Reducing Memory-Bus Energy Consumption of GPUs via Software-Based Bit-Flip Minimization

Alex Fallin
Department of Computer Science
Texas State University
San Marcos, USA
waf13@txstate.edu

Martin Burtscher
Department of Computer Science
Texas State University
San Marcos, USA
burtscher@txstate.edu

***Abstract*—Energy consumption is a major concern in high-performance computing. One important contributing factor is the number of times the wires are charged and discharged, i.e., how often they switch from '0' to '1' and vice versa. We describe a software technique to minimize this switching activity in GPUs, thereby lowering the energy usage. Our technique targets the memory bus, which comprises many high-capacitance wires that are frequently used. Our approach is to strategically change data values in the source code such that loading and storing them yields fewer bit flips. The new values are guaranteed to produce the same control flow and program output. Measurements on GPUs from two generations show that our technique allows programmers to save up to 9.3% of the whole-GPU energy consumption and 1.2% on average across eight graph-analytics CUDA codes without impacting performance.**

***Index Terms*—bit flips, energy consumption, memory, GPUs**

I. Introduction

Computing is often power constrained, and the primary means to building faster computers is to improve their energy efficiency [1]. Since many important aspects of our lives depend on software, lowering the energy consumption has the potential to accelerate lifesaving computations. Making computers more energy efficient is also important for other reasons. For example, the energy used by just the servers and data centers in the United States is equal to the annual energy usage of millions of U.S. households [2]. Moreover, a single supercomputer's energy consumption can result in the emission of over 185,000 metric tons of CO_2 annually, which is about the same as burning 21 million gallons of gasoline [3].

Most modern computer chips are based on CMOS technology. Such circuits constantly draw a little (static) leakage power whenever they are on [4]. However, most of the power draw is due to circuit switching activity during computations [4]. This dynamic power is the product of four factors:

$$P_{dynamic} = C \cdot V^2 \cdot N \cdot f$$

Here, C denotes the capacitance, V the voltage, N the switching activity, and f the clock frequency. The capacitance is mostly affected by the wire length, layout, and semiconductor material used, all of which tend to improve with each new chip generation [4]. The voltage has already been decreased to a level where lowering it further would greatly increase the leakage power and therefore negate any benefit in dynamic power savings [4]. Reducing the frequency diminishes the performance [5]. This leaves the switching activity, which has been studied the least, as a target for improving the energy efficiency of computers.

The switching activity is proportional to how often the wires of the circuit are charged and discharged, i.e., the number of times they change from '0' to '1' or '1' to '0'. The resulting energy consumption depends on the capacitance. Hence, the memory bus is of great interest because it is used frequently and comprises many high-capacitance wires.

Data Bus Inversion (DBI) is a commercial solution to reduce the number of bit flips on memory buses that first became available with DDR4 SDRAM [6]. It requires an extra wire for every eight bits on the bus and works as follows. It starts by comparing the next value to be sent over the bus with the value currently on the bus. This is done at byte granularity. If more than four bits differ for any byte, the corresponding bytes are inverted. The remaining bytes are placed on the bus unmodified. For each modified byte, the extra DBI wire is activated to tell the receiver which bytes must be inverted back to recreate the original value.

The approach we describe in this paper does not require any additional wires. It is entirely software based and can, therefore, readily be deployed on existing systems. It targets specific integer values that are often read from or written to memory such as *sentinels* and the *initial values* of data structures. For example, breadth-first search (BFS) computes the distance of every vertex in a graph from a given source vertex s in number of edges that must minimally be traversed to reach the vertex. BFS implementations typically start by setting the distance of s to 0 and the distance of all other vertices to ∞. In practice, a large value such as $UINT\_MAX$ is often used to represent ∞. This value may be transferred many times over the memory bus during the BFS computation. However, each copy of the value will eventually be overwritten with the vertex's distance, which tends to be a small integer. The problem is that the bits in the initial and final values differ in many positions. Whereas the initial values contain mostly '1' bits, the final values contain mostly '0' bits as illustrated in Table I, leading to a large number of bit flips and, therefore, high energy consumption.

We can reduce the bit flips by selecting a different initial

TABLE I
BINARY REPRESENTATION OF FOUR VALUES

32-bit representation	Value	Bit-flips with 5
11111111...1...11111111	$UINT\_MAX$	30
11111111...1...11111110	$UINT\_MAX - 1$	31
10000000...0...00000000	$UINT\_MAX/2 + 1$	3
00000000...0...00000101	5 (a small integer)	0

value such as $UINT\_MAX/2 + 1$, which is also a large value but contains mostly '0' bits as shown in Table I. If this new value is large enough to represent ∞ (i.e., larger than the diameter of the graph), using it instead of $UINT\_MAX$ will yield the same BFS result. However, the number of bit flips will be lower, thus saving energy. In summary, our approach is to replace frequently accessed values with new values that yield the same result and control flow but fewer bit flips.

This paper makes the following main contributions.

- We experimentally confirm that bit flips on memory buses contribute significantly to the GPU's energy consumption.
- We demonstrate that strategically replacing sentinel and initialization values in the source code can result in over 9% GPU-wide energy savings without impacting performance or changing the program's control-flow decisions.
- We show that such savings can be obtained on optimized and unoptimized codes across different GPU generations.

The bit-flip-reduced codes are available in open source at https://cs.txstate.edu/~burtscher/research/bit-flips/.

The rest of this paper is organized as follows. Section II explains our approach in detail. Section III summarizes related work. Section IV describes the experimental methodology. Section V presents and discusses the results. Section VI concludes the paper with a summary and future work.

