ASCS ADAPTIVE & SECURE COMPUTING SYSTEMS LABORATORY

BOSTON UNIVERSITY

Adaptive Approximate Cache Architecture

Rashmi Agrawal, Michel A. Kinsy

Department of Electrical & Computer Engineering, Boston University

Summary

In this work, as a step towards designing Self-Aware Polymorphic Architecture (SAPA) systems, we investigate self-organizing memory structures and adaptive memory hierarchies, particularly in the caching subsystem.

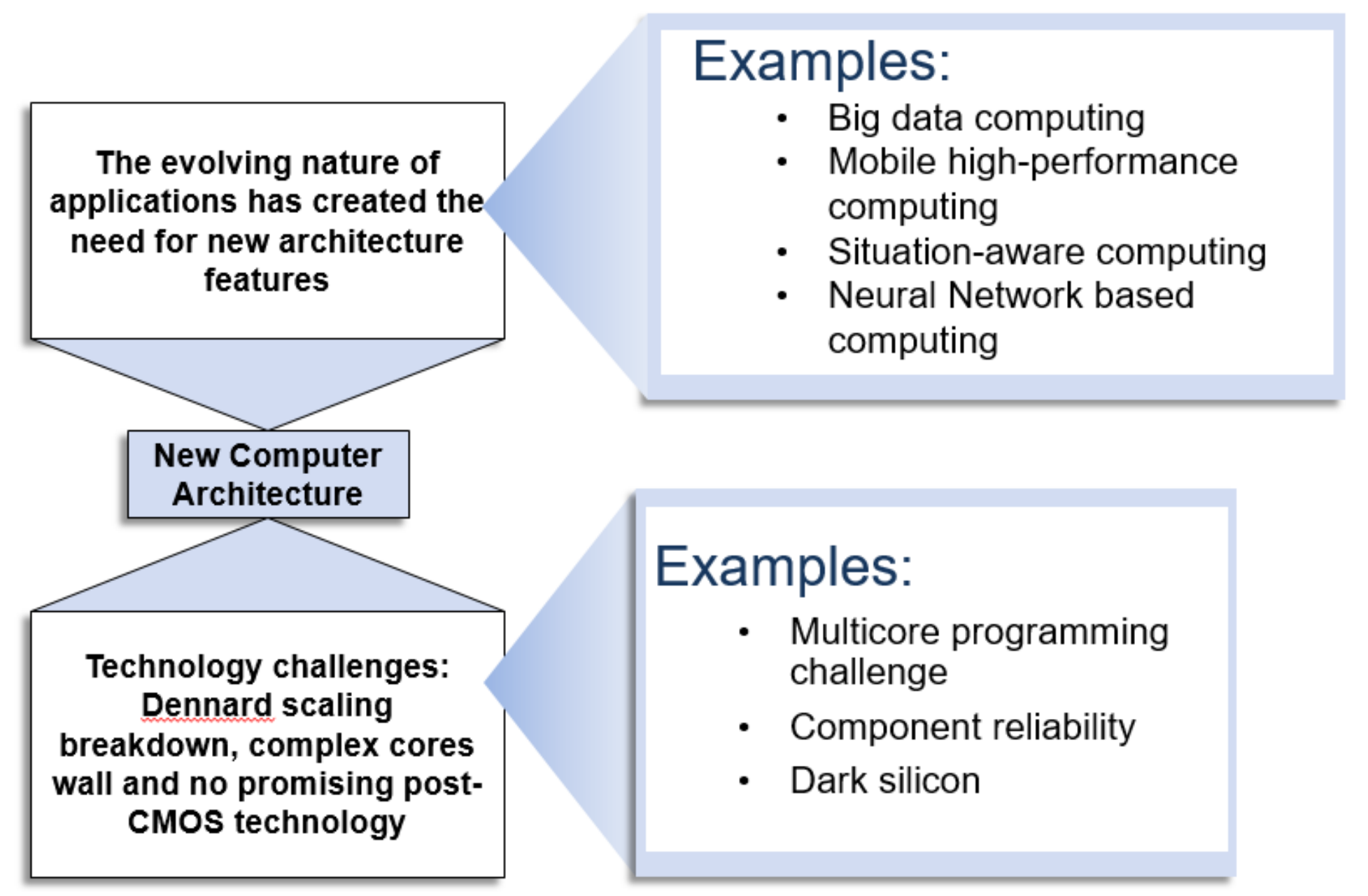
The concept introduced and explored in this work, Application-Aware Memory Organization Models (AMOM), provides a generalized framework for designing smart and reconfigurable memory subsystems.

