Enhancing Data Cache Performance via Dynamic Allocation

George Murillo
Scott Noel
Joshua Robinson
Paul Willmann

04 / 22 / 03

Agenda

- Introduction
- Our hypothesis
- Proposed architecture
- Experimental evaluation
- Results and analysis
- Conclusions

Introduction

- Increased use of media programs on the desktop
- Identify features of media processors
  - Typically constrained in cache size and complexity
  - Instructions that provide greater control over cache
  - Large working sets of read-once, read-only data
- Apply features to general purpose processors

Hypothesis

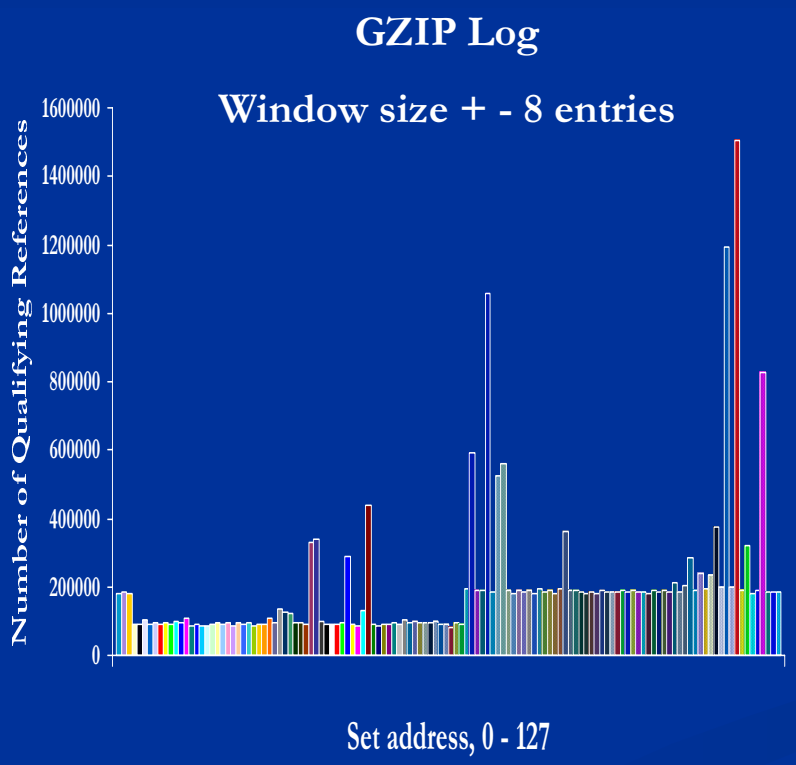
- Proposal
  - New cache management policy
  - Side buffer and load history table
  - Perform dynamic runtime analysis to predict read-only loads
  - Keep read-only data out of the cache, preventing pollution
- Expected results of our configuration
  - Perform better than unmodified same size caches
  - Perform similarly to unmodified larger caches

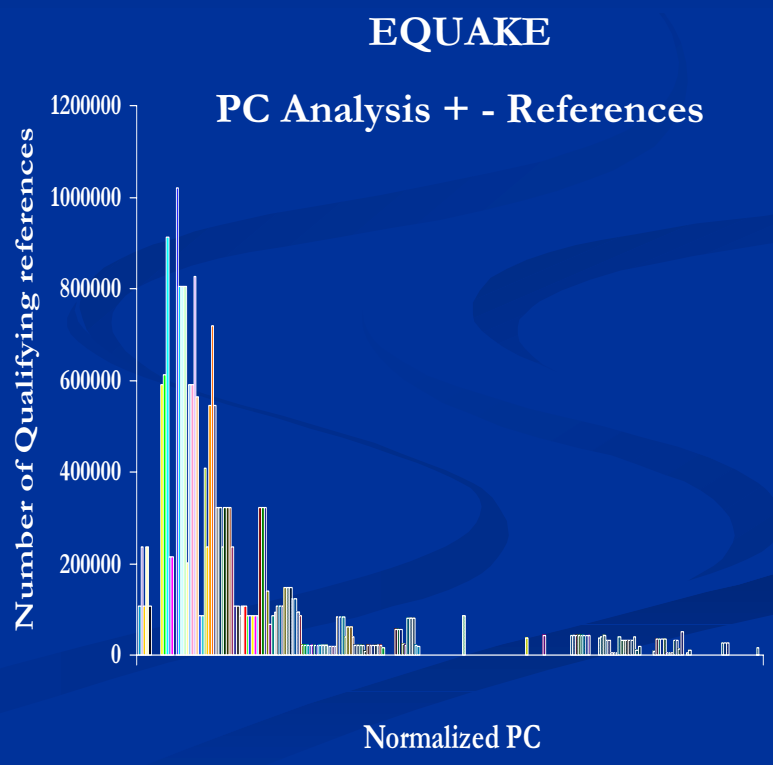
Motivation

- The problem with general purpose cache
  - Cache pollution is a major problem
  - Increase cache size to reduce cache misses
  - Large cache increase wire delay and overall latency
- Read-only data can cause cache pollution
  - Read-only data usually has less temporal locality
  - Forces important data out of the cache

