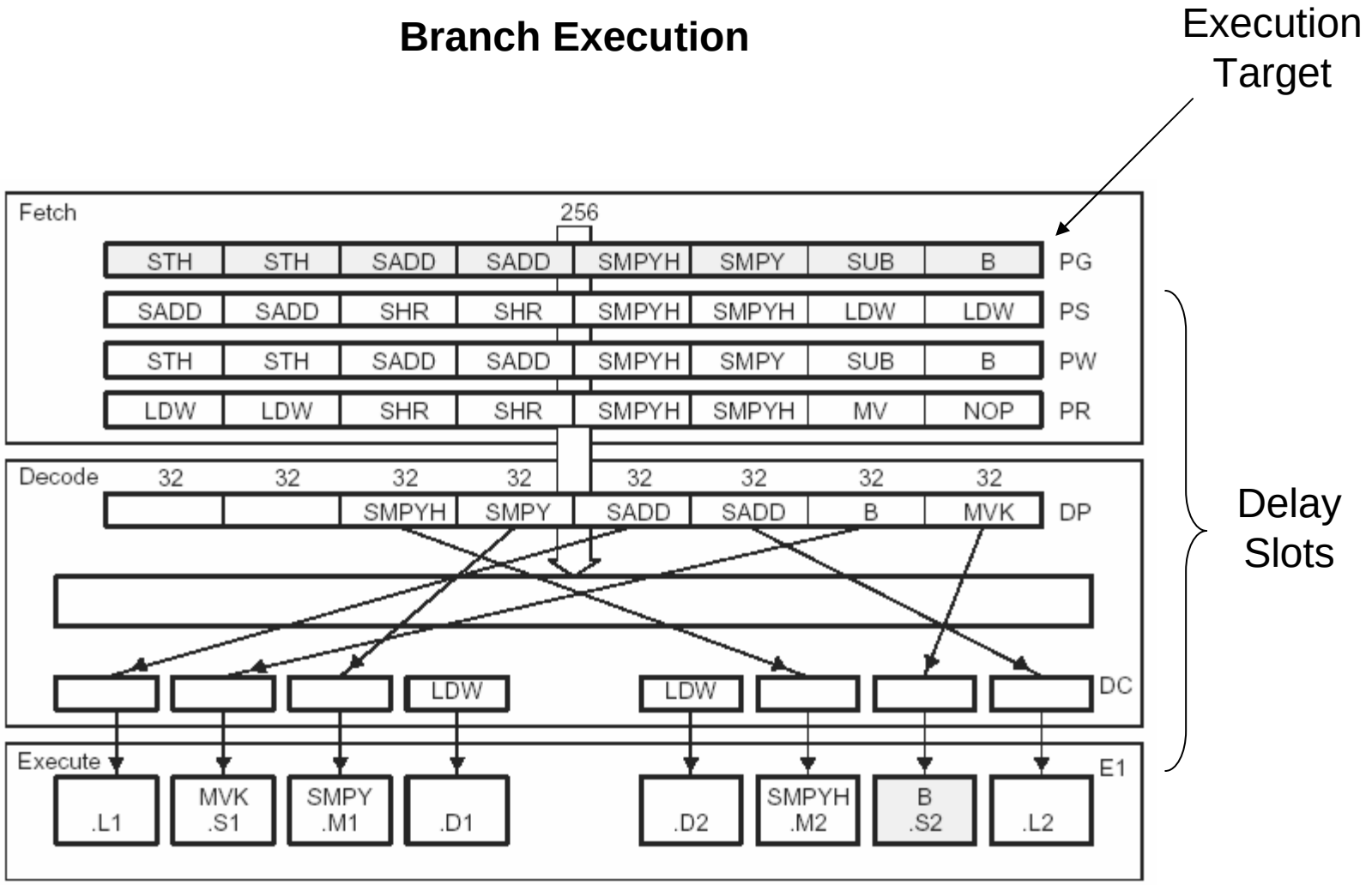
The Core


C62x/C64x/C67x device
Program cache/program memory
32-bit address
256-bit data
C62x/C64x/C67x CPU
Power down
Program fetch
Instruction dispatch (See Note)
Instruction decode
Control registers
Data path A
Data path B
Register file A
Register file B
Control logic
Test
Emulation
Interrupts
.L1
.S1
.M1
.D1
.D2
.M2
.S2
.L2
DMA, EMIF
Additional peripherals:
Timers, serial ports, etc.
Data cache/data memory
32-bit address
8-, 16-, 32-bit data (64-bit data, C64x only)

The Pipeline



Branch Execution



Clustering/Architectural Features

- Only one FU can use a crosspath at a time
- Limits placed on which FUs can use the crosspath for their srcs
  - L Unit: src1, src2
  - S Unit: src2
  - M Unit: src2
  - D Unit: None
- D Unit from other cluster can generate memory address for current cluster
- Load/Store with 15-bit offset only available to .D2 Unit and can only use registers B14/B15
- MVC (move to control register) only available to .S2

Code Compression Support

- Multi-cycle NOPs
  - Expands into up to 9 single cycle NOPs
  - Branches can interrupt the execution
  - Allows for compacting of strings of NOPs in the code

ISA

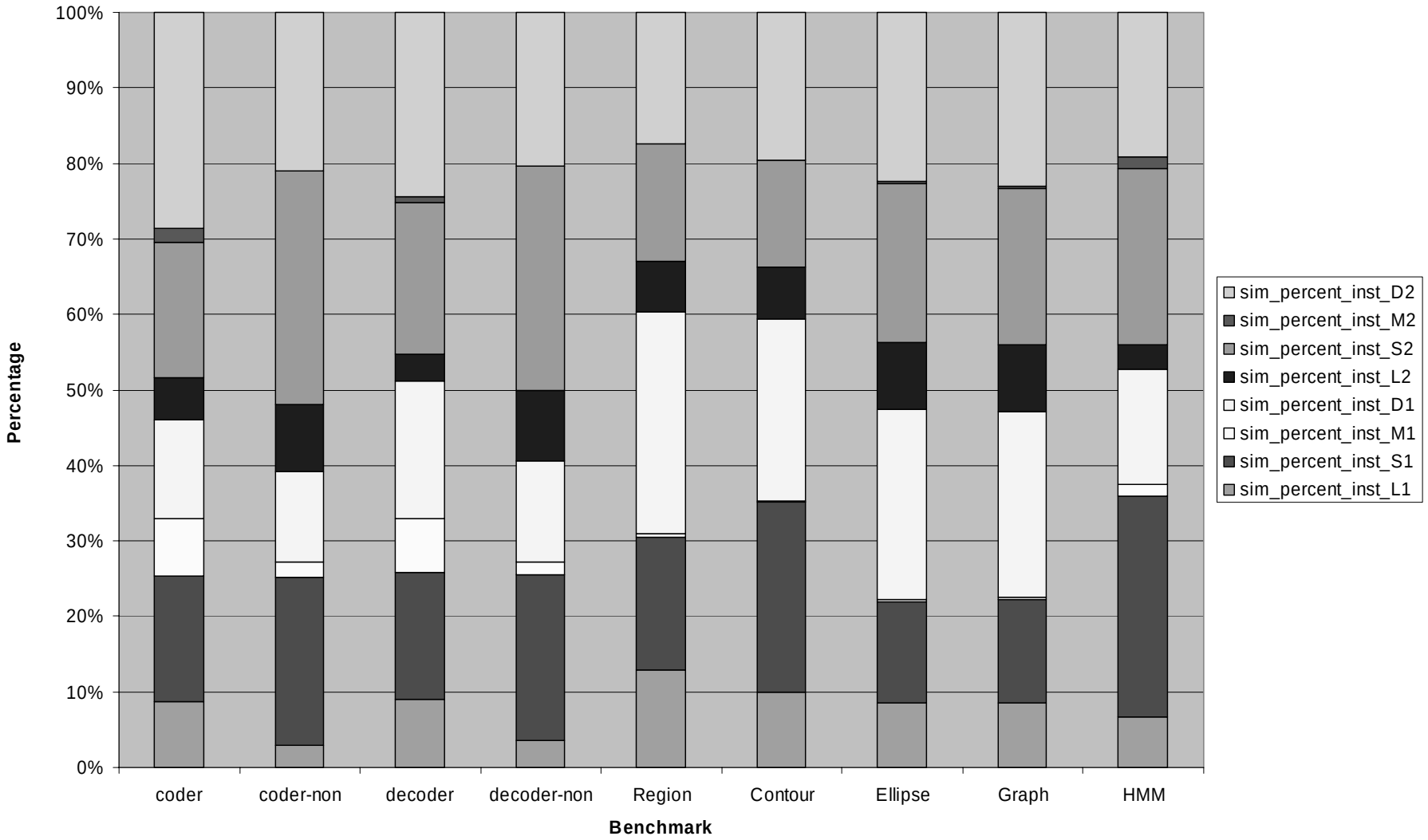
- RISC like
- Small set of fixed latency instruction types
  - Whole pipeline stalls on a cache miss
- Extras
  - Support for 40-bit arithmetic
  - Support for bit field manipulation (extract, set, clear, bit-count)
  - Support for saturation
  - Support for 5-bit constants for source 2 instead of a register

