STFL: Energy-Efficient Data Movement with Slow Transition Fast Level Signaling

Payman Behnam
University of Utah
behnam@cs.utah.edu

Mahdi Nazm Bojnordi
University of Utah
bojnordi@cs.utah.edu

ABSTRACT

Data movement in large caches consumes a significant amount of energy in modern computer systems. Low power interfaces have been proposed to address this problem. Unfortunately, the energy-efficiency of these techniques is largely limited due to undue latency overheads of low power wires and complex coding mechanisms. This paper proposes a hybrid technique for slow-transition, fast-level (STFL) signaling that creates a balance between power and bandwidth in the last level cache interface. Combined with STFL codes, the signaling technique significantly mitigates the performance impacts of low power wires, thereby improving the energy efficiency of data movement in memory systems. When applied to the last level cache of a contemporary multicore system, STFL improves the CPU energy-delay product by 9% as compared to a voltage-frequency scaled baseline. Moreover, the proposed architecture reduces the CPU energy by 26% and achieves 98% of the performance provided by a high-performance baseline.

ACM Reference Format:
Payman Behnam and Mahdi Nazm Bojnordi. 2019. STFL: Energy-Efficient Data Movement with Slow Transition Fast Level Signaling. In *The 56th Annual Design Automation Conference 2019 (DAC '19), June 2–6, 2019, Las Vegas, NV, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3316781.3317819

1 INTRODUCTION

Contemporary microprocessors employ last level caches that occupy significant die area, and incur large energy and delay overheads due to exchanging data over long capacitive wires. The last level cache and DRAM IO together dissipate 33% of the overall system energy.This has been the motivation behind numerous architectural mechanisms recently proposed for decreasing the data movement energy in last level cache and DRAM interface. Despite the existing proposals, current technology scaling trends indicate that data movement energy will be even a more serious problem in future computer systems [1].

Data movement on wires is typically carried out through signal transitions between two voltage levels *high* and *low* that differentiate the logical **1** and **0**. The maximum number of possible transitions per second that can occur on a wire determines the data movement bandwidth. The switching activity on the wires results in dynamic power dissipation. To reduce power consumption, one can decrease the switching activity or down-scale the swing voltage between **0** and **1**. Both these techniques may constrain bandwidth and degrade performance. Therefore, a careful balance between power and bandwidth must be struck to achieve the highest energy-efficiency in last level caches.

This paper presents an architectural solution to achieve a higher bandwidth in low power wires. As the transition speed is the key bandwidth bottleneck in low power wires, the goal is to reduce the number of transitions in every data block and employ a technique that transfers signal levels faster. To the best our knowledge, STFL is the first architectural technique for hybrid signaling on wires of cache interfaces. The proposed hybrid signaling supported by proper data encoding mechanisms becomes a suitable bandwidth optimization to low-power wires, thereby creating new opportunities for system designers to further optimize energy-efficiency of computer systems.

2 BACKGROUND AND MOTIVATION

Power Dissipation in On-Chip Wires. Figure 1 illustrates the trade-off between delay and supply voltage in three wire repeaters with 1X, 2X, and 4X strengths to drive capacitive wires of different sizes (4X is stronger). We model the repeaters at the 22nm CMOS technology node after the interconnect models used for last level cache in CACTI 6.5 [2]. As the supply voltage decreases from 0.7V to 0.48V, the delay increases by more than a 100%. This voltage reduction results in lowering both static and dynamic power of the interconnect; on the other hand, it may result in significant performance degradation.

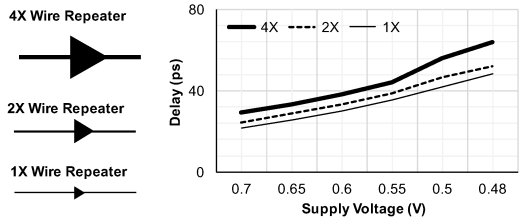

Figure 1: The impact of voltage scaling on wire repeaters at the 22*nm* CMOS technology node.

Energy-Efficient Data Encoding. Data encoding has been widely adopted in LLCs and memory interfaces to reduce power consumption by lowering the bus activity and termination current. Bus invert coding was first proposed by Stan *et al.* [3] to lower the dynamic power dissipation in data wires. The basic mechanism transfers either the true or complement of every data block that results in less switching activity, thereby reducing the peak dynamic power.

