L-CBF: A Low-Power, Fast Counting Bloom Filter Architecture

Elham Safi, Andreas Moshovos, and Andreas Veneris
University of Toronto
{elham, moshovos, veneris}@eecg.utoronto.ca

Abstract

We study the energy, latency and area characteristics of two Counting Bloom Filter *implementations using a commercial 0.13µm technology and full custom layouts. The first implementation, S-CBF, uses an SRAM array of counts and a shared counter. The second, L-CBF, utilizes an array of up/down linear feedback shift registers. Circuit level simulations demonstrate that for a 1K-entry CBF with a 15-bit count per entry, L-CBF is 3.7 or 1.6 times faster than the S-CBF depending on the operation. The L-CBF requires 2.3 or 1.4 times less energy per operation compared to the S-CBF. However, the L-CBF requires 3.2 times more area. We demonstrate that for one application of CBFs (early hit/miss detection for L1 caches [13] for an aggressive dynamically-scheduled superscalar processor) the energy consumed by the L-CBF is 60% of the energy consumed by the S-CBF for most of the SPEC CPU 2000 benchmarks.*

1 Introduction

An increasing number of architectural techniques rely on hardware *counting bloom filters* (CBFs) to improve upon the power, latency and complexity of various key processor structures. For example, CBFs have been used to improve the scalability of load/store scheduling queues [11], to reduce replays by assisting in early hit/miss determination at the L1 data cache [13], and to improve performance and power in snoop-coherent multiprocessor or multicore systems [9,10]. In these proposals CBFs are used to avoid accessing much larger and thus much slower and power-hungry content addressable memories [11], or cache tag arrays [9,10,13], or to avoid broadcasts over the interconnection network in multiprocessor systems [9].

In all aforementioned applications, the CBF is used to improve the energy and latency of membership tests (e.g., whether a memory block is currently cached). It does so by providing a definite answer for *most* but not *all* tests. Thus, the CBF does not replace the underlying conventional mechanism (e.g., cache tags). Instead, the CBF dynamically bypasses the conventional mechanism as frequently as possible. Accordingly, the benefits obtained through the use of a CBF depend on how frequently it can be utilized and on the CBF's energy and latency characteristics. The more tests are serviced by the CBF alone and the lower the power and latency of the CBF the higher the benefits.

In this work we are concerned with implementations of CBFs that improve on energy and latency. Conceptually, a CBF is an array of *counts* for which three operations are defined: increment by one, decrement by one, and test if zero. We will refer to the first two operations as *updates* and to the third as a *probe*. Previous work assumed a straightforward SRAM-based implementation which we will refer to as S-CBF (see Section 2.1). In this work we investigate the energy, latency and area of this implementation using a commercial 0.13µm CMOS technology. However, the key contribution of this work is L-CBF, a novel implementation of CBFs that relies on up/down linear feedback shift registers (LFSRs). We demonstrate that this implementation is significantly faster and it requires significantly less energy than the previously assumed SRAM-based implementation. Using architecture level simulation of most of the SPEC CPU 2000 programs we demonstrate that L-CBF can significantly reduce power for the early detection of L1 data cache misses [13].

In more detail, the contributions of this work are as follows:

- L-CBF, an LFSR-based counting bloom filter architecture is proposed.
- The energy, latency and area of L-CBF and S-CBF are compared using their circuit level and full-custom layouts in 0.13 fabrication technology.
- The relative energy dissipation of L-CBF and S-CBF is compared for most SPEC CPU 2000 programs and for the early detection of L1 data cache misses [13].

To the best of our knowledge this is the first work that investigates the energy, latency and area of full-custom implementations of CBFs using a commercial CMOS technology. The idea of using LFSRs for the design of CBF has been proposed before but no design or evaluation of its characteristics were reported [9].

The rest of this paper is organized as follows. In Section 2 we review CBFs and describe the S-CBF implementation in Section 2.1. In Section 2.2 we present the L-CBF design. In Section 3 we discuss our experimental results. We conclude in Section 4.

