Improving Error-Resilience of Emerging Multi-Value Technologies

Magnus Själander, Gustaf Borgström, and Stefanos Kaxiras
Department of Information Technology
Uppsala University
Uppsala, Sweden
[firstname.lastname]@it.uu.se

***Abstract*—There exist extensive ongoing research efforts on emerging technologies that have the potential to become an alternative to today's CMOS technologies. A common feature among the investigated technologies is that of multi-value devices and the possibility of implementing quaternary logic and memory. However, multi-value devices tend to be more sensitive to interferences and, thus, have reduced error resilience. We present an architecture based on multi-value devices where we can trade energy efficiency against error resilience. Important data are encoded in a more robust binary format while error tolerant data is encoded in a quaternary format. We show for eight benchmarks an energy reduction of 32% and 36% for the register file and level-one data cache, respectively, and for the two integer benchmarks, an energy reduction for arithmetic operations of 13% and 23%. We also show that for a quaternary technology to be viable it need to have a raw bit error rate of one error in 100 million or better.**

***Keywords*-Approximate Computing, Energy Efficiency, Quaternary Logic, Emerging Technologies**

I. Introduction

CMOS technology scaling, the current driver of Moores law, is faced with practical and fundamental limits. The search for low-power alternatives to CMOS is intensifying with a large number of emerging technologies being investigated, e.g., single atom and single electron transistors [1], [2]. Many such technologies present new features, which provide opportunities to develop new computer systems. Multi-level devices are one such common feature that is already employed by several commercial memory technologies, e.g., in multi-level flash [3] and phase change memory [4]. Multi-level devices increase the on-chip information density, as they are capable of representing more than two distinct logical levels. Thus, multi-level devices have the potential of providing smaller, faster, and more energy-efficient circuit implementations as fewer wires, memory cells, and logic gates are required to transfer, store, and process a fixed amount of data. The gain in information density is commonly achieved at the cost of reduced error resilience of the multi-level devices. A multi-level device has more physical states, which by necessity reduce the margins between the individual states. Thus, making the device more susceptible to interference, see Figure 1.

There is an increasing research focus on approximate computing, i.e., on trading power, performance, and *reliability* against each other [5]. Previous work has shown that approximate computing can be used to improve performance and endurance of multi-level phase change memory (PCM) [6].

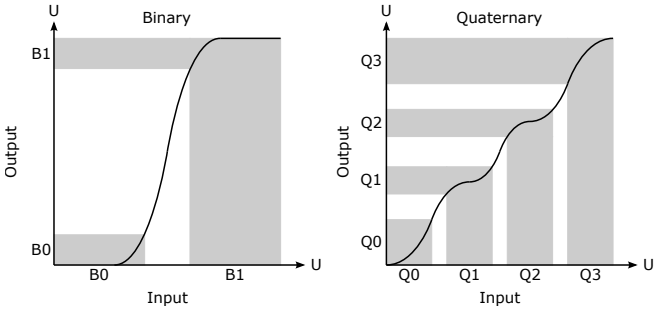

Figure 1. Illustration of input to output transfer functions of a fictitious binary and quaternary device, e.g., a memory cell or buffer. Note the much wider margins for the binary compared to the quaternary device.

In this work, we present a technique where error resilience of multi-value technologies can be modulated simply by the way data are encoded. We leverage approximate computing to create an architecture where operations and storage of data that are critical for the application are represented in a binary and more error resilient format, while less critical data are represented in a quaternary and more compact format.

II. Modulating Error Resilience

We consider any device where the output can be described as a continuous transfer function of its inputs. For multi-level logic this transfer function contains plateaus that presents a relatively stable output even if the input value varies slightly [7]. These plateaus appear as discrete ranges that can be used to represent logic states. Such devices have higher probability that the output of a device will shift, due to interference, to a nearby level than to a level far away from the expected level. Here we use the term interference for any physical property that affects the output such that it deviates from the ideal transfer function, e.g., input noise or manufacturing variances. We take advantage of this property to create two logic representations for representing approximate and precise data.

