BVF: Enabling Significant On-Chip Power Savings via Bit-Value-Favor for Throughput Processors

Ang Li
Pacific Northwest National Lab
ang.li@pnnl.gov

Wenfeng Zhao
University of Minnesota
zhaow@umn.edu

Shuaiwen Leon Song
Pacific Northwest National Lab
shuaiwen.song@pnnl.gov

ABSTRACT

Power reduction is one of the primary tasks for designing modern processors, especially for high-performance throughput processors such as GPU due to their high power budget. In this paper, we propose a novel circuit-architecture co-design scheme to harvest enormous power savings for GPU on-chip SRAM and interconnects. We propose a new 8T SRAM that exhibits asymmetric energy consumption for bit value 0/1, in terms of read, write and standby. We name this feature *Bit-Value-Favor* (BVF). To harvest the power benefits from BVF on GPUs, we propose three coding methods at architectural level to maximize the occurrence of bit-1s over bit-0s in the on-chip data and instruction streams, leading to substantial chip-level power reduction. Experimental results across a large spectrum of 58 representative GPU applications demonstrate that our proposed BVF design can bring an average of 21% and 24% chip power reduction under 28nm and 40nm process technologies, with negligible design overhead. Further sensitivity studies show that the effectiveness of our design is robust to DVFS, warp scheduling policies and different SRAM capacities.

CCS CONCEPTS

• **Computer systems organization** → **Single instruction, multiple data**; • **Hardware** → **Static memory**; **Power estimation and optimization**; Dynamic memory;

KEYWORDS

BVF, SRAM, Power, Energy, GPU, bit, transistor, 8T, 6T, encoder, decoder, ISA, bus, toggle, value simiarity, Hamming

ACM Reference format:
Ang Li, Wenfeng Zhao, and Shuaiwen Leon Song. 2017. BVF: Enabling Significant On-Chip Power Savings via Bit-Value-Favor for Throughput Processors. In *Proceedings of MICRO-50, Cambridge, MA, USA, October 14–18, 2017,* 14 pages.
https://doi.org/10.1145/3123939.3123944

1 INTRODUCTION

Despite Moore's Law is continuing, the fundamental tradeoff between performance scaling and power capping is increasingly becoming one of the primary concerns for modern processor designs [1]. From datacenters to mobile devices, it has been widely accepted that power efficiency or performance per watt is more crucial than raw performance when conducting system evaluation [2, 3]. This is especially the case for supercomputers [4, 5]. For instance, the national roadmap for Exascale computing has established a goal of achieving exaflop/s under 20 MW by 2023 [6], which corresponds to 50 Gflops/W. To achieve this goal, modern supercomputers heavily rely on massive amount of throughput accelerators, such as GPUs, to deliver high performance at a lower power budget. Figure 1 shows the power efficiency of NVIDIA Tesla series of GPUs (primarily designed for the HPC market) in each architecture generation according to their release date. We can observe that the power efficiency of modern GPUs continues to increase and has already surpassed 50 Gflops/W target in last year. While manufacturing process scaling plays an important role in raising processor's power efficiency, continuously capping the power budget under certain level (i.e., 300W) also contributes significantly to this fast growth.

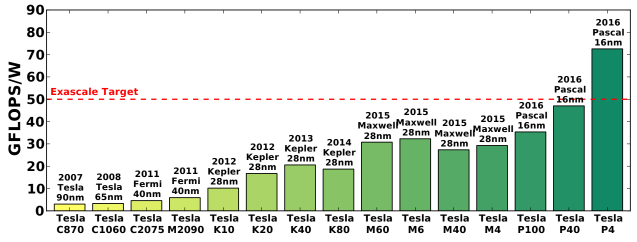

Figure 1: GPU Power Efficiency

To keep data close to the computing units, modern processors employ a hierarchy of on-chip memory, including registers, levels of caches, scratchpad memory, and so on. To connect these memory as well as the banks, arrays, mats and subarrays inside them, network-on-chip (NoC) and buses (e.g., H-tree) are also integrated [7]. These memory units and interconnects dominate chip's area, transistor count, and power consumption [8–10]. As on-chip memory are mostly SRAM due to its low access latency and high bandwidth, how to effectively shrink their power consumption remains a challenge [11]. This becomes a more critical design issue for GPU since it devotes a large portion of its chip area to various SRAM units such as huge register files.

One of the most effective approaches for reducing chip power is to lower the supply voltage, as power scales quadratically with V_{dd}. This is particularly true for SRAM memory as they usually determine the entire chip's supply voltage [8–10, 12]. However, reducing SRAM's V_{dd} has been very challenging since the traditional 6T SRAM cell may fail under lower supply voltages [13] due to the conflicting sizing requirement for read-stability and writability [14, 15]. With process scaling, this problem is exacerbated due to the more prominent effects of parameter variations [16] and telegraph noise [17]. To mitigate this reliability issue, 8T SRAM cell has been proposed as an appealing alternative for contemporary and future on-chip memory design [18, 19] because of its non-destructive read feature.

In this paper, we observe a special feature of the 8T SRAM cell: it consumes less energy for reading bit-1 than bit-0. We label this feature as *Bit-Value-Favor* or *BVF*. Meanwhile, we propose a BVF SRAM featuring asymmetric power consumption when writing bit-1 over bit-0. Additionally, low-level simulation demonstrates that the idling leakage for storing bit-1 is also smaller than bit-0.

MICRO-50, October 14–18, 2017, Cambridge, MA, USA
© 2017 Association for Computing Machinery.
ACM ISBN 978-1-4503-4952-9/17/10…$15.00
https://doi.org/10.1145/3123939.3123944

Putting them all together, we construct a novel 8T-SRAM array that has energy preference on bit-1 over 0 in terms of *read*, *write* and *standby*.

BVF-enabled circuit design can be applied to any modern processors (e.g., CPU, GPU, DSP, etc) with a reasonable size of SRAM. In this work, we use GPU to showcase the significant energy saving benefits from BVF-enabled designs because of GPU's large on-chip SRAM structures, application features (e.g., value similarity) and high power budget. To reinforce such asymmetric property of BVF for harvesting enormous power savings on GPU, we propose three architecture-level coders (encoders and decoders) to maximize the occurrence of bit-1 in data and instruction streams. Meanwhile, our coders can also reduce the toggling rate on the NoC and on-chip buses. In summary, our circuit-architecture co-design covers two of the three dominant power components in a modern processor: *storage* and *data movement*.

Contributions. This paper makes the following contributions:

- By leveraging the observation that 8T SRAM consumes less energy for reading bit-1 than bit-0 (i.e., BVF feature), we extend the same asymmetric feature onto the write operations of 8T SRAM by proposing a novel precharge circuit design. This design creates a unified 8T SRAM structure which presents BVF on both read and write. Additionally, rigorous low-level circuit simulation shows that the idling leakage for the new 8T SRAM design also exhibits BVF feature (Section 3).
- We offer architecture support to leverage and amplify the BVF property for modern GPUs. Based on reliable profiling results for *narrow values*, *value similarity* and *ISA-preference* on the latest Tesla-P100 GPU (Pascal) using 58 GPU applications, we propose three data coding approaches to maximize the occurrence of bit-1. Our method considerably increases the volume of bit-1s in the GPU data and instruction streams. As a positive effect, it also significantly reduces the bit toggles on the NoC connecting the SRAM units. All above substantially increase the energy-saving benefits from BVF, bringing enormous power reduction for the entire GPU chip (Section 4).
- Evaluation results show that our BVF-based circuit-archi-tecture co-design can effectively decrease the overall GPU chip power by 21% and 24% under 28nm and 40nm process, respectively, with negligible design overhead. We also demonstrate that the energy reduction benefits achieved by our design is not significantly affected by different process technologies, warp schedulers or memory configurations (Section 5 and 6).
- We show that BVF is not a unique feature for the proposed BVF 8T SRAM, it also appears in other types of memory cells such as eDRAM and 6T SRAM (Section 7). This could fundamentally provide another useful knob for reducing the on-chip SRAM power orthogonal to DVFS, where the savings are from reduced switching activity.

2 BACKGROUND

We review the state-of-the-art on-chip SRAM designs and their power consumption characteristics, including both 6T and 8T design alternatives. Furthermore, we extend our discussion based on an interesting observation over the read access power consumption of the 8T SRAM design.

2.1 6T SRAM Design

The conventional 6T bitcell is illustrated in Figure 2. A 6T bitcell consists of two cross-coupled CMOS inverters (*P0*, *N0* and *P1*, *N1*), forming a bi-stable circuit with positive feedback to hold binary data of either 0 or 1. Two access transistors (*N2* and *N3*) connect the

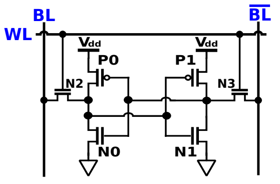

Figure 2: 6T SRAM cell

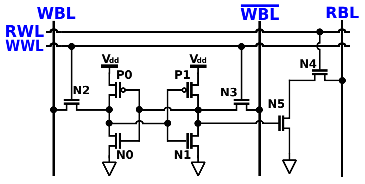

Figure 3: 8T SRAM cell

complementary storage nodes to two differential bitlines (*BL* and $\overline{BL}$), while the gates of the access transistors are controlled by the wordline signal *WL*. Both read and write operations require bitlines to be precharged to V_{dd} via the precharge logic controlled by $\overline{PC}$, then the *WL* signal is asserted, allowing the bitcell to be accessed via the differential bitlines. Here, it is assumed that the left node stores 1. During the read operation, *BL* remains high after precharge, while $\overline{BL}$ is discharged via the access transistor *N3* and the pull-down transistor *N1*. The voltage difference is then amplified and latched by the sense-amplifier, finishing a successful read operation. For write operation, bitline write drivers charges/discharges both bitlines to the desired states, overwriting the bitcell contents with the present data when the *WL* signal is asserted.

Adopting 6T SRAM design in on-chip memory (e.g., registers, caches, scratchpads, etc) ensures both high density and high performance, but its high and ever-growing percentage of total chip area contributes to significant portion of chip's power consumption. Lowering the power supply voltage would lead to considerable power savings, however, this poses potential concerns to the 6T SRAM design. As is generally known, the successful read and write accesses of 6T SRAM rely on the ratioed operation inside the bitcell. The conflicting sizing requirements make the design of 6T SRAM cells extremely susceptible to manufacturing-induced device variations [14], especially under scaled process nodes [15]. Voltage scaling further exacerbates the situation as the *static noise margin* (SNM[1]) is reduced with lowered supply voltage, which is undesired for DVFS.

2.2 8T SRAM Design

8T SRAM bitcell is proposed as an alternative solution to resolve the reliability issue of 6T SRAM bitcell [14, 15, 20]. Compared to a 6T bitcell, as shown in Figure 3, an 8T bitcell adds two transistors (*N4* and *N5*) and one additional read bitline (*RBL*), which forms a dedicated read-out buffer. The write operation of an 8T bitcell is identical to that of an 6T bitcell. However, the read operation of 8T bitcell is achieved via the 2T read-out buffer, which is an advantage over the 6T bitcell design. This modification ensures decoupling of the internal storage node from *RBL*, thus eliminating the read stability problem and the read/write sizing conflicts. Without such read disturbance, the SNM can be significantly enhanced. The improved reliability of the SRAM cell allows faster speed, lower voltage operation and promising technology scaling trajectories [20]. Regarding **performance**, it is shown that 8T SRAM is advantageous over 6T design if the access transistors are strengthened [20]. In future technology scaling, 8T is predicted to be the ideal choice for memory cell under 7nm process [15]. Besides, decoupling the read and write bitlines ensure dual-port operation, allowing read and write operation achieved in one cycle (i.e., *1R1W*). This is particularly beneficial for high-speed on-chip registers files [21, 22]) and caches [14] in GPUs. Regarding **area**, although an 8T cell is reported to have approximately 30% penalty over a dense 6T cell and $\sim$20% over a high-performance 6T cell, this cell size penalty is expected to be lowered or even disappeared with technology and voltage

[1]SNM is defined as the minimum voltage noise to flip the cell state.

scaling [14, 15]. Furthermore, recent work reported that 8T can have even smaller area under 7nm [15]. Regarding **power consumption**, several works [15, 23, 24] have reported that, with voltage scaling (e.g., from 1.2V to 0.41V), the leakage current of a Power processor can be reduced by a factor of over 60X [14]. Note that both 6T and 8T are potential design options for GPU on-chip SRAM. In fact, major foundry SRAM compilers support both 6T and 8T SRAM, while all the major GPU vendors have filed patents for 6T and 8T, including NVIDIA [18], AMD [19] and Intel [25]. In summary, 8T's area overhead can be significantly reduced under future process technology and it is much more sustainable to deep DVFS, thus becoming a more competitive choice for low-power SRAM design.

