Design and Analysis of Soft-Error Resilience Mechanisms for GPU Register File

Sparsh Mittal*, Haonan Wang†, Adwait Jog†, Jeffrey S. Vetter§
*IIT Hyderabad, India, †College of William and Mary, USA, §Oak Ridge National Laboratory, USA
Email: sparsh@iith.ac.in, {hwang07,adwait}@cs.wm.edu, vetter@computer.org

***Abstract*—Modern graphics processing units (GPUs) are using increasingly larger register file (RF) which occupies a large fraction of GPU core area and is very frequently accessed. This makes RF vulnerable to soft-errors (SE). In this paper, we present two techniques for improving SE resilience of GPU RF. First, we propose compressing the RF values for reducing the number of vulnerable bits. We leverage value similarity and the presence of narrow-width values to perform compression at warp or thread-level, respectively. Second, we propose selective hardening to design a portion of register entry with SE immune circuits. By collectively using these techniques, higher resilience can be provided with lower overhead. Without hardening, our warp and thread-level compression techniques bring 47.0% and 40.8% reduction in SE vulnerability, respectively.**

I. Introduction

Recent trends in processor design have aggravated the occurrence of faults in modern processors.. Ongoing voltage scaling accompanied with feature size scaling reduces the critical charge required to flip a bit. This allows even lower-energy particles to cause soft errors. Due to these reasons, soft-error rate at 16nm is expected to be more than 100 times that at 180nm [1]. As GPUs become mainstream computing systems, improving soft-error reliability of GPUs has become extremely important.

Out of different GPU components, RF is particularly vulnerable to soft-errors due to its large size and performance-optimized design. For example, a recent study performed on Titan supercomputer showed that over a period of 2 years, out of five GPU components (L1/L2 cache, texture memory, device memory and RF), 86% and 14% of double bit errors occurred in device memory and RF, respectively [2]. Further, as shown in Table I, the total RF size on GPUs is much larger than that of L1/L2 caches and has been increasing in recent GPU generations (SM = streaming multiprocessor). Similarly, AMD Radeon HD 7970 GPU has 16 KB L1 cache in each of 32 computing units and a total of 8.25 MB RF and 768 KB shared L2 cache [3]. By comparison, CPUs possess tiny RF and much larger caches, e.g., Intel's 32 nm Itanium 9560 processor has 22 KB integer RF and 20 KB floating point RF and 32 MB L3 cache [4]. Clearly, due to performance-criticality and vulnerability of GPU RF, along with its fundamental differences with CPU RF, novel mechanisms are required to improve its resilience.

Table I: Size of L1/L2 cache and RF on NVIDIA GPUs [5–7] (Sizes in KB, CC = compute capability, *maximum size of L1 cache, †size of unified L1/texture cache)

	Archi-tecture	CC	L1 size per SM	L2 size	RF size per SM	# of SMs	**Total RF size**
G80	Tesla	1.0	None	None	32	16	**512**
GT200	Tesla	1.3	None	None	64	30	**1920**
GF100	Fermi	2.0	48*	768	128	16	**2048**
GK110	Kepler	3.5	48*	1536	256	15	**3840**
GK210	Kepler	3.7	48*	1536	512	15	**7680**
GM204	Maxwell	5.2	48†	2048	256	16	**4096**
GP100	Pascal	6.0	48†	4096	256	56	**14336**

In this paper, we present two techniques to reduce SE vulnerability (SEV) of GPU RF. First, we propose compressing the register values which reduces the number of bits required for storing a value and thus, reduces the number of vulnerable bits. We do not use compression to store more data in registers and thus, we forgo the capacity advantage of compression in favor of reliability [8]. Second, we propose selective hardening, i.e., designing a portion of RF with radiation-hardened (i.e., SE immune) memory. Compression and hardening are dynamic and static techniques, respectively and they can be used individually or together. We propose performing compression at the level of each warp or each thread. For warp-level compression (WarpC), we use the insight that due to value similarity, many thread-registers store values which are identical, have low dynamic range or are zero. By exploiting this redundancy, thread-registers of a warp are compressed using base-delta immediate (BDI) compression [9]. However, we do not use WarpC for divergent warps due to its higher complexity and lower benefits for them (§III-A).