DAC '19, June 2–6, 2019, Las Vegas, NV, USA

ACM ISBN 978-1-4503-6725-7/19/06…$15.00
https://doi.org/10.1145/3316781.3317819

Later, a variant of this technique, called data bus inversion (DBI) [4], was applied to open drain interfaces to reduce the data movement power by lowering the Hamming weight of the transferred data. Recently, a cost aware flip optimization technique (CAFO) [5] has been proposed for asymmetric memories that reduces the number of **1** in a data block.

k limited weight codes (*k*-LWC) [6] belong to a sparse data encoding class, where the codewords have a Hamming weight of no more than *k*. For example, the 3-LWC maps every 11-bit data to a 23-bit codeword with at most three **1**s. A variation of the limited weight codes, called SETS, is proposed by Song et al. [7] to reduce bit-flips in last level caches. This method realizes an example of sparse codes that consumes 4× more wires to represent one-hot coded data transferred between the cache banks and the controller.

History based techniques have been explored to exploit the similarities between the current and old data blocks to reduce bus activity. BD encoding [8] compares the data to be sent over the data bus with previously transferred blocks. Based on the outcome of this comparison, only the difference between the data and the most similar block along with its index are sent to the receiver. DESC [9] employs synchronized counters at the receiver and transmitter to represent data in terms of delay cycles between subsequent transitions. As a result, this mechanism can significantly reduce the switching activity of the wires at the cost of longer transmission times. The work in [10] proposes an adaptive approach that monitors characteristics of applications and employs proper time-based codes for LLC interconnects to reduce energy consumption considerably.

3 STFL SIGNALING

To balance power and bandwidth in memory interfaces, the proposed coding scheme exploits low power wires to achieve a lower per-bit energy, and applies a novel STFL coding technique that increases the throughput of those wires. On every STFL data transfer, a *transmitter* encodes and sends information to a *receiver* over a set of low power wires. STFL replaces the conventional voltage level signaling with the transition signaling that makes it possible to directly control the wire flips via encoding. (Instead of representing **1**s with a high voltage level (V_{DD}) and **0**s with ground, every **1** is signaled using a transition between V_{DD} and ground; while, **0**s represented with the absence of transitions.) The proposed STFL transmitter is connected to low power wires and sends the data bits at the same rate of a high performance wire, which may result in signal deterioration if consecutive transitions (i.e., **1**s) are transferred. To avoid data loss due to the signal deterioration, the STFL transmitter needs to pause the transmission by injecting delay cycles after every transition (i.e., **1**). Following the same convention, the STFL receiver samples the low power wires at the high performance rate and removes the corresponding samples to those delay cycles inserted by the transmitter.

Design Principles. Figure 2 shows an illustrative example of data transmission using high speed wire, low power wire, and STFL interface. In this example, transition signaling is used to transfer four bits of data on a single wire. The high speed wire provides the fastest transmission time (t) at the cost of consuming more power, whereas the low power wire is able to decrease power consumption at the cost of doubling the transmission time ($2t$). STFL employs low power wire to keep the transmission power low; it injects a delay cycle (**D**) after every **1** in the original data to create new STFL codewords; and transfers them at the same rate as in the high speed wire. Therefore, STFL can reduce the transition time down to $1.5t$. This can be viewed as a hybrid data transmission technique that transfers **0**s at high speed and **1**s at low power. As a result of optimizing both power and time, STFL is now able to improve the energy efficiency of data transmission compared to the other two techniques.

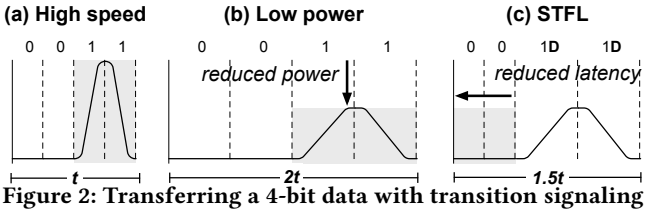

Figure 2: Transferring a 4-bit data with transition signaling on high speed wire (a), low power wire (b), and STFL interface (c).

Design Challenges. One difficulty in realizing the proposed STFL coding is the transmission time that increases with respect to the number of **1**s in the data—*a.k.a.*, the Hamming weight. For example, due to the additional delay cycles (**D**s), transferring an all **1** pattern requires double the transmission time of an all **0** block. (The longest transition time is the same as that of the low power technique.) STFL addresses this problem by leveraging encoding techniques that limit the number of transferred **1**s per transmission.

4 APPLYING STFL TO LARGE CACHES

Large caches are typically organized as a hierarchy of banks, sub-banks, mats, sub-arrays, and multiple H-trees that are disciplined by a cache controller. Independent banks are accessed simultaneously through a bank-level H-tree; each bank comprises a group of sub-banks that share the wires of a vertical H-tree; within every sub-bank, multiple mats are connected to a horizontal H-tree and supply different bits of the cache block in a bit parallel fashion. Every read and write access requires moving data over long and capacitive wires within the H-trees, which results in significant delay and power consumption [2, 9, 11]. STFL reduces the overall data movement energy in the cache interconnects through (1) using low power wires in data H-trees, and (2) integrating a set of STFL transmitters and receivers in the mats and cache controller to perform data transmission.