2.3 SRAM Power Consumption

The power dissipation of a general CMOS circuit (including SRAM) can be described by the following equation [26]:

$$P = \alpha C V_{\mathrm{dd}}^2 f_{clk} + V_{\mathrm{dd}} I_{leakage} \tag{1}$$

where the first and second expressions correspond to the *dynamic* power and *leakage* power, respectively. α is the switching activity factor, C is the lumped total capacitance, V_{dd} is the supply voltage and f_{clk} is the clock frequency. As can be observed from Eq. (1), introducing voltage scaling can bring quadratic dynamic power savings and linear leakage power savings. This is the major benefit of DVFS technique in the architectural designs for both GPUs and CPUs. DVFS scheme would be valid assuming the performance penalties are not a concern. However, this would not always be the case for real-world applications, where nominal supply voltage and high operating frequency are needed. As a result, techniques orthogonal to DVFS would be highly desired.

SRAM array consists of decoders, wordline drivers, bitline drivers, precharge logic, bitcells, sense amplifiers, and control logics, etc. The column-wise organization of bitlines are shared by a large amount of bitcells (e.g., up to 128 or 256), which contributes to the major array area and power consumption (due to large bitline parasitic capacitance). It is reported that more than 50% dynamic power of SRAM is consumed by the bitlines [27]. As such, reducing the switching activity α can lead to proportional power savings in bitline power.

Figure 4 (A) and (B) shows the bitline architecture and bitline swing behaviors of 6T SRAM and 8T SRAM. The differential bitlines in 6T SRAM are precharged before access (either write or read), and there is always one bitline discharged to ground regardless of the data to be written in or read out. The implication is that accessing power for data 1 and 0 are equal in conventional 6T SRAM. This also holds true for the write access of 8T SRAM. However, the single-end read port of 8T SRAM offers unique opportunity for read access power reduction. The *RBL* is precharged to V_{dd} during each read cycle in 8T SRAM, while the *RBL* voltage is determined by the data on the storage node Q. When Q equals 0, the *RBL* discharges and consumes dynamic power. However, *RBL* remains V_{dd} if Q equals 1, leading to almost no dynamic power consumption. Such data-dependent power consumption are generally overlooked in previous SRAM power models and simulators. Inspired by this, we aim to exploit such unbalanced power consumption at different abstraction levels, from circuit-level modification to architectural level support to optimize the power consumption of on-chip memory and interconnect in GPUs.

3 BIT-VALUE-FAVOR

BVF is an asymmetric property describing the unbalanced energy consumption in CMOS circuitry for accessing bit value 0 and 1. In this section, we describe how to achieve BVF with revised 8T SRAM featuring BVF bitline precharge scheme. Furthermore, we leverage architectural innovation to support BVF on NoC interconnects.

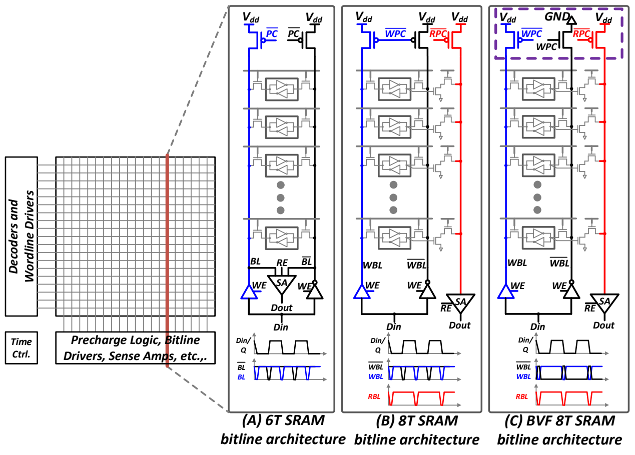
Decoders and Wordline Drivers
Time Ctrl.
Precharge Logic, Bitline Drivers, Sense Amps, etc.,.
Vdd
PC
BL
RE
WE
SA
Dout
Din
WPC
RPC
WBL
GND
(A) 6T SRAM bitline architecture
(B) 8T SRAM bitline architecture
(C) BVF 8T SRAM bitline architecture

Figure 4: Power consumption characteristics of SRAM with respect to different bitline architectures: (A) 6T SRAM, (B) 8T SRAM, and (C) BVF 8T SRAM

3.1 BVF 8T SRAM Design

As discussed in Section 2, the conventional 8T SRAM exhibits asymmetric read power consumption depending on the bit value stored. When $Q = 1$, the read bitline consumes much less power. Therefore, maximizing the occurrence of *1s* in SRAM would lead to maximal power benefit.

Conventional 8T SRAM shows barely different power consumption when writing bit 1 and 0. We aim to extend the BVF feature to write operation by introducing a novel 8T SRAM design with modified precharge logic. As illustrated in Figure 4-(C), we replace one precharge transistor from PMOS to NMOS so as to precharge *WBL* to V_{dd} and $\overline{WBL}$ to ground. This modification has no area penalty and speculatively presumes to favor writing bit value 1. If hit (it is 1), both bitlines consume little power. If miss (it is 0), however, the power consumption doubles. It is worth noting that, such a speculation has no influence on overall power consumption with 50% 0s and 50% 1s, but does impact a lot when architectural-level support is enforced to ensure sufficient high hit rate.

Figure 5 and 6 show the simulation results of the normalized access energy consumption of 8T SRAMs under three scenarios, where "*Avg*" represents conventional assumption (read and write 1 consume identical energy compared to read and write 0), "*Conv-8T*" represents the actual case where read 1 consumes less energy than read 0, and "*BVF-8T*" represents the proposed BVF-8T SRAM with both read and write 1 consume less energy than read and write 0, respectively. We also show the 6T results for reference. The simulation is performed in two commercial CMOS processes (28nm and 40nm, as described in Section 5) under nominal voltage of 1.2V (for 6T and 8T) and near-threshold voltage of 0.6V (for 8T only since 6T cannot operate). As can be observed, the energy benefit of access 1 other than 0 in the BVF 8T SRAM is obvious and consistent across different voltages and different technology nodes, while further efforts are needed to purposely increase the occurrence of data 1 through architectural supports. In addition, our simulation results also indicate that, the static leakage power consumption of BVF 8T SRAM is slightly reduced by 0.43% and 3.01% for storing bit 0 and 1 when compared to the original 8T SRAM design, respectively. This is partially due to the modified write bitline precharge logic, as only *WBL* (instead of two in original 8T SRAM) is connected to V_{dd} and thereby reducing one leakage path. In addition, storing

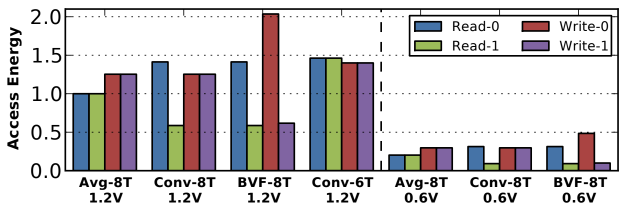

Figure 5: Energy for a single access under 28nm process. Set=32

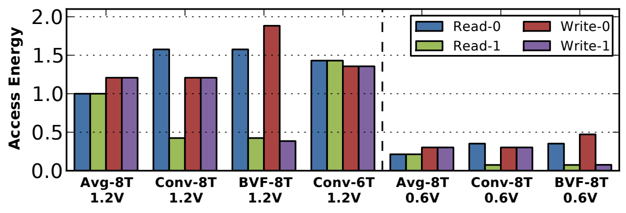

Figure 6: Energy for a single access under 40nm process. Set=32

1 costs 9.61% less power than storing 0. We can thus initialize the BVF SRAM cell to bit-1 so as to benefit first-time write and the unallocated SRAM resources during execution.

To the best of our knowledge, state-of-the-art memory simulators do not account for the power difference for accessing different data values, as illustrated in the BVF 8T SRAM design. Further extending this capability to the power models in these simulators not only yields to more accurate power estimation, but provides new insights for optimizing on-chip memory power by exploring the lower energy consumption of accessing data 1. As a result, we argue that if architectural support can be leveraged to enforce statistically more access of 1s over 0s, then we can gain energy reduction for read, write and standby. Essentially, the goal is to maximize the Hamming Weight within a data word when BVF 8T SRAM is adopted.

3.2 BVF Interconnects

There are two types of on-chip interconnect for data movement in modern processors: the *distribution network* within a storage device and the *communication network* among devices. Modern on-chip SRAM arrays are huge in capacity, therefore they are often partitioned into banks and further into subbanks, mats, subarrays and so on [28] for high throughput and low power purposes. These modules are interconnected by data distribution networks, i.e., the binary/H-tree networks in SRAM [29, 28], the crossbar networks in GPUs' register files and shared memory [30], etc. On the other hand, the communication among SRAM-based on-chip storage are achieved through NoC or data buses, e.g., the crossbar NoC between GPU SMs and L2 caches [31, 32], as shown in Figure 7.

In both scenarios, the data transmission energy on these interconnect wires are proportional to the *toggling rate* (i.e., the fraction of bits switching from 0 to 1 or 1 to 0 per data transfer in communication channels) of these interconnects [33, 34], due to the charging and discharging of the channel wire capacitance. Hereby the toggling rate corresponds to the activity factor α [34, 35] in Eq. (1).

The optimal scheme for reducing parallel bus power can be achieved via bus-invert code [36, 37], where random and uniform data distribution is assumed. One disadvantage of this scheme is that it uses additional parity-bit to indicate bus inversion where necessary. As a result, this would create significant overheads in memory design. In addition to this, bus-invert code aims to minimize the Hamming Distance between consecutive data words and poses no preference over the statistics of 1s and 0s within a data word. This is against the power saving mechanism of the BVF 8T SRAM, where Hamming Weight is maximized in a data word. In this work, we propose architectural-support BVF interconnects leveraging the data characteristics on the interconnects, which could maximize Hamming Weight per data word without inuring additional parity bits overheads and statically reduce the toggling rate due to increased occurrence of 1s, which leads to compatible design objective as the BVF 8T SRAM.

3.3 BVF Unified Objective Function

We define **BVF memory** as a piece of physical memory comprised of memory cells showing energy consumption asymmetry for bit value 0/1. **BVF space** thus describes a virtual memory space in which all the physical memory are BVF memory, showing the same preference on bit-value (e.g., BVF prefers bit-1 in our memory cell design) and share a unified coding format for data storage and communication in the space. In this way, all the function units in a BVF space can share the same "*wrapper*" for format conversion (i.e., encoding/decoding) when communicating with the external space, requiring no extra bit-line or formatting metadata. Therefore, **BVF optimization** is to design a coding system that maximizes the occurrence of 1s to reinforce BVF optimization for energy in a BVF space. We use mathematical expression to explain this process: if $\boldsymbol{B} = [b_0, \cdots, b_n]$ $(b_i \in (0,1))$ is the **B**aseline data in binary and $\boldsymbol{E} = [e_0, \cdots, e_n]$ $(e_i \in (0,1))$ is the **E**ncoded binary sequence, BVF optimization is to find a transformation $\boldsymbol{f} = \boldsymbol{B} \rightarrow \boldsymbol{E}$ to maximize $\sum_{i=0}^{n} e_i$ (*encoder*). Correspondingly, the reverse transformation is $\boldsymbol{f^{-1}} = \boldsymbol{E} \rightarrow \boldsymbol{B}$ (*decoder*).

There are two properties regarding to BVF space: (I) if a BVF space has multiple ports connected to external space, all the ports will adopt the same coding format (i.e., the same encoder and decoder); and (II) when different BVF spaces overlap, none of them affect each others' coding format and results. In other words, the encoding/decoding process of a BVF space should not impact how another independent space reconstructs its original data sequence. These principles will be applied to our BVF-enabled design on GPU.

4 BVF FOR GPU

Although BVF-directed design can be applied to any processor with SRAM (e.g., CPU, GPU, DSP and specialized accelerators), as the focus of this work, we choose to showcase the power-saving capability of BVF-enabled design using throughput-oriented GPU architecture due to three major reasons. First, GPU devotes a large portion of its chip area to various SRAM units, including register files (REG)[2], instruction cache (L1I), constant cache (L1C), L1 data cache (L1D), L2 cache (L2), texture cache (L1T) and scratchpad memory (SME). To achieve high bandwidth, most of these units are banked, interconnected by NoC (e.g., crossbar for registers and L2 cache banks). These units contribute significantly to the GPU's on-chip power consumption (~48%) [30]. Secondly, GPU applications are mostly data-parallel applications, showing strong correlation between neighborhood data elements, known as value similarity [39, 22, 40]. Lastly, GPU chips generally demand higher power budget compared to CPU, especially in a HPC setup. Reducing its power can drastically shrink the overall power envelope of the GPU-integrated HPC nodes.