II. Approach

Due to their high performance and energy efficiency, GPUs are widely used in HPC. For example, over half of the ten fastest supercomputers on the TOP500 list contain GPUs [7], and all but one of the ten most energy-efficient computers on the GREEN500 list are GPU-based [8]. Hence, GPUs make an interesting target for energy-reduction optimizations.

To demonstrate that our approach works across a range of programs, we applied it to four simple codes that we wrote and to four highly-optimized CUDA codes from the literature. In both categories, one code is regular with strided memory accesses and the remaining codes are irregular with complex memory access patterns [9]. All of them stem from the field of graph analytics and implement real-world algorithms.

Table II lists the eight programs. The top of the table shows the four high-performance GPU codes, two of which are ours, and from where we obtained them. The bottom shows the four simple codes we added to provide additional test cases. They are less optimized and not particularly fast. We included them to demonstrate that our approach benefits a wide range of codes and is not limited to HPC codes.

APSP: The all-pairs-shortest-paths codes implement the Floyd-Warshall algorithm [13] to compute the shortest distance

TABLE II
CUDA CODES USED IN EXPERIMENTS

Name	Source
ECL APSP	[10]
Gardenia BFS	[11]
ECL MIS	[12]
Gardenia SSSP	[11]
Simple APSP	ours
Simple BFS	ours
Simple MIS	ours
Simple SSSP	ours

between all possible pairs of vertices in a weighted, directed graph with n vertices and record the result in an $n \times n$ matrix. They perform $O(n^3)$ strided memory accesses as they make n passes over the matrix. For any edge from vertex u to vertex v, the matrix element $m[u][v]$ is initialized to the weight of the edge. All remaining matrix elements are initialized to $INT\_MAX/2$ in both codes. We replaced these values by $INT\_MAX/4 + 1$ in the bit-flip optimized versions. The initialization value must be larger than the largest edge weight and doubling it must not produce signed overflow.

BFS: The breadth-first-search codes compute the minimum number of hops from a source vertex to all other vertices (cf. Section I) in an unweighted graph and record the result in an n-element array. Since the vertices are processed in graph order, the array is accessed in a complex pattern. The Gardenia code employs a data-driven [14] implementation that processes the vertices level by level. The simple code employs a topology-driven [14] implementation that performs push-style [15] updates until no more updates can be made. Except for the source element, all array elements are initialized to 1 billion in the Gardenia code and to $INT\_MAX - 1$ in the simple code. We also ran a version of the Gardenia code that uses $INT\_MAX - 1$ to show the consumption when a more conventional initial value is used. In all three cases, we replaced the values by $INT\_MAX/2 + 1$ in the bit-flip optimized versions. The initialization value must be larger than the maximum hop count from the source vertex. In the simple code, adding 1 to it must not produce signed overflow.

MIS: The maximal-independent-set codes use Luby's randomized algorithm [16] to compute a maximal independent set of the vertices of an unweighted graph. They record whether a vertex is in the set, out of the set, or not yet decided in an n-element array. They further record a priority (random number) for each vertex. Again, the vertices are processed in graph order, and the data are accessed in an irregular fashion. The codes employ a topology-driven implementation. The simple code only performs pull-style updates [15] whereas the high-performance code executes push and pull updates until the status of every vertex is known.

ECL MIS encodes the combined status and priority of each vertex in a byte array as follows. If the top 7 bits of the byte are 0, the vertex is out of the set. If they represent 127, the vertex is in the set. Any value between 1 and 126 means the status is undecided and the value indicates the priority. The LSB is cleared for the two decided values and set for all undecided values to speed up a timing-critical test in the code "if (val & 0x01)" [12]. Hence, the array starts out with mostly random bits that are eventually replaced by either 0x00 or 0xFE. However, those two final values differ in 7 bit positions. To alleviate this problem, we changed the encoding as follows in our bit-flip optimized version. 0x00 still indicates that the vertex is out of the set and values between 1 and 126 represent priorities. However, 0x80 now means the vertex is in the set. Note that 0x00 and 0x80 differ in only one bit position. We further replaced the timing-critical test with the equally fast "if (val & 0x7F)". In addition to supporting a fast test to check if a vertex is undecided, the values must fit in a byte and maintain their greater-than relationship. The simple MIS code works similarly but uses an integer instead of a byte array.

SSSP: Both of the single-source-shortest-path codes compute the minimum distance from a given source vertex to all other vertices in a weighted graph. The simple code records the result in an n-element array. Again, the array is accessed following a complex pattern. The codes employ a topology-driven implementation that performs pull-style updates until all distances have converged. Except for the source element, all array elements are initialized to INT_MAX in the simple code and to $INT\_MAX/2$ in Gardenia. We replaced these values by $INT\_MAX/2{+}1$ in the bit-flip optimized versions. The initialization value must be larger than the maximum distance from the source vertex.

III. Related Work

We are not aware of prior software-based work on reducing energy consumption through bit-flip minimization. Hence, we briefly summarize hardware-based approaches as well as bit-flip-minimization techniques for non-energy-related purposes.

Section I already introduced DBI, a hardware-based mechanism to reduce bit flips on the memory bus by selectively inverting bytes. It requires an extra wire for each byte, that is 12.5% more wires. It automatically adapts to the data and potentially works on all values going over the bus. In contrast, our approach does not require any additional wires but only applies to some of the values being transferred. Also, our approach can employ more sophisticated encoding schemes than simple bit inversion, like we use in ECL MIS.

Li et al.'s work is based on the observation that 8T SRAM consumes less energy when reading a '1' than a '0'. They propose a new circuit design to exploit this asymmetric energy consumption [17]. They profiled many applications to generate a coding scheme that increases the quantity of '1' bits in both the data and instruction stream and, as a side effect, reduces the switching activity on the connections between SRAM units.