2 Counting Bloom Filters

Without the loss of generality, we restrict our attention to using CBFs for the early detection of L1 data cache misses [13]. The concepts and implementations we present are directly applicable to other CBF applications.

In the application we consider, the CBF determines whether a particular block of memory is currently cached in the L1 data cache. Given a block address *A* the CBF reports whether *A* appears in any of the tags of the data cache. The CBF provides two possible answers: (1) *"no,* *the block is definitely not cached"*, and (2) *"maybe the block is cached"*. In the first case, we determine that *A* is not cached and hence this access will miss. Provided that the CBF is much faster and dissipates much less power than the L1 tag arrays, we manage to obtain the desired answer much faster and to save power. In the second case, the CBF cannot provide a definite answer and thus we do have to access the L1 tags. In this case, we incur a power penalty since we had to also access the CBF. We may also incur a latency penalty if the CBF and the L1 tag accesses are serialized (we may avoid this latency penalty if we probe the CBF and the L1 tags in parallel, in which case power benefits will be possible only if we can terminate the L1 tag access in progress when the CBF provides a definite answer).

As shown in Figure 1, the CBF can be thought of as an array of counts that is indexed via a hash function of the address *A* and where three operations are defined: (1) *increment* count, (2) *decrement* count, and (3) test if count is zero, or *probe*. The first two operations increment or decrement the corresponding count by one, while the probe operation tests if the count is zero and returns true or false (single bit output). Simply using a portion of the address and not a more elaborate hash function has been shown to work well [9,13].

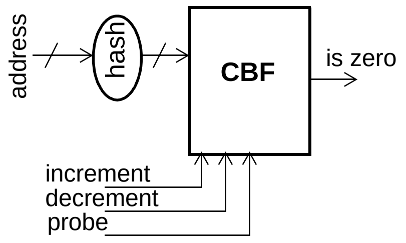


Figure 1: *A CBF for cache block address test membership.*

Initially, all CBF counts are zero and the L1 is empty. When a block is allocated into the L1, the corresponding CBF entry is incremented by one. When a block is evicted from the L1, the corresponding CBF entry is decremented by one. To test whether *A* currently exists in the L1, we inspect the corresponding CBF count. If the count is zero then *A* is definitely not in the L1 since we would have incremented the count the moment it was cached. If the count is non-zero then *A* may be cached. Since many blocks can map onto the same CBF count, it is possible that some other cache block incremented the count[1]. Therefore, in this case we need to check the L1 tags to determine whether *A* is cached. It is for the latter reason that a CBF is an *imprecise* representation of the cached blocks as it represents a superset of the cached blocks.

[1] Ideally, a separate entry would exist for every possible block address A. However, this would result in a prohibitively large table (e.g., a table with 32 million entries for a processor with a 4Gbyte address space and 32-byte cache blocks) and would negate any benefits. Accordingly, a small table is used and addresses are hashed onto the table. Hence multiple addresses may map onto the same table entry.

A CBF is characterized by the number of entries it contains and the width of the count of each entry. Multiple CBFs with different hash functions can be used to improve accuracy [10,13]. Also, count values are bounded. Since the same count entry is incremented and decremented on a block's allocation and eviction respectively, a count can never become negative and can never exceed the number of the total cache blocks. Counting bloom filters also have applications in software [7].

2.1 S-CBF: SRAM-Based CBF

Previous work assumed a straightforward CBF implementation comprising an SRAM array to hold the counts, a shared up/down counter, a zero-comparator and a small controller [10]. This is shown on Figure 2. Updates are implemented as read-modify-write sequences as follows: (1) the count is read from the SRAM, (2) it is adjusted using the counter, and (3) it is written back to the SRAM. The probe operation is implemented as a read from the SRAM and then a comparison with zero using the zero-comparator. A small controller coordinates all actions necessary to implement these CBF operations.