Our thread-level compression technique (ThreadC) compresses each thread-register value individually. For a 4B thread-register, it determines the effective width (K) as the smallest among four possible values, viz. 0B, 1B, 2B and 4B (uncompressed). Only K bytes are read from thread-registers which reduces the number of vulnerable bits.

ThreadC can be applied to individual active threads of divergent instructions and thus, it is especially useful for applications with many divergent warps. However, ThreadC does not exploit value similarity and hence, for non-divergent applications, it provides smaller benefit than WarpC. Thus, our work reveals the importance of accounting

Sparsh contributed to this paper while working at ORNL. Sparsh and Jeffrey acknowledge support from U.S. DoE, Office of Science, Advanced Scientific Computing Research. Haonan and Adwait acknowledge start-up grant from College of William and Mary. We use *warp-register* (or *register*) to denote the architectural register referenced by a warp and *thread-register* to denote the register corresponding to each thread in the warp.

for GPU application characteristics for choosing the optimal compression approach. Our key contributions are:

1. By detailed characterization of many GPU applications, we show that SEV of GPU RF can be significantly reduced via compression. However, performing compression either at warp-level or thread-level alone may not give optimal SEV reduction for all applications.

2. We propose that higher reduction in SE vulnerability can be obtained by leveraging the warp divergence properties of GPU applications to decide between warp and thread-level compression techniques at runtime.

3. For further reduction in SEV, we propose selective hardening and show the potential of compression in reducing the requirement of hardening for achieving a desired level of protection. To the best of our knowledge, this is the first work that collectively considers compression at warp and thread-level, and selective hardening for protecting RF for a wide range of GPU applications with minimal overheads.

4. Micro-architectural simulations using a cycle-accurate GPGPU simulator and diverse range of workloads have shown that without hardening, WarpC and ThreadC bring 47.0% and 40.8% reduction in SEV respectively. With increasing amount of hardening, SEV can be further reduced.

II. Motivation and Background

A. Existing RF protection techniques

Recent commercial GPUs, such as Fermi, Kepler and Pascal use single-error-correction double-error-detection (SEC-DED) ECC for protecting RF, L1/L2 caches, shared memories and DRAM. However, due to high frequency of RF access, computing/checking ECC incurs large energy overhead. Further, multi-bit ECCs incur extremely high overhead and ECC also fails to exploit characteristics of GPU applications. Our technique protects RF by using compression to leverage redundancy present in GPU execution. Also, designing RF with hardened memory provides protection from multi-bit errors.

Palframan et al. [10] propose a precision-aware RF protection technique for RF which hardens the sign and exponent bits corresponding to single-precision FP values. To also provide protection to integer values, they store them in FP-like format. This, however, requires changing the data storage format. Tan et al. [11] use compiler to observe the lifetime (number of instructions between the write and last read) of register values and map long-lived and short-lived values to STT-RAM and SRAM, respectively in the SRAM-STTRAM hybrid RF. However, due to major differences between SRAM and STT-RAM technologies, designing such hybrid RF may incur significant design complexity.

B. Measuring soft-error vulnerability

We measure soft-error vulnerability of RF based on the idea of architectural vulnerability factor (AVF) [12] as it captures soft-error characteristics independent of raw error rate. AVF shows the fraction of time RF is vulnerable to soft errors. A register value which propagates to other components is considered critical and the time period during which an error in a register propagates to other components is called critical time. Specifically, a register value is critical between write-to-read and read-to-read and not between write-to-write and read-to-write.