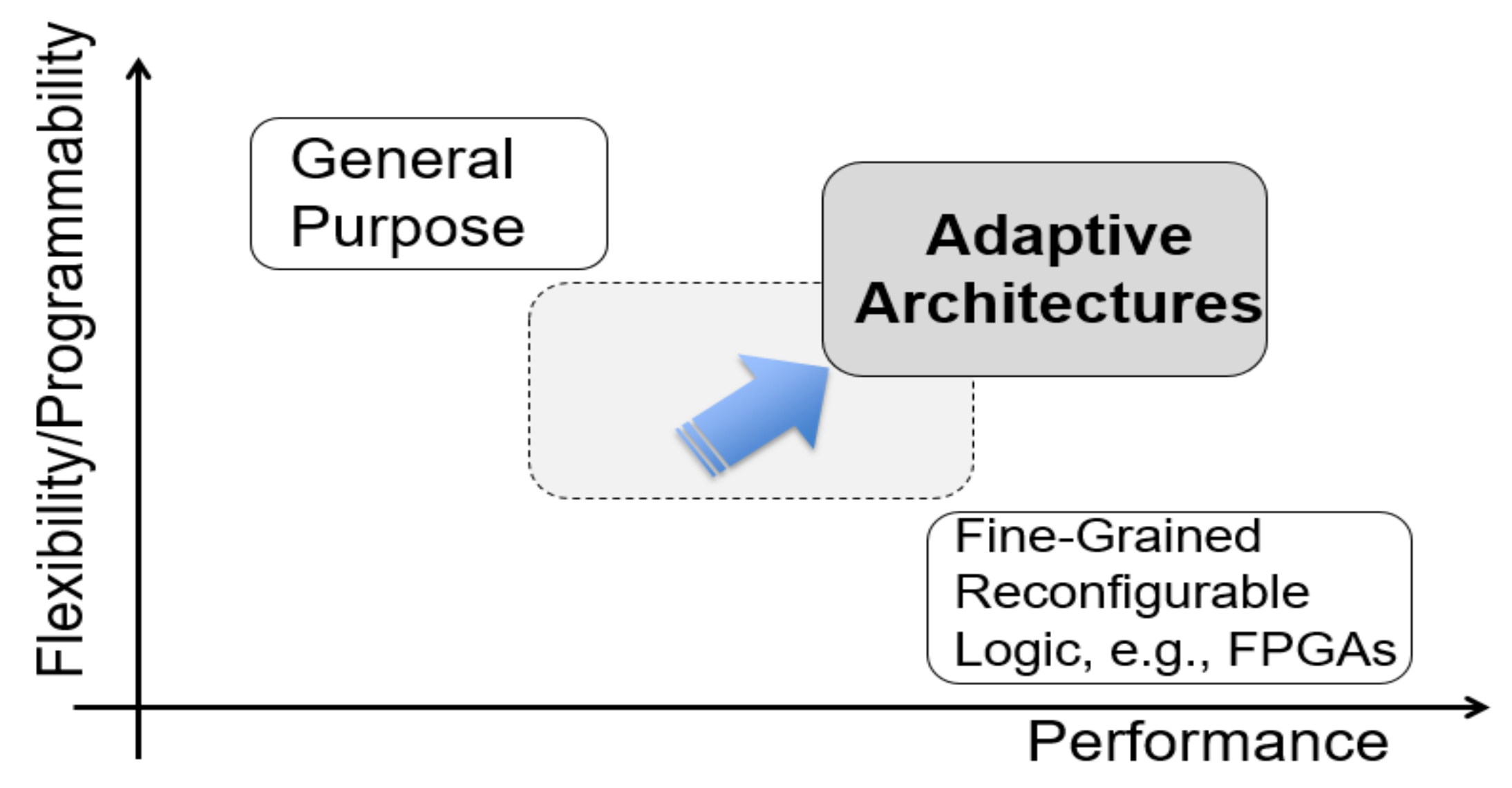
The proposed design uses hardware counters and other specialized hardware modules to learn the application's memory access pattern and estimate an optimal memory configuration, both at runtime.

Problem Statement

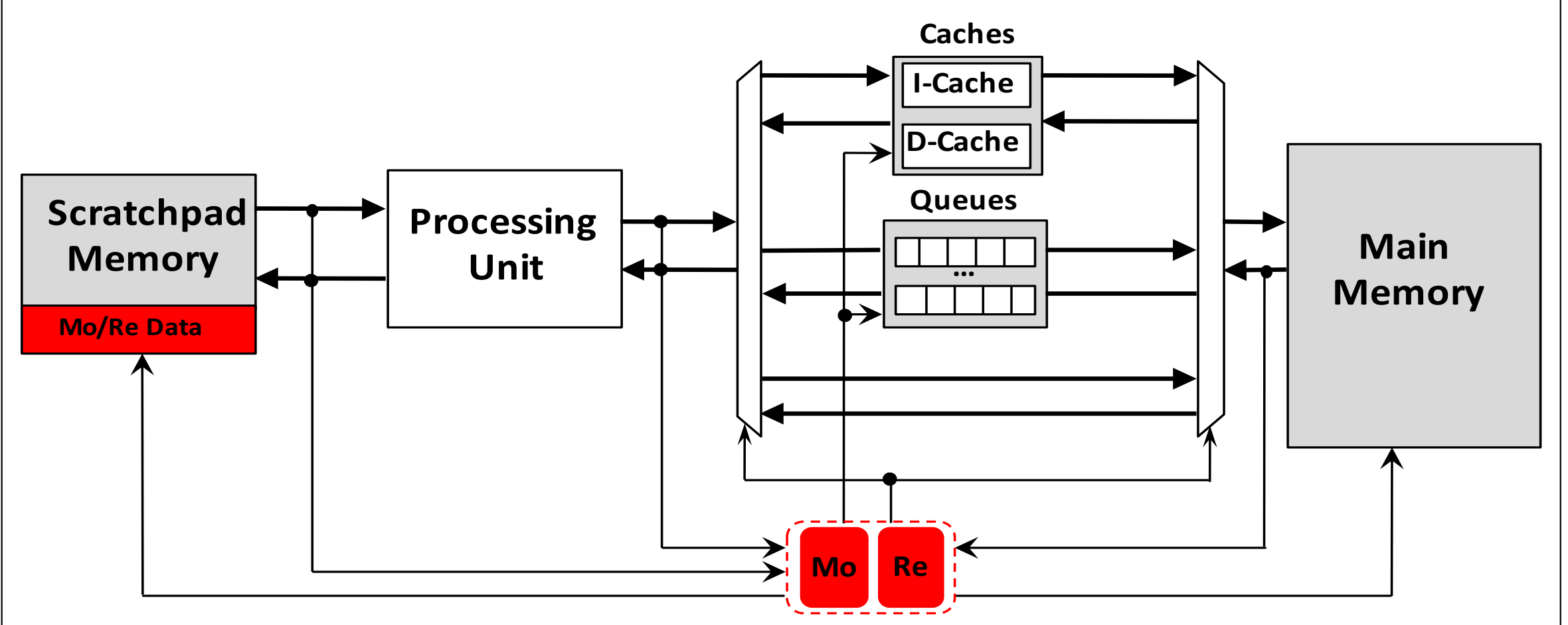
With the advent of technologies like mobile and cloud computing, context-aware computing, internet-of-things, autonomous car, computing systems must be redesigned to meet the performance requirements of these emerging applications.
Furthermore, the current multicore or manycore computer systems present application programmers with a great deal of challenges due to their ever-increasing complexity and heterogeneity. To make optimal use of the system components, programmers must first learn about system parameters and how to best leverage them for a given application.
A promising approach to address these computing challenges is via adaptive-approximate computer architectures with decision making capabilities for autonomous optimization and resource allocation based on the application under execution.