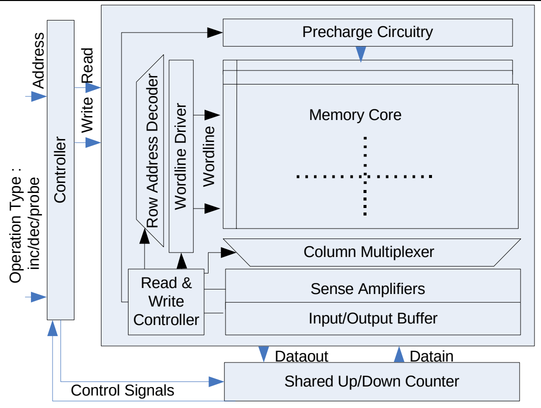


Figure 2: *S-CBF architecture: An SRAM holds the CBF counts and updates are implemented as read-modify-write sequences.*

An optimization was proposed to speedup probe operations and to reduce their power [10]. Specifically, an extra bit, *Z*, is added to each count. When the count is incremented from zero the Z is set to false and when the count is decremented to zero the Z is set to true. Probes can now simply inspect Z. The Z-bits can be implemented as a separate SRAM structure which is faster and requires much less power. We observe that this optimization can be applied to both the S-CBF and the L-CBF.

2.2 L-CBF

As we demonstrate in Section 3, much of the energy in S-CBF is consumed on the bitlines and wordlines.

Latency and energy suffers also because updates require two SRAM accesses per operation. Finally, the shared counter further increases energy and latency.

We could avoid accesses over long bitlines by building an array of up/down counters. Then there would be no need to read the value of each counter and updates would be localized. Unfortunately, up/down *arithmetic* counters require many gates and are slow, e.g., [12]. We make the following two observations: (1) the actual count sequence used in a CBF is not important, and (2) externally, we only care whether a count is “zero” or “non-zero”.

The L-CBF offers the benefits that are possible with an array of up/down counters while avoiding the overheads associated with using arithmetic counters. L-CBF uses *up/down LFSRs.* As we demonstrate in Section 3, L-CBF significantly reduces energy and latency compared to S-CBF albeit at the expense of increased area. However, this is a minor concern in modern processor designs for two reasons: (1) there is an abundance of resources, and (2) the CBF is tiny compared to most other processor structures (e.g., caches and branch predictors). It is unlikely that the same resources could improve performance if applied to other processor structures that are already much larger and optimized.

In the rest of this section, we review LFSRs, the construction of up/down, or reversible LFSR counters and finally present the organization of L-CBF.

2.2.1 Linear Feedback Shift Registers

In this section, we review LFSRs and the construction of up/down LFSR counters. An appropriately designed LFSR counter, or *maximum-length* LFSR of *n* bits sequences through 2^n-1 states. Without the loss of generality we restrict our attention to the Galois configuration of LFSRs [1]. Figure 3 shows a maximum-length LFSR of 8-bits. The LFSR comprises a shift register and a few XNOR gates. Each bit of the shift register is either shifted as-is to the next bit (no tap) or is XNORed with the output of bit 7 (tap). By appropriately selecting the tap locations it is always possible to build a maximum-length LFSR of any width that has either two or three taps [1,5]. Furthermore, ignoring wire length delays and the fan-out of the feedback line, the delay of the maximum-length LFSR is independent of its size [12]. As we show in Section 3.2, latency increases only slightly as the number of bits increases, primarily as a result of increased capacitance on the control lines.