Figure 2 illustrates a transfer function of a fictitious quaternary device in some unit (U) that could be, e.g.,

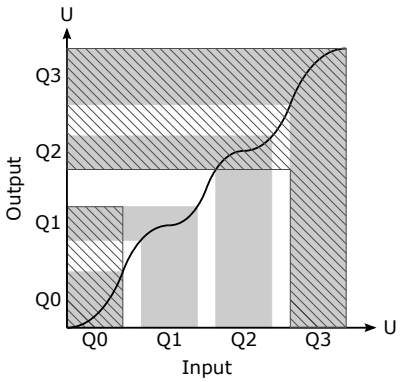

Figure 2. Illustration of a quaternary device that are used to operate on binary data. Gray regions represent the input and output ranges when working with quaternary data. Striped regions represent input and output ranges when working with binary data.

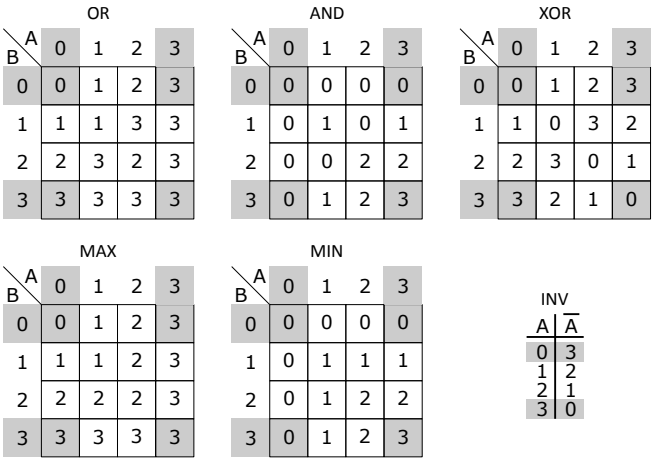

Figure 3. Truth tables of quaternary gates with their binary operations highlighted in gray.

voltage or current. The figure shows in light gray the four different states (Q0, Q1, Q2, and Q3) and how they are represented as ranges on the input and output of the device. When representing approximate data we use all logic states of the device. This maximizes the amount of data that can be represented per device, but it also increases the probability that a state shifts to a nearby state (e.g., Q2 gets interpreted as Q1 or Q3). Precise data is represented in binary form by only using the maximum (Q3) and minimum (Q0) states as input to the device. When the output is read nearby states are also considered as part of the maximum (Q2 and Q3) and minimum (Q0 and Q1) states (see striped regions in the figure). This increases the reliability of the device as the probability is low that Q0 would shift to Q2 or that Q3 would shift to Q1. However, it also requires (at least) twice the number of devices compared to representing the data approximately.

III. System Architecture

We have designed a complete system based on the principal of improved error resilience through the use of performing binary operations on quaternary devices (see Section II).

A. Logic and Arithmetic Operations

Logical operations on quaternary gates work without any special consideration when operating on binary data, as seen in Figure 3. A quaternary zero (Q0) also represents a binary zero (B0) while a quaternary three (Q3) represents a binary one (B1), highlighted in gray in the figure. From the figure it is apparent that if the three in the gray areas would be replaced with one then the gray areas represent the equivalent binary operation.

Performing additions on binary data using quaternary devices are not as straightforward. Considering the basic operation of a half (HA) and full adder (FA) the sum is not correct for all inputs. When adding two binary ones (quaternary three), the resulting sum becomes quaternary two when we instead need a quaternary three for a correct result, see Figure 4. Thus, it is not possible to design a ripple-carry adder simply from full-adder gates.