Introduction


The evolving nature of applications has created the need for new architecture features
Examples:
Big data computing
Mobile high-performance computing
Situation-aware computing
Neural Network based computing
New Computer Architecture
Technology challenges: Dennard scaling breakdown, complex cores wall and no promising post-CMOS technology
Examples:
Multicore programming challenge
Component reliability
Dark silicon

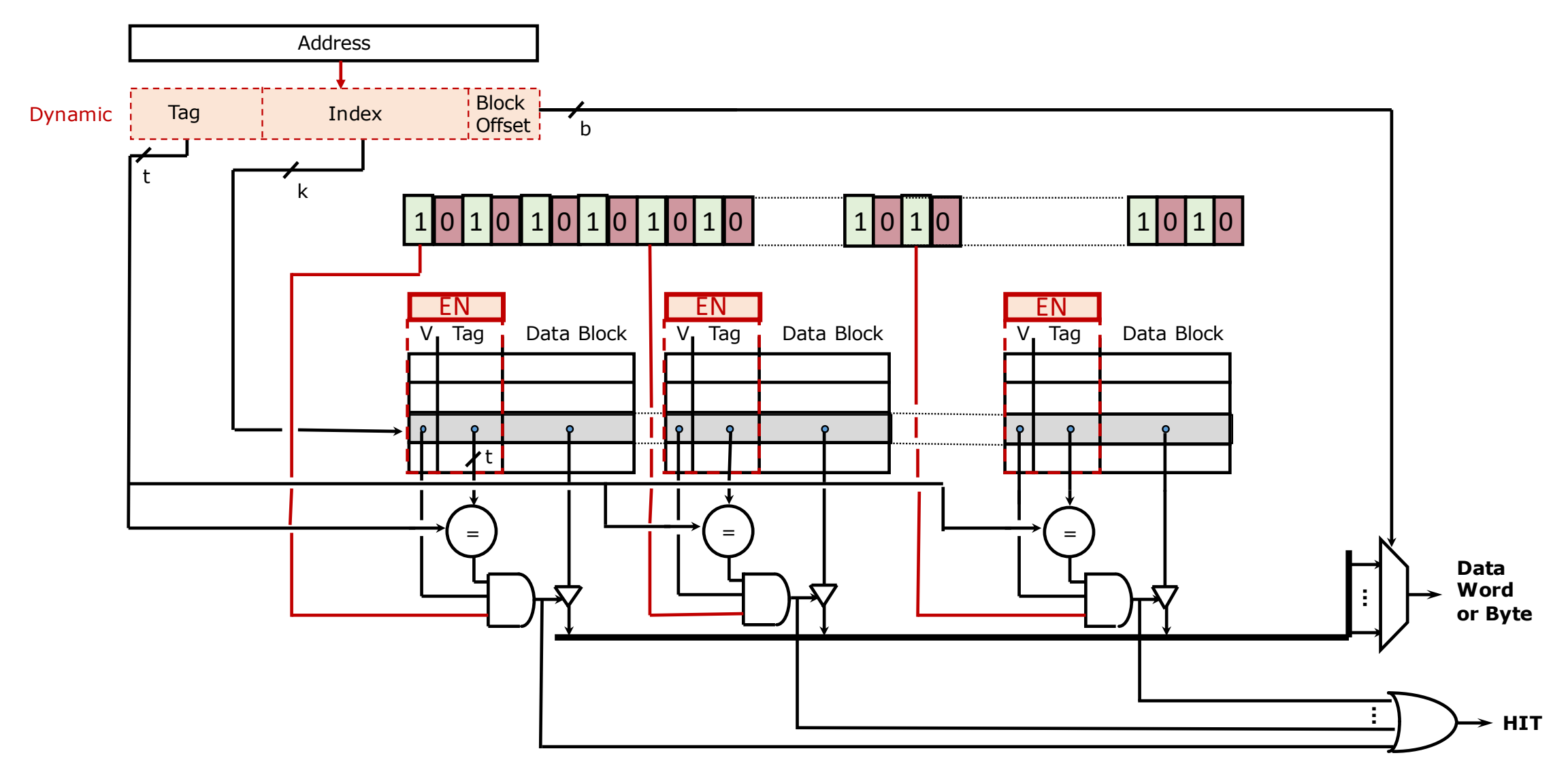

Flexibility/Programmability
Performance
General Purpose
Adaptive Architectures
Fine-Grained Reconfigurable Logic, e.g., FPGAs