Initial testing

- Percentage of memory references, read-only refs in a window
- Wide distribution, dispersion of accesses throughout the cache
- Predicable, dispersion by program counter

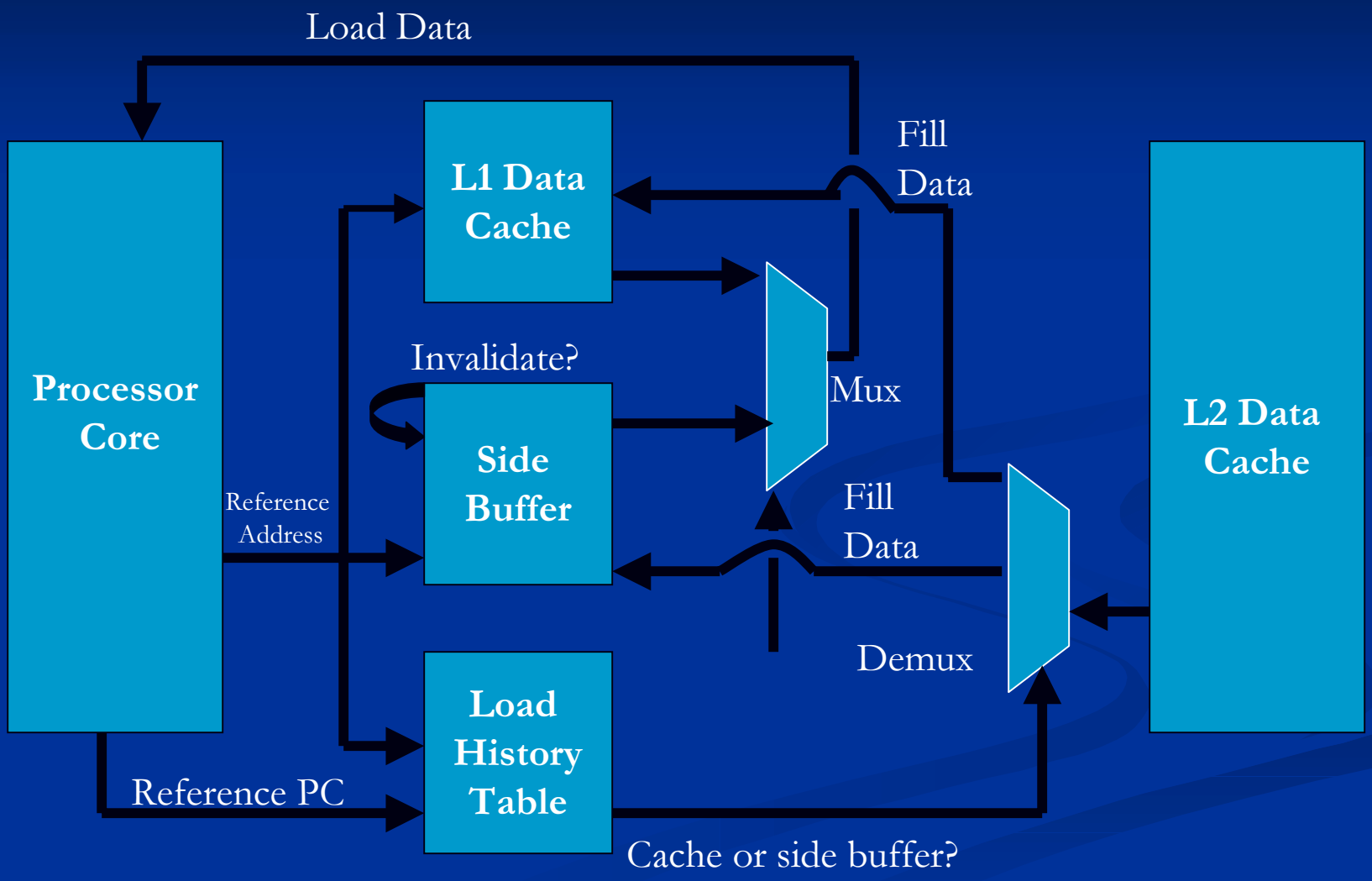




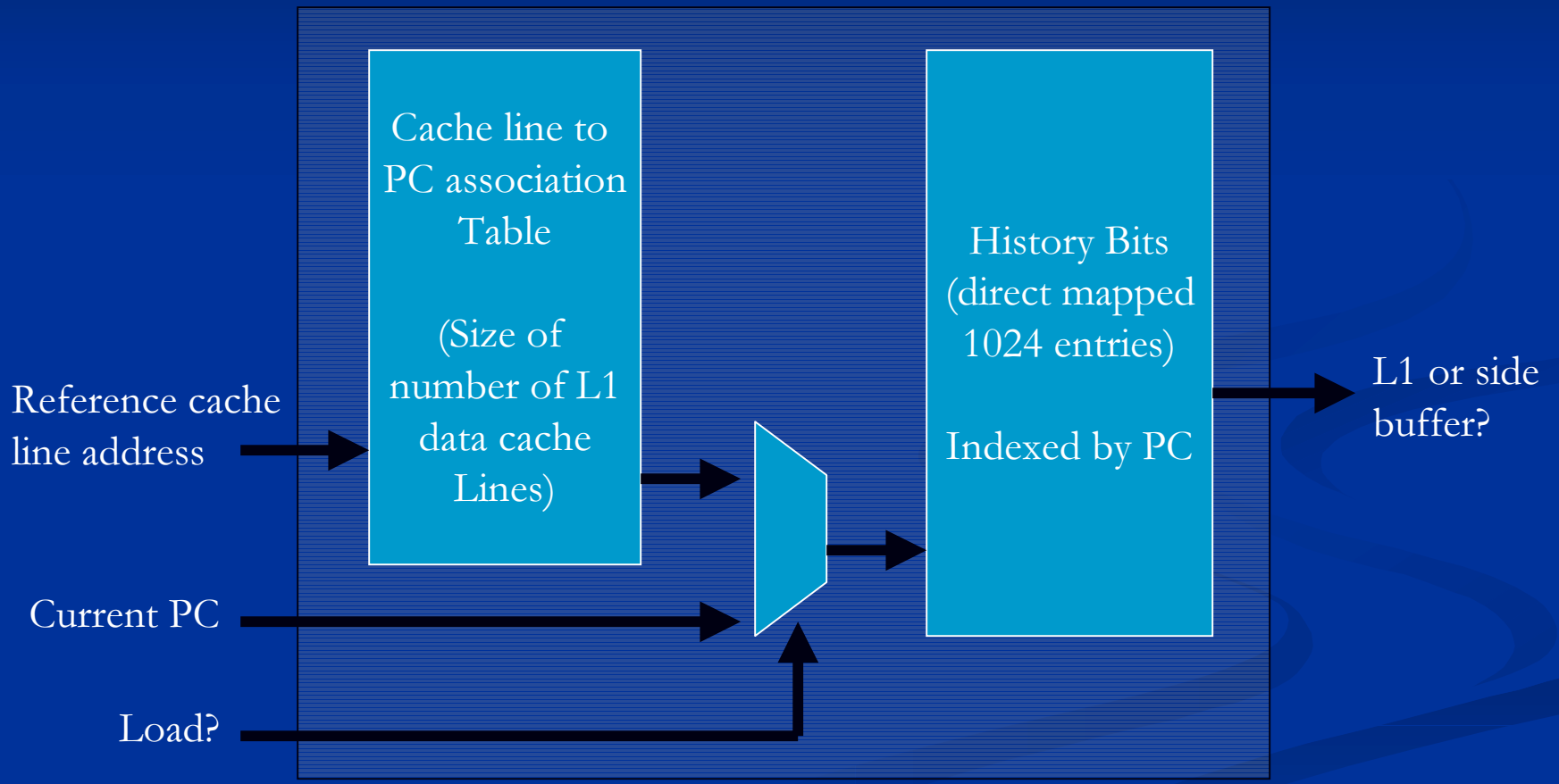
Architecture overview

- Read-only references
  - No writes within a span of + - 32 references
  - Repetitive, at least 3 loads from that line
- Load history table
  - Simple finite state machine
  - Predict if a load is part of read-only reference pattern
- Side buffer
  - Captures read-only spatial locality
  - Simplicity based on reduced wire delay

