DESC: Energy-Efficient Data Exchange using Synchronized Counters

Mahdi Nazm Bojnordi and Engin Ipek
University of Rochester
Rochester, NY 14627 USA
{bojnordi, ipek}@ece.rochester.edu

ABSTRACT

Increasing cache sizes in modern microprocessors require long wires to connect cache arrays to processor cores. As a result, the last-level cache (LLC) has become a major contributor to processor energy, necessitating techniques to increase the energy efficiency of data exchange over LLC interconnects.

This paper presents an energy-efficient data exchange mechanism using synchronized counters. The key idea is to represent information by the delay between two consecutive pulses on a set of wires, which makes the number of state transitions on the interconnect independent of the data patterns, and significantly lowers the activity factor. Simulation results show that the proposed technique reduces overall processor energy by 7%, and the L2 cache energy by 1.81× on a set of sixteen parallel applications. This efficiency gain is attained at a cost of less than 1% area overhead to the L2 cache, and a 2% delay overhead to execution time.

Categories and Subject Descriptors

B.3 [**Hardware**]: Memory Structures; B.4.3 [**Hardware**]: Input/Output and Data Communications—*Interconnections*

General Terms

Design, Management, Performance

Keywords

Low power, Interconnect, Caches, Data encoding, Signaling

1. INTRODUCTION

On-chip interconnect is a major contributor to microprocessor energy. Recent studies have shown that more than 60% of the dynamic power in modern microprocessors is dissipated in driving on-chip interconnects [1, 2, 3]. Not only do long wires present a highly capacitive load, but they also expend significant energy due to the repeaters inserted along the interconnect to linearize wire delay. Current scaling trends indicate that interconnect power will be even higher in future billion-transistor multicore processors [4, 5].

Last-level caches occupy significant area, and incur large energy and delay overheads in data exchange over long wires. Figure 1 shows the percentage of processor energy expended on an 8MB cache when running a set of parallel applications on a Sun Niagara-like multicore processor.[1] On average, the L2 cache is responsible for 15% of the total processor energy. Figure 2 breaks down the L2 cache energy into its constituent parts, and shows that communication over the long, capacitive H-tree interconnect is the dominant source of energy consumption (80% on average) in the L2 cache. (This assumes that the L2 cache employs low standby power devices that are optimized for energy efficiency, as explained in Section 4. L2 power would be even higher if high-speed—and thus, high leakage—transistors were used.) Consequently, numerous architectural techniques have been developed to mitigate interconnect energy in last-level caches.

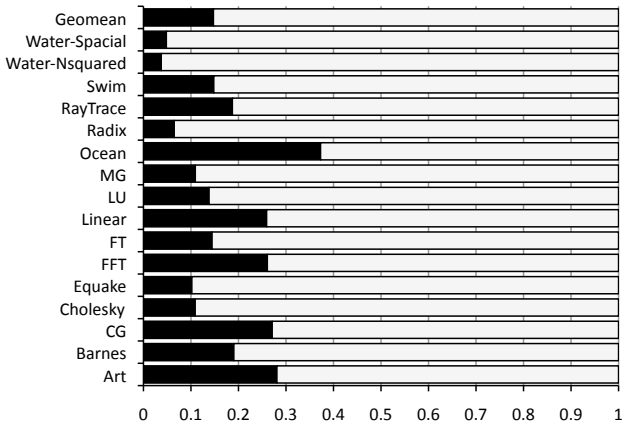

Figure 1: L2 energy as a fraction of total processor energy. The processor configuration is explained in Section 4.

An effective way of reducing cache interconnect energy is to reduce the activity factor—i.e., the switching probability—of the wires. This can be accomplished, for example, by encoding the data such that successive transmissions on the data bus flip the state of only a small number of wires, or by compressing the data in the last-level cache. Interconnect energy can also be reduced by reducing the voltage, or

[1] The experimental setup is explained in Section 4.

MICRO'46 December 7-11, 2013, Davis, CA, USA

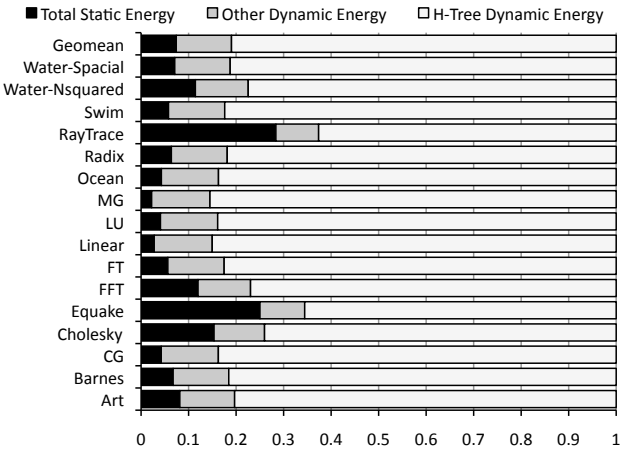

Figure 2: Major components of the overall energy consumption in an 8MB L2 cache. The L2 cache employs low standby power devices that are optimized for energy efficiency. More details on the experimental setup are provided in Section 4.

by using techniques to alleviate the adverse effects of capacitance on power and speed. Among these, techniques that reduce the activity factor are broadly applicable since they can be used on interconnects with different characteristics (e.g., transmission lines [6] or low-swing wires [7, 2]).

This paper presents DESC, an energy-efficient data exchange method using synchronized counters. The goal is to reduce the activity factor on the highly capacitive LLC interconnect. The key idea is to represent information by the number of clock cycles between two consecutive pulses on a set of wires that connect two on-chip communication points, thereby limiting the number of switching events to one per transmission. This results in a time-based data representation and creates new opportunities for further power optimization.

When applied to a Sun Niagara-like multicore processor with an 8MB L2 cache, the proposed approach reduces overall L2 cache energy by 1.81×. This results in 7% savings in overall processor energy at a cost of less than 2% increase in execution time, and less than 1% increase in the L2 cache area.

2. BACKGROUND

This paper builds upon existing optimization techniques for caches and on-chip communication. Kim *et al.* [8] propose a non-uniform cache architecture (NUCA) that allows optimizing access time and energy based on the proximity of the cache blocks to the cache controller. This optimization is orthogonal to DESC and can supplement it by reducing the average access time and energy.

Numerous proposals identify opportunities for cache compression [9, 10, 11], which influences overall cache energy by reducing the amount of information stored in SRAM arrays and transferred on the cache interconnect. In contrast, DESC provides an efficient data exchange mechanism to optimize cache interconnect energy, which can be combined with SRAM array optimizations. It also has mechanisms that exploit null and redundant blocks, and compares favorably to dynamic zero compression [12].

Balasubramonian *et al.* [13] propose a hybrid interconnect comprising wires with different latency, bandwidth, and power characteristics. Specifically, recent work [14] proposes techniques to exploit specific wires and topologies for the address network. In contrast, DESC improves cache energy efficiency by decreasing the number of data interconnects and the signal transition rate across those interconnects. This makes DESC applicable to either hybrid or single interconnects.

Techniques for energy efficient bus encoding are applicable to cache interconnects. One such technique, bus invert coding [15], sets an upper bound of $N/2$ bit-flips for any N-bit information transferred on N wires. This is achieved by the use of one extra wire to indicate the encoding of the value on the other wires (non-inverted or inverted). If the Hamming distance between the present value and the last value on an N-wire bus exceeds $N/2$, the inverted code is transmitted. Unlike bus invert coding, DESC employs a time-dependent representation of information on wires. This creates new opportunities for efficient data transfers, and ultimately delivers fewer bit flips.

Other proposals focus on reducing static cache power, such as Cache Decay [16], which takes a generational approach to reducing cache leakage power, and Drowsy Caches [17], which implement low power modes to reduce cache leakage. Recent work [18] proposes to optimize leakage power by applying different techniques to L1 and L2 caches. DESC is orthogonal to these proposals and can combine with them in highly energy-efficient cache design.

The concept of time-based information representation is used by pulse position modulation in telecommunications [19]. To the best of our knowledge, DESC is the first proposal that efficiently adopts this concept in on-chip caches.

3. DATA EXCHANGE USING SYNCHRONIZED COUNTERS

DESC is an energy-efficient data exchange technique for synchronous on-chip communication. A *DESC transmitter* sends information to a *DESC receiver* over a set of conventional data wires; the receiver and the transmitter also share a reset wire. The key idea is to represent information by the delay between two pulses (in clock cycles), one on the shared reset wire and the other on one of the data wires connecting the receiver and the transmitter. The receiver is synchronized with the transmitter through a *synchronization strobe* sent from the transmitter. DESC uses an address transition detection (ATD) circuit [20] for this synchronization.