Architecture Overview



- Self-Organization:
  - Dynamic cache line sizes and Dynamic associativity setting
- Hybrid cache/memory structures
- Multi-namespace memory
- Buffer queue structure

Self-Organized Cache Architecture Details



A logic is built around the conventional cache structure to enable dynamic reconfiguration
•Mask Register – An N-bit register to enable/disable a particular way and set cache line size. Helps in power-gating the cache blocks which are not in use.
•3-input AND gates instead of 2-input AND gates for detecting hits.
•2-bits for link - Helps in improving the access time by not matching the tags for same indexes.

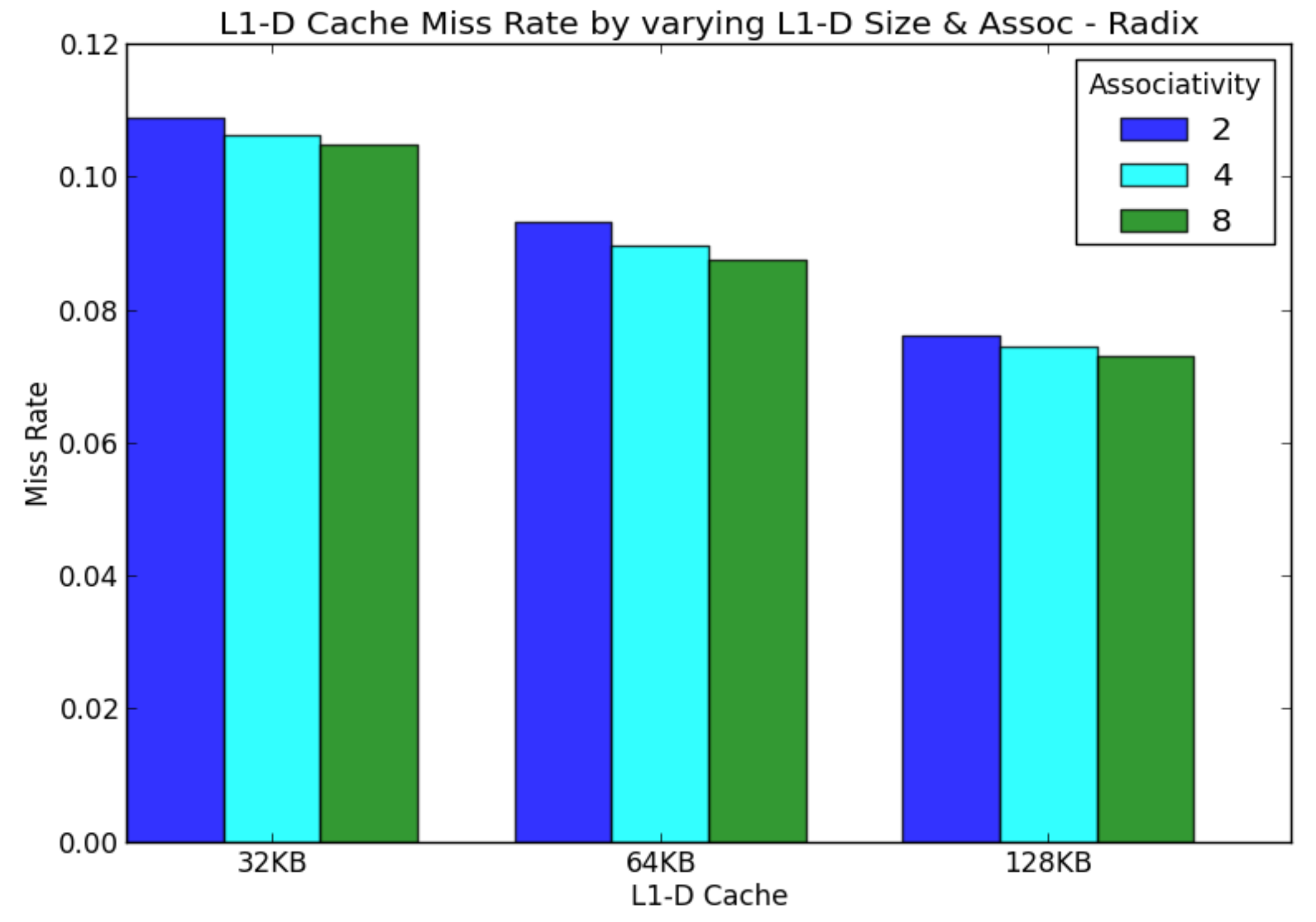
Evaluation Methodology

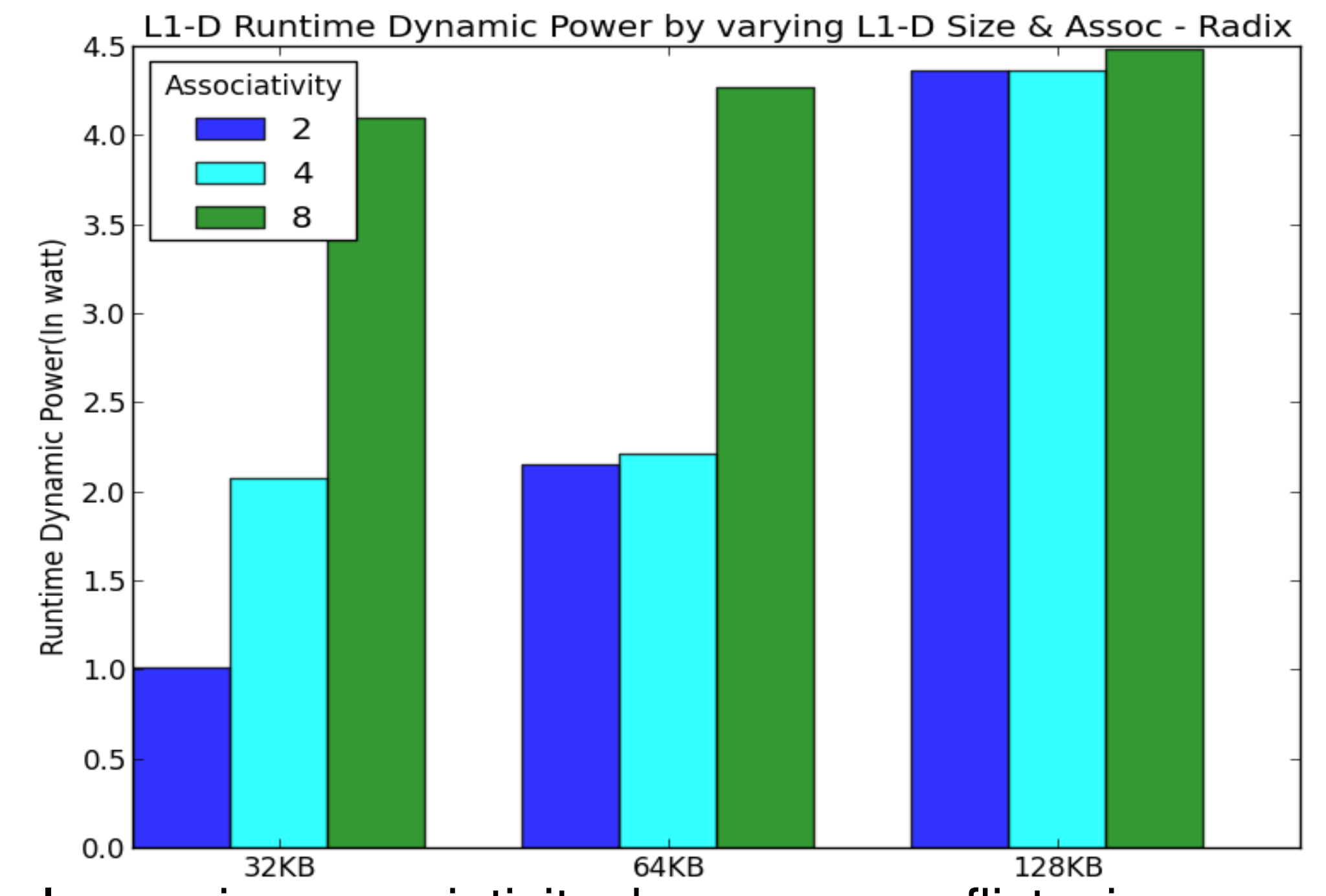
First, we study the effects of caching and cache structures on application runtime behaviors. Second, we explore the design space for cache structures that can adapt at runtime to application needs by changing the cache sizes, cache line widths and associativity.

•Simulators Used: Gem5, McPAT
•Benchmark: SPLASH-2
•ISA: Alpha & Simulation Mode: Full System

Parameter	Value
Cache line size	64B
L1-I Cache size	32 KB
L1-D Cache size	64 KB
L2 Cache size	2 MB
Associativity	2-way