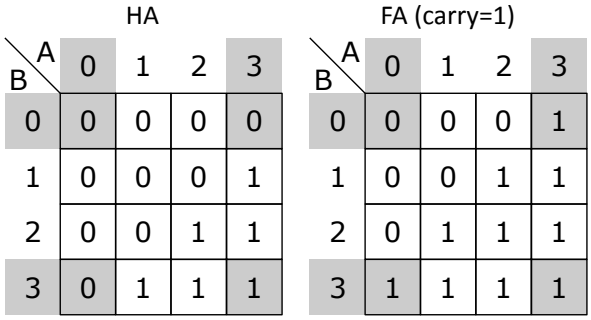

Figure 4. Truth table showing the carry of quaternary half and full adders. Gray regions highlight binary representation.

Ripple-carry adders are relatively slow and in most cases a more performant adder implementation is desirable. A carry-lookahead adder (CLA) is commonly used when implementing high-speed adders. The carry-lookahead technique is based on the creation of a generate (G) and propagate (P) signal for each significance pair of the input operands, see Figure 5. The generate and propagate signals are then used to calculate the carry for each digit. The key insight is that the carry (and the generate and propagate) signal is never anything other than zero or one, regardless, of which base the input operands have. The generate and propagate structure constitutes the majority of all gates in a CLA and all of them perform binary operations. Only the periphery circuits, i.e., the initial generate and propagate circuit and final summation circuit, operate at a higher base.

Figure 5 shows a Kogge-Stone [8] CLA that can perform both quaternary and binary additions. The gates highlighted in gray are quaternary gates while all other gates perform

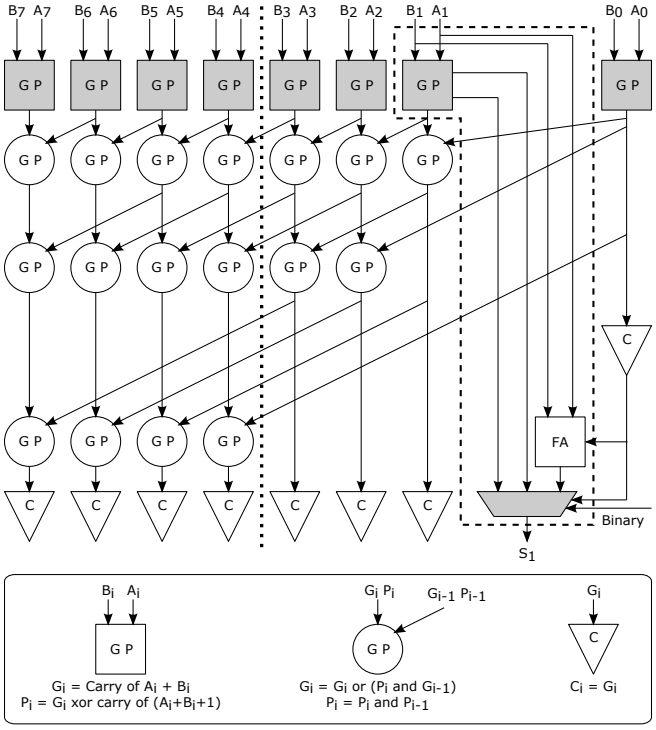

Figure 5. Illustration of a quaternary Kogge-Stone adder with the capability of operating on binary data. Gates in gray perform quaternary operations, all other gates perform binary operations.

binary operations. The initial generate signal is one when the input-pair of the operands would generate a carry and zero in all other cases, i.e., the carry of a HA. The initial propagate signal is one when the sum of the input-pair is exactly three, i.e., a carry is generated if a carry would come from the next lower significance level, hence the name propagate. The generate and propagate signals can hence be created as depicted in Figure 6.