In the following subsections, we will introduce three coder designs for BVF optimization on GPU: *narrow value*, *value similarity*

[2]Unlike that on CPU, GPU register files are very large and highly-banked dense SRAM arrays [38].

Table 1: Coder effective space. IFB stands for instruction fetching buffer.

BVF Coder	abbr.	BVF Space
Narrow Value	NV	REG, SME, L1D, L1T, L1C, NoC, L2
Value Similarity	VS	REG, L1D, L1T, L1C, NoC, L2
ISA Preference	ISA	IFB, L1I, NoC, L2

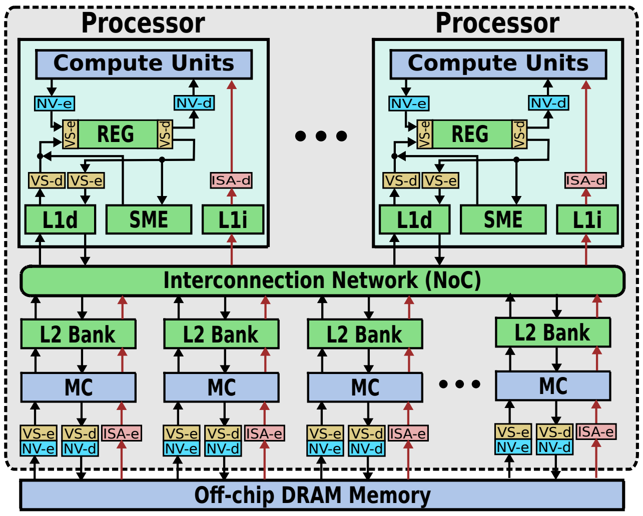

Figure 7: Overall architectural design. "REG" stands for register files. "L1d" is L1 data cache. "SME" is shared memory. "L1i" is L1 instruction cache. "MC" stands for memory controller. The black arrows indicate data stream while the red arrows refer to instruction stream. Our design is transparent to off-chip units.

and *ISA preference*. We use our novel 8T SRAM cell design introduced in Section 3 to explain our designs beyond this point. Other types of BVF memory cells will be briefly discussed in Section 7. Also, rather than dealing with each on-chip SRAM unit individually, we construct several BVF spaces so that (i) all the units in a space share the same set of encoder/decoder to reduce design overhead, and (ii) data transmission on the NoC of the space is unified (i.e., no extra bit-lines or metadata are required) to further reduce power (e.g., potentially reducing toggling rate on NoC when maximizing 1s). The on-chip hardware units in different BVF spaces that correspond to the proposed three coders are listed in Table 1. The coders' positions on GPU are marked in Figure 7. Note that our design does not impact off-chip bus or DRAM; it is transparent to off-chip units.

4.1 Coder I: Narrow Value (NV)

4.1.1 Motivation and Approach. A *narrow value* is a small value but occupies a large data type, e.g., a bool type stored as an integer. Narrow values frequently appear in application data, mainly due to three reasons: (1) *Over-provisioning*. Programmers often provision data types for extreme values despite most of them can be fit into a shorter data type [41]. (2) *Data alignment*. Modern processors often support limited bit-width. To be aligned for correct and high-efficient data access (e.g., coalescing on GPUs), smaller data types are usually padded to a larger unified data type. (3) *Performance consideration*. It is well-known that GPU delivers its best performance when processing single-precision (SP) floating-point numbers. Therefore, numerical-intensive GPU applications often convert integer or short-integer to SP floating-point for performance enhancement, e.g., *oceanFFT* and *imageDenoising* in CUDA SDK [42]. Narrow values appear for both integer and floating-point values; as for the latter, its exponent part comes before the mantissa [43].

Traditionally, optimizations for narrow values are conducted in two aspects: (i) from the computation perspective, determining the *effective-bit-range* (i.e., least number of bits to represent the data set) to help generate more efficient ASIC design in high-level synthesis [44], or shortening the data type in program compilation [45]; (ii) from the memory perspective, effective-bit-range is leveraged for data compression in caches [46, 41] and interconnects [47, 48]. All of them seek to truncate the redundant bits for power or performance gain on CPUs, but often suffer from outliers (i.e., values out of the chosen range) and high overhead on compile/runtime range analysis.

Unlike previous works, we utilize this narrow value feature in data to benefit our BVF-enabled design by maximizing 1's appearances in the bit strings on GPU rather than truncating. Figure 8 demonstrates this opportunity on GPU by showing the average number of leading 0s (and leading 1s for negative values) for 58 GPU applications from commonly-used GPU benchmarks (see Section 5) on state-of-the-art Tesla-P100 GPU [49] of a NVIDIA DGX-1 machine. The "*clz*" PTX instruction [50] is employed to count the bits of leading 0s for all the data values loaded from and stored to global memory. Negative values are bit-wise inverted before counting. If a warp branches, we only count the threads that take the branch when executing memory access instructions. We can observe that, for a 32-bit data word, there are an average of 9 leading bits that are 0s.

This observation suggests that it is potentially beneficial for power reduction if we invert these leading 0s to 1s based on the BVF feature. However, only flipping leading 0s requires memorization of how many bits are flipped at the encoder in order to correctly decode later, which complicates the design and incur substantial overhead. One simple alternative is to flip all the bits, including both leading 0s and the effective-bits. As long as the occurrence rate of 0s is higher than or even equal to 1s in the effective-bits (i.e., 32-9=23 bits), this simple strategy will benefit from BVF power reduction. To validate this assumption, we also profile the ratio of bit 0 and 1 in the data values of the 58 applications on Tesla P100, as shown in Figure 9. Note that 64-bit values such as double floating-point are divided as two 32-bit values in our profiling. The result shows that 22 out of 32 bits in a data-word are 0s on average, suggesting flipping all bits will reduce power for data access through BVF. We can also estimate the average ratio of 0 to 1 in the effective-bits as $(22-9=13):(32-22=10)$.

4.1.2 Implementation. The basic idea to implement the narrow-value coder (NV coder for short) is to *flip all the bits for positive data values but remain unchanged for negative data values.* This can be achieved by exclusive-noring (XNOR) the beginning bit (i.e., the leading sign bit) with the remaining bits. If the inversion of a bit is defined as $\overline{b_i} = |b_i - 1|$, XNOR of two bits can be defined as $b_i \overline{\oplus} b_j = |b_i - \overline{b_j}|$. Therefore, the NV encoder $\boldsymbol{f}$ is:

$$\boldsymbol{E} = \boldsymbol{f}(\boldsymbol{B}) = f([b_0, \cdots, b_n]) = [b_0, b_1 \overline{\oplus} b_0, \cdots, b_n \overline{\oplus} b_0]$$

An advantage of this design is that it is invertible, so the decoder is the same as the encoder:

$$\boldsymbol{B} = \boldsymbol{f^{-1}}(\boldsymbol{E}) = f^{-1}([e_0, \cdots, e_n]) = [e_0, e_1 \overline{\oplus} e_0, \cdots, e_n \overline{\oplus} e_0]$$

NV coder's implementation is depicted in Figure 10, which only consists of XNOR gates. As shown in Figure 7, the NV coders (i.e., NV-e and NV-d) are placed beneath the memory controllers as the lower interface and upon register ports as the upper interface. All the SRAM and NoC components in between the two interfaces are covered in the NV-coder BVF space for maximum BVF-induced power benefit (e.g., such design not only benefits SRAM structures but also NoC due to reduced toggling, since 5.6% of the total GPU dynamic power is consumed on interconnects [32]). From the perspective of architecture design in Figure 7, the NV encoders (NV-e) only affect bits inside data, without influencing other architectural components and how they function on GPU (e.g., execution flow). Similarly, the

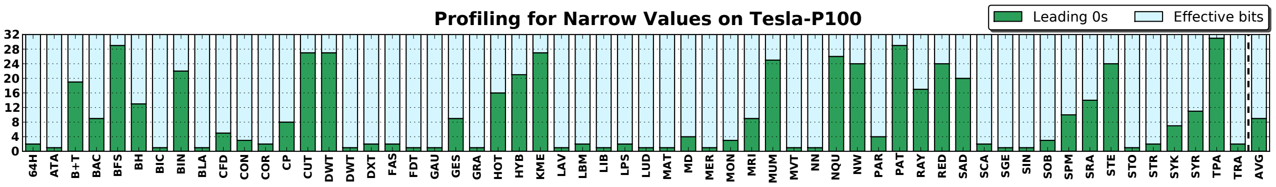

Figure 8: Narrow value profiling on Tesla P100.

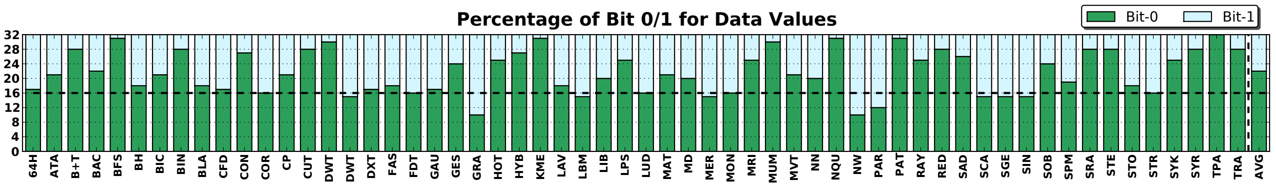

Figure 9: 0 and 1 ratio in the data values of the 58 applications running on Tesla P100.

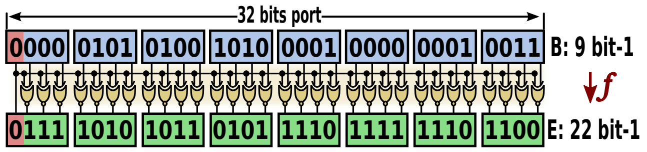

Figure 10: NV coder design. f is the BVF encoder or BVF-space write ports. f^{-1} is the BVF decoder or BVF-space read ports.

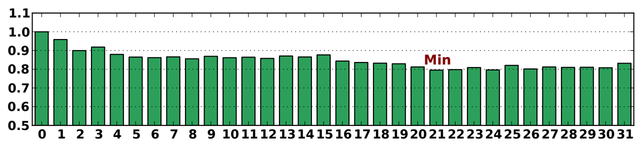

Figure 11: Normalized relative Hamming distance for 32 lanes (32 threads/warp) across 58 applications on Tesla P100.

NV decoders (NV-d) is only required to recover data format when data need to be processed by GPU compute units. Ideally, if GPU ISA can be adjusted to adopt the new data coding format, we will not need the upper interface above registers – the narrow value BVF space will then cover the entire GPU chip.

Note that BVF-inspired NV coder design is most effective when the frequency of data value 0 (all bits are 0) occurring in an application is high. Fortunately, previous studies [51–53] have suggested that value 0 is the most frequently encountered value in applications, e.g., 18% of the dynamic loads from application data turn out to be 0s for CPU SPEC applications [54], and can be as high as 62% for GPU deep-learning applications [55]. Our design can greatly benefit from this observation.

4.2 Coder II: Value Similarity (VS)

4.2.1 Motivation and Approach. Value similarity among warp lanes (i.e., the 32 lanes in a warp) has been recently exploited on GPU for the sake of performance and power efficiency [39, 22, 40]. *Value similarity* describes the scenario when two values only differ in their least-significant-bits (LSBs) of their binary representations [40]. In other words, the arithmetic distance is small. Meanwhile, *value locality*, which describes the recurrence of the previously accessed value, was also leveraged in GPU architectural design [56, 57], previous work have observed that 23.8% of the GPU instructions exhibit operand value similarity, while 34.2% exhibit operand value locality [40].

Essentially, both value locality and value similarity are aimed to eliminate the repeated bits of neighboring data elements so as to reduce operations, unnecessary memory transactions, area, as well as power for SIMD compute units [56, 57, 40], memory units [41, 22] and interconnects [58, 59]. In this paper, we also target on warp-level inter-lane similarly. However, instead of seeking to eliminate redundancy, our goal is to maximize bit-1s' appearances in the data for warp-level reads and writes. Hence, different from the existing value similarity works which adopt arithmetic similarity (measured by arithmetic distance), we apply *Hamming Distance* [60] (i.e., the number of positions at which the corresponding symbols are different between two strings of equal length) to reflect the 0/1 similarity of two bit strings. We further define the concept of **Hamming Similarity**: two data values are short in their Hamming distance. Note that there exists certain relevance between arithmetic distance and hamming distance. For instance, if two values are identical arithmetically, their hamming similarity is also the highest (i.e., hamming distance equals to 0). Furthermore, if two values are arithmetically similar, their most-significant-bits (MSBs) are generally the same, implying that their hamming distance is also likely to be small. But in some scenarios, they are quite different. For example, values `0x1000` and `0x0000` have high hamming similarity (i.e., hamming distance is 1) but their represented values are quite different arithmetically.