Figure 3 shows an example using three communication techniques: parallel, serial, and DESC. In the figure, one byte (01010011) is sent from a transmitter to a receiver; all wires are assumed to hold zeroes prior to the transmission. With parallel data transfer, eight physical wires transfer the value in one cycle, which results in four bit-flips on the interconnect. Serial data transfer requires only a single wire, but results in longer transfer time (eight cycles); the result of this transmission is five bit-flips. DESC needs three wires—two for data and one for reset. The data byte is divided into two four-bit *chunks*, and each chunk is sent by toggling one of the two data wires. The reset wire is shared by all data wires to specify the start of the data transmission (before the transmission of a cache block); the number of clock cycles between the reset signal and a bit-flip on a data

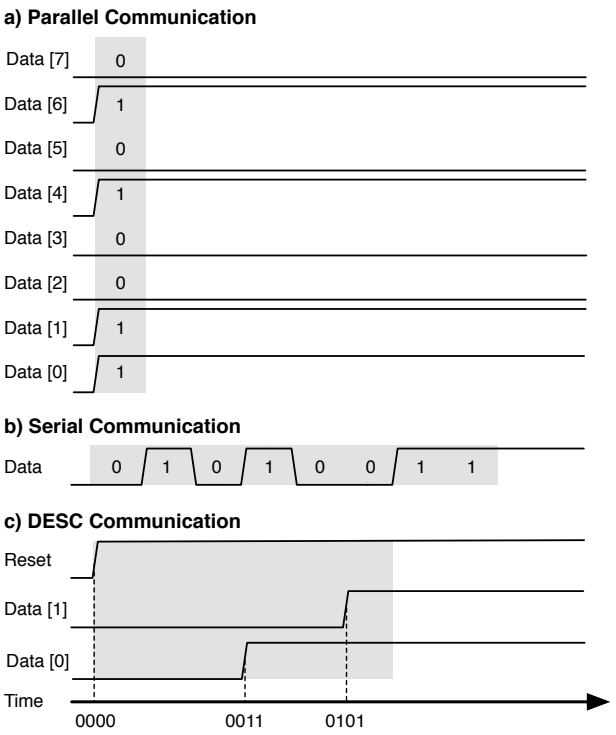

Figure 3: Illustrative example of three communication techniques: (a) parallel, (b) serial, and (c) DESC.

wire represents the value of the corresponding chunk. The transfer results in a total of three bit-flips across the reset and data wires. (Although the synchronization strobe is not shown in the figure, its overheads are accounted for in the evaluation.) This simple example suggests that DESC can reduce interconnect energy by using fewer wires than parallel transmission, and by restricting the number of bit-flips to one per chunk.

3.1 Communication Protocol

In DESC, cache blocks are partitioned into fixed-size, contiguous chunks, and each chunk is assigned to a specific wire; if the number of chunks is greater than the number of wires, multiple chunks are assigned to each wire, and are transmitted successively (Figure 4). Prior to transmission, every chunk is buffered in a FIFO queue dedicated to its wire.

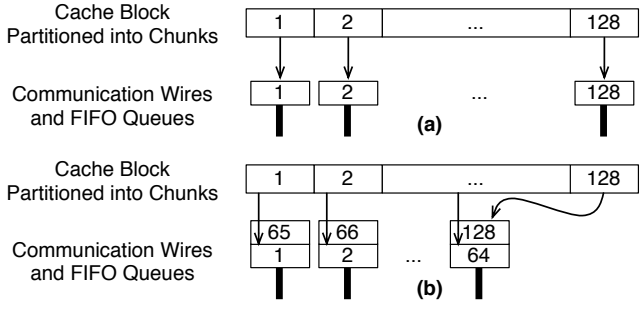

Figure 4: Assignment of chunks to wires when the number of wires is equal to (a) and less than (b) the number of chunks.

DESC employs a communication protocol comprising *reset* and *hold* signals. Every cache block transmission starts with the toggling of the shared reset wire. The transmission of the first chunk assigned to a data wire ends when that wire is toggled for the first time; the number of cycles between the toggling of the reset and data wires represents the transmitted value. This initial toggling of the data wire concludes the transmission of the first chunk, and starts the transmission of the next chunk. This second chunk, in turn, completes its transmission the next time the data wire is toggled; hence, the second chunk value is equal to the number of clock cycles between the first and the second time the data wire changes its state. Notice that this makes the transmission time value-dependent, while keeping the switching activity (and hence energy) on the data wires constant (one per chunk).

Data Transmission. At the beginning of every cache block transmission, the transmitter (1) sends a reset command to the receiver, and (2) resets an internal counter, which is incremented by one every cycle. Once the reset signal is sent, the transmitter starts monitoring the chunks at the heads of the FIFO queues, and checks if any chunk matches the current counter value. On a match, the transmitter sends a hold command to the receiver by toggling the matching data wire. As soon as the reset signal arrives at the receiver, the receiver resets its own counter to zero; later, when the hold command arrives, the receiver interprets its counter value as the value of the received chunk.

Figure 5 shows an example of the required signaling between a transmitter and a receiver. A single data wire is used for sending two chunks, with values two and one. In the example, each chunk is three bits wide, which requires two three-bit counters, one at the transmitter and one at the receiver. At the beginning of the transmission, both values are enqueued at the transmitter FIFO queue, and the transmitter sends them in FIFO order—first two and then one.

The transmitter sends the reset command to the receiver through a physical wire called the *reset strobe*, and resets its own counter. Thereafter, the transmitter counter is incremented by one each cycle until the count reaches two, which matches the chunk at the head of the FIFO queue. At this point, the transmitter sends a hold command to the receiver by toggling the data wire (i.e., by sending a data strobe); this DESC transmission takes three cycles, after which the transmission of the next chunk begins.

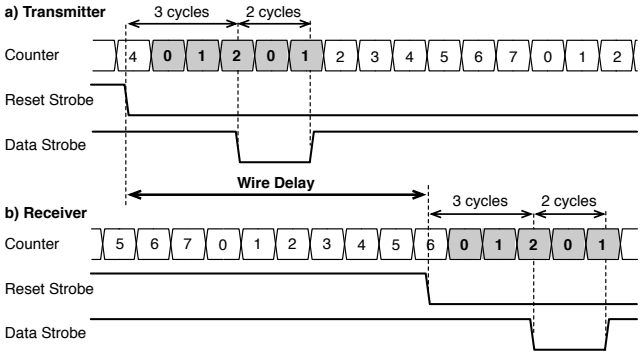

Figure 5: Illustrative example of the signaling necessary for transmitting two three-bit chunks using a single data wire.

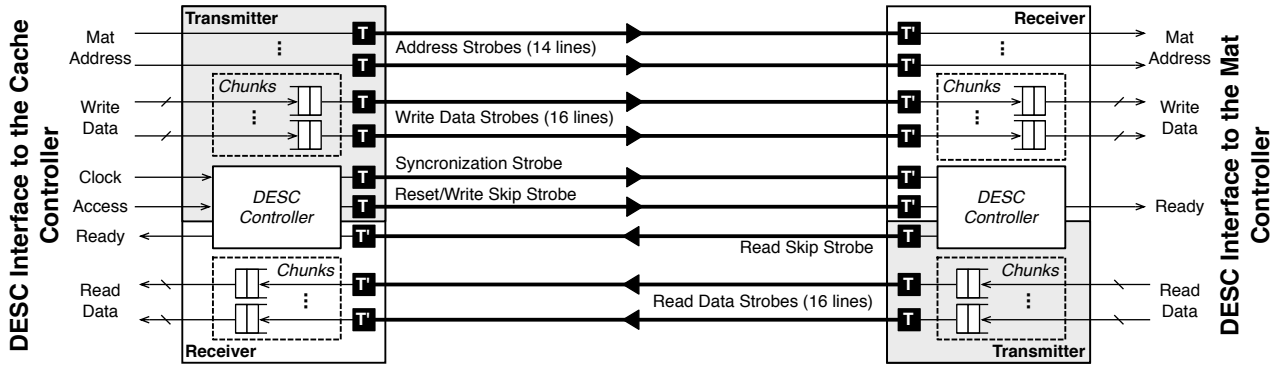

Figure 6: Illustrative example of the interface that connects a DESC transmitter and receiver on the cache controller to a DESC transmitter and receiver on a mat controller. Only a part of the block transfer is shown in the figure for simplicity. T and T' indicate toggle generator and toggle detector, respectively.

Ultimately, strobe signals sent by the transmitter arrive at the receiver. There, the first reset command resets the receiver's counter to zero. Three cycles later, a hold strobe arrives that makes the receiver latch in "two" as the value of the first chunk. Similarly, another data strobe indicating a hold command results in receiving the value "one".

Synchronization. The transmitter and the receiver involved in a data transfer need to operate synchronously with respect to either a global clock signal, or a local point-to-point synchronization strobe. In a synchronous cache design, all DESC controllers would be synchronized using the cache clock distribution network, whereas in an asynchronous cache (i.e., an ATM controlled cache [20]), a synchronization strobe between the cache controller and mats would be necessary. This paper applies DESC to asynchronous last-level caches due to their superior power efficiency [20, 21]. The transmitter and receiver counters operate at the same clock frequency (possibly with different clock phases). This requires a dedicated wire carrying the clock information from the transmitter to the receiver. To reduce the number of transitions on this wire, DESC generates a synchronization strobe signal that toggles at half the frequency of the clock, but triggers the receiver at both rising and falling edges.

3.2 Applying DESC to Last-Level Caches

Figure 7 shows an example last-level cache system with a DESC interface, comprising eight banks, four subbanks per bank, and four mats per subbank. Data communication between the cache controller and mats is performed through the DESC transmitter and receiver. The cache controller, as well as every mat controller, is equipped with a DESC interface unit that consists of a transmitter and a receiver for exchanging data between the cache controller and the activated mats. In a last-level cache, wire resources are often shared between subbanks to reduce the wiring overhead, as shown by the vertical H-tree in Figure 7. Transferring toggles over shared wires requires remembering the previous states of different segments of the vertical H-tree. This is accomplished by a toggle regenerator circuit that receives toggles from one of the two branches of the vertical tree (i.e., the branch connected to the active subbank), and transfers the toggles upstream. The selection between the branches is performed based on the address bits.