Lucas et al. demonstrate how neither reducing the number of zeros, nor reducing switching activity, are optimal techniques on their own in memory with '1' affinity [18]. They introduce a novel scheme that finds the optimal encoding given the power ratio between transmitting a zero and switching activity for this type of memory. Further, they show how their scheme could be implemented in hardware.

Dgien et al. suggest a compression architecture for non-volatile memory (NVM) that, when storing data, uses frequent pattern compression and read-modify-write accesses to modify as few bits as possible and only writes the bits that need to be changed [19]. The resulting reduction of bit writes can extend the lifetime of NVM.

Bittman et al. discuss how reducing bit flips is more important than reducing writes to NVM [20]. They present a linked-list implementation that is designed to reduce the number of bit flips in memory to extend the lifetime of phase change memory.

IV. Experimental Methodology

We collected our energy measurements using NVIDIA's NVML library [21], which accesses the GPU's on-board energy sensor. It is important to note that the energy readings are of the *entire* GPU and not just the memory bus. We ran each experiment on two GPUs from different generations: a TITAN V compute GPU (Volta architecture) with a core clock frequency of 1200 MHz, and a GeForce RTX 2070 SUPER consumer GPU (Turing architecture) with a core clock frequency of 1605 MHz. Table III provides details about the global (main) memory of these GPUs.

TABLE III
GPU memory specifications

GPU	Type	Size	Bandwidth	Frequency
TITAN V	HBM2	12 GB	653 GB/s	850 MHz
RTX 2070 SUPER	GDDR6	8 GB	448 GB/s	7000 MHz

These GPUs employ two distinct types of memory: GDDR6 and HBM2. GDDR6 uses non-return-to-zero encoding. HBM2 employs vertically stacked DRAM dies, which allows for a much wider bus than GDDR6.

The system used for these experiments is based on an AMD Ryzen Threadripper 2950X CPU that holds both the TITAN V and the RTX 2070 SUPER. It runs Fedora 34, nvcc 11.4, and g++ 11.2.1. The CUDA driver version is 470.63.

We evaluated the eight programs listed in Table II. We instrumented them to read the GPU energy sensor before and after executing the kernel(s) of interest and print the difference. In addition to this baseline version, we created a second version of each program that includes our bit-flip optimizations described in Section II. These modifications do not change the instructions generated by the compiler (except for some immediate fields) or the control flow of the programs.

We used the 18 graphs listed in Table IV as inputs. They were obtained from the Center for Discrete Mathematics and Theoretical Computer Science at the University of Rome (Dimacs) [22], the Galois framework (Galois) [23], the Stanford Network Analysis Platform (SNAP) [24], and the SuiteSparse Matrix Collection (SMC) [25]. The table lists the name, type, number of vertices, and number of edges for each input. Where necessary, we made the graphs undirected, removed self edges, or added weights to the edges. In the used CSR format, each undirected edge is represented by two directed edges.

We selected these inputs for their wide variety. The highest edge weight is 134 million, which allows us to implement our method as described in Section II without invalidating any invariants, i.e., without affecting program correctness.

Due to the quadratic memory usage, we had to resort to much smaller inputs for the APSP codes. We generated APSP inputs for several vertex counts with pseudo-random edges. The edge count is 4, 16, and 64 times the vertex count.

TABLE IV
INPUT GRAPHS USED IN EXPERIMENTS

Name	Type	Vertices	Edges
2d-2e20.sym	grid	1,048,576	4,190,208
amazon0601	co-purchases	403,394	4,886,816
as-skitter	Internet topo.	1,696,415	22,190,596
citationCiteseer	citations	268,495	2,313,294
cit-Patents	patent cites	3,774,768	33,037,894
coPapersDBLP	citations	540,486	30,491,458
delaunay_n24	triangulation	16,777,216	100,663,202
europe_osm	road map	50,912,018	108,109,320
in-2004	web links	1,382,908	27,182,946
internet	Internet topo.	124,651	387,240
kron_g500-logn21	Kronecker	2,097,152	182,081,864
r4-2e23.sym	random	8,388,608	67,108,846
rmat16.sym	RMAT	65,536	967,866
rmat22.sym	RMAT	4,194,304	65,660,814
soc-LiveJournal1	community	4,847,571	85,702,474
uk-2002	web links	18,520,486	523,574,516
USA-road-d.NY	road map	264,346	730,100
USA-road-d.USA	road map	23,947,347	57,708,624

To increase the accuracy of the energy measurements and make the comparisons as fair as possible, we employed the following methodology. We ran each baseline code in a loop until the total execution time exceeded 10 seconds and recorded how many loop iterations this required. We then ran the bit-flip optimized code with the same number of iterations. We repeated these alternating experiments 10 times for each input, yielding 10 matched pairs of results, each pair containing the consumption of the baseline code and the consumption of the corresponding bit-flip optimized code. These measurements were evaluated with a paired t-test using $\alpha = 0.05$.

We divided the bit-flip optimized energy consumption by the consumption of the baseline code to generate 10 ratios for each input. We report these consumption ratios in box and whisker plots that show the distribution of the data. The minimum and maximum values are displayed using whiskers that extend above and below the two boxes. The top box extends from the median value to the third quartile. The bottom box reaches from the median to the first quartile. A shaded box indicates failure of the t-test. In other words, all non-shaded boxes are inputs with a confidence of at least 95% that our bit-flip-minimization approach reduces the energy consumption.