Another novel observation here is that the existing works about value similarity on GPU exclusively adopt the leading lane (i.e., lane-0) of SIMD as the base value or pivot, while the remaining lanes amend data values by adding delta on top of this base value. However, the leading lane is often not the optimal choice compared to the lanes approximately located in the middle, because the leading lane more likely suffers from branch divergence. Thus, both its hamming distance and arithmetic distance to the other lanes tend to be larger than the middle-positioned lanes. Figure 11 confirms this argument by profiling the average hamming distance of each lane to the other 31 lanes in the same warp for 58 GPU applications on Tesla-P100. As expected, lane-21, rather than lane-0, shows the smallest average hamming distance (i.e., an average of 20% hamming distance difference between lane-0 and lane-21). We apply this empirical observation in our design and denote lane-21 as the pivot lane for all the other lanes in the same warp to compare against. Note that all the discussion above is from the presecutive of registers. Since we cannot profile which data element (e.g., float, double, int) at cache line level can represent the pivot, we simply choose the leading element value (data element 0) in a cache line to represent its pivot value (data similarity within cache line).

Note that we apply this simple statistical average (lane-21) here from empirically profiling a large number of representative GPU applications for two purposes: (1) creating a simple and low-overhead

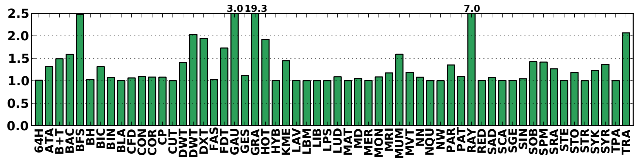

Figure 12: Normalized Hamming distance for lane-21 with respect to the optimal lane across 58 applications on Tesla P100.

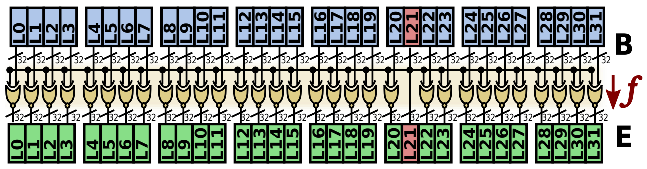

Figure 13: VS coder design from register's perspective. For cache, the design remains the same with only modifying the pivot to element 0 in that cache line.

design, and (2) showcasing that there is a design space here that can be explored instead of assuming lane-0 as the only pivot lane (adopted as default by existing studies [39, 22, 40]. However, it is possible that the optimal pivot-lane may be different for individual applications. To evaluate this, Figure 12 illustrates the Hamming distance of lane-21 with respect to the optimal lane for each application (i.e., the lane showing the minimal Hamming distance with the other lanes). As can be seen, for most applications, the choice of lane-21 is appropriate. Nevertheless, a more advanced design can be developed in future to dynamically adjust the pviot lane based on online or offline profiling for a particular application. However, properly managing the hardware and runtime overhead from extracting, storing and updating the pivot-lane information remains challenging.

4.2.2 Implementation. The value similarity coder (VS coder for short) based on hamming similarity can be designed as following. For each block of data fetched by a warp, all the non-pivot lanes perform XNOR with the pivot lane. In this way, *all the bits of the non-pivot lanes that are the same as the pivot lane are converted to 1*. If P is the **P**ivot data element in a data block. The encoder $\boldsymbol{f}$ (VS-e) can be described as:

$$\boldsymbol{E} = \boldsymbol{f}(\boldsymbol{B}) = \boldsymbol{B} \overline{\oplus} \boldsymbol{P} = [b_0 \overline{\oplus} p_0, \cdots, b_n \overline{\oplus} p_n]$$

From the registers' perspective, P is B_{21} based on our profiling. We obtain $E_{0-20,22-31}$ of a warp by XNORing $B_{0-20,22-31}$ with B_{21}. Since this encoder is also invertible, the decoder (VS-d) can remain identical:

$$\boldsymbol{B} = \boldsymbol{f}^{-1}(\boldsymbol{E}) = \boldsymbol{E} \overline{\oplus} \boldsymbol{P} = [e_0 \overline{\oplus} p_0, \cdots, e_n \overline{\oplus} p_n]$$

The implementation is depicted in Figure 13, utilizing only XNOR gates. With such a design, the maximum hamming similarity among the SIMD lanes are exploited to increase the appearances of 1s in each lane. Shown in Figure 7, the VS-coders are placed in two BVF spaces on the GPU chip: one is exclusively for registers (focusing on value similarity among lanes in a warp), one is for the other on-chip SRAM structures (e.g., L1D, L1C, L1T, L2) and NoC (focusing on value similarity among data in a cache line). VS-coders layout is not as simple as NV-coders' because value similarity attempts to extract the correlation among consecutive data rather than focusing on individual data, thus requiring handling data access divergence on GPU.

There are three types of divergence needed to be addressed on GPU: (A) *memory divergence*, when the requested data from a warp distribute across several L1 cache lines. It brings consistency issues as the pivot value used for encoding at MC port (see Figure 7) can be different from the pivot value for registers after data has been fetched into registers (e.g., the register of the 21st lane in the warp). If the original pivot information is lost, we can no longer correctly reconstruct the original data. To avoid this problem, we decode at L1 cache before data for warp lanes are collected from multiple cache lines. Note that for both L1 cache read and write, the updating of cache-line based pivot to register-based pivot in this design does not add extra delay to the critical path. This is because for read, the pivot is already available when fetched cache line fills the cache, so no extra pivot access is required. For write, since L1 cache of GPU adopts write-nonallocate write-evict policy [61, 62] (i.e., the entire L1 cache line is invalidated and written into L2), the pivot will be accessed regardlessly. (B) *branch divergence*, when partial registers of a warp are accessed in a branch. Under this condition, read will not cause any problem. But for write, if the pivot is the written target, all the other lanes may lose the original pivot information for reconstructing the original data for these lanes. To address this issue, we apply a similar method proposed by previous works [22, 40] which basically adds a dummy `mov` instruction for other lanes to first decode the data using the original pivot before it is written and then re-encode the data with the new pivot. Those studies also suggest this scheme encounters negligible overhead. (C) *shared memory divergence*, when the warp read from or write to the shared memory in an irregular way. Different from the register file in which the access is always coalesced, the shared memory access can be rather random and messy. Correctly tackling it will incur tremendous overhead. Thus we do not apply any VS-coders for the shared memory. As listed in Table 1, the value-similarity BVF spaces do not include SME.

Table 2: ISA preference masks for different generations of NVIDIA GPUs

Architecture	CC.	ISA Mask
Fermi	**2.0**	`0x4000-0000-0001-9c03`
Kepler	**3.7**	`0xe080-0000-001c-0012`
Maxwell	**5.0**	`0x4818-0000-0007-0205`
Pascal	**6.0**	`0x4818-0000-0007-0201`

4.3 Coder III: ISA Preference (ISA)

4.3.1 Motivation and Approach. The previous two coder designs focus on data stream. In this subsection, we propose a design for instruction stream. Compared to patterns of data values which are often volatile, the instruction values in instruction caches, instruction fetching buffers (IFB) and L2 are more fixed, as they are completely dictated by ISA design. The occurrence frequency of 0/1 in instructions is only dictated by the application code binary which is mostly independent of the actual execution context at runtime (e.g., loop iterations will not impact unless being unrolled). This observation implies that the BVF of instruction bits for an application can be assessed statically at compile time. In other words, when the executable binary is generated, we know which bit position preference on 0 and 1. To see the value patterns of GPU instructions, we analyze the 64-bit instruction binaries of 58 GPU applications, which consists of more than 130,000 instruction lines. The results for Tesla-P100 are shown in Figure 14. One major observation from this figure is that the ISA has strong preference on bit value for different bit positions in an instruction (e.g., the 25th bit favors 0).

The basic idea here is to perform a statistical analysis on the bit-position preference of GPU application instructions, and generate an average mask. The mask is then XNORed with all instructions to maximize bit-1s. Figure 14 profiles such statistical analysis on bit-position preference for the instructions of the 58 GPU applications. To extract the mask, *if a bit position generally prefers 0, the bit in the mask of that position is set to 0; otherwise, the mask is set to*

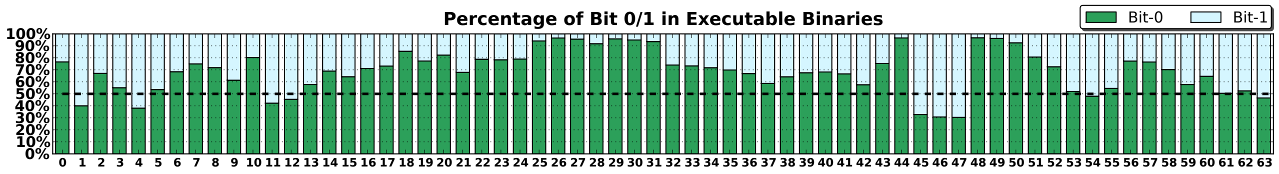

Figure 14: 0/1 occurrence possibility on every bit position among all the instruction binaries from 58 GPU applications. X-axis represents the 64 positions of 64-bit GPU instructions. Most positions prefer 0.

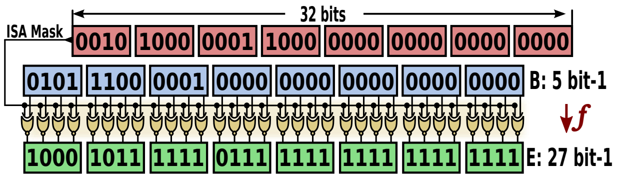

Figure 15: ISA BVF coder design.

System Overview	15 SMs, 32 threads/warp, 700MHz, 5-stage pipeline
Resources/SM	48 warps, 128KB REG, 48KB SME, 32 MSHRs
L1 Cache/SM	16KB/SM, 128B line, 4-way assoc
Shared Mem/SM	48KB, 32 banks
NoC	32B flit size
Unified L2 Cache	768KB, 128KB/bank, 6 banks, 128B line, 16-way assoc
DRAM	6 memory channels, FR-RCFS scheduling
Warp Scheduling	Greedy-then-oldest (GTO), round-robin (LRR), two-level

Table 3: Baseline architecture configuration.

1. Using this approach, we can obtain the mask for Pascal architecture: `0x4818-0000-0007-0201`. The suitable masks for other NVIDIA GPU architectures are also calculated and listed in Table 2. Note that GPU ISA changes with architecture generations.

4.3.2 Implementation. There are two ISA coder implementation methods: (1) A static method which directly applies the statistical masks listed in Table 2 at the ISA BVF-space lower interface (MC ports), shown as ISA-e in Figure 7; (2) A dynamic approach which adds a 64-bit *mask register* per ISA coder. The mask is then produced at compile-time, during which the assembler (e.g., ptxas) counts the occurrence of 0/1 in the generated binary, formulates the mask, and configures the mask register when kernel launching. After that, instructions perform XNOR with the mask before instruction decoding in cores. Although the dynamic method can provide more customized ISA preference optimization for individual application, it encounters larger design overhead (i.e., the mask registers) and compile/runtime latency (e.g., configure mask register at runtime). Thus we choose the simple static design to form a more unified BVF space. For ISA preference encoder $\boldsymbol{f}$ (ISA-e),

$$\boldsymbol{E} = \boldsymbol{f}(\boldsymbol{B}) = \boldsymbol{B} \overline{\oplus} \boldsymbol{M} = [b_0 \overline{\oplus} m_0, \cdots, b_n \overline{\oplus} m_n]$$

Again, this encoder is invertible. So the decoder (ISA-d) can remain identical:

$$\boldsymbol{B} = \boldsymbol{f}^{-1}(\boldsymbol{E}) = \boldsymbol{E} \overline{\oplus} \boldsymbol{M} = [e_0 \overline{\oplus} m_0, \cdots, e_n \overline{\oplus} m_n]$$

The actual ISA-coder implementation is shown in Figure 15. Note that the figure only illustrates the first half (32bits) of the coder due to space limitation.