Figure 6 shows an example in which a DESC transmitter

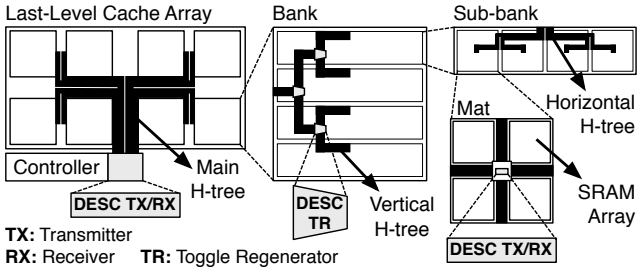

Figure 7: Illustrative example of a last-level cache with DESC. The DESC interface units on the cache controller and the selected mats are involved in every cache block transfer.

and receiver at a cache controller are connected to a DESC transmitter and receiver at a mat. The DESC interface is used for transferring 512-bit cache blocks between the cache controller and multiple mat controllers; the figure, however, simplifies the exposition by showing only one mat controller and a part of the cache controller.

3.2.1 Transmitter Architecture

The DESC transmitter receives requests from the last-level cache controller, and generates the necessary transitions on the communication wires. DESC can be applied to the entire request, including data, address, and control bits. Applying DESC to the address bits requires major modifications to the horizontal and vertical H-trees to support address-based branch selection. Moreover, the physical wire activity caused by the address bits in conventional binary encoding is relatively low, which makes it inefficient to apply DESC to the address wires. Consequently, the proposed technique opts out of applying DESC to address and control bits, and instead transmits these with conventional binary encoding.

Unlike the address and control bits, the data bits are first stored in the chunk transmitter latches, and are later modulated in time over the data wires. As shown in Figure 6, a 64-bit data block (a part of the cache block stored in the target mat) is divided into 16 chunks, each of which is stored in a four-bit register. (Transferring a 512-bit cache block requires 128 chunks in total.) The transmitter uses an internal counter triggered by an input clock signal. The clock is provided to the DESC transmitters at the cache controller

interface. When there is an ongoing data transfer, the clock signal is used to generate a synchronization strobe with a toggle generator, as shown in Figure 8-a, and is sent to the DESC receiver at the mat controller. The receiver recovers the clock from the synchronization strobe using a toggle detector as shown in Figure 8-b. The recovered clock triggers the internal counters of the receiver and the transmitter at the mat controller.

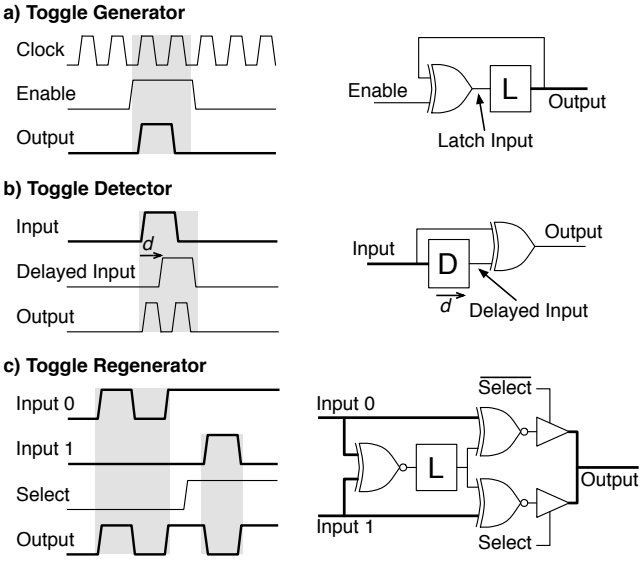

Figure 8: Illustrative example of the toggle generator (a), detector (b), and regenerator (b) used by DESC.

3.2.2 *Receiver Architecture*

The job of the DESC receiver is to detect the communication strobes sent by the transmitter, and to recover the data values. The DESC receiver consists of an internal counter and 16 chunk receivers. Each chunk receiver monitors a strobe wire to detect value changes. Once a strobe signal arrives at the receiver, the counter value is loaded into a four-bit register. Because of the equalized transmission delay of the wires in the cache H-tree, the content of the DESC receiver counter at the time the strobe is received is always the same as the content of the transmitter counter at the time the strobe is transmitted, which allows the chunk receiver to recover the original data sent by the transmitter. After all data chunks are received, a ready signal marks the end of the data transfer.

3.2.3 *Handling ECC Checks*

Applying DESC to an LLC requires no modifications to the SRAM arrays since data are stored in standard binary format. This makes it possible to use conventional error correcting codes (ECC) to protect the memory arrays. However, DESC creates a new set of challenges that must be met to protect the H-trees. Unlike conventional binary encoding, DESC transfers a chunk of data using a single transition on a wire; as a result, an error in the H-trees affects a chunk of information, and may corrupt up to four bits. Overcoming this problem requires adopting a carefully designed data layout for the parity bits used in single-error correction, double-error detection (SECDED) ECC.

Similar to conventional SECDED ECC, DESC uses parity bits to protect the H-trees and the SRAM arrays. Extra wires are added to the data bus to transfer a set of *parity chunks* (Figure 9). Each parity chunk is transferred by a single transition over ECC wires, and contains four parity bits, each of which belongs to a different data segment. For example, the (137, 128) Hamming code requires nine parity bits to protect a 128-bit data segment [22], and DESC supports this scheme by adding nine additional wires. A cache block is partitioned into four 128-bit data segments, each of which is augmented with nine parity bits. (Conventional ECC schemes use similar interleaving techniques for better protection against multiple adjacent errors [23].) Therefore, every data chunk contains at most one bit from each data segment; similarly, each parity chunk contains one parity bit from each segment. This guarantees that a single error in the H-trees cannot corrupt more than one bit from each segment. Similarly, a double error in a data segment can be detected because two errors in the H-trees may affect no more than two bits per segment. This allows DESC to support conventional SECDED ECC. So long as the segments are narrower than the data bus, DESC can accommodate conventional SECDED ECC without additional parity bits. This scheme is extendible to cache architectures with separate storage for ECC [24].

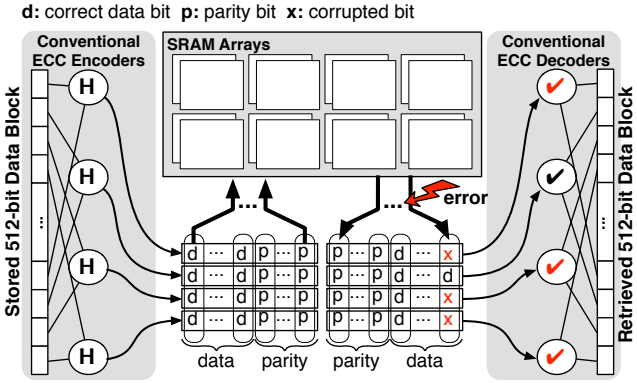

Figure 9: Illustrative example of ECC bits in DESC.

3.3 Energy and Delay Optimizations

DESC exploits data locality and null block prevalence in the last-level cache to optimize power and delay. By default, every chunk requires a single bit-flip between a transmitter and the corresponding receiver. This sets a lower bound on the number of required bit-flips to transmit a cache block; for example, to transmit a 512-bit cache block from a transmitter comprising 128 chunk transmitters, where a data chunk is four bits, 128 transitions are needed on the data wires. Experiments show that a significant number of the chunk values transferred using DESC are either zeroes or the same as the values transferred on the same wires in the previous cycle. DESC supports a value skipping mechanism to reduce the unnecessary energy and delay due to these repeated or null chunks.

DESC defines a time window during which a block is transferred. Figure 10-a shows an example time window. The window opens with a reset signal and closes with the last chunk signal. In this example, a data block consisting of four chunks (0, 0, 5, and 0) is transferred using five

bit-flips. Figure 10-b illustrates how the same chunk values are communicated with *value skipping* in DESC. The time window is defined by two bit-flips on the reset/skip wire, and each non-zero chunk (such as 5 in the figure) is transferred by a bit-flip within the time window. Those wires that remain silent over the duration of the time window are set to zeroes at the end of the transfer; hence, the four chunk values are transferred using three bit-flips. This scheme allows DESC to selectively specify a subset of the chunks to get assigned a predefined value at the cost of a bit-flip on the shared reset/skip wire. Unlike DESC, existing dynamic power optimization schemes [15, 12] require transferring extra information per chunk to support similar skipping operations. (As an alternative, these existing methods can implement value skipping at the cache block level to reduce the overheads; however, this reduces the effectiveness of these schemes.) For example, bus invert coding [15] can implement zero skipping with control overheads. To accomplish this, every chunk would need to determine if its value is skipped. Combining this new feature with the original bus invert coding results in three data transfer modes per chunk (non-inverted, inverted, and skipped). This requires extra wires and results in energy overheads. (A comparison against zero compression and bus invert coding + zero skipping is presented in Section 5.)

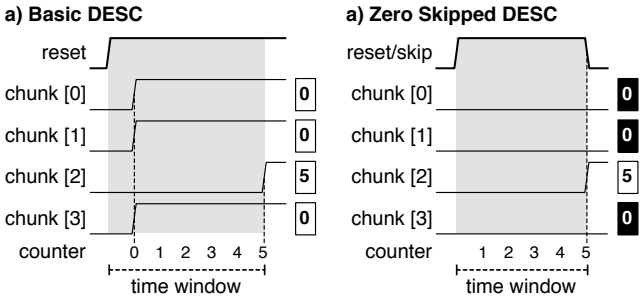

Figure 10: Illustrative example of time windows in the basic (a) and zero skipped (b) DESC techniques.