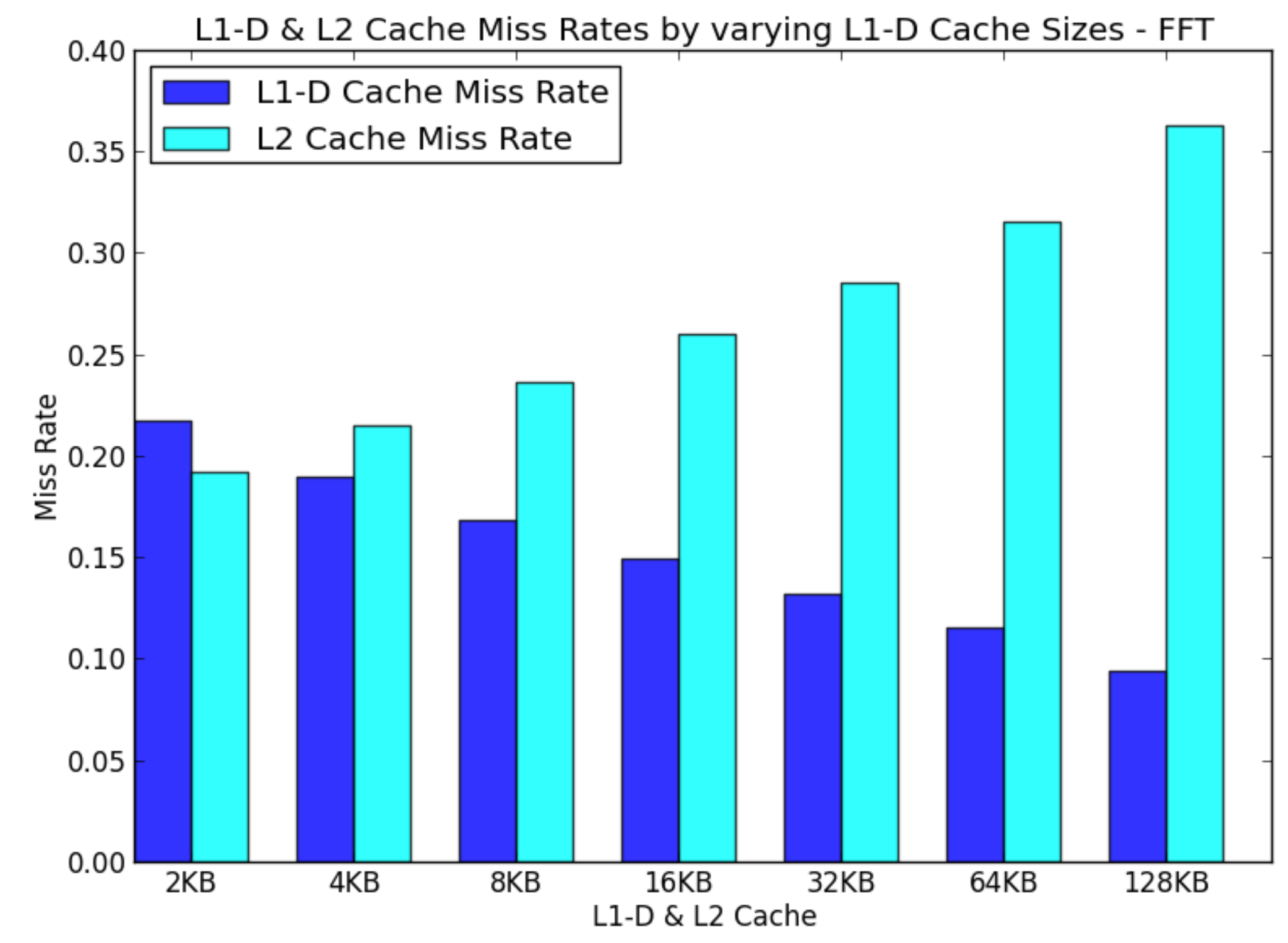
Performance Analysis – Varying Associativity