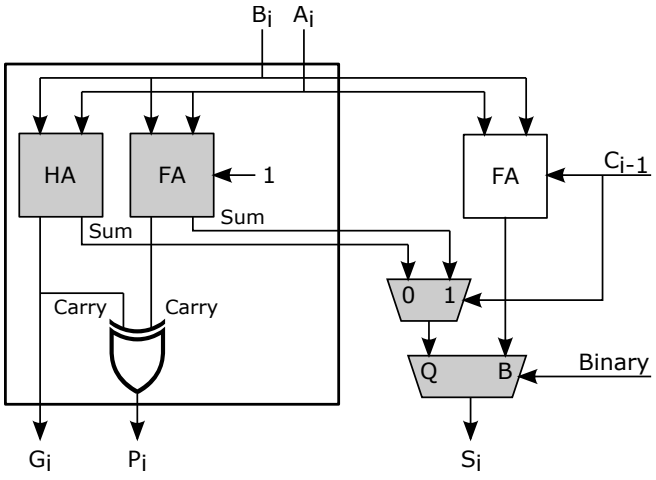

Figure 6. Circuits for the generate and propagate signals and the computation of the final sum.

As the quaternary HA and FA does not generate a correct binary sum (see Figure 4) a binary FA and a two-input multiplexor are required for each digit, see Figure 6. The delay through the binary FA can be made equal to that of the quaternary path. The additional multiplexor adds negligible delay to the critical path, since the *Binary* control signal of the multiplexor is available directly from the start of the addition. Thus, even though the control is a high-fanout signal it will have been applied to the multiplexor long before the sums have been calculated.

Subtraction can be handled as in conventional binary implementations by creating the complement of the subtrahend and then perform an addition with the minuend. The complement is created by inverting the subtrahend. Multiplication and division are more challenging and are left as future work. One simple approach would be to convert approximate data to precise data, perform the multiplication or division, and then convert it back.

B. Memory Hierarchy

We base the cache on the idea by M. Själander et. al. [9]. The memories storing data consist of quaternary memory cells, e.g., the single-electron memory by H. Inokawa et. al. [10]. The cache lines are organized such that each line stores N quaternary bytes (q-byte). Binary bytes (b-byte) can be stored by combining pairs of q-bytes as shown by the striped region of a quaternary word (q-word) in Figure 7.

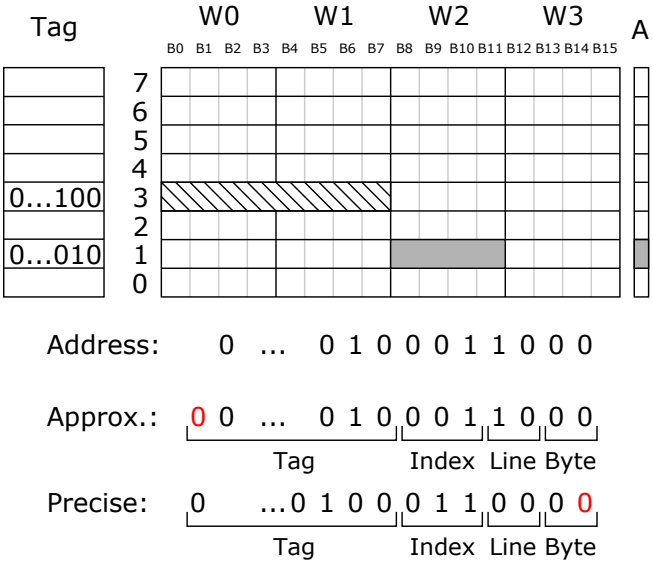

Figure 7. One way of an approximate cache, with eight sets, sixteen approximate q-bytes per cache line ($N = 16$), and four q-bytes per q-word.

Addressing memory becomes more complicated when bytes of different size have to coexist in the same memory system. The same address will have to access different physical locations depending on the type of the access (binary or quaternary). When accessing quaternary data the

address matches the granularity of the physical organization and, therefore, does not have to be modified. When accessing binary data the physical organization does not match, since two q-bytes are needed for one b-byte. Thus, only half the number of b-bytes fit in a cache line, which needs to be reflected by how the data are addressed. This is achieved by shifting the address one step to the left, (see Precise in Figure 7). The shift of the address for binary accesses increases the address size. To create equally sized addresses, an approximate address is prepended with a most significant zero, (see Approximate in Figure 7). The resulting addressing scheme has the affect that two different addresses could map to the same physical location. To distinguish between the two addresses a cache line is therefore only allowed to contain one type of data and a bit (A in Figure 7) is used to indicate the type of data that the line contains.