The tap locations for a maximum-length LFSR can be represented as a primitive polynomial *g(x)*. Figure 3 shows an example of such a polynomial. In general, an LFSR can be expressed as:

$$g(x) = \left(\sum_{i=0}^{n} C_i \times X^i\right)(C_0 = C_n = 1)$$

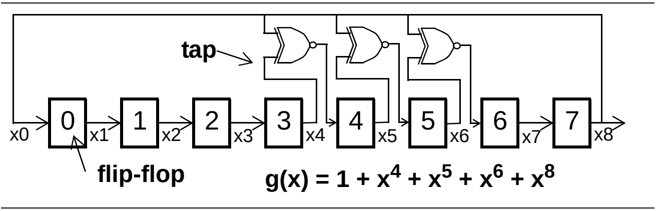
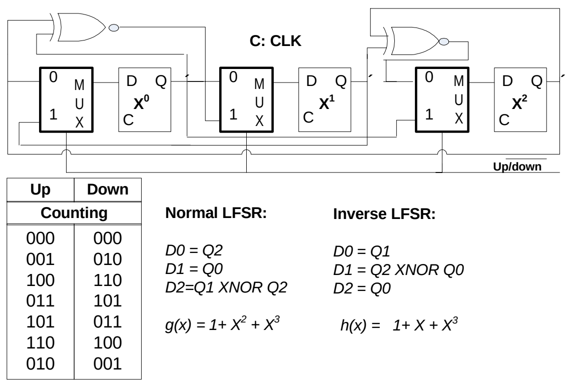


Figure 3: *A 8-bit maximum-length LFSR (sequence length = 255)*

where X^i corresponds to the output of the ith bit of the shift register and where the constants C_i are either *0* (no tap) or *1* (tap). This formula represents a uni-directional (“up”) LFSR. If the primitive polynomial for a maximum-length n-bit LFSR is *g(x)* (as defined by the preceding formula), then the primitive polynomial *h(x)* of an LFSR that generates the reverse sequence is [5]:

$$h(x) = \left(\sum_{i=0}^{n} C_i \times X^{n-i}\right)(C_0 = C_n = 1)$$

The superposition of the two LFSRs (the original and its reverse) forms a reversible, or an up/down LFSR. This reversible LFSR can be implemented using the same shift register, a 2-to-1 multiplexer per bit to control the direction of the shift and several XNOR gates one per tap. Figure 4 shows the construction of a 3-bit up/down LFSR. In general, it is possible to construct a maximum-length up/down LFSR of any width with either two or six XNOR gates (i.e., four or eight taps).



Up	Down
Counting	
000	000
001	010
100	110
011	101
101	011
110	100
010	001

Normal LFSR:

D0 = Q2
D1 = Q0
D2=Q1 XNOR Q2

$g(x) = 1 + X^2 + X^3$

Inverse LFSR:

D0 = Q1
D1 = Q2 XNOR Q0
D2 = Q0

$h(x) = 1 + X + X^3$

Figure 4: *A 3-bit maximum-length up/down LFSR.*

2.2.2 L-CBF Implementation

Figure 5 shows the high level organization of the L-CBF. The L-CBF includes a hierarchical decoder and several partitions each containing an array of up/down LFSRs. In each partition there are local zero detectors per LFSR counter. A hierarchical mux collects these local “is-zero” signals and provides the single “is-zero” output. The L-CBF accepts three inputs and produces a single output “is-zero”. The 2-bit input operation encodes any of the three possible operations or none. The address wires are used to specify the address in question and the reset signal is used once to initialize all LFSRs to the "zero" state. An external clock source is needed. The LFSRs use two non-overlapping phases which are generated internally using this external clock signal.