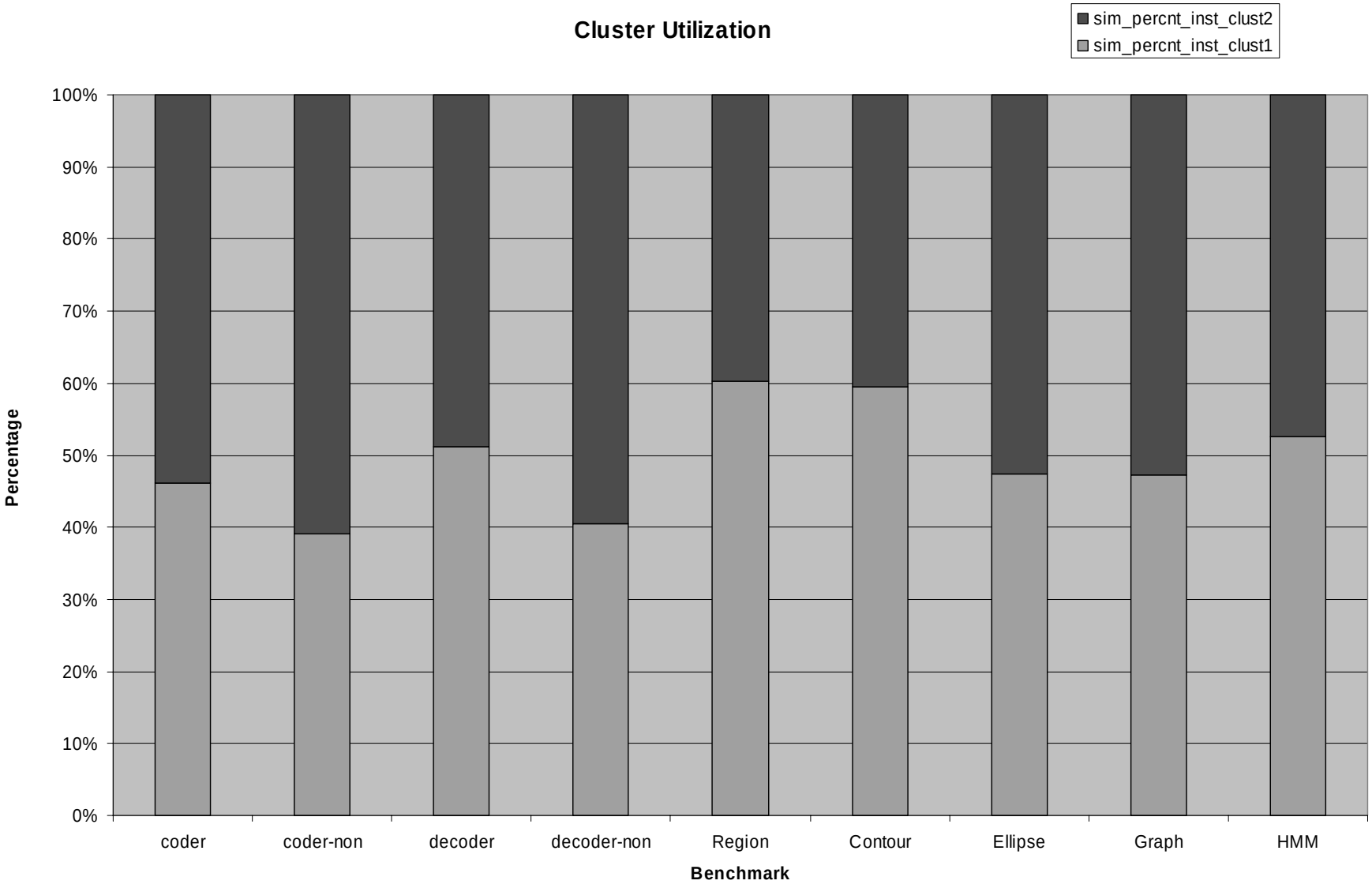
More ISA

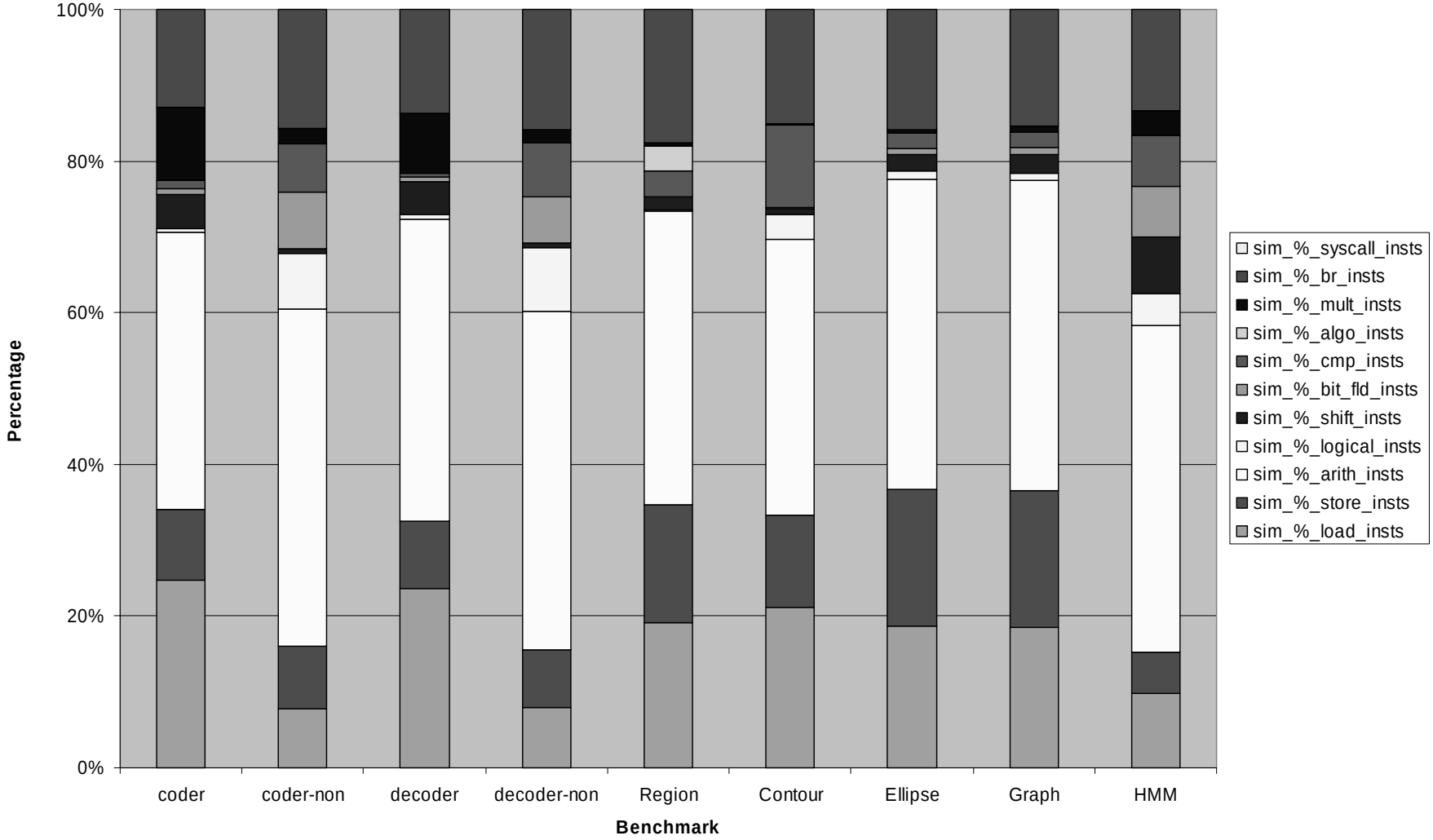
- Limitations
  - C62x has no floating point
  - Missing Integer Divide instruction (emulation provided by function call)
  - Integer Multiplies are only 16-bit (have to emulate 32-bit multiplies)
  - No Branch-and-Link (have to explicitly store the return address)
- Predication
  - 5 GPR registers used (A1,A2,B0,B1,B2)
  - Check either for Zero or Not Zero