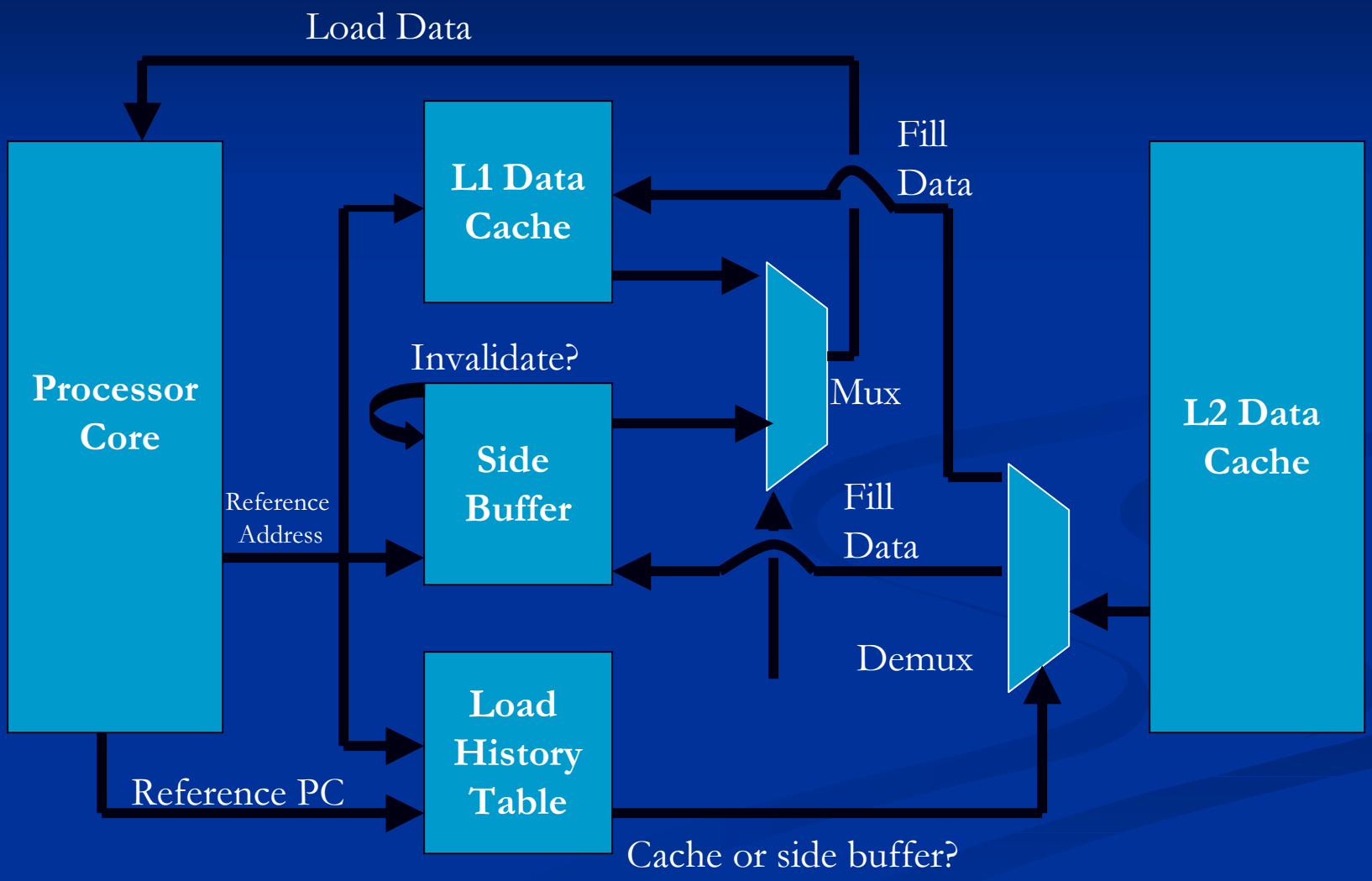
Architecture block diagram



Load history table



Architecture block diagram



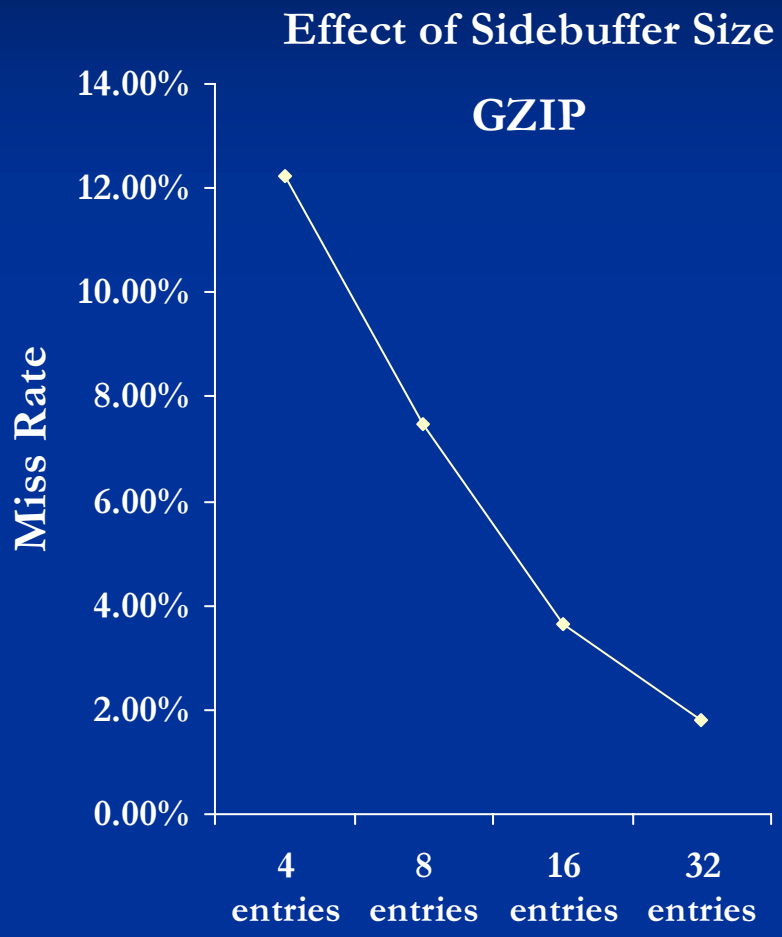
Experimental Methodology

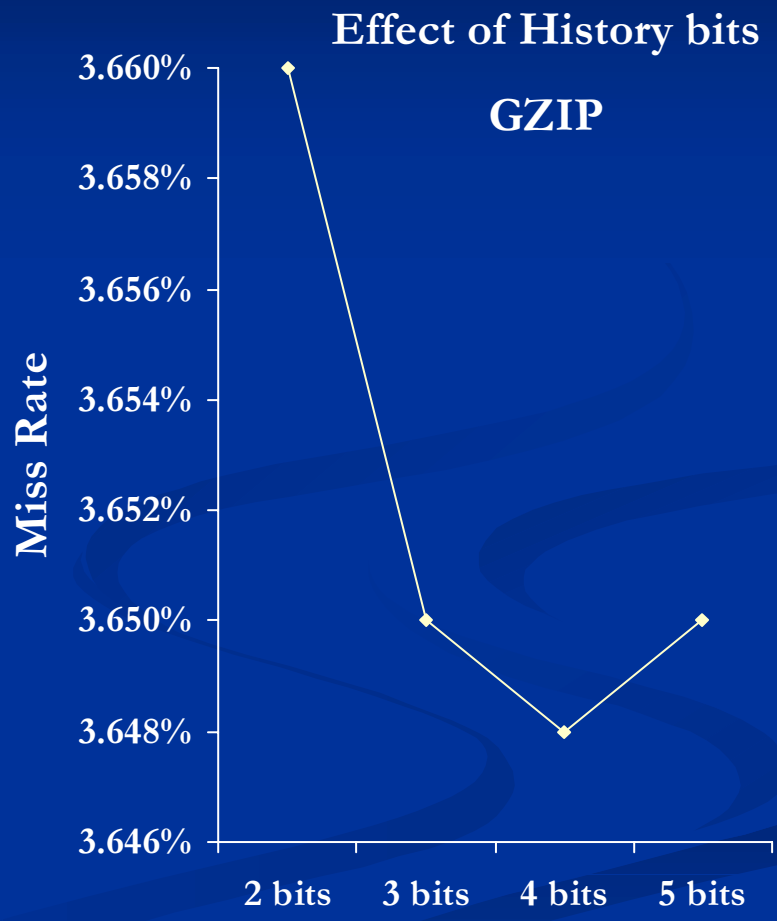
- First test to design the side buffer
  - Find out optimal configuration to minimize L1 hit rate
  - Compare vs. unmodified configurations
- Second test to compare against other architectures
  - Our configurations vs. state of the art
  - Itanium and Power 4
- Compared general purpose vs. media applications