In addition to energy, value skipping can also improve transmission latency. Applying value skipping to DESC results in a narrower time window since the skipped value is excluded from the count list. As shown in Figure 10, zero skipped DESC requires a time window of five cycles as opposed to the six cycles needed in the basic DESC.

DESC implements value skipping at both the cache controller and mat interfaces. Figure 11 shows how the transmitter and the receiver skip a value. Prior to transmitting a chunk, the transmitter compares the chunk's value to a pre-determined "skip value"; on a match, the transmitter sets an internal flag indicating that the chunk should be skipped; otherwise, the relevant chunk transmitter is activated to send the data strobe. After sending the required strobe signals for all unskipped data chunks, a skip signal indicating the end of the transmission is sent to the receiver by toggling the reset/skip wire. The receiver receives data strobes for unskipped chunks only; receiving a skip signal from the transmitter before all data strobes are toggled is interpreted as a skip command. When a skip command arrives at the receiver, the skip value is copied to all pending chunks. A transition on the DESC reset/skip wire is interpreted as (1) a counter reset command signaling the beginning of a new cache block transfer if there is no incomplete chunk at the receiver, or (2) a skip command if some of the chunk receivers have not yet received a signal from the transmitter.

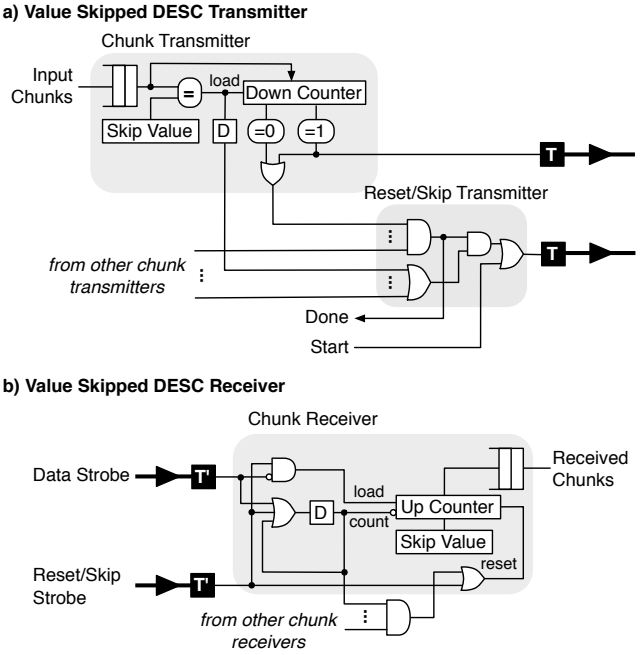

Figure 11: Illustrative example of value skipping. T and T' indicate toggle generator and toggle detector, respectively.

This paper experiments with two different variations of value skipping on DESC: *zero skipping* and *last-value skipping*.

Zero Skipping. Zero is the most common chunk value read from the last-level cache. Studying the behavior of different applications on an 8MB L2 cache with a 4-bit DESC interface, we find that 31% of the transmitted chunks are zeroes (Figure 12). This motivates us to employ zero skipping in DESC, wherein the skip value is set to zero.

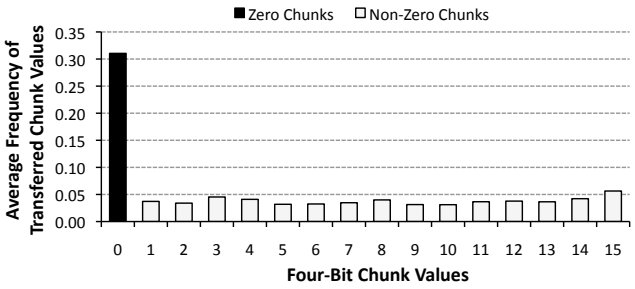

Figure 12: Distribution of four-bit chunk values transferred between the L2 cache controller and the data arrays.

We also considered adaptive techniques for detecting and encoding frequent non-zero chunks at runtime; however, the attainable delay and energy improvements are not appreciable. This is because of the relatively uniform distribution of chunk values other than zero, as seen in Figure 12.

Last-Value Skipping. Last-value skipping sets the skip

value for each chunk to the previous value transmitted on the wire assigned to that chunk. This technique creates more opportunities for skipping. Figure 13 shows the measured probability of transmitting two consecutive data chunks with the same values. On average, 39% of the transmitted chunks on a wire have the same value as the previous chunk.

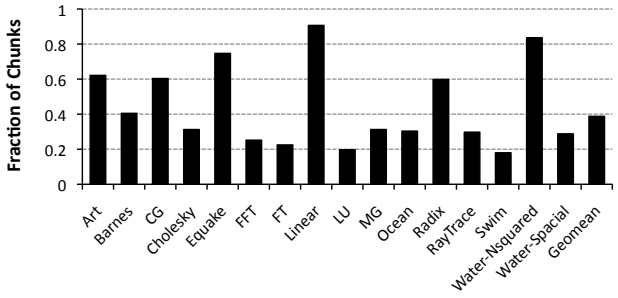

Figure 13: Fraction of chunks transferred between the processor and the L2 cache that match the previously transmitted chunk.

4. EXPERIMENTAL SETUP

We evaluate the area, frequency, and power overheads of DESC by implementing it in Verilog HDL, and synthesizing the proposed hardware. We apply the proposed techniques to UCA and S-NUCA-1 [8] L2 caches, using a modified version of CACTI 6.5 [14] to evaluate the energy efficiency and performance potential. We also implement the dynamic zero compression [12] and bus invert coding [15] techniques as baselines. We simulate 16 memory-intensive parallel applications, running on a heavily modified version of the SESC simulator [25]. Using McPAT [26], we estimate the overall processor power with and without DESC at the L2 cache.

4.1 Architecture

We modify the SESC simulator [25] to model two different kinds of systems: (1) a Niagara-like eight-core processor, and (2) a single-threaded out-of-order processor. Both systems have an 8MB L2 cache, interfaced to two DDR3-1066 DRAM channels. Table 1 shows the simulation parameters.

Core	Multithreaded	8 in-order cores, 3.2 GHz, 4 HW contexts per core
	Single-threaded	4-issue out-of-order core, 128 ROB entries, 3.2 GHz
IL1 cache (per core)	16KB, direct-mapped, 64B block, hit/miss delay 2/2	
DL1 cache (per core)	16KB, 4-way, LRU, 64B block, hit/miss delay 2/2, MESI protocol	
L2 cache (shared)	8MB, 16-way, LRU, 64B block, hit/miss delay 19/12	
Temperature	350 °K (77 °C)	
DRAM	2 DDR3-1066 memory channels, FR-FCFS	

Table 1: Simulation parameters.

Large SRAM caches have a significant impact on the energy and performance of microprocessors. Due to the large number of transistors dedicated to the last level cache, designing an energy-efficient microprocessor necessitates optimizations that reduce excessive leakage energy in the SRAM cells and the peripheral circuitry. Such techniques can reduce cache leakage by orders of magnitude. (See [27, 28, 29, 30, 31, 32, 33, 34] for examples of recent industrial designs with low-leakage devices in caches.)

We use CACTI 6.5 and SESC to explore the design space of the L2 cache, searching for the most energy efficient technology for the SRAM cells and the peripheral circuitry. The design options in this exploration are ITRS high performance (HP), ITRS low power (LOP), and ITRS low standby power (LSTP) devices [14]. In addition to the cell technologies, we explore the energy efficiency of the L2 cache for various bank counts (from 2 to 64) and data H-tree widths (from 8 to 512 bits). The goal is to set the most energy efficient configuration for the baseline L2 cache. Figure 14 shows the relative cache energy, the overall execution time, and the processor energy averaged over all 16 applications.[2] The results show that using LSTP technology for both the SRAM cells and the peripheral circuitry leads to a significant energy reduction in the L2 cache at the cost of a negligible loss in performance.[3] As a result, the most energy efficient L2 cache configuration consists of 8 banks with 64-bit data H-trees, implemented with LSTP devices, which minimizes both the L2 cache and total processor energy. This configuration adds only 2% overhead to the total execution time compared to the fastest L2 cache implemented with HP devices.

We modify SESC to support multiple interfaces at the L2 cache for conventional binary transmission, bus-invert coding [15], dynamic zero compression [12], S-NUCA-1 [8], and DESC. The L2 cache decodes an input address to activate a wordline in the data and tag arrays, which are co-located. The contents of a row are sensed and read out through the bitlines. On a hit, the comparator logic selects one way of the data array to be transferred on the H-trees.

Because bus-invert coding and dynamic zero compression are sensitive to the number of bus segments, we find the best segment size for each and use it in the baselines.[4] For further energy efficiency, we also combine bus invert coding with a zero skipping mechanism that eliminates unnecessary bit-flips when transferring zero values. We evaluate two variants of zero skipped bus invert coding. First, we assume that an extra wire is employed for transferring the zero skipping signal per data segment, which results in a large number of extra wires. This scheme takes into account the flips that would occur on the extra wires when deciding the best encoding scheme for each segment. In addition to this sparse representation, we also experiment with a denser representation that uses a single binary number to encode the transmission mode for all segments. This denser representation reduces the number of extra wires, but incurs additional decoding and encoding overheads.

Figure 15 shows the L2 cache energy for these techniques normalized to binary encoding. We select the best configuration for each technique as a baseline, and use these baselines to make comparisons against DESC. Finally, we implement three versions of DESC: *basic* (i.e., no value skipping), *zero skipped* (Section 3.3), and *last-value skipped* (Section 3.3).