Then, AVF of RF is the average critical time of all critical registers. Let M and R be number of critical registers and total registers, respectively, and bpR be number of bits in the register. CT_i shows critical time of a critical register and TT shows total execution time. Also, failure rates of all bits are assumed to be linearly uncorrelated. Then, $AVF = (\sum_{i=1}^{M} CT_i)/(TT \cdot R)$ and $SEV = R \cdot bpR \cdot AVF = (bpR \cdot \sum_{i=1}^{M} CT_i)/TT$

Thus, relative reduction in SEV due to a technique is shown as $SEV_{baseline}/SEV_{technique}$ and a higher value is better. It is clear that RF SEV can be reduced by (1) reducing number of bits in a critical register (2) designing RF with SE immune memory (e.g., non-volatile memory [11] or radiation-hardened memory [13, 14]) and (3) reducing critical time of register (e.g., by instruction rescheduling [1]). In this work, we exploit (1) and (2).

III. Design of Resilience Mechanisms

Figure 1 shows the GPU RF architecture assumed in this paper [15, 16]. Each RF entry is 128B wide and provides 32-bit operands to all 32 threads of a warp. Section IV-C discusses how our technique can work in other RF architectures. We follow CUDA terminology in this paper.

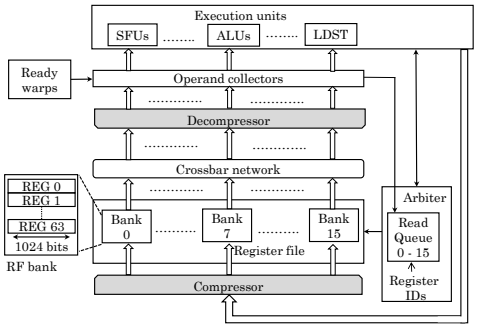

Figure 1: GPU RF architecture assumed in the paper

A. Warp-level compression

Key observation: Value similarity in GPU RF: In SIMT (single instruction multiple thread) execution model of GPUs, RF provides 32 times the number of source operands to the execution unit for any warp instruction, since there are 32 threads/warp. In this execution model, the thread registers of a warp may show significant value similarity due to multiple reasons. For several applications, the values operated by different threads may be similar (e.g., initialization by a constant, using a fixed number of iterations, etc.) or have low dynamic range. Also, many kernels assign data portions to different threads which access their data portions using

thread indices. Since thread indices differ by one, a warp register which accesses such data shows value similarity. Further, register values may be zero due to initialization, nature of program inputs (e.g., absence of any object in an image) and outputs (e.g., binary classification), operating on sparse matrices, etc.

Compression approach and algorithm: WarpC exploits value similarity to compress register values using BDI compression [9]. Register values with zero data are also compressed. This reduces the effective number of bits required for storing the data and hence, reduces the number of vulnerable bits. Compression and decompression are performed during RF write and read operations, respectively.

The BDI algorithm [9] attempts compression with base 2B, 4B and 8B. Denoting a compression state as BxDy, BDI uses B2D1, B4D1, B4D2, B8D0 (i.e. repeated values), B8D1, B8D2, B8D4 and AllZero states, where x and y are widths of base and delta in bytes, respectively. BDI compression was originally proposed for last level caches in CPU, where a large latency of (de)compression can be tolerated. However, since RF exists as the topmost level in memory hierarchy and is accessed very frequently, exploring multiple compression states at RF level can lead to large performance overhead. Also, since each thread-register is 4B, the value locality is best exploited on using a base size of 4B. For these reasons, we use three compression states: AllZero, B4D0 and B4D1. Thus, a register can be in either these three or the uncompressed state and only 2 bits are required to store this information.

A previous RF compression technique used B4D0, B4D1, B4D2 states [17]. We choose AllZero state since a large fraction of RF values are compressed to this state (§V). For registers with AllZero state, actual read/write to RF are not performed since the actual value can be recovered based on compression encoding only and thus, their SEV is reduced completely. Also, we do not choose B4D2 state to reduce metadata overhead and because including it provides only small additional benefit (§V-C). We now discuss challenges in ensuring effective use of compression.