V. RESULTS

In this section, we first describe the experiments we used to confirm that bit flips on the memory bus contribute substantially to the overall energy consumption of GPUs. Then, we present the energy savings of our technique on the highly-optimized CUDA codes and, finally, on the simple codes. *Since the runtimes are identical for the bit-flip optimized codes and the original codes, we do not show runtimes.*

A. Buffer Transfer Experiments

To evaluate the potential behind our approach, we transferred 10 MB buffers holding specific bit patterns back and forth across the memory bus using device-to-device copies. The three patterns are all '0' bits, all '1' bits, and a sequence of pseudo-random numbers. The median results of these measured transfers are listed in Table V, which shows the GPU-wide energy consumption in millijoules for the approximately ten seconds of operation. Table VI lists the consumption as a percentage relative to transferring only zeros.

TABLE V
BUFFER TRANSFER ENERGY IN MILLIJOULES

GPU	All 0s	All 1s	Random
TITAN V	944,507	955,481	1,211,705
RTX 2070 SUPER	1,176,937	1,180,755	1,468,913

TABLE VI
BUFFER TRANSFER ENERGY RELATIVE TO ALL 0S

GPU	All 0s	All 1s	Random
TITAN V	100%	101%	128%
RTX 2070 SUPER	100%	100%	124%

These result show that bit flips greatly affect the energy consumption even though the running time and the number of bytes transferred are the same. Transferring random data, with a 50% bit-flip probability on each wire of the memory bus, requires 28% more energy on the Titan V and 24% more energy on the RTX 2070 than transferring only zeros (but the same runtime). This substantial difference is the basis of our work and constitutes an approximate upper bound on the possible energy savings using bit-flip minimization.

B. High-Performance Codes

This subsection studies the energy reduction in the high-performance CUDA codes that we modified as outlined in Section II to minimize bit flips. For reference, the result charts include a blue line marking the 100% ratio. *A value below this line indicates that the bit-flip-reduced code is more energy efficient than the original code.*

ECL APSP: The ECL APSP results are shown in Figures 1 and 2. For the Titan V, no median ratio is above 100%, and for the RTX 2070 only two median ratios are above 100%, indicating that our bit-flip minimization technique is saving energy in the majority of the cases. On both GPUs, the median savings range from -0.5% to 3.2%. In the worst case, our modified code requires 2% more energy on the Titan V and 1.5% on the RTX 2070. In the best case, it requires 9.3% and 3.2% less energy. On the Titan V, 6 of 18 inputs did not meet the t-test, and on the RTX 2070 8 inputs did not. There is a clear trend on both GPUs. Whereas the overall variance is high as indicated by the long whiskers, the sparser the input (adjacency matrix) the higher the energy savings is for all tested vertex counts. This is due to the sparser matrices containing a larger number of the modified initialization value, resulting in a higher reduction in bit flips and, therefore, a lower energy consumption relative to the original code.

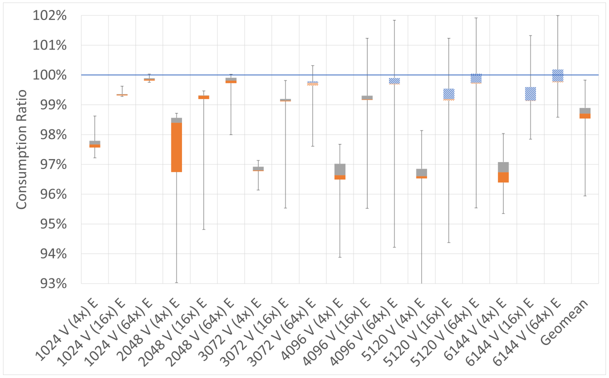

Fig. 1. Titan V ECL APSP results

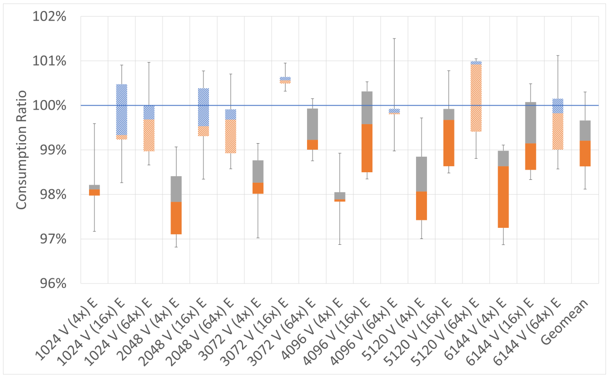

Fig. 2. RTX 2070 ECL APSP results

Gardenia BFS: The Gardenia BFS results are shown in Figures 3 and 4. The Titan V has one input with a median above 100% and the RTX 2070 has two. These inputs failed the t-test, as did one additional input for the RTX 2070. On all remaining inputs, we are saving energy. On the Titan V, the worst case requires 0.5% more energy whereas the best case requires 1.8% less. On the RTX 2070, the worst case requires 0.4% more energy and the best case 1.8% less. The average savings is 0.4% on both GPUs. When we replaced the default constant of 1,000,000,000 with $INT\_MAX - 1$ as discussed in Section II, the savings increased on both GPUs as expected (to 0.7% on average) since one billion has fewer '1' bits in binary than $INT\_MAX - 1$.