We apply STFL to the input and output data buses transferring a cache block between the cache controller and the selected mats during every cache access. Figure 3 depicts transferring a 64-byte cache block using an STFL interface with 16 groups. STFL divides every cache block into multiple groups of four bytes. Each group is converted to four STFL codewords transferred over four low power data wires. STFL employs an existing low power wire to transfer the encoding modes used for the four codewords. Finally, the receiver detects the signals and converts the codes to the original data block.

4.1 Data Encoding with STFL

The proposed STFL mechanism exploits the similarities between adjacent bytes (i.e., spatial locality) in every cache block to reduce

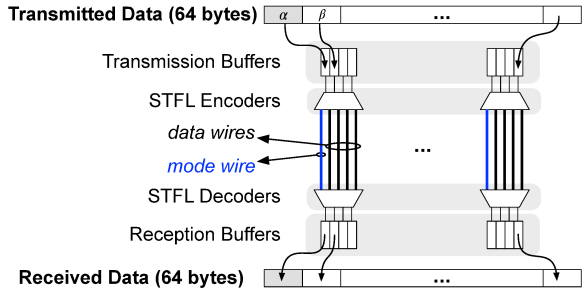

Figure 3: Illustrative example of transferring a 64-byte cache block using the STFL interface.

the Hamming weight of the codewords. This optimization is implemented through defining multiple encoding modes for every byte (α) to be transferred. The STFL encoder estimates the energy and delay costs for all of the possible codewords through computing the Hamming weight (Φ) of each candidate. Therefore, STFL selects the codeword with less Hamming weight to be transferred for the data byte. Table 1 shows the three possible encoding modes and the corresponding codewords for every α. The mode is set to 000 if the original data (α) is selected as the codeword. This mode is useful for transferring low Hamming weight bytes, such as 00000000.[1] STFL employs mode 01**D** for transferring the inverted data ($\overline{\alpha}$) to reduce the number of **1**s in heavy bytes, such as 11111111. The 1**D**0 mode is used for transferring the difference between α and its adjacent byte β within the same group. Notice that the mode bits of each group are serially transferred on a low power wire, thereby requiring a **D** after every **1**.

Table 1: STFL encoding.

Condition	Codeword	Mode
$(\Phi(\alpha) \leqslant 4) \wedge (\Phi(\alpha \oplus \beta) < \Phi(\alpha))$	$\alpha \oplus \beta$	1D0
$(\Phi(\alpha) > 4) \wedge (\Phi(\alpha \oplus \beta) \geqslant \Phi(\overline{\alpha}))$	$\overline{\alpha}$	01**D**
Otherwise	α	000

To avoid delay and energy overheads, STFL limits the XOR coding in mode 1**D**0 to every data group only. The rightmost byte of each group may be XORed with a fixed constant value (01010101) rather than its adjacent byte. Figure 4 illustrates the proposed encoding mechanism for STFL. The encoder employs two population counters and a simple encoding logic to prepare data prior to transmission on a data wire. Based on Table 1, the logic generates three mode bits indicating which encoding is applied to the data (α). STFL generates a total of 12 mode bits for all of the bytes in every data group and transmits them using a single mode wire. Similarly, the decoder employs Table 1 to convert the received codewords into the original data. In addition to the encoder and decoder units, STLF employs a transmitter and a receiver to generate and detect the corresponding signals with every codeword.

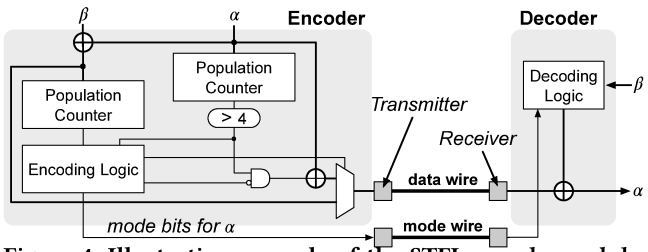

Figure 4: Illustrative example of the STFL encoder and decoder.

[1]Prior work [9] shows that about 30% of the transferred bytes may be zero.

4.2 Low Power Signaling with STFL

STFL builds upon existing low-power signaling techniques and on-chip interconnects. Low voltage swing wires are used to decrease power consumption at the cost of a delay in the circuit due to the increase in setup and hold time during transitions, which impacts the interconnect bandwidth. Figure 5(a) shows the level converter circuits used in STFL, which are designed based on the prior work in internal bus architectures and low power interconnects [11–13].