1. Accounting for Criticality of Base: During decompression, the original data is obtained by adding the delta values to the base. Due to this, any error in the base during compressed state can spread to all the thread-registers and thus, even a single-bit error in base can manifest as a multi-bit error. Clearly, due to compression, the base value becomes crucial in terms of reliability and hence, naively applying compression may not achieve a right tradeoff between compression ratio and reliability improvement.

To address this, WarpC stores two copies of the base. This leads to a slight increase in the compressed width, but provides higher protection to the base. In this work, we only assume single-bit error model [18] and leave addressing multi-bit error to future work. We assume that any single-bit error can be detected by using a parity bit at byte granularity. For base value, correction can also be performed by leveraging two copies. For this, during decompression, an error in one copy can be detected by consulting the parity bit. Assuming single-bit errors, the same bit position in both copies of the base are unlikely to have errors. Thus, on detecting an error, the other copy can be taken as correct base value. This allows immediate recovery of base without raising an exception.

We do not duplicate the base when hardening is performed since at least 4B are always hardened (§III-C) and thus, base is always stored in hardened memory.

2. Handling Divergent Warps: When a divergent warp reaches writeback stage, the thread-registers of only active warps are written. Since we use delta compression on all threads of a warp, compressing divergent warps presents additional challenges. There are some ways to address this:

(i) Assuming that active threads are contiguous, one option is that only active threads are compressed. However, with decreasing number of active-threads in a warp, benefit from compression reduces. Also, the active threads of a divergent warp may not be contiguous, e.g., for needle (NED) benchmark, the active mask of one warp is $[0_2, 1, 0, 1, 0_2, 1_3, 0, 1_2, 0_{19}]$, where 0_k or 1_k denote k-consecutive 0s and 1s, respectively. Similarly, for gaussian (GSS) benchmark, an example of active mask is $[0_3, 1, 0_3, 1, 0_3, 1, 0_3, 1, 0_{16}]$. For such cases, compressing only active threads causes fragmentation since base and delta can no longer be placed in contiguous manner.

(ii) Since compressing divergent warps requires dealing with different number and position of active threads, it increases the complexity of (de)compressor circuits and their metadata and timing overheads, e.g., the compressor would need to account for active mask, and decompressed values would need to be transferred to different positions based on active mask. Also, an increase in the latency of BDI compression may make it unacceptable for RF.

(iii) Another option is to read the thread-registers of inactive threads also and then compress all the 32 threads. However, since these thread-registers are written by different warp-instructions, they are likely to have much smaller value locality. In BDI algorithm, even if one delta has larger than the specified width, the entire value is incompressible.

For these reasons, WarpC does not compress divergent warps. During RF write, if stored data is compressed, it is first decompressed and then a write is performed.

B. Thread-level Compression

ThreadC compresses each thread-register value individually and thus, requires 32 compressors. Hence, we use a simple compression approach, specifically, we exploit narrow values. Narrow values occur when a large size data type may be reserved for handling the worst-case scenario but the actual value may require fewer bytes [19], e.g., only 1B data may be stored in a 4B integer. ThreadC determines

the width K of a value from 4 possible values, viz. 0B, 1B, 2B and 4B (uncompressed) and from this, the smallest width is chosen. On a register read, only lower K bytes are read which reduces the number of vulnerable bits.

Note that WarpC and ThreadC have different strengths and limitations. WarpC exploits value similarity and zero values whereas ThreadC leverages narrow and zero values. WarpC cannot benefit divergent warps, whereas ThreadC can compress both non-divergent and (active threads of) divergent warps. However, ThreadC does not exploit value similarity and hence, it cannot compress values with low dynamic range. Due to this, for non-divergent applications, it is less effective in compressing data than WarpC.

To achieve the best of both WarpC and ThreadC, we propose the following approach. Since warps retain the same divergence behavior for long execution periods, the divergence behavior of an application can be recorded for first 1M cycles. Based on it, for highly-divergent applications, WarpC can be disabled and ThreadC can be used. Conversely, WarpC can be used for regular applications.