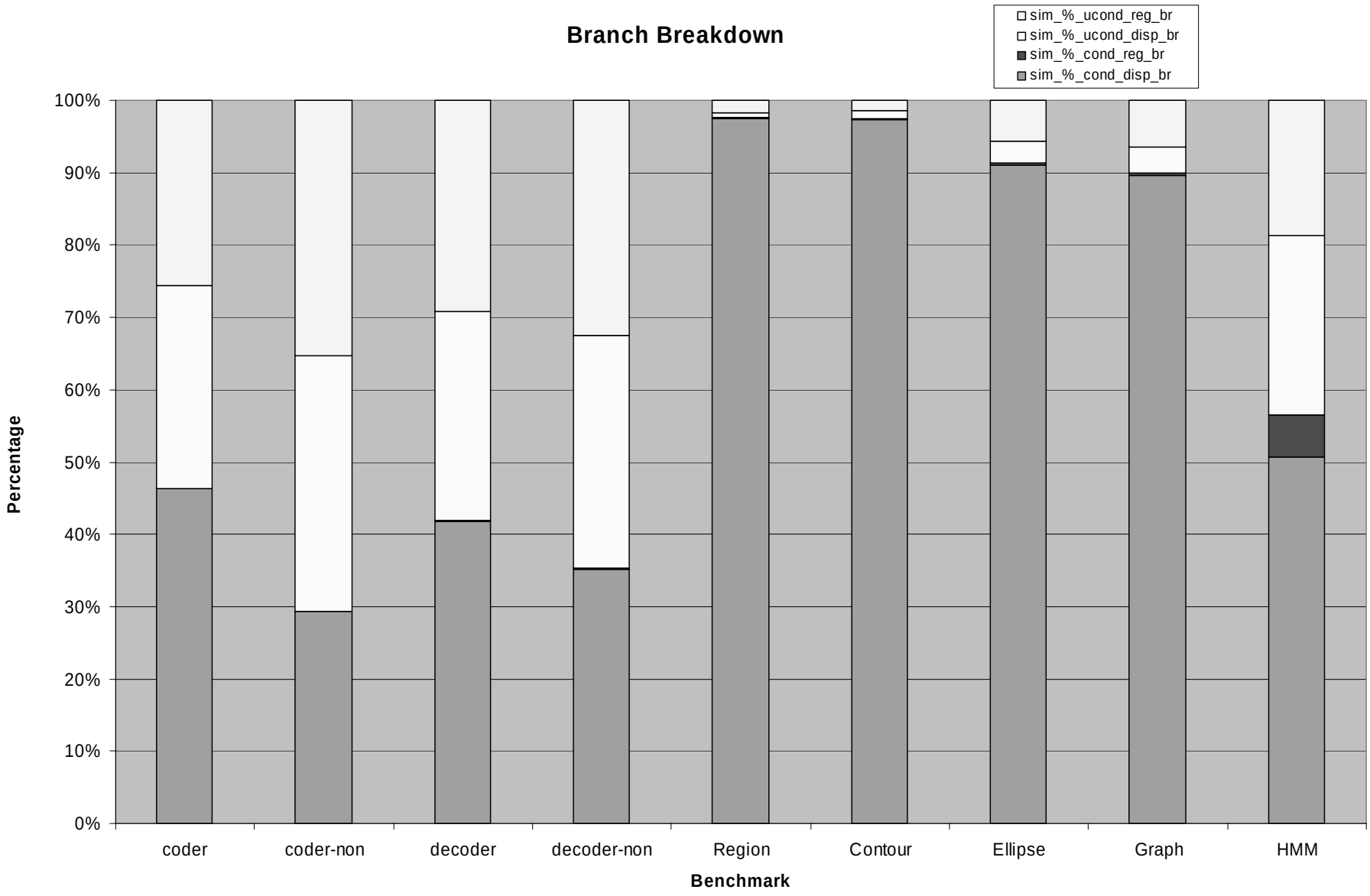
Simulation Results

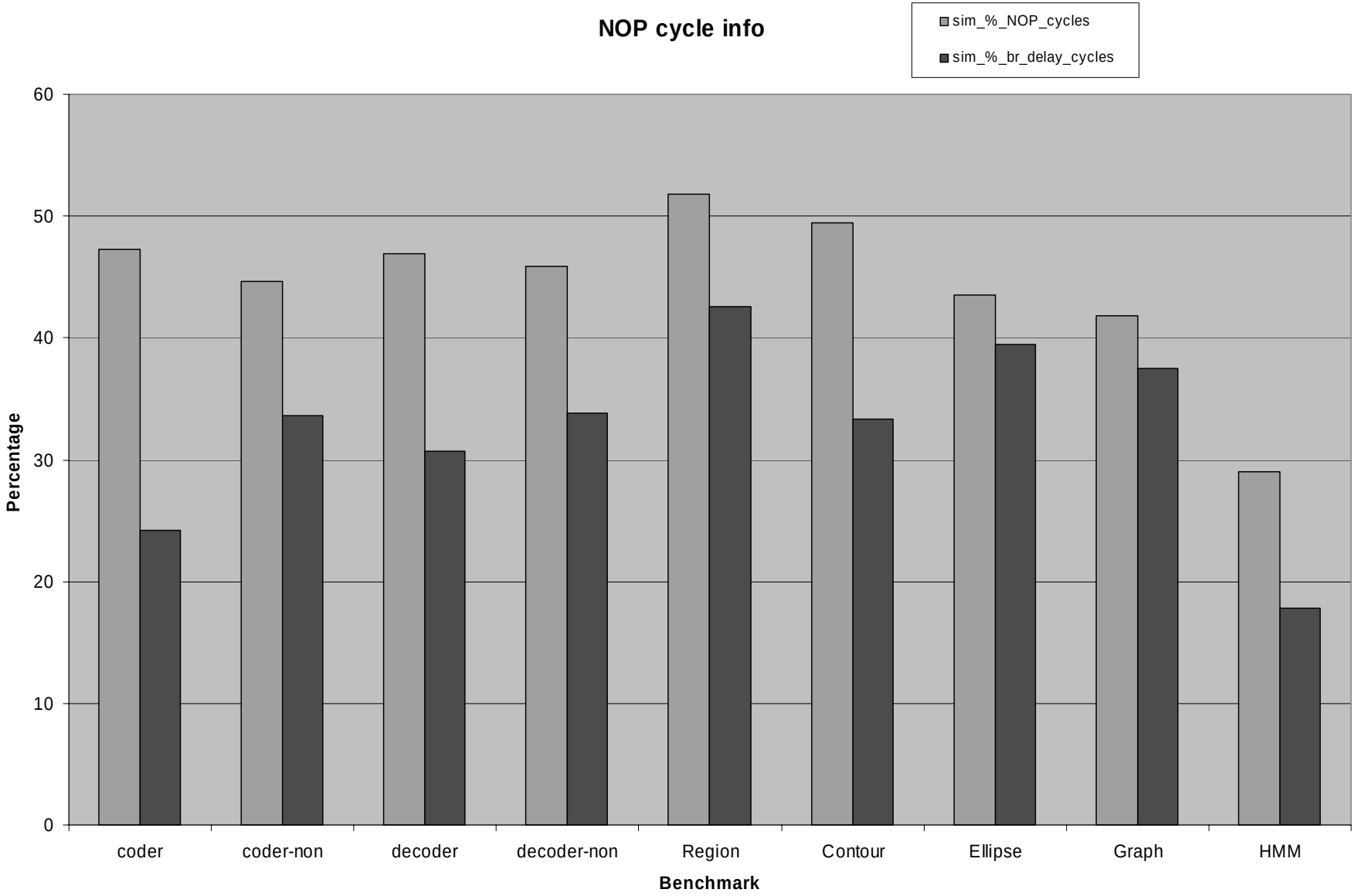
- Added a validated TMS320C6200 core to SimpleScalar (www.simplescalar.org)
- SS includes StrongARM core
- SS includes Alpha 21264 core