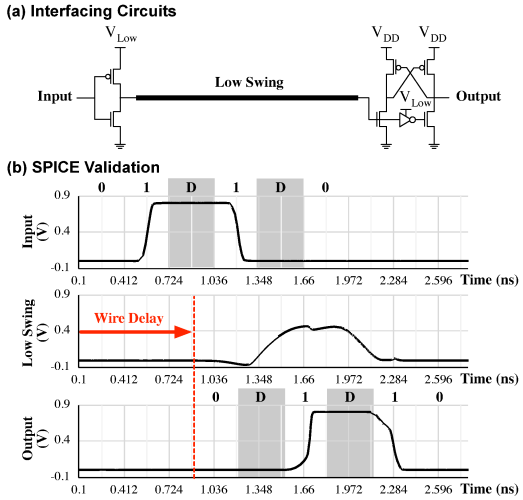

Figure 5: STFL interfacing circuits.

Figure 5(b) shows SPICE validation of the required STFL signaling for a four-bit data (0110) at the 22nm technology node. Due to the lower bandwidth of the wire, STFL translates every transition from the full-swing domain into a rise or fall in the low swing domain spanning two cycles. The low swing signals travel the wire and arrive at the receiver after a certain *wire delay*. Due to the delay of signal conversion at the receiver (T_c) and the transition time in the low swing domain ($T_l = 2$), toggling the output latch on a low swing transition is delayed by one cycle ($T_{out} = T_l/2 + T_c = 1 + T_c$).[2] Unlike the input stream, the delay cycles appear before transitions (**1**s) in the output. Moreover, STFL detects every delay cycle as a dummy **0** before a **1** at the receiver, which has to be removed before recovering the original data. For example, 01D1D0 will be received as 001010 at the receiver. To further validate signal integrity of the circuit, Figure 6 shows the eye diagrams plotted for three clock periods of the low swing and output signals when transferring a stream of 10000 **0**s and **1**s.[3] We consider a $\pm 20mV$ noise on the Vdd, Gnd, and low swing Vdd lines for this experiment.

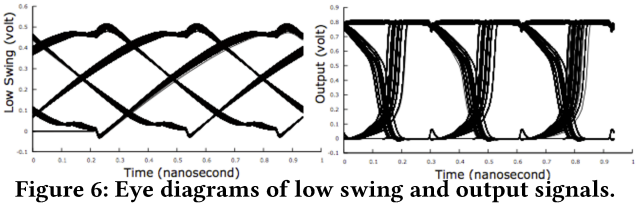

Figure 6: Eye diagrams of low swing and output signals.

Overall, STFL trades speed for power efficiency by lowering the voltage supplied to the communication wires (recall Figure 1).

[2]Assuming that a transition is detected within 35 – 65% of the voltage swing, T_c ensures the one cycle delay before a transition being captured by the output latch.
[3]We observed a repeated 01 stream to be the worst case.

Therefore, STFL is able to reduce both the static and dynamic power dissipation in the cache interface. Transferring a codeword via the proposed signaling technique requires a transmitter and a receiver. Figure 7 shows how an 8-bit codeword is transferred over an example STFL interface that includes three mechanisms for transmitting codewords, receiving signals, and transferring mode bits.

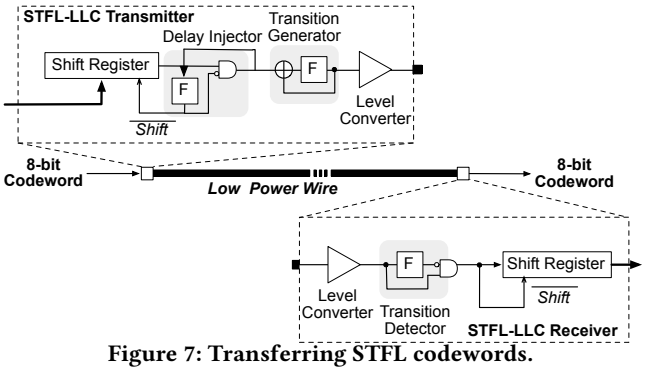

Figure 7: Transferring STFL codewords.