Binary and quaternary data should not be intermixed too finely in the address space to avoid fragmentation with unusable space. For efficient use of available cache space the smallest allocated space for binary or quaternary data should be a whole cache line that is cache line aligned. This is likely smaller than the smallest size that is desirable for main memory. For simplicity, the OS page size can be used as the smallest used granularity. The page table then becomes a natural place to store the information if a page contains binary or quaternary data.

Registers can be handled such that even registers can be used to store binary data while all registers can be used to store quaternary data. If an even register (i) holds binary data then the following odd register ($i + 1$) cannot be used for storing quaternary data. A bit for each pair of registers is used to indicate if it contains binary data.

C. System

An application has to be able to convey information about what type of operation or memory access that is to be performed. To avoid the need of a specialized instruction set architecture (ISA) with both approximate and precise instructions we instead keep this information together with the data in the architecture. The memory access type is known through the information provided in the page table. Loaded data from main memory are tagged in the architecture (the A bit for each cache line and pair of registers). This additional metadata are used to determine the mode (precise or approximate) when performing logic and arithmetic operations. Two new instructions are introduced to convert between approximate and precise data.

The precise representation is not immune to interference and unintentional changes from Q0 to Q1 or from Q3 to Q2 might occur. Depending on a particular technologies probability for this type of change precise data could, e.g., be restored when reading data from the cache (low probability), from the register file (medium probability), or from pipeline registers (high probability).

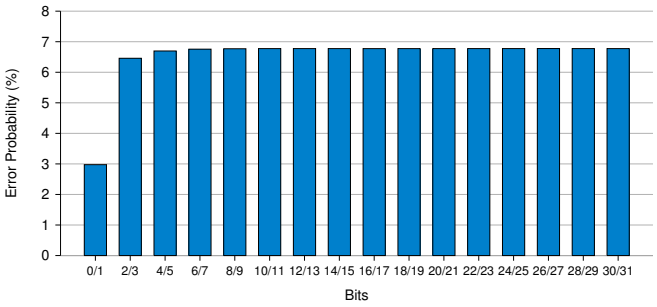

Figure 8. Error probability for each output bit when performing quaternary additions on a 32-bit Kogge-Stone adder. Errors are modeled with a 1% chance of an error at the input to the quaternary gates in Figure 6.

IV. Evaluation

We have used EnerJ [11] to classify data as either approximate or precise and to simulate the proposed architecture. EnerJ is a type system for Java and enables programmers to specify which data that can be treated approximately. The type system assures that approximate data are guarded in such a way that it does not bleed into and affect precise operations and data.

The EnerJ classified data are only available within the java runtime environment (JRE) and are not exposed to the architecture itself. Thus, it is not possible to use a conventional architectural simulator in combination with EnerJ. Instead, the architectural modeling needs to be performed within the JRE. We have extended the EnerJ JRE with our own cache model (described in Section III-B) and replaced error models with error models for our intended quaternary technology.

Performing the architectural simulation within the EnerJ JRE instead of on an architectural simulator imposes limits on the evaluations that can be performed. It is, e.g., not possible to accurately determine timing, as the architectural modeling is not separated from the execution of the application. There is no way of determining how much time is used for executing the application versus the time used for modeling the architectural behavior. We are therefore only evaluating energy aspects even though the use of quaternary data improves the hit rate for caches and enables more data to be kept in registers, which will have a positive impact on performance.

A. Benchmarks

We use seven benchmarks (scimark2 [lu, smm, fft, sor], imagefill, jmeint, and raytrace) that been developed together with the EnerJ framework and one benchmark (sobel) that we have developed ourselves to evaluate the proposed architecture. All benchmarks have been annotated with EnerJ's approximate keyword to indicate which data and operations that can be treated approximately by the architecture.