C. Selective Hardening

Since many real-world GPU applications show irregular behavior or have wide data values, another technique is also required for benefitting all the applications. We propose selective hardening of registers, i.e., left-most H bytes are designed with radiation-hardened circuits, where H is a multiple of 4. For example, with no compression and H=16B, only 112B (=128-16) remain vulnerable. With B4D0 compression (compressed size of 4B) and H=16B, entire register is stored in hardened memory and its SEV is reduced to zero. Note that designing entire RF with SE immune memory (i.e., $H = 128B$) would incur unacceptably large overheads, e.g., replacing an SRAM RF with an STT-RAM RF can reduce performance by 70% [11]. Clearly, by only performing selective hardening and also using compression, our technique incurs lower hardware and latency costs. Also, while Palframan et al. [10] harden selected bits of all the thread registers, we harden all the bits of selected thread registers. Further, WarpC exploits value similarity to compress the thread registers for preferentially storing them in hardened byte, whereas Palframan et al. do not exploit this opportunity.

IV. Implementation and overhead assessment

A. Implementation of Compression

For ThreadC, both divergent and non-divergent warps are considered compressible, whereas, for WarpC, only non-divergent warps are compressible and the divergent warps bypass the (de)compressor and thus, do not incur corresponding latency overheads. Compressor and decompressor are both implemented in pipelined manner and have one port for each bank which allows serving all banks in one cycle. For WarpC, serving both compressible and incompressible warps at the same cycle can cause additional conflicts since requests from different cycles will request the bank at the same cycle. To address this, we use two reservation arrays in the arbiter to solve the conflict for request to the compressor and request to the register banks separately. When the requests for compressible and incompressible data meet at the register bank, compressible writes are prioritized over incompressible writes and they are prioritized over reads. This ensures stall-free operation.

Latency overhead: WarpC uses BDI compression which takes 2 and 1 cycles for compression and decompression, respectively [9]. ThreadC uses narrow value detection (NVD) and since NVD circuits are much simpler than a compressor [19], ThreadC compression takes 1 cycle. Note that BDI compressor itself uses NVD circuit to find the width of each delta [9]. Decompression in ThreadC incurs no additional latency since it only involves reading lower K bytes of a narrow value. Thus, latency and hardware overhead of ThreadC are lower than that of WarpC.

Storage overhead: To store the compression state, WarpC uses 2 bits/warp and ThreadC uses 2 bits/thread (i.e., 64 bits/warp). By virtue of using compression, our technique reduces data access and wire movement energy [17]. Especially for AllZero data, read/write to RF are completely avoided. Further, BDI circuit and NVD circuit only involve addition/subtraction and/or bit-comparison [9, 19].

B. Implementation of Hardened memory

As for hardened memory, we use the 10T SRAM cell [13] which showed 98% less SE rate than the standard 6T SRAM cell and thus, data stored in this cell is assumed to be invulnerable. Compared to 6T cell, 10T cell has 7%, 72%, 43% and 40% overhead in write time, static power, dynamic power and area, respectively [14], e.g., on hardening 32 out of 128 bytes, overhead in static power, dynamic power and area are 18%, 11%, 10%, respectively. These overheads are comparable to that incurred with ECC [10]. Further, 10T cell provides 14% less SE rate than the ECC-protected SRAM [13]. When using (de)compression, the additional latency of hardened memory can be hidden with that of (de)compression and in other cases, 1 cycle penalty is incurred. The compression state encoding bits are also stored in hardened memory in the arbiter and due to their small size, their overhead is assumed to be negligible.

C. Implementation on Other GPU RF Architectures

Since the exact details of RF in commercial products are not known, previous work has assumed different RF organizations. One RF design [20] assumes that registers are split in 32 banks. Each bank is 4B wide and provides data only to one thread within the warp. Each bank provides data to one processing element (PE) only. In another organization [21], four PEs form a cluster and each cluster has its own RF which is 16B wide. The RF provides four 32b values to

four PEs associated with it. Our technique can easily work with these RF organizations. Also, hardening can be done at bank-granularity and WarpC reduces the number of banks consulted for accessing a data value.