[2] Although we do a full design space exploration on the number of banks and bus width for all device types, the figure illustrates a representative subset of the results for clarity.

[3] Although using high-performance devices in both the cells and the periphery results in approximately 2× faster access time compared to low standby power devices, the end-to-end system performance exhibits less than 2% sensitivity to this difference.

[4] The paper ignores energy and latency overheads of the necessary control logic in these baseline schemes–e.g., population counters and zero detection logic.

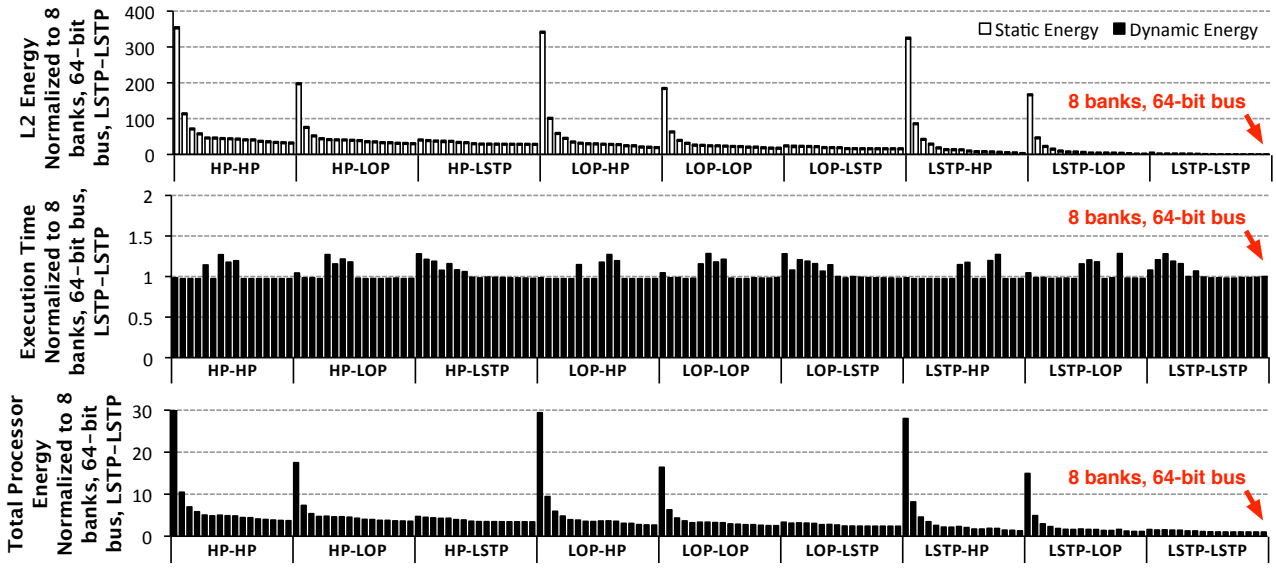

Figure 14: Design space exploration of the L2 cache for energy and execution time.

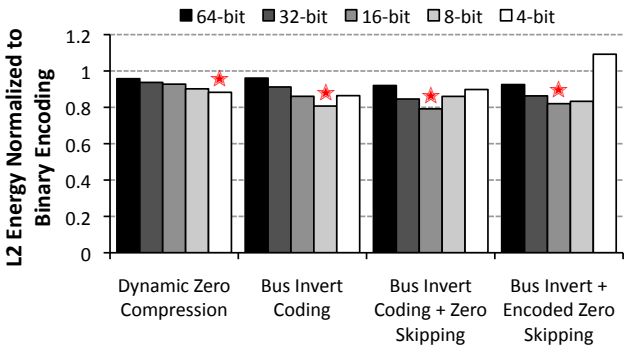

Figure 15: L2 cache energy as a function of the data segment size for the Dynamic Zero Compression, Bus Invert Coding, Bus Invert + Zero Skipping, and Bus Invert + Encoded Zero Skipping schemes. The best configuration for each scheme (marked with a star) is selected as a baseline for evaluation.

	Benchmarks	Suite	Input
Parallel	Linear Regression	Phoenix	50MB key file
	MG	NAS OpenMP	Class A
	CG	NAS OpenMP	Class A
	Swim-Omp	SPEC OpenMP	MinneSpec-Large
	Equake-Omp	SPEC OpenMP	MinneSpec-Large
	Art-Omp	SPEC OpenMP	MinneSpec-Large
	Barnes	SPLASH-2	16K Particles
	Cholesky	SPLASH-2	tk 15.0
	Ocean	SPLASH-2	514×514 ocean
	FFT	SPLASH-2	1M points
	Radix	SPLASH-2	2M integers
	LU	SPLASH-2	512×512 matrix, 16×16 blocks
	Ocean	SPLASH-2	258×258 ocean
	RayTrace	SPLASH-2	car
	Water-Spatial	SPLASH-2	512 molecules
	Water-NSquared	SPLASH-2	512 molecules
Single-threaded	bzip2	SPECint 2006	reference
	mcf	SPECint 2006	reference
	omnetpp	SPECint 2006	reference
	sjeng	SPECint 2006	reference
	lbm	SPECfp 2006	reference
	milc	SPECfp 2006	reference
	namd	SPECfp 2006	reference
	soplex	SPECfp 2006	reference

Table 2: Applications and data sets.

4.2 Applications

Evaluated parallel workloads represent a mix of sixteen data-intensive applications from Phoenix [35], SPLASH-2 [36], SPEC OpenMP [37], and NAS [38] suites. We also assess the tolerance of single threaded applications to the access latency of DESC by evaluating eight applications from the SPEC 2006 suite. In this experiment, we use SimPoint [39] and determine a representative 200 million instruction region from each SPEC 2006 application to reduce the execution time. The parallel applications are simulated to completion. Table 2 summarizes the evaluated benchmarks and their input sets. All applications are compiled using GCC with the -O3 optimization flag.

4.3 Synthesis

We evaluate the area and power overheads of DESC by implementing it in Verilog HDL, and synthesizing the design using Cadence Encounter RTL Compiler [40] with FreePDK [41] at 45nm. The results are then scaled to 22nm (relevant parameters are shown in Table 3).

Technology	Voltage	FO4 Delay
45nm	1.1 V	20.25ps
22nm	0.83 V	11.75ps

Table 3: Technology parameters [42, 43].

5. EVALUATION

We first present synthesis results on the area, power, and delay figures for the DESC transmitter and receiver. Next, we compare cache energy, power, and delay when using DESC and conventional cache interfaces; for the latter, we

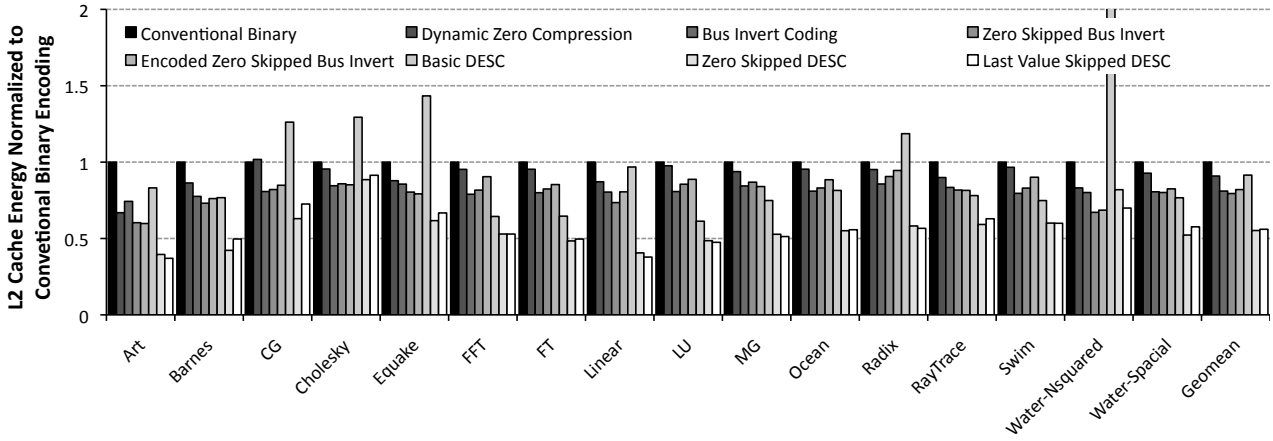

Figure 16: L2 cache energy achieved by different data transfer techniques.

experiment with baselines incorporating bus invert coding and dynamic zero compression. We then evaluate the impact of DESC on overall system energy efficiency, and conduct a cache design-space exploration over all sixteen applications. We also evaluate applying DESC to an S-NUCA-1 cache. We study the sensitivity of DESC to the number of cache banks, H-tree width, and chunk size. We then evaluate the performance and energy potential of DESC under SECDED ECC. Finally, we assess the performance impact of DESC when running latency sensitive, single threaded applications.

5.1 Synthesis Results

Synthesis results on the area, power, and delay contributions of the DESC transmitter and receiver are shown in Figure 17. A synthesized DESC interface occupies $2120\mu m^2$ at each cache mat, which adds less than 2% area overhead to the mats, and a total area overhead of 1% to the 8MB L2 cache. The peak power consumption of a DESC interface, including the transmitter and the receiver, is $46mW$. (Notice that DESC consumes dynamic power only during data transfers.) A pair of fully synthesized DESC interfaces adds $625ps$ of logic delay to the round trip cache access time. Power, delay, and area can be further improved by using custom circuit design rather than synthesis.