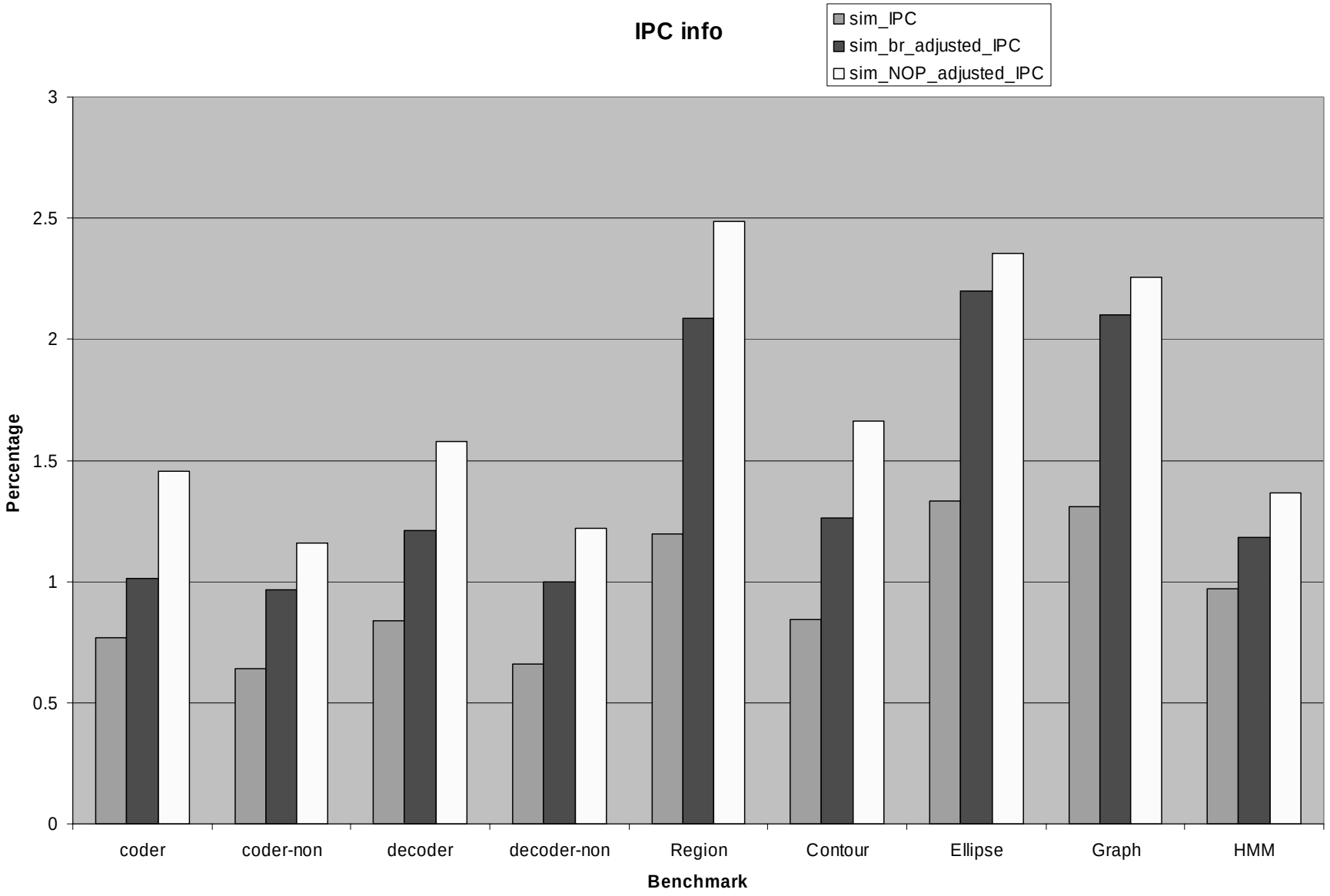

Functional Unit Usage
100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%
Percentage
coder
coder-non
decoder
decoder-non
Region
Contour
Ellipse
Graph
HMM
Benchmark
sim_percent_inst_D2
sim_percent_inst_M2
sim_percent_inst_S2
sim_percent_inst_L2
sim_percent_inst_D1
sim_percent_inst_M1
sim_percent_inst_S1
sim_percent_inst_L1


Cluster Utilization
sim_percnt_inst_clust2
sim_percnt_inst_clust1
100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%
Percentage
coder
coder-non
decoder
decoder-non
Region
Contour
Ellipse
Graph
HMM
Benchmark

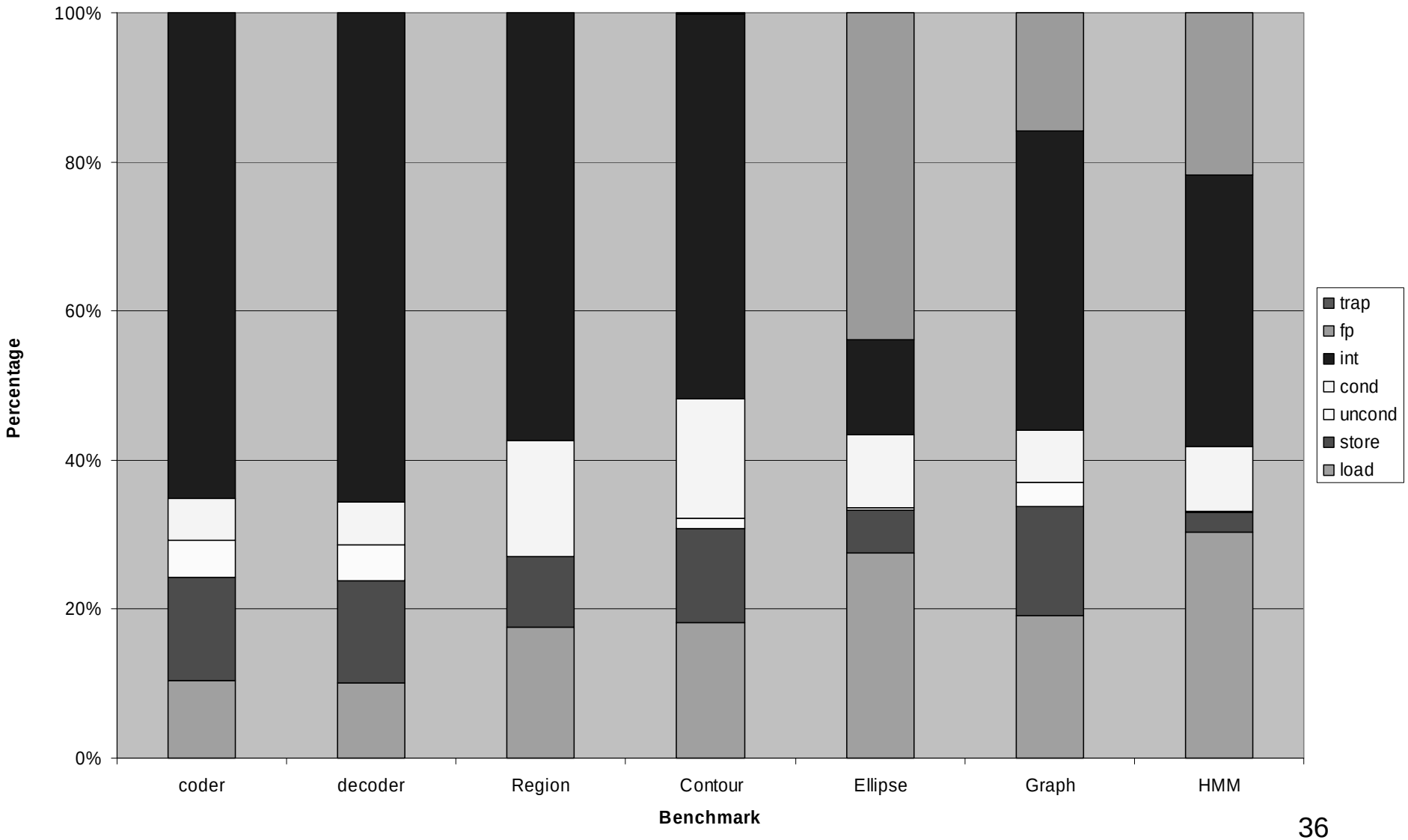

Inst Class Breakdown
100%
80%
60%
40%
20%
0%
Percentage
coder
coder-non
decoder
decoder-non
Region
Contour
Ellipse
Graph
HMM
Benchmark
sim_%_syscall_insts
sim_%_br_insts
sim_%_mult_insts
sim_%_algo_insts
sim_%_cmp_insts
sim_%_bit_fld_insts
sim_%_shift_insts
sim_%_logical_insts
sim_%_arith_insts
sim_%_store_insts
sim_%_load_insts