ECL MIS: The ECL MIS results are shown in Figures 5 and 6. The RTX 2070 has two inputs with a median above 100%, and the Titan V has none. On the Titan V, the average savings is 1.5%, the worst case is an 0.3% increase in consumption, and the best case is a savings of 2.8%. On the RTX 2070, the average savings is 1.2%, the worst case is a 1.1% increase, and the best case is a 3.4% saving. One input does not pass the t-test on the Titan V and 3 of the 18 inputs do not on

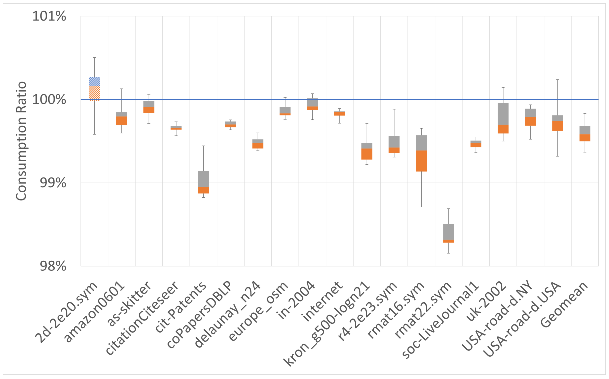

Fig. 3. Titan V Gardenia BFS results

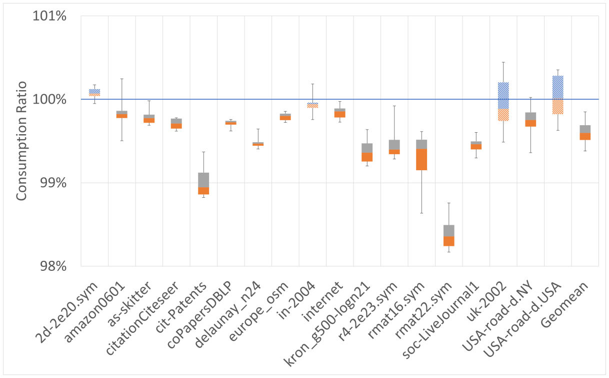

Fig. 4. RTX 2070 Gardenia BFS results

the RTX 2070. This GPU yields the highest savings on inputs with a low average vertex degree. Conversely, the three worst medians belong to the graphs with the highest average degrees.

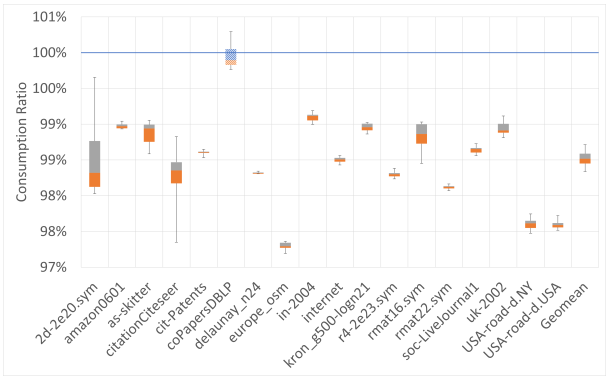

Fig. 5. Titan V ECL MIS results

Gardenia SSSP: The Gardenia SSSP results are shown in Figures 7 and 8. For both GPUs, there is only one input with a median ratio above 100%, meaning we save energy on average on most inputs. On the Titan V, the average savings is 0.7%, the worst case is a 1.7% increase in consumption, and the best case is a 2.6% savings. On the RTX 2070, the average savings is also 0.7%, but the worst case is a 0.4% loss in consumption and the best case is a savings of 2.6%. Two inputs

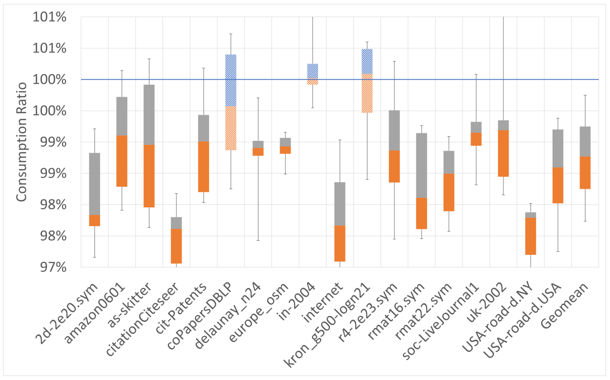

Fig. 6. RTX 2070 ECL MIS results

on each GPU fail the t-test, meaning we cannot say with 95% confidence that we are saving energy even if the median is below 100%.

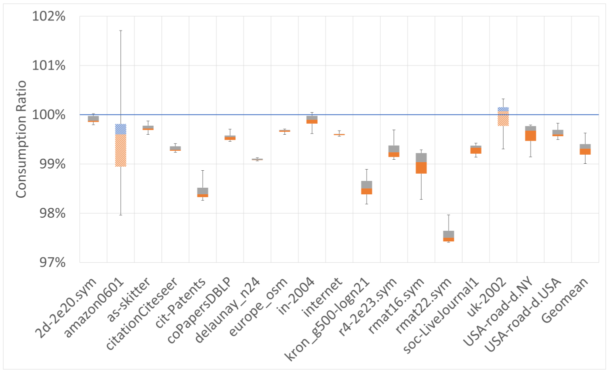

Fig. 7. Titan V Gardenia SSSP results

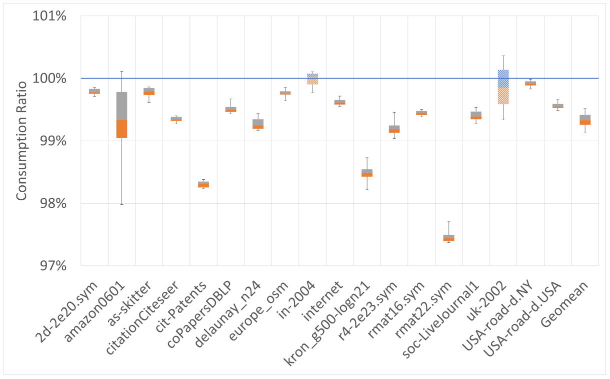

Fig. 8. RTX 2070 Gardenia SSSP results

In summary, we find that, on all four high-performance codes, our bit-flip reduction technique yields statistically significant energy savings on average on most of the inputs. *In three of these programs, the savings are due to changing a key constant in the source code.* Note that none of our modifications affect the running time or the correctness of the programs.