Transmitting Encoded Data. To ensure the transition signals are properly generated for each codeword, multiple steps are followed by the STFL transmitter. First, STFL stores the eight-bit codeword generated by the encoder in a parallel-in, serial-out shift register. (Due to using the encoding modes as explained in Table 1, it is guaranteed that every codeword contains no more than four **1**s.) STFL reads the code bits serially from the shift register and converts them to transition signals using a transition generator comprising a latch and an XOR gate. A delay injector controls the shift register and maintains the previous output of the shift register. The delay injector is connected to the shift register via an (active low) shift signal; every **1** transmitted in the previous cycle disables the shift register in the current cycle, thereby injecting a **D** after every **1** in the code. Since the shift register can now contain up to four **1**s, the longest generated codeword is 12 bits long. To avoid the complexity of variable length encoding, the transmitter is set to produce fixed 12-bit codes (zero padding is required for the codes with fewer **1**s). The STFL codes are serially fed into a transition generator circuit that translates every **1** into a flip on its output. Finally, STFL employs the level converter to prepare the signals prior to transmission on the low-power wires by converting from full to low-swing.

Transmitting Mode Bits. Unlike codewords, mode bits can be directly converted to the transition signals on the wire with no need for delay injection. The STFL encoder generates a total of 12 mode bits for every four data bytes, where **1**s are spaced out by dummy **0**s in the resultant bit pattern. Similarly to the data codewords, transferring the mode bits requires 12 cycles.

Receiving STFL Signals. The STFL receiver makes use of a transition detector, consisting of an XOR gate and a flip-flop, to convert the transition signals into **1**s. From data wires, the result is sent to a serial-in, parallel-out shift register. On every cycle, a newly detected bit is fed to the shift register; moreover, the same bit controls the shift operation. Every **0** results in shifting the content and inserting the bit in the register; a **1**, however, disables the shift operation and overwrites the previously sampled value—which is a dummy **0**. Therefore, STFL removes all of the additional delay cycles by the transmitter at the receiver. Finally, the result is sent to the STFL decoder for extracting the original data block (Figure 4).

5 EVALUATIONS

5.1 Methodology

The area, delay, and power for the STFL encoders and decoders are based on hardware synthesis with the FreePDK [14] library at the 45nm CMOS technology, which are then scaled to 22nm. We create SPICE models using PTM [15] high-performance 22nm transistors for all of the interfacing circuits and perform circuit simulations to estimate energy and delay overheads. To make the interfacing circuits practical, we sized the transistors for a safe setup and hold time [16, 17]. Using McPAT [18], we estimate the overall processor power consumption. A heavily modified version of ESESC [19] is used to model the STFL interface in a multicore system that simulates 12 memory-intensive applications from various multi-threaded benchmark suites [20–22] (Table 2).

Table 2: Applications and data sets.

Label	Benchmarks	Input	Label	Benchmarks	Input
FT	Fourier Transform	Class A	LU	LU	1024 × 1024 Matrix
IS	Integer Sort	Class A	RAY	Ray Trace	car
MG	Multi-Grid	Class A	OCN	Ocean	514x514 ocean
CG	Conjugate Gradient	Class A	FFT	FFT	1048576 data points
BT	Block Tri-diagonal	Class A	BRN	Barnes	16K particles
HIST	Histogram	100MB file	WCNT	Word Count	10MB text file

We model the existing encoding techniques such as bus invert coding [3], time-based data representation with DESC [9], sparse encoding with SETS [7], history based BD encoding [8], and two-dimensional block coding with CAFO [5]. These encoding techniques are compared with the conventional binary encoding when applied to the last level cache interfaces. CAFO-LLC is applied to 8×8 bit data blocks. Due to the significant area, delay, and energy overheads introduced by large tables of BD encoding, (1) a smaller entry size is chosen for last level cache and (2) BD encoding is applied to the bank level H-trees of last level cache. Moreover, we model a voltage and frequency scaled (VFS) baseline for comparisons through exploring the application of binary encoding in low power wires with reduced frequency for the last level cache interface. This is accomplished via employing low voltage-swing wires in the H-trees of last level cache. Table 3 shows the simulation parameters for the evaluated systems.

Table 3: System parameters.

Core	four 4-issue OoO cores, 128 ROB entries, 3.2 GHz
IL1/DL1 cache	32KB, 4-way, LRU, 64B block, hit/miss delay 1/1
L2 cache (shared)	4MB, 8-way, LRU, 64B block, hit/miss delay 8/2, MESI protocol
Temperature	360 K (77 °C)
DDR4-2400	tRCD: 14.16, tCL: 13.32, tWL: 16, tCCD: 4, tWTR: 7.5, tWR: 12, tRTP: 7.5, tRP: 13.32, tRRD: 4, tRAS: 32, tRC: 45.32, tFAW: 30

Exploring the Cache Design Space. A modified version of CACTI 6.0 [23] is used to find the best configurations for last level caches using the baseline full swing and STFL interfaces. We employ energy-delay product (EDP) [24] as the energy-efficiency metric for finding the best cache configurations. We explore the design space of caches with and without the STFL interface by varying different cache parameters such as the number of banks, the data bus width, associativity, and the device types from ITRS high performance (HP) to

low standby power (LSTP) and low power (LOP) transistors [25].[4] We employ the parameters provided by the prior work [26] on interconnect optimization to estimate the delay and energy of the low power wires used for STFL.