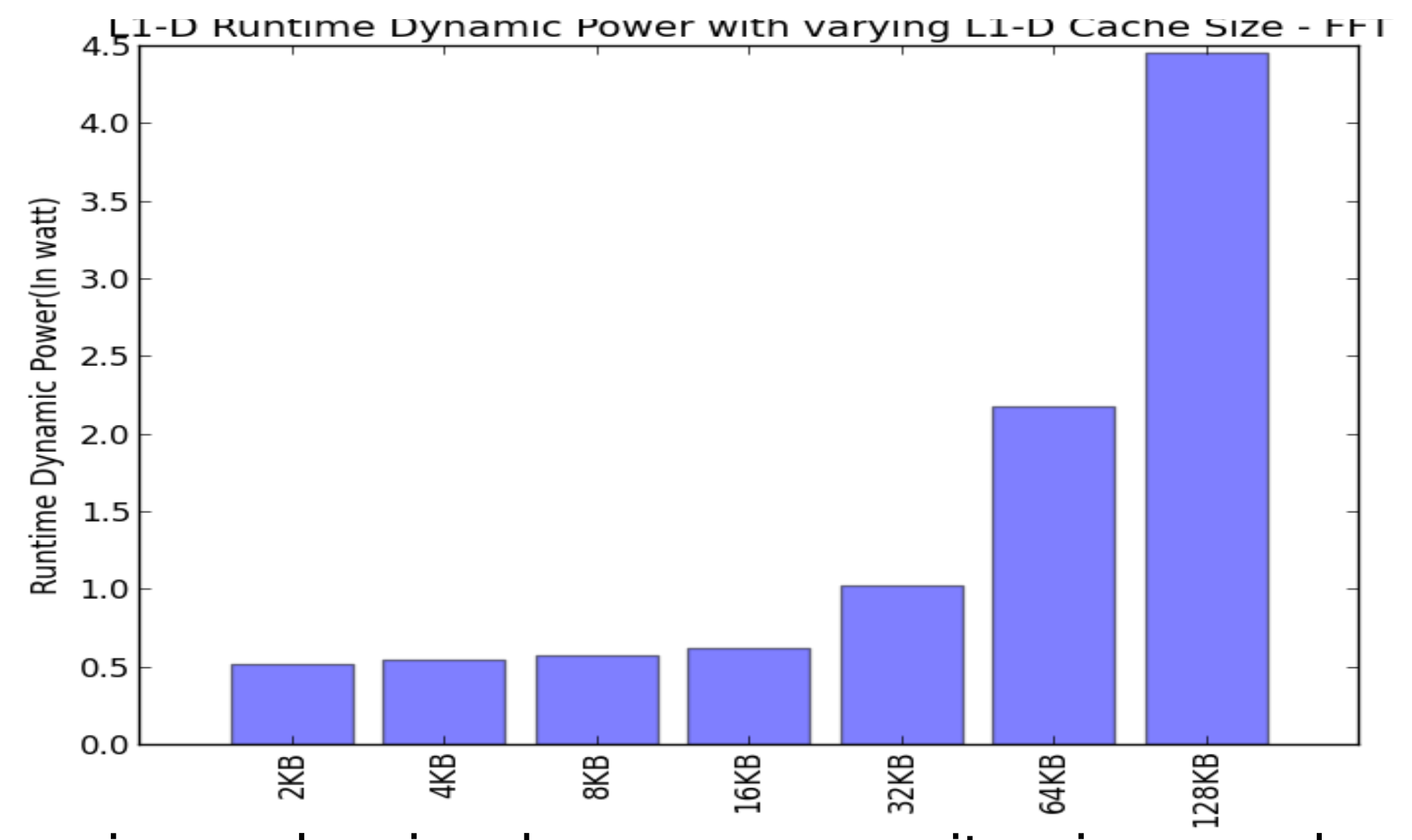




Increasing associativity decreases conflict misses and increases access time and power.