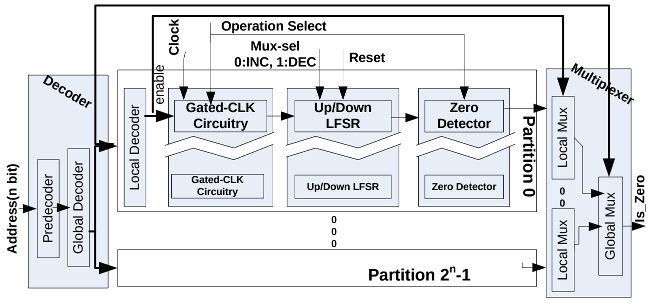


Figure 5: *L-CBF architecture.*

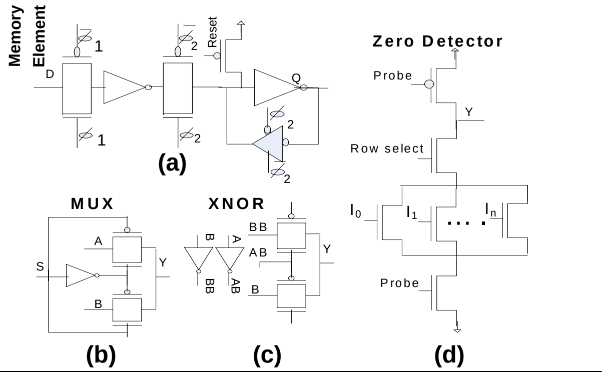


Figure 6: *The cells used to implement each up/down LFSR: (a) the two-phase flip-flop (b) the 2-to-1 mux cell (c) XNOR gate, (one cell per bit of the LFSR) (d) a bit-slice of the embedded zero detector*

We use hierarchical decoders for the address lines to minimize energy and delay [2]. The decoder consists of a predecoding stage, a global decoder that selects the appropriate partition and a set of local decoders, one per partition. Each partition contains an array of up/down LFSRs. Each row in each partition contains an up/down LFSR and a zero detector. Finally, we use a hierarchical multiplexer for selecting the appropriate zero-detector output for the "is-zero" operation. Figure 6 shows the custom cells that we used to implement each LFSR. Shown are the flip-flop we utilized for the shift register cells, the multiplexers used to control the direction of change ("up"/"down"), the XNOR gate of the up/down LFSR, and a bit-slice of the zero-detector. Due to space limitations we do not provide additional details on the L-CBF implementation.

3 Experimental Results

In this section, we compare the energy consumption, delay, and area of the S-CBF and L-CBF implementations. We first compare the designs on a per operation basis and then report energy savings with L-CBF over S-CBF for L1 hit/miss detection using architectural simulation of several SPEC 2000 benchmarks. We made the layout of all designs using Cadence(R) tools and for a commercial 0.13µm fabrication technology. We did not use an automated process to generate the designs. Instead, we used full-custom design and attempted to optimize the energy and latency of both designs as much as possible. We used the *Spectre* simulator for circuit simulations. This is the vendor recommended simulator for design validation prior to manufacturing.

The rest of this section is organized as follows. We initially consider a 1K-entry CBF with 15 bit entries as it is representative of the CBFs used in previous proposals [10,13]. In Section 3.1 we compare the energy, delay and area of the two designs for each of the three operations (increment, decrement and probe). In Sections 3.2 we study how energy and delay change as we vary the number of entries, the width of the counters and the number of taps. In Section 3.3 we demonstrate that L-CBF can reduce energy to 60% compared to S-CBF when used for early L1 cache hit/miss determination.

3.1 Delay and Energy per Operation

We compare implementations of a 1K-entry, 15-bit count per entry CBF. For S-CBF, we use an SRAM with a total capacity of 15Kbits. We partitioned the SRAM in order to minimize its power/delay product. For the S-CBF we do not consider the delay and energy overhead of the shared counter since our goal is to demonstrate that the L-CBF consumes less energy and it is also faster. To further reduce energy for probes in the S-CBF design, we introduce an extra bit per entry which is updated only when the count changes from or to zero as described in Section 2.1 (z-bits). On a probe, we only read this bit. Furthermore we applied a number of latency and power optimizations on the S-CBF [2,3,8,4] as follows. The Divided Word Line (DWL) technique which adopts a two-stage hierarchical row decoder structure was used to improve speed and power. We also used pulse operation techniques for word-lines, periphery circuits and sense amplifiers to reduce power. Also to reduce power more, multi-stage static CMOS decoding and current-mode read and write operations based on current-based circuit techniques were utilized. For the L-CBF implementation we use 16-bit LFSRs so that the LFSR can count at least 2^{15} values.