5 EVALUATION METHODOLOGY

Circuit-Level Simulation. Existing memory simulators such as CACTI [7] incorporate SRAM power models which assume equal power consumption for accessing bit 0 and 1. Hence, rigorous low-level reliable simulations have to be performed to extract useful power consumption information. We built the proposed BVF 8T SRAM array (including bitcell, bitline precharge, sensing amplifier, wordline driver, decoder, etc) and the conventional 8T SRAM array using commercial *Process Design Kits (PDK)* of 28nm and 40nm technology nodes. Both were designed in schematic entry with Cadence Virtuoso Design Environment [63] and simulated using Spectre Simulator [64]. All devices were sized under logic rule constraints, as are available in the commercial PDK. Moreover, we performed extensive simulations with respect to the memory configurations used in modern GPUs (i.e., varying sets and block sizes). In addition, both 8T SRAMs are simulated with supply voltage ranging from nominal voltages (1.2V) to near-threshold voltages (0.6V).

Architecture-Level Simulation. We evaluate the impact of our BVF enabled coder designs on GPU by modifying GPGPU-Sim v3.2.1 [65] to dump the access trace (including target addresses, SM-id, warp-id, lane-id, L2-bank-id, access type, data content, etc) of all the reads, writes and fills to the BVF units, including L1 instruction cache, L1 data cache, constant cache, texture cache, register, shared memory, NoC and L2 cache. Table 3 shows the major simulation parameters of the GPU architecture used in this paper. To analyze huge generated traces (e.g., up to tens of GBs per application) due to data-oriented study, we developed a parser to process the accesses to each BVF unit. For the SRAM units, we count the volume of bit 0/1 in the data contents in terms of reads and writes. For NoC, we count the volume of bit transition for every two consecutive flit transmission in the same channel. Finally, we implemented the three proposed coders and analyzed the new statistics with these coders enabled. To evaluate the energy consumption of the entire GPU chip and its individual hardware components, we resort to GPUWattch [30] with a default 65nm process technology and scaled it to 28nm/40nm based on process technology and frequency.

Evaluated Applications. We have evaluated a large group of fifty-eight GPU representative applications from Rodinia [66], Parboil [67], SDK [42], Shoc [68], Lonestar [69], Polybench [70] and GPGPU-Sim [65]. If an application contains multiple kernels, we sum up the statistics. In our experiments, all the workloads run to completion on the simulator.

6 RESULTS AND ANALYSIS

In this section, we quantitatively evaluate our proposed BVF-enabled circuit-architecture co-design on GPU with the following subjects: (i) impact on energy consumption at both component-level and chip level; (ii) sensitivity study involving DVFS, warp schedulers and SRAM capacity; and (iii) design overhead analysis.

6.1 Impact on Energy Consumption

Component-Level Energy Reduction. With the number and energy consumption of accessing bit 0/1 before and after applying each proposed coder, we calculate the component-level energy consumption difference with and without our proposed design on GPU. The normalized energy consumption for each BVF unit under 28nm and 40nm process technologies is illustrated in Figure 16 and 17. All the data points are normalized to individual component's baseline scenario (before applying any BVF coder). From the figure, we can obverse that our BVF-enabled coder designs are very effective on

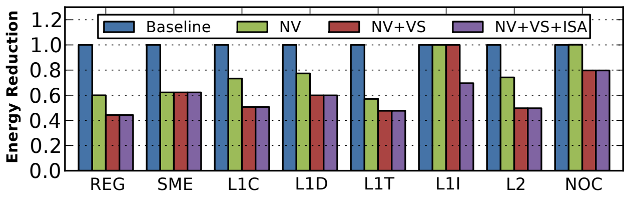

Figure 16: Average energy reduction for individual on-chip units under 28nm.

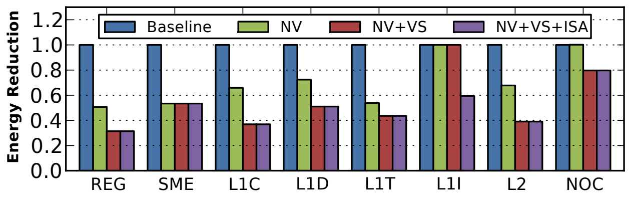

Figure 17: Average energy reduction for individual on-chip units under 40nm.

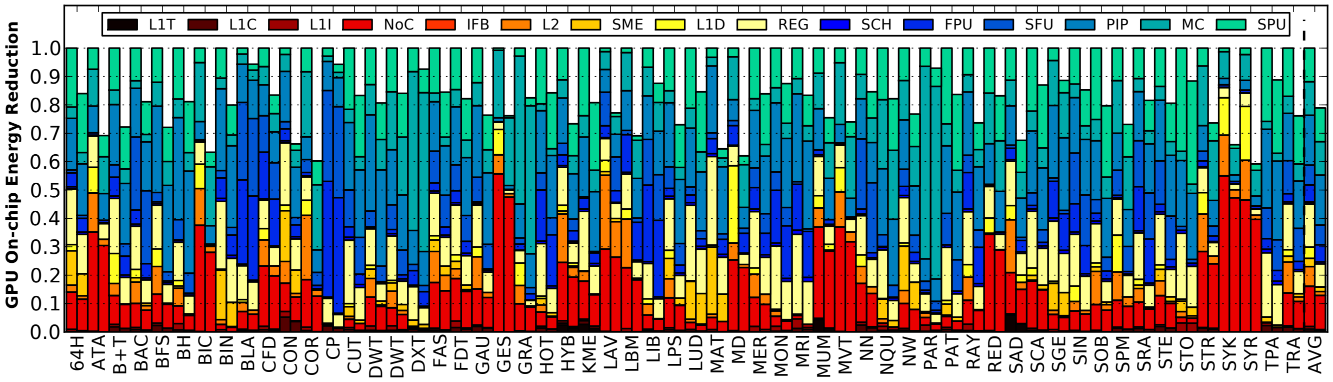

Figure 18: 28nm GPU chip-level energy reduction under three BVF coders. The average GPU chip energy reduction is around 21% across 58 applications. Each application has two bars: the left represents the baseline; the right represents the BVF design. The colors in a bar break down the different on-chip units of a GPU. The warm colors (from black to yellow) on the bottom side of each bar indicate the units that can largely benefit from BVF (e.g., SRAM units and NoC). The cool colors (from blue to green) on the top side of each bar refer to the units that are insensitive to BVF (e.g., computation units and MC). The rightmost bar titled with"AVG" is the average energy reduction percentage across the 58 applications.

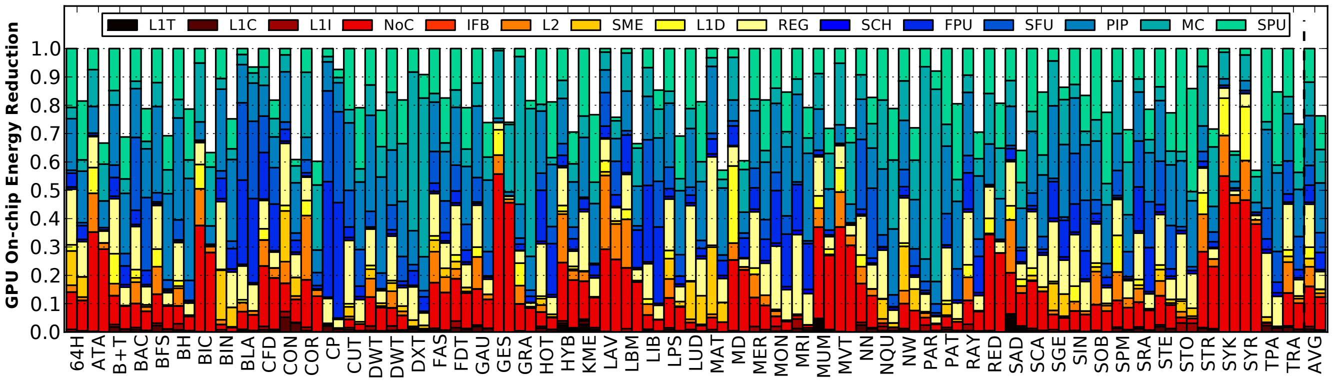

Figure 19: 40nm GPU chip-level energy reduction under three BVF coders. The average GPU chip energy reduction is around 24% across 58 applications.

reducing the energy consumption of on-chip SRAM structures and NoC, with different focuses based on their designated BVF space(s). For instance, NV-coders impact a large BVF space, covering all the on-chip SRAM structures and NoC (Figure 7). They exhibit impressive energy reduction on register files (40%), shared memory (38%) and texture cache (42%). Additionally, NV-coders have no impact on L1 instruction cache and NoC for quite obvious reasons: these coders only impact on data stream and data-bits inversion enabled by them does not change toggling rate in NoC. For VS-coders, they also cover a large BVF space, making them effective in bringing down the energy consumption of register file, L1C, L1D, L1T, L2 and NoC. Similar to NV-coders, VS-coders have no impact on the hardware units that are not included in their BVF spaces, such as shared memory and L1 instruction cache. Moreover, ISA-coders seem to only reduce energy for the instruction cache although their BVF space also includes L2 and NoC (see Table 1). This is because instruction stream only occupies a tiny fraction of the entire data stream through NoC and L2.

We can also find that the power consumption of NoC is reduced by about 20% (Figure 16 and 17), which mainly benefits from VS-encoder. This ensures a large amount of 0s converted into 1s and eventually leads to considerable amount of toggling rate reduction, which is due to the fact that consecutive transmitted flits in NoC have the same bit value (mainly 1s). Note that our evaluation adopts a conservative flit size of 32B (Table 3) as existing works [65, 35]. However, we expect that a larger flit size can further reduce the NoC toggling rate and lead to more aggressive power savings. Furthermore, the objective function of BVF optimization is to maximize bit-1s in the data stream, which is different from parallel bus toggling-rate reduction via maximizing consecutive 1s or 0s in flits. Although the former can partially contribute to the latter, they are not completely overlapped. This implies a target function that can be

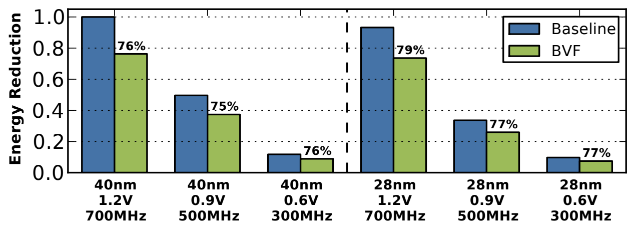

Figure 20: Normalized average energy reduction under DVFS. All data points are normalized to 40nm 1.2V (700MHz). The percentage numbers on BVF bars are further normalized to their corresponding baseline bars.

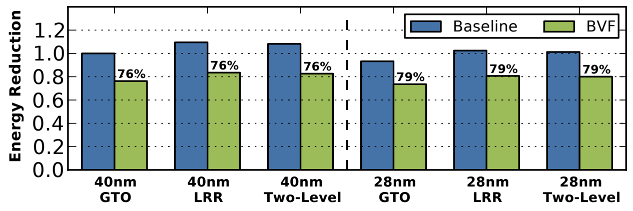

Figure 21: Normalized average energy reduction under various warp schedulers. All data points are normalized to 40nm GTO. The percentage numbers on BVF bars are further normalized to their corresponding baseline bars.

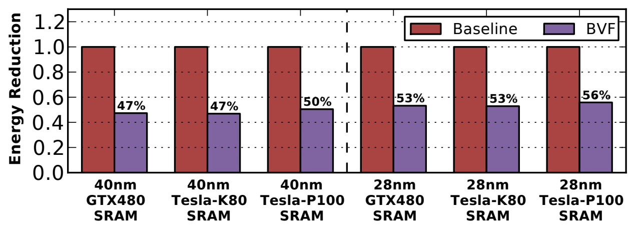

Figure 22: Normalized average SRAM power reduction under different SRAM configurations.

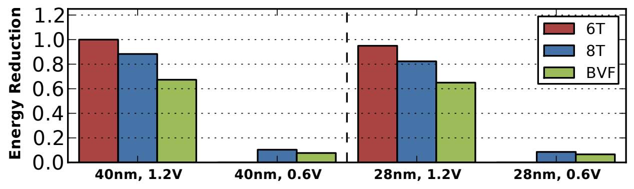

Figure 23: Normalized average energy reduction for 6T, 8T and 8T-BVF with nominal voltage (1.2V, 700MHz) and near-threshold voltage (0.6V, 300MHz). All data points are normalized to 40nm 1.2V 6T SRAM.

more effective to unify these two objectives may lead to even higher power reduction (e.g., via pareto optimization), but this is beyond the scope of this paper.