Experimental Parameters

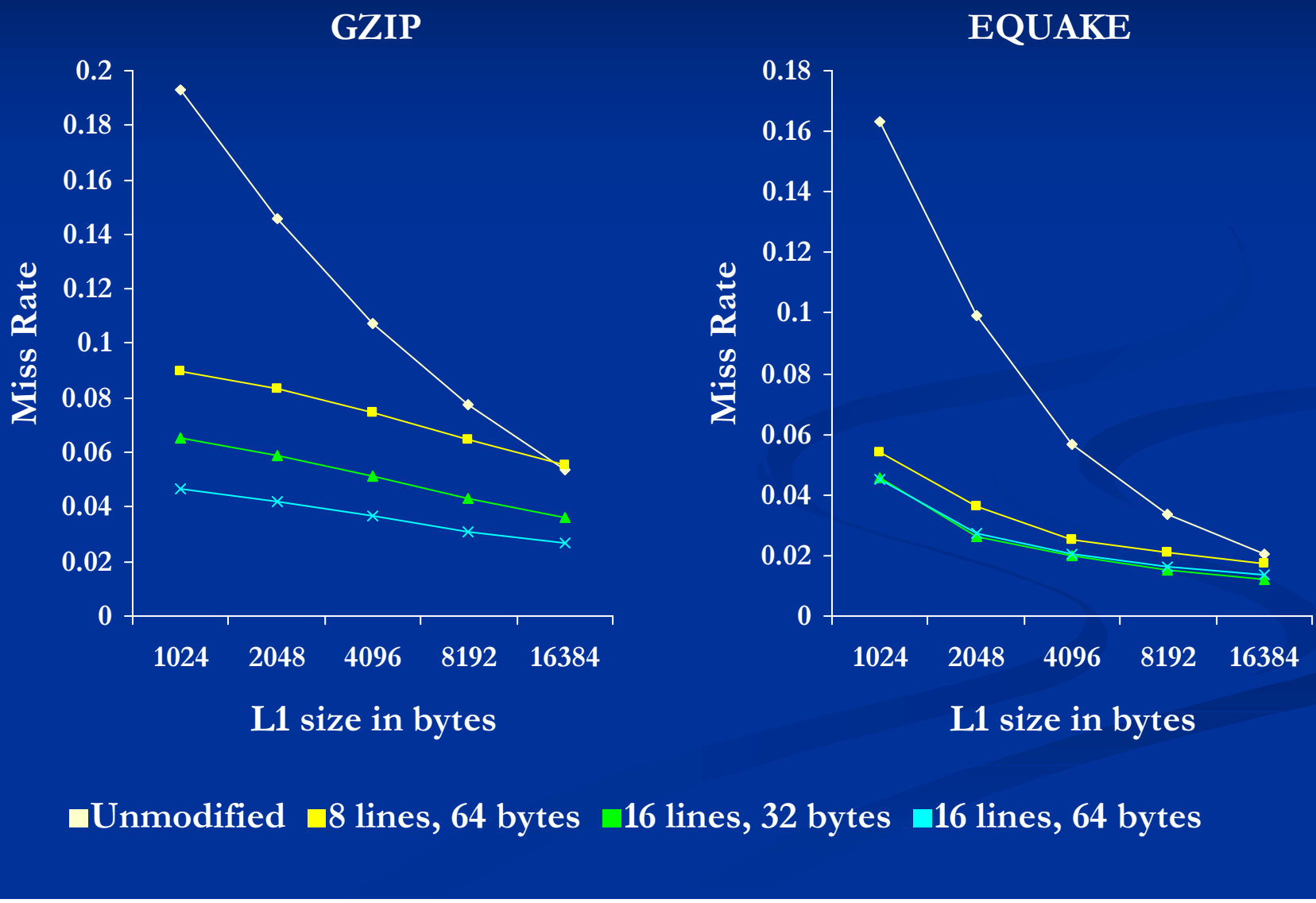
- SimpleScalar 3.0 simulator
  - Integrated our changes into sim-outorder
  - Ran SPEC 2000 reduced data sets
  - Ran MediaBench+
- Our optimal configuration
  - LHT 1024 entries with 4 bits, 512 byte structure
  - Side Buffer 16 lines, 32 byte block size