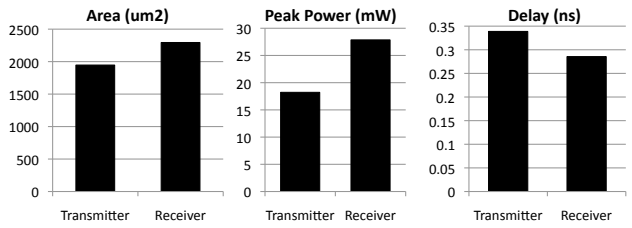

Figure 17: Synthesis results for DESC transmitter and receiver, each comprising 128 chunks.

5.2 Energy

Simulation results on the L2 cache energy are shown in Figure 16. Dynamic zero compression, bus invert coding, and zero skipped bus invert coding result in an average of 10%, 19%, and 20% reduction in the overall cache energy, respectively; basic DESC provides 11% cache energy savings, which is better than zero compression, but worse than bus invert coding. This is due to the minimum dynamic energy required on the interconnect, even for applications with few bit flips—CG, Cholesky, Equake, Radix, and Water-Nsquared.

Applying value skipping to DESC results in a significant energy reduction. Zero skipped DESC reduces overall cache energy by 1.81× on average, while last-value skipped DESC achieves a 1.77× energy reduction. Although the frequency of skipped values in the last-value skipped DESC is higher than that in the zero-skipped DESC (Figures 12 and 13), the latter shows superior energy efficiency because of the lower access energy. In the last-value skipped DESC, the cache controller needs to track the last values written to every mat. This requires extra storage at the cache controller to record the last value exchange with every mat, in addition to broadcasting data bits for each write operation across subbanks (through the vertical and horizontal H-trees, as shown in Figure 7). As a result, communication in the last-value skipped DESC consumes higher energy.

Figure 18 shows static and dynamic energy components of the L2 cache for different data transfer techniques, averaged over all applications. The results show that the superior energy efficiency of zero skipped DESC is achieved through a 2× reduction in dynamic cache energy, even though it adds a 3% static energy overhead to the baseline conventional cache.

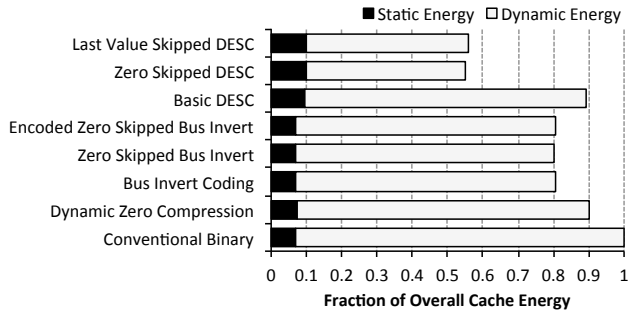

Figure 18: Contribution of dynamic and static energy components to the overall L2 cache energy for different data transfer techniques.

Figure 19 shows the impact of zero skipped DESC on overall system energy; the entire processor consumes 7% less energy with DESC.

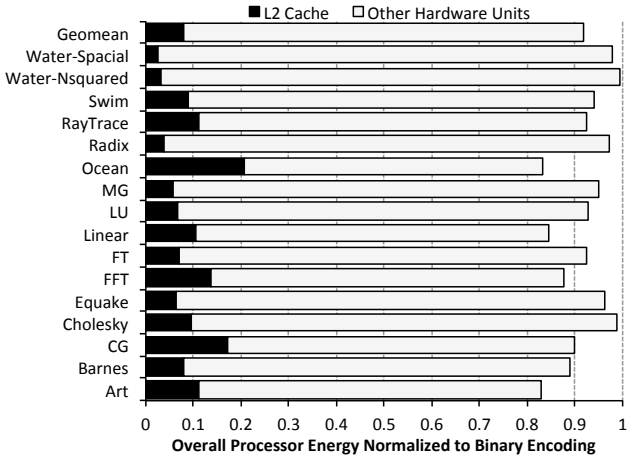

Figure 19: Overall processor energy reduction achieved by applying zero skipped DESC to the L2 cache.

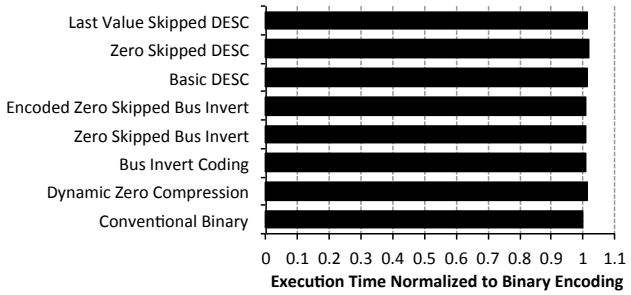

Figure 20: Performance of different data communication schemes.

5.3 Performance

We evaluate the performance of several data communication shemes: (1) conventional binary encoding; (2) dynamic zero compression; (3) bus-invert coding, with and without zero skipping; (4) DESC, and its variants with zero- and last-value skipping (Figure 20). The results show that the zero skipped and last-value skipped DESC techniques add less than 2% overhead to the overall execution time. (As shown in Figure 20, the zero compression and bus invert coding baselines also add a 1% overhead to the overall execution time due to the extra wires.) This performance loss is due to the longer access latency of the L2 cache with DESC. We further analyze the performance of zero skipped DESC by measuring its impact on the average access latency. DESC is not applied to the address and control wires, and thus has no effect on the miss penalty. However, the hit time includes the transfer latency of a cache block, which is determined by the chunk values in DESC. Experiments show that the average value transferred by the zero skipped DESC is approximately five. This value determines the throughput of each bank in DESC, and constitutes an extra latency in addition to the latency of the H-trees and internal DESC circuitry.

Figure 21 shows the average hit time of the conventional binary encoding and zero skipped DESC when applied to 64- and 128-wire data buses. Zero skipped DESC increases the average hit time by 31.2 and 8.45 cycles, respectively, for 64- and 128-wire buses. The corresponding slowdowns are 10% for a 64-wire data bus, and 2% for a 128-wire data bus.

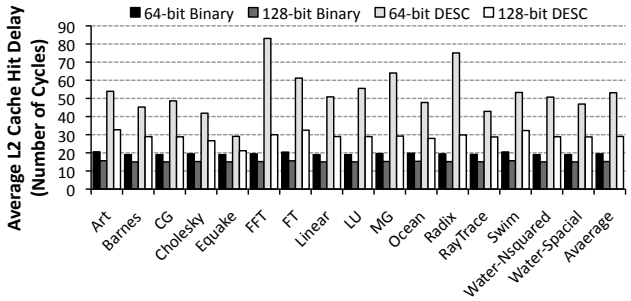

Figure 21: Average hit delay of the L2 cache with conventional binary encoding and DESC.

5.4 Exploring the Design Space: Last-Level Cache Energy and Delay

DESC expands the last-level cache design space for higher energy efficiency. Figure 22 shows the design possibilities when using conventional binary encoding and last-value skipped DESC data transfer mechanisms. The cache size is fixed at 8MB in both configurations; the data access width and the number of banks for both techniques are varied to generate the data. (In addition, the chunk size is varied for DESC.) Delay and energy are averaged over all sixteen applications, and normalized to the baseline L2 cache (8MB, 8 banks, and a 64-bit data bus). The plot shows that applying DESC to the L2 cache creates new design possibilities with higher energy efficiency, without significantly increasing the access latency.

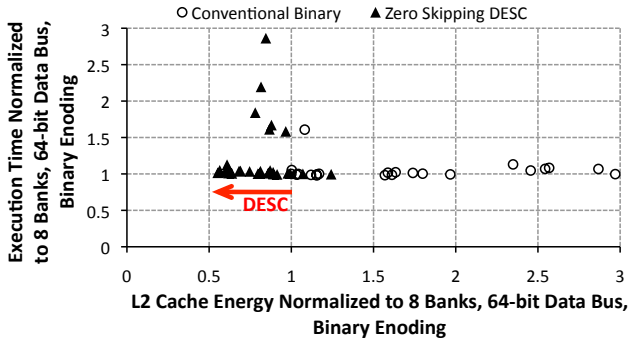

Figure 22: Cache design space possibilities when using conventional binary encoding and last-value skipped DESC techniques.

5.5 Application to S-NUCA-1

We apply DESC to an 8MB S-NUCA-1 cache comprising 128 banks, each of which has a 128-bit port. Cache ports are statically routed to the cache controller without using any switches. The access latency of the banks ranges from 3 to 13 core cycles. Figure 23 shows the relative execution

time averaged over the sixteen applications for an S-NUCA-1 cache with zero skipped DESC. The results show that the zero skipped DESC incurs an execution time penalty of 1% over conventional binary encoding.

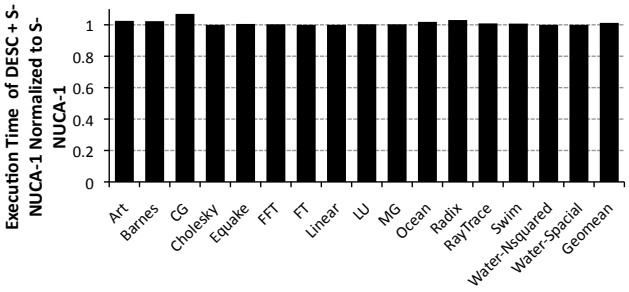

Figure 23: Normalized execution time when DESC is applied to an 8MB S-NUCA-1 cache.