Table 1 shows the delay in picoseconds, the energy (static and dynamic) per operation in picojoules and the area in square micrometers for both the L-CBF and the S-CBF. The last column reports the ratio of S-CBR over L-CBF per metric. We report two rows per category, one for the update operation and one for the probe operation.

For delay and energy we report the worst case which we measured selecting appropriate input vectors. Given that we do not consider the overhead (latency and energy) of the shared counter, the measurements for the S-CBF are optimistic and in practice will be worse. The L-CBF is 3.7 and 1.6 times faster than the S-CBF during updates and probes respectively. In addition, the L-CBF consumes 2.3 and 1.4 times less energy compared to the S-CBF for updates and probes respectively. These significant gains in speed and energy consumption come at the expense of increased area. The L-CBF is about 3.2 times larger than the S-CBF.As we explained in Section 2.2 this is less of a concern in modern processor designs.

Table 1. Energy, delay and area of the S-CBF and L-CBF implementations of an 1K-entry, 15-bit CBF.

	Operation	L-CBF	S-CBF	S-CBF/ L-CBF
Delay (ps)	inc/dec	447.26	1670	3.7
	probe	580.32	910.12	1.6
Energy (pj)	inc/dec	38.73	88.98	2.3
	probe	30.36	41.02	1.4
Area (um^2)		945825	295570	0.31

Figure 7 shows a per component breakdown of energy consumption for the two designs and for the two operation categories. For the S-CBF, we can observe that most of the energy (79% and 74% respectively for updates and probes) is consumed by the memory core (worldlines, bitlines and SRAM cells). The decoder and the sense-amplifiers consume considerably less energy. This is expected as we applied aggressive energy and latency optimizations to these components. Finally, a small percentage of overall energy is consumed by peripheral circuitry such as the precharge and write logic.

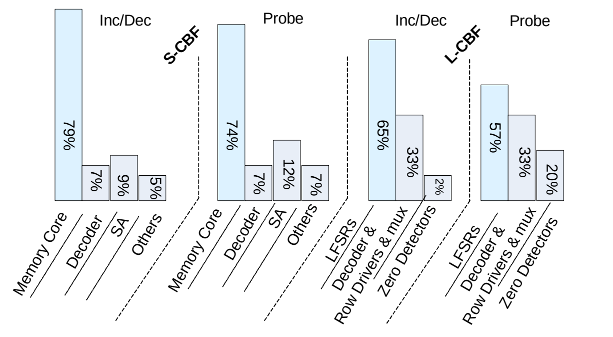


Figure 7: *Per component energy consumption for the S-CBF and the L-CBF designs. Two sets of results are shown per design, one for the update operations (Inc/Dec) and one for the probe operation.*

3.2 Sensitivity Analysis

Thus far we have focused on a specific CBF. In this section we consider varying the number of entries and the width of the counts. Figure 8 reports the energy per operation for CBFs of 64 through 1K entries in power of two steps. We observe that the L-CBF always consumes less energy than the S-CBF. The relative difference increases slightly for larger entry counts.

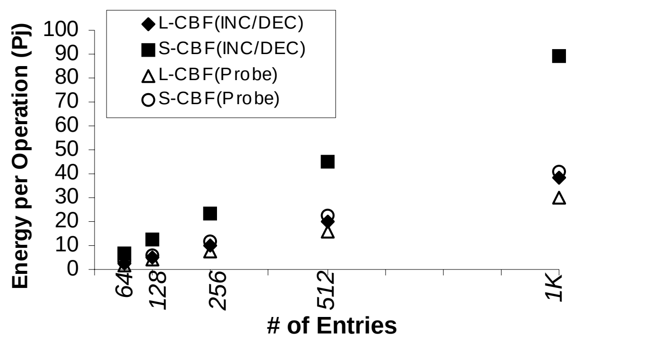


Figure 8: *Energy per operation as a function of entry count for L-CBF and S-CBF for 15-bit counts.*