B. Error Models

We have implemented a functional model of the Kogge-Stone adder described in Section III-A and used it to create

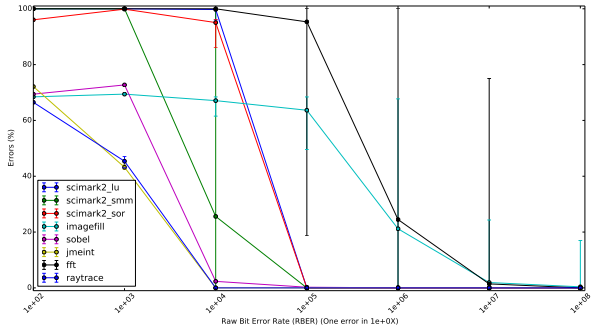

Figure 9. Output correctness of eight benchmarks when the raw bit error rate (RBER) is changed from one in 100 to one in 100 million.

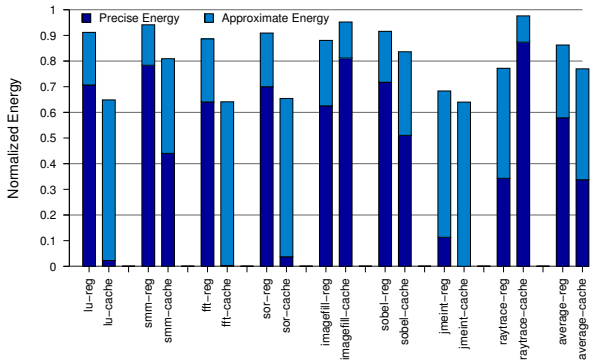

Figure 10. Relative register file (reg) and cache energy when executing approximately.

an error model. We inject errors at the input of the quaternary gates and observe how these propagate to the final result. The probability for an error at the output is then recorded and used for the architectural simulations, see Figure 8.

Error correcting codes (ECC) are commonly used to protect data stored in memories. This is especially true for multi-value memories (Flash, PCM), which have high raw bit error rates (RBER). We assume that all data is protected by an ECC that can detect and correct three bits and use the equation by N. Mielke et al. to calculate the effective bit error rate given a RBER [12]. For logical operations we apply the RBER directly on each bit and use the functional model to estimate the bit error rate for individual bits of additions and subtractions.

Figure 9 shows the correctness of the output for the eight benchmarks when the REBR are changed from one error in 100 to one in 100 million. For the figure we ran each benchmark 100 time for each REBR and the graph shows the minimum and maximum correctness. As seen in the figure, seven of the benchmarks produce correct results for all 100 runs at an REBR of one in 100,000. However, for imagefill an REBR of one in 100 million is needed to get close to reliable outputs. An REBR of one in 10-100 million has been reported for both PCM and Flash [12]–[15]. Our results show that a quaternary technology would have to reach at least this level of correctness for it to be usable.

C. Energy Estimation

To estimate the energy of the adder in precise (binary) and approximate (quaternary) mode we use switching gate equivalents. The gates that switch for creating the initial generate and propagate signals and the final sum are reduced by half when operating on quaternary data. The upper half of the input operands does not switch and no carries are created for the upper half of the final result. The carry-tree (the white GP-dots in Figure 5) has a higher switching activity as the generate and propagate signals of the lower half (right side of the dotted line) enters the upper half (left side of the dotted line). This causes switching at the inputs of GP-dots in the upper half, but will not cause any switching of their outputs. For a 32-bit adder, 80 of the 129 GP-dots will have inputs that switch. We therefore conservatively estimate the energy of an approximate addition to be 62% $(80/129)$ of a precise addition.