As shown in Figure 24, DESC improves the energy of the S-NUCA-1 cache. The results indicate that zero skipped DESC improves cache energy by 1.62×, reduces the average L2 power by 1.64×, and improves the L2 energy-delay product by 1.59×.

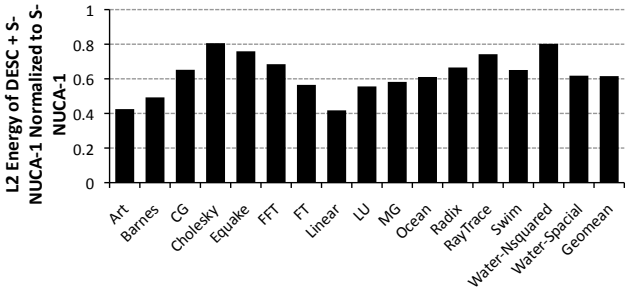

Figure 24: The energy of an 8MB S-NUCA-1 cache with zero skipped DESC.

5.6 Sensitivity Analysis

We study the sensitivity of DESC to the number of cache banks, chunk size, and L2 size.

5.6.1 *Number of Banks*

Figure 25 shows the relative energy and execution time averaged over all applications. Increasing the number of cache banks from one to two results in a significant reduction in execution time (due to a significant reduction in bank conflicts), and yields a commensurate improvement in the energy-delay product. As the number of banks increases from two to eight, both L2 energy and execution time improve slightly, reaching a minimum at eight banks; from eight to 64 banks, however, the energy-delay product increases due to the higher energy overhead of the individual banks and the DESC circuitry.

5.6.2 *Chunk Size*

We study the sensitivity of DESC to the chunk size parameter by measuring the L2 energy and total execution time for chunk sizes ranging from one to eight, when the cache capacity, cache block size, and the number of banks are fixed at 8MB, 512 bits, and 8 banks, respectively. This

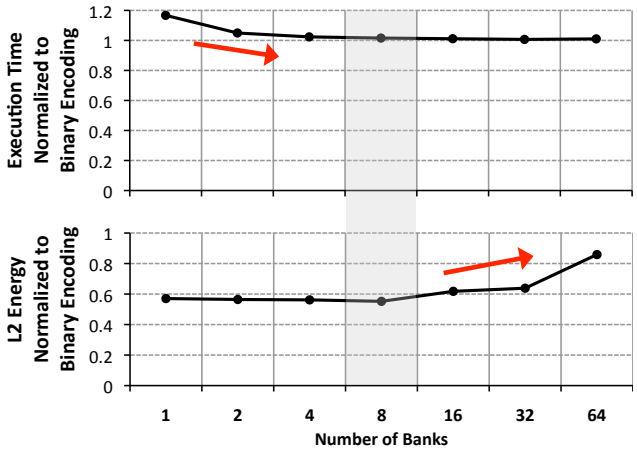

Figure 25: Energy and execution time sensitivity to the number of banks.

experiment considers multiple possibilities for transferring fixed-size cache blocks, employing buses with the number of data wires ranging from 32 to 256. Figure 26 shows the L2 energy and the total execution time normalized to the baseline (conventional) L2 cache for all studied chunk sizes. As the chunk size increases from one to eight, the range of the values that can be represented by each chunk increases; however, the transmission time increases as well. Each wire can now transfer a larger value, which results in a longer latency, but exhibits lower dynamic energy due to fewer transitions. This explains why the relative energy decreases and the execution time increases when moving from one-bit to four-bit chunks (Figure 26). Employing eight-bit chunks, however, results in higher energy and execution time due to the long transfer delay and the attendant increase in leakage energy. A chunk size of four with 128 wires results in the best energy-delay product at the L2 cache.

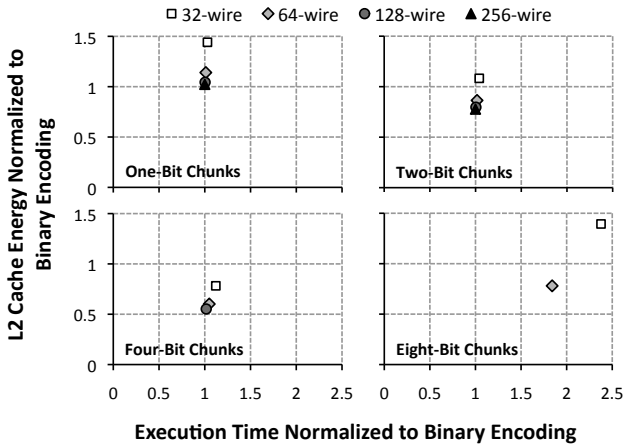

Figure 26: Energy and execution time sensitivity of zero skipped DESC to chunk size.

5.6.3 *Cache Size*

Experiments show that the size of the L2 cache has an impact on both the system power and the execution time (and consequently, energy). As the L2 capacity increases, cache power increases due to more cells and higher access energy;

however, the total execution time decreases because of fewer cache misses. For the evaluated multicore design, most of the L2 dynamic energy is consumed in the H-trees, because the LSTP transistors significantly reduce the static power of the arrays and the peripheral circuits. Figure 27 shows the impact of capacity on the L2 energy when the cache size ranges from 512KB to 64MB. The results indicate that increasing the cache size results in higher energy consumption for both conventional binary and DESC. However, DESC improves the average cache energy by 1.87× to 1.75×, when the cache size ranges from 512K to 64MB.

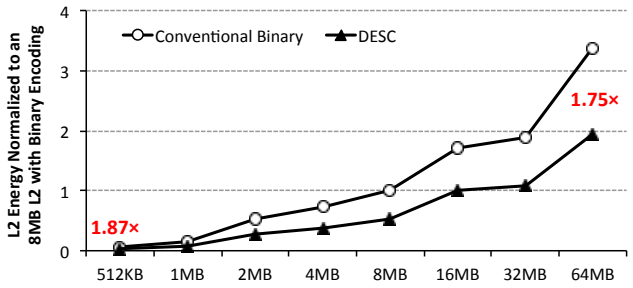

Figure 27: The impact of L2 capacity on cache energy.

5.7 Performance and Energy Characterization under SECDED ECC

We apply DESC to an 8MB cache comprising eight banks protected by SECDED ECC. We evaluate the execution time and the L2 energy when the (72, 64) and (137, 128) Hamming codes are applied with binary encoding and DESC. As shown in Figure 28, the zero skipped DESC incurs an average execution time penalty of 1% over conventional binary encoding.

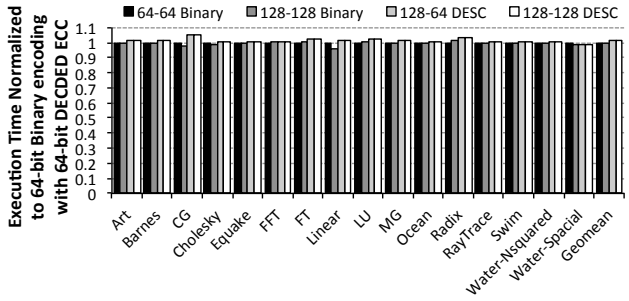

Figure 28: Normalized execution time for binary encoding and DESC for various configurations (W-S), where W represents the number of data wires and S is the segment size to which the Hamming code is applied.

Figure 29 shows that the zero skipped DESC improves cache energy by 1.82× and 1.92× for the (72, 64) and (137, 128) Hamming codes, respectively.

5.8 Impact on Latency Tolerance

Unlike a throughput-oriented design such as the multicore system with multithreaded cores evaluated in Sections 5.3 - 5.7, a latency sensitive design may suffer from a significant performance degradation when using DESC. To study the

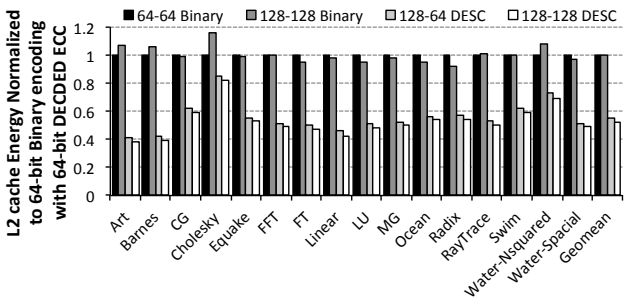

Figure 29: Normalized L2 energy for binary encoding and DESC for various configurations (W-S), where W represents the number of data wires and S is the segment size to which the Hamming code is applied.

effects of the long data transfer latency of DESC on such systems, we model an out-of-order processor running single threaded applications from the SPEC CPU 2006 suite [44]. The simulation results indicate that applying DESC to the L2 increases the execution time by 6% on average (Figure 30).

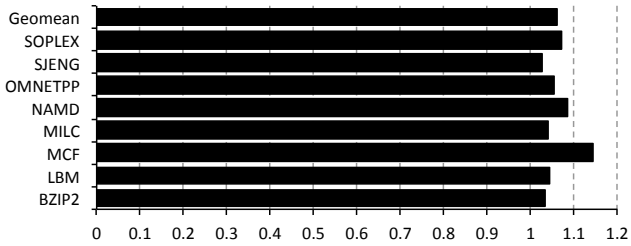

Figure 30: Execution time normalized to binary encoding.