Performance Analysis – Varying Cache Size

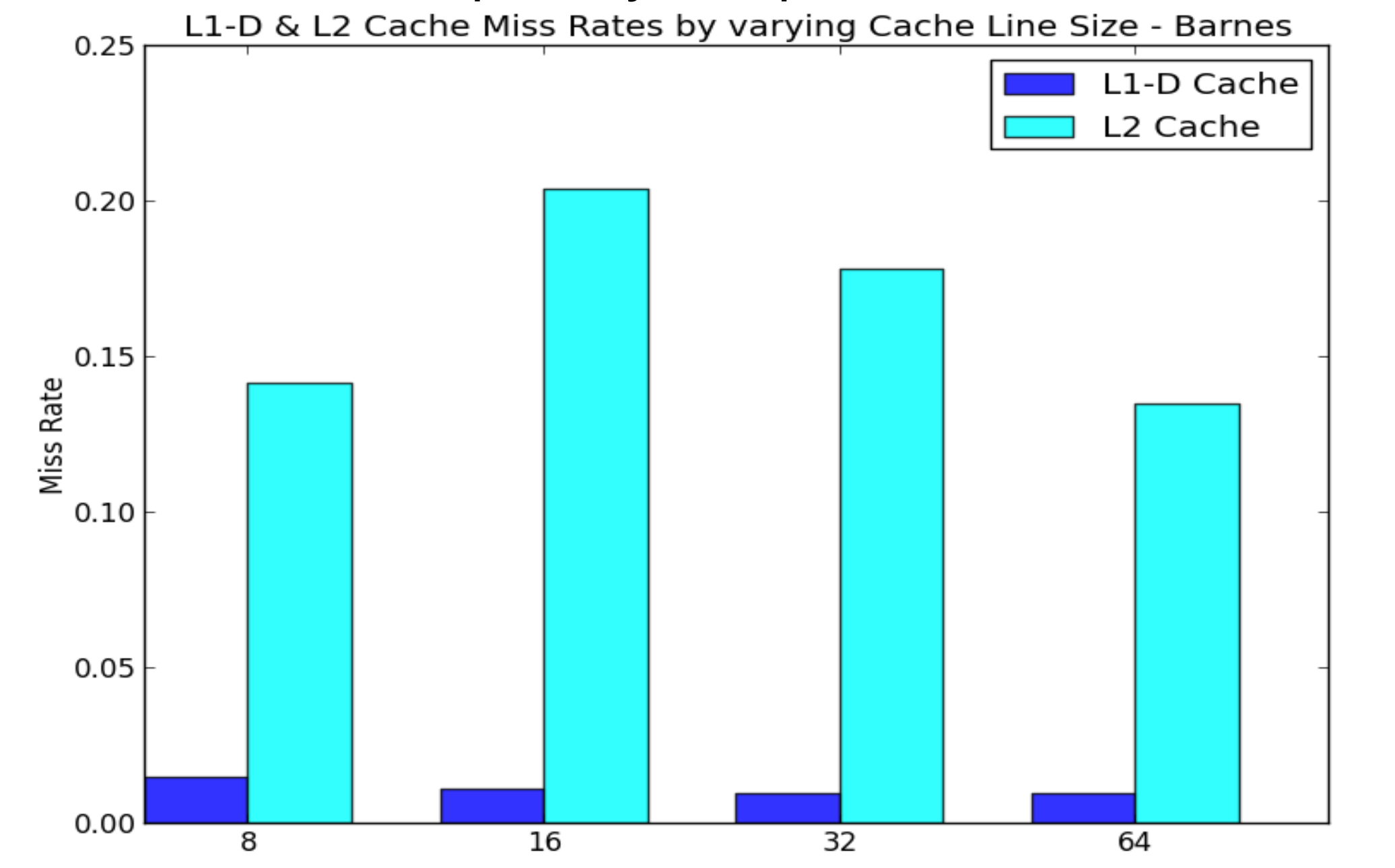


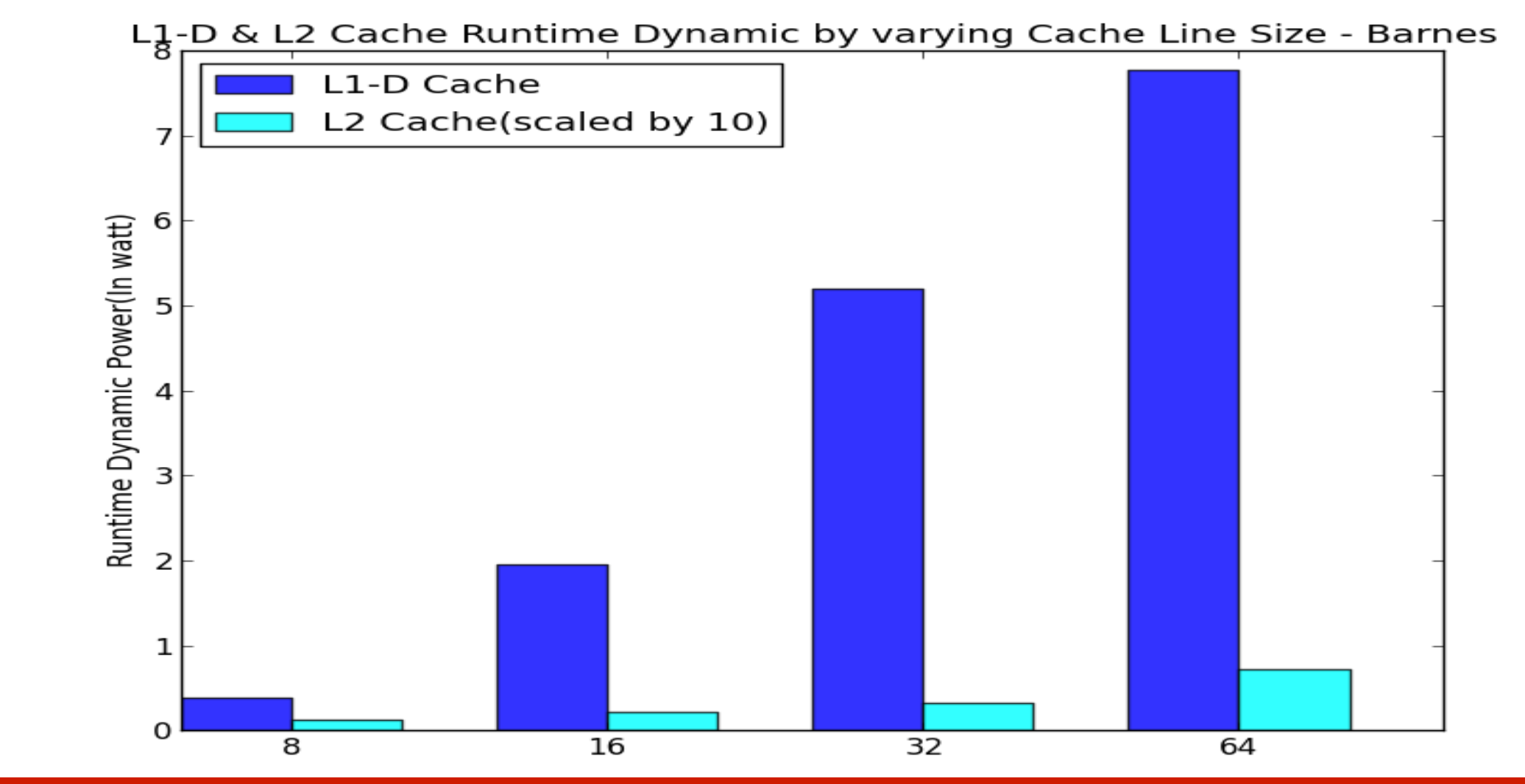


Increasing cache size decreases capacity misses and increases access time and power.

Performance Analysis – Varying Cache Line Size

Increasing cache line size decreases compulsory misses and increases miss penalty and power.





Key Takeways

To achieve optimum energy efficiency, cache parameters should be able to adapt themselves at run-time in response to the changing requirements of the running applications. These adaptations can be based on monitoring the miss rates and setting up the appropriate power envelopes.

Conclusion

For execution context adaptation, an architecture requires:
•The ability to modify hardware parameters dynamically;
•The ability to monitor performance as a function of program execution and collect statistics.
In this work, we show a low-hardware overhead design for an adaptive cache architecture.

References