Figure 9 reports the energy per operation as a function of count width in the range of four to 16 bits. In this experiment we limit our attention to a 64-entry CBF. Along the L-CBF measurements we also report the number of taps needed by each count width (either four or six). We observe that L-CBF's energy scales better than S-CBF's. L-CBF energy increases slightly for wider counts. Communication in the L-CBF is primarily between adjacent cells. For this reason, increasing the number of cells does not impact overall energy significantly. S-CBF's energy increases at a greater rate because additional bitlines and sense amplifiers are introduced and to a lesser extent because the wordlines become longer. As it can be seen in Figure 9 changing the number of taps in an L-CBF does not significantly impact energy.

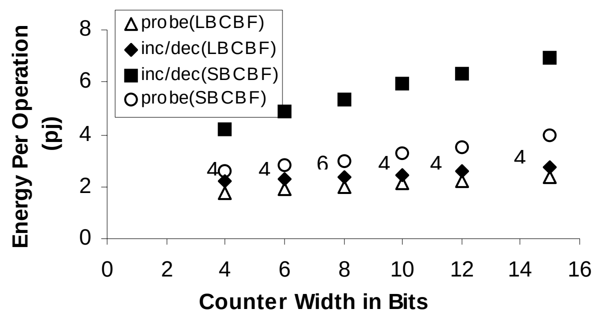


Figure 9: *Energy per operation as a function of count width for a 64-entry CBF.*

3.3 Energy Savings for Early Hit/Miss detection

Finally, we demonstrate that L-CBF can reduce energy significantly compared to a S-CBF for a practical application. We consider early L1 data cache hit/miss detection as proposed by Peir et. al. [13]. Early hit/miss detection can significantly reduce the number of instruction scheduling replays due to L1 data cache misses and hence improve performance.

We used Simplescalar v3.0 [6] to simulate the processor detailed in Table 2. We compiled the SPEC CPU 2000 benchmarks for the Alpha 21264 architecture using HP's compilers and for the Digital Unix V4.0F using the SPEC suggested default flags for peak optimization. All benchmarks were ran using a reference input data set. To obtain reasonable simulation times, samples were taken for five billion committed instructions per benchmark. We skipped 100 billion committed instructions prior to collecting measurements for all benchmarks except for art and parser for which we only skipped 20 billion instructions.

Table 2. Base processor configuration

Branch Predictor	**Fetch Unit**
8K-entry GShare and 8K-entry bi-modal 16K selector 2 branches per cycle	Up to 8 instr. per cycle 64-entry Fetch Buffer Non-blocking I-Cache
Issue/Decode/Commit	**Scheduler**
any 8 instr./cycle	128-entry64-entry LSQ
FU Latencies	**Main Memory**
Default simplescalar values	Infinite, 200 cycles
L1D/L1I Geometry	**UL2 Geometry**
64KBytes, 4-way set-associative with 64-byte blocks	2Mbytes, 8-way set-associative with 64-byte blocks
L1D/L1I/L2 Latencies	**Cache Replacement**
3/3/16 cycles	LRU
Fetch/Decode/Commit Latencies	
4 cycles + cache latency for fetch	

We simulate a 512-entry CBF with 11-bit counts. The CBF is indexed using nine continuous address bits starting immediately after the last bit that is used as an offset within a cache block. Figure 10 shows the ratio of the energy consumed by the L-CBF over the energy consumed by the S-CBF for this application. A breakdown also in terms of updates and probes is shown. Overall, L-CBF reduces energy by about 40%. Should a larger CBF was used, the energy savings would be higher. Moreover, in this experiment we do not consider any energy savings that would be possible by voltage scaling in the L-CBF. Because the L-CBF is faster than the S-CBF it may be possible to reduce power further by scaling its voltage supply.