C. Simple Codes

APSP: The results for the simple APSP code are shown in Figures 9 and 10. The RTX 2070 has two inputs with a median above 100%. Due to the large variance seen in the results, six inputs on each GPU fail the t-test. On average, the modified initialization values save 1.1% energy on the Titan V. In the worst case, they increase the energy consumption by 2.5%, and in the best case they yield a savings of 6.2%. On the RTX 2070, the median savings is 1%, the worst case is a 1.2% increase in energy, and the best case is a 4% reduction. This simple version performs similarly to its high-performance counterpart, sparser inputs yield higher savings. Moreover, the smallest 4x input exhibits substantially lower savings than the larger 4x inputs. This is due to the fact that the 1024-by-1024 adjacency matrix fits entirely into the L2 cache of the Titan V and mostly into the L2 cache of the RTX 2070. The larger inputs substantially exceed the L2 size, thus resulting in more global memory accesses and, therefore, more opportunity for saving energy.

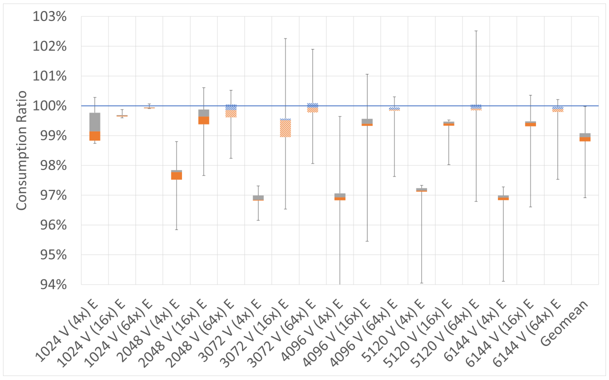

Fig. 9. Titan V simple APSP results

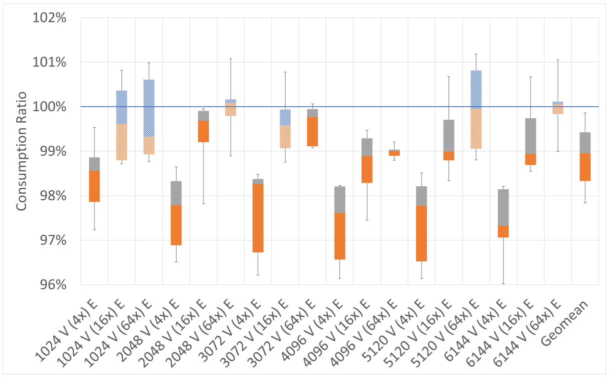

Fig. 10. RTX 2070 simple APSP results

BFS: The results for the simple BFS code are shown in Figures 11 and 12. BFS is an input-dependent irregular code. Despite this input dependence, the medians show that applying our bit-flip reduction technique always saves some energy on both GPUs. On the Titan V, every ratio is below 100%, and no inputs fail the t-test. On the RTX 2070, three inputs fail the t-test even though no median is above 100%. On average, the Titan V with bit-flip reduction consumes 1.9% less energy. Even in the worst case, the energy consumption is reduced by 0.5%, and in the best case it decreases by 4.8%. The RTX 2070 consumes an average of 1.2% less energy with bit-flip reduction. The worst case is a 1% increase and the best case a 3.4% savings.

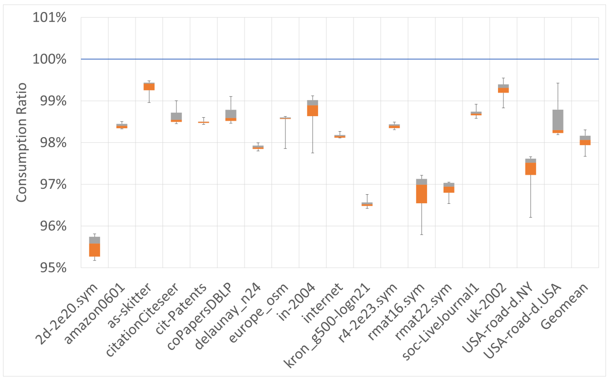

Fig. 11. Titan V simple BFS results

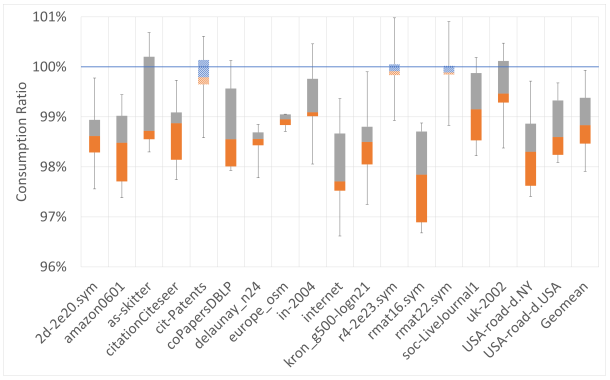

Fig. 12. RTX 2070 simple BFS results

MIS: The results for the simple MIS code are shown in Figures 13 and 14. MIS performs similarly to BFS. For both GPUs, 4 of 18 inputs fail the t-test. On the Titan V, the median savings is 3.1%, the largest loss is 2.7%, and the maximum savings is 10.6%. On the RTX 2070, bit-flip reduction saves 2.2% on average, increases the energy consumption by 1.7% in the worst case, and decreases it by 8.3% in the best case. Again, both GPUs tend to exhibit only small energy savings with higher-degree inputs such as coPapersDBLP and rmat16. The low-degree road maps, in contrast, yield some of the highest savings.