The cache has to be designed in such a way that data of variable width can be read from it to save energy. This can be achieved by designing the cache such that when precise data access two SRAM memories, while approximate data access only one of them. We estimate the energy savings by modeling a single way of a cache with a 4kB data array and 32 byte cache line size. The tag array is modeled as a 128 byte SRAM with a 32-bit wide port. According to CACTI [16] (version 6.5) the energy dissipation of the tag array is 32% of the data array (when using the 32nm ITRS-HP technology at 350K and optimizing for energy). The estimate of a 2kB 16-bit wide data array the energy dissipation is estimated to be 52.4% of the 4kB data array (100%). Designing the cache with two 16-bit SRAMs instead of a single 32-bit SRAM incurs a slight overhead of 4% $((32\% + 2 \times 52.4\%)/(32\% + 100\%))$ for precise accesses while approximate accesses only dissipate 64% $((32\% + 52.4\%)/(32\% + 100\%))$.

For the main memory we do not save in terms of power as a whole cache line is read and written at a time. The benefit of having approximate data is that twice the number of data items are read/written than compared to precise data. This reduces the total number of required accesses to main memory, which saves energy.

D. Results

Figure 10 shows the energy usage for the register file and cache normalized to a system which would only use precise operations. The average register file and cache energy savings are 14% and 33%, respectively. Jmeint shows the greatest improvement with 32% register file and 36% cache energy savings. This is due to the high fraction of data that can be treated approximately in jmeint.

Only two of the benchmarks use integer operations. Imagefill achieves a 13% energy reduction for its arithmetic operations while sobel reduce it by as much as 23%.

Table I
L1 DC MISS RATE

Benchmark	Original (%)	Approximate (%)	Improvement (%)
lu	5.11	2.67	48%
smm	7.00	4.36	38%
fft	15.91	0.004	100%
sor	2.13	1.08	49%
imagefill	17.26	11.62	33%
sobel	0.10	0.05	50%
jmeint	1.30	0.65	50%
raytrace	0.50	0.25	50%

Table I shows the miss rate for the modeled level one (L1) data cache (DC), 4-way, 16KiB, with 32 byte line size. The column denoted original shows the miss rate for the eight benchmarks when approximate computing is not used. The last column shows the improvements in miss rate that are achieved when storing data approximately in the cache. For most of the benchmarks the improvements is close to 50% while for fft almost all cache misses can be avoided as the working set now fit in the L1 data cache when storing it approximately. The reduced miss rate has the effect of fewer main memory accesses, which improves both performance and energy efficiency.

V. CONCLUSION

The digital age has come to rely on rapid improvements in performance, which has been driven by ever cheaper devices as predicted by Moore's law. However, contemporary CMOS technologies are reaching their practical and physical limits. An alternative could be one of the many researched and emerging nano technologies. Operating at the nano scale poses many challenges, one of which is reliable operations. We have presented a technique where reliability can be modulated for a multi-value technology. By encoding data either as binary or quaternary it is possible to trade energy efficiency against reliability. With this technique we might be one step closer to building systems based on emerging multi-value technologies.

ACKNOWLEDGMENT

This work was supported by the proactive collaborative project TOLOP (318397) of the Seventh Framework Program of the European Commission.

REFERENCES

[1] J. Verduijn, G. C. Tettamanzi, and S. Rogge, "Wave function control over a single donor atom," *Nano Letters*, vol. 13, no. 4, pp. 1476–1480, 2013.

[2] V. Deshpande, R. Wacquez, M. Vinet, X. Jehl, S. Barraud, R. Coquand, B. Roche, B. Voisin, C. Vizioz, B. Previtali, L. Tosti, P. Perreau, T. Poiroux, M. Sanquer, B. De Salvo, and O. Faynot, "300 K operating full-CMOS integrated single electron transistor (SET)-FET circuits," in *International IEEE Electron Devices Meeting*, Dec. 2012, pp. 8.7.1–8.7.4.