V. Results and Analysis

We use GPGPUSim v3.2.2 simulator [22] and a configuration similar to NVIDIA Fermi GTX480 GPU. There are 15 SMs, each runs up to 48 warps. SM frequency is 700 MHz and 'greedy then oldest' (GTO) scheduling policy is used. RF has 16 banks and 128KB size. We simulate a diverge range of workloads from Lonestar, ISPASS09, Rodinia, Parboil, and CUDA SDK suites. In total, we simulate 20 workloads and they are shown in Table II.

Table II: Workloads and their acronyms

mst (MST), sp (SUP), sssp (SSP), bfs (BFS), LPS (LPS)
NN (NEN), NQU (NQU), gaussian (GSS), heartwall (HWA)
hotspot(HOS), needle (NED), particlefilter (PFL), pathfinder (PAF)
cutcp(CUT), mri-q(MRQ), tpacf(TPF), alignedtypes (ALT)
matrixmul (MML), reduction (RDC), streams (STR)

A. Results on Compression Techniques

Figure 2(a) shows the percentage of non-divergent warps and SEV reduction. Figure 2(b) shows percentage of RF writes compressed using each state to give insight into compressibility of RF values. The SEV reduction for any application depends on the fraction of non-divergent warps (for WarpC) and compressibility of RF values. For several applications, most instructions are non-divergent and hence, WarpC provides larger SEV reduction than ThreadC, e.g., MST, PAF, CUT, MRQ, TPF, ALT, RDC, STR, etc.

However, for some applications, most warps are divergent, e.g., for NED, GSS, SSP and BFS, 100%, 99.7%, 97.5% and 97.1% (respectively) warps are divergent. NED (Needleman-Wunsch) has limited parallelism in every iteration due to dependencies of processing data values in diagonal strip manner. In NED, no warp has more than 16 active threads. GSS (Gaussian elimination) solves system of equations using Gaussian elimination approach and requires synchronization between iterations. In BFS (breadth first search), the connectivity and distance of a node depend on the input graph and in SSP (single-source shortest paths), the shortest distance of a node depends on input graph. Due to these, both show irregular memory access pattern [23]. Hence, these applications do not benefit from WarpC. For these and a few other applications, e.g., NQU, HWA, HOS, etc., ThreadC provides larger SEV reduction than WarpC. Clearly, although WarpC on average performs better than ThreadC, a single compression technique cannot be taken as optimal for all applications.

From Figure 2(b), WarpC and ThreadC can compress 50.1% and 49.3% of writes on average. In ALT and STR, WarpC compresses all the writes and several other applications are also highly compressible, e.g., LPS, PAF, CUT, TPF, etc. For ThreadC, compressibility depends on presence of narrow values, e.g., for many applications, many RF values are zero, e.g. MST, SUP, LPS, PFL, etc.

Figure 2(c) shows the RF critical time (refer §II-B) averaged over entire execution . Clearly, RF critical times can be very high, for example, for SUP and BFS, critical times are 2662 and 1092 cycles, respectively and on average, the critical time is 459 cycles. Thus, RF values remain vulnerable for long periods which highlights the importance of reducing their SEV. Previous works (e.g., [15]) have also observed that RF inter-access times (the time period between two RF accesses) range in hundreds of cycles. Finally, as shown in Figure 2(d), our techniques incur less than 1% performance loss compared to a baseline that does not use any RF protection scheme or hardening. This is in acceptable range and is comparable to that with other reliability techniques, such as ECC.

B. Results on Hardening

To see the benefit from using both compression and hardening, for each application, we show the number of bytes required to be hardened to reduce SEV by $P\%$ compared to the baseline. Without compression, hardening $(P\times128)/100$ bytes can reduce SEV by $P\%$ for a fully non-divergent application. We look for ability of compression to reduce this requirement.