6. CONCLUSIONS

We have presented DESC, an energy-efficient data exchange technique using synchronized counters. DESC creates new opportunities for designing low power on-chip interconnects. We have seen that applying DESC to the L2 cache can reduce overall cache energy by 1.81× by lowering dynamic power at the cache H-tree. This results in 7% savings in overall processor energy at a cost of less than 2% increase in the overall execution time, and a negligible area overhead. We have also observed that DESC achieves better power improvements compared to existing work on bus-invert coding and dynamic zero compression by exploiting more efficient mechanisms for skipping redundant values. We conclude that DESC holds the potential to significantly improve the energy efficiency of on-chip communication in future computer systems.

7. ACKNOWLEDGMENTS

The authors would like to thank anonymous reviewers for useful feedback. This work was supported in part by NSF grant CCF-1217418.

8. REFERENCES

[1] Nir Magen, Avinoam Kolodny, Uri Weiser, and Nachum Shamir. Interconnect-power dissipation in a microprocessor.

International Workshop on System Level Interconnect Prediction, 2004.

[2] A.N. Udipi, N. Muralimanohar, and R. Balasubramonian. Non-uniform power access in large caches with low-swing wires. *International Conference on High Performance Computing*, 2009.

[3] A. Lambrechts, P. Raghavan, M. Jayapala, F. Catthoor, and D. Verkest. Energy-aware interconnect optimization for a coarse grained reconfigurable processor. *International Conference on VLSI Design*, 2008.

[4] G. Chandra, P. Kapur, and K.C. Saraswat. Scaling trends for the on chip power dissipation. *International Interconnect Technology Conference*, 2002.

[5] Nikos Hardavellas, Michael Ferdman, Anastasia Ailamaki, and Babak Falsafi. Power scaling: the ultimate obstacle to 1k-core chips. *Technical Report NWU-EECS-10-05*, 2010.

[6] Bradford M. Beckmann and David A. Wood. TLC: Transmission line caches. *International Symposium on Microarchitecture*, 2003.

[7] Hui Zhang and J. Rabaey. Low-swing interconnect interface circuits. *International Symposium on Low Power Electronics and Design*, 1998.

[8] Changkyu Kim, Doug Burger, and Stephen W. Keckler. An adaptive, non-uniform cache structure for wire-delay dominated on-chip caches. *International Conference on Architectural Support for Programming Languages and Operating Systems*, 2002.

[9] Daniel Citron. Exploiting low entropy to reduce wire delay. *IEEE Computer Architecture Letters*, 2004.

[10] A. Seznec. Decoupled sectored caches: conciliating low tag implementation cost. In *International Symposium on Computer Architecture*, 1994.

[11] Julien Dusser, Thomas Piquet, and André Seznec. Zero-Content Augmented Caches. Rapport de recherche RR-6705, INRIA, 2008.

[12] Luis Villa, Michael Zhang, and Krste Asanovic. Dynamic zero compression for cache energy reduction. *International Symposium on Microarchitecture*, 2000.

[13] Rajeev Balasubramonian, Naveen Muralimanohar, Karthik Ramani, and Venkatanand Venkatachalapathy. Microarchitectural wire management for performance and power in partitioned architectures. *International Symposium on High-Performance Computer Architecture*, 2005.

[14] N. Muralimanohar, R. Balasubramonian, and N. Jouppi. Optimizing NUCA organizations and wiring alternatives for large caches with CACTI 6.0. *International Symposium on Microarchitecture*, 2007.

[15] Mircea R. Stan and Wayne P. Burleson. Bus-invert coding for low-power I/O. *IEEE Transactions on VLSI Systems*, 1995.

[16] S. Kaxiras, Z. Hu, and M. Martonosi. Cache decay: Exploiting generational behavior to reduce cache leakage power. *International Symposium on Computer Architecture*, 2001.

[17] K. Flautner, N. S. Kim, S. Martin, D. Blaauw, and T. Mudge. Drowsy caches: simple techniques for reducing leakage power. *International Symposium on Computer Architecture*, 2002.

[18] Nam Sung Kim, David Blaauw, and Trevor Mudge. Leakage power optimization techniques for ultra deep sub-micron multi-level caches. *International Conference on Computer-Aided Design*, 2003.

[19] J. G. Proakis. *Digital Communications*. Third Edition, McGraw-Hill, 1995.

[20] K. Itoh, K. Sasaki, and Y. Nakagome. Trends in low- power RAM circuit technologies. *Symposium on Low Power Electronics*, 1994.

[21] J. Wuu, D. Weiss, C. Morganti, and M. Dreesen. The asynchronous 24MB on-chip level-3 cache for a dual-core Itanium®-family processor. *International Solid-State Circuits Conference*, 2005.

[22] C.W. Slayman. Cache and memory error detection, correction, and reduction techniques for terrestrial servers and workstations. *IEEE Transactions on Device and Materials Reliability*, 2005.

[23] Y.Q. Shi, Xi Min Zhang, Zhi-Cheng Ni, and N. Ansari. Interleaving for combating bursts of errors. *IEEE Circuits and Systems Magazine*, 2004.

[24] Doe Hyun Yoon and Mattan Erez. Memory mapped ecc: Low-cost error protection for last level caches. *International Symposium on Computer Architecture*, 2009.

[25] Jose Renau et al. SESC simulator, Jan. 2005. `http://sesc.sourceforge.net`.

[26] Sheng Li, Jung Ho Ahn, Richard D. Strong, Jay B. Brockman, Dean M. Tullsen, and Norman P. Jouppi. McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures. *International Symposium on Computer Architecture*, 2009.

[27] K. Kanda, T. Miyazaki, Min Kyeong Sik, H. Kawaguchi, and T. Sakurai. Two orders of magnitude leakage power reduction of low voltage SRAMs by row-by-row dynamic VDD control (RRDV) scheme. *International ASIC/SOC Conference*, 2002.

[28] S. Rusu, S. Tam, H. Muljono, D. Ayers, and J. Chang. A dual-core multi-threaded xeon processor with 16MB L3 cache. *International Solid-State Circuits Conference*, 2006.

[29] Varghese George, Sanjeev Jahagirdar, Chao Tong, K. Smits, Satish Damaraju, Scott Siers, Ves Naydenov, Tanveer Khondker, Sanjib Sarkar, and Puneet Singh. Penryn: 45-nm next generation Intel Core 2 processor. *Asian Solid-State Circuits Conference*, 2007.

[30] D. James. Intel Ivy Bridge unveiled - the first commercial tri-gate, high-k, metal-gate CPU. *Custom Integrated Circuits Conference*, 2012.

[31] E. Karl, Yih Wang, Yong-Gee Ng, Zheng Guo, F. Hamzaoglu, U. Bhattacharya, K. Zhang, K. Mistry, and M. Bohr. A 4.6GHz 162MB SRAM design in 22nm tri-gate CMOS technology with integrated active vmin-enhancing assist circuitry. *International Solid-State Circuits Conference*, 2012.

[32] N. Maeda, S. Komatsu, M. Morimoto, and Y. Shimazaki. A 0.41 ua standby leakage 32kb embedded SRAM with low-voltage resume-standby utilizing all digital current comparator in 28nm hkmg CMOS. *Symposium on VLSI Circuits*, 2012.

[33] Masaki Fujigaya, Noriaki Sakamoto, Takao Koike, Takahiro Irita, Kohei Wakahara, Tsugio Matsuyama, Keiji Hasegawa, Toshiharu Saito, Akira Fukuda, Kaname Teranishi, Kazuki Fukuoka, Noriaki Maeda, Koji Nii, Takeshi Kataoka, and Toshihiro Hattori. A 28nm high-k metal-gate single-chip communications processor with 1.5GHz dual-core application processor and LTE/HSPA+-capable baseband processor. *International Solid-State Circuits Conference*, 2013.

[34] Fumihiko Tachibana, Osamu Hirabayashi, Yasuhisa Takeyama, Miyako Shizuno, Atsushi Kawasumi, Keiichi Kushida, Azuma Suzuki, Yusuke Niki, Shinichi Sasaki, Tomoaki Yabe, and Yasuo Unekawa. A 27% active and 85% standby power reduction in dual-power-supply SRAM using BL power calculator and digitally controllable retention circuit. *International Solid-State Circuits*, 2013.

[35] Richard M. Yoo, Anthony Romano, and Christos Kozyrakis. Phoenix rebirth: Scalable MapReduce on a large-scale shared-memory system. *International Symposium on Workload Characterization*, 2009.

[36] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta. The SPLASH-2 programs: Characterization and methodological considerations. In *International Symposium on Computer Architecture*, 1995.

[37] L. Dagum and R. Menon. OpenMP: An industry-standard API for shared-memory programming. *IEEE Computational Science and Engineering*, 1998.

[38] D. H. Bailey et al. NAS parallel benchmarks. Technical report RNR-94-007., NASA Ames Research Center, 1994.

[39] Greg Hamerly, Erez Perelman, Jeremy Lau, and Brad Calder. Simpoint 3.0: Faster and more flexible program analysis. *Journal of Instruction Level Parallelism*, 2005.

[40] Encounter RTL compiler. `http://www.cadence.com/products/ld/rtl_compiler/`.

[41] Free PDK 45nm open-access based PDK for the 45nm technology node. `http://www.eda.ncsu.edu/wiki/FreePDK`.

[42] ITRS. *International Technology Roadmap for Semiconductors*. `http://www.itrs.net/links/2010itrs/home2010.htm`.

[43] Wei Zhao and Yu Cao. New generation of predictive technology model for sub-45nm design exploration. *International Symposium on Quality Electronic Design*, 2006.

[44] John L. Henning. SPEC CPU2006 benchmark descriptions. *SIGARCH Computer Architecture News*, 2006.