[3] D.-S. Byeon, S.-S. Lee, Y.-H. Lim, J.-S. Park, W.-K. Han, P.-S. Kwak, D.-H. Kim, D.-H. Chae, S.-H. Moon, S.-J. Lee, H.-C. Cho, J.-W. Lee, M.-S. Kim, J.-S. Yang, Y.-W. Park, D.-W. Bae, J.-D. Choi, S.-H. Hur, and K.-D. Suh, "An 8 Gb multi-level NAND flash memory with 63 nm STI CMOS process technology," in *Proceedings of the IEEE International Solid-State Circuits Conference*, Feb. 2005, pp. 46–47 Vol. 1.

[4] H.-S. Wong, S. Raoux, S. Kim, J. Liang, J. P. Reifenberg, B. Rajendran, M. Asheghi, and K. E. Goodson, "Phase change memory," *Proceedings of the IEEE*, vol. 98, no. 12, pp. 2201–2227, Dec. 2010.

[5] J. Han and M. Orshansky, "Approximate computing: An emerging paradigm for energy-efficient design," in *Proceedings of the IEEE European Test Symposium*, May 2013, pp. 1–6.

[6] A. Sampson, J. Nelson, K. Strauss, and L. Ceze, "Approximate storage in solid-state memories," in *Proceedings of the ACM/IEEE Annual International Symposium on Microarchitecture*, Dec. 2013, pp. 25–36.

[7] M. Seo, C. Hong, S.-Y. Lee, H. K. Choi, N. Kim, Y. Chung, V. Umansky, and D. Mahalu, "Multi-valued logic gates based on ballistic transport in quantum point contacts," *Scientific Reports*, vol. 4, Jan. 2014.

[8] P. M. Kogge and H. S. Stone, "A parallel algorithm for the efficient solution of a general class of recurrence equations," *IEEE Transactions on Computers*, vol. 100, no. 8, pp. 786–793, 1973.

[9] M. Själander, N. S. Nilsson, and S. Kaxiras, "A tunable cache for approximate computing," in *Proceedings of the IEEE International Symposium on Nanoscale Architecture*, Jul. 2014, pp. 88–89.

[10] H. Inokawa, A. Fujiwara, and Y. Takahashi, "A multiple-valued SRAM with combined single-electron and MOS transistors," in *Proceedings of the Device Research Conference*, Jun. 2001, pp. 129–130.

[11] A. Sampson, W. Dietl, E. Fortuna, D. Gnanapragasam, L. Ceze, and D. Grossman, "EnerJ: Approximate data types for safe and general low-power computation," in *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation*, Jun. 2011, pp. 164–174.

[12] N. Mielke, T. Marquart, N. Wu, J. Kessenich, H. Belgal, E. Schares, F. Trivedi, E. Goodness, and L. R. Nevill, "Bit error rate in NAND flash memories," in *Proceedings of the IEEE International Reliability Physics Symposium*. IEEE, 2008, pp. 9–19.

[13] D. Ielmini, A. L. Lacaita, and D. Mantegazza, "Recovery and drift dynamics of resistance and threshold voltages in phase-change memories," *IEEE Transactions on Electron Devices*, vol. 54, no. 2, pp. 308–315, 2007.

[14] S. Yeo, N. H. Seong, and H.-H. S. Lee, "Can multi-level cell PCM be reliable and usable? Analyzing the impact of resistance drift," in *Proceedings of the Workshop on Duplicating, Deconstructing and Debunking*, Jun. 2012.

[15] Y. Cai, E. F. Haratsch, O. Mutlu, and K. Mai, "Error patterns in MLC NAND flash memory: Measurement, characterization, and analysis," in *Proceedings of the Conference on Design, Automation and Test in Europe*. IEEE, 2012, pp. 521–526.

[16] N. Muralimanohar, R. Balasubramonian, and N. P. Jouppi, "CACTI 6.0: A tool to model large caches," HP Laboratories, Tech. Rep. HPL-2009-85, Apr. 2009.