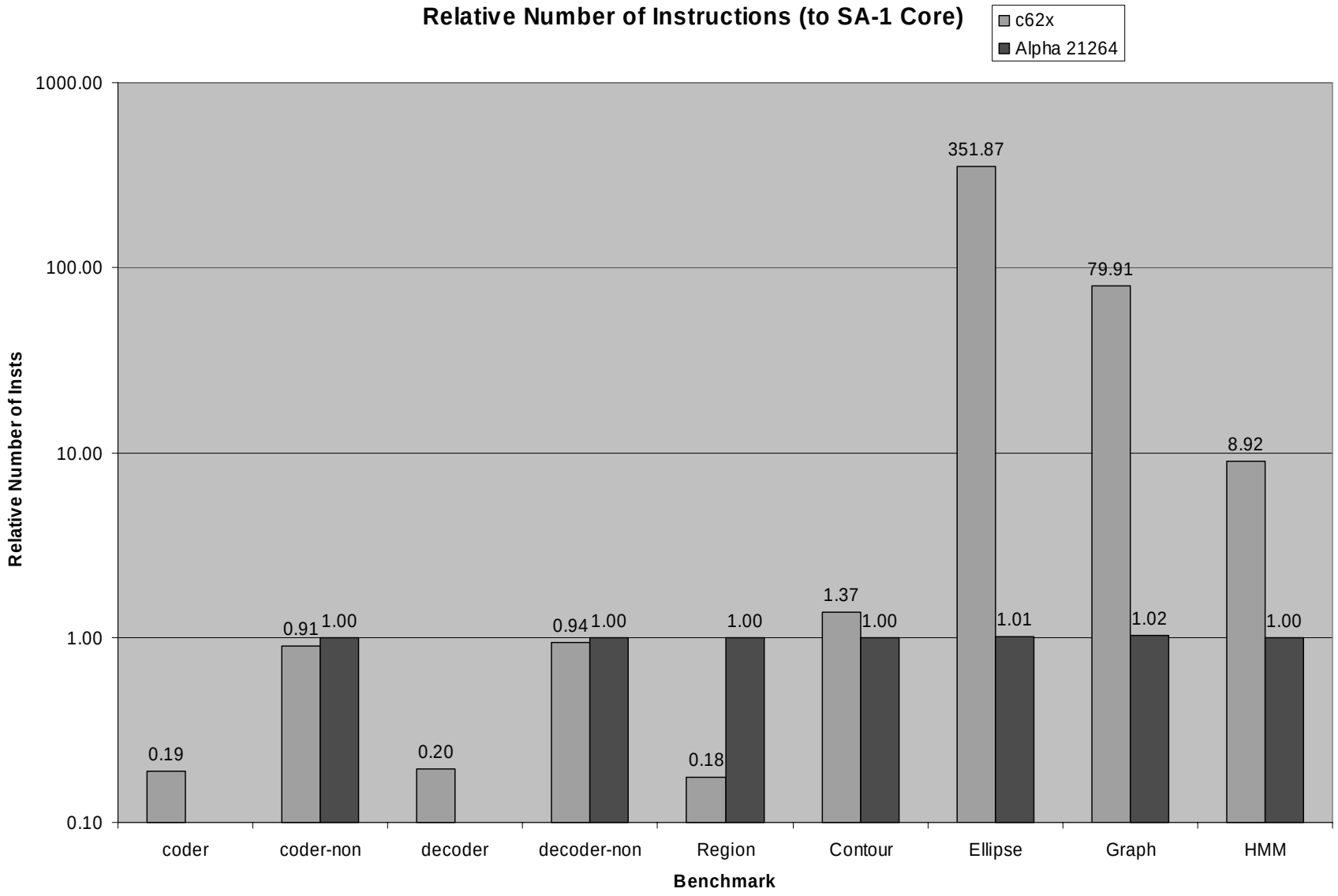

Branch Breakdown
sim_%_ucond_reg_br
sim_%_ucond_disp_br
sim_%_cond_reg_br
sim_%_cond_disp_br
100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%
Percentage
coder
coder-non
decoder
decoder-non
Region
Contour
Ellipse
Graph
HMM
Benchmark

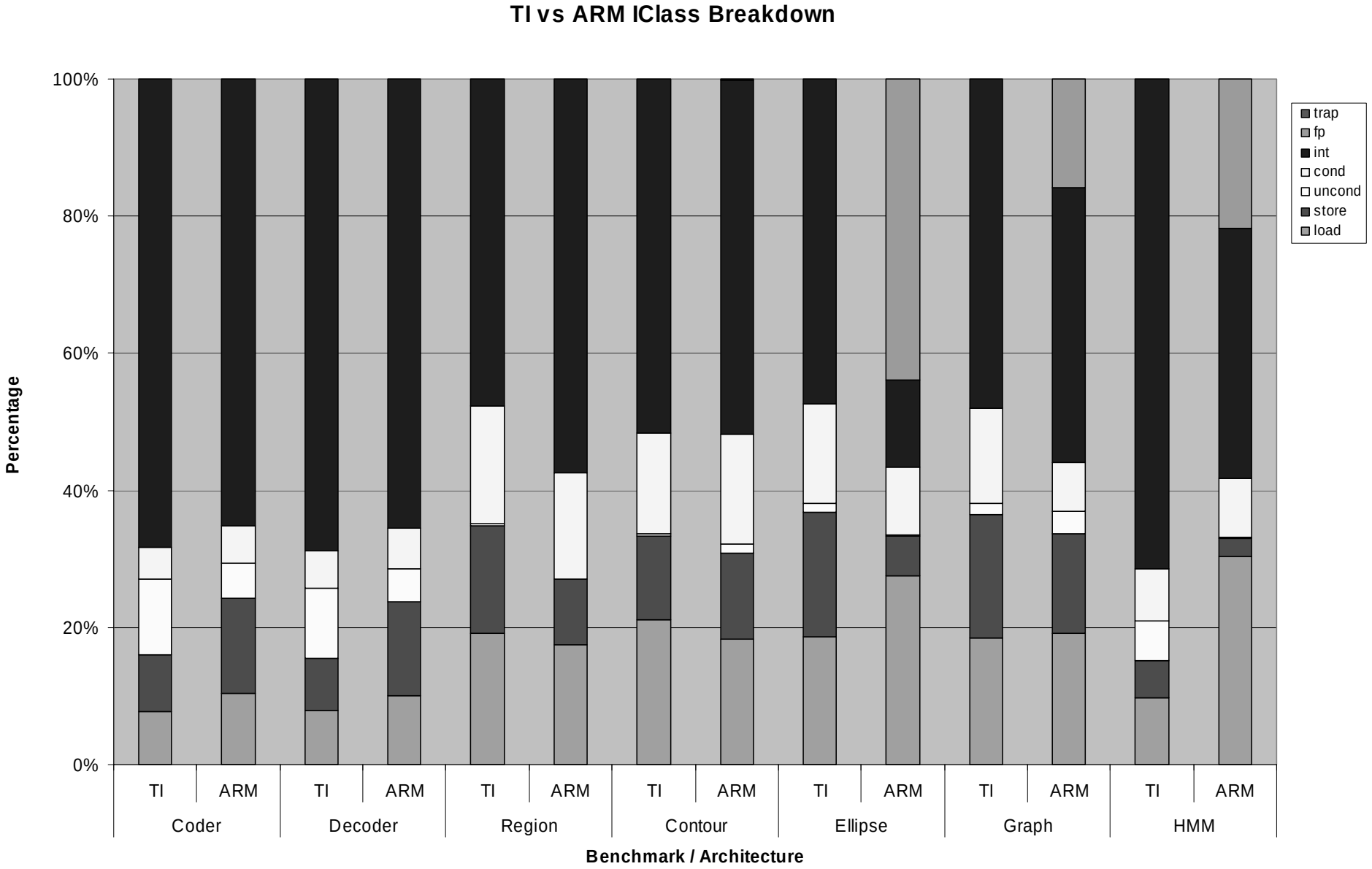

NOP cycle info
sim_%_NOP_cycles
sim_%_br_delay_cycles
60
50
40
30
20
10
0
Percentage
coder
coder-non
decoder
decoder-non
Region
Contour
Ellipse
Graph
HMM
Benchmark