Figure 3 shows these results for $P = 50\%$ and $P = 90\%$. On average, without compression, 62 and 110 bytes need to be hardened for reducing SEV by 50% and 90% and WarpC can reduce this to 18 and 74 bytes, and ThreadC can reduce this to 21 and 102 bytes, respectively. For several benchmarks, compression alone can reduce SEV by at least 50%, obviating the need of hardening. For WarpC, this happens for MST, LPS, HWA, PAF, CUT, MRQ, ALT, MML, RDC, STR, etc. Even for reducing SEV by 90%, several benchmarks require only few hardened bytes on using compression. For WarpC, the examples of such benchmarks are ALT (4B), STR (20B), CUT (28B). For divergent applications, ThreadC reduces requirement of hardening more than WarpC, e.g., for 90% reduction with ThreadC, SSP, NED and GSS require 112B, 56B and 52B, respectively, which are smaller than that required for WarpC. On using hardening, the smaller benefit of ThreadC compared to WarpC is because WarpC stores the register in left-aligned manner to preferentially use hardened memory, whereas ThreadC stores each thread-register in its own place. Thus, by combining hardening with WarpC or ThreadC, stronger protection can be provided to a wide variety of GPU applications with only small overhead.

C. Parameter Sensitivity Results

Including B4D2 state in WarpC: On including B4D2 state in WarpC, average SEV reduction increases from 47.01% to 48.00%, although the metadata requirement of

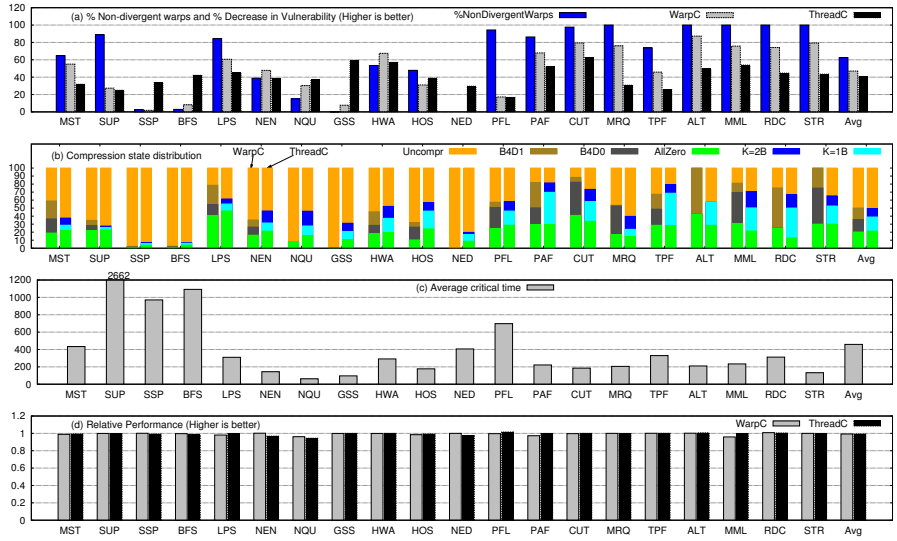

Figure 2: Results with WarpC and ThreadC: (a) percentage of divergent warps and SEV reduction (b) compression states (K=2B and K=1B are 2B and 1B narrow values in ThreadC) (c) average critical times and (d) relative performance (IPC)

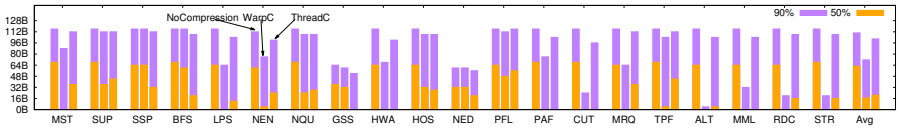

Figure 3: Hardening requirement for reducing SEV by 50% and 90% (value for 90% is the height of the whole bar)

WarpC increases from 2 bits/warp to 3 bits/warp. On average, 2.6% RF writes are compressed with B4D2 and a highest value of 24.7% is seen in RDC. Thus, for our workloads, including B4D2 state leads to only small benefit which confirms our choice of not using B4D2 state.