5.2 Results

Synthesis. Table 4 shows the area overhead, critical path delay, and power consumption of the required encoders and decoders used for 64-bit interfaces using DBI, STFL, BD, CAFO, DESC, and SETS encoding. (The table does not show the overheads of wires.) Notice that the energy, delay, and area overheads of the encoders and decoders for processing data blocks are negligible as compared with those of the data wires and interfacing circuits. We implement each encoding technique while minimizing their direct and indirect impacts on the system efficiency. Overall, STFL logic consumes less area compared to CAFO and BD, while incurring an acceptable delay and power overheads. We expect the additional hardware impose negligible impact on die area and yield of modern microprocessors [27].

Table 4: Overheads of various encoders and decoders.

		DBI-LLC	STFL	BD-LLC	CAFO	DESC	SETS
Encoder Interface	Area (μm^2)	112.252	1642.816	2183.565	2638.72	1890.136	76.35
	Delay (ns)	0.197	0.336	0.74	0.705	0.34	0.16
	Power (mW)	0.24	1.86	0.13	1.41	18.4	15.3
Decoder Interface	Area (μm^2)	25.536	170.24	2183.565	51.072	2236.921	68.32
	Delay (ns)	0.016	0.046	0.24	0.033	0.28	0.157
	Power (mW)	0.71	1.62	0.28	0.71	27.6	15.0

Energy. Figure 8 shows the impact of various encoding mechanisms on the switching energy in the last level cache. (Additional energy consumed for encoding/decoding data blocks is included in the results.) The proposed STFL codes reduce the switching energy of the last level cache by an average of 60% across all 12 benchmark applications. This reduction is 4.3× of the energy savings achieved by DBI, which employs a simple coding logic. The STFL savings are at least 10% better than those gained by VFS, DESC, and SETS; however, these techniques expose significant indirect overheads to the system. Notice that CAFO requires complex encoders and decoders at the communication ends that result in large energy overheads, thereby making it ill-suited for the last level cache interface.

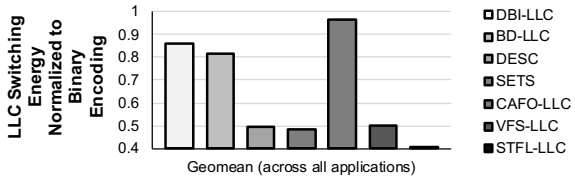

Figure 8: Total switching energy consumed by LLC.

Consuming less switching energy in the last level cache interface may lead to a reduction in the overall processor energy only if the encoding/decoding and indirect overheads are minimal. Figure 10 shows the overall processor energy when the proposed STFL and baseline interfaces are applied to the last level cache interface. STFL reduces the overall CPU energy by 27% averaged across all of the evaluated applications. This energy reduction is 1.2× of the average savings obtained by VFS, which is because of employing STFL codes to improve bandwidth and reduce switching activity. Similar to VFS, DESC and SETS are (even more significantly) impacted by the undue performance overheads of data encoding and signaling. The proposed STFL achieves an average of at least 21% processor energy reduction over the DESC and SETS baselines. BD encoding requires table lookups on every cache data transfer that increases the execution time and per access energy, thereby diminishing the energy benefits from switching reduction.

Performance. Figure 11 shows the relative system performance of STFL and the baseline encoding techniques applied to the last level cache. The results indicate that DESC and SETS suffer from significant performance loss due to the large bandwidth overheads consumed for signaling[5]. VFS employs low power wires and encounters an average of 6% performance loss, which also results in consuming more static system energy. In addition to reducing the overall system energy, STFL alleviates the adverse performance impacts of low power wires and achieves 98% of the performance gained by the high-performance binary encoding baseline.

Energy-Delay Product. Improving both energy and delay by STFL results in a superior energy-efficiency compared to all of the evaluated baselines. Our simulation results indicate that SETS, DESC, and CAFO result in a higher CPU energy-delay products. STFL and VFS, however, can significantly reduce the energy-delay products. STFL achieves 9% better CPU energy-delay product over VFS due to improving bandwidth and reducing the switching energy.

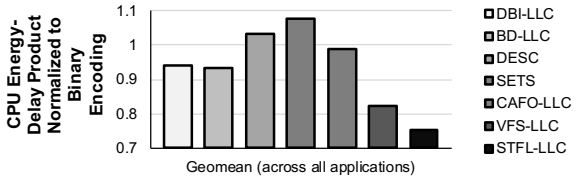

Figure 9: CPU energy-delay product.