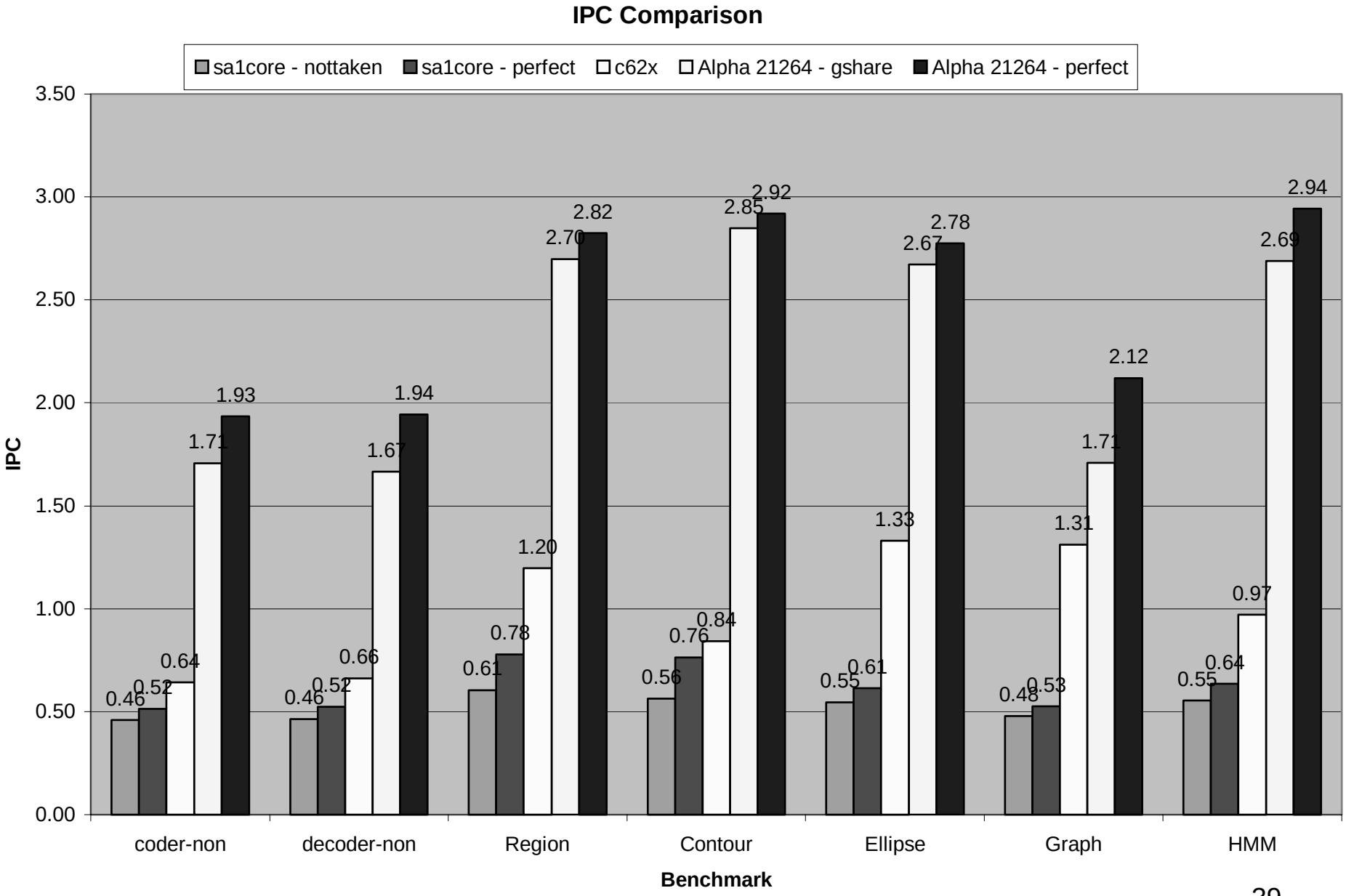

IPC info
sim_IPC
sim_br_adjusted_IPC
sim_NOP_adjusted_IPC
Percentage
3
2.5
2
1.5
1
0.5
0
coder
coder-non
decoder
decoder-non
Region
Contour
Ellipse
Graph
HMM
Benchmark

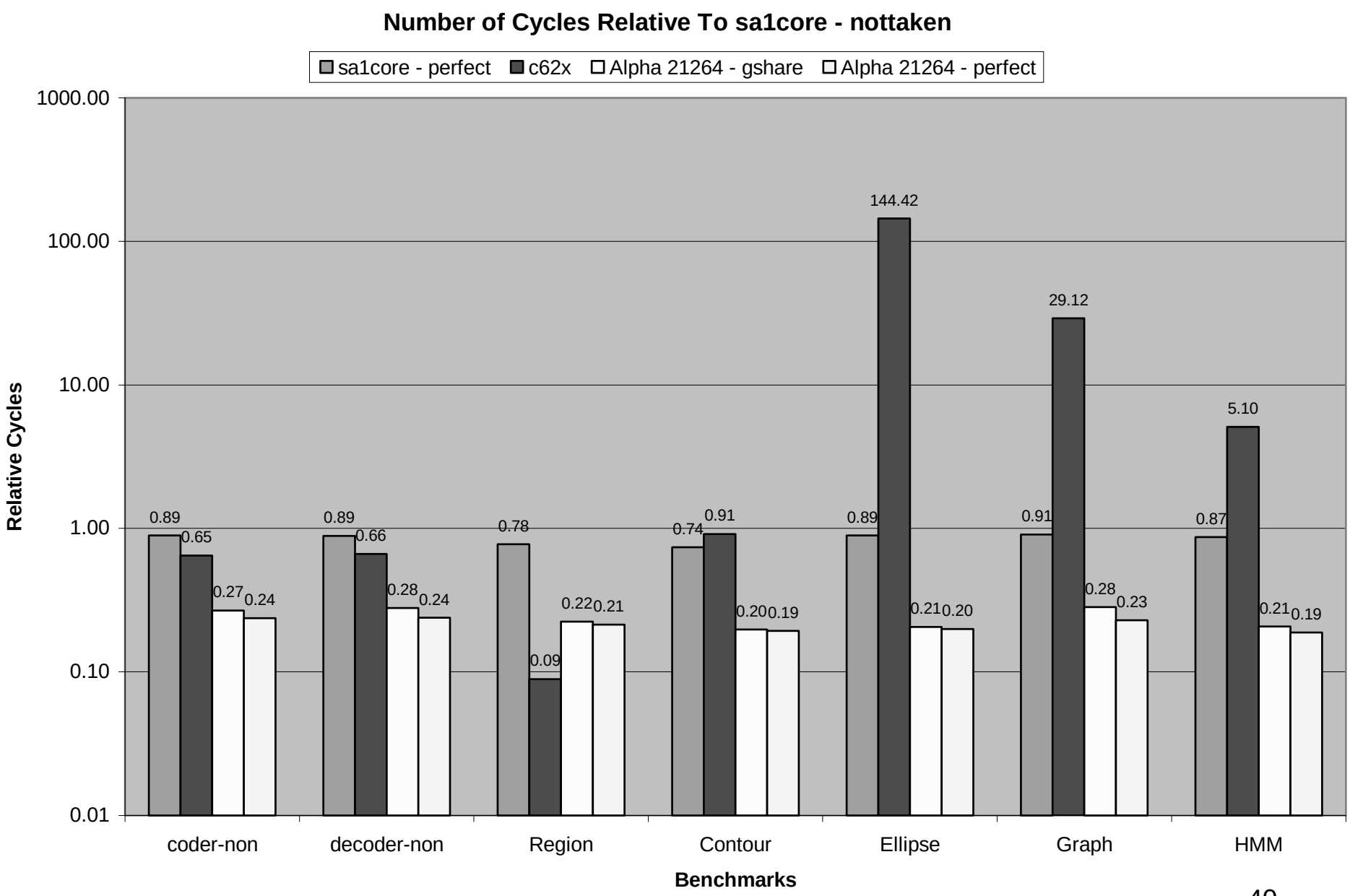
StrongARM vs C62 Stats


ARM Inst Class Breakdown
100%
80%
60%
40%
20%
0%
Percentage
coder
decoder
Region
Contour
Ellipse
Graph
HMM
Benchmark
trap
fp
int
cond
uncond
store
load