SSSP: The results for the simple SSSP code are shown in Figures 15 and 16. This code exhibits a lot of variance in consumption. As a result, eight inputs failed the t-test on the Titan V, and 13 failed on the RTX 2070. On the Titan V, the median savings is 0.4%, the maximum loss is 1.7%, and the maximum savings is 2.4%. On the RTX 2070, the median savings is 0.3%, the maximum loss is 3.2%, and the maximum

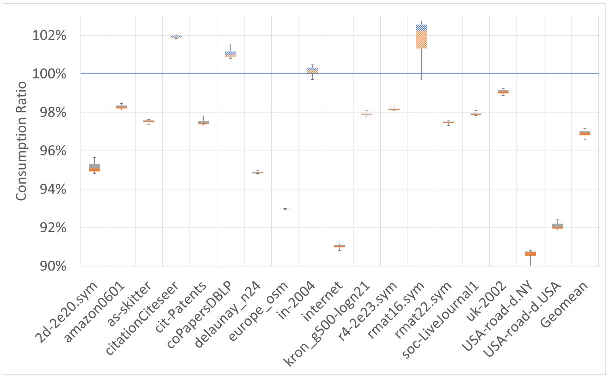

Fig. 13. Titan V simple MIS results

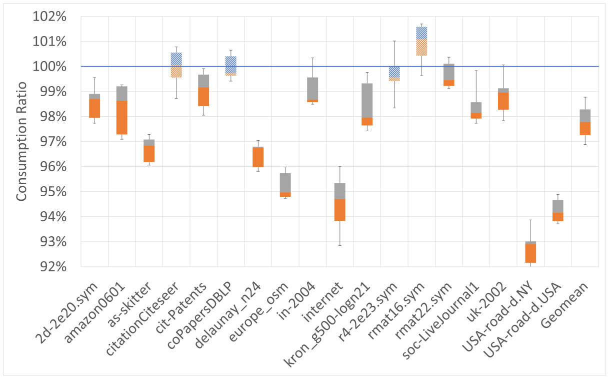

Fig. 14. RTX 2070 simple MIS results

savings is 4.2%. The trends seen in SSSP are less pronounced but still similar to those of the other simple codes.

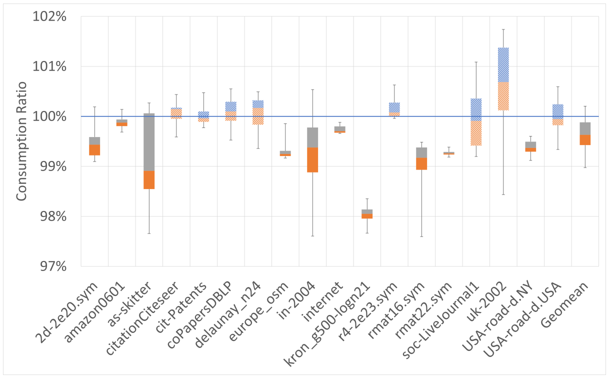

Fig. 15. Titan V simple SSSP results

In summary, all four simple programs yield noticeable energy savings on most of the inputs on average. Note that we focus on the averages and medians as the variance is quite significant in many of our measurements. *A high variance is expected because the programmer cannot influence the timing behavior of the threads nor the ordering of the memory accesses that the memory controller produces, both of which are likely to change from run to run even when using the same input, executable, and GPU.* In programs where such

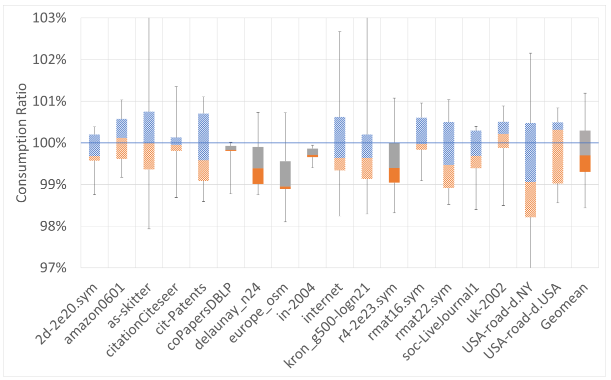

Fig. 16. RTX 2070 simple SSSP results

reorderings do not matter, e.g., our buffer transfer experiments, the variance is much smaller. In general, the RTX 2070 tends to exhibit less variance than the Titan V, and the simple codes tend to have more variance than the highly optimized codes. Like in the highly-optimized codes, none of our modifications affect the running time or the correctness of the programs.

VI. Conclusion and Future Work

CMOS-based computers primarily expend energy when flipping bits. Hence, reducing bit flips can save energy, particularly on high-capacitance wires such as those of the memory bus. Measuring the energy when copying fixed-size buffers filled with random or identical values revealed that the random values require 28% more GPU energy on a Titan V with HBM2 memory and 24% more GPU energy on an RTX 2070 with GDDR6 memory, even though the copying takes the same amount of time in all cases.

In this paper, we demonstrate how programmers can modify their source code to minimize bit flips on the memory bus. Being a software-only approach, *our technique works on existing hardware*. We show the benefit of such modifications on four high-performance and four unoptimized codes, two with regular and six with irregular memory access patterns. Surprisingly, it is often possible to reduce the energy consumption (without affecting the running time) with an intelligent change of the initialization value of a key data structure. We select the new initialization value to maintain existing program invariants such as exceeding the largest data value while also being closer to the expected final values in terms of the number of bits that differ (the Hamming distance). We further provide an example where we successfully changed a complex encoding scheme.

In future work, we plan to use memory profiling to determine which constants are being used often and are, therefore, potential targets for our technique. The ultimate goal is to automate as much as possible to lessen the human efforts required to identify and apply these optimizations.