Adapting to Random Data Patterns. As modern computer systems may adopt compression and encoding mechanisms in memory channels [28, 29], we study the impact of randomness in data patterns on the energy savings by DBI, BD, CAFO, VFS, and STFL interfaces. We develop a C/C++ program that writes a synthetic stream of random data to the memory system. A tunable parameter p is used to determine the probability of setting every bit of the data block to **1**. Figure 12 shows the relative switching energy of the last level cache normalized to the conventional binary encoding with $p = 0.1$. STFL provides superior energy savings due to the low power wires and the proposed encoding mechanism. For heavy blocks, where $p > 0.8$, VFS can save more switching energy than STFL due to the encoding/decoding overheads.

6 CONCLUSIONS

STFL is a hybrid technique for slow-transition, fast-level signaling that creates a balance between power and bandwidth in the last level cache interface. The proposed scheme create new opportunities for enhancing the energy efficiency of low power interfaces and designing efficient memory systems. As the need for data intensive computing is expected to further grow in future, the proposed

[4]Similar to prior work on DESC [9], our study indicates that using LSTP devices for the SRAM cells and LOP for the peripheral circuits can significantly reduce dynamic and static energy.

[5]This is mainly because of consuming more wires or longer transmission time.

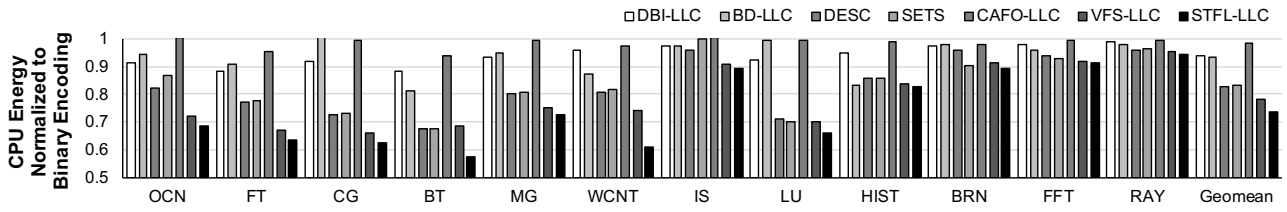

Figure 10: Overall processor energy with the cache optimization techniques.

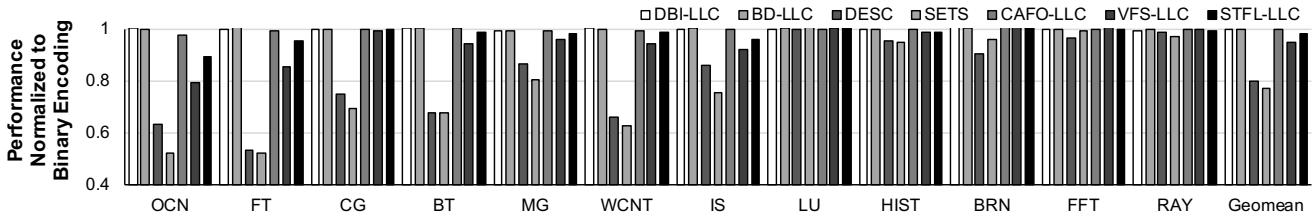

Figure 11: Relative performance of STFL and other encodings techniques applied to last level cache.

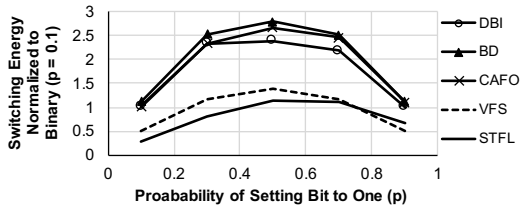

Figure 12: Switching energy versus random patterns.

technique holds the promise to help the system designers build energy-efficient computing system.

ACKNOWLEDGMENTS

The authors would like to thank anonymous reviewers for useful feedback. This work was supported in part by the National Science Foundation (NSF) under Grant CCF-1755874.

REFERENCES

[1] M. Anders *et al.*, "A transition-encoded dynamic bus technique for high-performance interconnects," *IEEE Journal of Solid-State Circuits*, vol. 38, no. 5, pp. 709–714, 2003.

[2] N. Muralimanohar *et al.*, "Optimizing nuca organizations and wiring alternatives for large caches with cacti 6.0," in *Proceedings of the Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 3–14, 2007.

[3] M. R. Stan *et al.*, "Bus-invert coding for low-power i/o," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 3, no. 1, 1995.