Using only one fixed width in ThreadC: We experiment with using a fixed width in ThreadC (called 'ThreadC-single'), such that, if the width of a value is at most K bytes, it is considered narrow, otherwise it is taken as wide, e.g., for $K = 0$B, only 0B values are considered as narrow and others as wide. For K = 0B, 1B and 2B, ThreadC-single provides SEV reduction of 22.99%, 24.96% and 24.63%, which are much lower than 40.77% achieved with ThreadC. Clearly, ThreadC finds the most compact width for a narrow value The benefit of ThreadC-single, however, it is that it uses simpler compressor circuit and only requires 1 bit/thread compared to 2 bit/thread in ThreadC.

VI. Conclusion

In this paper, we presented compression and selective hardening to reduce SE vulnerability of GPU RF and demonstrated its effectiveness over a range of workloads.

References

[1] S. Mittal *et al.*, "A Survey of Techniques for Modeling and Improving Reliability of Computing Systems," *IEEE TPDS*, 2015.

[2] D. Tiwari *et al.*, "Reliability lessons learned from GPU experience with the Titan supercomputer at Oak Ridge leadership computing facility," in *SC*, 2015.

[3] AMD HD7000 Graphics, http://goo.gl/PZBjLN, 2012.

[4] Intel Itanium Processor 9500, http://goo.gl/xy5m7G, 2012.

[5] "GeForce GTX Titan X," http://goo.gl/XajvIj, 2015.

[6] NVIDIA Maxwell, http://goo.gl/8NV82n, 2014.

[7] S. Mittal, "A Survey of Techniques for Architecting and Managing GPU Register File," *IEEE TPDS*, 2016.

[8] S. Mittal *et al.*, "Reducing Soft-error Vulnerability of Caches using Data Compression," *GLSVLSI*, pp. 197–202, 2016.

[9] G. Pekhimenko *et al.*, "Base-delta-immediate compression: practical data compression for on-chip caches," in *PACT*, 2012, pp. 377–388.

[10] D. Palframan *et al.*, "Precision-aware soft error protection for GPUs," *HPCA*, 2014.

[11] J. Tan *et al.*, "Soft-error reliability and power co-optimization for GPGPUS register file using resistive memory," in *DATE*, 2015.

[12] S. S. Mukherjee *et al.*, "A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor," *MICRO*, 2003.

[13] S. M. Jahinuzzaman *et al.*, "A soft error tolerant 10T SRAM bit-cell with differential read capability," *IEEE TNS*, 2009.

[14] G. Zhang *et al.*, "A novel single event upset hardened CMOS SRAM cell," *IEICE Electronics Express*, vol. 9, no. 3, pp. 140–145, 2012.

[15] M. Abdel-Majeed *et al.*, "Warped register file: A power efficient register file for GPGPUs," in *HPCA*, 2013, pp. 412–423.

[16] J. Leng *et al.*, "GPUWattch: enabling energy optimizations in GPGPUs," *ISCA*, pp. 487–498, 2013.

[17] S. Lee *et al.*, "Warped-compression: enabling power efficient GPUs through register compression," *ISCA*, pp. 502–514, 2015.

[18] A. Chakraborty *et al.*, "E < MC2: less energy through multi-copy cache," in *CASES*, 2010, pp. 237–246.

[19] J. Hu *et al.*, "On the exploitation of narrow-width values for improving register file reliability," *IEEE TVLSI*, 2009.

[20] W. Yu *et al.*, "SRAM-DRAM hybrid memory with applications to efficient register files in fine-grained multi-threading," *ISCA*, 2011.

[21] M. Gebhart *et al.*, "Energy-efficient mechanisms for managing thread context in throughput processors," in *ISCA*, 2011, pp. 235–246.

[22] A. Bakhoda *et al.*, "Analyzing CUDA workloads using a detailed GPU simulator," in *IEEE ISPASS*, 2009, pp. 163–174.

[23] M. Burtscher *et al.*, "A quantitative study of irregular programs on GPUs," *IISWC*, 2012.