Chip-Level Energy Reduction. Figure 18 and 19 show the average GPU chip-level energy reduction breakdown across 58 applications under 28nm and 40nm, respectively. As can be seen, our BVF-enabled circuit-architecture co-design enables significant chip energy reduction for memory-intensive applications (i.e., incurs more memory-hierarchy access and NoC transactions) such as ATA, BFS, BIC, CON, COR, GES, SYK, SYR and MD, but less impressive reduction for compute-intensive applications such as BLA, CP, DXT, LIB, NQU, PAR, PAT and SGE. On average, our BVF design reduces the total energy consumption of on-chip BVF units under 28nm and 40nm by 47% and 53% respectively, corresponding to a **21%** and **24%** energy reduction of entire GPU chip. Note that although we only show big SRAM units such as cache and register, our technique also can benefit various queues, buffers, buses, tables and special registers on-chip.

6.2 Sensitivity Study

(A) Impact of DVFS. Due to the short-channel effects in deep-submicron processes, leakage current reduces dramatically with voltage scaling [71]. Therefore, the combined effect of voltage scaling and declined leakage current lead to significant power reduction. The energy difference of accessing bit 0/1 may change under voltage and frequency scaling, as already discussed in Section 3. Figure 20 shows the normalized average on-chip energy reduction for 28nm and 40nm under DVFS across all the 58 applications. We tested three P-states: 700MHz, 500MHz and 300MHz, corresponding to 1.2V, 0.9V and 0.6V, respectively. As can be seen, the energy saving percentage under our BVF design is consistent to DVFS.

(B) Impact of Warp Schedulers. Different warp schedulers affect the memory access sequence for the SRAM units and NoC, potentially affecting the energy behavior of these on-chip units. Thus we evaluate the impact of three commonly adopted GPU warp schedulers on our design's efficiency, including the Greedy-Then-Oldest (GTO, which is the baseline shown in Table 3), Loose Round-Robin (LRR) and Two-Level scheduler [72]. The results of the normalized average energy reduction for 28nm and 40nm GPU chip under three schedulers across all the 58 applications are demonstrated in Figure 21. We can observe that, although LRR and Two-Level incur slightly higher baseline chip energy consumption than GTO, the effectiveness of our design which is reflected by the BVF energy reduction ratio against individual baseline remains pretty consistent across different process technologies and warp schedulers.

(C) Impact of SRAM Capacity. We also would like to explore how energy reduction using our design scales with SRAM capacity. This is a difficult task since the currently public-available state-of-the-art simulator such as GPGPU-Sim does not model newer GPU architectures like Pascal. Therefore, we can only scale our configurations of newer architectures with various SRAM sizes based on GPGPU-Sim's model, and estimate the total energy reduction on the BVF units only. Table 4 lists the SRAM capacities of three generations of NVIDIA GPUs. And Figure 22 shows the average relative energy reduction for 28nm and 40nm across 58 applications under different SRAM configurations. We can observe that our BVF design on GPU achieves consistently high energy reduction on on-chip SRAM structures (e.g., around 52% on 40nm and 48% on 28nm) under different SRAM capacities, suggesting that our proposed design can be generally applied across generations of GPUs.

(D) Compared with 6T SRAM.

Figure 23 shows the normalized average energy reduction of BVF with respect to 6T and 8T SRAM across the evaluated applications. We take the static power penalty from the area overhead of 8T over 6T into consideration. The results show that under 1.2V where 6T design can operate, our 8T BVF-based design reduces the total chip energy consumption by an average of 31.6% and 32.7% for the entire GPU chip over the conventional 6T SRAM, under 28nm and 40nm. It also matches the discussion in Section 2.2 that significant energy benefit can be achieved from the deep DVFS (e.g., 0.6V) enabled by the 8T design.

6.3 Design Overhead

We measure the area and power for XNOR gate using the commercial 28nm and 40nm PDK. In total, our coders introduce 133,920 XNOR gates for the entire GPU chip. They account for 46.5 mW and 60.5 mW dynamic power, 18.7 μW and 24.2 μW static power for 28nm and 40nm process respectively, which correspond to approximately

Table 4: Different SRAM capacity configurations of modern GPUs.

GPU	Arch.	CC.	SM	REG/SM	IC/SM	L1D/SM	L2	L1T/SM	L1C/SM	SHM/SM
GTX-480	Fermi	2.0	15	128KB	2KB	16KB	768KB	12KB	8KB	48KB
Tesla-P100	Pascal	3.7	56	256KB	16KB	16KB	1536KB	48KB	8KB	112KB
Tesla-K80	Kepler	6.0	13	512KB	16KB	48KB	4096KB	48KB	10KB	64KB

0.037% and 0.000058% dynamic and static on-chip power for GPU with 40nm process. Note that this is a very conservative estimation since these gates are definitely not active every cycle; the memory access instructions are only a small fraction of all the executed instructions and only a tiny portions of them have to access the off-core NoC/L2 and off-chip DRAM. In addition, we assume all the units have independent read and write port based on 8T SRAM design. Since all three types of coders we propose are invertible, a R/W port can benefit from sharing the single coder for encoding and decoding. Putting everything together, the overall area overhead is $0.207mm^2$ and $0.294mm^2$ under 28nm and 40nm, including the wiring area overhead as well. This corresponds to 0.056% of the total die area of the baseline GPU. Compared to the huge energy benefit our design brings, such design overhead is negligible.

For the delay, although we add the encoder and decoder at the register port, the basic delay for one coder is just a logic XNOR gate which is negligible. This is because GPU pipeline uses operand collectors [73] to buffer the operands for the scheduled warps before being issued to execute stage. So delay for one or two logic gates will not impact the critical path. Besides, register reads and writes perform in the pipeline stage of decoding and writeback on GPU. Thus execution involving our coders does not bring evident overhead to the overall performance.

Regarding changing from pull-up PMOS in the conventional 8T SRAM to pull-down NMOS in the precharging circuit of BVF 8T SRAM (Section 3.1), no extra area is introduced. This is because under the same CMOS technology, NMOS normally delivers 1.5X to 2X more current than PMOS with the same sizing. Therefore, to deliver the same current, the sizing of NMOS is smaller than PMOS. For the two adopted technology nodes (28nm and 40nm), the NMOS pull-down transistor sizing is 2X smaller than the PMOS pull-up transistor, resulting in no extra area overhead.

7 DISCUSSION

We have demonstrated the significant power benefits of using BVF design on GPU platform. In this section, we intend to generalize the BVF techniques to other prevailing memory technologies (e.g., 6T SRAM and gain-cell based eDRAM) and discuss the correlation of BVF with other GPU power-reduction techniques, e.g., data compression.

7.1 BVF for 6T SRAM

Traditional 6T SRAM cell is symmetric, showing no significant difference when accessing 0 and 1. This is also potentially the reason why previous memory simulators do not distinguish the 0/1 energy difference. As already discussed in Section 3, the novel 8T SRAM design enforces write BVF by precharging $\overline{WBL}$ to ground for write. As 6T SRAM uses similar write-bitline structure (see Figure 4-(A)), it is intuitive to directly adopt the same modification to 6T SRAM (i.e., precharged $\overline{BL}$ to ground). In fact, such modification works equally well on 6T SRAM, allowing 6T SRAM to favor writing bit-1. At the same time, reading bit-1 is also preferred, provided the reading action does not flip the precharged bitline status when 1 is stored on the left storage node.

However, this precharge scheme could potentially lead to read failure with the increase of cells per bitline, because the 6T SRAM

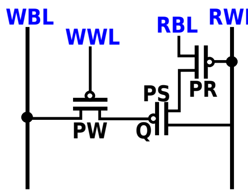

Figure 24: Gain-cell based eDRAM. PMOS transistor is adopted rather than NMOS is due to its less gate tunneling leakage current, which can extend data retention time [74].

read is naturally a "*destructive*" read; increased bitcells per bitline may lead to a larger bitline parasitic capacitance. Consequently, the big amount of charges on the bitlines (i.e., *BL* to 1 and $\overline{BL}$ to 0) may cause read failure when the cell stores 0 (on left storage node). We simulate the above 6T BVF SRAM design and evaluate the scenario of read 0 using 28nm process (see Section 5). The results show that when the cells per bitline exceeds 16, reading 0 may flip the data content (i.e., the 1/0 bitline causes 0/1 storage to flip upon successful read). As a comparison, for 8T SRAM, the decoupled 2T read-port makes the read naturally non-destructive, therefore avoiding this reliability concern in the 6T BVF design. A potential solution for 6T BVF to work is to revise the differential read operation to a single-end access. However, this will increase access failure and generally is not preferred in commercial IPs. Since no further silicon measurement is yet performed to validate such speculation, we will leave this discussion for the future work.

7.2 BVF for eDRAM

Gain-cell based embedded-DRAM (eDRAM) designs are proposed as a promising replacement for on-chip SRAM [75–78], due to its high density and low leakage features [79, 74]. An example of 3T gain-cell eDRAM is shown in Figure 24, where the data is stored on the internal node *Q*. Readers can refer to [74] for more detailed description of the operation principle of gain-cell eDRAM.

In fact, the design shown in Figure 24 favors bit-value 1 for both access and refresh. For read access with single-end read port, the *RBL* is precharged to V_{dd} and *RWL* (active low) is asserted, while all PMOS eDRAM cell favors 1 on the storage node *Q* as *RBL* will remain V_{dd} as *PS* is off. On the contrary, when storage node *Q* holds 0, the *RBL* would be discharged via *PR* and *PS*. This read operation is similar to that of the BVF 8T SRAM discussed in previous sections. For write access, the *WBL* is also precharged to V_{dd}, which leads to efficient write 1 (*WBL* remains V_{dd}) operation than writing 0 (*WBL* discharged to ground). The write operation is essentially a single-end write, which is different from the differential write operation of BVF 8T SRAM, where a miss causes doubled power consumption. Since both read and write favor 1, the refresh operation (dummy read and write-back operation before data is lost) also favors 1.

In summary, our discussion indicates that both 6T SRAM and gain-cell based eDRAM exhibit BVF feature, as proposed for the BVF 8T SRAM. This could fundamentally provide another useful knob for reducing the on-chip SRAM power orthogonal to DVFS, where the savings are from reduced switching activity.

7.3 BVF and Other Power Reduction Schemes

Due to GPU's large power envelope, multiple power reduction techniques have been proposed, e.g., [80, 81, 12, 22, 82] for register files, [21, 83] for caches, [57, 82] for execution units, [84, 35] for NoC, and [85] for DRAM. Since our proposal enables BVF-based SRAM cells and works at the bit-level of the data stream, it is orthogonal to most of these architecture-level techniques. Meanwhile, all these techniques ultimately aim to reduce power for a single GPU on-chip component, which is often limited by the actual power proportion of the component (e.g, as the largest SRAM structure on GPU, register file consumes around 15% of the total chip power, out of which previous register power-reduction techniques save about 20% to 30%). Moreover, jointly applying these individual techniques to form a unified solution may be challenging since they can conflict with each other. Our technique, however, is a global and systematic approach that benefits all on-chip SRAM and NoC units.

Among these techniques, data compression can potentially interfere with BVF [58, 86, 22, 59, 35, 55, 82], since it may change the original data pattern. We discuss data compression in two aspects: *bus compression* and *memory compression*. For bus compression, to the best of our knowledge, the existing works [58, 86, 59, 55] focus on conserving off-chip bandwidth. On the contrary, our BVF technique focuses on on-chip SRAM and it is transparent to all the off-chip units. Recall Figure 7, our simple coder design will not impact the off-chip data pattern. Thus, BVF design is compatible with these off-chip bus compression schemes. For memory compression, existing works mostly focus on GPU register files [22, 82]. We can apply BVF-design prior to memory compression. These compression schemes will not be affected by the NV and ISA coder designs since they work at bit-level and instruction stream, respectively. For the VS coder, its power-reduction may slightly decrease but without incurring any correctness issues. This is because VS-encoders mostly do not break the value-similarity pattern that these register compression schemes rely on, since all the non-pivot-lanes still hold similar values after VS-coding. How to design a highly-effective BVF-based compression scheme is indeed an interesting future research topic. Finally, we want to emphasize that most lossless data-compression techniques are expensive (in terms of delay, area and energy), e.g., the subtractor arrays and control logic for BDI [22], the metadata cache and off-line profiling for CABA [59]. However, our design only requires a single gate for each coder without any extra control logic or bit-lines, incurring far less overhead.

8 RELATED WORK

Since we have discussed the related work regarding 6T and 8T SRAM cells and their features in Section 2 and 3, we focus on discussing energy reduction techniques for GPU.