[4] T. M. Hollis, "Data bus inversion in high-speed memory applications," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 56, no. 4, pp. 300–304, 2009.

[5] R. Maddah *et al.*, "Cafo: Cost aware flip optimization for asymmetric memories," in *International Symposium on High Performance Computer Architecture (HPCA)*, pp. 320–330, 2015.

[6] M. R. Stan and W. P. Burleson, "Limited-weight codes for low-power i/o," in *International Workshop on Low Power Design*, vol. 6, pp. 6–8, Citeseer, 1994.

[7] Y. Song *et al.*, "Energy-efficient data movement with sparse transition encoding," in *IEEE International Conference on Computer Design (ICCD)*, pp. 399–402, 2015.

[8] H. Seol *et al.*, "Energy efficient data encoding in dram channels exploiting data value similarity," in *ACM/IEEE Annual International Symposium on Computer Architecture (ISCA)*, pp. 719–730, 2016.

[9] M. Bojnordi and E. Ipek, "Desc: Energy-efficient data exchange using synchronized counters," in *Proceedings of the Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 234–246, 2013.

[10] P. Behnam *et al.*, "Adaptive time-based encoding for energy-efficient large cache architectures," in *Proceedings of the 5th International Workshop on Energy Efficient Supercomputing (E2SC)*, 2017.

[11] A. N. Udipi *et al.*, "Non-uniform power access in large caches with low-swing wires," in *International Conference on High Performance Computing (HiPC)*, pp. 59–68, 2009.

[12] Y. Nakagome *et al.*, "Sub-1-v swing internal bus architecture for future low-power ulsis," *IEEE Journal of Solid-State Circuits*, vol. 28, no. 4, pp. 414–419, 1993.

[13] H. Zhang and J. Rabaey, "Low-swing interconnect interface circuits," in *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED)*, pp. 161–166, 1998.

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

[15] W. Zhao and Y. Cao, "New generation of predictive technology model for sub-45nm design exploration," in *International Symposium on Quality Electronic Design (ISQED)*, 2006.

[16] H. Zhang *et al.*, "Low-swing on-chip signaling techniques: effectiveness and robustness," *IEEE Transactions on very large scale integration (VLSI) systems*, vol. 8, no. 3, pp. 264–272, 2000.

[17] A. Majumder *et al.*, "A variation tolerant current mode low swing signaling approach for gigascale on-chip interface circuit," *AEU-International Journal of Electronics and Communications*, vol. 93, pp. 140–149, 2018.

[18] S. Li *et al.*, "Mcpat: An integrated power, area, and timing modeling framework for multicore and manycore architectures," in *IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 469–480, 2009.

[19] E. Ardestani *et al.*, "Esesc: A fast multicore simulator using time-based sampling," in *Proceedings of the 19th International Symposium on High Performance Computer Architecture (HPCA)*, pp. 448–459, 2013.

[20] D. H. Bailey *et al.*, "The nas parallel benchmarks—summary and preliminary results," in *Proceedings of the ACM/IEEE Conference on Supercomputing*, pp. 158–165, ACM, 1991.

[21] S. C. Woo *et al.*, "The splash-2 programs: Characterization and methodological considerations," in *Proceedings of the Annual International Symposium on Computer Architecture (ISCA)*, ACM, 1995.

[22] R. M. Yoo *et al.*, "Phoenix rebirth: Scalable mapreduce on a large-scale shared-memory system," in *Proceedings of the IEEE International Symposium on Workload Characterization (IISWC)*, pp. 198–207, 2009.

[23] N. Muralimanohar *et al.*, "Cacti 6.0: A tool to model large caches," *HP Laboratories*, pp. 22–31, 2009.

[24] J. H. Laros III *et al.*, "Energy delay product," in *Energy-Efficient High Performance Computing*, pp. 51–55, Springer, 2013.

[25] N. Muralimanohar and R. Balasubramonian, "Interconnect design considerations for large nuca caches," *SIGARCH Comput. Archit. News*, vol. 35, no. 2, pp. 369–380, 2007.

[26] H. Zarrabi *et al.*, "An interconnect-aware delay model for dynamic voltage scaling in nm technologies," in *Proceedings of the ACM Great Lakes Symposium on VLSI (GLSVLSI)*, pp. 45–50, 2009.

[27] "Products formerly skylake." https://ark.intel.com/products/codename/37572/Skylake.

[28] T. A. Dye, "Memory controller including compression/decompression capabilities for improved data access," Apr. 9 2002. US Patent 6,370,631.

[29] A. Shafiee *et al.*, "Memzip: Exploring unconventional benefits from memory compression," in *IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 638–649, 2014.