Relative Number of Instructions (to SA-1 Core)
c62x
Alpha 21264
1000.00
100.00
10.00
1.00
0.10
Relative Number of Insts
0.19
0.91
1.00
0.20
0.94
1.00
0.18
1.00
1.37
1.00
351.87
1.01
79.91
1.02
8.92
1.00
coder
coder-non
decoder
decoder-non
Region
Contour
Ellipse
Graph
HMM
Benchmark

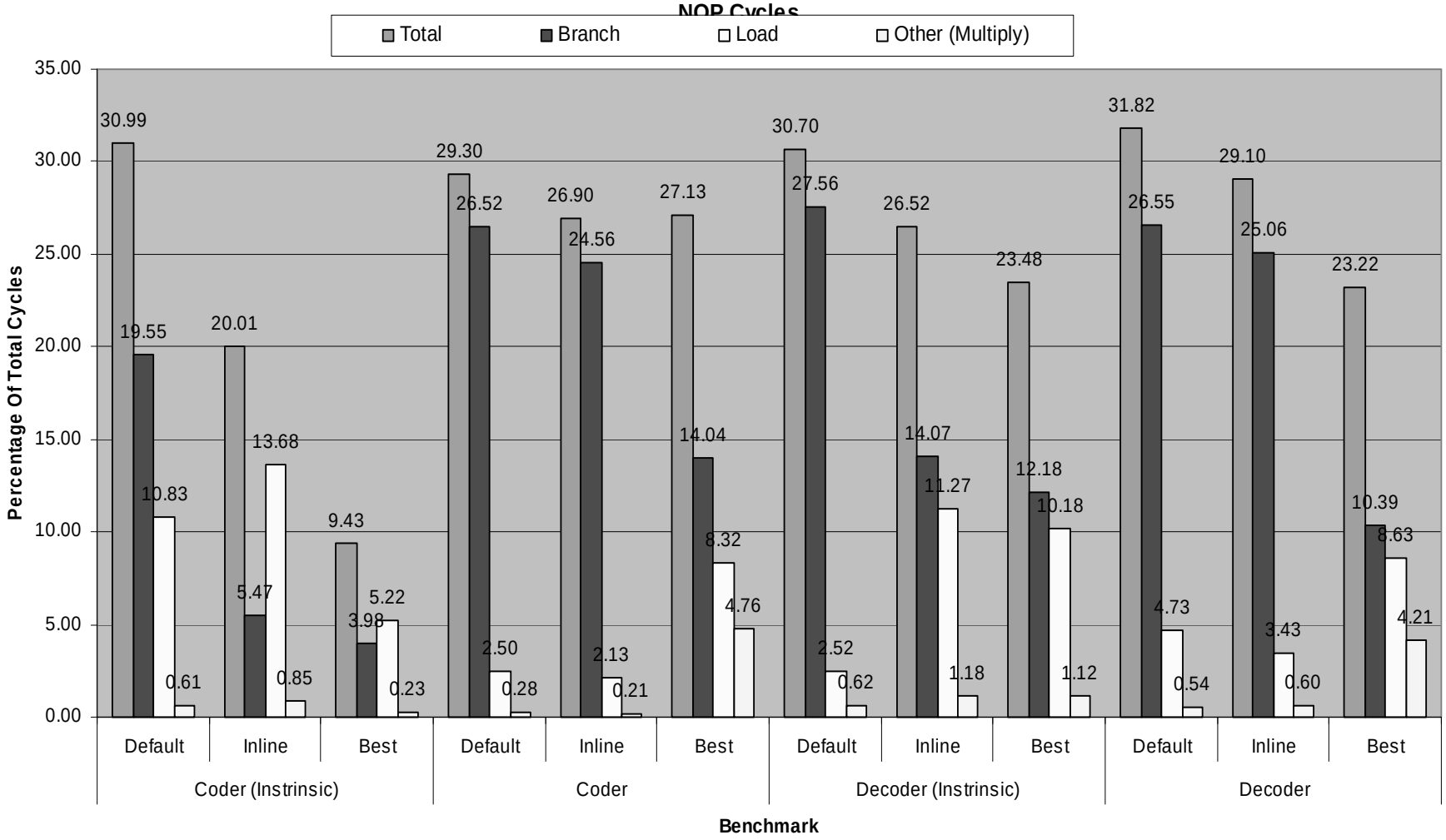

TI vs ARM IClass Breakdown
100%
80%
60%
40%
20%
0%
Percentage
trap
fp
int
cond
uncond
store
load
TI
ARM
Coder
Decoder
Region
Contour
Ellipse
Graph
HMM
Benchmark / Architecture