Effectively reducing power & energy consumption has become one of the first-order concerns in GPU architecture design [87, 30]. There have been a large body of works proposed to optimize GPU power consumption and they primarily focus on optimizing power for one or several on-chip hardware units, covering compute units, registers, SRAM integration, DVFS, process variation and NoC. For compute units, Gilani et al. [57] proposed four optimization techniques specially to reduce GPU compute units' power, including integer operation packing, representative scaler computation, compacting and descent-precision operation multi-issuing. For SRAM integration, Gebhart et al. [38] integrated cache, register and shared memory on GPUs as a unified SRAM memory, and dynamically partitioned it according to the preference of the applications for reducing chip-level power consumption. For DVFS and process variation, Thomas et al. [88] investigated the joint-effect of process variation and voltage noise on GPU, and clock-gating the vulnerable cores when voltage noise on the critical path is detected to avoid timing violation. Paul et al. [89] dynamically tuned hardware configurations (i.e., active cores, core frequency and memory bus frequency) to maintain the balance between computation power and memory access power, in hoping that power can be reduced with minimal performance impact. For GPU NoC, Pekhimenko et al. [35] observed that data compression on the GPU on-chip and off-chip buses might lead to increased bit toggles, thus proposed two compensate techniques. Finally, there have been several works targeting energy reduction on GPU register file due to its large capacity and on-chip voltage influence. For example, Yu et al. [75] integrated SRAM with eDRAM on-chip to reduce GPU register power. Gebhart et al. [21] plunged a small fast cache ahead of the GPU registers, and modified the warp scheduler accordingly to make better utilization of the register cache and reduce energy. Abdel-Majeed et al. [80] adjusted the GPU warp scheduler to enable profitable opportunities for power gating on register file when the execution units are idle. Lee et al. [22] proposed a register compression technique to save GPU register power based on intra-warp operand value similarity. Wong et al. [40] used a single warp lane as the representative thread on behalf of the entire warp to compute and access register, while approximated other lanes according to the inter-lane value similarity. Tan et.al. [12] applied register dead-time to patch vulnerable register accesses due to lowered voltage and process variation to conserve energy for the entire GPU chip.

Majority of these work, however, focused on a single on-chip hardware unit and proposed novel architectural design to reduce its power. The source of their power savings mainly comes from their architecture designs. In comparison, our proposed BVF spaces can cover a large number of on-chip hardware units (e.g., major SRAM structures and NoC) as long as they can be partitioned in a BVF space. More importantly, our technique adopts a novel circuit-architecture co-design approach: the modified circuit offers the energy saving asymmetric property, while the architectural design reinforces such effect and fully exploits this opportunity. Our optimization strategy is also fairly simple — maximizing bit 1 in data value and data transmission.

9 CONCLUSION

In this paper, we proposed BVF, a fundamental property for 8T SRAM and other types of memory, which implies that the energy consumption for accessing and storing bit 0 and 1 is dramatically different. To amplify and harvest energy gain from such difference, we proposed three architectural coder designs to maximize the occurence of bit-1s in the GPU data and instruction streams, known as BVF optimization. Additionally, the new coders also lead to significant bit toggle reduction in the NoC and buses on GPU chip. All these together bring substantial on-chip power reduction for modern GPUs. Although our circuit-architecture co-design is proposed and evaluated only for GPU in this paper, it can be easily applied to other SRAM integrated processors with moderate modifications. We believe the general concept of BVF can potentially become another tuning knob like DVFS for on-chip SRAM power reduction.

ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers for their constructive comments and suggestions for improving this work. This research is supported by the U.S. DOE Office of Science, Office of Advanced Scientific Computing Research, under award 66150: "CENATE - Center for Advanced Architecture Evaluation". The Pacific Northwest National Laboratory is operated by Battelle for the U.S. Department of Energy under contract DE-AC05-76RL01830.

REFERENCES

[1] David A Patterson. *Computer Architecture: A Quantitative Approach*. Elsevier, 2011.

[2] William J. Dally. Moving the Needle, Computer Architecture Research in Academe and Industry. In *37th Annual International Symposium on Computer Architecture*, ISCA. ACM, 2010.

[3] S Keckler. Life after Dennard and how I learned to love the picojoule. *Keynote at MICRO*, 2011.

[4] Keren Bergman, Shekhar Borkar, Dan Campbell, William Carlson, William Dally, Monty Denneau, Paul Franzon, William Harrod, Kerry Hill, Jon Hiller, et al. Exascale computing study: Technology challenges in achieving exascale systems. *Defense Advanced Research Projects Agency Information Processing Techniques Office (DARPA IPTO), Tech. Rep*, 15, 2008.

[5] Peter M Kogge and Timothy J Dysart. Using the TOP500 to trace and project technology and architecture trends. In *International Conference for High Performance Computing, Networking, Storage and Analysis*, SC. ACM, 2011.

[6] Andy White. Exascale challenges: Applications, technologies, and co-design. In *From Petascale to Exascale: R&D Challenges for HPC Simulation Environments ASC Exascale Workshop*, 2011.

[7] Naveen Muralimanohar, Rajeev Balasubramonian, and Norman P Jouppi. CACTI 6.0: A tool to model large caches. *HP Laboratories*, pages 22–31, 2009.

[8] Chris Wilkerson, Hongliang Gao, Alaa R Alameldeen, Zeshan Chishti, Muhammad Khellah, and Shih-Lien Lu. Trading off cache capacity for reliability to enable low voltage operation. In *35th International Symposium on Computer Architecture*, ISCA. IEEE, 2008.

[9] Anys Bacha and Radu Teodorescu. Dynamic reduction of voltage margins by leveraging on-chip ECC in Itanium II processors. In *40th International Symposium on Computer Architecture*, ISCA. ACM, 2013.

[10] Anys Bacha and Radu Teodorescu. Using ECC feedback to guide voltage speculation in low-voltage processors. In *47th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO. IEEE, 2014.

[11] Juan-Antonio Carballo, Wei-Ting Jonas Chan, Paolo A Gargini, Andrew B Kahng, and Siddhartha Nath. ITRS 2.0: Toward a re-framing of the Semiconductor Technology Roadmap. In *IEEE 32nd International Conference on Computer Design*, ICCD. IEEE, 2014.

[12] Jingweijia Tan, Shuaiwen Leon Song, Kaige Yan, Xin Fu, Andres Marquez, and Darren Kerbyson. Combating the reliability challenge of GPU register file at low supply voltage. In *International Conference on Parallel Architecture and Compilation Techniques*, PACT. IEEE, 2016.

[13] David J Palframan, Nam Sung Kim, and Mikko H Lipasti. iPatch: Intelligent fault patching to improve energy efficiency. In *21st International Symposium on High Performance Computer Architecture*, HPCA. IEEE, 2015.

[14] Leland Chang, Robert K Montoye, Yutaka Nakamura, Kevin A Batson, Richard J Eickemeyer, Robert H Dennard, Wilfried Haensch, and Damir Jamsek. An 8T-SRAM for variability tolerance and low-voltage operation in high-performance caches. *IEEE Journal of Solid-State Circuits*, 2008.

[15] Alireza Shafaei, Yanzhi Wang, Xue Lin, and Massoud Pedram. Fincacti: Architectural analysis and modeling of caches with deeply-scaled finfet devices. In *IEEE Computer Society Annual Symposium on VLSI*. IEEE, 2014.

[16] Azeez J Bhavnagarwala, Xinghai Tang, and James D Meindl. The impact of intrinsic device fluctuations on CMOS SRAM cell stability. *IEEE Journal of Solid-State Circuits*, 2001.

[17] Naoki Tega, Hiroshi Miki, Masanao Yamaoka, Hitoshi Kume, Toshiyuki Mine, Takeshi Ishida, Yuki Mori, Renichi Yamada, and Kazuyoshi Torii. Impact of threshold voltage fluctuation due to random telegraph noise on scaled-down SRAM. In *IEEE International Reliability Physics Symposium*. IEEE, 2008.

[18] J. Yang, H.K. Lin, J. Shen, Y. Li, and H. Chen. Eight transistor (8T) write assist static random access memory (SRAM) cell, 2015. US Patent 9,183,922.

[19] J.J. Wuu, D.R. Weiss, K.E. WILCOX, A.W. SCHAEFER, and K.V. UNDERHILL. Alternating wordline connection in 8t cells for improving resiliency to multi-bit ser upsets, 2013. WO Patent App. PCT/US2012/050,542.

[20] Leland Chang, David M Fried, Jack Hergenrother, Jeffrey W Sleight, Robert H Dennard, Robert K Montoye, Lidija Sekaric, Sharee J McNab, Anna W Topol, Charlotte D Adams, et al. Stable SRAM cell design for the 32 nm node and beyond. In *Symposium on VLSI Technology Digest of Technical Papers*. IEEE, 2005.

[21] Mark Gebhart, Daniel R Johnson, David Tarjan, Stephen W Keckler, William J Dally, Erik Lindholm, and Kevin Skadron. Energy-efficient mechanisms for managing thread context in throughput processors. In *38th International Symposium on Computer Architecture*, ISCA. ACM, 2011.

[22] Sangpil Lee, Keunsoo Kim, Gunjae Koo, Hyeran Jeon, Won Woo Ro, and Murali Annavaram. Warped-compression: enabling power efficient GPUs through register compression. In *42nd International Symposium on Computer Architecture*, ISCA. ACM, 2015.

[23] Zhe Zhang, Michael A Turi, and José G Delgado-Frias. SRAM leakage in CMOS, FinFET and CNTFET technologies: Leakage in 8T and 6T SRAM cells. In *Proceedings of the Great Lakes Symposium on VLSI*. ACM, 2012.

[24] Haruki Mori, T Nakagawa, Y Kitahara, Y Kawamoto, K Takagi, S Yoshimoto, S Izumi, K Nii, H Kawaguchi, and M Yoshimoto. A 298-fJ/writecycle 650-fJ/readcycle 8T three-port SRAM in 28-nm FD-SOI process technology for image processor. In *IEEE Custom Integrated Circuits Conference*, CICC. IEEE, 2015.

[25] Jonathan Dama and Andrew Lines. Pseudo dual-port SRAM and a shared memory switch using multiple memory banks and a sideband memory, 2013. US Patent 8,370,557.

[26] Sheng Li, Jung Ho Ahn, Richard D Strong, Jay B Brockman, Dean M Tullsen, and Norman P Jouppi. McPAT: an integrated power, area, and timing modeling framework for multicore and manycore architectures. In *42nd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO. ACM, 2009.

[27] Mahmut E Sinangil and Anantha P Chandrakasan. Application-Specific SRAM Design Using Output Prediction to Reduce Bit-Line Switching Activity and Statistically Gated Sense Amplifiers for Up to 1.9× Lower Energy/Access. *IEEE Journal of Solid-State Circuits*, 2014.

[28] Shyamkumar Thoziyoor, Naveen Muralimanohar, Jung Ho Ahn, and Norman P Jouppi. CACTI 5.1. Technical report, HPL-2008-20, HP Labs, 2008.

[29] Subhasis Bhattacharjee and Dhiraj K Pradhan. LPRAM: a novel low-power high-performance RAM design with testability and scalability. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2004.

[30] Jingwen Leng, Tayler Hetherington, Ahmed ElTantawy, Syed Gilani, Nam Sung Kim, Tor M Aamodt, and Vijay Janapa Reddi. GPUWattch: enabling energy optimizations in GPGPUs. In *40th Annual International Symposium on Computer Architecture*, ISCA. ACM, 2013.

[31] Ali Bakhoda, John Kim, and Tor M Aamodt. Throughput-effective on-chip networks for manycore accelerators. In *43rd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO. IEEE, 2010.

[32] Vignesh Adhinarayanan, Indrani Paul, Joseph L. Greathouse, Wei Huang, Ashutosh Pattnaik, and Wu-chun Feng. Measuring and Modeling On-Chip Interconnect Power on Real Hardware. In *International Symposium on Workload Characterization*, IISWC. IEEE, 2016.

[33] Bradford M Beckmann and David A Wood. TLC: Transmission line caches. In *36th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO. IEEE, 2003.

[34] Mahdi Nazm Bojnordi and Engin Ipek. DESC: energy-efficient data exchange using synchronized counters. In *46th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO. ACM, 2013.

[35] Gennady Pekhimenko, Evgeny Bolotin, Nandita Vijaykumar, Onur Mutlu, Todd C Mowry, and Stephen W Keckler. A case for toggle-aware compression for GPU systems. In *International Symposium on High Performance Computer Architecture*, HPCA. IEEE, 2016.

[36] Mircea R Stan and Wayne P Burleson. Bus-invert coding for low-power I/O. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 1995.

[37] Ji Gu and Hui Guo. A segmental bus-invert coding method for instruction memory data bus power efficiency. In *IEEE International Symposium on Circuits and Systems*, ISCAS. IEEE, 2009.