[1] R. Bitirgen, E. Ipek, and J. F. Martinez, "Coordinated management of multiple interacting resources in chip multiprocessors: A machine learning approach," in Proceedings of the 41st Annual IEEE/ACM International Symposium on Microarchitecture, ser. MICRO 41. Washington, DC, USA: IEEE Computer Society, 2008, pp. 318–329.

[2] A. V. Veidenbaum, W. Tang, R. Gupta, A. Nicolau, and X. Ji, "Adapting cache line size to application behavior," in Proceedings of the 13th International Conference on Supercomputing, ser. ICS '99. New York, NY, USA: ACM, 1999, pp. 145–154.

[3] M. K. Qureshi, A. Jaleel, Y. N. Patt, S. C. Steely, and J. Emer, "Adaptive insertion policies for high performance caching," in Proceedings of the 34th Annual International Symposium on Computer Architecture, ser. ISCA '07. New York, NY, USA: ACM, 2007, pp. 381–391.

[4] A. Jaleel, K. B. Theobald, S. C. Steely, Jr., and J. Emer, "High performance cache replacement using re-reference interval prediction (rrip)," SIGARCH Comput. Archit. News, vol. 38, no. 3, pp. 60–71, June 2010.

[5] D. A. Jim´enez and C. Lin, "Neural methods for dynamic branch prediction," ACM Trans. Comput. Syst., vol. 20, no. 4, pp. 369–397, Nov. 2002.