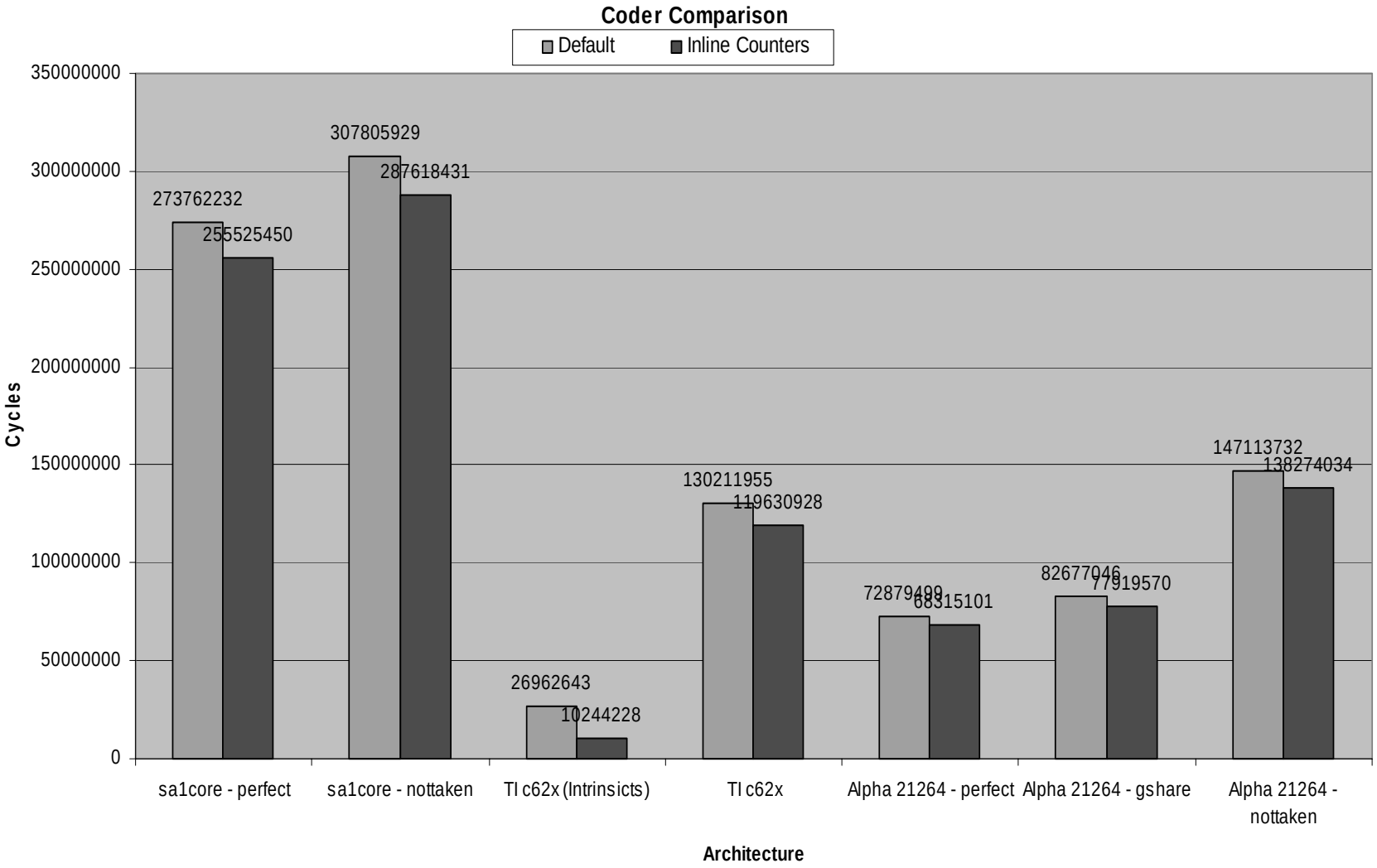

IPC Comparison
sa1core - nottaken
sa1core - perfect
c62x
Alpha 21264 - gshare
Alpha 21264 - perfect
IPC
3.50
3.00
2.50
2.00
1.50
1.00
0.50
0.00
0.46
0.52
0.64
1.71
1.93
0.46
0.52
0.66
1.67
1.94
0.61
0.78
1.20
2.70
2.82
0.56
0.76
0.84
2.85
2.92
0.55
0.61
1.33
2.67
2.78
0.48
0.53
1.31
1.71
2.12
0.55
0.64
0.97
2.69
2.94
coder-non
decoder-non
Region
Contour
Ellipse
Graph
HMM
Benchmark

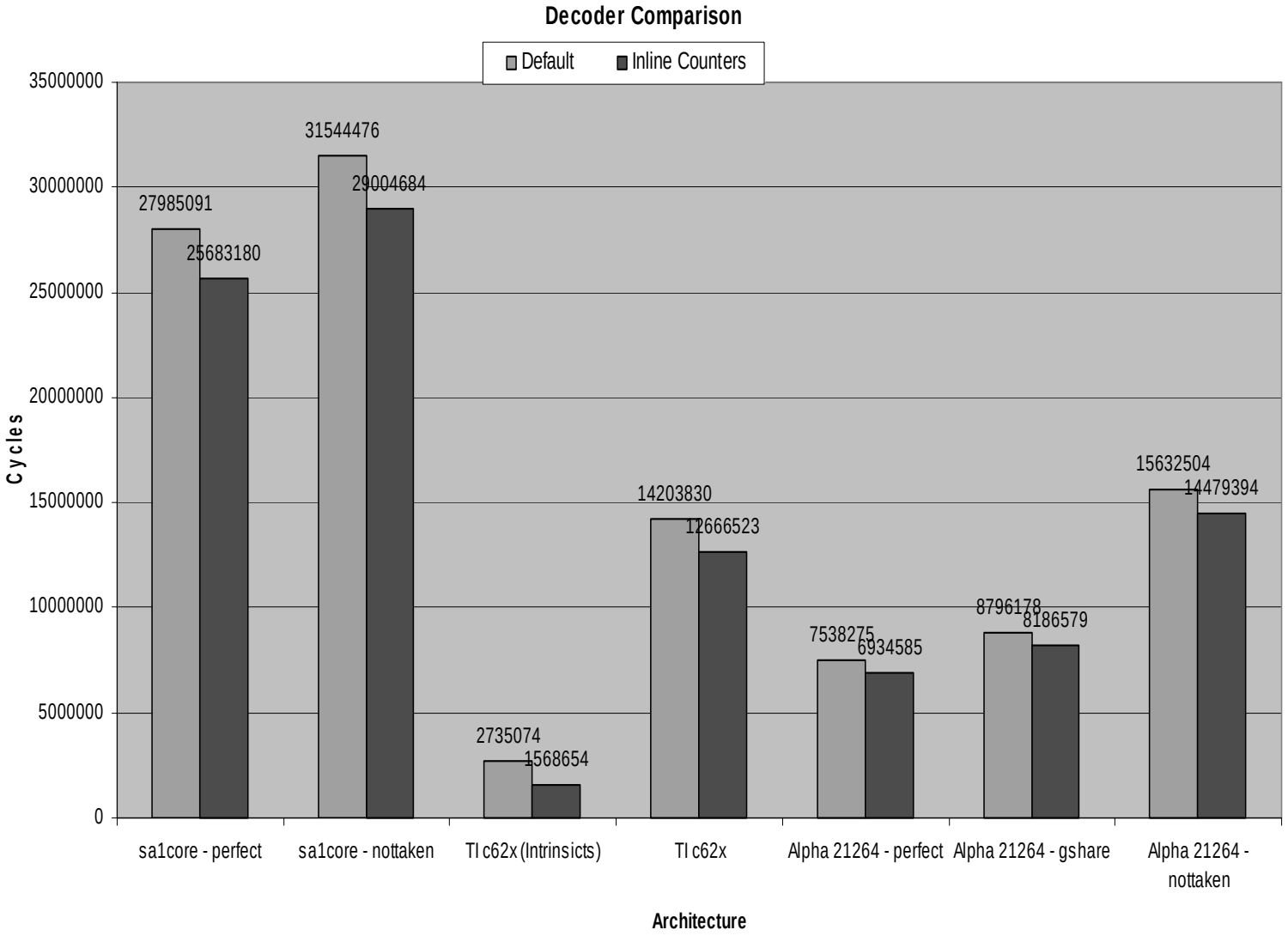

Number of Cycles Relative To sa1core - nottaken
sa1core - perfect
c62x
Alpha 21264 - gshare
Alpha 21264 - perfect
Relative Cycles
1000.00
100.00
10.00
1.00
0.10
0.01
0.89
0.65
0.27
0.24
0.89
0.66
0.28
0.24
0.78
0.09
0.22
0.21
0.74
0.91
0.20
0.19
0.89
144.42
0.21
0.20
0.91
29.12
0.28
0.23
0.87
5.10
0.21
0.19
coder-non
decoder-non
Region
Contour
Ellipse
Graph
HMM
Benchmarks

Diagnosis

- C62 compiler does not inline aggressively enough
- Codec is a better comparison because it does not include floating point


NOP Cycles
Total
Branch
Load
Other (Multiply)
Percentage Of Total Cycles
35.00
30.00
25.00
20.00
15.00
10.00
5.00
0.00
30.99
19.55
10.83
0.61
20.01
5.47
13.68
0.85
9.43
3.98
5.22
0.23
29.30
26.52
2.50
0.28
26.90
24.56
2.13
0.21
27.13
14.04
8.32
4.76
30.70
27.56
2.52
0.62
26.52
14.07
11.27
1.18
23.48
12.18
10.18
1.12
31.82
26.55
4.73
0.54
29.10
25.06
3.43
0.60
23.22
10.39
8.63
4.21
Default
Inline
Best
Coder (Instrinsic)
Coder
Decoder (Instrinsic)
Decoder
Benchmark


Coder Comparison
Default
Inline Counters
350000000
300000000
250000000
200000000
150000000
100000000
50000000
0
Cycles
273762232
255525450
307805929
287618431
26962643
10244228
130211955
119630928
72879499
68315101
82677046
77919570
147113732
138274034
sa1core - perfect
sa1core - nottaken
TI c62x (Intrinsicts)
TI c62x
Alpha 21264 - perfect
Alpha 21264 - gshare
Alpha 21264 - nottaken
Architecture


Decoder Comparison
Default
Inline Counters
35000000
30000000
25000000
20000000
15000000
10000000
5000000
0
Cycles
27985091
25683180
31544476
29004684
2735074
1568654
14203830
12666523
7538275
6934585
8796178
8186579
15632504
14479394
sa1core - perfect
sa1core - nottaken
TI c62x (Intrinsicts)
TI c62x
Alpha 21264 - perfect
Alpha 21264 - gshare
Alpha 21264 - nottaken
Architecture

Conclusion: Intrinsics more important

- In this case intrinsics include
- Saturated add/sub/mul
- Absolute value
- Various 16 field extracts

Question?

- Is there a “convergent” architecture that combines both and covers a large class of embedded applications

The End