[38] Mark Gebhart, Stephen W Keckler, Brucek Khailany, Ronny Krashinsky, and William J Dally. Unifying primary cache, scratch, and register file memories in a throughput processor. In *45th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO. IEEE, 2012.

[39] Ji Kim, Christopher Torng, Shreesha Srinath, Derek Lockhart, and Christopher Batten. Microarchitectural mechanisms to exploit value structure in SIMT architectures. In *40th Annual International Symposium on Computer Architecture*, ISCA. ACM, 2013.

[40] Daniel Wong, Nam Sung Kim, and Murali Annavaram. Approximating warps with intra-warp operand value similarity. In *International Symposium on High Performance Computer Architecture*, HPCA. IEEE, 2016.

[41] Gennady Pekhimenko, Vivek Seshadri, Onur Mutlu, Phillip B Gibbons, Michael A Kozuch, and Todd C Mowry. Base-delta-immediate compression: practical data compression for on-chip caches. In *21st International Conference on Parallel Architectures and Compilation Techniques*, PACT. ACM, 2012.

[42] NVIDIA. CUDA SDK Code Samples, 2015.

[43] William Kahan. IEEE standard 754 for binary floating-point arithmetic. *Lecture Notes on the Status of IEEE*, 754(94720-1776):11, 1996.

[44] Marcel Gort and Jason H Anderson. Range and bitmask analysis for hardware optimization in high-level synthesis. In *18th Asia and South Pacific Design Automation Conference*, ASP-DAC. IEEE, 2013.

[45] Victor Hugo Sperle Campos, Raphael Ernani Rodrigues, Igor Rafael de Assis Costa, and Fernando Magno Quintao Pereira. Speed and precision in range analysis. In *Brazilian Symposium on Programming Languages*, pages 42–56. Springer, 2012.

[46] Alaa R Alameldeen and David A Wood. Frequent pattern compression: A significance-based compression scheme for L2 caches. *Technical Report*, 2004.

[47] Daniel Citron and Larry Rudolph. Creating a wider bus using caching techniques. In *First IEEE Symposium on High-Performance Computer Architecture*, HPCA. IEEE, 1995.

[48] Yuho Jin, Ki Hwan Yum, and Eun Jung Kim. Adaptive data compression for high-performance low-power on-chip networks. In *41st Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO. IEEE, 2008.

[49] NVIDIA. NVIDIA Tesla P100: The Most Advanced Datacenter Accelerator Ever Built Featuring Pascal GP100 the World's Fastest GPU. http://images.nvidia.com/content/pdf/tesla/whitepaper/pascal-architecture-whitepaper.pdf, 2016.

[50] NVIDIA. Parallel Thread Execution ISA. http://docs.nvidia.com/cuda/parallel-thread-execution.

[51] Jun Yang, Youtao Zhang, and Rajiv Gupta. Frequent value compression in data caches. In *33rd Annual ACM/IEEE International Symposium on Microarchitecture*, MICRO. ACM, 2000.

[52] Saisanthosh Balakrishnan and Gurindar S Sohi. Exploiting value locality in physical register files. In *36th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO. IEEE, 2003.

[53] Magnus Ekman and Per Stenstrom. A robust main-memory compression scheme. In *32nd Annual International Symposium on Computer Architecture*, ISCA. IEEE, 2005.

[54] Mafijul Md Islam and Per Stenstrom. Zero-value caches: Cancelling loads that return zero. In *18th International Conference on Parallel Architectures and Compilation Techniques*, PACT. IEEE, 2009.

[55] Minsoo Rhu, Mike O'Connor, Niladrish Chatterjee, Jeff Pool, and Stephen W Keckler. Compressing DMA Engine: Leveraging Activation Sparsity for Training Deep Neural Networks. *arXiv preprint arXiv:1705.01626*, 2017.
[56] Ping Xiang, Yi Yang, Mike Mantor, Norm Rubin, Lisa R Hsu, and Huiyang Zhou. Exploiting uniform vector instructions for GPGPU performance, energy efficiency, and opportunistic reliability enhancement. In *27th International ACM conference on International Conference on Supercomputing*, ICS. ACM, 2013.
[57] Syed Zohaib Gilani, Nam Sung Kim, and Michael J Schulte. Power-efficient computing for compute-intensive GPGPU applications. In *19th International Symposium on High Performance Computer Architecture*, HPCA. IEEE, 2013.
[58] Vijay Sathish, Michael J Schulte, and Nam Sung Kim. Lossless and lossy memory I/O link compression for improving performance of GPGPU workloads. In *21st International Conference on Parallel Architectures and Compilation Techniques*, PACT. ACM, 2012.
[59] Nandita Vijaykumar, Gennady Pekhimenko, Adwait Jog, Abhishek Bhowmick, Rachata Ausavarungnirun, Chita Das, Mahmut Kandemir, Todd C Mowry, and Onur Mutlu. A case for core-assisted bottleneck acceleration in GPUs: enabling flexible data compression with assist warps. In *42nd Annual International Symposium on Computer Architecture*, ISCA. ACM, 2015.
[60] Richard W Hamming. Error detecting and error correcting codes. *Bell System Technical Journal*, 1950.
[61] Ang Li, Shuaiwen Leon Song, Weifeng Liu, Xu Liu, Akash Kumar, and Henk Corporaal. Locality-aware cta clustering for modern gpus. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS, pages 297–311. ACM, 2017.
[62] Ang Li, Gert-Jan van den Braak, Akash Kumar, and Henk Corporaal. Adaptive and transparent cache bypassing for gpus. In *High Performance Computing, Networking, Storage and Analysis, 2015 SC-International Conference for*, SC, pages 1–12. IEEE, 2015.
[63] Cadence. Virtuoso Analog Design Environment.
[64] Cadence. Virtuoso Multi-Mode Simulation with Spectre Platform.
[65] Ali Bakhoda, George L Yuan, Wilson WL Fung, Henry Wong, and Tor M Aamodt. Analyzing CUDA workloads using a detailed GPU simulator. In *IEEE International Symposium on Performance Analysis of Systems and Software*, ISPASS. IEEE, 2009.
[66] Shuai Che, Michael Boyer, Jiayuan Meng, David Tarjan, Jeremy W Sheaffer, Sang-Ha Lee, and Kevin Skadron. Rodinia: A benchmark suite for heterogeneous computing. In *IEEE International Symposium on Workload Characterization*, IISWC. IEEE, 2009.
[67] John A Stratton, Christopher Rodrigues, I-Jui Sung, Nady Obeid, Li-Wen Chang, Nasser Anssari, Geng Daniel Liu, and Wen-mei W Hwu. Parboil: A revised benchmark suite for scientific and commercial throughput computing. *Center for Reliable and High-Performance Computing*, 127, 2012.
[68] Anthony Danalis, Gabriel Marin, Collin McCurdy, Jeremy S Meredith, Philip C Roth, Kyle Spafford, Vinod Tipparaju, and Jeffrey S Vetter. The scalable heterogeneous computing (SHOC) benchmark suite. In *3rd Workshop on General-Purpose Computation on Graphics Processing Units*, GPGPU. ACM, 2010.
[69] Milind Kulkarni, Martin Burtscher, Calin Casçaval, and Keshav Pingali. Lonestar: A suite of parallel irregular programs. In *IEEE International Symposium on Performance Analysis of Systems and Software*, ISPASS. IEEE, 2009.
[70] Scott Grauer-Gray, Lifan Xu, Robert Searles, Sudhee Ayalasomayajula, and John Cavazos. Auto-tuning a high-level language targeted to GPU codes. In *Innovative Parallel Computing*, InPar. IEEE, 2012.
[71] Stanley Wolf. Excerpt from: Silicon Processing for the VLSI Era-vol. 4. 2004.
[72] Veynu Narasiman, Michael Shebanow, Chang Joo Lee, Rustam Miftakhutdinov, Onur Mutlu, and Yale N Patt. Improving GPU performance via large warps and two-level warp scheduling. In *44th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO. ACM, 2011.
[73] Craig M Wittenbrink, Emmett Kilgariff, and Arjun Prabhu. Fermi GF100 GPU architecture. *IEEE Micro*, 31(2):50–59, 2011.
[74] Ki Chul Chun, Pulkit Jain, Jung Hwa Lee, and Chris H Kim. A sub-0.9 V logic-compatible embedded DRAM with boosted 3T gain cell, regulated bit-line write scheme and PVT-tracking read reference bias. In *Symposium on VLSI Circuits*. IEEE, 2009.
[75] S Yu Wing-kei, Ruirui Huang, Sarah Q Xu, Sung-En Wang, Edwin Kan, and G Edward Suh. SRAM-DRAM hybrid memory with applications to efficient register files in fine-grained multi-threading. In *38th Annual International Symposium on Computer Architecture*, ISCA. IEEE, 2013.
[76] Naifeng Jing, Yao Shen, Yao Lu, Shrikanth Ganapathy, Zhigang Mao, Minyi Guo, Ramon Canal, and Xiaoyao Liang. An energy-efficient and scalable eDRAM-based register file architecture for GPGPU. In *40th Annual International Symposium on Computer Architecture*, ISCA. ACM, 2013.
[77] Dinesh Somasekhar, Yibin Ye, Paolo Aseron, Shih-Lien Lu, Muhammad Khellah, Jason Howard, Greg Ruhl, Tanay Karnik, Shekhar Y Borkar, Vivek De, et al. 2GHz 2Mb 2T gain-cell memory macro with 128GB/s bandwidth in a 65nm logic process. In *IEEE International Solid-State Circuits Conference-Digest of Technical Papers*. IEEE, 2008.
[78] Ang Li, Weifeng Liu, Mads RB Kristensen, Brian Vinter, Hao Wang, Kaixi Hou, Andres Marquez, and Shuaiwen Leon Song. Exploring and analyzing the real impact of modern on-package memory on hpc scientific kernels. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC, 2017.
[79] W Luk, Jin Cai, R Dennard, M Immediato, and S Kosonocky. A 3-transistor DRAM cell with gated diode for enhanced speed and retention time. In *2006 Symposium on VLSI Circuits Digest of Technical Papers*. IEEE, 2006.
[80] Mohammad Abdel-Majeed and Murali Annavaram. Warped register file: A power efficient register file for GPGPUs. In *19th International Symposium on High Performance Computer Architecture*, HPCA. IEEE, 2013.
[81] Mohammad Abdel-Majeed, Alireza Shafaei, Hyeran Jeon, Massoud Pedram, and Murali Annavaram. Pilot Register File: Energy Efficient Partitioned Register File for GPUs. In *23rd IEEE International Symposium on High Performance Computer Architecture*, HPCA. IEEE, 2017.
[82] Zhenhong Liu, Syed Gilani, Murali Annavaram, and Nam Sung Kim. G-Scalar: Cost-Effective Generalized Scalar Execution Architecture for Power-Efficient GPUs. In *23rd IEEE International Symposium on High Performance Computer Architecture*, HPCA. IEEE, 2017.
[83] Minsoo Rhu, Michael Sullivan, Jingwen Leng, and Mattan Erez. A locality-aware memory hierarchy for energy-efficient GPU architectures. In *46th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO. ACM, 2013.
[84] Amir Kavyan Ziabari, José L Abellán, Yenai Ma, Ajay Joshi, and David Kaeli. Asymmetric NoC Architectures for GPU Systems. In *9th International Symposium on Networks-on-Chip*, NOCS. ACM, 2015.
[85] Hoseok Seol, Wongyu Shin, Jaemin Jang, Jungwhan Choi, Jinwoong Suh, and Lee-Sup Kim. Energy efficient data encoding in DRAM channels exploiting data value similarity. In *43rd ACM/IEEE Annual International Symposium on Computer Architecture*, ISCA. IEEE, 2016.
[86] NVIDIA. Whitepaper: NVIDIA GeForce GTX 980, 2014.
[87] Sunpyo Hong and Hyesoon Kim. An integrated GPU power and performance model. In *37th Annual International Symposium on Computer Architecture*, ISCA. ACM, 2010.
[88] Renji Thomas, Kristin Barber, Naser Sedaghati, Li Zhou, and Radu Teodorescu. Core Tunneling: Variation-aware voltage noise mitigation in GPUs. In *22nd International Symposium on High Performance Computer Architecture*, HPCA. IEEE, 2016.
[89] Indrani Paul, Wei Huang, Manish Arora, and Sudhakar Yalamanchili. Harmonia: Balancing compute and memory power in high-performance GPUs. In *42nd Annual International Symposium on Computer Architecture*, ISCA. ACM, 2015.