4 Summary

We presented two designs of CBFs, one based on a SRAM array of counts and one based on an array of linear feedback shift register counters. We evaluated the energy, latency and area of the two implementations of CBFs using a commercial semiconductor technology. Finally, we studied energy consumption for a practical application of CBFs using architectural simulation. The L-CBF design is superior than the previously SRAM-based design in both latency and energy at the expense of more area.

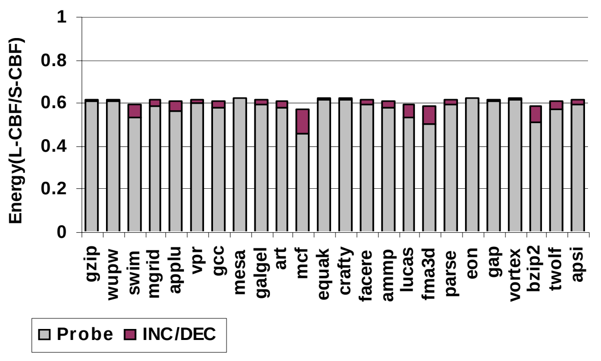


Figure 10: *Energy ratio of L-CBF over S-CBF for a 512-entry, 11-bit count CBF for early L1 data cache hit/miss detection.*

References

[1] P. Alfke, "Efficient Shift Registers, LFSR Counters, and Long Pseudo-Random Sequence Generators", Xilinx, Application Note 052 July 7,1996.

[2] B. S. Amrutur and M.A. Horowitz, "Fast Low-Power Decoders for RAMs", IEEE Journal of Solid-State Circuits, 36(10):1506- 1515, Oct 2001.

[3] B. S. Amrutur, "Design and Analysis of Fast Low Power SRAMs,", Ph.D. dissertation, Electrical Engineering Department, Stanford University, 1999.

[4] B.S. Amrutur and M.A. Horowitz. "Speed and Power Scaling of SRAM's". IEEE Journal of Solid-State Circuits, 35(2):175-185, February 2000.

[5] P. H. Bardell, W. H. McAnney, and J. Savir,"Built-In Test for VLSI: Pseudorandom Techniques", John Wiley & Sons, Inc., 1987.

[6] D. Burger and T. Austin. "*The Simplescalar Tool Set v2.0*", Technical Report UW-CS-97-1342. Computer Sciences Department, University of Wisconsin-Madison, June 1997.

[7] L. Fan, P. Cao, J. Almeida, and A. Z. Broder. "Summary Cache: A Scalable Wide-area Web Cache Sharing Protocol", IEEE/ACM Transactions on Networking, 8(3):281–293, 2000.

[8] M. Margala, "Low-power SRAM Circuit Design", In Proceedings of IEEE Workshop on Memory Technology, Design and Testing, 115 - 122, Aug. 1999.

[9] A. Moshovos, "RegionScout: Exploiting Coarse-Grain Sharing in Snoop-Coherence", In Proc. Annual International Symposium on Computer Architecture, Jun. 2005.

[10] A. Moshovos, G. Memik, B. Falsafi., and A. Choudhary, "Jetty: Filtering Snoops for Reduced Energy Consumption in SMP Servers", In Proceedings of the Annual International Conference on High-Performance Computer Architecture, 85–96, Feb. 2001.

[11] S. Sethumadhavan, R. Desikan, D. Burger, C.R. Moore, S.W. Keckler, "Scalable Hardware Memory Disambiguation for High-ILP Processors", IEEE Micro, 24(6):118 - 127, Nov. 2004.

[12] M. R. Stan, "Synchronous Up/Down Counter with Clock Period Independent of Counter Size", IEEE Symposium on Computer Arithmetic, 274-281, July 1997.

[13] J.K. Peir, S.C. Lai, S.L. Lu, J. Stark, and K. Lai, "Bloom Filtering Cache Misses for Accurate Data Speculation and Prefetching", Annual International Conference on Supercomputing, June 2002.