In summary, we show how changing just one key constant in the source code can yield surprisingly large GPU-wide energy savings without affecting program correctness or performance. Having demonstrated the potential and feasibility of this idea, we believe our work so far is only the tip of the iceberg. We hope that it will raise awareness in software developers to carefully select data-structure initialization values, encourage hardware designers to provide more support for bit-flip reduction, and inspire researchers to search for further opportunities to minimize bit flips and save energy.

VII. Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant No. 1955367 and by an equipment donation from NVIDIA Corporation. We thank Arthi Nagarajan for her architectural study that gave us the idea presented in this paper.

References

[1] S. Ashby, P. Beckman, J. Chen, P. Colella, B. Collins, D. Crawford, J. Dongarra, D. Kothe, R. Lusk, P. Messina, and et al., “The opportunities and challenges of exascale computing.” [Online]. Available: https://science.osti.gov/-/media/ascr/ascac/pdf/reports/Exascale_subcommittee_report.pdf

[2] [Online]. Available: https://www.energystar.gov/ia/partners/prod_development/downloads/EPA_Datacenter_Report_Congress_Final1.pdf?7e9c-bbd7

[3] “Greenhouse gas equivalencies calculator.” [Online]. Available: https://www.epa.gov/energy/greenhouse-gas-equivalencies-calculator

[4] H. Veendrick, *Nanometer CMOS ICs: From Basics to ASICs.* Springer International Publishing, 2017.

[5] E. Le Sueur and G. Heiser, “Dynamic voltage and frequency scaling: The laws of diminishing returns,” in *Proceedings of the 2010 International Conference on Power Aware Computing and Systems*, ser. HotPower’10. USA: USENIX Association, 2010, p. 1–8.

[6] 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.

[7] “Top500.” [Online]. Available: https://www.top500.org/lists/top500/2022/06/

[8] “Green500.” [Online]. Available: https://www.top500.org/lists/green500/2022/06/

[9] M. Burtscher, R. Nasre, and K. Pingali, “A quantitative study of irregular programs on GPUs,” in *2012 IEEE International Symposium on Workload Characterization*, 2012, pp. 141–151.

[10] [Online]. Available: https://cs.txstate.edu/~burtscher/research/ECL-APSP/

[11] Z. Xu, X. Chen, J. Shen, Y. Zhang, C. Chen, and C. Yang, “GARDENIA: A domain-specific benchmark suite for next-generation accelerators,” *CoRR*, vol. abs/1708.04567, 2017. [Online]. Available: http://arxiv.org/abs/1708.04567

[12] M. Burtscher, S. Devale, S. Azimi, J. Jaiganesh, and E. Powers, “A high-quality and fast maximal independent set implementation for gpus,” *ACM Trans. Parallel Comput.*, vol. 5, no. 2, Dec. 2018. [Online]. Available: https://doi.org/10.1145/3291525

[13] R. W. Floyd, “Algorithm 97: Shortest path,” *Commun. ACM*, vol. 5, no. 6, p. 345, Jun. 1962. [Online]. Available: https://doi.org/10.1145/367766.368168

[14] R. Nasre, M. Burtscher, and K. Pingali, “Data-driven versus topology-driven irregular computations on gpus,” in *2013 IEEE 27th International Symposium on Parallel and Distributed Processing*, 2013, pp. 463–474.

[15] V. Jatala, R. Dathathri, G. Gill, L. Hoang, V. K. Nandivada, and K. Pingali, “A study of graph analytics for massive datasets on distributed multi-gpus,” in *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2020, pp. 84–94.

[16] M. Luby, “A simple parallel algorithm for the maximal independent set problem,” *SIAM Journal on Computing*, vol. 15, no. 4, pp. 1036–1053, 1986. [Online]. Available: https://doi.org/10.1137/0215074

[17] A. Li, W. Zhao, and S. L. Song, “Bvf: Enabling significant on-chip power savings via bit-value-favor for throughput processors,” in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-50 ’17. New York, NY, USA: Association for Computing Machinery, 2017, p. 532–545. [Online]. Available: https://doi.org/10.1145/3123939.3123944

[18] J. Lucas, S. Lal, and B. Juurlink, “Optimal dc/ac data bus inversion coding,” in *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2018, pp. 1063–1068.

[19] D. B. Dgien, P. M. Palangappa, N. A. Hunter, J. Li, and K. Mohanram, “Compression architecture for bit-write reduction in non-volatile memory technologies,” in *2014 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*, July 2014, pp. 51–56.

[20] D. Bittman, M. Gray, J. Raizes, S. Mukhopadhyay, M. Bryson, P. Alvaro, D. D. Long, and E. L. Miller, “Designing data structures to minimize bit flips on nvm,” in *2018 IEEE 7th Non-Volatile Memory Systems and Applications Symposium (NVMSA)*, 2018, pp. 85–90.

[21] “Nvidia management library (nvml),” Jan 2021. [Online]. Available: https://developer.nvidia.com/nvidia-management-library-nvml

[22] “Dimacs data sets.” [Online]. Available: https://www.diag.uniroma1.it/challenge9/download.shtml

[23] [Online]. Available: https://iss.oden.utexas.edu/?p=projects/galois

[24] J. Leskovec and A. Krevl, “SNAP Datasets: Stanford large network dataset collection,” http://snap.stanford.edu/data, Jun. 2014.

[25] T. A. Davis and Y. Hu, “The university of florida sparse matrix collection,” *ACM Trans. Math. Softw.*, vol. 38, no. 1, Dec. 2011. [Online]. Available: https://doi.org/10.1145/2049662.2049663