Impact on Miss Rate

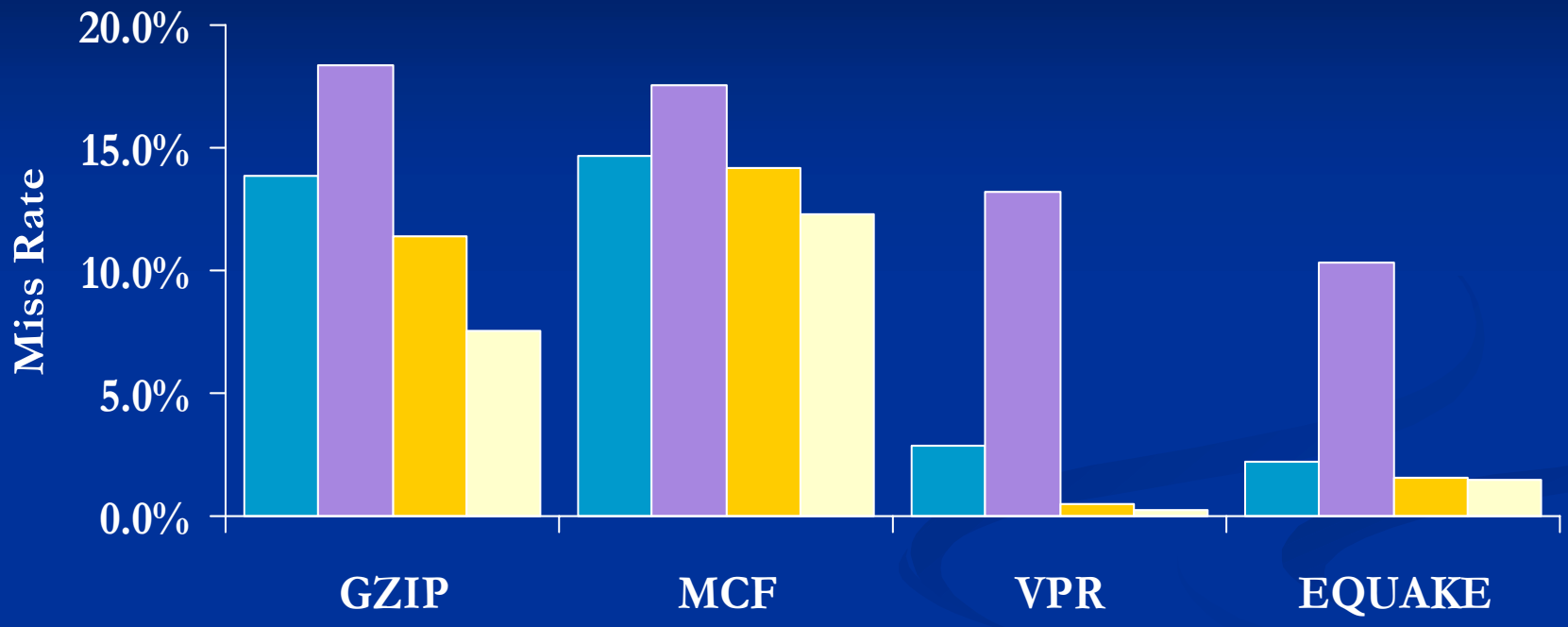




Impact on Miss Rate

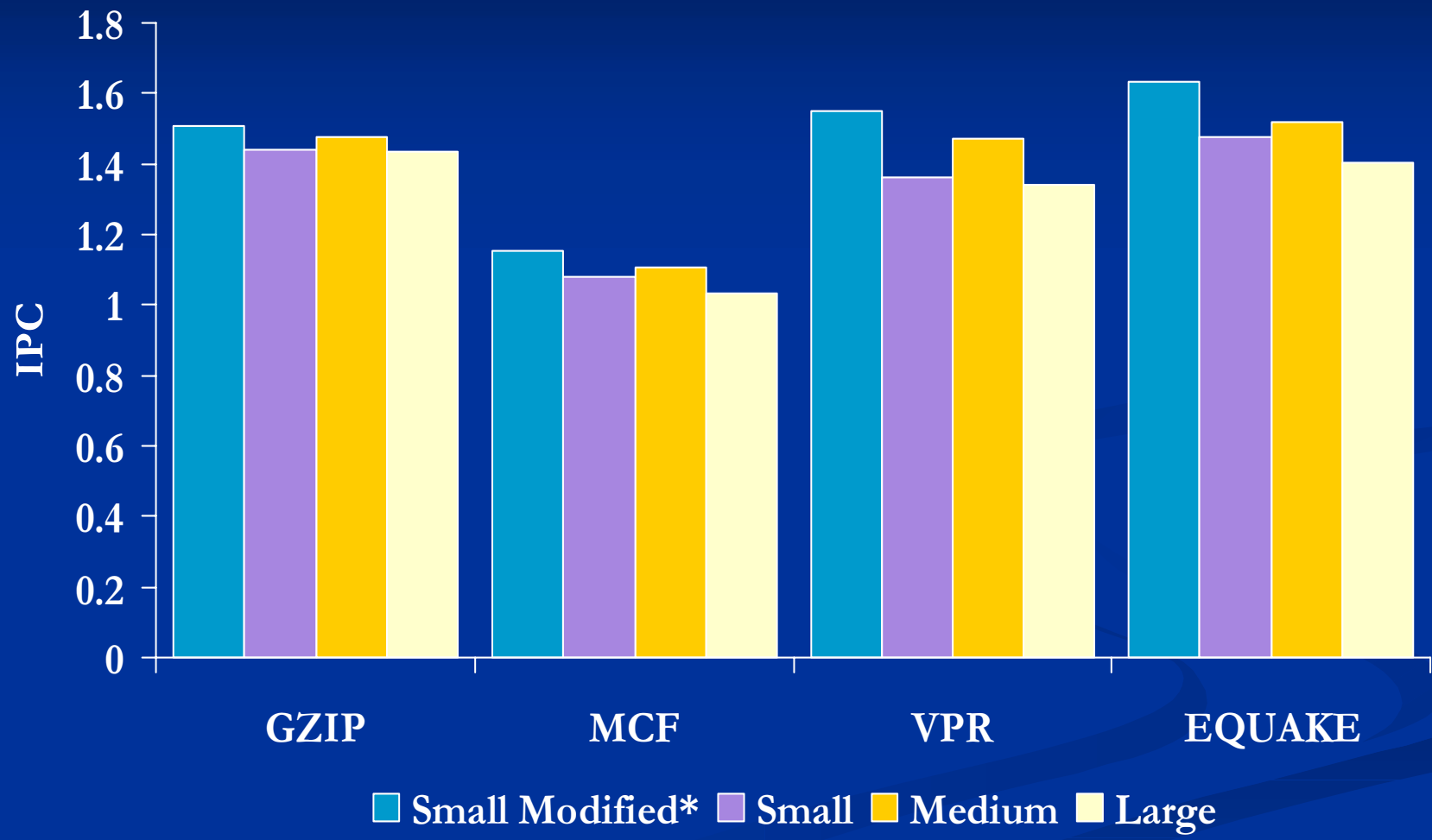


SPEC 2000 Results

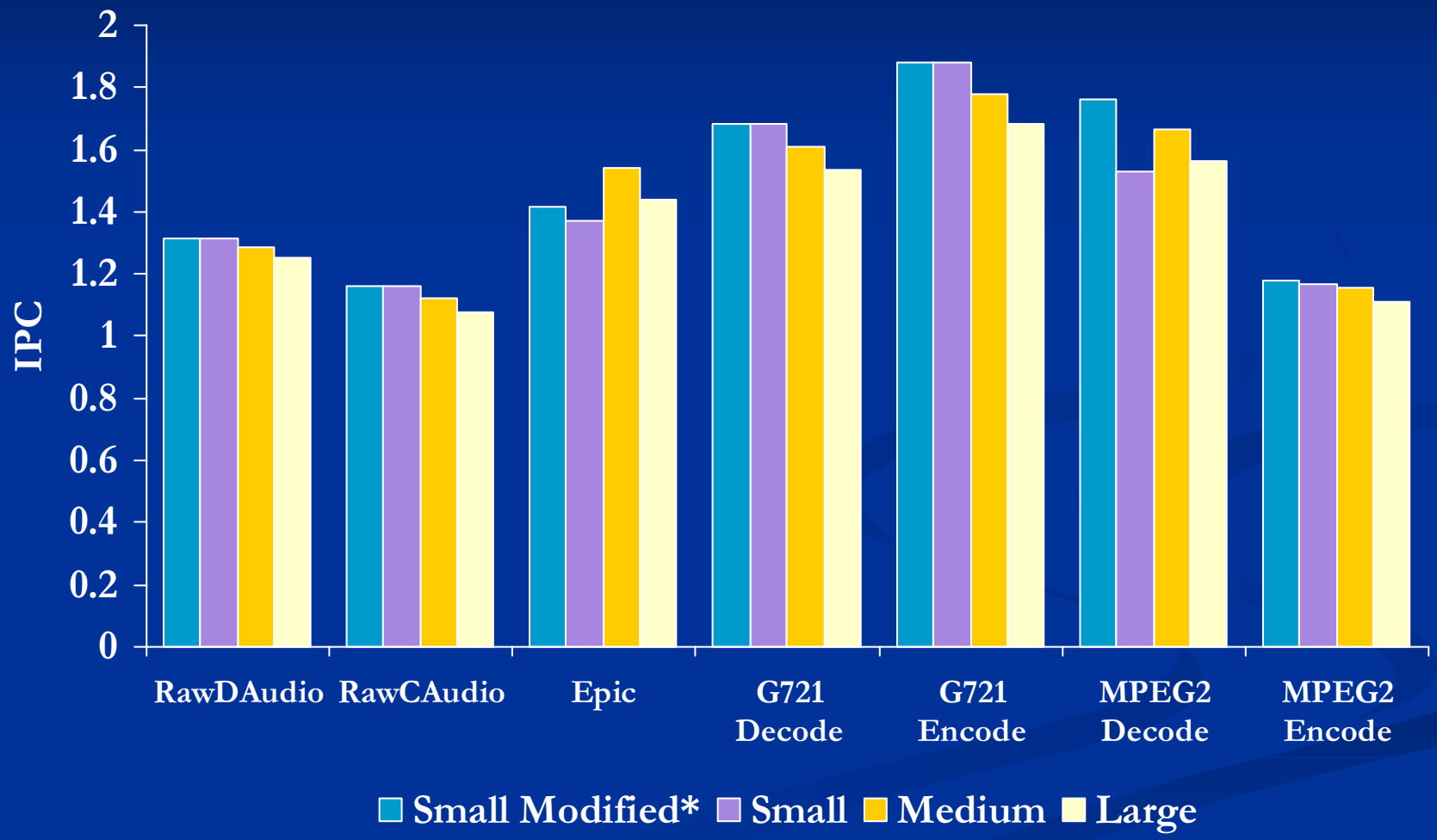


- Small Modified*, Small configuration + Side Buffer
- Small: 64 entries, 32 byte blocks, direct-mapped, 1 cycle latency
- Medium: 128 entries, 32 byte blocks, 4-way, 2 cycle latency
- Large: 512 entries, 32 byte blocks, 2-way, 3 cycle latency

SPEC 2000 Results



MediaBench+ Results



SPEC vs. MediaBench+

- We hypothesized that Media applications would get more benefit
  - Some don't benefit from our configuration
  - Pollution is not a problem with some media apps
- General purpose applications have more uniform benefits from our configuration
- Up to 16.7% increase in performance for SPEC and MediaBench applications

Our Results

- We perform better than larger caches
  - Increased hit rate cannot offset increased access time
  - Our scheme maintains high hit rates and low latency
- Scales better with increasing wire delay
  - Relative performance increases as caches shrink
  - More intelligent use of cache

Conclusion

- Benefits of adding the side buffer
  - Better use of transistors than a larger cache
  - Maintain low latency cache with a higher hit rate
  - Very little impact on critical path
- Simplicity of implementation
  - Simple design and few transistors
  - Similar to branch history tables

Questions and Answers

Additional MediaBench+ Results